This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2017 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
## Table of Contents

### CA OLQ Session

- Start A Session ................................................. 20
- Enter Commands ................................................ 20
- End a Session ...................................................... 21
- Customize Session Options ................................. 21
- Suspend a Session ................................................ 21

### CA OLQ Command Summary

- Default PF Key Assignments ................................ 22
- System management commands .......................... 23
- Data Retrieval Commands ................................. 23
- Report formatting commands ............................ 24
- Report output commands ................................. 25
- Qfile commands .................................................. 26
- Table processing commands ............................. 26

### CA OLQ Commands and Syntax

- BYE ....................................................................... 30
- Syntax ................................................................... 30
- Example ............................................................... 30
- CLEAR CURRENCY ................................................. 30
- Syntax ................................................................... 30
- CLEAR FUNCTION ................................................ 31
- Syntax ................................................................... 31
- Parameters ......................................................... 31
- Example ............................................................... 31
- COMPUTE ............................................................... 31
- Syntax ................................................................... 31
- Parameters ......................................................... 32
- Examples ............................................................. 32
- COMPUTE .. GROUP BY ......................................... 33
- Syntax ................................................................... 33
<table>
<thead>
<tr>
<th>Command</th>
<th>Syntax</th>
<th>Parameters</th>
<th>Examples</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE FILE</td>
<td></td>
<td></td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>DEFINE PATH</td>
<td></td>
<td></td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>DELETE COMPUTATION</td>
<td></td>
<td></td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>DELETE QFILE</td>
<td></td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>DELETE REPORT</td>
<td></td>
<td></td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>DELETE TABLE -- OLQ access mode</td>
<td></td>
<td></td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>DELETE TABLE -- IDMS access mode</td>
<td></td>
<td></td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>DELETE USER</td>
<td></td>
<td></td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>DISPLAY</td>
<td></td>
<td></td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>EDIT</td>
<td></td>
<td></td>
<td></td>
<td>53</td>
</tr>
</tbody>
</table>
FIND / GET WITHIN Index SET .......................................................................................................................... 84
Syntax ................................................................................................................................................................. 85
Parameters .......................................................................................................................................................... 85
Examples ........................................................................................................................................................... 87
FIND / GET WITHIN SET ....................................................................................................................................... 88
Syntax ................................................................................................................................................................. 88
Parameters .......................................................................................................................................................... 89
Considerations ................................................................................................................................................... 91
Examples ........................................................................................................................................................... 91
FIND / GET WITHIN SET Using SORTKEY ........................................................................................................ 92
Syntax ................................................................................................................................................................. 92
Parameters .......................................................................................................................................................... 93
Considerations ................................................................................................................................................... 96
Examples ........................................................................................................................................................... 96
FUNCTION ............................................................................................................................................................ 97
Syntax ................................................................................................................................................................. 97
Parameter ......................................................................................................................................................... 97
Example ........................................................................................................................................................... 98
HELP .................................................................................................................................................................... 98
Syntax ................................................................................................................................................................. 98
Parameters .......................................................................................................................................................... 99
Example ........................................................................................................................................................... 100
MENU .................................................................................................................................................................. 104
Syntax ................................................................................................................................................................. 105
Parameters .......................................................................................................................................................... 105
Example ........................................................................................................................................................... 106
OPTIONS ............................................................................................................................................................. 107
Syntax ................................................................................................................................................................. 107
Parameters .......................................................................................................................................................... 108
Examples ........................................................................................................................................................... 109
PAGE HEADER / FOOTER .................................................................................................................................... 116
Syntax ................................................................................................................................................................. 116
Parameters .......................................................................................................................................................... 117
Example ........................................................................................................................................................... 117
PRINT .................................................................................................................................................................. 118
Syntax ................................................................................................................................................................. 118
Parameters .......................................................................................................................................................... 119
Considerations ................................................................................................................................................... 121
Example ........................................................................................................................................................... 122
QFILE (OLQ Command) ...................................................................................................................................... 122
Syntax ................................................................................................................................................................. 122
SAVE QFILE .................................................................................................................. 124
  Syntax ....................................................................................................................... 124
  Parameters ............................................................................................................... 124
  Example .................................................................................................................... 125
SAVE REPORT ............................................................................................................... 125
  Syntax ....................................................................................................................... 126
  Parameters ............................................................................................................... 126
  Example .................................................................................................................... 126
SELECT -- OLQ access mode .................................................................................... 127
  Syntax ....................................................................................................................... 127
  Parameters ............................................................................................................... 127
  Examples .................................................................................................................. 130
SELECT -- IDMS access mode ................................................................................... 134
  Syntax ....................................................................................................................... 134
  Coding considerations ............................................................................................ 135
SEND TABLE -- OLQ access mode ............................................................................ 135
  Syntax ....................................................................................................................... 136
  Parameters ............................................................................................................... 136
  Considerations ........................................................................................................ 139
  Examples .................................................................................................................. 139
SEND TABLE -- IDMS access mode ........................................................................... 140
  Syntax ....................................................................................................................... 140
  Parameters ............................................................................................................... 140
  Considerations ........................................................................................................ 142
  Examples .................................................................................................................. 142
SET .............................................................................................................................. 142
  Syntax ....................................................................................................................... 143
  Parameters ............................................................................................................... 144
  Examples .................................................................................................................. 148
SIGNON ......................................................................................................................... 150
  Syntax ....................................................................................................................... 150
  Parameters ............................................................................................................... 150
  Examples .................................................................................................................. 151
SIGNON TABLE ............................................................................................................ 152
  Syntax ....................................................................................................................... 152
  Parameters ............................................................................................................... 152
  Example .................................................................................................................... 153
SORT .............................................................................................................................. 153
  Syntax ....................................................................................................................... 153
Built-In Functions and Syntax ................................................................. 178

Invoking Built-In Functions ................................................................. 178
Parameters Of Built-In Functions ......................................................... 186

Absolute Value .................................................................................. 187
Syntax ............................................................................................... 187
Invocation Names ............................................................................. 187
Parameter ......................................................................................... 187

Arc Cosine ......................................................................................... 187
Syntax ............................................................................................... 187
Invocation Names ............................................................................. 188
Parameters ....................................................................................... 188
Example ............................................................................................. 188

Arc Sine ............................................................................................. 188
Syntax ............................................................................................... 188
Invocation Names ............................................................................. 188
Parameter ......................................................................................... 189
Example ............................................................................................. 189

Arc Tangent ....................................................................................... 189
Syntax ............................................................................................... 189
Invocation Names ............................................................................. 189
Parameter ......................................................................................... 189
Example ............................................................................................. 190

Average ............................................................................................ 190
Syntax ............................................................................................... 190
Invocation Names ............................................................................. 190
Parameter ......................................................................................... 190
Example ............................................................................................. 190

Capitalization .................................................................................... 190
Syntax ............................................................................................... 191
Invocation Name .............................................................................. 191
Parameter ......................................................................................... 191
Example ............................................................................................. 191

Concatenate ....................................................................................... 191
Syntax ............................................................................................... 191
Invocation Names ............................................................................. 191
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>200</td>
</tr>
<tr>
<td>Invocation Name</td>
<td>200</td>
</tr>
<tr>
<td>Parameter</td>
<td>200</td>
</tr>
<tr>
<td>Example</td>
<td>200</td>
</tr>
<tr>
<td>Insert (Function)</td>
<td>200</td>
</tr>
<tr>
<td>Syntax</td>
<td>201</td>
</tr>
<tr>
<td>Invocation Names</td>
<td>201</td>
</tr>
<tr>
<td>Parameter</td>
<td>201</td>
</tr>
<tr>
<td>Example</td>
<td>201</td>
</tr>
<tr>
<td>Invert Sign</td>
<td>201</td>
</tr>
<tr>
<td>Syntax</td>
<td>201</td>
</tr>
<tr>
<td>Invocation Names</td>
<td>201</td>
</tr>
<tr>
<td>Parameter</td>
<td>201</td>
</tr>
<tr>
<td>Example</td>
<td>201</td>
</tr>
<tr>
<td>Left Justify</td>
<td>201</td>
</tr>
<tr>
<td>Syntax</td>
<td>202</td>
</tr>
<tr>
<td>Invocation Names</td>
<td>202</td>
</tr>
<tr>
<td>Parameter</td>
<td>202</td>
</tr>
<tr>
<td>Example</td>
<td>202</td>
</tr>
<tr>
<td>Length</td>
<td>202</td>
</tr>
<tr>
<td>Syntax</td>
<td>202</td>
</tr>
<tr>
<td>Invocation Names</td>
<td>202</td>
</tr>
<tr>
<td>Parameter</td>
<td>203</td>
</tr>
<tr>
<td>Example</td>
<td>203</td>
</tr>
<tr>
<td>Logarithm</td>
<td>203</td>
</tr>
<tr>
<td>Syntax</td>
<td>203</td>
</tr>
<tr>
<td>Invocation Names</td>
<td>203</td>
</tr>
<tr>
<td>Parameter</td>
<td>203</td>
</tr>
<tr>
<td>Example</td>
<td>203</td>
</tr>
<tr>
<td>Lowercase</td>
<td>204</td>
</tr>
<tr>
<td>Syntax</td>
<td>204</td>
</tr>
<tr>
<td>Invocation Names</td>
<td>204</td>
</tr>
<tr>
<td>Parameter</td>
<td>204</td>
</tr>
<tr>
<td>Example</td>
<td>204</td>
</tr>
<tr>
<td>Maximum</td>
<td>204</td>
</tr>
<tr>
<td>Syntax</td>
<td>204</td>
</tr>
<tr>
<td>Invocation Names</td>
<td>205</td>
</tr>
<tr>
<td>Parameter</td>
<td>205</td>
</tr>
<tr>
<td>Example</td>
<td>205</td>
</tr>
<tr>
<td>Minimum</td>
<td>205</td>
</tr>
<tr>
<td>Syntax</td>
<td>205</td>
</tr>
<tr>
<td>Invocation Names</td>
<td>205</td>
</tr>
<tr>
<td>Parameter</td>
<td>205</td>
</tr>
<tr>
<td>Example</td>
<td>205</td>
</tr>
</tbody>
</table>
Modulo ................................................................. 206
Syntax ................................................................. 206
Invocation Names .................................................. 206
Parameters .......................................................... 206
Example .................................................................. 206

Next Integer Equal or Higher .................................. 206
Syntax ................................................................. 206
Invocation Names .................................................. 207
Parameter ........................................................... 207
Example .................................................................. 207

Next Integer Equal or Lower .................................. 207
Syntax ................................................................. 207
Invocation Names .................................................. 207
Parameter ........................................................... 207
Example .................................................................. 208

Product ................................................................. 208
Syntax ................................................................. 208
Invocation Names .................................................. 208
Parameter ........................................................... 208
Example .................................................................. 208

Random Number ................................................... 208
Syntax ................................................................. 209
Invocation Names .................................................. 209
Parameter ........................................................... 209
Example .................................................................. 209

Right Justify .......................................................... 209
Syntax ................................................................. 209
Invocation Names .................................................. 210
Parameter ........................................................... 210
Example .................................................................. 210

Sign Value ............................................................ 210
Syntax ................................................................. 210
Invocation Names .................................................. 210
Parameter ........................................................... 210
Example .................................................................. 211

Sine ....................................................................... 211
Syntax ................................................................. 211
Invocation Names .................................................. 211
Parameter ........................................................... 211
Example .................................................................. 211

Square Root ........................................................... 211
Syntax  ........................................................................................................................................... 212
Invocation Names .................................................................................................................................. 212
Parameter ............................................................................................................................................ 212
Example ................................................................................................................................................ 212
Standard deviation ................................................................................................................................... 212
Syntax  ................................................................................................................................................. 213
Invocation Name ....................................................................................................................................... 213
Parameter ............................................................................................................................................... 213
Example .................................................................................................................................................. 213
Standard deviation population .................................................................................................................. 213
Syntax  .................................................................................................................................................. 214
Invocation Name ................................................................................................................................. 214
Parameter ............................................................................................................................................... 214
Example .................................................................................................................................................. 214
Substring ............................................................................................................................................... 214
Syntax  .................................................................................................................................................. 214
Invocation Names ............................................................................................................................... 214
Parameters ............................................................................................................................................ 214
Example .................................................................................................................................................. 214
Sum ....................................................................................................................................................... 215
Syntax  .................................................................................................................................................. 215
Invocation Names ............................................................................................................................... 215
Parameter ............................................................................................................................................... 215
Example .................................................................................................................................................. 215
Tangent .................................................................................................................................................. 216
Syntax  .................................................................................................................................................. 216
Invocation Names ............................................................................................................................... 216
Parameter ............................................................................................................................................... 216
Example .................................................................................................................................................. 216
Today .................................................................................................................................................... 217
Syntax  .................................................................................................................................................. 217
Invocation Names ............................................................................................................................... 217
Parameters ............................................................................................................................................ 217
Examples ................................................................................................................................................ 217
Tomorrow ............................................................................................................................................... 218
Syntax  .................................................................................................................................................. 218
Invocation Names ............................................................................................................................... 218
Parameters ............................................................................................................................................ 218
Examples ................................................................................................................................................ 219
Translate ............................................................................................................................................... 219
Syntax  .................................................................................................................................................. 220
Using qfiles

Data Retrieval

SELECT (IDMS access mode) ........................................... 227
SELECT (OLQ access mode) ........................................... 228
Logical Records .............................................................. 229
Using qfiles .................................................................... 229

Tailoring the CA OLQ Environment for Ease of Use .......... 227
Building qfiles ................................................................. 230
Executing qfiles ............................................................ 230
Reporting on qfiles ......................................................... 231
Special Uses of qfiles ..................................................... 231
  PF Key Module ............................................................... 231
  Signon Profile qfiles ..................................................... 232
Including Parameters in qfiles ........................................ 233
  Parameters ................................................................... 233
  Examples of Nesting qfiles ........................................... 236
Defining Report Headers ................................................ 236
Synonyms ....................................................................... 237
Code Tables .................................................................... 237
Date Option ..................................................................... 238
External Pictures ............................................................ 239

Using CA OLQ Efficiently ................................................. 240
Controlling Data Retrieval .............................................. 240
  Qfiles ....................................................................... 240
  Logical records ............................................................. 240
OLQ DML User Exit ....................................................... 241
  Sample Exit ................................................................ 241
  Assembly and Link Edit (z/OS) ...................................... 244
  Assembly and Link Edit (z/VSE) .................................... 244
Interrupt count ............................................................... 245
Using db-keys for retrieval .............................................. 246
Controlling Resource Consumption .................................. 246
  Sorts ........................................................................... 246
  Saved reports ............................................................... 247
  Db-key list .................................................................. 249
  Saving qfiles .............................................................. 249

Security ........................................................................ 250
Assigning Authority to Access CA OLQ ......................... 250
Limiting Access through Central Security ....................... 250
Initiating CA OLQ Dictionary Security ............................. 250
Securing User Access to CA OLQ Components ................ 251
  Subschema Access ........................................................ 251
  Qfile Access ................................................................ 252
  Securing Retrieval Interruption .................................... 253
Using LRF to Secure The Database ................................................................. 253
Security for ASF tables .................................................................................. 254
Security for Saved Reports ............................................................................. 254

**Batch Processing** .................................................................................... 256
JCL for z/OS and CMS Commands for OLQBATCH ........................................ 258
  z/OS JCL (central version) .......................................................................... 258
  z/OS JCL (local mode) .............................................................................. 259
  z/OS Local Mode Considerations .............................................................. 260
  CMS commands (central version) .............................................................. 261
  Usage ........................................................................................................ 261
  CMS commands (local mode) ...................................................................... 263
  z/VSE JCL (central version) ........................................................................ 263
  z/VSE JCL (local mode) ............................................................................. 264
  IDMSLBLS Procedure ............................................................................... 265
Setting Defaults for Batch Processing .............................................................. 269
  Defining files ............................................................................................ 269
  Signing on in batch ................................................................................... 271
OLQBatch Notification .................................................................................... 272
  OLQBNOTE Example for z/OS ................................................................. 272
  OLQBNOTE example for CMS .................................................................. 273
  OLQBNOTE example for z/VSE ............................................................... 274
Batch Class Specification ............................................................................... 275
Operating System Dependent Installation Instructions .................................. 275
Examples of Batch ....................................................................................... 277
  Creating Multiple Reports in One Job ....................................................... 278
  Wide Reports ............................................................................................ 278
Creating a Report with SELECT ................................................................... 278
Writing to a Disk File with SELECT .............................................................. 279

**Setting Defaults** ..................................................................................... 281
System Generation Options ........................................................................... 281
  Integrated Data Dictionary Options .......................................................... 284

**OLQ Coding Considerations** .................................................................. 290
Commands .................................................................................................... 290
  Using Abbreviations .................................................................................. 290
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ending Command Strings</td>
<td>290</td>
</tr>
<tr>
<td>Issuing Multiple Commands</td>
<td>290</td>
</tr>
<tr>
<td>Invoking Function Keys</td>
<td>290</td>
</tr>
<tr>
<td>Continuation Character</td>
<td>291</td>
</tr>
<tr>
<td>Defining Session Characters</td>
<td>291</td>
</tr>
<tr>
<td>Comment Character</td>
<td>291</td>
</tr>
<tr>
<td>Separator Character</td>
<td>291</td>
</tr>
<tr>
<td>Continuation Character</td>
<td>292</td>
</tr>
<tr>
<td>Entering Data</td>
<td>292</td>
</tr>
<tr>
<td>Specifying Data Values</td>
<td>292</td>
</tr>
<tr>
<td>Specifying Subscripts</td>
<td>293</td>
</tr>
<tr>
<td>Examples</td>
<td>294</td>
</tr>
</tbody>
</table>
CA Online Query (CA OLQ™) is a query tool and report writer used to retrieve information from a CA IDMS/DB database or other external files.

With CA OLQ command mode, you can do the following:

- Set up the CA OLQ environment to suit your needs (system management commands)
- Retrieve information from a CA IDMS/DB database
- Build report files based on information retrieved (report output commands)
- Perform sorts, computations, and functions on report files (report formatting commands and built-in functions)
- Create and save sequences of CA OLQ commands for regular use (qfile commands)
- Create and retrieve from stored tables (table processing commands)
- Run jobs in batch mode (CA OLQ batch commands)

For more information, see the following topics:

- CA OLQ Session (see page 20)
- CA OLQ Command Summary (see page 22)
- CA OLQ Commands and Syntax (see page 29)
- Global Syntax (see page 163)
- Built-In Functions and Syntax (see page 178)
- Tailoring the CA OLQ Environment for Ease of Use (see page 227)
- Using CA OLQ Efficiently (see page 240)
- Security (see page 250)
- Batch Processing (see page 256)
- Setting Defaults (see page 281)
- OLQ Coding Considerations (see page 290)
CA OLQ Session

A CA OLQ session is the environment in which you use the OLQ query and report writer commands to retrieve information from a CA IDMS/DB database or other external files.

- Start A Session (see page 20)
- Enter Commands (see page 20)
- End a Session (see page 21)
- Customize Session Options (see page 21)
- Suspend a Session (see page 21)

Start A Session

To start a CA OLQ session, log into your CA IDMS system and type `olq` beneath the system prompt, as follows:

```
ENTER NEXT TASK CODE: olq
```

The system enters CA OLQ command mode. See following example:

```
OLQ 091057 00 Please enter next command
```

This prompt is normally positioned on the fifth line of the screen. You enter your commands on the **command line** (the first four lines on the screen). You can set the number of lines dedicated to entering commands during system generation.

Enter Commands

In CA OLQ command mode, the command line is the top line of the screen, where you type the commands. See following example for entering the `edit` command:

```
edit
```

```
OLQ 091057 00 Please enter next command
```

You can enter multiple commands, up to the number of commands that fill the number of lines you set during system generation—four lines are set by default. Following are the rules for entering multiple commands:

- Use **command separators** to separate commands. The default command separator is the exclamation point (!).

- If the command exceeds the length allowed, use the **continuation character** to extend the command beyond the pseudo-converse. The default continuation character is the hyphen (-). You can also change the length of the command line so you can enter longer commands.
End a Session

To end an CA OLQ session, type the command **bye**. You are returned to the CA IDMS system prompt.

Customize Session Options

You can customize your CA OLQ environment to suit your needs. Customized options remain set for only the current session. At the next session, all the session options are returned to defaults.

Among other options, you can customize the following:

- **Echo/No echo**: User-entered commands are repeated by CA OLQ on the 3270-type device.
- **Full/sparse**: Specify whether or not the display format for path retrieval report lines suppresses repeating column values.
- **Header/No header**: Specify whether or not the report file displays a header line.

Suspend a Session

Suspending a session does not end it. You can return to a session after suspending it, and session options you previously set are still in effect.

You use the SUSPEND command to suspend a current session and return control to the transfer control facility or CA IDMS/DC or CA IDMS UCF.
CA OLQ Command Summary

CA OLQ provides commands for system management, data retrieval, reporting, and table processing. This topic provides an overview of these command types.

- Default PF Key Assignments (see page 22)
- System management commands (see page 23)
- Data Retrieval Commands (see page 23)
- Report formatting commands (see page 24)
- Report output commands (see page 25)
- Qfile commands (see page 26)
- Table processing commands (see page 26)

Default PF Key Assignments

Following are the PF key assignments as defined at system installation:

<table>
<thead>
<tr>
<th>PF Key</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>[PF1]</td>
<td>Help</td>
</tr>
<tr>
<td>[PF2]</td>
<td>Help commands</td>
</tr>
<tr>
<td>[PF3]</td>
<td>Bye</td>
</tr>
<tr>
<td>[PF4]</td>
<td>Display help</td>
</tr>
<tr>
<td>[PF5]</td>
<td>Display</td>
</tr>
<tr>
<td>[PF6]</td>
<td>Menu</td>
</tr>
<tr>
<td>[PF7]</td>
<td>Page backward</td>
</tr>
<tr>
<td>[PF8]</td>
<td>Page forward</td>
</tr>
<tr>
<td>[PF9]</td>
<td>Swap</td>
</tr>
<tr>
<td>[PF10]</td>
<td>Display left</td>
</tr>
<tr>
<td>[PF11]</td>
<td>Display right</td>
</tr>
<tr>
<td>[PF12]</td>
<td>Print</td>
</tr>
<tr>
<td>[Clear]</td>
<td>Undefined</td>
</tr>
<tr>
<td>[Enter]</td>
<td>Process input</td>
</tr>
<tr>
<td>[PA1]</td>
<td>Undefined</td>
</tr>
<tr>
<td>[PA2]</td>
<td>Undefined</td>
</tr>
</tbody>
</table>
System management commands

You can govern the general use of CA OLQ with these system management commands and tailor the CA OLQ session environment to suit your needs.

The following table summarizes the system management commands available.

<table>
<thead>
<tr>
<th>Use ...</th>
<th>Or ...</th>
<th>To ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYE</td>
<td>EXIT, GOOFBYE, OFF, QUIT, SIGNOFF</td>
<td>Terminate a CA OLQ session.</td>
</tr>
<tr>
<td>CLEAR CURRENCY</td>
<td></td>
<td>Release all database currencies that have been established by a CA OLQ session.</td>
</tr>
<tr>
<td>CLEAR FUNCTION</td>
<td></td>
<td>Clear control key functions in command mode CA OLQ.</td>
</tr>
<tr>
<td>DELETE USER</td>
<td></td>
<td>Delete the report directory associated with a particular user ID.</td>
</tr>
<tr>
<td>FUNCTIO N</td>
<td></td>
<td>Invoke control key functions in command mode CA OLQ.</td>
</tr>
<tr>
<td>HELP</td>
<td>SHOW</td>
<td>Display how to use CA OLQ commands and provide information about the data the current subschema can access.</td>
</tr>
<tr>
<td>MENU</td>
<td></td>
<td>Switch between CA OLQ command mode and a specific screen of the menu facility.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td></td>
<td>Set default parameters for a session.</td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td>Set system management parameters.</td>
</tr>
<tr>
<td>SIGNON</td>
<td></td>
<td>Initiate a CA OLQ session.</td>
</tr>
<tr>
<td>SUSPEND</td>
<td></td>
<td>Suspend the current session and return control to the transfer control facility or CA IDMS/DC or CA IDMS UCF.</td>
</tr>
<tr>
<td>SWAP</td>
<td></td>
<td>Switch from CA OLQ command mode to the menu facility.</td>
</tr>
<tr>
<td>SWITCH</td>
<td></td>
<td>Pass control to another CA IDMS/DC product.</td>
</tr>
</tbody>
</table>

Data Retrieval Commands

You can retrieve data from the database with these data retrieval commands. The commands available for data retrieval are presented in the table below. For the syntax and syntax rules of these commands, see the alphabetical listing of commands in Commands and Syntax (see page 29).

<table>
<thead>
<tr>
<th>Use ...</th>
<th>To ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE PATH</td>
<td>Place CA OLQ in database path definition mode.</td>
</tr>
<tr>
<td>Use ...</td>
<td>To ...</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>END PATH</td>
<td>Terminate path definition mode.</td>
</tr>
<tr>
<td>EXECUTE PATH</td>
<td>Execute the retrieval commands specified in the database path definition and build a report file of retrieved records.</td>
</tr>
<tr>
<td>FIND/GET logical record</td>
<td>Retrieve records by using DBA-defined paths through the database.</td>
</tr>
<tr>
<td>FIND/GET MOST RECENT</td>
<td>Retrieve the current of record type for the specified record name.</td>
</tr>
<tr>
<td>FIND/GET OWNER WITHIN SET</td>
<td>Retrieve the owner of a database set occurrence.</td>
</tr>
<tr>
<td>FIND/GET PHYSICAL SEQUENTIAL</td>
<td>Retrieve records based on their physical position in a database area.</td>
</tr>
<tr>
<td>FIND/GET using STORAGE KEY</td>
<td>Retrieve records based on their CALC-key or database-key value.</td>
</tr>
<tr>
<td>FIND/GET WITHIN DBKEYLIST</td>
<td>Retrieve records based on the results of previous retrieval commands.</td>
</tr>
<tr>
<td>FIND/GET WITHIN index SET</td>
<td>Retrieve records by using the name of an index set and the index-sort-key fields specified in the WHERE clause.</td>
</tr>
<tr>
<td>FIND/GET WITHIN SET</td>
<td>Retrieve records based on their membership in a database set.</td>
</tr>
<tr>
<td>FIND/GET WITHIN SET using SORTKEY</td>
<td>Retrieve member records in sorted database sets based on a specified sort key.</td>
</tr>
<tr>
<td>REPEAT for each of the above FIND/GET commands</td>
<td>Duplicate an immediately preceding FIND/GET command.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Retrieve information using the SELECT command.</td>
</tr>
<tr>
<td>SHOW PATH</td>
<td>Display the current path.</td>
</tr>
</tbody>
</table>

### Report formatting commands

You can specify the display format of reports containing data retrieved by CA OLQ commands. The following table presents the commands available for formatting reports in CA OLQ.

<table>
<thead>
<tr>
<th>Use ...</th>
<th>To ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTE</td>
<td>Perform computations on fields in a report file by using:</td>
</tr>
<tr>
<td>TE ... Group By</td>
<td>Arithmetic expressions</td>
</tr>
<tr>
<td>... GROUP BY</td>
<td>Built-in functions</td>
</tr>
<tr>
<td>DELETE COMPUTATION</td>
<td>Perform summary computations.</td>
</tr>
<tr>
<td>EDIT</td>
<td>Delete computed fields.</td>
</tr>
</tbody>
</table>
| EDIT | Edit a field for display by specifying:
**Report output commands**

You can save, display, print, and delete report files in CA OLQ command mode with report output commands.

The report output commands available are presented in the table below.

<table>
<thead>
<tr>
<th><strong>Use ...</strong></th>
<th><strong>To ...</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE REPORT</td>
<td>Delete the report specified</td>
</tr>
<tr>
<td>PAGE HEADER /FOOTER</td>
<td>Include a user-specified page header or footer in a report</td>
</tr>
<tr>
<td>SORT</td>
<td>Request that records within a report file be ordered by user-specified order criteria.</td>
</tr>
<tr>
<td>UNSORT</td>
<td>Return the report file to the original retrieval sequence following one or more SORT commands.</td>
</tr>
</tbody>
</table>

**Table:**

<table>
<thead>
<tr>
<th><strong>Use ...</strong></th>
<th><strong>To ...</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT COMPTATION</td>
<td>Edit a computed field for display by specifying:</td>
</tr>
<tr>
<td>EDIT GROUP BY</td>
<td>Edit the group defined by the COMPUTE...GROUP BY command. Use this command to change the level number of a group, specify spacing between groupings or specify the separator character which separates the grouping from the computed value.</td>
</tr>
<tr>
<td>ON BREAK</td>
<td>Display computed values at designated points within the report file. This command is provided for compatibility with prior releases but its use is discouraged. Use COMPUTE...GROUP BY instead.</td>
</tr>
<tr>
<td>USE ...</td>
<td>Use ...</td>
</tr>
<tr>
<td>EDIT ...</td>
<td>Edit characteristics, such as hexadecimal display, lead zeros, commas, a specific external picture, and a code table translation</td>
</tr>
<tr>
<td>A report heading</td>
<td>A report heading</td>
</tr>
<tr>
<td>Sparse, to suppress the display of repeating column values</td>
<td>Sparse, to suppress the display of repeating column values</td>
</tr>
<tr>
<td>The alignment of a column</td>
<td>The alignment of a column</td>
</tr>
<tr>
<td>The column under which a computed field is displayed</td>
<td>The column under which a computed field is displayed</td>
</tr>
<tr>
<td>EDIT GROUP BY</td>
<td>Edit the group defined by the COMPUTE...GROUP BY command. Use this command to change the level number of a group, specify spacing between groupings or specify the separator character which separates the grouping from the computed value.</td>
</tr>
<tr>
<td>ON BREAK</td>
<td>Display computed values at designated points within the report file. This command is provided for compatibility with prior releases but its use is discouraged. Use COMPUTE...GROUP BY instead.</td>
</tr>
<tr>
<td>PAGE HEADER /FOOTER</td>
<td>Include a user-specified page header or footer in a report</td>
</tr>
<tr>
<td>SORT</td>
<td>Request that records within a report file be ordered by user-specified order criteria.</td>
</tr>
<tr>
<td>UNSORT</td>
<td>Return the report file to the original retrieval sequence following one or more SORT commands.</td>
</tr>
</tbody>
</table>
CA IDMS Reference - 19.0

Use ... To ...
DISPLAY Direct CA OLQ to send a page of report file data to the user's terminal.
PRINT Direct a formatted CA OLQ report to a specific printer for a hard copy.
SAVE REPORT Associate a name with a report file and save it in the user’s directory for future use.
SHOW DIRECTORY List the reports available for the specified user.

Qfile commands

A qfile, like a path, is a sequence of commands used to build online reports. Unlike paths, qfiles are stored in the data dictionary and can contain any CA OLQ command, not just retrieval commands. With qfiles you can set up defaults for the CA OLQ operating environment, as well as construct reports.

Use the qfile commands presented in the table below to create, save, and execute qfiles in CA OLQ command mode.

<table>
<thead>
<tr>
<th>Use ...</th>
<th>To ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE QFILE</td>
<td>Delete the named qfile.</td>
</tr>
<tr>
<td>QFILE</td>
<td>Execute the named qfile.</td>
</tr>
<tr>
<td>SAVE QFILE</td>
<td>Save the current path and report formatting commands as the named qfile.</td>
</tr>
<tr>
<td>SHOW QFILE=</td>
<td>List the commands in the named qfile.</td>
</tr>
<tr>
<td>SHOW QFILES</td>
<td>List the qfiles available.</td>
</tr>
</tbody>
</table>

Table processing commands

You can use data table processing commands to maintain information in either ASF or SQL tables.

ASF tables:

ASF tables refers to tables associated with the IDMSR schema.

In order to use ASF tables for the session, you must set the access switch to \texttt{olq}.

\textbf{Note:} For more information about ASF tables, see the \textit{CA IDMS ASF User Guide}.

SQL tables:

SQL tables refers to tables associated with an SQL schema.
In order to use SQL tables for the session, you must set the access switch to idms.

**Note:** For more information about SQL tables, see the *CA IDMS SQL Quick Reference Guide*.

**How to specify the access switch:**

The access switch can be set in the following ways:

1. At system generation time
2. For an individual user in the Dictionary (IDD)
3. For the session, interactively (or until the switch is changed)

**Note:** For more information about setting the access switch, see SET (see page 142).

**Table processing commands:**

The following table lists the CA OLQ table processing commands:

<table>
<thead>
<tr>
<th>Use this command</th>
<th>To do this</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE TABLE</td>
<td>Delete SQL and ASF tables.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Retrieve specific information from SQL and ASF tables, logical and database records, and sequential files (batch only).</td>
</tr>
<tr>
<td>SEND TABLE</td>
<td>Store information from the current or named report file in SQL and ASF tables.</td>
</tr>
<tr>
<td>SIGNON TABLE</td>
<td>Access a specific ASF table to increase efficiency. <strong>This command is only relevant in OL Q access mode.</strong></td>
</tr>
<tr>
<td>HELP TABLES</td>
<td>List tables saved by the current or named user.</td>
</tr>
</tbody>
</table>

**Invalid OLQ commands:**

The following figure shows which OLQ commands become invalid when you set the access switch to IDMS access mode:

![Diagram showing invalid OLQ commands for IDMS access mode]

When you invoke:

- SET ACCESS OLQ
  - valid
  - BYE

- SET ACCESS IDMS
  - invalid
  - BYE
<table>
<thead>
<tr>
<th>CLEAR CURRENCY</th>
<th>CLEAR FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTE</td>
<td>COMPUTE</td>
</tr>
<tr>
<td>COMPUTE ...GROUP BY</td>
<td>COMPUTE ...GROUP BY</td>
</tr>
<tr>
<td>DEFINE FILE</td>
<td>DEFINE FILE</td>
</tr>
<tr>
<td>DEFINE PATH</td>
<td>DEFINE PATH</td>
</tr>
<tr>
<td>DELETE COMPUTATION</td>
<td>DELETE COMPUTATION</td>
</tr>
<tr>
<td>DELETE QFILE</td>
<td>DELETE QFILE</td>
</tr>
<tr>
<td>DELETE REPORT</td>
<td>DELETE REPORT</td>
</tr>
<tr>
<td>DELETE TABLE</td>
<td>DELETE TABLE</td>
</tr>
<tr>
<td>DELETE USER</td>
<td>DELETE USER</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>DISPLAY</td>
</tr>
<tr>
<td>EDIT</td>
<td>EDIT</td>
</tr>
<tr>
<td>EDIT COMPUTATION</td>
<td>EDIT COMPUTATION</td>
</tr>
<tr>
<td>EDIT GROUP</td>
<td>EDIT GROUP</td>
</tr>
<tr>
<td>END PATH</td>
<td>END PATH</td>
</tr>
<tr>
<td>EXECUTE PATH</td>
<td>EXECUTE PATH</td>
</tr>
<tr>
<td>FIELDS FOR</td>
<td>FIELDS FOR</td>
</tr>
<tr>
<td>FIND/GET...</td>
<td>FIND/GET...</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>HELP</td>
<td>HELP</td>
</tr>
<tr>
<td>MENU</td>
<td>MENU</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>OPTIONS</td>
</tr>
<tr>
<td>PAGE HEADER/FOOTER</td>
<td>PAGE HEADER/FOOTER</td>
</tr>
<tr>
<td>PRINT</td>
<td>PRINT</td>
</tr>
<tr>
<td>QFILE</td>
<td>QFILE</td>
</tr>
<tr>
<td>SAVE QFILE</td>
<td>SAVE QFILE</td>
</tr>
<tr>
<td>SAVE REPORT</td>
<td>SAVE REPORT</td>
</tr>
<tr>
<td>SELECT</td>
<td>SELECT</td>
</tr>
<tr>
<td>SEND TABLE</td>
<td>SEND TABLE</td>
</tr>
<tr>
<td>SET</td>
<td>SET</td>
</tr>
<tr>
<td>SIGNON</td>
<td>SIGNON</td>
</tr>
<tr>
<td>SIGNON TABLE</td>
<td>SIGNON TABLE</td>
</tr>
<tr>
<td>SORT</td>
<td>SORT</td>
</tr>
<tr>
<td>SUSPEND</td>
<td>SUSPEND</td>
</tr>
<tr>
<td>SWAP</td>
<td>SWAP</td>
</tr>
<tr>
<td>SWITCH</td>
<td>SWITCH</td>
</tr>
<tr>
<td>UNSORT</td>
<td>UNSORT</td>
</tr>
</tbody>
</table>

**Note:** For the syntax and syntax rules of CA IDMS SQL commands, see the **CA IDMS SQL Quick Reference Guide**.
CA OLQ Commands and Syntax

This topic describes the following CA OLQ™ commands and syntax:

- **BYE** (see page 30)
- **CLEAR CURRENCY** (see page 30)
- **CLEAR FUNCTION** (see page 31)
- **COMPUTE** (see page 31)
- **COMPUTE ... GROUP BY** (see page 33)
- **DEFINE FILE** (see page 36)
- **DEFINE PATH** (see page 40)
- **DELETE COMPUTATION** (see page 41)
- **DELETE QFILE** (see page 42)
- **DELETE REPORT** (see page 43)
- **DELETE TABLE -- OLQ access mode** (see page 43)
- **DELETE TABLE -- IDMS access mode** (see page 44)
- **DELETE USER** (see page 45)
- **DISPLAY** (see page 46)
- **EDIT** (see page 53)
- **EDIT COMPUTATION** (see page 60)
- **EDIT GROUP** (see page 63)
- **END PATH** (see page 64)
- **EXECUTE PATH** (see page 64)
- **FIELDS FOR** (see page 64)
- **FIND / GET Logical Record** (see page 67)
- **FIND / GET MOST RECENT** (see page 69)
- **FIND / GET PHYSICAL SEQUENTIAL** (see page 71)
- **FIND / GET OWNER WITHIN SET** (see page 74)
- **FIND / GET Using Storage Key** (see page 76)
- **FIND / GET WITHIN DBKEYLIST** (see page 81)
- **FIND / GET WITHIN Index SET** (see page 84)
- **FIND / GET WITHIN SET** (see page 88)
- **FIND / GET WITHIN SET Using SORTKEY** (see page 92)
- **FUNCTION** (see page 97)
- **HELP** (see page 98)
- **MENU** (see page 104)
- **OPTIONS** (see page 107)
- **PAGE HEADER / FOOTER** (see page 116)
- **PRINT** (see page 118)
- **QFILE (OLQ Command)** (see page 122)
- **SAVE QFILE** (see page 124)
- **SAVE REPORT** (see page 125)
BYE

BYE terminates the CA OLQ session. When you terminate a session with BYE, CA OLQ deletes the current report file.

Syntax

►►─┬─ ➔/>.EXIT ➔/>.GOODbye ➔/>.OFF ➔/>.QUIt ➔/>.SIGNOFF◄◄

Example

When you issue BYE, CA OLQ displays the following message:

OLQ 100029 00 Signoff accepted - CA OLQ session terminated

CLEAR CURRENCY

CLEAR CURRENCY releases all database currencies for the current subschema. With this command, you can start new retrievals without repeating the signon procedure.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

Syntax

►►─┐─ CLear CurrenCy ┌─────────────────────────────────────────────────────────►◄
CLEAR FUNCTION

CLEAR FUNCTION nullifies one or all of the function key settings.

Syntax

```
CLEAR FUNCTION control-key-value
orre

control-key-value

Parameters

- **control-key-value**
  Specifies a single control key whose assigned command is to be cleared. Valid values are [PA1], [PA2], and 1 through 99 (corresponding to PF keys 1 through 99).

- **ALL**
  Clears all control keys of their current functions (default).

Example

This example nullifies the [PF8] key:

```
clear function 8
```

COMPUTE

COMPUTE performs computations on fields in a report file. Computed fields can be manipulated similarly to database records.

To display a computed field in a structured report, specify the COLS= parameter in the DISPLAY commands.

Syntax

```
COMput e report-name = expression
```

```
report-name

expression
```
Parameters

- **REPORT= report-name**
  Identifies the saved report for which the computation is performed. If you don't specify the name of the report, the computations are performed on fields in the current report.

- **USER= user-name** -- Identifies the user ID of the report owner. If you don't specify the user ID, the report is retrieved from the current user's directory.

- **compute-name**
  Specifies the field name to be used to reference the computation in any of the OLQ reporting functions (that is, DISPLAY COLS=, EDIT, SORT). A **compute-name** that contains embedded blanks or special characters must be enclosed in quotation marks.

- **expression**
  Defines the computations used to create the new column.

Examples

The following examples use the report built by the SELECT statement:

```sql
select emp-last-name-0415,salary-amount-0420,bonus-percent-0420
from employee, emposition where emp-emposition
```

Compute Bonus, Total Salary

COMPUTE commands are used to define the computation of the BONUS, and TOTAL SALARY fields:

```sql
compute bonus=salary-amount-0420 * bonus-percent-0420
compute 'total salary'=salary-amount-0420 + bonus
display columns=emp-last-name-0415, 'total salary', bonus
```

<table>
<thead>
<tr>
<th>EMP-LAST-NAME-0415</th>
<th>TOTAL SALARY</th>
<th>BONUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINGER</td>
<td>38654.000</td>
<td>154.000</td>
</tr>
<tr>
<td>TERNER</td>
<td>13852.000</td>
<td>52.000</td>
</tr>
<tr>
<td>LINGER</td>
<td>42797.500</td>
<td>297.500</td>
</tr>
<tr>
<td>LINGER</td>
<td>38152.000</td>
<td>152.000</td>
</tr>
<tr>
<td>PENMAN</td>
<td>39156.000</td>
<td>156.000</td>
</tr>
<tr>
<td>LINGER</td>
<td>85858.000</td>
<td>850.000</td>
</tr>
<tr>
<td>LINGER</td>
<td>75750.000</td>
<td>750.000</td>
</tr>
<tr>
<td>LITERATA</td>
<td>37762.500</td>
<td>262.500</td>
</tr>
<tr>
<td>WILCO</td>
<td>80800.000</td>
<td>800.000</td>
</tr>
<tr>
<td>HEARROWITZ</td>
<td>33231.000</td>
<td>231.000</td>
</tr>
<tr>
<td>TYRO</td>
<td>20080.000</td>
<td>80.000</td>
</tr>
<tr>
<td>KAHALLY</td>
<td>20080.000</td>
<td>80.000</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>101000.000</td>
<td>1000.000</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>90900.000</td>
<td>900.000</td>
</tr>
</tbody>
</table>

To see more of the report, page down:

```sql
display next page
```
EMPLOYEE/EMPOSITION REPORT

mm/dd/yy

<table>
<thead>
<tr>
<th>EMP-LAST-NAME-0415</th>
<th>TOTAL SALARY</th>
<th>BONUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>46322.000</td>
<td>322.000</td>
</tr>
<tr>
<td>KING</td>
<td>14558.000</td>
<td>58.000</td>
</tr>
<tr>
<td>CLOUD</td>
<td>53119.250</td>
<td>369.250</td>
</tr>
<tr>
<td>HENDON</td>
<td>242408.000</td>
<td>2480.000</td>
</tr>
<tr>
<td>PEOPLES</td>
<td>80800.000</td>
<td>800.000</td>
</tr>
<tr>
<td>DOUG</td>
<td>33231.000</td>
<td>231.000</td>
</tr>
<tr>
<td>ORGATZI</td>
<td>39273.000</td>
<td>273.000</td>
</tr>
<tr>
<td>WAGNER</td>
<td>47329.000</td>
<td>329.000</td>
</tr>
<tr>
<td>GALLMAY</td>
<td>33231.000</td>
<td>231.000</td>
</tr>
<tr>
<td>GARDNER</td>
<td>14956.000</td>
<td>56.000</td>
</tr>
<tr>
<td>JACOBI</td>
<td>55385.000</td>
<td>385.000</td>
</tr>
<tr>
<td>WILDER</td>
<td>90900.000</td>
<td>900.000</td>
</tr>
<tr>
<td>MUNYON</td>
<td>36252.000</td>
<td>252.000</td>
</tr>
<tr>
<td>CLOTH</td>
<td>38266.000</td>
<td>266.000</td>
</tr>
</tbody>
</table>

**Compute with a Built-In Function**

This example uses a built-in function to define a compute statement:

```sql
compute name = concatenate(extract(emp-last-name-0415), ',', extract(emp-first-name-0415))
display columns = name, salary-amount-0420
```

**COMPUTE ... GROUP BY**

COMPUTE GROUP BY performs computations on fields in a report file. Computed fields can be manipulated similarly to database records.

These computations are then displayed at break points that have been defined by the GROUP BY clause.
Parameters

- **REPORT=report-name**
  Identifies the saved report for which the computation is performed. If you don't specify the name of the report, the computations are performed on fields in the current report.
  - **USER=user-name** -- Identifies the user ID of the report owner. If you don't specify the user ID, the report is retrieved from the current user's directory.

- **compute-name**
  Specifies the field name to be used to reference the computation in any of the OLQ reporting functions (that is, DISPLAY COLS=, EDIT, SORT). A compute-name that contains embedded blanks or special characters must be enclosed in quotation marks.

- **expression**
  Defines the computations used to create the new column. Expression typically contains an aggregate function.

- **GROUP BY**
  Specifies a break will occur.
  - **group-by-expression** -- Specifies the field to break on; can be any compute expression that does not contain an aggregate function
  - **ALL** -- Final break processing for the entire report

- **HAVING**
  Applies selection criteria to the groupings of data values determined by the GROUP BY expression.
  - **criteria** -- Restricts the groups displayed; can contain aggregate functions; for example, HAVING AVE(SALARY) > 40000
  - **ALL** -- Specifies that no selection criteria on groups should be applied

**Note:** For more information about HAVING clause, see FIND / GET Selection Criteria (see page 168).
• **LEVEL n**
  Specifies the break level, in numeric order, with 1 representing the highest level. For example, if LEVEL 1 is specified, a break at the highest level causes a break at each subsequent lower level. The default $n$ value is 1.

### Examples

The following example uses the report built by the SELECT statement:

```
select dept-id-0410, emp-last-name-0415, salary-amount-0420
from department, employee, emposition
where dept-employee and emp-emposition
```

**Compute Total Salary**

COMPUTE commands are used to define the computation of the 'TOTAL SALARY' field:

```sql
compute 'total salary' = sum(salary-amount-0420)
group by dept-id-0410 ! display
```

```
DEPARTMENT/EMPLOYEE/EMPLACEMENT REPORT
mm/dd/yy

DEPT-ID-0410 EMP-LAST-NAME-0415 SALARY-AMOUNT-0420
6666 HENDON 240000.00
6666 PAPAZEUZ 100000.00
6666 PAPAZEUZ 90000.00
6666 RUPEE 80000.00
6666 RUPEE 76000.00
6666 WILDER 90000.00
```

```
TOTAL SALARY 676000.00
```

```
2000 BLOOMER 15000.00
2000 HUTTON 44000.00
2000 JENSON 82000.00
2000 KIMBALL 45000.00
2000 KING 14500.00
```

```
- 1 -
```

```
DEPARTMENT/EMPLOYEE/EMPLACEMENT REPORT
mm/dd/yy

DEPT-ID-0410 EMP-LAST-NAME-0415 SALARY-AMOUNT-0420
2000 NICEMAN 14000.00
```

```
TOTAL SALARY 214500.00
```

```
- 2 -
```

**Compute Having**

The HAVING clause is used to display the total salary of any department with more than 5 employees:

```sql
compute 'total salary' = sum(salary-amount-0420)
group by dept-id-0410 having count > 5 ! display
```

```
DEPARTMENT/EMPLOYEE/EMPLACEMENT REPORT
mm/dd/yy

DEPT-ID-0410 EMP-LAST-NAME-0415 SALARY-AMOUNT-0420
6666 HENDON 240000.00
```
### DEFINE FILE

DEFINE FILE allows OLO/Batch to include a data set as input to a query or to send to a data set the unedited data that was captured during a query. A DEFINE command identifies a file and relates an input data set to an IDD-defined record.

- File name
- Record name
- Dictionary name
- Dictionary node
- Logical unit size
- Record format
Data set organization
Device type
Logical record length
Block size
Tape labels

This article describes the following information:

- Syntax (see page 37)
- Parameters (see page 38)
- Examples (see page 39)

Syntax

```plaintext
DEFine File    file-name

RE Cord        record-name (version)

DICtname       dictionary-name

DICTNode       dictionary-node-name

LOGical unit   sysnn

RECFm  F F B  V VB U

DSOrg  PS VSAM

DEVtype  DISK TAPE CARD LIST PRINTER

LREcl  logical-record-length

BLKsize  block-size

TAPe labels are  OMITted STAandard NONstandard USER
```
Parameters

- **file-name**
  In OS and CMS, a 1- to 8-character alphabetic file name that assigns a file name and file characteristics to an input or an output file. In DOS, file-name can be a maximum of 7 characters long. The file name must be referenced by a DD name in your batch job stream.

- **RECORD= record-name**
  A 1- to 32-character alphabetic name of a record stored in the data dictionary that names an Integrated Data Dictionary (IDD®) record corresponding to the file definition.
  - **(version)** -- An integer that indicates the record version (defaults to 1).

- **DICTNAME= dictionary-name**
  A 1- to 8-character alphabetic dictionary name that identifies the dictionary where the IDD record definition resides.

- **DICTNODE= dictionary-node-name**
  A 1- to 8-character alphabetic dictionary node name, Distributed Database System (DDS) only, that identifies the dictionary node in which the IDD record definition resides.

- **LOGICAL UNIT= sysnnn**
  Specifies the name of the logical unit.

- **RECFM= record-format**
  Specifies the record format of the named file. *Record-format* can be:
  - F -- Fixed
  - FB -- Fixed block
  - V -- Variable
  - VB -- Variable block
  - U -- Undefined
  
  *Record-format* is required for DOS.

- **DSORG= data-set**
  Specifies the data set organization of the named file.
  - PS -- Physical sequential
  - VSAM -- VSAM entry sequenced data set

- **DEVTYPE= device-type**
  Specifies the device type for the named file. *Device-type* can be:
DISK
TAPE
CARD
LIST
PRINTER

LRECL= **logical-record-length**
Specifies the logical record length, in bytes, of the named file. *Logical-record-length* is an integer in the range of 1 to 32,767. This parameter is required for DOS.

BLKSIZE= **block-size**
Specifies the block size, in bytes, of the named file. *Block-size* is an integer in the range 1 to 32,767.
This parameter is required for DOS.

TAPE LABELS ARE **label-status**
Specifies tape labels for the file definition. *Label-status* can be:

- OMITTED
- STANDARD
- NONSTANDARD
- USER

OUTPUT
Identifies the file as an output file.

Examples

**Example 1:**

To use a data set as input to a query, describe the data in the input file as a record in IDD such as:

```
ADD
  RECORD NAME IS INPUT-REC VERSION IS 1.
02 INPUT-ELE-1
    PICTURE IS S999
    USAGE IS COMP-3.
02 INPUT-ELE-2
    PICTURE IS 9(18)
    USAGE IS DISPLAY.
```

and include input cards such as:

```
//SYSIPT    DD *
  DEFINE FILE INFILE RECORD INPUT-REC
  SELECT * FROM INPUT-REC
  DISPLAY
//INFILE    DD *
```
The corresponding job control language statement must name the file in the DD name:

<table>
<thead>
<tr>
<th>Op. System</th>
<th>JCL Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>//INFILE DD DSN=infile,DISP=SHR</td>
</tr>
<tr>
<td>z/VSE</td>
<td>// DLBL INFILE,'infile'</td>
</tr>
<tr>
<td></td>
<td>// EXTENT sysnnn,nnnnnn,,,1,ssss</td>
</tr>
<tr>
<td></td>
<td>// ASSIGN sysnnn,DISK,VOL=nnnnnn,SHR</td>
</tr>
<tr>
<td>CMS</td>
<td>FILEDEF INFILE DISK filename filetype filemode (OPTIONS</td>
</tr>
</tbody>
</table>

**Example 2:**

To use a data set as output from a query, include commands:

```
//OUTFILE DD DSN=
//SYSIPT DD *
SIGNON SS=EMPSS01
DEFINE FILE OUTFILE OUTPUT
SELECT 'DEPARTMENT'.*, 'EMPLOYEE'.* FROM 'DEPARTMENT', 'EMPLOYEE' -
OUTPUT OUTFILE
```

It is not required to define a record in the data dictionary to describe the output. The output will contain all the fields from the DEPARTMENT record followed by every field in the EMPLOYEE record. No editing of any fields will be done, and the fields will be contiguous.

5300BLUE SKIES2 03210023CATHERINE O’HEAR
5100BRAINSTORMING 00150023CATHERINE O’HEAR
2000ACCOUNTING AND PAYROLL 00110023CATHERINE O’HEAR

**DEFINE PATH**

DEFINE PATH places CA OLQ in database path definition mode.

**Access mode:** The syntax below is invalid when the access switch is set to IDMS.

**Syntax**

```
DEFINE path-name PATH
```
Parameter

- **path-name**
  The 1- to 32-character name of the path being defined. When a path name is specified, it appears as a title on each report file output page.

Example

This example defines a simple path:

```plaintext
define path
get all seq office
get all employee in office-employee
execute path
display cols=office-code-0450 emp-last-name-0415 emp-city-0415
```

This path retrieves all the occurrences of the office record, the employees belonging to that office, and displays the office code, the employees' last names, and the employees' cities.

DELETE COMPUTATION

DELETE COMPUTATION deletes the named computed fields from the report.

Syntax

```
DELETE COMPUTATION compute-name
```

Parameters

- **compute-name**
  Specifies the name of the computed field to be deleted.

- **ALL**
  Specifies that all computed fields be deleted.

Example

This example operates on the report created and modified by the following SELECT and COMPUTE statements:

```sql
select emp-id-0415, salary-amount-0420 from employee, emposition where emp-emposition
!compute 'average salary' = avg(salary-amount-0420) group by all !display
```

<table>
<thead>
<tr>
<th>EMP-ID-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>1204</td>
<td>25000</td>
</tr>
<tr>
<td>1140</td>
<td>23000</td>
</tr>
<tr>
<td>0145</td>
<td>35000</td>
</tr>
</tbody>
</table>
To delete the computed field AVERAGE SALARY:

```
delete computation 'average salary' ! display
EMP-ID-0415    SALARY-AMOUNT-0420
```

```1204   25000
1140   23000
0145   35000
0532   30000```

**DELETE QFILE**

DELETE QFILE allows you to delete qfiles saved in the dictionary with the SAVE QFILE command.

**Syntax**

```
DELETE QFile qfile-name (version) = (version)
DICtname dictionary-name = dictionary-name
DICTNODE dictionary-node-name = dictionary-node-name
USER user-name = user-name
```

**Parameters**

- **qfile-name**
  Identifies the qfile to be deleted

- **(version)**
  Specifies the version number of the named qfile.

- **DICtname= dictionary-name**
  Identifies the dictionary where the named qfile is stored.

- **DICTNODE= dictionary-node-name**
  Identifies the dictionary node that controls the named dictionary.

- **USER= user-name**
  Specifies the owner of the qfile.
DELETE REPORT

DELETE REPORT allows you to delete any reports you have passkey authority to delete.

**Batch considerations:** DELETE REPORT is invalid when running local mode.

**Syntax**

```
  DELete REPORT [ ALL REPORTs ] [ REPORT=report-name ] [ USER=user-name ]
```

**Parameters**

- **ALL REPORTS**
  Specifies that all saved reports are to be deleted.

- **REPORT= report-name**
  Specifies deletion of a specific report.

- **USER= user-name**
  Specifies the user who saved the specified report. If no user is specified, the default is the current user.

**Note:** In order to specify the deletion of another user's table, you must have the appropriate passkeys.

DELETE TABLE -- OLQ access mode

DELETE TABLE allows you to delete ASF tables provided you hold the appropriate passkeys.

**Access mode:** The syntax below is invalid when the access switch is set to IDMS.

**Syntax**

```
  DELete TABLE asf-table-name [ OWNER=user-name ]
```
Parameters

- **asf-table-name**
  Specifies the ASF table to be deleted

- **OWNER= user-name**
  Specifies the user ID of the owner of the ASF table. If *user-name* isn’t specified, the default is the current user ID.

- **CATALOG= dictionary-name**
  Specifies the dictionary containing the catalog entry for the named ASF table.

- **LOCATION= dictionary-node**
  Specifies the DDS node controlling the catalog.

Example

In this example, the table definition and associated occurrences for the EMP-HOSPITAL table are deleted:

```
delete table=emp-hospital owner=bdm catalog=asfdict
OLQ 116006  TABLE DEFINITION EMP-HOSPITAL SUCCESSFULLY DELETED
```

DELETE TABLE -- IDMS access mode

DELETE TABLE allows you to delete SQL tables provided you hold the appropriate security.

**Access mode:** The syntax below is **invalid** when the access switch is set to **OLQ**.

Syntax

```
DELETE TABLE
  DROP TABLE
    schema. sql-table-name

DICtname dictionary-name
```

Parameters

- **sql-table-name**
  Specifies the SQL table to be deleted
• **schema**
  The name of the schema associated with the SQL table.

• **DICTNAME= dictionary-name**
  Specifies the dictionary containing the catalog entry for the named SQL table.

### Example

In this example, the table definition and associated occurrences for the EMP-HOSPITAL table are deleted:

```
drop table=emp-hospital
OLQ 090016 00  Table "EMP-HOSPITAL" successfully DELETED.
```

### DELETE USER

DELETE USER deletes the report directory associated with a particular user ID.

When DELETE USER is issued, all saved reports in the specified user's directory are deleted.

⚠️ **Note:** The DELETE USER command does not remove a user from the dictionary.

### Syntax

```
\[\text{DELETE USER [user-name]}\]
```

### Parameters

- **USER= user-name**
  Specifies the user ID whose report directory is to be deleted.

⚠️ **Note:** *User-name* cannot be the user ID for the user currently signed on to CA OLQ.

### Example

The following DELETE USER command deletes the report directory for user TDB:

```
delete user = tdb
OLQ 107009 00  "DELETE USER " COMMAND SUCCESSFULLY COMPLETED.
```
DISPLAY

DISPLAY lists report files page-by-page on your terminal screen.

Subsequent DISPLAY commands keep the parameters set in previous DISPLAY commands unless overwritten.

This article describes the following information:

- Syntax (see page 46)
- Parameters (see page 47)
- Considerations (see page 48)
- Examples (see page 49)

Syntax

```
►─┬────────────────────────────────────────────────────────────────►
├─ REPort ─┬─────┬─ report-name ─┬─────┬─ USER ─┬─────┬─ user-name ─┘
└─ SUMmary ─┬─────┬─ DETail ─┬─────┬─ ENTire ─┘
├─ CURrent ─┬─ report ─┘
├─ HELp ─┬── report ─┘
├─ RIGHt ─┬─ LEFt ─┘
├─ PAGE ─┬─ LINE ─┘
├─ FIRst ─┬─ LAST ─┬─ NEXt ─┬─ PRIor ─┤
├─ * ─┬─ * ─┬─ page/line-count ─┤
│ page/line-count ─┘
├─ SPRead ─┬── EVEN ─┬── MAXimum ─┬── LEFt ─├── nn ─┘
└─ COLumns ─┬── COLS ─┤ field-reference ─┐
                                                  └─ field-reference ┘
```

Parameters

- **REPORT= report-name**
  Specifies the saved report to be displayed.

- **USER= user-name**
  The user whose report dictionary contains the named report file.

- **SUMMARY**
  Requests output of summary report lines only.

- **DETAIL**
  Requests output of detail report lines only.

- **ENTIRE**
  Request output of both detail and summary lines.

- **CURRENT REPORT**
  Requests output of the last report displayed.

- **HELP REPORT**
  Requests output of the help report file built by the last HELP command.

- **RIGHT/LEFT**
  Specifies horizontal movement within the report file.

- **PAGE/LINE**
  Requests the report begin with a specific page or line:
  - **PAGE** requests that the report file output begins at the current or specified page number.
  - **LINE** requests that the report file page begins at the current or specified line number.
  The default is PAGE. When PAGE or LINE is specified without DISPLAY, the optional parameters SUMMARY/DETAIL/ENTIRE and CURRENT/HELP REPORT do not apply.

- **FIRST**
  Outputs a page of report file data, beginning at page 1, line 1.

- **LAST**
  Outputs the last page of report file data.

- **NEXT**
  Outputs a page of report file data, beginning at the page or line number immediately following the current page or line number.

- **PRIOR**
  Outputs a page of report file data, beginning at the page or line number immediately preceding the current page or line number.

- *** **
  Outputs the current page of report file data, beginning at the first line of that page.
* + -
Outputs a page of report file data, beginning \( n \) pages or lines before (-) or after (+) the current page or line number. The asterisk (*) is a required character that explicitly references the current page or line.

- **page** -- The number of pages
- **line-count** -- The number of lines

**page/line-count**
Specifies the starting point of the output relative to the current page and line number: \( \text{Page/line-count} \) outputs a report page, beginning at the specified page or line number.

**SPREAD EVEN/MAXIMUM/LEFT/nn**
Specifies the space between the columns.

- **EVEN** -- The same number of spaces between each column (Space the columns evenly).
- **MAXIMUM** -- The maximum number of spaces between each column.
- **LEFT** -- Displays columns starting in the left most position with one space separating each column.
- **nn** -- \( nn \) spaces between each column. Zero is not a valid number. The minimum number of spaces allowed is one.

**COLUMNS**
Specifies the columns included in the output and, optionally, the order and width of those columns. Column specifications remain in effect until altered by a subsequent DISPLAY command.

- **MAXIMUM** -- The output of as many columns, starting with column 1, as can appear on one page of the report. Excess columns are ignored. No warning message is produced.
- **field-reference** -- The columns and the number of characters in each output column.

**FIXED COLUMNS**
Specifies the columns, and their order, to remain on the screen when paging left and right. The columns specified with this parameter precede the columns specified in the COLUMNS= parameter. They remain fixed on the left side of the terminal screen. Column specifications remain in effect until altered by a subsequent DISPLAY command, or a FIXED COLUMNS=NONE command.

- **NONE** -- No report fields are fixed on the screen. This cleans out the fixed columns list.
- **field-reference** -- The columns and the number of characters in each column to be output.

**Considerations**
Report files sometimes contain information that cannot be displayed:

- **(@)** -- The at sign indicates an unprintable character. CA OLQ provides a translation function that handles all characters written to a terminal or to the print queue.
Note: For an explanation of how to modify the CA OLQ translation table, see the CA IDMS installation guide for your operating system.

- (*) -- The asterisk indicates invalid data. The invalid data flag appears when data is not stored in the defined format or when a COMPUTE command yields invalid results (as with decimal overflows and division by zero). If you want to view the characters represented, you can use EDIT HEXADECIMAL to display the value in its hexadecimal representation.

Null character considerations

The null character is by default a period (.). You can override this by invoking the SET NULL command.

Note: For more information about the SET command see, SET (see page 142), later in this chapter.

Data retrieved in SQL tables can contain null values. To display them, CA OLQ pads the entire length of the display field with the null character.

Examples

The report file used for these examples has been built by executing the SELECT statement shown below:

```
select dept-name-0410, emp-last-name-0410, salary-amount-0420
from department, employee, emposition
where dept-employee and emp-emposition
```

DISPLAY

When the DISPLAY keyword is specified with no subsequent parameters, the first page of the report file is output as shown below. Specification of PAGE, LINE, PAGE FIRST, PAGE 1, LINE FIRST, or LINE 1 parameters produces the same results:

```
display
OLQ 104009 04  DISPLAY RIGHT to see more report columns
DEPARTMENT/EMPLOYEE/EMPOSITION REPORT
mm/dd/yy
DEPT-NAME-0410       EMP-LAST-NAME-0415
EXECUTIVE ADMINISTRATION   HENDON
EXECUTIVE ADMINISTRATION   PAPAZEUS
EXECUTIVE ADMINISTRATION   PAPAZEUS
EXECUTIVE ADMINISTRATION   RUPEE
EXECUTIVE ADMINISTRATION   RUPEE
EXECUTIVE ADMINISTRATION   WILDER
ACCOUNTING AND PAYROLL     BLOOMER
ACCOUNTING AND PAYROLL     HUTTON
ACCOUNTING AND PAYROLL     JENSON
```
The DISPLAY RIGHT command displays the report right side of the report file, if the report is too wide to fit on the screen:

```
display right

DEPARTMENT/EMPLOYEE/EMPOSITION REPORT
mm/dd/yy
EMP-LAST-NAME-0415 SALARY-AMOUNT-0420
HENDON 240000.00
PAPAZEUS 180000.00
PAPAZEUS 90000.00
RUPEE 80000.00
RUPEE 76000.00
WILDER 90000.00
BLOOMER 150000.00
HUTTON 440000.00
JENSON 82000.00
KIMBALL 450000.00
KING 145000.00
NICEMAN 140000.00
FITZHUGH 138000.00
JOHNSON 135000.00
```

**DISPLAY COLUMNS**

You can display whichever columns you want in any order with the COLS= parameter:

```
display cols=2,3

DEPARTMENT/EMPLOYEE/EMPOSITION REPORT
mm/dd/yy
EMP-LAST-NAME-0415 SALARY-AMOUNT-0420
HENDON 240000.00
PAPAZEUS 180000.00
PAPAZEUS 90000.00
RUPEE 80000.00
RUPEE 76000.00
WILDER 90000.00
BLOOMER 150000.00
HUTTON 440000.00
JENSON 82000.00
KIMBALL 450000.00
KING 145000.00
NICEMAN 140000.00
FITZHUGH 138000.00
JOHNSON 135000.00
```

**SPREAD LEFT**

You can use the SPREAD parameter to specify the distances between the columns. In this example the following report was modified with the SPREAD LEFT command:
### DEPARTMENT/EMPLOYEE REPORT

**mm/dd/yy**

<table>
<thead>
<tr>
<th>DEPT-ID-0410</th>
<th>EMP-ID-0415</th>
<th>EMP-ZIP-FIRST-FIVE-0415</th>
</tr>
</thead>
<tbody>
<tr>
<td>6666</td>
<td>30</td>
<td>02198</td>
</tr>
<tr>
<td>6666</td>
<td>471</td>
<td>03256</td>
</tr>
<tr>
<td>6666</td>
<td>1</td>
<td>02312</td>
</tr>
<tr>
<td>6666</td>
<td>472</td>
<td>03145</td>
</tr>
<tr>
<td>2000</td>
<td>69</td>
<td>01675</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
<td>02176</td>
</tr>
<tr>
<td>2000</td>
<td>11</td>
<td>02176</td>
</tr>
<tr>
<td>2000</td>
<td>67</td>
<td>01239</td>
</tr>
<tr>
<td>2000</td>
<td>106</td>
<td>02176</td>
</tr>
<tr>
<td>2000</td>
<td>101</td>
<td>02176</td>
</tr>
<tr>
<td>1080</td>
<td>81</td>
<td>03458</td>
</tr>
<tr>
<td>1080</td>
<td>8683</td>
<td>10996</td>
</tr>
<tr>
<td>1080</td>
<td>51</td>
<td>02546</td>
</tr>
<tr>
<td>1080</td>
<td>91</td>
<td>06182</td>
</tr>
</tbody>
</table>

### DISPLAY FIXED COLUMNS

This example illustrates the use of FIXED COLUMNS to keep a column on the screen while you page right and left to look at other columns:

```
display fixed columns = emp-last-name-0415
```

OLQ 104009 04 DISPLAY RIGHT to see more report columns

DEPARTMENT/EMPLOYEE/EMPOSITION REPORT

**mm/dd/yy**

<table>
<thead>
<tr>
<th>EMP-LAST-NAME-0415</th>
<th>DEPT-NAME-0410</th>
</tr>
</thead>
<tbody>
<tr>
<td>HENDON</td>
<td>EXECUTIVE ADMINISTRATION</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>EXECUTIVE ADMINISTRATION</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>EXECUTIVE ADMINISTRATION</td>
</tr>
<tr>
<td>RUPEE</td>
<td>EXECUTIVE ADMINISTRATION</td>
</tr>
<tr>
<td>RUPEE</td>
<td>EXECUTIVE ADMINISTRATION</td>
</tr>
<tr>
<td>WILDER</td>
<td>EXECUTIVE ADMINISTRATION</td>
</tr>
<tr>
<td>BLOOMER</td>
<td>ACCOUNTING AND PAYROLL</td>
</tr>
<tr>
<td>HUTTON</td>
<td>ACCOUNTING AND PAYROLL</td>
</tr>
<tr>
<td>JENSON</td>
<td>ACCOUNTING AND PAYROLL</td>
</tr>
<tr>
<td>KIMBALL</td>
<td>ACCOUNTING AND PAYROLL</td>
</tr>
<tr>
<td>KING</td>
<td>ACCOUNTING AND PAYROLL</td>
</tr>
<tr>
<td>NICEMAN</td>
<td>ACCOUNTING AND PAYROLL</td>
</tr>
<tr>
<td>FITZHUGH</td>
<td>PERSONNEL</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>PERSONNEL</td>
</tr>
</tbody>
</table>

**** BUFFER OVERFLOW; DISPLAY LINES LOST ****
Now if you page right, the EMP-LAST-NAME-0415 column remains on the page.

<table>
<thead>
<tr>
<th>EMP-LAST-NAME-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>HENDON</td>
<td>240000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>100000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>90000.00</td>
</tr>
<tr>
<td>RUPEE</td>
<td>80000.00</td>
</tr>
<tr>
<td>RUPEE</td>
<td>76000.00</td>
</tr>
<tr>
<td>WILDER</td>
<td>90000.00</td>
</tr>
<tr>
<td>BLOOMER</td>
<td>150000.00</td>
</tr>
<tr>
<td>HUTTON</td>
<td>44000.00</td>
</tr>
<tr>
<td>JENSON</td>
<td>82000.00</td>
</tr>
<tr>
<td>KIMBALL</td>
<td>45000.00</td>
</tr>
<tr>
<td>KING</td>
<td>145000.00</td>
</tr>
<tr>
<td>NICEMAN</td>
<td>140000.00</td>
</tr>
<tr>
<td>FITZHUGH</td>
<td>130000.00</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>135000.00</td>
</tr>
</tbody>
</table>

Use DISPLAY SPREAD EVEN to put an equal number of spaces between the columns:

```
DEPARTMENT/EMPLOYEE/EMPOSITION REPORT
mm/dd/yy
EMP-LAST-NAME-0415 SALARY-AMOUNT-0420
HENDON 240000.00
PAPAZEUS 100000.00
PAPAZEUS 90000.00
RUPEE 80000.00
RUPEE 76000.00
WILDER 90000.00
BLOOMER 150000.00
HUTTON 44000.00
JENSON 82000.00
KIMBALL 45000.00
KING 145000.00
NICEMAN 140000.00
FITZHUGH 130000.00
JOHNSON 135000.00
```

Truncating columns

You can fit all report columns on the screen by truncating the values in one or more alphanumeric fields:

```
display col=1(15),2,3
```

```
DEPARTMENT/EMPLOYEE/EMPOSITION REPORT
mm/dd/yy
DEPT-NAME-0410 DEPT-ID-0410 SALARY-AMOUNT-0420
EXECUTIVE ADMIN 6666 240000.00
EXECUTIVE ADMIN 6666 100000.00
EXECUTIVE ADMIN 6666 90000.00
EXECUTIVE ADMIN 6666 80000.00
EXECUTIVE ADMIN 6666 76000.00
EXECUTIVE ADMIN 6666 98000.00
ACCOUNTING AND 2000 150000.00
ACCOUNTING AND 2000 44000.00
ACCOUNTING AND 2000 82000.00
ACCOUNTING AND 2000 45000.00
ACCOUNTING AND 2000 145000.00
```
EDIT

EDIT edits a field for display. To edit a computed field for display, see EDIT COMPUTATION (see page 60).

This article describes the following information:

- Syntax (see page 53)
- Parameters (see page 54)
- Examples (see page 59)

Syntax

```
EDIT record-name. [view-id.] logical-record-name.element-name.

field-name [(-) subscript]

VIEW [=(view-id.)]

Display HEXadecimal
COMmas
NOCOMmas

NO$ LEAdzeros
NOleadzeros

OLQheader [=(olq-header)]
NOOLQheader

PICTure [=(external-picture)]
NOPICTure

code-table-expansion
NOcode-table

SPArse

FULL

ALIGN = LEFT CENTER RIGHT

Expansion of code-table

CODetable = code-table-name (version) Display WIDTH width
```
Parameters

- **view-id**
  The alternate ID of the record or logical record name.

- **record-name**
  The database record name containing the field to be edited.

- **logical-record-name.element-name**
  The logical record element name to be edited.

- **field-name**
  The name of the field to be edited. You can request editing for fields in database records and for dbkey fields. When *field-name* is a subscripted field, only OLQHEADER/NOOLQHEADER applies to individual occurrences of the repeating field.

  - *(subscript)* -- One or more occurrences of a repeating field. Each occurrence is identified by a subscript enclosed in parentheses. Multiple entries are separated by commas and are limited to the number specified in the OCCURS clause of the schema record description. If a repeating field name is specified without a subscript, all fields are displayed. If a repeating field requires more than one subscript, a second set of parentheses is required.

- **VIEW=view-id**
  Specifies the alternate ID of the record or logical record name.

- **DISPLAY/HEXADECIMAL**
  Specifies whether report fields are output in display or hexadecimal format. When HEXADECIMAL is specified, all other output format options except OLQHEADER are ignored.

- **COMMAS/NOCOMMAS**
  Specifies whether report fields are displayed with or without commas; non-numeric fields are unaffected.
  When you specify COMMAS, commas follow every third digit of numeric displays (counting backwards from implicit or explicit decimal positions).

- **$ NO$**
  Specifies whether report fields are displayed with or without dollar signs; non-numeric fields are unaffected.

- **LEADZEROS/NOLEADZEROS**
  Specifies whether the numeric fields in the field list are displayed with or without leading zeros; non-numeric fields are not affected.

- **OLQHEADER= olq-header**
  Specifies that CA OLQ headers are used as headers for displayed data.
  If the OLQHEADER option is used, associated CA OLQ headers replace the field names if any CA OLQ headers are defined in the data dictionary or if any are specified by the user with *olq-header*. *olq-header* specifies one or more lines of user-supplied field headers. Any number of lines can be specified, up to one less than the maximum number of lines output on the terminal. CA OLQ reserves space for the display of at least one report detail line. If a blank space is included in any
header line, it must be enclosed in quotation marks. When header-line is not specified for a particular field, CA OLQ uses the field header previously defined either in the data dictionary or through CA OLQ.

**Note:** To display CA OLQ headers (whether defined in the dictionary or supplied by the user), OPTIONS=OLQHEADER must be in effect.

- NOOLQHEADER
  Specifies that field names are used as headers for displayed data.

- PICTURE=
  Specifies that external pictures are used to edit report fields.

  - external-picture -- An external picture to edit a report field. External pictures override editing characteristics specified with the LEADZEROS/NOLEADZEROS, COMMAS/NOCOMMAS, and $/NO$ parameters of the EDIT command.

    - If an external picture is constructed with the EDIT command, CA OLQ uses it to edit the named report field.
    - If an external picture is defined for the field in the data dictionary, CA OLQ uses the stored external picture to format the field.

A user-specified external picture overrides any external picture that exists for a field in the data dictionary.

The characters available for constructing alphanumeric, alphabetic, and numeric external pictures are presented in Table 3.

The following rules apply to external picture construction:

- A user-specified external picture must contain at least one X, 9, G, A, Z, or *. CA OLQ uses the first X, 9, G, A, Z, or * in an external picture to determine whether the picture describes an alphanumeric, numeric, graphic, or alphabetic field, as follows:

  - If the first significant edit character is an X, CA OLQ recognizes the field as containing alphanumeric data; characters other than X, B, or parentheses are treated as insertion characters.

  - If the first significant edit character is an A, CA OLQ recognizes the field as containing alphabetic data; characters other than A, B, or parentheses are treated as insertion characters.

  - If the first significant edit character is a 9, Z, or *, CA OLQ recognizes the field as containing numeric data; characters other than 9, Z, $, *, +, -, B, or parentheses are treated as insertion characters.

  - If the first significant edit character is a G, CA OLQ recognizes the field as containing double-byte character string (DBCS) characters; characters other than G, B, or parentheses are treated as insertion characters.
A user-specified external picture can contain one or more insertion characters. In alphanumeric and alphabetic pictures, CA OLQ displays all insertion characters, regardless of their position in the picture format. In numeric pictures, characters other than 9, Z, $, *, +, -, B, or parentheses are recognized as insertion characters only when embedded in a series of 9, Z, or * characters; insertion characters at the beginning or end of a numeric picture description are suppressed.

**Note:** When the value of a field is negative, CA OLQ will display insertion characters at the end of the numeric picture, thereby allowing the user to specify accounting information.

**Note:** External picture formats, whether specified with the EDIT command or defined in the data dictionary, are only used to edit fields when OPTIONS=PICTURE is in effect.

**NOPICTURE**

Requests a default picture to edit a report field. The default picture for a report field is derived from the internal picture stored in the data dictionary. Keep in mind the following:

- The default picture length is determined by the number of characters that is specified by the internal picture.
- The default picture data type is the same as that defined for the internal picture.
- The following translations are made for numeric internal picture characters:
  - The internal picture character $ for a numeric field translates to a plus sign (+) or a negative sign (-) in the default picture.
  - The internal picture character V for a numeric field translates to a decimal point (.) in the default picture.

**CODETABLE= codetable(version)**

 Specifies whether a code table is used to format a report field. Code tables are defined and stored in the data dictionary by using the IDD DDDL Compiler.

**Note:** For further information on how to create code tables, see the *CA IDMS IDD DDDL Reference Guide*.

A sample code table of months is shown below. For each encoded number found in the report file, CA OLQ displays a decoded month value. If an invalid number is found, the literal 'INVALID MONTH' is displayed:

<table>
<thead>
<tr>
<th>ENCODED VALUE</th>
<th>DECODED VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>JANUARY</td>
</tr>
<tr>
<td>02</td>
<td>FEBRUARY</td>
</tr>
<tr>
<td>03</td>
<td>MARCH</td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>DECEMBER</td>
</tr>
</tbody>
</table>
Two types of code tables exist as follows:

- **Built-in tables** are part of the elements for which they are defined. When a HELP REPORT command is issued, CA OLQ indicates built-in tables defined for report fields by displaying **DICTIONARY** in the corresponding (CODE TABLE) columns. The built-in table that exists for a field in the data dictionary can be overridden by specification of a stand-alone table.

- **Stand-alone tables** are defined separately from the elements to which they pertain. To list the names of stand-alone tables, issue the DISPLAY ALL TABLES DDDL statement in IDD.

Code table editing overrides all other editing that might be requested for a report column, including leading dollar signs, commas, leading zeros, hexadecimal notation, and external picture formatting.

**Note:** When code table formatting is requested for a report field, CA OLQ uses the decoded values for subsequent processing in both the SORT command and the WHERE clause.

- **NOCODETABLE**
  Specifies that no code table is used to format a report field.

- **SPARSE**
  Specifies that only the first occurrence of a repeating column value is displayed. Note that OLQ suppresses a column display only when all of the following conditions are met:

  - The column has been assigned a sparse attribute.
  - The previous row contains an identical value for that column.
  - The columns to the left of that column have not changed values.
  - The column is not the last (right-most column) in the display. If you want the last column sparsed, you must change its display sequence so it is no longer the last column.

- **FULL**
  Specifies that all the occurrences of a repeating report line are displayed.

- **ALIGN**
  Indicates that the data values for the specified columns are to be aligned within the column boundaries as follows:

  - LEFT -- Aligned on the left
  - CENTER -- Centered
  - RIGHT -- Aligned on the right

  The default for numeric fields is right, and the default for all other fields is left.
Table 3. Characters Used for Constructing External Pictures

<table>
<thead>
<tr>
<th>Character Type of Data Described by the Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>A single alphanumeric character or double-byte character string (DBCS) character stored with no shiftstrings.</td>
</tr>
<tr>
<td>B</td>
<td>A single blank character. B can appear anywhere in the picture.</td>
</tr>
<tr>
<td>(n)</td>
<td>Follows any character to represent n consecutive repetitions of the specified character. N must be an integer in the range 1 through 9999.</td>
</tr>
<tr>
<td>other</td>
<td>Characters other than A, B, or parentheses can be used as insertion characters.</td>
</tr>
<tr>
<td>9</td>
<td>A single numeric character (0 through 9).</td>
</tr>
<tr>
<td>G</td>
<td>A double-byte character string (DBCS) character stored with no shiftstrings.</td>
</tr>
<tr>
<td>Z</td>
<td>Z is an insertion character when it is preceded by a 9, a decimal point, or a zero-suppression character. Otherwise, Z is zero-suppression character.</td>
</tr>
<tr>
<td>$</td>
<td>Multiple dollar signs at the beginning of an external picture represent a floating dollar sign. The dollar sign is an insertion character when it is preceded by a 9, a decimal point, or a zero-suppression character.</td>
</tr>
<tr>
<td>*</td>
<td>Multiple asterisks at the beginning of an external picture provide check protection. The asterisk is an insertion character when it is preceded by a 9, a decimal point, or a zero-suppression character.</td>
</tr>
<tr>
<td>+</td>
<td>A plus sign in the first position of an external picture indicates signed data. Multiple plus signs at the beginning of an external picture represent a floating sign. The plus sign is an insertion character when it is preceded by a 9, a decimal point, or a zero-suppression character.</td>
</tr>
<tr>
<td>-</td>
<td>A minus sign in the first position of an external picture indicates signed data. Multiple minus signs at the beginning of an external picture represent a floating sign. The minus sign is an insertion character when it is preceded by a 9, a decimal point, or a zero-suppression character.</td>
</tr>
<tr>
<td>.</td>
<td>The period character is used as a decimal point. Data is aligned with the decimal point in an external picture, and is truncated or padded when necessary. The decimal point terminates zero suppression when zero-suppression precede the decimal point. Zero-suppression characters become insertion characters if placed after a decimal point. The first period in a series of period characters is the decimal point in a picture. If no decimal point exists in the data, a decimal point is assumed after the rightmost numeric character.</td>
</tr>
</tbody>
</table>

Table 4. Examples of User-Specified Pictures

<table>
<thead>
<tr>
<th>Data Stored</th>
<th>Internal Picture</th>
<th>External Picture</th>
<th>Data Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>123400M</td>
<td>X(7)</td>
<td>X(7)</td>
<td>123400M</td>
</tr>
<tr>
<td>123400M</td>
<td>X(7)</td>
<td>XBXXXXXBX</td>
<td>1 23400 M</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>A(4)</td>
<td>A(4)</td>
<td>JOHN</td>
</tr>
<tr>
<td>TWOWORDS</td>
<td>A(9)</td>
<td>A(3)BA(5)</td>
<td>TWO WORDS</td>
</tr>
<tr>
<td>2350000</td>
<td>9(7)</td>
<td>9(7)</td>
<td>2350000</td>
</tr>
<tr>
<td>2350000</td>
<td>9(7)</td>
<td>9(7).99</td>
<td>2350000.00</td>
</tr>
<tr>
<td>2350000</td>
<td>9(7)</td>
<td>$$$$,$$$,$$9.99</td>
<td>$2,350,000.00</td>
</tr>
</tbody>
</table>
### Data Stored

<table>
<thead>
<tr>
<th>Data Stored</th>
<th>Internal Picture</th>
<th>External Picture</th>
<th>Data Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>2350000</td>
<td>9(7)</td>
<td>99/99/999</td>
<td>23/50/000</td>
</tr>
<tr>
<td>00120</td>
<td>9(5)</td>
<td>ZZZZZ</td>
<td>120</td>
</tr>
<tr>
<td>9876</td>
<td>9(4)</td>
<td>+++99</td>
<td>+9876</td>
</tr>
</tbody>
</table>

### Examples

The following examples illustrate the use of the EDIT command to format report fields, based on the report built with the SELECT statement shown below:

```sql
select emp-last-name-0415, ss-number-0415, salary-amount-0420
from employee, emposition where emp-emposition
```

#### EMPLOYEE/EMPOSITION REPORT

<table>
<thead>
<tr>
<th>EMP-LAST-NAME-0415</th>
<th>SS-NUMBER-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINGER</td>
<td>92345812</td>
<td>38500.00</td>
</tr>
<tr>
<td>TERNER</td>
<td>45672222</td>
<td>13000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19556712</td>
<td>42500.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19556712</td>
<td>39800.00</td>
</tr>
<tr>
<td>PENMAN</td>
<td>14593186</td>
<td>85000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19073343</td>
<td>75000.00</td>
</tr>
<tr>
<td>LITERATA</td>
<td>23567831</td>
<td>37500.00</td>
</tr>
<tr>
<td>WILCO</td>
<td>111000023</td>
<td>80000.00</td>
</tr>
<tr>
<td>HEAROWITZ</td>
<td>31896154</td>
<td>33000.00</td>
</tr>
<tr>
<td>TYRO</td>
<td>19893456</td>
<td>20000.00</td>
</tr>
<tr>
<td>KAHALLY</td>
<td>29661234</td>
<td>20000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>22887770</td>
<td>100000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>22887770</td>
<td>90000.00</td>
</tr>
</tbody>
</table>

#### Edit a Field

The report built above is too wide to fit on one terminal screen. Use EDIT to fit all the columns on one terminal screen:

```sql
edit emp-last-name-0415 pic 'x(10)' ! display
```

#### EMPLOYEE/EMPOSITION REPORT

<table>
<thead>
<tr>
<th>EMP-LAST-NAME-0415</th>
<th>SS-NUMBER-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINGER</td>
<td>92345812</td>
<td>38500.00</td>
</tr>
<tr>
<td>TERNER</td>
<td>45672222</td>
<td>13000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19556712</td>
<td>42500.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19556712</td>
<td>39800.00</td>
</tr>
<tr>
<td>PENMAN</td>
<td>14593186</td>
<td>85000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19073343</td>
<td>75000.00</td>
</tr>
<tr>
<td>LITERATA</td>
<td>23567831</td>
<td>37500.00</td>
</tr>
<tr>
<td>WILCO</td>
<td>111000023</td>
<td>80000.00</td>
</tr>
<tr>
<td>HEAROWITZ</td>
<td>31896154</td>
<td>33000.00</td>
</tr>
<tr>
<td>TYRO</td>
<td>19893456</td>
<td>20000.00</td>
</tr>
<tr>
<td>KAHALLY</td>
<td>29661234</td>
<td>20000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>22887770</td>
<td>100000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>22887770</td>
<td>90000.00</td>
</tr>
</tbody>
</table>

- 1 -
Edit Commas

In this example, the EDIT command requests display of the SALARY-AMOUNT-0420 field with commas and dollar signs:

```
edit salary-amount-0420 commas $ ! display
```

<table>
<thead>
<tr>
<th>EMP-LAST-NAME-0415</th>
<th>SS-NUMBER-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINGER</td>
<td>92345812</td>
<td>$38,500.00</td>
</tr>
<tr>
<td>TERNER</td>
<td>45672222</td>
<td>$13,000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19556712</td>
<td>$42,500.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19556712</td>
<td>$38,000.00</td>
</tr>
<tr>
<td>PENMAN</td>
<td>14593186</td>
<td>$39,000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19673343</td>
<td>$85,000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>19673343</td>
<td>$75,000.00</td>
</tr>
<tr>
<td>LITERATA</td>
<td>23567831</td>
<td>$37,500.00</td>
</tr>
<tr>
<td>WILCO</td>
<td>111000023</td>
<td>$80,000.00</td>
</tr>
<tr>
<td>HEAROWITZ</td>
<td>31896154</td>
<td>$33,000.00</td>
</tr>
<tr>
<td>TYRO</td>
<td>19893456</td>
<td>$20,000.00</td>
</tr>
<tr>
<td>KAHALLY</td>
<td>29661234</td>
<td>$20,000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>22887770</td>
<td>$100,000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>22887770</td>
<td>$90,000.00</td>
</tr>
</tbody>
</table>
- 1 -

Edit External Picture

This example shows EDIT describing an external picture for the JOB-ID-0440 field that specifies an insertion character:

```
edit ss-number-0415 pic '999-99-9999' ! display
```

<table>
<thead>
<tr>
<th>EMP-LAST-NAME-0415</th>
<th>SS-NUMBER-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINGER</td>
<td>092-34-5812</td>
<td>$38,500.00</td>
</tr>
<tr>
<td>TERNER</td>
<td>045-67-2222</td>
<td>$13,000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>019-55-6712</td>
<td>$42,500.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>019-55-6712</td>
<td>$38,000.00</td>
</tr>
<tr>
<td>PENMAN</td>
<td>014-59-3186</td>
<td>$39,000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>010-67-3343</td>
<td>$85,000.00</td>
</tr>
<tr>
<td>LINGER</td>
<td>010-67-3343</td>
<td>$75,000.00</td>
</tr>
<tr>
<td>LITERATA</td>
<td>023-56-7831</td>
<td>$37,500.00</td>
</tr>
<tr>
<td>WILCO</td>
<td>111-00-0023</td>
<td>$80,000.00</td>
</tr>
<tr>
<td>HEAROWITZ</td>
<td>031-89-6154</td>
<td>$33,000.00</td>
</tr>
<tr>
<td>TYRO</td>
<td>019-89-3456</td>
<td>$20,000.00</td>
</tr>
<tr>
<td>KAHALLY</td>
<td>029-66-1234</td>
<td>$20,000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>022-88-7770</td>
<td>$100,000.00</td>
</tr>
<tr>
<td>PAPAZEUS</td>
<td>022-88-7770</td>
<td>$90,000.00</td>
</tr>
</tbody>
</table>
- 1 -

EDIT COMPUTATION

EDIT COMPUTATION allows you to edit computed fields for display. You have the same options as with the EDIT statement.
Syntax

```plaintext
EDIT COMPUTATION compute-name

OFFSET field-name

OLQHEADER header-name

display-option
```

Parameters

- **compute-name**
  The name of the field to be edited.

- **OFFSET field-name**
  Specifies under which column the computed field specified by a COMPUTE GROUP BY command should be displayed. The default is the first column named in the COMPUTE statement's GROUP BY expression.

- **OLQHEADER header-name**
  Provides a report heading containing the column value of the group field. The user-supplied header takes on the edit characteristics of the computed field. `Header-name` can consist of:
  - A symbolic parameter that contains a dollar sign ($) preceding a report column name (for example, $DEPT-NAME-0410, which would translate into the name of the department on the displayed report)
  - A user-supplied title (for example, DEPARTMENT NAME instead of DEPT-NAME-0410)
  - A combination of the above two

- **display-option**
  Any of the valid display options for the EDIT statement. These options are listed under the EDIT statement.

Examples

These examples are based on the report built with the following commands:

```sql
select dept-id-0410, emp-last-name-0415, salary-amount-0420
from department, employee, emposition
where dept-employee and emp-emposition
compute average-salary=avg(salary-amount-0420)
group by dept-id-0410 ! display

DEPARTMENT/EMPLOYEE/EMPOSITION REPORT
mm/dd/yy

DEPT-ID-0410  EMP-LAST-NAME-0415  SALARY-AMOUNT-0420
```
### Edit Olqheader

You can change the header for the AVERAGE-SALARY field:

```plaintext
edit computation average-salary
olqheader 'avg-sal for dept $dept-id-0410' ! display
```

<table>
<thead>
<tr>
<th>DEPT-ID-0410</th>
<th>EMP-LAST-NAME-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>6666</td>
<td>HENDON</td>
<td>240000.00</td>
</tr>
<tr>
<td>6666</td>
<td>PAPAZEUS</td>
<td>100000.00</td>
</tr>
<tr>
<td>6666</td>
<td>PAPAZEUS</td>
<td>90000.00</td>
</tr>
<tr>
<td>6666</td>
<td>RUPEE</td>
<td>80000.00</td>
</tr>
<tr>
<td>6666</td>
<td>RUPEE</td>
<td>76000.00</td>
</tr>
<tr>
<td>6666</td>
<td>WILDER</td>
<td>90000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVG-SAL FOR</td>
<td>6666</td>
<td>112666.66</td>
</tr>
</tbody>
</table>

| 2000         | BLOOMER            | 15000.00           |
| 2000         | HUTTON             | 44000.00           |
| 2000         | JENSON             | 82000.00           |
| 2000         | KIMBALL            | 45000.00           |
| 2000         | KING               | 14500.00           |

- 1 -

### Edit $, Commas

You can include dollar signs and commas in the computed field:

```plaintext
edit computation average-salary $ commas ! display
```

<table>
<thead>
<tr>
<th>DEPT-ID-0410</th>
<th>EMP-LAST-NAME-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>6666</td>
<td>HENDON</td>
<td>240000.00</td>
</tr>
<tr>
<td>6666</td>
<td>PAPAZEUS</td>
<td>100000.00</td>
</tr>
<tr>
<td>6666</td>
<td>PAPAZEUS</td>
<td>90000.00</td>
</tr>
<tr>
<td>6666</td>
<td>RUPEE</td>
<td>80000.00</td>
</tr>
<tr>
<td>6666</td>
<td>RUPEE</td>
<td>76000.00</td>
</tr>
<tr>
<td>6666</td>
<td>WILDER</td>
<td>90000.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVG-SAL FOR</td>
<td>6666</td>
<td>$112,666.66</td>
</tr>
</tbody>
</table>

| 2000         | BLOOMER            | 15000.00           |
| 2000         | HUTTON             | 44000.00           |
| 2000         | JENSON             | 82000.00           |
| 2000         | KIMBALL            | 45000.00           |

- 1 -
EDIT GROUP

EDIT GROUP allows you to edit fields defined by COMPUTE ... GROUP BY. You can also specify how many lines to skip between groups and define the separator character that separates a grouping from its computed value.

Syntax

```
EDIT GROUP

group-by-expression

ALL

LEVEL n

SKIP n lines/PAGE

SEPARATOR CHARACTER separator-character
```

Parameters

- **group-by-expression**
  An expression specified in the GROUP BY clause of the COMPUTE command is edited. This `group-by-expression` must identically match the expression specified in the GROUP BY clause of the COMPUTE command.

- **ALL**
  All of the expressions specified in the GROUP BY clause of the COMPUTE command are edited.

- **LEVEL n**
  Specifies the break level, in numeric order, with 1 representing the highest level. The default `n` value is 1.

- **SKIP n LINES/PAGE**
  Specifies to skip `n` lines or a page between the computed expression and the next grouping.

- **SEPARATOR CHARACTER**
  Defines the character that separates the grouping from its computed expression. `Separator-character-value` must be one character in length.

Example

If you specify a separator character of - (hyphen), your report would look like:
END PATH

END PATH terminates path definition. Subsequent path commands delete the existing path.

**Access mode:** The syntax below is invalid when the access switch is set to IDMS.

**Syntax**

```
```  

EXECUTE PATH

EXECUTE PATH executes the retrieval commands specified in the database path definition and builds a report file of retrieved records.

**Access mode:** The syntax below is invalid when the access switch is set to IDMS.

**Syntax**

```
```  

FIELDS FOR

FIELDS FOR modifies the internal field list for a record. Use this command to reduce the size of report files. The field list you specify applies to all subsequent retrievals of the named record type. FIELDS FOR can be associated with either database record retrieval or logical record retrieval.

**Access mode:** The syntax below is invalid when the access switch is set to IDMS.
Syntax

```
FIELDS FOR view-id record-name logical-record-name

ARE
ARE ALSO
ARE NOT

(field-list)

DISPLAY
HEXADECIMAL
COMmas
NOCommas
$ NO$
LEAdzeros
NOLeadzeros
SPArse
FULl
```

Parameters

- **view-id**
  The qualifying ID for the record or logical record name. Use `view-id` when you are signed on to multiple subschemas. `View-id` refers to the subschema where the record or logical record can be found.

  - **record-name** -- The database record type to which field list modifications apply.
  - **logical-record-name** -- The logical record type to which field list modifications apply.

- **ARE/ARE ALSO/ARE NOT**
  - **ARE** -- The named fields replace those in the internal field list.
  - **ARE ALSO** -- The named fields are added to the internal field list.
  - **ARE NOT** -- The named fields are deleted from the internal field list.

  - **(field-list)** -- The database record or logical record fields to be substituted for, added to, or deleted from the existing field list.

- **DISPLAY/HEXADECIMAL**
  Specifies whether named fields are output in display or hexadecimal format.

- **COMMAS/NOCOMMAS**
  Specifies whether the named fields are displayed with or without commas. Nonnumeric fields are unaffected.

- **$ NO$**
  Specifies whether named fields are displayed with or without dollar signs. Nonnumeric fields are unaffected.

- **LEADZEROS/NOLEADZEROS**
  Specifies whether the numeric fields in the field list are displayed with or without leading zeros; nonnumeric fields are unaffected.
- **SPARSE/FULL**
  Specifies whether repeating column values are displayed or not.

**Examples**

The fields of the JOB record are listed below:

<table>
<thead>
<tr>
<th>JOB</th>
<th>PAGE 1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(LEVEL)</td>
<td>(FIELD NAME)</td>
</tr>
<tr>
<td>02</td>
<td>JOB-ID-0440</td>
</tr>
<tr>
<td>02</td>
<td>TITLE-0440</td>
</tr>
<tr>
<td>03</td>
<td>DESCRIPTION-LINE-0440</td>
</tr>
<tr>
<td>02</td>
<td>REQUIREMENTS-0440</td>
</tr>
<tr>
<td>03</td>
<td>REQUIREMENT-LINE-0440</td>
</tr>
<tr>
<td>02</td>
<td>MINIMUM-SALARY-0440</td>
</tr>
<tr>
<td>02</td>
<td>MAXIMUM-SALARY-0440</td>
</tr>
<tr>
<td>02</td>
<td>SALARY-GRADES-0440</td>
</tr>
<tr>
<td>02</td>
<td>NUMBER-OF-POSITIONS-0440</td>
</tr>
<tr>
<td>02</td>
<td>NUMBER-OPEN-0440</td>
</tr>
<tr>
<td>02</td>
<td>FILLER</td>
</tr>
</tbody>
</table>

**Fields Are**

When you issue a retrieval command for the JOB record without a field-list, you receive all the fields above providing OPTION ALL is in effect. You can, however, modify the field-list before or after retrieval with a FIELDS FOR command:

fields for job are (job-id-0440, title-0440)

The internal field list becomes:

JOB-ID-0440
TITLE-0440

**Fields Are Also**

If you want to add fields to the field list:

fields for job are also (minimum-salary-0440)

The internal field list becomes:

JOB-ID-0440
TITLE-0440
MINIMUM-SALARY-0440

**Fields Are Not**

You can omit fields from the field list:

fields for job are not (title-0040)

The internal field list becomes:

JOB-ID-0440
MINIMUM-SALARY-0440
FIND / GET Logical Record

FIND/GET logical record retrieves records by using paths (defined by DBAs) through the database. Retrieval of logical records continues until either the number of records specified in the retrieval command is obtained or an error in processing is encountered.

**Access mode:** The syntax below is **invalid** when the access switch is set to **IDMS**.

**FIND** locates database records but does not retrieve them into the report file. **GET** locates database records and does retrieve them into the report file.

This article describes the following information:

- Syntax (see page 67)
- Parameters (see page 67)
- Considerations (see page 68)
- Examples (see page 69)

### Syntax

```plaintext
►─┬─Find─┬─┬──────────────────────────────┬────────────────────────────────►
    └─GET ──┘ ├─First◄─┬────────────────┬─┤
        │       └─
record-count
        ├─Next─┬────────────────┬────┤
        │      └─
record-count
        └─ALL ─────────────────┘
►─┬────────────┬─┬────────────────┬─records──────►
    └─view-id.─┘ ──┴────────────────────────────────┘
logical-record-nameFIELD-LISTrecords
►─┬──────────────────┬───────────────────────────────────────────────────────►◄
    └─WHEREcriteria─┘
►─┬───Repeatwith─┬─┬────────────────────────────┬─────────────────────────────►
    ├─First◄─┬────────────────┬─┤
        │       └─
record-count
        ├─Next─┬────────────────┬──┤
        │      └─
record-count
        └─ALL ───────────────┘
►─┬─────────────────────────────────────────────────────────┬─records──►
    └─view-id.─┘ ──┬────────────────────────────────┘
logical-record-nameFIELD-LISTrecords
►─┬──────────────────┬───────────────────────────────────────────────────────►◄
    └─WHEREcriteria─┘
```

### Parameters

- **FIRST record-count**
  Retrieves the first \( n \) (where \( n \) defaults to 1) occurrences of the named logical record (default).

- **NEXT record-count**
  Retrieves the next \( n \) (where \( n \) defaults to 1) occurrences of the named logical record.

- **ALL**
  Retrieves all occurrences of the named logical record.
• **view-id**
  The qualifying ID for the logical record name. Use **view-id** when you are signed on to multiple subschemas. **View-id** refers to the subschema where the logical record can be found.

• **logical-record-name**
  Specifies the logical record to be retrieved (required).

• **field-list**
  Specifies fields within **logical-record-name** to be displayed in the report. **Field-list** must be enclosed in parentheses.
  When a logical record contains more than one field with the same name, field names entered in the field list can be qualified with the name of a database record. For example, the field name X is used in database records EMP and DEPT. If both of these fields are included in the logical record being retrieved, the user can distinguish them by specifying EMP.X or DEPT.X, respectively.

  **Note:** CA OLQ treats the database records that make up a logical record as group level fields. When you specify a database record name in the field list for a logical record, CA OLQ includes all fields from the named database record that are contained in the logical record.

• **WHERE**
  Specifies criteria to be used by CA OLQ when selecting a record occurrence.

• **REPEAT**
  Duplicates an immediately preceding FIND/GET LOGICAL RECORD (or REPEAT for FIND/GET LOGICAL RECORD) command, with modifications as specified.

• **FIRST record-count**
  Repeats the previous command and specifies the retrieval of the first \( n \) record occurrences.

• **NEXT record-count**
  Repeats the previous command and specifies the retrieval of the next \( n \) record occurrences.

• **ALL**
  Repeats the previous command and specifies the retrieval of all occurrences.

  • **logical-record-name** -- The logical record name being retrieved
    • **field-list** -- The logical record fields being retrieved

• **WHERE**
  Specifies the criteria used in selecting records.

### Considerations

Path status is a Logical Record Facility (LRF) concept used to indicate the result of a logical record retrieval request. Path statuses can be system- or user-defined. System-defined path statuses are:

• **LR-FOUND** -- Returned when a logical record request has been successfully executed
LR-NOT-FOUND -- Returned when LRF is unable to construct the requested logical record because one or more necessary database occurrences are not found

LR-ERROR -- Returned when LRF is unable to construct the requested logical record because of an error

A report file is built whenever the path status is LR-FOUND. Alternatively, a report file will be built for any user-defined path status if OPTIONS=PATHSTATUS is in effect.

Examples

Get Logical Record

A GET logical record command retrieves the first EMP-JOB-LR logical record:

```
get first emp-job-lr (emp-name-0415 dept-name-0410 title-0440)
EMP-JOB-LR
  EMP-NAME-0415 :
    EMP-FIRST-NAME-0415 : 'PHINEAS'  
    EMP-LAST-NAME-0415 : 'FINN'       
  DEPT-NAME-0410 : 'THERMOREGULATION' 
  TITLE-0440      : 'KEEPER OF BALLOONS'
END OF RECORD
```

Repeat Get

A REPEAT command retrieves the first record that meets the specified selection criteria:

```
repeat with first emp-job-lr where dept-name-0410 = thermoregulation
EMP-JOB-LR
  EMP-NAME-0415 :
    EMP-FIRST-NAME-0415 : 'PHINEAS'  
    EMP-LAST-NAME-0415 : 'FINN'       
  DEPT-NAME-0410 : 'THERMOREGULATION' 
  TITLE-0440      : 'KEEPER OF BALLOONS'
END OF RECORD
```

Logical Record Keyword

A logical record keyword is used in the following example to retrieve the first record for which the TITLE-0440 field value is PROGRAMMER/ANALYST:

```
get first emp-job-lr where progrmr-analysts
EMP-JOB-LR
  EMP-NAME-0415 :
    EMP-FIRST-NAME-0415 : 'JAMES'     
    EMP-LAST-NAME-0415 : 'GALWAY'     
  DEPT-NAME-0410 : 'INTERNAL SOFTWARE' 
  TITLE-0440      : 'PROGRAMMER/ANALYST'
END OF RECORD
```

FIND / GET MOST RECENT

FIND/GET MOST RECENT retrieves the current of record type for the specified record name. If currency has not been established, an error occurs and the records cannot be retrieved.
Access mode: The syntax below is invalid when the access switch is set to IDMS.

**FIND** locates database records but does not retrieve them into the report file. **GET** locates database records and does retrieve them into the report file.

### Syntax

```
  ┌─ FIND ┐  most RECENT ┐
  └──────┘           ┘
  ┌─ GET    ┐ record ┘
  └──────┐  ┌─ view-id. ┐
             └─ record-name ┘
             ┌─ (field-list) ┘
             └─ record ┘
  ┌─ REPeat with ┐
  └──────┐  ┌─ view-id. ┐
             └─ record-name ┘
             ┌─ (field-list) ┘
             └─ record ┘
```

### Parameters

- **MOST RECENT**
  Retrieves the most recent occurrence of a particular record type.

- **view-id**
  The qualifying ID for the record or logical record name. Use **view-id** when you are signed on to multiple subschemas. View-id refers to the subschema where the record (or logical record) can be found.

- **record-name**
  The record type to be retrieved (required). Currency must be established for the named record. If a database record name is the same as a CA OLQ keyword, the name should be enclosed in quotation marks.

- **(field-list)**
  The fields within **record-name** to be displayed in the report file. **Field-list** must be enclosed in parentheses.

- **REPEAT**
  Duplicates an immediately preceding FIND/GET MOST RECENT (or REPEAT for FIND/GET MOST RECENT) command with modifications.

- **view-id**
  The qualifying ID for the record or logical record name. Use **view-id** when you are signed on to multiple subschemas. View-id refers to the subschema where the record (or logical record) can be found.

- **record-name**
  Specifies the database record type to retrieve.

  - **(field-list)** The fields to retrieve

Since a FIND/GET MOST RECENT command retrieves only one record, the REPEAT command will duplicate the same data unless the **field-list** parameter is used.
Examples

The following examples illustrate the use of the FIND/GET MOST RECENT and associated REPEAT commands.

Get Most Recent

A GET command retrieves the most recent DEPARTMENT record:

```
get most recent department (dept-name-0410 dept-id-0410)
```

```
DEPARTMENT
   DEPARTMENT-DBKEY : 2/5007103:1
   DEPT-ID-0410 : 5200
   DEPT-NAME-0410 : 'THERMOREGULATION'
END OF RECORD
```

Get with a Field List

A GET command retrieves the most recent EMPLOYEE record and specifies a field list:

```
get most recent employee (emp-id-0415 emp-city-0415)
```

```
EMPLOYEE
   EMPLOYEE-DBKEY : 1/5007045:1
   EMP-ID-0415 : 479
   EMP-CITY-0415 : 'EASTON'
END OF RECORD
```

Get Most Recent

A GET command retrieves the most recent OFFICE record:

```
get most recent office (office-code-0450 office-phone-0450(1))sp.1
```

```
OFFICE
   OFFICE-DBKEY : 0/5007135:1
   OFFICE-CODE-0450 : '005'
   OFFICE-PHONE-0450(1) : 4578123
END OF RECORD
```

FIND / GET PHYSICAL SEQUENTIAL

FIND/GET PHYSICAL SEQUENTIAL retrieves records based on their physical position in a database area.

**Access mode:** The syntax below is invalid when the access switch is set to IDMS.

FIND locates database records but does not retrieve them into the report file. GET locates database records and does retrieve them into the report file.

This article describes the following information:

- Syntax (see page 72)
- Parameters (see page 72)
- Examples (see page 74)
### Parameters

- **FIRST record-count**
  Retrieves the first $n$ (where $n$ defaults to 1) records at the beginning of the database area containing the named record type. The records retrieved are those with the lowest dbkey values.

- **LAST record-count**
  Retrieves the last $n$ (where $n$ defaults to 1) records at the end of the database area containing the named record type. The records retrieved are those with the highest dbkey values.

- **NEXT record-count**
  Retrieves the next $n$ (where $n$ defaults to 1) occurrences of the named record type with the next-highest dbkey value. Currency for retrieval is based on the last record retrieved in the same database area. If currency has not been established, record retrieval cannot occur and no report file is built.

  **Note:** Because NEXT records are retrieved based on current of area, the record retrieved may not always be the record required.

- **PRIOR record-count**
  Retrieves the previous $n$ (where $n$ defaults to 1) occurrences of the named record type with the next lower dbkey value. Currency for retrieval is based on the last record retrieved in the same database area. If currency has not been established, record retrieval cannot occur and no report file is built.
Note: Because PRIOR records are retrieved based on current of area, the record retrieved may not always be the record required.

- **ALL**
  Retrieves all occurrences of the specified record type within its associated database area.

- **PHYSICAL SEQUENTIAL**
  Specifies a serial sweep of the database. PHYSICAL and SEQUENTIAL are synonymous keywords; one is required, but both can be specified.
  - *view-id* -- The qualifying ID for the record or logical record name. Use *view-id* when you are signed on to multiple subschemas. *View-id* refers to the subschema where the record (or logical record) can be found.
  - *record-name* -- The record type to be retrieved. If a database record name is the same as a CA OLQ keyword, the name should be enclosed in quotation marks.

- **(field-list)**
  Specifies the fields within *record-name* to be displayed in the report file. *Field-list* must be enclosed in parentheses.

- **WHERE**
  Specifies criteria to be used by CA OLQ to select record occurrences.

- **REPEAT**
  Duplicates an immediately preceding FIND/GET PHYSICAL (or REPEAT for FIND/GET PHYSICAL) command.

- **FIRST record-count**
  Specifies that the first *n* records be retrieved.

- **LAST record-count**
  Specifies that the last *n* records be retrieved.

- **NEXT record-count**
  Specifies the next *n* records be retrieved.

- **PRIOR record-count**
  Specifies the prior *n* records be retrieved.

- **ALL**
  Specifies that all of the records be retrieved.

- **view-id**
  The qualifying ID for the record or logical record name. Use *view-id* when you are signed on to multiple subschemas. *View-id* refers to the subschema where the record (or logical record) can be found.

- **record-name**
  Specifies the database record type:
- (field-list) -- The fields to retrieve

If no parameters are specified, the previous command is duplicated. If the field-list parameter or WHERE clause is used, record-name must also be specified.

Examples

The following examples illustrate the use of the FIND/GET and associated REPEAT commands, based on the set occurrence diagram shown below:

FIND / GET PHYSICAL SEQUENTIAL

FIND / GET OWNER WITHIN SET

FIND/GET OWNER WITHIN SET retrieves the owner record of a database set occurrence. For all set membership options other than mandatory automatic (that is, mandatory manual, optional manual, and optional automatic), an occurrence of a member record type need not be a member of a set occurrence unless the parameter BELONGING TO is specified.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

FIND locates database records but does not retrieve them into the report file. GET locates database records and does retrieve them into the report file.

Syntax
Parameters

- **OWNER**
  Specifies the retrieval of the owner record. This parameter is required if `owner-record-name` is not specified.

- **view-id**
  The qualifying ID for the record or logical record name. Use `view-id` when you are signed on to multiple subschemas. `View-id` refers to the subschema where the record (or logical record) can be found.

- **owner-record-name**
  Specifies the owner record type in the set. Owner-record-name must be used if the OWNER keyword is not specified. If `member-record-name` and `set-name` are not specified, the owner record type must be the owner of only one set type in the subschema view.
  
  - `(field-list)` -- The fields within `owner-record-name` to be displayed in the report file. Field-list must be enclosed in parentheses.

- **BELONGING TO THIS member-record-name**
  Retrieves the owner record of the current member record type. If `owner-record-name` is not used, the member record type must participate as member in only one set in the subschema view. If both member and owner record types are named, the specified records must participate as member and owner within only one set in the subschema view.

- **WITHIN set-name**
  Specifies the `owner-record-name` set and retrieves the owner occurrence of that set type. You must use `set-name` if the owner record participates as owner in more than one set in the subschema view. Using `set-name` may be required to resolve ambiguity.

- **WHERE**
  Specifies criteria for selecting a record occurrence. If you specify a WHERE clause, you must also specify the `owner-record-name`.

- **REPEAT**
  Duplicates an immediately preceding FIND/GET OWNER WITHIN SET (or REPEAT for FIND/GET OWNER WITHIN SET) command.

- **OWNER= owner-record-name**
  Specifies retrieval of the owner record type.
  
  - `(field-list)` -- The owner record fields retrieved

- **WHERE**
  Specifies criteria used in selecting record occurrences.
Examples

The following examples illustrate the use of FIND/GET OWNER WITHIN SET and associated REPEAT commands based on the set occurrence diagram shown below:

![Set Occurrence Diagram]

**FIND / GET OWNER WITHIN SET**

*Get First Physical*

GET enters the database and establishes currency:

```plaintext
get first phys office (office-code-0450 office-phone-0450(1)) where calckey = 002
OFFICE
  OFFICE-DBKEY : 0/5007132:1
  OFFICE-CODE-0450 : '002'
  OFFICE-PHONE-0450(1) : 9562377
END OF RECORD
```

*Repeat Get*

A REPEAT command retrieves the next OFFICE record in the database:

```plaintext
repeat with next
OFFICE
  OFFICE-DBKEY : 0/5007142:1
  OFFICE-CODE-0450 : '003'
  OFFICE-PHONE-0450(1) : 3297700
END OF RECORD
```

**FIND / GET Using Storage Key**

FIND/GET using storage key retrieves records based on their CALC key or dbkey value.

*Access mode*: The syntax below is invalid when the access switch is set to IDMS.

**FIND** locates database records but does not retrieve them into the report file. **GET** locates database records and does retrieve them into the report file.
This article describes the following information:

- Syntax (see page 77)
- Parameters (see page 78)
- Examples (see page 81)

### Syntax

 expansion of record

```plaintext
view-id. record-name (field-list) records
WHEre CALckey calc-field-name = record. field-name
CALckey = concatenated-key-value
DBKey = db-key-value
AND criteria
REPeat with
view-id. record-name (field-list) records
WHEre CALckey calc-field-name = record. field-name
CALckey = concatenated-key-value
DBKey = db-key-value
AND criteria
```

expansion of record
Parameters

- **FIRST record-count**
  Retrieves the first \( n \) (where \( n \) defaults to 1) records in the database that have the specified key value. Multiple records with the same CALC key are known as duplicates. Multiple records with the same dbkey value are not allowed.

- **NEXT record-count**
  Retrieves the next \( n \) (where \( n \) defaults to 1) records in the database that have the specified key value.

- **ALL**
  Retrieves all records in the database that have the specified key value.

- **view-id**
  The qualifying ID for the record or logical record name. Use view-id when you are signed on to multiple subschemas. View-id refers to the subschema where the record (or logical record) can be found.

- **record-name**
  The record type to be retrieved (required). If a database record name is the same as a CA OLQ keyword, the name should be enclosed in quotation marks.

  - *(field-list)* -- The fields within record-name to be in the report file. Field-list must be enclosed in parentheses.

- **WHERE**
  Specifies the criteria for selecting records:

  - **CALCKEY=** -- The CALC key used for record retrieval. For duplicate CALC keys, the order of retrieval depends on the DUPLICATES specification in the schema record description.

    - **Note:** If the CALC key defined for the named record type is a group item, all subordinate fields must be specified.

    - **calc-field-name** -- The field within record-name containing the CALC key, or the name of any field in the record that redefines the CALC key.

      - **record** -- The qualifying record name for a field-name. Expanded syntax can be found following the REPEAT parameters.

      - **field-name** -- The field containing the CALC key value.

      - **calc-key-value** -- The associated values of the CALC key or calc-field-name used for record retrieval. Separate multiple values with blanks or commas.

    - **CALCKEY=** -- A concatenated CALC key value used for record retrieval.
• **concatenated-key-value** -- The concatenated key value of the record to be retrieved. Separate each of the partial key values with blanks or commas.

• **DBKEY=** -- A dbkey value used for record retrieval:
  
  • **record** -- The qualifying record name for a field-name. Expanded syntax can be found following the REPEAT parameters.
  
  • **field-name** -- The field name containing the dbkey value.
  
  • **dbkey-value** -- The dbkey value of the record to be retrieved.

Separate multiple values with blanks or commas.

**Note:** The NEXT ordinal clauses cannot be specified for dbkey value retrieval.

• **AND criteria**
  Specifies selection criteria used for retrieving record occurrences.

• **REPEAT**
  Duplicates an immediately preceding FIND/GET using storage key (or REPEAT for FIND/GET using storage key) command.

• **FIRST record-count**
  Specifies the retrieval of the first n record occurrences.

• **NEXT record-count**
  Specifies the retrieval of the next n record occurrences.

• **ALL**
  Specifies the retrieval of all record occurrences.

• **view-id**
  The qualifying ID for the record or logical record name. Use view-id when you are signed on to multiple subschemas. View-id refers to the subschema where the record (or logical record) can be found.

• **record-name**
  Specifies the database record name.

  • **(field-list)** -- The fields to be retrieved

• **WHERE**
  Specifies criteria used in selecting records:

  • **CALCKEY=** -- The CALC key is used for record retrieval. For duplicate CALC keys, the order of retrieval depends on the DUPLICATES specification in the schema record description.
- *calc-field-name* -- The field within *record-name* containing the CALC key, or the name of any field in the record that redefines the CALC key.

- *record* -- Specifies the database record that qualifies *field-name*. Expanded syntax for *record* following the REPEAT parameter.

- *field-name* -- The name of the field containing the CALC key value.

- *calc-key-value* -- The associated values of the CALC key or *calc-field-name* used for record retrieval. Separate multiple values with commas or blanks.

- *CALCKEY=* -- A concatenated CALC key value is used for record retrieval.

  - *concatenated-key-value* -- The concatenated key value of the record retrieved. Separate each of the partial key values with blanks or commas.

- *DBKEY=* -- A dbkey value is used for record retrieval:

  - *record* -- The qualifying record name for a *field-name*. Expanded syntax can be found following the REPEAT parameters.

  - *field-name* -- The name of the field containing the dbkey value.

  - *dbkey-value* -- The dbkey value of the record retrieved.

  Separate multiple values with blanks or commas.

- **Note:** The NEXT ordinal clauses cannot be specified for dbkey value retrieval.

- *AND criteria*  
  Specifies criteria used in selecting record occurrences.  
  If no parameters are specified and the last retrieval was performed by database-key, the same record is retrieved. The CALC key or database key does not change unless a change is specified in the REPEAT command; any new key value replaces the previous value.

- *record*  

  - *view-id* -- The qualifying ID for the record or logical record name. Use *view-id* when you are signed on to multiple subschemas. *View-id* refers to the subschema where the record (or logical record) can be found.

  - *record-name (occurrence)* -- The record in the path where the *field-name* occurs.  
    If the *field-name* occurs in more than one record, use *record-name*.  
    If the record occurs more than once in the same path, use *occurrence*. If a database record name is the same as a CA OLQ keyword, the name should be enclosed in quotation marks.
Examples

The following examples illustrate the use of FIND/GET using storage key and associated REPEAT commands, based on the set occurrence diagram shown below:

FIND / GET Using Storage Key

Get Using Storage Key

A GET using storage key command retrieves the first DEPARTMENT record with a dbkey value of 5007108:1:

get first department (dept-id-0410,dept-name-0410) where calckey=200

DEPARTMENT
  DEPT-ID-0410 : 2000
  DEPT-NAME-0410 : ACCOUNTING AND PAYROLL
END OF RECORD

Repeat Get

A REPEAT command with a WHERE clause retrieves the same fields as the previous command for the new database-key value:

repeat with department where calckey=1000

DEPARTMENT
  DEPT-ID-0410 : 1000
  DEPT-NAME-0410 : PERSONNEL
END OF RECORD

FIND / GET WITHIN DBKEYLIST

FIND/GET WITHIN DBKEYLIST retrieves records, based on the list of dbkeys collected during previous retrieval commands.

Access mode: The syntax below is invalid when the access switch is set to IDMS.
CA OLQ automatically stores the database key of each record as it is retrieved (when OPTION DBKEY is specified). When a path definition is executed, CA OLQ stores only database keys for record occurrences associated with the primary record type in the path. The resulting dbkey list provides the basis for GET WITHIN DBKEYLIST retrieval.

Use dbkey list retrieval to optimize retrieval of a large number of records using FIND. Issue GET WITHIN DBKEYLIST commands to specify progressively more restrictive selection criteria until your exact retrieval requirements are met. The optional WHERE clause is often associated with this command.

Note: The GET WITHIN DBKEYLIST command cannot be used to access Key Sequence Data Set (KSDS) VSAM files; database keys have no meaning for KSDS VSAM records.

FIND locates database records but does not retrieve them into the report file. GET locates database records and does retrieve them into the report file.

This article describes the following information:

- Syntax (see page 82)
- Parameters (see page 82)
- Examples (see page 83)

Syntax

```
FIND
GET
FIRST record-count
ALL
view-id.
record-name (field-list)
WITHIN DBKEYLIST
WHERE criteria
REPEAT with
FIRST record-count
ALL
view-id.
record-name (field-list)
WHERE criteria
```

Parameters

- **FIRST record-count**
  Retrieves the first \( n \) (where \( n \) defaults to 1) records in the database key list.
ALL
Retrieves all the records in the dbkey list.

view-id
The qualifying ID for the record or logical record name. Use view-id when you are signed on to multiple subschemas. View-id refers to the subschema where the record (or logical record) can be found.

record-name
The name of the record type last retrieved. Use record-name if you specify a field list. If a database record name is the same as a CA OLQ keyword, enclose the name in quotation marks.

(field-list) -- The fields within record-name to be stored in the report file. Field-list must be enclosed in parentheses.

WITHIN DBKEYLIST
Specifies retrieval of records directly, by means of the database key list (required).

WHERE
Specifies criteria used in selecting a record occurrence.

REPEAT
REPEAT FIND/GET WITHIN DBKEYLIST duplicates an immediately preceding FIND/GET WITHIN DBKEYLIST (or REPEAT for FIND/GET WITHIN DBKEYLIST) command.

FIRST record-count
Retrieves the first n (where n defaults to 1) records in the database key list.

ALL
Retrieves all the records in the database key list.

view-id
The qualifying ID for the record or logical record name. Use view-id when you are signed on to multiple subschemas. View-id refers to the subschema where the record (or logical record) can be found.

record-name
The record type last retrieved. Use record-name if specifying a field list. Enclose the database record in quotation marks if it is the same as a CA OLQ keyword.

(field-list) -- The fields within record-name. Enclose field-list in parentheses.

WHERE
Specifies criteria used in selecting a record occurrence.

Examples
The following examples illustrate the use of FIND/GET WITHIN DBKEYLIST and associated REPEAT commands, based on the set occurrence diagram shown below:
FIND / GET WITHIN DBKEYLIST

Find

FIND retrieves 56 EMPLOYEE records from the database:

```
option dbkey !find all sequential employee records
OLQ 098006 00 57 whole lines and 0 partial lines in report.
OLQ 098007 00 57 records read. 57 records selected.
```

Get Within Dbkeylist

A GET WITHIN DBKEYLIST command retrieves all the EMPLOYEE records in which the EMP-CITY-0415 field value is WESTWOOD:

```
get all within dbkeylist where emp-city-0415 is westwood
OLQ 098006 00 3 whole lines and 0 partial lines in report.
OLQ 098007 00 50 records read. 3 records selected.
```

Repeat Within Dbkeylist

A REPEAT WITHIN DBKEYLIST selects all the EMPLOYEE records in which the EMP-CITY-0415 field value is ARLINGTON:

```
repeat with all where emp-city-0415 is arlington
OLQ 098006 00 3 whole lines and 0 partial lines in report.
OLQ 098007 00 50 records read. 3 records selected.
```

FIND / GET WITHIN Index SET

FIND/GET WITHIN index SET retrieves records using the name of an index set and the index-sort-key fields specified in the WHERE clause.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

FIND locates database records but does not retrieve them into the report file. GET locates database records and does retrieve them into the report file.

This article describes the following information:

- Syntax (see page 85)
- Parameters (see page 85)
- Examples (see page 87)
Syntax

```
FINd GET
     │  ┌─ FIRst record-count
     │  └─ LASt record-count
     │     ├─ NEXt record-count
     │     │   └─ PRIor record-count
     │     └─ ALL
     └─ view-id. record-name (field-list)

     withIN index-set-name set

     WHERE index-criteria AND criteria

     REPeat with
     │  ┌─ FIRst record-count
     │  └─ LASt record-count
     │    ├─ NEXt record-count
     │    │   └─ PRIor record-count
     │    └─ ALL
     └─ view-id. record-name (field-list)

     WHERE index-criteria AND criteria
```

Parameters

- **FIRST record-count**
  Retrieves the first \( n \) (where \( n \) defaults to 1) records at the beginning of the current occurrence of the set type.

- **LAST record-count**
  Retrieves the last \( n \) (where \( n \) defaults to 1) records at the end of the current occurrence of the set type.

- **NEXT record-count**
  Retrieves the next \( n \) (where \( n \) defaults to 1) records, starting with the record that follows the current record in the current occurrence of the set.

- **PRIOR record-count**
  Retrieves the prior \( n \) (where \( n \) defaults to 1) records starting with the record that precedes the current record in the current occurrence of the set.

- **ALL**
  Retrieves all member records in the current occurrence of the set type.
• **view-id**
  The qualifying ID for the record or logical record name. Use `view-id` when you are signed on to multiple subschemas. `View-id` refers to the subschema where the record (or logical record) can be found.

• **record-name**
  The name of the indexed record to be retrieved. If the WITHIN SET clause does not specify the index-set name, use `record-name`. `Record-name` must be a record in an index set. If you have a WHERE clause, specify `record-name`. If a database record name is the same as a CA OLQ keyword, the name should be enclosed in quotation marks.

  - *(field-list)* -- Specifies the fields within `record-name` to be displayed in the report file. `Field-name` must be enclosed in parentheses.

• **WITHIN index-set-name**
  Specifies the name of the index set used for retrieval. If `record-name` is not specified or if `record-name` participates in more than one index set, this parameter is required.

• **WHERE**
  Specifies comparison expressions based on the index field or its subfields. The criteria is evaluated for each entry in the index; a record is retrieved only when an index satisfies the conditions specified in this parameter.

• **AND criteria**
  Specifies criteria to be used by CA OLQ when selecting a record occurrence.

• **REPEAT**
  Duplicates an immediately preceding FIND/GET WITHIN index SET (or REPEAT for FIND/GET WITHIN index SET) command.

• **FIRST record-count**
  Retrieves the first `n` (where `n` defaults to 1) records at the beginning of the current set.

• **LAST record-count**
  Retrieves the last `n` (where `n` defaults to 1) records at the end of the current set.

• **NEXT record-count**
  Retrieves the next `n` (where `n` defaults to 1) records starting with the record that follows the current record in the current set.

• **PRIOR record-count**
  Retrieves the prior `n` (where `n` defaults to 1) records starting with the record that precedes the current record in the current set.

• **ALL**
  Retrieves all member records in the current set.

• **view-id**
  The qualifying ID for the record or logical record name. Use `view-id` when you are signed on to multiple subschemas. `View-id` refers to the subschema where the record (or logical record) can be found.
- **record-name**
  The indexed record to be retrieved. Use this parameter if the WITHIN SET clause does not specify the index-set name. *Record-name* must be a record in an index set. If you specify a WHERE clause, also specify *record-name*. Enclose the database record name in quotation marks if it is the same as a CA OLQ keyword.

  - *(field-list)* -- The fields within *record-name* enclosed within parentheses.

- **WHERE**
  Specifies comparison expressions based on the index field or its subfields. The criteria is evaluated for each entry in the index; a record is retrieved only when an index satisfies the conditions specified in this parameter.
  The rules for *index-criteria* are the same as for the WHERE clause *criteria* parameter.

- **AND criteria**
  Specifies criteria used in selecting a record occurrence.
  If the selection criteria are changed and index-set retrieval is preferred, specify the record name in the REPEAT command.

### Examples

The following examples illustrate the use of the GET WITHIN index SET command and associated REPEAT commands based on the set occurrence diagram shown below:

**FIND / GET WITHIN Index SET**

*Get First*

A GET command retrieves the first SKILL record in the SKILL-NAME-NDX index set:

```
get first skill (skill-id-0455 skill-name-0455 skill-dbkey) record in skill-name-ndx
set SKILL
```
Repeat

A REPEAT command retrieves the next SKILL record in the SKILL-NAME-NDX index set:

```
repeat with next
  SKILL
  SKILL-DBKEY     :  0/5007107:1
  SKILL-ID-0455   :  1040
  SKILL-NAME-0455 :  'DEV MGT  '  
END OF RECORD
```

**FIND / GET WITHIN SET**

FIND/GET WITHIN SET retrieves records based on their membership in a database set. Use this command to retrieve records only after currency is established within the object set. (Currency need not have been established previously for system-owned index sets.)

**Access mode:** The syntax below is invalid when the access switch is set to IDMS.

**FIND** locates database records but does not retrieve them into the report file. **GET** locates database records and does retrieve them into the report file.

This article describes the following information:

- Syntax (see page 88)
- Parameters (see page 89)
- Considerations (see page 91)
- Examples (see page 91)

**Syntax**

```
FIND
  GET
  FIrst record-count
  LASt record-count
  Next record-count
  PRIor record-count
  ALL

  BELonging to this member-record-name record
  BElonging to this view-id. member-record-name record
  withIN set-name set
  WHEre criteria
```

**Parameters**

- **view-id.**
- **member-record-name**
- **field-list**
- **set-name**
- **criteria**

**Examples**

```
FIND
  GET
  WITHIN set-name set
  WHEre criteria
```
Parameters

- **FIRST record-count**
  Retrieves the first $n$ (where $n$ defaults to 1) records at the beginning of the current occurrence of the set type.

- **LAST record-count**
  Retrieves the first $n$ (where $n$ defaults to 1) records at the end of the current occurrence of the set type.

- **NEXT record-count**
  Retrieves the next $n$ (where $n$ defaults to 1) records, starting with the record following the current record in the current occurrence of the set.

- **PRIOR record-count**
  Retrieves the prior $n$ (where $n$ defaults to 1) records, starting with the record preceding the current record in the current occurrence of the set. If no PRIOR pointers have been defined for the record, CA OLQ displays an error message.

- **ALL**
  Retrieves all member records in the current occurrence of the set type.

- **view-id**
  The qualifying ID for the record or logical record name. Use **view-id** when you are signed on to multiple subschemas. **View-id** refers to the subschema where the record (or logical record) can be found.

- **member-record-name**
  Specifies a member record type in the set. If a member record type is named, the retrieval applies only to records of that type. If no member record type is specified, all member record types in the set are retrieved.

  - **(field-list)** -- Fields within **member-record-name** to be stored in the report file. **Field-list** must be enclosed in parentheses.

- **BELONGING TO THIS owner-record-name**
  Specifies the owner record type in the set. If you don’t specify **set-name or member-record-name**, then the specified owner record type must participate as owner in only one set in the subschema view.
- **WITHIN set-name**
  Specifies the set where the participating record is retrieved. Set-name is required if the named or implied member record participates in more than one set. It is optional if the set name can be determined from the named member or owner record type.

  **Note:** To retrieve all member record types in the set, specify either set-name or owner-record-name, but do not specify member-record-name.

- **WHERE**
  Specifies criteria used to select record occurrences.

- **REPEAT**
  Duplicates an immediately preceding FIND/GET WITHIN SET (or REPEAT for FIND/GET WITHIN SET) command.

- **FIRST record-count**
  Retrieves the first $n$ (where $n$ defaults to 1) records at the beginning of the current set.

- **LAST record-count**
  Retrieves the first $n$ (where $n$ defaults to 1) records at the end of the current set.

- **NEXT record-count**
  Retrieves the next $n$ (where $n$ defaults to 1) records, starting with the record that follows the current record in the current set.

- **PRIOR record-count**
  Retrieves the prior $n$ (where $n$ defaults to 1) records, starting with the record that precedes the current record in the current set. If no PRIOR pointers have been defined for the record, CA OLQ displays an error message.

- **ALL**
  Retrieves all member records in the current set.

- **view-id**
  The qualifying ID for the record or logical record name. Use view-id when you are signed on to multiple subschemas. View-id refers to the subschema where the record (or logical record) can be found.

- **member-record-name**
  The member record type in the set. If a member record type is named, the retrieval applies only to records of that type. If no member record type is specified, all member record types in the set are retrieved.

  - **(field-list)** -- The fields within member-record-name to be stored in the report file. Enclose field-list in parentheses.

- **WHERE**
  Specifies criteria used in selecting record occurrences.
Considerations

Use the member-record-name, owner-record-name, and set-name clauses to resolve ambiguity. At least one of these clauses must be specified.

If member-record-name is specified, but owner-record-name and set-name are not specified as listed in the syntax, be sure the named member record type participates as member in only one set in the subschema view of the database.

Examples

The following examples illustrate the use of FIND/GET WITHIN SET and associated REPEAT commands based on the set occurrence diagram shown below:

```
FIND / GET OWNER WITHIN SET

Get First

A GET command is used to enter the database and establish currency:

```
get first office (office-code-0450 office-phone-0450(1)) where calckey = 002
  office
    OFFICE-CODE-0450 : '002'
    OFFICE-PHONE-0450(1) 9562377
  END OF RECORD
```

Get With Set

A GET WITHIN SET command is used to retrieve the first EMPLOYEE record in the OFFICE-EMPLOYEE set:

```
get first employee (emp-last-name-0415 emp-first-name-0415) in office-employee set
  employee
    EMP-FIRST-NAME-0415 : 'MICHAEL'
    EMP-LAST-NAME-0415 : 'ANGELO'
  END OF RECORD
```

Repeat Next
A REPEAT NEXT command duplicates the immediately preceding GET command for the next EMPLOYEE record:

```
repeat next
  EMPLOYEE
    EMP-FIRST-NAME-0415 : 'MONTE     '
    EMP-LAST-NAME-0415 : 'BANK         '
  END OF RECORD
```

**FIND / GET WITHIN SET Using SORTKEY**

FIND/GET WITHIN SET using SORTKEY retrieves member records in sorted database sets based on a specified sort key.

**Access mode:** The syntax below is **invalid** when the access switch is set to **IDMS**.

**FIND** locates database records but does not retrieve them into the report file. **GET** locates database records and does retrieve them into the report file.

This article describes the following information:

- Syntax (see page 92)
- Parameters (see page 93)
- Considerations (see page 96)
- Examples (see page 96)

**Syntax**

```
  FIND          GET
    FIRst       NEXt
    record-count
    record-count

  view-id.    member-record-name  (field-list) records

  BELONGing to this    view-id.    owner-record-name record

  within sorted-set-name set

  WHERE    SORtkey = sort-key-value
            sort-field-name = field-name
            record.
            SORtkey = -▼- concatenated-key-value

  AND criteria

  REPeat with
    FIRst
    NEXt
    record-count
    record-count

  member-record-name records
```
Parameters

- **FIRST record-count**
  Retrieves the first \( n \) (where \( n \) defaults to 1) records in the current occurrence of the set type with the specified sort key value.

- **NEXT record-count**
  Retrieves the next \( n \) (where \( n \) defaults to 1) records in the current occurrence of the set type with the specified sort key value.

  **Note:** Records with the same sort key (duplicates) can be retrieved by specifying the REPEAT WITH NEXT command without changing the preceding sort key value. If retrieval is requested for a record with a different sort key value, use the GET FIRST form of the command.

- **ALL**
  Retrieves all records in the current occurrence of the set type with the specified sort key value.

- **view-id**
  The qualifying ID for the record or logical record name. Use **view-id** when you are signed on to multiple subschemas. **View-id** refers to the subschema where the record (or logical record) can be found.

- **member-record-name**
  The member record type in the set (required).

  ▪ **(field-list)** -- The fields within **record-name** to be stored in the report file.

  *Field-list* must be enclosed in parentheses.

- **BELONGING TO THIS owner-record-name**
  Specifies the owner record type in the set. This clause is required only as needed to resolve ambiguity.

- **WITHIN sorted-set-name**
  Specifies the set type. This clause is required only as needed to resolve ambiguity.
WHERE
Supplies the criteria for selecting records:

- `SORTKEY=` -- The sort key used for record retrieval.
- `sort-field-name=` -- The name of the sort key field used for record retrieval.

⚠️ **Note:** If the sortkey is defined as more than one field, the SORTKEY keyword should be used rather than a list of the individual field names.

- `sort-key-value` -- The sort key value of the member record retrieved. In the event of duplicate sort keys, the order of retrieval depends on the DUPLICATES specification for the set type.

If the sort key is a group item, all subordinate fields must be specified.

- `record` -- The qualifying record name for a `field-name`. Expanded syntax can be found following the REPEAT parameter.
- `field-name` -- The name of the field containing the sort key value.
- `SORTKEY=concatenated-key-value` -- A concatenated sort key value used for record retrieval.
- In the `concatenated-key-value`, separate each of the partial key values with blanks or commas.

- **AND criteria**
  Specifies criteria used in selecting a record occurrence.

- **REPEAT**
  REPEAT GET WITHIN SET using SORTKEY duplicates an immediately preceding FIND/GET WITHIN SET using SORTKEY (or REPEAT for FIND/GET WITHIN SET using SORTKEY) command.

- **FIRST record-count**
  Retrieves the first \( n \) (where \( n \) defaults to 1) records in the current set with the specified sort key value.

- **NEXT record-count**
  Retrieves the next \( n \) (where \( n \) defaults to 1) records in the current set with the specified sort key value.

⚠️ **Note:** You can retrieve records with duplicate sort keys by specifying the REPEAT WITH NEXT command without changing the preceding sort key value. If you request retrieval for a record with a different sort key value, use the GET FIRST form of the command.

- **ALL**
  Retrieves all records in the current set with the specified sort key value.
• **view-id**
The qualifying ID for the record or logical record name. Use `view-id` when you are signed on to multiple subschemas. `View-id` refers to the subschema where the record (or logical record) can be found.

• **member-record-name**
The member record type (required).
  - *(field-list)* -- The fields within `record-name`. Enclose `field-list` in parentheses.

• **WHERE**
Specifies criteria used in selecting records.
  - `SORTKEY= -- The sort key used for record retrieval.`
  - `sort-field-name= -- The sort key field used for record retrieval.`

  **Note:** If the sortkey is defined as more than one field, the `SORTKEY` keyword should be used rather than a list of the individual field names.

  - `sort-key-value` -- The sort key value of the member record retrieved. In the event of duplicate sort keys, the order of retrieval depends on the DUPLICATES specification for the set type.

If the sort key is a group item, all subordinate fields must be specified.
  - `record` -- The qualifying record name for a `field-name`. The expanded syntax can be found below.
  - `field-name` -- The field name containing the sort key value.
  - `SORTKEY= concatenated-key-value` -- A concatenated sort key value used for record retrieval.

Separate each of the partial key values with blanks or commas.

• **AND criteria**
Specifies criteria used in selecting record occurrences.

• **record**
The qualifying record name for a `field-name`.
  - `view-id` -- The qualifying ID for the record or logical record name. Use `view-id` when you are signed on to multiple subschemas. `View-id` refers to the subschema where the record (or logical record) can be found.
  - `record-name (occurrence)` -- The record in the path where the `field-name` occurs.
  - If the `field-name` occurs in more than one record, use `record-name`. 
- If the record occurs more than once in the same path, use occurrence. If a database record name is the same as a CA OLQ keyword, the name should be enclosed in quotation marks.

Considerations

If criteria is respecified for the REPEAT command, the original selection criteria clause (including the sort key value) is replaced and a REPEAT WITHIN SET command is executed without an implied SORTKEY clause. Therefore, if the selection criteria have been changed and sort key retrieval optimization is preferred, the sort key value must always be used in a REPEAT command.

Examples

The following examples illustrate the use of the FIND/GET WITHIN SET using SORTKEY and associated REPEAT commands, based on the set occurrence diagram shown below:

FIND / GET WITHIN SET Using SORTKEY

Get Using Storage Key

A GET using storage key command is used to enter the database and establish currency for the SKILL record:

```
get first skill where calckey = 2080
```

```
SKILL
   SKILL-DBKEY : 0/5007116:1
   SKILL-ID-0455 : 2080
   SKILL-NAME-0455 : 'RPGII    '
   SKILL-DESCRIPTION-0455 : '
END OF RECORD
```

Get Within Set

A GET WITHIN SET using SORTKEY command retrieves the first EXPERTISE record with a sort key value of 03:
get first expertise in skill-expertise set where sortkey = 03

Repeat With First

A REPEAT command retrieves the next EXPERTISE record for which the EXPERTISE-YEAR-0425 field value is greater than or equal to 70:

repeat with first expertise where expertise-year-0425 ge 70

Repeat

A REPEAT command specifies the 03 sort key again, to achieve sort-key optimization:

repeat with first expertise where sortkey = 03 and expertise-year-0425 ge 70

FUNCTION

FUNCTION permits you to invoke a control key value from a terminal that doesn't have control keys.

Syntax

```
FUNCTION control-key-value
```

Parameter

- **control-key-value**
  Specifies the control key that is invoked. Valid values are [PA1], [PA2], and 1 through 99 (corresponding to PF keys 1 through 99).
Note: When a control key that is not currently associated with a line command is invoked, CA OLQ responds as if the [Enter] key was pressed.

Example

In the following example, the function associated with the PF10 key is invoked from the command line:

```
function 10
```

HELP

HELP (SHOW) explains CA OLQ commands and provides information about the data accessed by the current subschema. HELP and SHOW are synonymous.

Access mode: Some HELP parameters are invalid when the access switch is set to IDMS.

This article describes the following information:

- Syntax (see page 98)
- Parameters (see page 99)
- Example (see page 100)

Syntax

```
HELP
SHOW

BUILTIN functions
  BIF

COMmands
  COMmand = command-name

DESTination
  DIREcory
    USEr = user-name

FUNctions
  PFKey
    FUNCTION = control-key-value

LOGical records
  LRS
    VIEW = view-id

MESSage
  message-id

OPTions

PATh

QFIles
  dictionary-specification

QFILE
  qfile-name

RECords
  VIEW
    view-id
```

Parameters

- **BUILTIN FUNCTIONS**
  Lists information about built-in functions:
  - `ALL` -- Lists all the built-in function commands.
  - `AGGREGATE` -- Lists all the aggregate functions.
    An aggregate function is one whose argument includes one or more columns and operates on one or more values in each column. A few examples are `SUM`, `AVERAGE`, and `COUNT`.
  - `NONAGGREGATE` -- Lists all the nonaggregate functions.
    A nonaggregate function is one whose argument includes a single value within a column and only operates that value. A few examples are `LOG10`, `MODULO`, and `ABSOLUTE-VALUE`.
  - `builtin-function-name` -- Lists information about a particular built-in function.

- **COMMANDS**
  Lists the syntax and definitions of all CA OLQ commands.

  - `COMMAND= command-name`  
    Lists the syntax for the specified CA OLQ command.
HELP SIGNON lists syntax for the SIGNON command.

- **DESTINATION**
  Lists printer classes and destinations defined for the CA IDMS/DC or CA IDMS UCF system under which CA OLQ is executing. These printer classes and destinations can be specified in the PRINT command.

- **DIRECTORY USER= user-name**
  Lists all saved CA OLQ report files for a specified user. USER= user-name is an optional qualifier; user-name defaults to the current user. You must have a browse passkey for the specified user in order to see the directory.

- **FUNCTIONS**
  Lists CA OLQ commands associated with control keys for command mode.

- **FUNCTION= control-key-value**
  Displays the CA OLQ command associated with the named control key.

- **LOGICAL RECORDS**
  Displays the following information:
  
  **Access mode**: This parameter is **invalid** when the access switch is set to **IDMS**.
  
  - All logical records defined in the current subschema associated with an OBTAIN PATH command
  - Any keywords associated with each logical record's OBTAIN path
  - Any comments associated with each logical record
  - **VIEW=view-id** -- The qualifying ID for the record or logical record name. Use view-id when you are signed on to multiple subschemas. View-id refers to the subschema where the record (or logical record) can be found.

- **MESSAGE= message-id**
  Lists detailed information about the CA OLQ error message identified by message-id.

- **OPTIONS**
  Lists all session options, indicating the current setting and the alternate setting for each option.

- **PATH**
  Displays the sequence of CA OLQ retrieval commands for the current path definition, including the WHERE criteria.
  
  **Access mode**: This parameter is **invalid** when the access switch is set to **IDMS**.

- **QFILES**
  In secure installations, lists all qfiles in the data dictionary associated with the signed on user. In installations that do not use CA OLQ security features, lists all qfiles defined in the data dictionary.
- **dictionary-specification** -- Specifies the dictionary and dictionary node containing the named qfile.

- **QFILE= qfile-name**
  Displays the CA OLQ commands that make up the named qfile.
  - **(version)** -- Specifies the version number of the named qfile.
  - **dictionary-specification** -- Specifies the dictionary and dictionary node containing the named qfile.

- **RECORDS**
  Lists all database record types and set relationships defined in the current subschema.
  **Access mode:** This parameter is **invalid** when the access switch is set to **IDMS**.
  - **VIEW=view-id** -- The qualifying ID for the record or logical record name. Use **view-id** when you are signed on to multiple subschemas. **View-id** refers to the subschema where the record (or logical record) can be found.

- **RECORD=**
  Lists subschema fields and their usage.
  **Access mode:** This parameter is **invalid** when the access switch is set to **IDMS**.
  - **record-name** -- Specifies the data record type.
  - **logical-record-name** -- Specifies the logical record type.
    - **VIEW=view-id** -- The qualifying ID for the record or logical record name. Use **view-id** when you are signed on to multiple subschemas. **View-id** refers to the subschema where the record (or logical record) can be found.

- **REPORT= report-name**
  Displays a detailed description of the named report. When not specified, **report-name** defaults to the name of the current report.
  - **USER=user-name** -- Optional qualifier that allows a user to view report files saved by other users. This parameter is effective only if the user has been assigned the passkeys necessary for accessing other user's reports.

- **SUBSCHEMAS**
  Lists the subschemas available to the current user.
  - **DICTNAME= dictionary-name** -- Specifies the dictionary containing the specified subschemas. **DICTNAME** can be used to list information saved in other dictionaries without requiring you to sign on to them.
  - **DICTNODE= dictionary-node** -- Specifies the dictionary node. **DICTNODE** can be used to list information saved in other dictionaries without requiring you to sign on to them.
  - **dictionary-specification** -- Specifies the dictionary and dictionary node containing the named qfile.
TABLES
Lists the tables saved by the current or named user.
When the access switch is set to idms, CA OLQ only lists SQL schema entries to which the user has access.

Note: For more information about access switch, see, SET (see page 142) later in this chapter.

table-options
Explicitly identifies where the list of tables originates:

- DICTNAME/CATALOG=dictionary-name -- Specifies the dictionary under which the tables are saved.
- LOCATION=dictionary-node -- Specifies the dictionary node name. Access mode: This parameter is invalid when the access switch is set to IDMS.
- OWNER=user-name -- Allows you to view the ASF tables of a specific user. To view another user's tables, you must have BROWSE passkey authorization for the user's directory. Access mode: This parameter is invalid when the access switch is set to IDMS.

Note: For further information on passkeys, see &U$ICMSAD.

- SCHEMA=schema-name -- Allows you to view the SQL tables and views associated with the schema-name. Access mode: This parameter is invalid when the access switch is set to OLQ.

Examples:

HELP DESTINATION

The following HELP DESTINATION command lists available printer classes and destinations:

```
help destination
PRINT CLASSES/DESTINATIONS PAGE 1.1
LINE 1
(LTERM) (CLASS/DESTINATION)
PRT7026 CLASS=06
PRT7025 CLASS=63
PRT7024 CLASS=01
PRT7023 CLASS=06
PRT7022 CLASS=60
PRT7021 CLASS=02
PRT7020 CLASS=23
PRT7019 CLASS=25
PRT7018 CLASS=54
PRT7017 CLASS=53
PRT7016 CLASS=52
PRT7015 CLASS=51
PRT7014 CLASS=11
PRT7013 CLASS=02
PRT7012 CLASS=02
PRT7011 CLASS=02,04
PRT7010 CLASS=24
```
HELP QFILES

In the example below, a HELP QFILES command displays the qfiles available to the user currently signed on:

help qfiles

AVAILABLE QFILES PAGE 1.1
LINE 1
(QFILE NAME)

DICTIONARY NAME *DEFAULT*
DICTIONARY NODE *DEFAULT*

SRKITEM(1)  CNUMORD(1)  CNAMORD(1)  CUSTITEM(1)  ORDITEM(1)
EMP-REPT-01(100)  EMP-REPT(100)  EMP-REPT(1)  EMP-PROF(1)  DENTAL(1)  EMP-SKILL(1)

HELP RECORDS

The following HELP RECORDS command displays all database record types and set relationships defined in the current subschema:

help records

RECORDS IN EMPSS01 PAGE 1.1
LINE 1
(RECORD NAME)  (OWNS SETS)  (MEMBER OF SETS)

STRUCTURE  MANAGES  REPORTS-TO
VIA MANAGES

SKILL  SKILL-EXPERTISE  SKILL-NAME-NDX
CALC: SKILL-ID-0455

OFFICE  OFFICE-EMPLOYEE
CALC: OFFICE-CODE-0450

NON-HOSP-CLAIM  COVERAGE-CLAIMS
VIA COVERAGE-CLAIMS

JOB  JOB-EMPOSITION  JOB-TITLE-NDX
CALC: JOB-ID-0440

INSURANCE-PLAN
CALC: INS-PLAN-CODE-0435
PF 88

RECORDS IN EMPSS01 PAGE 2.1
LINE 18
(RECORD NAME)  (OWN SETS)  (MEMBER OF SETS)

HOSPITAL-CLAIM  COVERAGE-CLAIMS
VIA COVERAGE-CLAIMS

EXPERTISE  EMP-EXPERTISE
VIA EMP-EXPERTISE  SKILL-EXPERTISE

EMPOSITION  EMP-EMPOSITION
VIA EMP-EMPOSITION  JOB-EMPOSITION
HELP RECORD=EMPLOYEE

The following HELP RECORD=EMPLOYEE command displays the subschema fields and descriptions for the EMPLOYEE database record:

```
help record=employee
```

HELP SUBSCHEMAS

The following HELP SUBSCHEMAS command lists the subschemas available to the current user:

```
help subschemas dictname=docanwk
```

MENU

MENU allows you to switch from CA OLQ command mode to the menu facility.

This article describes the following information:

- Syntax (see page 105)
- Parameters (see page 105)
Syntax

Parameters

- **BATCH**
  The Batch Processing screen

- **BREAK**
  The Report Format - Sort screen

- **HEADER/COLUMN HEADER**
  The Report Format - Header screen

- **COLUMNS SELECT/FIELDS SELECT**
  The Column Select screen

- **TABLE**
  The Table Processing screen

- **DISPLAY/REPORT**
  The Display Report screen

- **EDIT**
  The Report Format - Edit screen

- **GROUP BY/SUMMARY**
  The Report Format - Group By screen
Example

The command MENU brings you to the Menu screen:

<table>
<thead>
<tr>
<th>Select Option</th>
<th>Pfkey</th>
<th>Description</th>
<th>Command/Screen Name</th>
<th>Show Help</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>---&gt; Data Source for Report</td>
<td>TABle</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>Choose tables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_</td>
<td></td>
<td>Choose subschema</td>
<td>SUBschema</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>---&gt; Retrieval Activity</td>
<td>RECord</td>
<td></td>
</tr>
<tr>
<td>_</td>
<td></td>
<td>Choose records from selected subschema</td>
<td>COLumn</td>
<td></td>
</tr>
<tr>
<td>_</td>
<td></td>
<td>Choose columns for report</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The command MENU OPTIONS brings you to the Options screen:

```
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RETrieve</td>
<td>Retrieve data to build report</td>
</tr>
<tr>
<td>LINkage</td>
<td>Alter database access strategy</td>
</tr>
<tr>
<td>QFiLe</td>
<td>Execute or create a predefined routine</td>
</tr>
<tr>
<td>SAVe</td>
<td>View existing or save current report</td>
</tr>
<tr>
<td>BATch</td>
<td>Submit batch report request</td>
</tr>
</tbody>
</table>
```

Options:

```
The OPTIONS command sets default options for a CA OLQ session.

This article describes the following information:

- Syntax (see page 107)
- Parameters (see page 108)
- Examples (see page 109)
```

Syntax:

```
<table>
<thead>
<tr>
<th>Options</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Options</td>
<td>ALL</td>
</tr>
<tr>
<td>NONE</td>
<td>Options</td>
<td>NONE</td>
</tr>
<tr>
<td>CODetable</td>
<td></td>
<td>CODetable</td>
</tr>
<tr>
<td>NOCODetable</td>
<td></td>
<td>NOCODetable</td>
</tr>
<tr>
<td>COMments</td>
<td></td>
<td>COMments</td>
</tr>
<tr>
<td>NOComments</td>
<td></td>
<td>NOComments</td>
</tr>
<tr>
<td>DBKey</td>
<td></td>
<td>DBKey</td>
</tr>
<tr>
<td>NOOdbkey</td>
<td></td>
<td>NOOdbkey</td>
</tr>
<tr>
<td>ECHO</td>
<td></td>
<td>ECHO</td>
</tr>
<tr>
<td>NOEcho</td>
<td></td>
<td>NOEcho</td>
</tr>
</tbody>
</table>
```
Parameters

- **ALL/NONE**
  Specifies whether the default internal field list contains all or none of a record's fields. The signon default for ALL/NONE remains in effect for all records retrieved in a session until changed by a user-specified FIELDS FOR command or by a user-specified field list in a FIND/GET command. Although the default value of ALL is usually taken, it is recommended that, for lengthy records, the internal field list be limited to only those fields the user requires. The field list can be restricted by specifying OPTIONS=NONE followed by either a FIELDS FOR command or a field-list parameter for each record retrieved.

- **CODETABLE/NOCODETABLE**
  Specifies whether code tables are used to translate internal codes in one or more report columns into a screen display format.
  OPTIONS=CODETABLE requests that code tables be applied to report columns. If a stand-alone table is specified with the EDIT command, CA OLQ uses that table to format the named report column. Otherwise, if a built-in table exists for the field in the data dictionary, this table is used for formatting.
  OPTIONS=NOCODETABLE requests that no code table formatting be applied to report fields. When you issue HELP REPORT, CA OLQ lists all code tables currently associated with report columns.

- **COMMENTS/NOCOMMENTS**
  Specifies whether comments are included in the HELP report files built by the HELP RECORDS, HELP SUBSCHEMAS, and HELP QFILES commands.

- **DBKEY/NODBKEY**
  Specifies whether dbkey columns are included in report files built by multiple record retrievals. Database keys qualified by page group are only included in a report file if the DBKEY option is in effect when the report file is initially created.

⚠️ **Note:** There is a significant storage overhead associated with retrieving a large number of records with the DBKEY option on.
- **ECHO/NOECHO**
  Specifies whether the user-entered command is repeated by CA OLQ on the 3270-type output device.

- **FILLER/NOFILLER**
  Specifies whether filler field values will be available to the user. If you specify `OPTIONS=NOFILLER` (default), filler fields aren't displayed and don't have to be specified along with other fields in selection criteria. If you specify `OPTIONS=FILLER`, filler fields are displayed and have to be specified in selection criteria for a group. Filler fields can only be accessed when `OPTIONS=FILLER` is in effect. Once a record is retrieved, changing the FILLER option has no effect on the internal field list.

- **FULL/SPARSE**
  Specifies how the format of displayed path retrieval report lines. FULL displays data associated with a record type once for each retrieved occurrence. SPARSE, used with a SELECT command, displays only the first of a repeating data value; SPARSE, used with a path command, displays only the first of a repeating record type.

- **HEADER/NOHEADER**
  Specifies whether the report built by multiple record retrievals will be displayed with a header line.

- **INTERRUPT/NOINTERRUPT**
  Sets or disables the processing interrupt feature for multiple record retrievals. INTERRUPT limits the number of retrievals that can occur without intervention. The limit is established during system generation. When interrupt occurs, you can use the OPTIONS command to change processing control before issuing a YES response to the question of Continue processing?

## Examples

If `OPTIONS=NOINTERRUPT` is specified before a YES response, all remaining records are retrieved regardless of the interrupt level.

When `OPTIONS=NOINTERRUPT` is specified, the entire retrieval is executed with no interruptions. We recommend, however, that the default value of INTERRUPT be chosen to permit interruption of processing at known intervals. Users who regularly execute extensive retrievals and who do not wish to monitor CA OLQ for the duration of command execution may elect to enlarge the interrupt count at installation time or change it using the SET INTERRUPT COUNT command.

⚠️ **Note:** Authorization for a user to select the NOINTERRUPT option is enabled through the data dictionary ADD USER DDDL statement. For more information about ADD USER DDDL statement and on CA OLQ security, see Assigning Authority to Access CA OLQ (see page 250).
• **OLQHEADER/NOOLQHEADER**

Specifies whether field names or user-specified CA OLQ headers will be used as headers for displayed data.

If you specify `OPTIONS=OLQHEADER`, field names are replaced with associated CA OLQ headers if any are defined in the data dictionary or if any are defined by the user. If you specify `OPTIONS=NOOLQHEADER` (the default), field names are used as report headers.

• **PATHSTATUS/NOPATHSTATUS**

Specifies the conditions for logical record retrieval. Path statuses are issued during execution of logical record paths. Path statuses are 1 to 32-character strings. Path statuses can be either standard or defined by the DBA in the subschema. The three standard path status values are:

- **LR-FOUND** -- Returned when the logical record request has been successfully executed
- **LR-NOT-FOUND** -- Returned when the specified logical record cannot be found, either because no such record exists or because all such occurrences have already been retrieved
- **LR-ERROR** -- Returned when an error occurs in the processing of the logical record path

When `OPTIONS=NOPATHSTATUS` is in effect, CA OLQ retrieves a logical record only when the path status is LR-FOUND. If a DBA-defined path status is returned, CA OLQ treats it as if LR-ERROR were the path status. No report file is generated and CA OLQ displays the following message:

```
OLQ 095017 08 ERROR OCCURRED WHILE ATTEMPTING TO RETRIEVE A LOGICAL RECORD. PATH STATUS=dba-defined-path-status
```

The path status can be displayed or printed; the field name is PATH STATUS. PATH STATUS is automatically added to the internal field list whenever `OPTIONS=PATHSTATUS` is specified.

**Note:** To refer to the PATH STATUS field, enclose it in quotation marks ('PATH STATUS').

For further information on the status of logical record paths, see *CA IDMS Logical Record Facility Guide*.

• **PICTURE/NOPICTURE**

Specifies whether external pictures or default pictures are used to edit report fields.

- `OPTIONS=PICTURE` requests that external pictures be used to format report fields. If an external picture has been constructed with the EDIT command, this picture edits the named report field. Otherwise, if an external picture exists for the field in the data dictionary, the stored picture is used for formatting. When `OPTIONS=PICTURE` is in effect, user-specified pictures override external pictures defined for fields in the data dictionary.
- `OPTIONS=NOPICTURE` requests that default pictures be used to edit report fields. Default pictures are derived from the internal pictures defined for fields in the data dictionary. When a HELP REPORT command is issued, external pictures for report fields are listed.

• **TERSE/VERBOSE**

Specifies the amount of identifying information provided when ON BREAK computations are requested.

When multiple computations are requested for a single output command, it is often difficult to determine which breaks and computations apply to which fields. `VERBOSE` provides complete field and record names for all breaks and computations. `TERSE` allows more information to be displayed on the terminal screen but this information may be more difficult to read.
WHOLE/PARTIAL
Specifies the content of displayed path retrieval report lines.

If OPTIONS=WHOLE is in effect, only those rows containing a retrieved occurrence for every record type specified in the path definition are displayed.

If OPTIONS=PARTIAL is in effect, all lines retrieved are displayed even if the line doesn’t contain each of the records specified in the path.

SYNONYM/NOSYNONYM
Specifies that synonyms for record names are (or aren’t) defined in the data dictionary and can (or can’t) be used in place of record names in CA OLQ. This option only applies to the record and column screens of menu mode CA OLQ.

Examples:

Options = Nocodetable

When OPTIONS=NOCODETABLE is in effect, stored values are displayed for the START-MONTH-0415 field:

```
options = nocodetable!
select emp-last-name-0415 start-year-0415 start-month-0415
from employee ! edit start-month-0415 codetable montab
```

```
  EMPLOYEE REPORT
  09/21/99
EMP-LAST-NAME-0415  START-YEAR-0415  START-MONTH-0415
LINGER             77               12
TERNER             82               5
LINGER             78               5
PENMAN             77               9
LINGER             78               1
LITERATA           80               9
WILCO              79               11
HEAROWITZ          81               9
TYRO               80               12
KAHALLY            79               9
PAPAZEUS           78               9
ARM                77               12
KING               80               8
CLOUD              77               3

 - 1 -
```

Options = Codetable

When OPTIONS=CODETABLE is in effect, a decoded value is substituted for each encoded value found for the START-MONTH-0415 field in the report file:

```
options = codetable!
select emp-last-name-0415 start-year-0415 start-month-0415
from employee ! edit start-month-0415 codetable montab
```

```
  EMPLOYEE REPORT
  09/21/99
EMP-LAST-NAME-0415  START-YEAR-0415  START-MONTH-0415
LINGER             77               DECEMBER
TERNER             82               MAY
LINGER             78               MAY
PENMAN             77               SEPTEMBER
LINGER             78               JANUARY
LITERATA           80               SEPTEMBER
```
When OPTIONS=DBKEY is in effect, a database key column qualified by page group appears in any report file that is built by retrieval of more than one record:

```
opt dbkey!
get all sequential department ! display
```

When OPTIONS=NODBKEY is in effect, no column for database key values is displayed:

```
opt nodbkey! get all sequential department
```

When OPTIONS=FULL is in effect, all information associated with the record type is displayed:

```
options=full !
select dept-id-0410, emp-last-name-0415 from department, employee
where dept-employee ! display
```

---

<table>
<thead>
<tr>
<th>DEPT-ID-0410</th>
<th>DEPT-NAME-0410</th>
</tr>
</thead>
<tbody>
<tr>
<td>6666</td>
<td>EXECUTIVE ADMINISTRATION</td>
</tr>
<tr>
<td>2000</td>
<td>ACCOUNTING AND PAYROLL</td>
</tr>
<tr>
<td>1010</td>
<td>EXECUTIVE WEATHER MANAGEMENT</td>
</tr>
<tr>
<td>1011</td>
<td>WEATHER MANAGEMENT REPORTING</td>
</tr>
<tr>
<td>1000</td>
<td>PERSONNEL</td>
</tr>
<tr>
<td>3124</td>
<td>WEATHER REPORTING DIV.</td>
</tr>
<tr>
<td>3180</td>
<td>INTERNAL SOFTWARE</td>
</tr>
<tr>
<td>3125</td>
<td>ORDERING DEPARTMENT (3)</td>
</tr>
<tr>
<td>5380</td>
<td>BLUE SKIES</td>
</tr>
<tr>
<td>3290</td>
<td>COMPUTER OPERATIONS</td>
</tr>
<tr>
<td>3121</td>
<td>WEATHER INFORMATION SERVICES</td>
</tr>
<tr>
<td>4000</td>
<td>PUBLIC RELATIONS</td>
</tr>
<tr>
<td>5180</td>
<td>BRAINSTORMING</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>DEPT-ID-0410</th>
<th>EMP-LAST-NAME-0415</th>
</tr>
</thead>
<tbody>
<tr>
<td>6666</td>
<td>HENDON</td>
</tr>
<tr>
<td>6666</td>
<td>PAPAZEUS</td>
</tr>
<tr>
<td>6666</td>
<td>RUPEE</td>
</tr>
<tr>
<td>6666</td>
<td>WILDER</td>
</tr>
<tr>
<td>2000</td>
<td>BLOOMER</td>
</tr>
<tr>
<td>2000</td>
<td>HUTTON</td>
</tr>
<tr>
<td>2000</td>
<td>JENSON</td>
</tr>
<tr>
<td>2000</td>
<td>KIMBALL</td>
</tr>
<tr>
<td>2000</td>
<td>KING</td>
</tr>
<tr>
<td>2000</td>
<td>NICEMAN</td>
</tr>
<tr>
<td>1000</td>
<td>FITZUH</td>
</tr>
<tr>
<td>1000</td>
<td>HEDGEHOG</td>
</tr>
<tr>
<td>1000</td>
<td>JOHNSON</td>
</tr>
<tr>
<td>1000</td>
<td>ORGRAATZI</td>
</tr>
</tbody>
</table>

---
When OPTIONS=SPARSE is in effect, repeating column values are displayed only once:

```
opt=sparse !
select dept-id-0410, emp-last-name-0415 from department, employee
where dept-employee ! display

DEPARTMENT/EMPLOYEE REPORT
09/21/99

DEPT-ID-0410 EMP-LAST-NAME-0415

6666    HENDON
        PAPAZEUS
        RUPEE
        WILDER

2000    BLOOMER
        HUTTON
        JENSON
        KIMBALL
        KING
        NICEMAN

1000    FITZHUGH
        HEDGEHOG
        JOHNSON
        ORGRATZI

- 1 -
```

Options = Header

Output for OPTIONS=HEADER is shown below:

```
opt=header !
select dept-id-0410, emp-last-name-0415 from department, employee
where dept-employee ! display

DEPARTMENT/EMPLOYEE REPORT
09/21/99

DEPT-ID-0410 EMP-LAST-NAME-0415

6666    HENDON
        PAPAZEUS
        RUPEE
        WILDER

2000    BLOOMER
        HUTTON
        JENSON
        KIMBALL
        KING
        NICEMAN

1000    FITZHUGH
        HEDGEHOG
        JOHNSON
        ORGRATZI

- 1 -
```

Output for OPTIONS=NOHEADER is shown below:

```
opt=noheader !
select dept-id-0410, emp-last-name-0415 from department, employee
where dept-employee ! display

6666    HENDON
        PAPAZEUS
        RUPEE
        WILDER

2000    BLOOMER
        HUTTON
        JENSON
        KIMBALL
        KING
```
Options = Interrupt

In the following example, the OPTIONS=INTERRUPT is in effect and the retrieval limit before interruption is 25 records. After selecting 25 records, CA OLQ interrupts with a CONTINUE prompt. A YES or RESUME response directs CA OLQ to continue retrieving data until either the interrupt limit is reached again or processing is completed:

```
options= interrupt !
set interrupt=25
```

```
OLQ 092010 00 The interrupt count has been modified.
select dept-id-0410, emp-last-name-0415 from department, employee
OLQ 098006 00 17 whole lines and 0 partial lines in report.
OLQ 098007 00 25 records read. 21 records selected.
OLQ 098008 00 17 of 47 primary record pages read.
OLQ 098009 00 Continue (yes/no)?
```

When OPTIONS=NOINTERRUPT is in effect, CA OLQ retrieves all records without further interruption:

```
options=nointerrupt !
select dept-id-0410, emp-id-0415, emp-zip-0415 from department, employee
```

```
OLQ 098006 00 57 whole lines and 0 partial lines in report.
OLQ 098007 00 75 records read. 66 records selected.
```

display cols 1,2,3

```
DEPARTMENT/EMPLOYEE REPORT
10/08/99
```

```
DEPT-ID-0410 | EMP-ID-0415 | EMP-ZIP-FIRST-FIVE-0415
-------------|-------------|--------------------------
6666         | 30          | 02198                    
6666         | 471         | 03256                    
6666         | 1           | 021212                   
6666         | 472         | 031145                   
2008         | 69          | 01675                    
2008         | 100         | 02176                    
2008         | 11          | 02176                    
2008         | 67          | 01239                    
2008         | 106         | 02176                    
2008         | 101         | 02176                    
1000         | 81          | 03458                    
1000         | 8683        | 10996                    
1000         | 51          | 02546                    
1000         | 91          | 06182                    
```

Options = No Olqheader

When OPTIONS=NOOLQHEADER is in effect, field names appear as report headers:

```
DEPARTMENT/EMPLOYEE REPORT
09/21/99
```

```
DEPT-ID-0410 | EMP-LAST-NAME-0415
-------------|-------------------
When OPTIONS=OLQHEADER is in effect, predefined CA OLQ headers are used:

<table>
<thead>
<tr>
<th>DEPT NUMBER</th>
<th>EMPLOYEE NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>6666</td>
<td>HENDON</td>
</tr>
<tr>
<td>6666</td>
<td>PAPAZEUS</td>
</tr>
<tr>
<td>6666</td>
<td>RUPEE</td>
</tr>
<tr>
<td>6666</td>
<td>WILDER</td>
</tr>
<tr>
<td>2000</td>
<td>BLOOMER</td>
</tr>
<tr>
<td>2000</td>
<td>HUTTON</td>
</tr>
<tr>
<td>2000</td>
<td>JENSON</td>
</tr>
<tr>
<td>2000</td>
<td>KIMBALL</td>
</tr>
<tr>
<td>2000</td>
<td>KING</td>
</tr>
<tr>
<td>2000</td>
<td>NICEMAN</td>
</tr>
<tr>
<td>1000</td>
<td>FITZHUGH</td>
</tr>
<tr>
<td>1000</td>
<td>HEDGEHOG</td>
</tr>
<tr>
<td>1000</td>
<td>JOHNSON</td>
</tr>
<tr>
<td>1000</td>
<td>ORGRATZI</td>
</tr>
</tbody>
</table>

Options = Picture

When OPTIONS=NOPICTURE is in effect, default pictures are used to format report fields:

<table>
<thead>
<tr>
<th>DEPT-ID-0410</th>
<th>EMP-LAST-NAME-0415</th>
</tr>
</thead>
<tbody>
<tr>
<td>6666</td>
<td>HENDON</td>
</tr>
<tr>
<td>6666</td>
<td>PAPAZEUS</td>
</tr>
<tr>
<td>6666</td>
<td>RUPEE</td>
</tr>
<tr>
<td>6666</td>
<td>WILDER</td>
</tr>
<tr>
<td>2000</td>
<td>BLOOMER</td>
</tr>
<tr>
<td>2000</td>
<td>HUTTON</td>
</tr>
<tr>
<td>2000</td>
<td>JENSON</td>
</tr>
<tr>
<td>2000</td>
<td>KIMBALL</td>
</tr>
<tr>
<td>2000</td>
<td>KING</td>
</tr>
<tr>
<td>2000</td>
<td>NICEMAN</td>
</tr>
<tr>
<td>1000</td>
<td>FITZHUGH</td>
</tr>
<tr>
<td>1000</td>
<td>HEDGEHOG</td>
</tr>
<tr>
<td>1000</td>
<td>JOHNSON</td>
</tr>
<tr>
<td>1000</td>
<td>ORGRATZI</td>
</tr>
</tbody>
</table>

When OPTIONS=PICTURE is in effect, external pictures are used to format report fields. In the following example, a dynamic external picture is specified with the EDIT command:

```
edit dept-id-0410 picture='9-999'
display
```
PAGE HEADER / FOOTER

PAGE HEADER/FOOTER enables you to locate a user-specified report page header or footer where you want it on the page or the terminal screen.

⚠️ Note: The page headers should not contain the column heading.

This article describes the following information:

- Syntax (see page 116)
- Parameters (see page 117)
- Example (see page 117)

**Syntax**

```
<<PAGE HEAder FOOTer>>
<<SkiP blank BEFore AFTER n>>
<<- LINE n header-string ALIGN RIGHT LEFT CENter>>
```
Parameters

- **PAGE HEADER**
  Specifies that you want to include a user-specified page header in the report.

- **PAGE FOOTER**
  Specifies that you want to include a user-specified page footer in the report.

- **SKIP**
  Specifies that a user-specified number of lines are to be skipped between the header/footer and the report.

- **BEFORE= n**
  Specifies that \( n \) lines are skipped before the report page header/footer is inserted. \( N \) must be greater than zero and less than 10.

- **AFTER= n**
  Specifies that \( n \) lines are skipped after the report page header/footer is inserted. \( N \) must be greater than zero and less than 10.

- **LINE \( n \)= header-string**
  Specifies the line number and the contents of the header/footer. For example, the following command specifies that the first line of the report page header reads 'DEPARTMENT REPORT' and the second line of the report page header reads the date the report was built:

  ```plaintext
  line 1 = 'DEPARTMENT REPORT'
  line 2 = '$DATE'
  ```

- **ALIGN RIGHT/LEFT/CENTER**
  Specifies that the report page header/footer line is aligned on the right, left, or centered on the page.

Example

This example demonstrates the commands necessary to include both a user-specified report page header and footer in a report:

```plaintext
page header skip before 1 skip after 2 line 1 = 'fiscal report'
line 2 = '$date' ! page footer skip before 2 skip after 1
line 1 = 'page $page' line 2 = 'CA, Inc. ! display

FISCAL REPORT
09/22/99

<table>
<thead>
<tr>
<th>EMP-ID-0415</th>
<th>EMP-LAST-NAME-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>9999</td>
<td>LINGER</td>
<td>38500.00</td>
</tr>
<tr>
<td>48</td>
<td>TERNER</td>
<td>13800.00</td>
</tr>
<tr>
<td>23</td>
<td>LINGER</td>
<td>42500.00</td>
</tr>
<tr>
<td>23</td>
<td>LINGER</td>
<td>38000.00</td>
</tr>
<tr>
<td>149</td>
<td>PENMAN</td>
<td>39000.00</td>
</tr>
<tr>
<td>15</td>
<td>LINGER</td>
<td>85000.00</td>
</tr>
<tr>
<td>15</td>
<td>LINGER</td>
<td>75000.00</td>
</tr>
<tr>
<td>35</td>
<td>LITERATA</td>
<td>37500.00</td>
</tr>
<tr>
<td>349</td>
<td>WILCO</td>
<td>00000.00</td>
</tr>
</tbody>
</table>
```
PRINT

PRINT directs a formatted CA OLQ report to a specific CA IDMS/DC printer for a hard copy.

**Batch considerations:** PRINT is invalid when running local mode.

This article describes the following information:

- Syntax (see page 118)
- Parameters (see page 119)
- Considerations (see page 121)
- Example (see page 122)

**Syntax**

```
PRINT
REPORT report-name USER user-name
COPY copy-number COPIES
AT DESTINATION destination-name CLASS class-number
FOR page-count ALL PAGES
SUMMARY DETAIL ENTIRE
CURRENT HELP
RIGHT LEFT
PAGE LINE
FIRST LAST NEXT PRIOR
* + page-line-count
```
Parameters

- **REPORT= report-name**
  Specifies the name of the report to be printed.

- **USER= user-name**
  Specifies the user whose report directory contains the named report file.

- **copy-number**
  Specifies the number of report copies printed. Copy-number is an integer in the range 1 through 255. The default value is 1.

- **DESTINATION destination-name**
  Specifies the printer destination the report is sent to. Destination-name can be any printer destination specified at CA IDMS/DC system generation. Use SHOW DESTINATION for a list of valid destinations.

- **CLASS class-number**
  Specifies the printer class the report file is sent to. Class-number-n is an integer in the range 1 through 64.

**Note:** If the DESTINATION or CLASS options are not used, the report is printed at the default printer defined at CA IDMS/DC system generation for the terminal in use.

- **page-count**
  Specifies the number of report pages printed. Page-count is an integer.

- **ALL**
  Specifies that all the report pages are printed.

- **SUMMARY**
  Specifies the printing of summary report lines only.

- **DETAIL**
  Specifies the printing of detail report lines only. If summary lines are present in the report file, they are not output.
- **ENTIRE**
  Specifies the printing of both detail and summary report lines.

- **CURRENT REPORT**
  Requests output of the last report displayed; a help report or a data report.

- **HELP REPORT**
  Requests output of the help report file built by the last HELP command.

  **Note:** For more information, see HELP (see page 98), earlier in this chapter.

- **RIGHT/LEFT**
  Specifies horizontal movement within the report file.

- **PAGE/LINE**
  Requests the data in the report file be output relative to a designated line or page of the report file:
  - PAGE requests the data in the report file be output beginning at the current or specified page number.
  - LINE requests a page of the report file be output beginning at the current or specified line number.
    The default is PAGE.

- **FIRST**
  Outputs a page of report file data, beginning at page 1, line 1.

- **LAST**
  Outputs the last page of report file data.

- **NEXT**
  Outputs a page of report file data, beginning at the page or line number immediately following the current page or line number.

- **PRIOR**
  Outputs a page of report file data, beginning at the page or line number immediately preceding the current page or line number.

- **Asterisk (**) --**
  Outputs the current page of report file data, beginning at the first line of that page.

- **Asterisk (**) --**
  Outputs a page of report file data beginning $n$ pages or lines:
  - Minus sign (-) -- *before* the current page or line number.
  - Plus sign (+) -- *after* the current page or line number.
  - Asterisk (*) -- is a required character that explicitly references the current page or line.
- **page** -- The number of pages
- **line-count** -- The number of lines

- **page/line-count**
  Specifies the starting point of the output relative to the current page and line number. `Page/line-count` outputs a page of report file data, beginning at the specified page or line number.

- **SPREAD EVEN/MAXIMUM/LEFT nn**
  Specifies the space between the columns.
  - **EVEN** -- The same number of spaces between each column (Space the columns evenly).
  - **MAXIMUM** -- The maximum number of spaces between each column.
  - **LEFT** -- Displays columns starting in the left most position with one space separating each column.
  - **nn** -- `nn` spaces between each column. Zero is not a valid number. The minimum number of spaces allowed is one.

- **COLUMNS**
  Specifies the columns included in the output and, optionally, the order of appearance and width of those columns. Column specifications remain in effect until altered by a subsequent PRINT command. Column specification is:
  - **MAXIMUM** -- Output of as many sequential columns, starting with column 1, as can appear on one page of the report. Excess columns are ignored, and no warning message is produced.
  - **field-reference** -- Output of specific columns and the number of characters in each column.

- **FIXED COLUMNS**
  Specifies the columns, and their order, to remain on the screen when paging left and right. The columns specified with this parameter precede the columns specified in the `COLUMNS=` parameter. They remain fixed on the left side of the report. Column specifications remain in effect until altered by a subsequent PRINT command or a `FIXED COLUMNS = NONE` command.
  - **NONE** -- No report fields are fixed in the report. This cleans out the fixed columns list.
  - **field-reference** -- The columns and the number of characters in each column to be output.

### Considerations

Report files sometimes contain information that cannot be displayed. CA OLQ indicates data that cannot be displayed, as follows:

- The at sign (@) indicates an unprintable character. CA OLQ provides a translation function that handles all characters written to a terminal or to the print queue. If you want to view the characters represented, you can use `EDIT HEXADECIMAL` to display the character in its hexadecimal representation.
Note: For an explanation of how to modify the CA OLQ translation table, see the CA IDMS installation guide for your operating system.

- The asterisk (*) indicates invalid data. The invalid data flag appears when data is not stored in the defined format or when a COMPUTE command yields invalid results (as with decimal overflows and division by zero).

Null character considerations

The null character is by default a period (.). You can override this by invoking the SET NULL command.

Note: For more information about the SET command see, SET (see page 142), later in this chapter.

Data retrieved in SQL tables can contain null values. To display them, CA OLQ pads the entire length of the display field with the null character.

Example

This example demonstrates the commands necessary to print 2 copies of the employee table:

print dest=la copies=2 columns=maximum

QFILE (OLQ Command)

QFILE accesses CA OLQ command sequences stored in the data dictionary. When you specify QFILE qfile-name, all CA OLQ commands contained within the named qfile are automatically executed.

Note: For more information about qfiles, see Using qfiles (see page 229).

Syntax

QFILE qfile-name (version)

DICtname = dictionary-name

DICTNode = dictionary-node-name
Parameters

- **qfile-name**
  The name of the qfile to be executed. The named qfile must be saved in the data dictionary. CA OLQ interprets any nonreserved first word in a command as a qfile name.

- **(version)**
  Identifies the version number of the named qfile. The default is the highest version defined for the named qfile. If used, version must be enclosed in parentheses.

- **DICTNAME= dictionary-name**
  Identifies the dictionary where the named qfile is stored.

- **DICTNODE= dictionary-node-name**
  Identifies the dictionary node controlling the named dictionary.

- **parameter-value**
  Specifies one or more values to be substituted into variable parameters in the qfile at execution time.

Example

**Show Qfile**

SHOW QFILE displays the REPORT qfile definition, which was previously saved:

```
show qfile report
REPORT
DICTIONARY NAME TSTDICT
DICTIONARY NODE *DEFAULT*
SET DICTNAME TSTDICT
SIGNON SS EMPSS01 SCHEMA EMPSCHM ( 100 )
OPTIONS ALL HEADER ECHO NOFILLER FULL WHOLE INTERRUPT NOOLQHEADER -
NOPATHSTAT NOSTAT COMMENT VERBOS NODEKEY NOPICTURE NOCODETAB NOSYN
SET ACCESS OLQ
SELECT EMP-LAST-NAME-0415 DEPT-NAME-0410 SALARY-AMOUNT-0420 -
FROM EMPLOYEE, DEPARTMENT, EMPOSITION WHERE DEPT-EMPLOYEE AND -
EMP-EMPOSITION
PAGE HEADER BLANK LINES AFTER 1 -
LINE 1 'DEPARTMENT/EMPLOYEE/EMPOSITION REPORT' CENTER -
LINE 2 'DATE' CENTER
PAGE FOOTER BLANK LINES BEFORE 1 -
LINE 1 '-PAGE-' CENTER
EDIT EMP-LAST-NAME-0415 -
   ALIGN LEFT -
   OLQHEADER 'THIS IS NAME HEADER'
DISPLAY SPREAD EVEN COLUMNS = EMP-LAST-NAME-0415 DEPT-NAME-0410 -
   SALARY-AMOUNT-0420
END OF REPORT
```

Qfile Report
The above qfile is invoked by specifying the REPORT qfile name. The QFILE REPORT command builds a list of employee names, their departments and their salaries:

<table>
<thead>
<tr>
<th>DEPT-ID-0410</th>
<th>EMP-LAST-NAME-0415</th>
<th>SALARY-AMOUNT-0420</th>
</tr>
</thead>
<tbody>
<tr>
<td>6666</td>
<td>HENDON</td>
<td>240000.00</td>
</tr>
<tr>
<td>6666</td>
<td>PAPAZEUS</td>
<td>100000.00</td>
</tr>
<tr>
<td>6666</td>
<td>MUNYOUN</td>
<td>90000.00</td>
</tr>
<tr>
<td>6666</td>
<td>RUMPLEST</td>
<td>80000.00</td>
</tr>
<tr>
<td>6666</td>
<td>RUPEE</td>
<td>76000.00</td>
</tr>
<tr>
<td>6666</td>
<td>WILDER</td>
<td>90000.00</td>
</tr>
<tr>
<td>2000</td>
<td>BLOOMER</td>
<td>15000.00</td>
</tr>
<tr>
<td>2000</td>
<td>HUTTON</td>
<td>44000.00</td>
</tr>
<tr>
<td>2000</td>
<td>JENSON</td>
<td>82000.00</td>
</tr>
<tr>
<td>2000</td>
<td>KIMBALL</td>
<td>45000.00</td>
</tr>
<tr>
<td>2000</td>
<td>KING</td>
<td>14500.00</td>
</tr>
<tr>
<td>2000</td>
<td>NICEMAN</td>
<td>14000.00</td>
</tr>
</tbody>
</table>

SAVE QFILE

SAVE QFILE stores the retrieval and report formatting commands necessary to recreate the current report. The qfile is saved into the data dictionary.

Batch considerations: SAVE QFILE is invalid when running local mode.

Syntax

```
```

Parameters

- **qfile-name**
  The name of the qfile to be stored in the data dictionary.

- **(version)**
  The version number of the named qfile enclosed in parentheses. **Version** defaults to 1.
**Example**

*Save Qfile*

The following qfile retrieves, formats, and then displays data:

```
options = sparse
select dept-id-0410, emp-id-0415, emp-name-0415, salary-amount-0420 -
from department, employee, emposition where dept-employee and emp-empositioncompute
&xq.average salary' = avg(salary-amount-0420) - group by dept-id-0410display
```

*Save Qfile*

The SAVE QFILE command is used to save the path listed above as the EMP-SAL qfile:

```
save qfile emp-sal
```

109017  THE REQUESTED OPERATION FOR EMP-JOB(1) HAS SUCCESSFULLY COMPLETED

When the EMP-SAL qfile is executed, CA OLQ retrieves the ids, names, and salaries for all company employees.

**SAVE REPORT**

SAVE REPORT stores a report (definition and data) so that it can be viewed at a later date.

⚠️ **Note:** For the security associated with saved reports, see Security for ASF tables (see page 254).

**Batch considerations:** SAVE REPORT is invalid when running local mode.
Syntax

```
SAVE REPORT report-name
USER user-name
RETENTION PERIOD days
```

Parameters

- **report-name**
  The 1 to 32-character name of the report to be saved.

- **USER= user-name**
  Specifies the user into whose directory the report is saved. If `user-name` is not specified, the report is saved under the current user name.

- **RETENTION PERIOD= days**
  Specifies the number of days for which the report is saved.
  A default retention period and a maximum retention period are established at DC/UCF system generation. Reports are automatically deleted at the end of their associated retention periods.

Example

The report file used for this example was initially created by execution of the following SELECT statement:

```sql
select office-code-0450, office-phone-0450(1), emp-id-0415
from office, employee
where office-employee
```

The following SAVE REPORT command stores the report, created by the SELECT statement shown above, in the data dictionary and associates the report with the name OFFICE#8. After the OFFICE#8 report has been stored, CA OLQ displays a message to indicate that processing has been completed successfully:

```
save report=office#8 retention period=10 days
OLQ 107001 00  The OFFICE#8 report has been saved successfully
```

This report can then be viewed at a later time using DISPLAY:

```
display report = office#8
```
SELECT -- OLQ access mode

The SELECT statement in OLQ access mode retrieves data for display. A single retrieval request can combine data from multiple ASF tables, database records, logical records, subschemas, and sequential files. Sequential files are used for batch retrieval only.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

This article describes the following information:
- Syntax (see page 127)
- Parameters (see page 127)
- Examples (see page 130)

Syntax

```
SELECT [DISTINCT] 
  * | qualifying-name.
    column-expression | column-name [AS alt-column-heading]
  FROM table-name ...
  WHERE criteria
  GROUP BY group-by-expression [HAVING criteria]
  UNION [ALL] select-statement
  ORDER BY column-name [ASCending | DESCending] column-position
  OUTPUT file-name = ...
```

Parameters

- **DISTINCT**
  Eliminates all but one duplicate row occurrence from the report, based on specified columns.
- Asterisk (*) -- Lists all the columns of the named tables or records in the report. The order in which the columns are listed in the component tables or records is the order in which they appear, from left to right, in the report.

- `qualifying-name` -- A prefix denoting the table or record from which the column is being retrieved. `Qualifying-name` must be separated from its object by a period (.). The qualifying name can modify an asterisk (*), requesting all columns of the named table. `Qualifying-name` can be one of the following:

  - A stored table
  - A table derived from one or more stored tables or network records (a view)
  - A network database record
  - An alternative table name or record name
  - A group-level qualification of a record element

  You can specify more than one qualifying name for a column.

- `column-expression` -- The columns you want displayed in the report:

  - A table column or record column
  - A computed arithmetic expression, based on column values
  - An aggregate function, modifying a column expression
  - A built-in function, modifying a column expression
  - An occurrence of a multiply-occurring field

  - A fixed value, such as a literal string
    Fixed value expressions containing blanks or special characters must be enclosed in quotation marks.

- `column-name` -- The object of a qualifying table name. `Column-name` can be the name of a column in the object table or record, or a group-level qualification of a column.

- **AS alt-column-heading**
  Specifies an alternative column heading. You can use this to assign a column heading for any column reference, including computed fields or fields modified by a built-in or aggregate function. Commas separating column specifications are optional, but recommended.

- **FROM table-name**
  Represents the table from which data is retrieved. Commas separating multiple table and record entries are required.

- **FROM view-name**
  Specifies the name of the table derived from one or more stored data tables or records.
FROM view-id
Specifies a user-supplied name identifying a particular subschema. View-id is defined by the VIEW operand of the SIGNON statement. This is used to qualify record names.

FROM record-name
Specifies the name of a record.

alt-source-name
Specifies an alternative name for the table or record.

WHERE criteria
Represents criteria used by SELECT to retrieve records.

```
Note: Syntax for criteria used in the WHERE clause are expanded in SELECT Selection Criteria (see page 163).
```

GROUP BY group-by-expression
Groups rows into sets that contain like column values. Group-by-expression represents a column value. In each GROUP BY set, all rows contain the same value of the group-by-expression. The GROUP BY group-by-expression must be the same value as the column expression that is the object of the SELECT statement and can include aggregate and/or built-in functions.

HAVING criteria
Applies selection criteria to the result of the GROUP BY expression.

```
Note: Syntax and Purpose for criteria used in the HAVING clause are expanded in SELECT Selection Criteria (see page 163).
```

UNION ALL select-statement
Concatenates two or more SELECT statements containing like columns. Columns correspond positionally. Corresponding columns must have the same data type, picture, and decimal representation. UNION eliminates duplicate rows from the report.

- ALL -- Retains duplicate rows in the report.
- select-statement -- Concatenates two or more SELECT statements containing like columns.

ORDER BY
Sorts the rows of the report by the value of the column you specify. You can sort by:

- column-name -- The columns being sorted in the report. You can specify any type of column expression in terms of its column position. You cannot specify column-name when any of the following types of column expressions are the object of the sort:
  - Computed fields
  - A column expression containing an AS parameter
- The UNION operand
- Subscripted fields (arrays)
  - column-position -- The position in the column-list, from left to right, of the column to be sorted. Column-position can refer to computed columns or to columns modified by built-in or aggregate functions.

- ASCENDING/DESCENDING
  Specifies the order in which to sort the columns.

- OUTPUT= file-name
  Specifies that the report is saved as a sequential file. This option is only valid when running CA OLQ batch.

Examples

Select all columns

Instead of specifying all the columns of the table, or fields of the record that you want to retrieve, you can specify an asterisk (*) and receive all columns or fields:

select * from employee

The result report displays the first column of the EMPLOYEE table first, the second column of the EMPLOYEE table second, and so on.

Select column expression

You can specify the names of the columns or fields you want to retrieve explicitly:

select emp-name, dept-name from employee, department

The result report displays two columns, the employee name from the EMPLOYEE table, and the department name from the DEPARTMENT table.

Select Distinct

Distinct eliminates all but one duplicate row occurrence from the report, based on specific columns.

Example:

If there are two occurrences of the last name field BURR, you can eliminate the row containing the duplicate value:

select DISTINCT emp-last-name from employee ! display

EMP LAST NAME
----------
BURR
GOLD
ILTIS
LIGARE
WAKEFIELD
WONES
Select qualifying name

Qualifying-name is a prefix denoting the table or record from which the column is being retrieved. Qualifying-name must be separated from its object by a period (.). This is useful when two tables contain columns with the same name.

Example:

If both the EMPLOYEE and the DEPARTMENT tables contain DEPT-ID fields, you can qualify the column names:

```
select EMPLOYEE.dept-id, DEPARTMENT.dept-id
from employee, department
where employee.dept-id = department.dept-id
```

The qualifying name can also modify an asterisk (*).

You can also specify a group-level qualification as a qualifying name:

```
select EMP-NAME-0415.EMP-LAST-NAME-0415 from employee
```

Select two references

You can use the AS parameter to distinguish one of two references to the same column name:

```
select manage.emp-id-0415 AS MANAGER,
works-for.emp-id-0415 AS EMPLOYEE
from employee manages, employee works-for
```

Select and sort

A simple SELECT command retrieves the EMPLOYEE record specified with a column list and sorts the fields by last name:

```
select emp-id-0415, emp-name-0415 from employee order by emp-last-name-0415 descending
```

```
EMP-ID-0415  EMP-FIRST-NAME-0415  EMP-LAST-NAME-0415
0124        SUSAN            SPELLMAN
0528        EDWARD           MCCARTHY
0512        CHERYL           MAYOR
1042        SHARON           CIVITTOLO
0954        ANGELA           BELVAL
0320        JOSEPH           ANTHONY
```

Joining tables based on equal values

To retrieve hiring information on all employees in a department:

1. Specify a SELECT clause, listing the columns you want (EMP-LAST-NAME, START-YEAR, and DEPT-NAME).

2. Specify a FROM clause, naming the tables from which you are retrieving data (EMPLOYEE and DEPARTMENT).

3. Specify a WHERE clause, indicating join criteria linking the two tables. For example: the DEPT-ID from the EMPLOYEE table is equal to the DEPT-ID from the DEPARTMENT table.

This example lists employees, the year they started, and their department name:
select employee.emp-last-name, employee.start-year, department.dept-name from employee, department where employee.dept-id = department.dept-id

<table>
<thead>
<tr>
<th>EMP-LAST-NAME</th>
<th>START-YEAR</th>
<th>DEPT-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>WONES</td>
<td>79</td>
<td>DEVELOPMENT</td>
</tr>
<tr>
<td>WAKEFIELD</td>
<td>83</td>
<td>PERSONNEL</td>
</tr>
<tr>
<td>BURR</td>
<td>80</td>
<td>PERSONNEL</td>
</tr>
<tr>
<td>LIGARE</td>
<td>85</td>
<td>DEVELOPMENT</td>
</tr>
<tr>
<td>BURR</td>
<td>84</td>
<td>MARKETING</td>
</tr>
<tr>
<td>SCHLEY</td>
<td>80</td>
<td>PLANNING</td>
</tr>
<tr>
<td>ILTIS</td>
<td>81</td>
<td>PERSONNEL</td>
</tr>
<tr>
<td>GOLD</td>
<td>80</td>
<td>MARKETING</td>
</tr>
</tbody>
</table>

Joining two tables

Using a single SELECT statement, you can produce a report containing data from more than one table. The selection criteria in the WHERE clause provide column join and key information. The WHERE clause can contain other comparison expressions.

Example:

The following SELECT statement joins the EMPLOYEE and DEPARTMENT tables on like DEPT-ID values, and lists only those employees who started working before 1980:

```sql
select employee.*, department.*
from employee, department
where employee.dept-id = department.dept-id
and employee.start-year lt '80'
```

You do not have to display the fields on which you are joining.

Example:

The following SELECT statement joins the EMPLOYEE and DEPARTMENT stored tables, but lists only the employee ID numbers and the name:

```sql
select distinct employee.emp-id, department.dept-name
from employee, department
where employee.dept-id = department.dept-id
```

Reflexive joins

Reflexive joins combine two different rows of the same table. When you are joining a table with itself, it is useful to supply alternative table names to distinguish the two references to the column name.

This example lists employees and their managers. EMPLOYEE MANAGE and EMPLOYEE WORKS-FOR are alternative names for the same record:

```sql
select works-for.emp-last-name as 'worker', manage.emp-last-name as 'manager'
from employee works-for, employee manage
where works-for.manager-id = manage.emp-id
```
Joining tables and records residing in multiple subschemas

This example uses a SELECT statement to create a report containing data from an ASF-generated table and a network table:

- **For ASF-generated tables**, you do not have to sign on to any subschemas before issuing your SELECT request. You should name the dictionary in which the table is stored.

- **For database records**, you must sign on to the corresponding subschemas with a SIGNON statement before you issue your SELECT request.

In this example, the EMPLOYEE table resides in the EMPSS01 subschema. The DEPARTMENT table is an ASF-generated table:

1. Sign on to the EMPSS01 subschema:
   ```
   signon ss empss01 dictname testdict id=emp
   ```
2. Sign on to the TEST01 subschema:
   ```
   signon table department dictname asfdict id=dept
   ```
3. Issue your SELECT statement joining the two tables:
   ```
   select *
   from emp.employee a, dept.department b
   where a.dept-id = b.dept-id
   ```

Nesting SELECT statements

You can issue multiple SELECT statements in a single retrieval request. By using more than one SELECT statement, you can apply a more specific search condition than is possible in a single WHERE clause. You can combine SELECT statements in a retrieval request in either of the following two ways:

- You can specify the subselect in the WHERE clause of the higher level SELECT statement.

Example:

To list the departments containing more than two employees:

```
select * from department where 2<(select count(*) from employee where employee.dept-id = department.dept-id) ! display
```

<table>
<thead>
<tr>
<th>DEPARTMENT NAME</th>
<th>DEPT ID</th>
<th>DEPT HEAD ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVELOPMENT</td>
<td>20</td>
<td>1127</td>
</tr>
<tr>
<td>PERSONNEL</td>
<td>30</td>
<td>4430</td>
</tr>
</tbody>
</table>

- You can include **existential quantifiers** (EXISTS or NOT EXISTS) in the higher level SELECT statement. CA OLQ evaluates the higher level SELECT statement in terms of whether (EXISTS) or not (NOT EXISTS) the nested condition is true.

Nested SELECT statements are enclosed in parentheses. There is no limit to the number of nested SELECT statements, but bear in mind that the statement becomes hard to understand after three or four nesting levels.

The column specification of the higher level SELECT statement must be an asterisk (*), indicating all columns.
Using existential quantifiers

This example lists which department employee Schley works in:

```sql
select * from department where exists (select * from employee where employee.dept-id = department.dept-id and emp-last-name = 'schley')
```

<table>
<thead>
<tr>
<th>DEPARTMENT NAME</th>
<th>DEPT ID</th>
<th>DEPT HEAD ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVELOPMENT</td>
<td>20</td>
<td>1127</td>
</tr>
</tbody>
</table>

Stringing together SELECT statements (UNION)

You can concatenate two or more tables containing like columns, using the UNION statement. The result table contains data found in one or both source tables. The UNION option eliminates duplicate rows from the report. To display duplicate rows, specify UNION ALL.

The two SELECT statements must have the same number of columns. Corresponding columns must have the same:
- Data length
- Data type. For example, floating point, binary, numeric
- Decimal representation

The DISTINCT operand cannot be specified when using the UNION option.

This example lists employee information for all employees in the Development departments of the Massachusetts and New York EMPLOYEE tables. Because ALL is specified, duplicate rows are displayed:

```sql
select * from mass.employee
union all
select * from ny.employee
```

SELECT -- IDMS access mode

Retrieves values from one or more SQL tables and views for display in CA OLQ.

Syntax

```
idms-sql-select-statement
```

⚠️ **Note:** For the syntax, authorization, parameters, usage notes, and examples for the SELECT (IDMS access mode) command, see the CA IDMS SQL Reference Guide.
Coding considerations

The parsing rules are different for SELECT depending on how the access switch is set. For instance, you cannot follow SELECT (IDMS access mode) with a separator or comment character.

Also the use of abbreviations, literals, and operators differs:

<table>
<thead>
<tr>
<th>In IDMS mode</th>
<th>In OLQ mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>:display.SELECT * from emp where emp-lname = 'Smith'</td>
<td>:display.SEL * from emp where emp-lname EQ Smith</td>
</tr>
</tbody>
</table>

In the above example for OLQ mode:

- SELECT is abbreviated to SEL
- The operator EQ is used
- The character string Smith is not enclosed in quotes

You cannot do this with the SELECT command in IDMS mode.

**Note:** Consult *CA IDMS SQL Reference Guide*, for more information about rules for using SELECT (IDMS access mode).

SEND TABLE -- OLQ access mode

SEND TABLE stores information from the current or named report file as an ASF table.

**Access mode:** The syntax below is invalid when the access switch is set to IDMS.

The columns that make up the new ASF table are derived from existing fields in the current report file.

This article describes the following information:

- Syntax (see page 136)
- Parameters (see page 136)
- Considerations (see page 139)
- Examples (see page 139)
### Syntax

- **SEND TABLE**
  - `asf-table-name`
- **REPORT**
  - `report-name`
  - **USER**
    - `user-name`
- **OWNER**
  - `user-name`
- **CATalog**
  - `dictionary-name`
- **LOCATION**
  - `dictionary-node`
- **TYPE**
  - **CREate**
    - **APPend**
    - **ADD**
    - **REPlace**
  - **ONLine**
    - `NO` - `YES`
- **AREa**
  - `area-name`
- **MODify**
  - **ERase**
    - **access**
      - `NO` - `YES`
  - **STOrage sequence**
    - `NEXT`
    - `PRIor`
    - `LASt`
    - `FIRst`
- **INDices**
  - `field-name`
    - **ASCending**
    - **DESCending**
    - **LAST**
    - **FIRst**
    - **NOT ALLOWed**
    - **DISPLAY**
    - **STORage**
- **SUMmary**
- **DETail**
- **COLumns**
  - `field-reference`
    - `MAXimum`

### Parameters

- **asf-table-name**
  - The name of the ASF table to be saved in CA IDMS or CA IDMS/DB.
- **REPORT= report-name**
  Identifies the saved report containing the columns that define the ASF table. *Report-name* names a report that was created with the SAVE REPORT command. If REPORT=report-name is not specified, the ASF table is defined by using columns from the current report.

- **USER= user-name**
  Identifies the user who saved the named report. If USER=user-name is not specified, the report is retrieved from the current user’s directory.

- **OWNER= user-name**
  Specifies the user ID for the owner of the ASF table. If user-name is not specified, the ASF table is saved under the current user’s ID.

- **CATALOG= dictionary-name**
  Specifies the name of the IDB or ASF dictionary where the catalog entry to the named ASF table is added.

- **LOCATION= dictionary-node**
  Specifies the name of the Distributed Database System (DDS) node controlling the named dictionary.

- **TYPE=**
  Specifies whether the named ASF table is being created, added to, or replaced. Type can be:
  - **CREATE** -- The ASF table is new and is being assigned an initial definition.
  - **APPEND/ADD** -- New data is added to an existing table definition.
  - **REPLACE** -- The ASF table already exists and is replaced by new data.

  **Note:** When either TYPE=APPEND/ADD or TYPE=REPLACE is specified, the columns in the current report must be the same as the columns in the existing table definition.

- **ONLINE= NO/YES**
  Specifies whether a map and dialog are built for the ASF table. The default is NO.

- **AREA= area-name**
  Names an alternative area to store the ASF table. Users must have DBA authority to specify this option.

- **MODIFY ACCESS= NO/YES**
  Specifies whether or not individual rows in the stored ASF table can be modified through ASF. The default is NO. A logical record MODIFY path is not built in the table subschema.

- **ERASE ACCESS= NO/YES**
  Specifies whether or not individual rows in the stored ASF table can be deleted through ASF. The default is NO.

- **STORAGE SEQUENCE=**
  Specifies how data is added to the database:
NEXT -- Each new DATA record occurrence is connected immediately after the record occurrence that is current of set.

PRIOR -- Each new DATA record occurrence is connected immediately before the record occurrence that is current of set.

LAST -- Each new DATA record occurrence is connected immediately preceding the owner record.

FIRST -- Each new DATA record occurrence is connected to the set in the position immediately following the owner record.

The STORAGE SEQUENCE cannot be specified if the STORAGE parameter is specified in the INDICES statement.

Notes:
- For more information about set order, see the CA IDMS Performance Monitor System Administration Guide.
- For more information about ASF, see the CA IDMS ASF User Guide.

INDICES=
Defines characteristics of the index set for the ASF table.

- field-name -- A single or concatenated key field. You can specify more than one field-reference value for a table.

- ASCENDING/DESCENDING -- The order in which record occurrences are connected to a set sorted by key value. The default is ASCENDING.

- LAST -- A new record with a duplicate sort key value is stored immediately after the existing duplicate record.

- FIRST -- A new record with a duplicate sort key value is stored immediately before the existing duplicate record.

- NOT ALLOWED -- Duplicate sort keys are not allowed.

- DISPLAY -- Specifies a display sequence of that defined as the display sequence for the table in ASF.

- STORAGE -- Specifies a display sequence of that defined as the storage sequence for the table in ASF.

Note: You can specify STORAGE and DISPLAY only once each for any table. You cannot specify STORAGE SEQUENCE= and STORAGE for the same table.
SUMMARY
Specifies that summary report lines only be included in the ASF table.

Note: In addition to specifying SUMMARY, you can create a table which contains summary information only by specifying DISPLAY SUMMARY or by selecting SUMMARY ONLY from the Sort screen in menu mode. Any subsequent SEND TABLE will contain only the summary information.

DETAIL
Specifies that all detail report lines be included in the ASF table.

COLUMNS=
Specifies the report file columns included in the ASF table. Columns can be:

- MAXIMUM -- All sequential columns are saved as columns in the ASF table.

- field-reference -- The columns and the number of characters in each column saved in the ASF table.

Considerations
When an ASF table is created, the names of the field columns in the report file are assigned to the columns in the table definition.

If CA OLQ headers have been assigned to any report fields, these headers are retained as column names; both dynamic headers and CA OLQ headers retrieved from the data dictionary can be assigned to columns in an ASF table.

For multiple line headers, CA OLQ uses the field name for the internal name.

Examples

Send Table

The SEND TABLE command can be used to instruct CA OLQ to replace all data previously associated with an ASF table with new data occurrences from the current report file. The following example replaces the EMP-HOSPITAL table with a single record:

```
send table=emp-hospital owner=bdm catalog=asfdict type=replace
```

OLQ 102017 TABLE PROCESSING HAS BEEN SUCCESSFULLY COMPLETED

Send Table Indices

This example presents the use of INDICES to define the characteristics of the index set for the ASF table:

```
send table=emp-salary online=yes
indices=(emp-id-0415) not allowed
(emp-last-name-0415, emp-first-name-0415)
```
SEND TABLE -- IDMS access mode

SEND TABLE stores information from the current or named report file as an SQL table.

**Access mode:** The syntax below is *invalid* when the access switch is set to **OLQ**.

Whatever data used to construct a report file can be stored as an SQL table.

This article describes the following information:

- Syntax (see page 140)
- Parameters (see page 140)
- Considerations (see page 142)
- Examples (see page 142)

**Syntax**

```
SEND TABLE [ = ] [ schema. ] sql-table-name
```

```
REPORT [ = ] report-name
```

```
USER [ = ] user-name
```

```
DICtname [ = ] dictionary-name
```

```
TYPE [ = ] CREATE
```

```
ADD
```

```
REPLACE
```

```
AREA [ = ] segment.area-name
```

```
INDices [ = ] ( -- field-name )
```

```
SUMmary
```

```
DETail
```

```
COLumns [ = ] ( Maximum field-reference )
```

**Parameters**

- **sql-table-name**
  
  The name of the SQL table to be saved in CA IDMS/DB.
- **schema**
  The name of the schema associated with the SQL table to be saved in CA IDMS/DB.

- **REPORT= report-name**
  Identifies the saved report containing the columns that define the SQL table. *Report-name* names a report that was created with the SAVE REPORT command. If REPORT=*report-name* is not specified, the SQL table is defined by using columns from the current report.

- **USER= user-name**
  Identifies the user who saved the named report. If USER=*user-name* is not specified, the report is retrieved from the current user’s directory.

- **DICTNAME= dictionary-name**
  Specifies the name of the SQL catalog where the named SQL table is added.

- **TYPE=**
  Specifies whether the named SQL table is being created, added to, or replaced. Type can be:
  - **CREATE** -- The SQL table is new and is being assigned an initial definition.
  - **APPEND/ADD** -- New data is added to an existing table definition.
  - **REPLACE** -- The SQL table already exists and is replaced by new data.

  **Note:** When either TYPE=APPEND/ADD or TYPE=REPLACE is specified, the columns in the current report must be the same as the columns in the existing table definition.

- **AREA= segment.area-name**
  Names an alternative area to store the SQL table.

- **INDICES=**
  Defines characteristics of the index for the SQL table.
  - **field-name** -- A single or concatenated key field. You can specify more than one *field-reference* value for a table.
  - **ASCENDING/DESCENDING** -- The order in which record occurrences are connected to a set sorted by key value. The default is ASCENDING.
  - **NOT ALLOWED** -- Duplicate sort keys are not allowed. The key must be unique.

- **SUMMARY**
  Specifies that summary report lines only be included in the ASF table.

  **Note:** In addition to specifying SUMMARY, you can create a table which contains summary information only by specifying **DISPLAY SUMMARY** or by selecting SUMMARY ONLY from the Sort screen in menu mode. Any subsequent SEND TABLE will contain only the summary information.
• **DETAIL**
  Specifies that all detail report lines be included in the ASF table.

• **COLUMNS=**
  Specifies the report file columns included in the SQL table. Columns can be:
  - **MAXIMUM** -- All sequential columns are saved as columns in the SQL table.
  - **field-reference** -- The columns and the number of characters in each column saved in the SQL table.

### Considerations

When an SQL table is created, the names of the field columns in the report file are assigned to the columns in the table definition.

If CA OLQ headers have been assigned to any report fields, these headers are retained as column names; both dynamic headers and CA OLQ headers retrieved from the data dictionary can be assigned to columns in an SQL table.

For multiple line headers, CA OLQ uses the field name for the internal name.

### Examples

#### Send Table

The SEND TABLE command can be used to instruct CA OLQ to replace all data previously associated with an SQL table with new data occurrences from the current report file. The following example replaces the EMP-HOSPITAL table with a single record:

```
send table=employee.hospital dictname=empdict type=replace
```

OLQ 102017   TABLE PROCESSING HAS BEEN SUCCESSFULLY COMPLETED

#### Send Table Indices

This example presents the use of INDICES to define the characteristics of the index set for the SQL table:

```
send table=emp-salary
indices=(emp-id-0415) not allowed
(emp-last-name-0415, emp-first-name-0415)
```

### SET

SET permits the user to set system management parameters in a signon profile or during a session. While some parameters are initially defined at system generation, others are assigned values during the signon process. For more information about system generation, see Using System Generation (https://docops.ca.com/display/IDMS/Using+System+Generation) and Administrating IDMS System Operations (https://docops.ca.com/display/IDMS/Administrating+IDMS+System+Operations).
This article describes the following information:

- **Syntax** (see page 143)
- **Parameters** (see page 144)
- **Examples** (see page 148)

### Syntax

```
SET ACCEss OLQ
  default SCHEMA null-character
  NULL null-character
  INTERRUpt count
  CONTInuation character
  SEPARator character
  COMMENT character
  default DBName
  default DBNODe
  default DICTIONary name
  default DICTIONODe

PRInt SIZe
  LINe
  PRInt LINe
  COUNT
  PRInt LINe
  FUNCTION control-key-value
  MAXIMUM SORt SIZe
  REPORT DICTIONary name
  user-specification
  DATE
  CODetable

USEr user-name
PASSword user-password
INTernal storage page size
REPORT file page size
REPORT DICTIONary name
```

### Expansion of user-specification

```
USEr
PASSword
INTernal storage page size
REPORT file page size
REPORT DICTIONary name
```
Parameters

- **ACCESS=**
  Specifies the type of table CA OLQ will access.
  - **OLQ --** Allows you access to ASF tables.

The OLQ setting also allows you to use the SELECT statement with the following entities:

- ASF tables
- Logical records
- Database records
- Sequential files (batch only)

This is the default.

- **IDMS --** Allows you access to SQL tables when using the SELECT statement:

  **Access mode:** Some CA OLQ commands are invalid when the access switch is set to **IDMS**.

  **Note:** For more information about what commands become invalid, see the particular CA OLQ command in this chapter.

- **DEFAULT SCHEMA=**
  Sets a default schema so that any reference to table-name becomes schema.table-name.
  **Access mode:** This parameter has no meaning when the access switch is set to **OLQ**.
  Specifies the name of the schema. **Schema-name** be a name 1-18 characters long that follows the conventions for SQL identifiers.

  **Note:** For more information about schemas, see the **CA IDMS SQL Reference Guide**.

- **NULL=**
  Specifies a display character used to portray data columns containing nulls. The default is the period (.).

  **Note:** For more information about null processing, refer to the **CA IDMS SQL Reference Guide**.

- **INTERRUPT COUNT=**
  Specifies the number of records read before a retrieval is interrupted.
CONTINUATION CHARACTER=
Specifies the character used to denote the continuation of a command. The continuation character is used in qfiles or batch input when the length of a CA OLQ command is greater than one line. The continuation character is a one-character alphanumeric value or special character. The system generation default is the hyphen (-).

SEPARATOR CHARACTER=
Specifies the character used to denote the separation of commands. The separator character is used to concatenate CA OLQ commands, permitting the entry of multiple commands on a single input line. The separator character is a one-character alphanumeric value or special character. The system generation default is the exclamation point (!).

Use with SELECT (IDMS mode)
Anytime you use separators with SELECT (IDMS mode) they must precede the SELECT statement. For instance, the following is valid syntax:

```
delete table employee.job!select all from emp_id
```

However, CA IDMS/DB does not accept the syntax below because the separator (!) comes after the SELECT statement:

```
select all from emp_id!delete table employee.job
```

COMMENT CHARACTER=
Specifies the character used to denote the beginning of comments. The comment character signifies a remark; all text following the comment character is ignored during execution. The comment character is a one-character alphanumeric value or special character. The system generation default is the semicolon (;). All text following the comment character is ignored during execution.

Access mode: You cannot use comment characters with SELECT (IDMS mode) statements.

DEFAULT DBNAME=
Specifies the default database name.

DEFAULT DBNODE=
Specifies the default database node.

DEFAULT DICTNAME/CATALOG=
Specifies the default dictionary name.

DEFAULT DICTNODE=
Specifies the default dictionary node.

PRINT LINE SIZE=
Specifies the print line size.

PRINT LINE COUNT=
Specifies the print line count.
- **FUNCTION=**
  SET FUNCTION specifies a value for a control key.
  
  - *control-key-value* -- The control key that is being assigned a value. Valid control keys values are [PA1], [PA2], and 1 through 99 (corresponding to PF keys 1 through 99).
  
  - *command-line* -- The CA OLQ command that is assigned to the specified control key. If the command contains any special characters or embedded blanks, enclose *command-line* in quotation marks.

- **MAXIMUM SORT SIZE=**
  Specifies the amount of storage allocated for sorts.
  
  - *sort-size* -- The maximum size, in kilobytes, of main memory available for sorts. *Sort-size* is an integer in the range 1 to 32767. After this space is used, CA OLQ uses the scratch area for sorts.

- **REPORT DICTNAME=**
  Specifies the dictionary used for saving report information. Job control language for batch jobs is also stored here.
  This command is valid only in CV batch mode.
  
  - *dictionary-name* -- The dictionary where the catalog containing saved report information resides. *Dictionary-name* is a 1 to 8-character alphabetic name.

  **Note:** If you don’t set the dictionary name, the report is saved in the primary dictionary. You must set the dictionary name to match the one defined during system generation for storing reports. This keeps the online and batch reports stored in the same place. If you do not match the dictionary names, you can’t use online CA OLQ to access the reports saved through batch.

- **user-specification**
  
  - *USER=* -- Identifies a user to CA OLQ in online and batch environments. CA OLQ uses the ID and password you assign with the SET USER statement in place of the DC/UCF signon user ID and password.
  
  You can issue this command either in an online command mode CA OLQ session or in your batch job stream.
  
  - *user-id* -- The 1 to 32-byte alphanumeric user’s identifier.
  
  - PASSWORD=*user-password* -- Assigns a password to the user. *User-password* is a 1 to 8-character alphanumeric literal.
  
  - INTERNAL STORAGE PAGE SIZE=*storage-page-size* -- For use in the CA OLQ batch environment only. Specifies the internal storage page size in bytes. *Storage-page-size* is an integer value in the range 1 to 32,767 and should be equivalent to the page size of the CA IDMS/DC region.
Note: For more information about how to specify page sizes for the CA IDMS/DC region, see the CA IDMS System Generation Guide.

- REPORT FILE PAGE SIZE=report-file-page-size -- For use in the CA OLQ batch environment only. Specifies the report file page size in bytes. Report-page-size is an integer value in the range 256 to 32,767 and should be equivalent to the page size of the CA IDMS/DC region.

- REPORT DICTNAME=dictionary-name -- For use in the CA OLQ batch environment only. Specifies the name of the dictionary where the catalog containing saved report information resides. Dictionary-name is a 1 to 8-byte alphabetic name.

Note: The SET USER command should always be the first CA OLQ batch command.

- DATE= Specifies the date format in CA OLQ. You can use this date option to change the format of the current date by including $DATE in a PAGE HEADER/FOOTER command.

CODETABLE=

- literal-string -- A literal string that can consist of any combination of the following:
  - MONTH -- The word MONTH specifies the name of the month in capital letters.
  - MON -- The word MON specifies just the first three letters of the month name, in capital letters.
  - Mon -- The word Mon specifies just the first three letters of the month name, with the initial letter capitalized.
  - Month -- The word Month specifies the name of the month spelled out with the initial letter capitalized.
  - month -- The word month specifies the name of the month spelled out in lowercase letters.
  - mon -- The word mon specifies the first three letters of the month name in lowercase letters.
  - MM/ZM -- The letters MM or ZM represent the month: MM specifies the month be displayed with leading zeros; for example, February would appear as 02. ZM specifies the month be displayed without leading zeros; for example, February would appear as 2.
  - DD/ZD -- The letters DD or ZD: DD specifies the day be displayed with zeros; for example, Feb 9 would appear as Feb 09. ZD specifies the day be displayed without zeros; for example, Feb 09 would appear as Feb 9.
  - CC/YY/YYYY -- The letters CC, YY, or YYYY to represent the year: CC specifies the year be displayed as the century number. For example, 20 would appear for any date between 1900 and 1999.
YY specifies the year be displayed as the last two digits of the year. For example, the year 1996 would appear as 96. YYY specifies the year be displayed as the entire four digit year. For example, the year 1996 would appear as 1996.

You can specify these parameters in any order. Note that you can substitute the values themselves in place of the variables. For instance, instead of specifying DATE=Month DD, YYY, you can specify DATE = January 27, 1996.

- Specifies the code table to translate menu mode syntax and month literals used in the $DATE function.

**Examples**

**INTERRUPT COUNT**

In the following example, a SELECT statement retrieves all data occurrences for the EMPLOYEE record:

```sql
select * from employee
```

OLQ 098006 00 57 whole lines and 0 partial line in report.
OLQ 098007 00 57 records read. 57 records selected.

The following SET INTERRUPT COUNT command reduces the number of records read before a retrieval interruption occurs:

```sql
set interrupt count = 25
```

When another SELECT statement is issued for the EMPLOYEE record, CA OLQ retrieves only 25 record occurrences:

```sql
select * from employee
```

SET INTERRUPT COUNT 25
OLQ 092010 00 The interrupt count has been modified.
OLQ 098006 00 25 whole lines and 0 partial lines in report.
OLQ 098007 00 25 records read. 25 records selected.
OLQ 098008 00 38 of 98 primary record pages read.
OLQ 098009 00 Continue (yes/no)?

**SEPARATOR CHARACTER**

In the following example, the SET SEPARATOR CHARACTER command identifies the percent sign (%) as the separator character:

```sql
set separator character = '%'
```

OLQ 092014 00 The SEPARATOR CHARACTER has been modified.

After the SET SEPARATOR CHARACTER command has been issued, the designated separator character is used to concatenate two CA OLQ commands:

```sql
select office-code-0450,office-street-0450 from office %display
```

OFFICE REPORT

<table>
<thead>
<tr>
<th>mm/dd/yy</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFICE-CODE-0450</td>
</tr>
</tbody>
</table>

| 002 |
| 567 BOYLSTON ST |

| 003 |
| 38 of 98 primary record pages read. |

| 004 |
| Continue (yes/no)? |

15-Jan-2018 148/294
COMMENT CHARACTER :p

In the following example, the SET COMMENT CHARACTER command identifies the pound sign (#) as the comment character:

```sql
set comment character = '#'
```

The established comment character is used to enter remark text after a SELECT statement is issued:

```sql
select * from employee
  where emp-last-name like '%ing'
end
```

DEFAULT DICTNAME

In the following example, the SET DEFAULT DICTNAME command identifies DOCANWK as the default dictionary name:

```sql
set default dictname = docanwk
```

When the following HELP command is issued, CA OLQ lists all subschemas that exist within the designated default dictionary:

```sql
help subschemas
```

SET FUNCTION

In the following example, a SHOW statement has been assigned to PF8:

```sql
set function 8 'show report'
```

When [PF8] is pressed in command mode, CA OLQ displays the description of the current report.
SIGNON

SIGNON indicates to CA OLQ that a named subschema is to be used to perform retrievals. Associating a view-id with a subschema allows access to multiple subschemas during a retrieval.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

When you enter SIGNON during a session in progress, CA OLQ releases all database currencies previously established and automatically signs you on to the new database view.

Note: Control key settings and session options maintain their values across subschema signons.

This article describes the following information:

- Syntax (see page 150)
- Parameters (see page 150)
- Examples (see page 151)

Syntax

```
SIGNon SUBschema subschema-name
   [SCHema schema-name (version)]
   [DICtname dictionary-name]
   [DICTNODE dictionary-node]
   [DBName database-name]
   [DBNODE database-node]
   [VIEW view-id]
```

Parameters

- **SUBSCHEMA= subschema-name**
  Identifies the subschema for the current CA OLQ session. The named subschema must exist as a compiled description in the data dictionary and as a load module in either the dictionary load area or in the load (core-image) library.
• **SCHEMA= schema-name (version)**
  Specifies the schema associated with the subschema:

  - `schema-name` -- The schema associated with the named subschema. `Schema-name` defaults to the schema under which the subschema was last compiled.
  
  - `(version)` -- The version number, in the range 1 through 9999, of the named schema. `Version` defaults to the highest version number existing for that schema. If specified, `version` must be enclosed in parentheses.

• **DICTNAME= dictionary-name**
  Specifies the dictionary from which CA OLQ takes record and set definitions.

  ![](Note: To determine which dictionaries and databases are available, use the DCMT DISPLAY DBNAME TABLE command of DC/UCF.)

  See the *CA IDMS System Tasks and Operator Commands Guide* for details on the DCMT DISPLAY DBNAME TABLE command.

• **DICTNODE= dictionary-node**
  Specifies the Distributed Database System (DDS) node controlling the named dictionary.

• **DBNAME= database-name**
  Specifies the database from which CA OLQ retrieves data. DBNAME can identify a user database or a database dictionary.

• **DBNODE= database-node**
  Specifies the DDS node that controls the named database.

• **VIEW= view-id**
  Specifies an ID by which the subschema can be referred in retrieval commands. `VIEW=view-id` is required for retrieval from multiple subschemas.

### Examples

Default values for dictionary name, dictionary node, database name, and database node can be set with the DCUF SET DBNAME command, initiation of a CA OLQ session through the transfer control facility, or with the SET command.

**Signon subschema**

This example shows a user signon to a subschema and the OLQ response. A subschema (EMPSS01) and dictionary name (DOCUNET) are provided in the SIGNON command:

```
SIGNON dic=docunet ss=empss01
```

```
OLQ 100021 00  Ready to retrieve data from subschema EMPSS01
OLQ 100022 00  Schema:  EMPSCHM    Version: 100
OLQ 100025 00  Dictionary name:  DOCUNET
```

**Assigning a view ID to the subschema**
This example shows a user signon to CA OLQ and the OLQ response. A subschema (EMPSS01) and subschema view ID (EMP1), and dictionary name (TSTDICT) are provided in the SIGNON command:

```signon ss=empss01 view=emp1 dic=tstdict dbname=data1! signon ss=empss01 view=emp2 dic=tstdict dbname=data2
```

SIGNON SS=EMPSS01 VIEW=EMP1 DIC=TSTDICT
OLQ 100022 00 Schema: EMPSCHM Version: 100
OLQ 100023 00 Database name: DATA1
OLQ 100025 00 Dictionary name: TSTDICT
OLQ 100027 00 View Id: EMP1
SIGNON SS=EMPSS01 VIEW=EMP2 DIC=TSTDICT
OLQ 100021 00 Ready to retrieve data from subschema EMPSS01
OLQ 100022 00 Schema: EMPSCHM Version: 100
OLQ 100023 00 Database name: DATA2
OLQ 100025 00 Dictionary name: TSTDICT
OLQ 100027 00 View Id: EMP2

### SIGNON TABLE

SIGNON TABLE allows you to efficiently process multiple requests against a single ASF table.

**Access mode:** The syntax below is **invalid** when the access switch is set to **IDMS**.

#### Syntax

```
SIGNon TABLE asf-table-name [view-id] CATALOG=dictionary-name
LOCation=dictionary-node
OWNer=owner-name
VIEW=view-id
```

#### Parameters

- **SIGNon TABle**
  - `asf-table-name`:
    - Specifies a 1 to 32-character alphanumeric table name.
  - `view-name`:
    - Specifies a 1 to 8-character alphanumeric table view ID.
  - **CATALOG**=`dictionary-name`:
    - Specifies the name of the dictionary containing the catalog entry for the named ASF table.
LOCATION= dictionary-node
Specifies the name of the Distributed Database System (DDS) node controlling the named dictionary.

OWNER= owner-name
Specifies the user ID for the owner of the ASF table. If owner-name isn't specified the current user ID is used.

VIEW= view-id
Specifies a user-supplied label identifying the subschema. View-id is a 1 to 8-character label used to qualify entity names.

Example
This example shows a user signon to an ASF table and the CA OLQ response. The ASF table name and owner name are supplied in the signon command:

```
signon table=employee owner=dmc
```

```
OLQ 100021 00 Ready to retrieve data from subschema RU000426
OLQ 100022 00 Schema: IDMSR Version: 1
OLQ 100023 00 Database name: ASFDICT
OLQ 100025 00 Dictionary name: ASFDICT
```

SORT
SORT sequences records within a report file in a user-specified order. A subsequent DISPLAY command displays the report lines in the specified sequence.

- Syntax (see page 153)
- Parameters (see page 154)
- Considerations (see page 156)
- Examples (see page 156)

Syntax
```
SOR
  | REPort = report-name USER = user-name
  | UNIque
  | EQUals
  | ALTseq CODE = (sort-sequence)
  | ON field-reference
```
Parameters

- **REPORT= report-name**
  Identifies the saved report to be sorted. If *report-name* isn't specified, the default is the current report.

- **USER= user-name**
  Specifies the user ID of the user who owns the report. If *user-name* isn't specified, the report retrieved is from the current user's directory.

- **UNIQUE**
  Eliminates extra report lines containing duplicate sort keys. Specify UNIQUE after the SORT verb and before the sort field reference. Note that specifying UNSORT after issuing UNIQUE does not restore the lost lines.

- **EQUALS**
  Maintains the original order of report records with duplicate sort keys. Specify EQUALS after the SORT verb and before the sort field reference.

- **ALTSEQ CODE**
  Specifies an alternative sort sequence. ALTSEQ follows the SORT statement and appears before the first sort field.

  - *(sort-sequence)* -- The alternative sort sequence. The list consists of encode/decode pairs:
    - The encode value represents the hexadecimal value of the data in the database.
    - The decode value represents the hexadecimal value with which to replace the data.
The sort-sequence list must be enclosed in parentheses.

- **code-table-name** -- A code table stored in the data dictionary. The code table is defined in IDD using DDDL syntax. **Code-table-name** cannot be enclosed in parentheses. The table version number must be enclosed in parentheses. Define the code table with the following options:
  - **TYPE=CODE**
  - **SEARCH=LINEAR**
  - **DATA=ALPHANUMERIC**
  - **DUPLICATES NOT ALLOWED**

- **version** -- The version number of the code table. If **version** isn't specified, it defaults to the highest existing version number of the named code table.

- **ON field-reference**
  Identifies the report field used to resequence a record. Both elementary and computed fields can be specified.

- **WITHIN record-name (occurrence)**
  Specifies the scope of the sort. If this clause is omitted, scope defaults to WITHIN REPORT. The scope of the sort is:
  - **Record-name (occurrence)** -- Sequencing is restarted for each new occurrence of the named record. **WITHIN record-name** has no effect if the record name occurs lower on the path than the record that contains the field used for sorting. When the path that created the report file contains more than one retrieval command for the same record type, use **occurrence** to identify the desired occurrence. If specified, **occurrence** must be enclosed in parentheses.

- **REPORT**
  Specifies that sequencing continues over the entire report without regard to a change in record occurrence.

- **AQ**
  Flags fields requiring the alternate collating (sequence) change. **AQ** can appear anywhere after the field name and before the AND for the next field. **AQ** cannot interrupt a SCOPE clause; for example, **ON DEPT-ID IN AQ REPORT**.

- **IN ASCENDING/DESCENDING ORDER**
  Specifies whether records are sequenced in ascending or descending order. The default is ASCENDING.

- **minor-sort-field-specification**
  Specifies a lower level (minor) sort field. Fields named by **field-reference** are sorted within the previously sorted field. The scope and order of the lower level field can be specified as follows:
WITHIN record-name (occurrence)/REPORT specifies the scope of the sequencing. If you specify record-name, sequencing restarts each time a new occurrence of the named record is encountered. The optional occurrence parameter identifies the occurrence when the path that created the report file contains more than one retrieval command for the same record. If specified, occurrence must be enclosed in parentheses.

IN ASCENDING/DESCENDING ORDER -- Fields are sequenced in ascending or descending order. The default is ASCENDING.

Considerations

You can specify up to 22 fields in one SORT command.

The report file can be returned to its original sequence at any time by using the UNSORT command described later in this chapter.

Note that the UNIQUE parameter permanently removes lines from the report.

When running batch you can use the OLQ internal sort or the sort facility of your operating facility.

Examples

The following examples illustrate the use of the SORT command to arrange records in a report file, based on the SELECT statement shown below:

```sql
select dept-name-0410, emp-id-0415
from department, employee
where dept-employee
```

Sort ascending

In the following example, the report is sorted in ascending alphabetical order by department name:

```
sort on dept-name-0410 ! display
```

<table>
<thead>
<tr>
<th>DEPT-NAME-0410</th>
<th>EMP-ID-0415</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>67</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>11</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>101</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>106</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>69</td>
</tr>
<tr>
<td>ACCOUNTING AND PAYROLL</td>
<td>109</td>
</tr>
<tr>
<td>BLUE SKIES</td>
<td>371</td>
</tr>
<tr>
<td>BLUE SKIES</td>
<td>321</td>
</tr>
<tr>
<td>BLUE SKIES</td>
<td>366</td>
</tr>
<tr>
<td>BRAINSTORMING</td>
<td>467</td>
</tr>
<tr>
<td>BRAINSTORMING</td>
<td>341</td>
</tr>
<tr>
<td>BRAINSTORMING</td>
<td>458</td>
</tr>
<tr>
<td>BRAINSTORMING</td>
<td>334</td>
</tr>
<tr>
<td>BRAINSTORMING</td>
<td>457</td>
</tr>
</tbody>
</table>

- 1 -
Sort on ... and on

The following SORT command sorts the records in the report by employee last name within department name; both sort fields are sorted in ascending order:

```
sort on dept-name-0410 and on emp-id-0415 ! display
```

```
DEPARTMENT/EMPLOYEE REPORT
09/22/99

DEPT-NAME-0410            EMP-ID-0415
ACCOUNTING AND PAYROLL      11
ACCOUNTING AND PAYROLL      67
ACCOUNTING AND PAYROLL      69
ACCOUNTING AND PAYROLL      100
ACCOUNTING AND PAYROLL      101
ACCOUNTING AND PAYROLL      106
BLUE SKIES                  321
BLUE SKIES                  366
BLUE SKIES                  371
BRAINSTORMING               301
BRAINSTORMING               334
BRAINSTORMING               341
BRAINSTORMING               457
BRAINSTORMING               458

- 1 -
```

**Sort descending**

In the final example, records are sorted in descending order by EMP-ID-0415 within the scope of the entire report:

```
sort on emp-id-0415 in descending order ! display cols=2,1
```

```
DEPARTMENT/EMPLOYEE REPORT
09/22/99

EMP-ID-0415            DEPT-NAME-0410
9999  PUBLIC RELATIONS
8683  PERSONNEL
479   THERMOREGULATION
476   PUBLIC RELATIONS
472   EXECUTIVE ADMINISTRATION
471   EXECUTIVE ADMINISTRATION
469   THERMOREGULATION
467   BRAINSTORMING
466   BRAINSTORMING
458   BRAINSTORMING
457   BRAINSTORMING
371   BLUE SKIES
366   BLUE SKIES
355   THERMOREGULATION

- 1 -
```

**SUSPEND**

SUSPEND allows you to suspend the current session and return control to the transfer control facility, DC/UCF.
When a CA OLQ session is initiated under the transfer control facility, the SUSPEND command can be used to return control to either a previously suspended task or to the Selection screen, which lists all tasks available within the facility. Otherwise, control is returned to DC/UCF.

When a session is suspended with the SUSPEND command, CA OLQ retains the current report file.

**Syntax**

```bash
SUSpend
```

**SWAP**

SWAP allows you to switch between CA OLQ command mode and the menu facility.

When in command mode, you can specify a menu facility screen name and switch to a particular screen.

This article describes the following information:

- Syntax (see page 158)
- Parameters (see page 159)
- Example (see page 160)
Parameters

- **BATCH**
  The Batch Processing screen

- **BREAK**
  The Report Format - Sort screen

- **HEADER/COLUMN HEADER**
  The Report Format - Header screen

- **COLUMNS SELECT/FIELDS SELECT**
  The Column Select screen

- **TABLE**
  The Table Processing screen

- **DISPLAY/REPORT**
  The Display Report screen

- **EDIT**
  The Report Format - Edit screen

- **GROUP BY/SUMMARY**
  The Report Format - Group By screen

- **LINKAGE SELECT**
  The Linkage Select screen

- **MENU**
  The Menu screen

- **OPTIONS/SESSION**
  The Session Options screen

- **PAGE HEADER/PAGE FOOTER**
  The Page Header/Footer screen

- **PICTURE**
  The Report Format - Picture screen

- **PRINT/DESTINATION**
  The Print Processing screen

- **QFILE**
  The Qfile Processing screen

- **RECORD SELECT**
  The Record Select screen

- **SAVE/SAVE REPORT**
  The Save Report screen
Example

The command SWAP brings you to the Menu screen:

```
CA, Inc.
CA OLQ Release 19.0
*** Menu ***
Page 1 of 3
122000 Select an option and press the ENTER key
Pfkey Option Description Command/ Screen Name Show Help
X --- Data Source for Report ---
  Choose tables TABLE
  Choose subschema SUBschema
--- Retrieval Activity ---
  Choose records from selected subschema RECORD
  Choose columns for report COLUMN
  Retrieve data to build report RETrieve
  Alter database access strategy LINKage
--- Processing Mode ---
  Execute or create a predefined routine QFILE
  View existing or save current report SAVE
  Submit batch report request BATCH
1=HELP 2=GLOBAL HELP 3=QUIT 4=MESSAGE 8=FWD
```

The command SWAP OPTIONS brings you to the Options Screen:

```
CA OLQ Release 19.0
*** Session Options ***
Page 1 of 2
107017 CA OLQ Release 19.0
107019 Copyright(C) yyyy CA, Inc.
Current interrupt count: 100 Current underline character: -
Access IDMS SQL tables: Y (Y/N) Current SQL NULL data value: .
User options: Page Columns Spread: (L-Left,E-Even,M-Max,nn)
Help Change Current option Alternate option
Option -> Report Processing Options <-
  NOFiller FILLer
  FULL SPARse
  HEADER NOHEADER
-> Column Processing Options <-
  OLOHEADER NOOLO HEADER
  PICTURE NOPicture
  CODETABLE NOCODATA
```
SWITCH

SWITCH passes control to the CA IDMS/DC transfer control facility. With the SWITCH command, you can specify any valid CA IDMS/DC or CA IDMS UCF task code to transfer to another DC/UCF task. For example, you can transfer to IDD or the CA-ADS® dialog generator.

Syntax

```
SWItch  [task-code]
```

Parameter

- **task-code**
  Specifies the DC/UCF task to which control is passed. When the SWITCH command is issued without a designated task, control is returned to the last task suspended. If no previous suspended session exists, the transfer control facility Selection screen is displayed.

**Note:** For information on the tasks that are supported by the transfer control facility, see Using Common Facilities (https://docops.ca.com/display/IDMS19/Using+Common+Facilities).

UNSORT

UNSORT returns a report file to its original retrieval sequence, following one or more SORT commands.

**Note:** UNSORT does not return any rows removed by UNIQUE processing.

Syntax

```
UNSort  [REPORT  [=  report-name]  USER  [=  user-name]]
```
Parameters

- **REPORT= report-name**
  Identifies the saved report to be unsorted. If *report-name* is not specified, the current report file is returned to its original sequence.

- **USER= user-name**
  Specifies the user ID of the report owner. If *user-name* is not specified, the report is retrieved from the current user's directory.

Example

In the example presented below, the report file that was resequenced in several ways in the SORT command examples is now returned to its original sequence:

```
unsort ! display
DEPARTMENT/EMPLOYEE REPORT
  09/22/99
EMP-ID-0415  DEPT-NAME-0410

  30     EXECUTIVE ADMINISTRATION
  471    EXECUTIVE ADMINISTRATION
  472    EXECUTIVE ADMINISTRATION
  69     ACCOUNTING AND PAYROLL
  100    ACCOUNTING AND PAYROLL
  11     ACCOUNTING AND PAYROLL
  67     ACCOUNTING AND PAYROLL
  106    ACCOUNTING AND PAYROLL
  101    ACCOUNTING AND PAYROLL
  81     PERSONNEL
  8683   PERSONNEL
  51     PERSONNEL
  91     PERSONNEL
```

- 1. Note that the rows removed by UNIQUE processing are not restored.
Global Syntax

This section presents syntax and rules for the following CA OLQ™ global syntax:

- SELECT Selection Criteria (see page 163)
- SELECT Comparison-Expression (see page 165)
- FIND / GET Selection Criteria (see page 168)
- FIND / GET Comparison-Expression (see page 170)
- Expression (see page 171)
- FIND / GET and COMPUTE Field-Reference Clause (see page 173)
- DISPLAY and SORT Field-Reference Clause (see page 174)
- Field-List Clause (see page 176)

SELECT Selection Criteria

The WHERE clause criteria specifies criteria for selecting record occurrences based on field values of a named database record, logical record, or ASF table.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

This selection criteria is an expansion of the:

- SELECT WHERE criteria
- SELECT HAVING criteria

Syntax

```
 NOT comparison-expression 
   dba-designated-keyword

 EXISTS
 NOT exists (subquery)
```

Syntax Rules

- comparison-expression
  Specifies a comparison operation to be performed using the named operands and operator.
Note: Complete syntax rules for comparison-expression are discussed later in this chapter.

- **dba-designated-keyword**
  Specifies a logical record keyword that is predefined by the DBA. *DBA-designated-keyword* is a keyword that applies to the logical record named in the command. The keyword represents an operation to be performed at the logical record path level and serves only to route the logical record request to the appropriate path; it has no meaning to CA OLQ. When a SHOW LOGICAL RECORDS command is issued, CA OLQ lists the keywords associated with each logical record defined in the current subschema.

Note: This parameter is used in the WHERE clause only.

- **EXISTS/NOT EXISTS (subquery)**
  Evaluates the outcome of *subquery* in terms of whether it is true (EXISTS) or false (NOT EXISTS). *Subquery* is a nested SELECT statement. The SELECT statement must be enclosed in parentheses. The column list of SELECT statements containing the EXISTS or NOT EXISTS operands must be an asterisk (*), specifying all columns. The subselect statement cannot contain:
  - GROUP BY clauses
  - HAVING clauses
  - DISTINCT option
  - ORDER BY clauses
  - UNION clauses

Note: This parameter is used in the WHERE clause only.

- **AND/OR/NOT**
  Names the logical operators to be used in evaluating the WHERE clause. 
  *Table 1* lists the logical operators and their meanings in order of precedence. NOT has the highest precedence, followed by AND, then OR. Parentheses can be used to force the order of evaluation.

**Table 1: Logical Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>NOT comparison</td>
<td>If comparison is false, expression is true.</td>
</tr>
<tr>
<td>AND</td>
<td>Comparison A AND comparison B</td>
<td>If comparison A and comparison B are both true, expression is true.</td>
</tr>
<tr>
<td>OR</td>
<td>Comparison A OR comparison B</td>
<td>If either comparison A or comparison B is true, expression is true.</td>
</tr>
</tbody>
</table>
SELECT Comparison-Expression

Comparison-Expression is used in the SELECT WHERE criteria clause and in the SELECT HAVING criteria clause.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

Syntax

Syntax Rules

- expression
  Specifies a series of constants or variables separated by operators that yields a single value.

- IN data-value
  Compares an expression to a data value or a list of data values:
  - IN -- Matches an expression to a list of one or more data values. The comparison is true if the expression matches one or more of the data values. The IN predicate is equivalent to coding a series of OR expressions. For example:

    ```sql
    select * from employee
    where emp-last-name in ('jones','tanaka', 'anderson')
    ```

    is equivalent to:

    ```sql
    select * from employee
    where emp-last-name = 'jones' or
    emp-last-name = 'tanaka' or
    emp-last-name = 'anderson'
    ```

  - NOT IN -- Compares a column expression to a list of one or more data values. The comparison is true if the column expression does not match any of the data values. The NOT IN predicate is equivalent to coding a series of AND expressions. For example:

    ```sql
    select * from employee
    where emp-last-name not in ('jones','tanaka','anderson')
    ```
is equivalent to:

```sql
select * from employee
where emp-last-name <> 'jones' and
    emp-last-name <> 'tanaka' and
    emp-last-name <> 'anderson'
```

- **data-value** -- The data value or list of data values *(data-value,...)* to which the expression is compared. Each data value must be enclosed in quotation marks. If more than one data value is specified, the list must be enclosed in parentheses and the data values separated by commas.

- **LIKE data-value**
  Searches the expression for a data value.

  - **LIKE** -- Determines whether an expression contains a data value. The comparison is true if the column expression contains the data value.
  
  - **NOT LIKE** -- Determines whether an expression does not contain a data value. The comparison is true if the expression does not contain the data value.

- **Data-value** -- The data value to which the expression is compared. The data value can contain:
  - Alphanumeric characters for an exact match
  - Special characters to use as wild cards
  - Escape characters to exactly match the special characters

<table>
<thead>
<tr>
<th>Object String</th>
<th>Data Value</th>
<th>Example of Syntax</th>
<th>Example of True comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underscore (_)</td>
<td>Any single character</td>
<td>NAME LIKE 'S_J'</td>
<td>True if NAME is exactly 3 characters long and the first character is S</td>
</tr>
<tr>
<td>Percent sign (%)</td>
<td>Any sequence of zero or more characters</td>
<td>NAME LIKE '%C_J'</td>
<td>True if NAME is 3 or more characters long AND the third from last character is C</td>
</tr>
<tr>
<td>Single alphanumeric character</td>
<td>Exact match to that alphanumeric character</td>
<td>NAME LIKE 'MAC'</td>
<td>True if NAME is MAC</td>
</tr>
<tr>
<td>Escape character + underscore (_)</td>
<td>Exact match to the underscore (_)</td>
<td>PARTNUM LIKE 'J115' ESCAPE '*'</td>
<td>True if PARTNUM is 'J115'</td>
</tr>
<tr>
<td>Escape character + percent sign (%)</td>
<td>Exact match to the percent sign (%)</td>
<td>PARTNUM LIKE '%15' ESCAPE '*'</td>
<td>True if PARTNUM is '%15'</td>
</tr>
<tr>
<td>Escape character alone</td>
<td>Exact match to the escape character</td>
<td>PARTNUM LIKE '*****' ESCAPE '*' (note below)</td>
<td>True if PARTNUM is **</td>
</tr>
</tbody>
</table>

The escape character can be any single alphanumeric character and is set by specifying ESCAPE 'escape-character' in your SELECT statement.
MATCHES/CONTAINS
Specifies search conditions as follows:

- **MATCHES** -- A *character-string or mask-value* against which the named field is to be evaluated, character by character. The match must be exact, starting with the first character in the mask. The special characters that can be used for the mask are:
  - Asterisk (*) specifies any character. If an asterisk is specified, the entire MATCHES string must be enclosed in quotation marks.
  - At sign (@) specifies any alphabetic character.
  - Pound sign (#) specifies any numeric character.

If you specify any other character, the match is for that character only. Only the left-most significant characters of the mask need be specified explicitly when the remaining characters in the field are allowed to have any value. For example, to retrieve all addresses where the first two digits of the zip code are 02, the mask value can be specified as follows:

'02'

Unspecified mask characters are treated as if any character were specified. However, if you want to test the zip code field for numeric values only, the mask must be specified as '02##'.

- **CONTAINS** -- Specifies a character string or mask value that you want to search for. The CONTAINS value can appear anywhere in the named field. For example, the character string *EL* appears in *FIELD*, in *ELEMENT*, and in *COMPEL*.

**Note:** MATCHES and CONTAINS apply only to fields with a usage of DISPLAY and do not allow values that contain double-byte string characters.

- **= <> > < >= <=**
  Specifies the comparison operator:
  - = means Equal to
  - <> means Not equal to
  - > means Greater than
  - < means Less than
  - >= means Greater than or equal to
  - <= means Less than or equal to

- **expression**
  Specifies the expression the named condition is compared to.
Note: See the expansion of Expression (see page 171), later in this chapter.

- **field-reference**
  Identifies a field.

Note: For more information, see the expansion of FIND / GET and COMPUTE Field-Reference Clause (see page 173), later in this chapter.

- **function-expression**
  An expression containing a built-in function. Note that built-in functions can be nested.

- **= <> data-value**
  *Data-value* represents data values to which the named field is compared. Specifies that a column expression or column name equals (=) or doesn't equal (<> the specified data value.

- **TO/THRU data-value**
  Specifies a range of data values to which the named field is compared. THRU indicates an inclusive range. TO indicates an exclusive range.

- **BETWEEN data-value and data-value**
  Specifies a range of data values to which the named field is compared. BETWEEN indicates that the named field meets the requirements inclusive of the boundaries specified by *data-value AND data-value*. NOT BETWEEN indicates that the named field doesn't meet the requirements inclusive of the boundaries specified by *data-value AND data-value*.

- **set-name**
  Identifies a set relationship. This sub-clause is valid in the WHERE clause only. *Set-name* is required.

- **record-name**
  is the name of either the owner or member record of the set. Either the owner or member record can be specified or omitted.

- **alt-source-name**
  Specifies an alternative name used to identify records with the same name. *Alt-source-name* is a 1- to 8-character alphanumeric literal.

### FIND / GET Selection Criteria

The **WHERE** clause selection criteria specifies criteria for selecting record occurrences based on field values of a named database record or logical record.

**Access mode:** This criteria clause is **invalid** when the access switch is set to **IDMS**.

This selection criteria is an expansion of the:

- **FIND/GET WHERE criteria**
• COMPUTE GROUP BY HAVING criteria

Syntax

```
| NOT | comparison-expression | dba-designated-keyword |
```

```
| AND | comparison-expression | OR | NOT | dba-designated-keyword |
```

Syntax Rules

• comparison-expression
  Specifies a comparison operation to be performed using the indicated operands and operator.

  ! Warning: Complete syntax rules for comparison-expression are presented later in this chapter.

• dba-designated-keyword
  Specifies a logical record keyword that is predefined by the DBA. Dba-designated-keyword is a keyword that applies to the logical record named in the command. The keyword represents an operation to be performed at the logical record path level and serves only to route the logical record request to the appropriate path; it has no meaning to CA OLQ. When a SHOW LOGICAL RECORDS command is issued, CA OLQ lists the keywords associated with each logical record defined in the current subschema.

  ! Warning: This parameter is used in the WHERE clause only.

• AND/OR/NOT
  Specifies the logical operators to be used in evaluating the WHERE clause. Table 2 lists the logical operators and their meanings in the WHERE clause. Logical operators are evaluated from left to right in order of precedence; NOT has the highest precedence, followed by AND, then OR. You can use parentheses to force the order of evaluation.

Table 2: Logical Operators:

<table>
<thead>
<tr>
<th>Operator Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>NOT comparison</td>
</tr>
<tr>
<td>AND</td>
<td>Comparison A AND comparison B</td>
</tr>
<tr>
<td>OR</td>
<td>Comparison A OR comparison B</td>
</tr>
</tbody>
</table>
FIND / GET Comparison-Expression

Comparison-expression is used in the FIND/GET WHERE criteria clause and in the COMPUTE GROUP BY HAVING criteria clause.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

Syntax

```
function-expression
field-reference

data-value
THRU
data-value

matches
mask-value
contains
character-string

= ¬=
EQ
IS
<=
NE
>=
GT
EXP
LE
LT
```

Syntax Rules

- **function-expression**
  Allows you to use built-in aggregate functions to evaluate data. You can nest built-in functions.

- **field-reference**

  Note: For more information, see FIND / GET and COMPUTE Field-Reference Clause (see page 173), later in this chapter.

- **= ¬= data-value**
  Specifies a range of data values to which the named field is compared.

- **THRU/TO data-value**
  Specifies a range of data values to which the named field is compared. THRU indicates an inclusive range; TO indicates an exclusive range.

- **MATCHES mask-value**
  Specifies a mask value against which the named field is evaluated, character by character. The match is exact, starting with the first character in the mask. The following characters are available for use in the MATCHES clause:
Asterisk (*) specifies any character. If an asterisk is specified, the entire MATCHES string must be enclosed in quotation marks.

- At sign (@) specifies any alphabetic character.
- Pound sign (#) specifies any numeric character.
- Any alphanumeric character -- Specifies a match against itself.

CONTAINS
Specifies a character string or mask value for which the named field is searched. The CONTAINS value can appear anywhere within the named field. For example, the character string EL appears in FIELD, in ELEMENT, and in COMPEL.

- character-string -- An exact sequence of characters for which the named field is searched.
- mask-value -- A combination of specific (for example: T, 22) and general (for example, any numeric digit) characters for which the named field is to be searched. The special mask characters described above for MATCHES also apply to CONTAINS.

**Note:** MATCHES and CONTAINS apply only to fields with a usage of display and do not allow values that contain double-byte character strings.

- = ¬= >= > <= < expression
  Specifies the comparison operator with which the named field is compared.

  **Note:** For more information, see the expansion of expression (see page 171) later in this chapter.

**Expression**

Expression is used in the WHERE criteria clause and in the COMPUTE GROUP BY HAVING criteria clause. Expression can include fields of the following data types:

- Doubleword binary -- COMP PIC S9(16)
- Fullword binary -- COMP PIC S9(8)
- Halfword binary -- COMP PIC S9(4)
- Packed decimal -- COMP-3
- Zoned decimal -- DISPLAY PIC 9(n) or PIC S9(n)
- Floating point -- COMP-1 and COMP-2
CA IDMS Reference - 19.0

- **Display** -- DISPLAY PIC X(n)

Expressions that include bit fields, nonnumeric fields, or nonnumeric constants are flagged as errors.

**Syntax**

- **(expression)**

**data-value**

**field-reference**

**function-expression**

**Syntax Rules**

- **-**
  Minus sign denotes a negative value in the expression.

- **(expression)**
  Allows you to nest expressions. Parentheses override the standard order of precedence.

- **data-value**

- **field-reference**

**Note:** For more information, see FIND / GET and COMPUTE Field-Reference Clause (see page 173) and DISPLAY and SORT Field-Reference Clause (see page 174), later in this chapter.

- **function-expression**
  Allows you to use built-in aggregate functions to evaluate data. You can nest built-in functions.

- **+ - * /**
  Specify the arithmetic operation to be performed, as follows:

  - Plus sign (+) means addition
  - Minus sign (-) means subtraction (the minus sign must be surrounded by blank spaces)
  - Asterisk (*) means multiplication
  - Slash (/) means division

When evaluating expressions, CA OLQ observes the standard order of precedence: multiplication, division, addition, and subtraction, from left to right, with operations in parentheses resolved first.
FIND / GET and COMPUTE Field-Reference Clause

Field-reference is used in the expression clause, which is used in the FIND/GET WHERE criteria clause and the COMPUTE GROUP BY HAVING criteria clause.

Access mode: The syntax below is invalid when the access switch is set to IDMS.

Syntax

Syntax Rules

- **compute-title**
  Specifies the name of a previously computed field used as an operand in the expression. This specification is valid only for the COMPUTE command.

- **view-id**
  Qualifies the record name by specifying the alternate name of the subschema or table from which the record is retrieved.

- **record-name (occurrence)**
  Specifies the record in which the selected field participates. If more than one occurrence of the record appears in the report file or in a retrieval path, the occurrence number of the record can be specified.
  If you use occurrence, separate it from the field name with a period. When this parameter is not used, it defaults to the first record type retrieved in the path definition.

- **group-name**
  Fields in the record can be specified as follows:

  - **olq-header** -- The alternative header defined for CA OLQ use in the data dictionary or by the user. When the header contains more than one line, only the first line is displayed.

  - **field-name (subscript)** -- A field in a one-, two-, or three-dimensional array. Enclose the subscript parameter in parentheses.

Examples

Matches mask-value

When specifying a match, if a character other than one of the mask-value characters is specified, the match is for that character only.
matches 'string'

This MATCHES clause specifies that the characters of the string to be found must be STRING.

Matching any characters

Only the left-most significant characters of the mask need be specified explicitly when the remaining characters in the field are allowed to have any value.

To retrieve all addresses where the first two digits of the zip code are 02, the mask value can be specified as follows:

'02'

Unspecified mask characters are treated as if * (any character) were specified.

Numeric values

If the zip code is to be tested for numeric values only, the mask must be specified as:

'02###'

If the specified mask value is longer than the field being checked, the extra mask characters will be ignored.

DISPLAY and SORT Field-Reference Clause

This field-reference clause is used in the DISPLAY, and SORT commands and allows you to identify a field in several different ways. With the field-reference clause, you are not restricted to specifying a field name when manipulating or displaying report files. You can identify fields in any of the following ways:

- Relative column position of the field in the report file (1 for the first column)
- Computed column header (AVG-SALARY for the computed average salary)
- CA OLQ header for the field (MANAGER-NAME)
- Field name for a particular record, or a field name, which can be further qualified either by a record name and record occurrence or by subscripts.

Syntax

```
column-position  compute-column-title  field-name  olq-header
  {  record.  field-name  (  subscript  )  }
```

Expansion of record
Syntax Rules

- **column-position**
  Specifies the field in the nth column of the report file. Column numbers appear under the header COL in the HELP REPORT display.

- **compute-column-title**
  Specifies the name of a field created by a COMPUTE command. If the computed field name includes blanks or delimiters, it must be enclosed in quotation marks. Computed fields are listed as *COMPUTED* in the HELP REPORT display.

- **record**
  - **view-id** -- Qualifies the record name by specifying the alternate name of the subschema or table from which the record is retrieved.
  - **record-name (occurrence)** -- Specifies the record in which the selected field participates. If the record appears more than once in the path, you can specify the occurrence number of the record. If you use occurrence, separate it from the field specification with a period. Record occurrence names appear under the header RECORD in the HELP REPORT display.
  - **field-name** -- The name of the field in the report file. Db-key field names can be specified in the SORT command, but not in the WHERE clause or the COMPUTE statement. Field names appear under the header FIELD in the HELP REPORT display.
  - **olq-header** -- The alternative record header defined for the field in the data dictionary or by the user. This name appears under the header OLQ HEADER in the HELP REPORT display. When the header comprises more than one line, only the first header line is displayed in the HELP REPORT file.
  User-defined headers are specified by using the EDIT command; these headers apply only to the session in which they are specified.
  - **field-name (subscript)** -- The fields that require subscripts in a one-, two-, or three-dimensional array. Enclose the subscript parameter in quotation marks if followed by **character-count** (see below). The FIELD column in the HELP REPORT display identifies the fields that require subscripts.
  - **character-count** -- The number of left-most characters of the field sorted on or displayed. This parameter allows truncation of a field whose usage is DISPLAY so that an increased number of fields can be displayed at one time. Enclose **character-count** in parentheses. This parameter is not valid for the ON BREAK command.

Example

DEPTPHONE for the phone field in the Department record, PHONE(2) for the second occurrence of the field that contains the phone number.
The DISPLAY command can also reference the initial portion of a report-file field whose usage is DISPLAY.

The HELP/SHOW REPORT command displays the field names, column numbers, and CA OLQ headers available for use in the `field-reference` clause.

**Field-List Clause**

The `field-list` clause is used in data retrieval commands and specifies the fields in the designated record type to be included in the report.

The `field-list` clause overrides the ALL/NONE option in effect at signon, as well as any previously specified FIELDS FOR command or `field-list` clauses for the named record type. The established field list for a record remains in effect until another field list is specified for that record.

Enclose `field-list` in parentheses when you use it in a retrieval command.

**Syntax**

```
►►─┬─ ALL ─────────────────────────────────────────┬─►◄
├─ NONE ────────────────────────────────────────┤
│ ┌──────────────────── , ────────────┐ │
└─▼- ─┬───────────────┬─▼-┘
               field-name
              ( ▼- subscript ▼- )
```

**Syntax Rules**

- **ALL/NONE**
  Specifies whether all or none of the fields within a retrieved record type are included in the internal field list for that record.
  Filler fields are not displayed unless OPTIONS=FILLER is specified.

- **field-name**
  Specifies one or more fields within a retrieved record type included in the internal field list for that record. Keep in mind these rules when specifying `field-name`:
  - Specification of a group item automatically places all elementary fields within the group item in the internal field list.
  - Specification of an elementary field name refers to that field name only.
  - Fields that redefine other fields aren't placed in the internal field list unless specified individually.
(subscript)
Specifies one or more occurrences of a repeating field. Each occurrence is identified by a subscript enclosed in parentheses. Multiple entries are separated by commas and are limited to the number specified in the OCCURS clause of the schema record description. If a repeating field name is specified without a subscript, a second set of parentheses is required.
If a repeating field is specified with one or more references to a repetition, only the specified repetitions are displayed. These rules apply to nested repeating fields:

- Specifying a high-level field displays all associated repetitions of lower-level fields.
- Specifying a low-level field displays that repetition of the low-level field in all repetitions of the high-level field.
- Specifying a low-level field followed by a two- or three-part subscript (the repetition of the high-level field and the repetition of low-level fields) displays a single repetition of that field.
Built-In Functions and Syntax

Built-in functions are predefined functions in CA OLQ™ that allow you to:

- Evaluate expressions according to predefined operations and return results that can be used in command mode processing.
- Perform predefined string, arithmetic, trigonometric, and date/time functions.
- Perform aggregate calculations that are based on the GROUP BY processing of the COMPUTE and SELECT commands. These aggregate calculations include:
  - Sum
  - Average
  - Maximum
  - Minimum
  - Count
  - Product
  - Standard deviation
  - Standard deviation population
  - Variance
  - Variance population

Invoking Built-In Functions

Built-in functions are invoked by specifying an invocation name.

There are five types of built-in functions: aggregate, arithmetic, date, string, and trigonometric. For a list of built-in functions and what they do, see the tables below.

Where you use them:

You can specify arithmetic, date, string, and trigonometric built-in functions in CA OLQ anywhere you would normally specify arithmetic or comparison expressions. You can specify aggregate built-in functions in a:

- Column list of a SELECT statement that has a GROUP BY clause
- COMPUTE expression that has a GROUP BY clause
**HAVING clause**

**Table 5: CA OLQ Aggregate Built-In Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Invocation Example</th>
</tr>
</thead>
</table>
| Return the average (median) value | AVERAGE  
AVERAGE  
AVE  
compute ave-sal = ave(salary) group by dept |
| Return the number of elements | COUNT  
COU  
NUMBER  
NUM  
select count(*) from employee group by dept |
| Return the highest value | MAXIMUM  
MAX  
HIVAL  
HIV  
select max(salary) as 'Top Salary' from employee group by dept |
| Return the lowest value | MINIMUM  
MIN  
LOVAL  
LOV  
select min(salary) as 'Low Sal' from employee group by dept |
| Return the product for all values of a break | PRODUCT  
PROD  
select product (interest-rate) from mutual-funds group by all |
| Return the sum of all values | TOTAL  
SUM  
select sum(salary) as 'Total Salaries' from employee group by dept |
| Return the sample standard deviation of all values | STD  
select std(salary) as 'Standard Deviation Based on Sample' from employee group by job-id |
| Return the population standard deviation of all values | STDP  
select stdp(salary) as 'Standard Deviation Based on Population' from employee group by job-id |
| Return the sample variance of all values | VAR  
select var(salary) as 'Variance Based on Sample' from employee group by job-id |
| Return the population variance of all values | VARP  
select varp(salary) as 'Variance Based on Population' from employee group by job-id |

**Table 6: CA OLQ Arithmetic Built-In Functions**
<table>
<thead>
<tr>
<th>Function</th>
<th>Invocation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return the absolute value of a number</td>
<td>ABSOLUTE-VALUE</td>
<td>select abs(oper1) as 'Difference' from table1</td>
</tr>
<tr>
<td></td>
<td>ABS-VAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@ABS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABS</td>
<td></td>
</tr>
<tr>
<td>Return the value of a number multiplied by -1</td>
<td>INVERT-SIGN</td>
<td>select inv(oper1) as 'Inverted Value' from table1</td>
</tr>
<tr>
<td></td>
<td>INVERT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INV</td>
<td></td>
</tr>
<tr>
<td>Return the natural logarithm of a number</td>
<td>LOG-BASE-E</td>
<td>select loge(oper1) as 'Log Base E' from table1</td>
</tr>
<tr>
<td></td>
<td>LOGNAT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NATLOG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@LN</td>
<td></td>
</tr>
<tr>
<td>Return the common logarithm of a number</td>
<td>LOG-BASE-10</td>
<td>select log10(oper1) as 'Log Base 10' from table1</td>
</tr>
<tr>
<td></td>
<td>LOGCOM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMLOG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOG10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@LOG</td>
<td></td>
</tr>
<tr>
<td>Return the modulus (remainder) of a division operation</td>
<td>MODULO</td>
<td>select mod(oper1 - oper2) as 'Remainder' from table1</td>
</tr>
<tr>
<td></td>
<td>MOD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@MOD</td>
<td></td>
</tr>
<tr>
<td>Return the smallest integer that is equal to or greater than the specified number</td>
<td>NEXT-INT-EQHI</td>
<td>select nexih(balance-due) as 'Balance Due' from invoice</td>
</tr>
<tr>
<td></td>
<td>NEXTINTEH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEXIH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@CEIL</td>
<td></td>
</tr>
<tr>
<td>Return the largest integer that is lower than the specified number</td>
<td>NEXT-INT-EQLO</td>
<td>select nexil(balance-due) as 'Balance Due' from invoice</td>
</tr>
<tr>
<td>Function</td>
<td>Invocation</td>
<td>Example</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>Return a pseudo-random number based on a seed number</td>
<td>RANDOM-NUMBER</td>
<td>compute 'number' = random(13549)</td>
</tr>
<tr>
<td></td>
<td>RANDOM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>@RAND</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RAN</td>
<td></td>
</tr>
<tr>
<td>Return a +1, 0, or -1 depending on whether a number is positive, zero, or negative</td>
<td>SIGN-VALUE</td>
<td>select sigv(op1) as 'Sign' from table1</td>
</tr>
<tr>
<td></td>
<td>SIGN-VAL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIGV</td>
<td></td>
</tr>
<tr>
<td>Return the square root of a number</td>
<td>SQUARE-ROOT</td>
<td>select sqrt(op1) as 'Square Root' from table1</td>
</tr>
<tr>
<td></td>
<td>@SQRT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SQRT</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: CA OLQ Date Built-In Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Invocation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return the conversion of a specified date from one format (Gregorian, calendar, European, or Julian) to another format</td>
<td>DATECHG</td>
<td>compute calendar = datechg(start-date,'G','C')</td>
</tr>
<tr>
<td></td>
<td>DATECHGX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCDATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GCDATEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GJDATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GJDATEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CEDATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CEDATEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EGDATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EGDATEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EJDATE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EJDATEX</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Invocation Example</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>JCDATE</td>
<td>select datefig(start-date, end-date) as 'Senior' from table1</td>
<td></td>
</tr>
<tr>
<td>GEDATE</td>
<td>compute newdate = dateoff (start-date,4)</td>
<td></td>
</tr>
<tr>
<td>CGDATE</td>
<td>select today('C') as 'Day-off'</td>
<td></td>
</tr>
<tr>
<td>CJDATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JGDATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JEDATE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Return the number of days between two specified dates**: DATEDIF
- **Return the date resulting from adding a specified number of days to a specified date**: DATEOFF
- **Return today's date in the format requested**: TODAY
- **Return tomorrow's date in the format requested**: TOMORROW
- **Return the weekday (Monday, Tuesday, etc.) of a specified Gregorian, calendar, European, or Julian date**: WEEKDAY
## CA OLQ String Built-In Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Invocation Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return yesterday's date in the format requested</td>
<td>YESTERDAY select yesterday('C') as 'Day-off'</td>
</tr>
<tr>
<td>Return the concatenation of a specified list of strings</td>
<td>CONCATENATE select concatenate(emp-first-name,emp-last-name) from employee</td>
</tr>
<tr>
<td>Return the substring that results from removing leading and trailing spaces from a string</td>
<td>EXTRACT select extract(emp-last-name) from employee</td>
</tr>
<tr>
<td>Return a fixed-length string of 20, 40, 60, or 80 characters</td>
<td>FIX20 select fix40 concat(extract(emp-fname), ' ', extract(emp-lname))</td>
</tr>
<tr>
<td>Return the string resulting when the first letter in the specified source string is capitalized and all other characters in the string are converted to lowercase</td>
<td>INITCAP compute new-emp-lname = initcap(emp-lname)</td>
</tr>
<tr>
<td>Return the string resulting from inserting one string into another</td>
<td>INSERT select insert(emp-name,'**',1) from employee where emp-city eq boston</td>
</tr>
<tr>
<td>Return the string that results from left justifying a string</td>
<td>LEFT-JUSTIFY select lefjus(emp-last-name) from employee</td>
</tr>
<tr>
<td>Function</td>
<td>Invocation Example</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Return the starting position of a specified substring</td>
<td>STRING-INDEX select * from invoice where index(prod-code,'ABC') ne 0</td>
</tr>
<tr>
<td>Return the length of a string</td>
<td>STRING-LENGTH select length(extract (emp-first-name)) from employee</td>
</tr>
<tr>
<td>Return the substring of a string, starting from a specified position, and continuing for a specified length</td>
<td>SUBSTR select substr(emp-id, 3,2) from employee</td>
</tr>
<tr>
<td>Return the string that results from right justifying a string</td>
<td>RIGHT-JUSTIFY select rightjus(emp-name) from employee</td>
</tr>
<tr>
<td>Return the string that results from converting all characters to lowercase</td>
<td>TOLOWER compute new-emp-lname = tolower(emp-lname)</td>
</tr>
<tr>
<td>Return the string that results from converting all characters to uppercase.</td>
<td>TOUPPER compute new-emp-lname = toupper(emp-lname)</td>
</tr>
<tr>
<td>Return the string that results from translating characters in a string that also occur in a selection string to corresponding characters in a substitution string</td>
<td>TRANSLATE select trans (course-id,'123','abc') from course-list</td>
</tr>
<tr>
<td>Return the position of the first character in a string that doesn’t occur in a second specified string</td>
<td>VERIFY select emp-name from employee where verify (emp-id, '1234567890 ') ne 0</td>
</tr>
<tr>
<td>Return the string resulting when the first letter of each word in the specified source string is capitalized and all other characters in the string are converted to lowercase</td>
<td>WORDCAP compute new-emp-lname = wordcap(emp-lname)</td>
</tr>
</tbody>
</table>
**Table 9: CA OLQ Trigonometric Built-In Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Invocation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return the arc cosine of a number that represents an angle in degrees</td>
<td>ARCCOSINE-DEGREES</td>
<td>compute 'Arc Cosine' = acosd(angle-in-degrees)</td>
</tr>
<tr>
<td></td>
<td>ARCCOSDEG</td>
<td>ACOSD</td>
</tr>
<tr>
<td>Return the arc cosine of a number that represents an angle in radians</td>
<td>ARCCOSINE-RADIANS</td>
<td>compute 'Arc Cosine' = acosr(angle-in-radians)</td>
</tr>
<tr>
<td></td>
<td>ARCCOSRAD</td>
<td>ACOSR</td>
</tr>
<tr>
<td>Return the arc sine of a number that represents an angle in degrees</td>
<td>ARCSINE-DEGREES</td>
<td>compute 'Arc Sine' = asind(angle-in-degrees)</td>
</tr>
<tr>
<td></td>
<td>ARCSINDEG</td>
<td>ASIND</td>
</tr>
<tr>
<td>Return the arc sine of a number that represents an angle in radians</td>
<td>ARCSINE-RADIANS</td>
<td>compute 'Arc Sine' = asinr(angle-in-radians)</td>
</tr>
<tr>
<td></td>
<td>ARCSINRAD</td>
<td>ASINR</td>
</tr>
<tr>
<td>Return the arc tangent of a number that represents an angle in degrees</td>
<td>ARCTAN-DEGREES</td>
<td>compute 'Arc Tangent' = atand(angle-in-degrees)</td>
</tr>
<tr>
<td></td>
<td>ARCTANDEG</td>
<td>ATAND</td>
</tr>
<tr>
<td>Return the arc tangent of a number that represents an angle in radians</td>
<td>ARCTAN-RADIANS</td>
<td>:display. compute 'Arc Tangent' = atand(angle-in-radians)</td>
</tr>
<tr>
<td></td>
<td>ARCTANRAD</td>
<td>ATANR</td>
</tr>
<tr>
<td>Return the cosine of a number that represents an angle in degrees</td>
<td>COSINE-DEGREES</td>
<td>compute 'Cosine' = cosd(angle-in-degrees)</td>
</tr>
<tr>
<td></td>
<td>COSDEG</td>
<td>COSD</td>
</tr>
<tr>
<td>Return the cosine of a number that represents an angle in radians</td>
<td>COSINE-RADIANS</td>
<td>compute 'Cosine' = cosr(angle-in-radians)</td>
</tr>
<tr>
<td></td>
<td>COSRAD</td>
<td></td>
</tr>
</tbody>
</table>
### Parameters Of Built-In Functions

When coding parameters of built-in functions, use the following guidelines:

- Parameters of a built-in function must be enclosed in parentheses and should be separated by commas.

- Each parameter must appear in a specific position relative to the other parameters.

- Parameters in built-in functions are either string values or numeric values:
  - A **string** value is coded as an EBCDIC variable data field, a nonnumeric literal, or a built-in function that returns a string value.
  - A **numeric** value is coded as an arithmetic expression, a numeric variable data field, a numeric literal, or a built-in function that returns a numeric value.

- Some function parameters have restrictions on the values they can contain.

**Example:**

<table>
<thead>
<tr>
<th>Function</th>
<th>Invocation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>COSR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return the sine of a number that represents an angle in degrees</td>
<td>SINE-DEGREES</td>
<td>compute 'Sine' = sind(angle-in-degrees)</td>
</tr>
<tr>
<td></td>
<td>SINEDEG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SIND</td>
<td></td>
</tr>
<tr>
<td>Return the sine of a number that represents an angle in radians</td>
<td>SINE-RADIANS</td>
<td>compute 'Sine' = sinr(angle-in-radians)</td>
</tr>
<tr>
<td></td>
<td>SINERAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SINR</td>
<td></td>
</tr>
<tr>
<td>Return the tangent of a number that represents an angle in degrees</td>
<td>TANGENT-DEGREES</td>
<td>compute 'Tangent' = tand(angle-in-degrees)</td>
</tr>
<tr>
<td></td>
<td>TANDEG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TAND</td>
<td></td>
</tr>
<tr>
<td>Return the tangent of a number that represents an angle in radians</td>
<td>TANGENT-RADIANS</td>
<td>compute 'Tangent' = tanr(angle-in-radians)</td>
</tr>
<tr>
<td></td>
<td>TANRAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TANR</td>
<td></td>
</tr>
</tbody>
</table>
The value specified in a square root function cannot be negative. These restrictions are specified in the command's syntax rules.

**Absolute Value**

The absolute value function returns the absolute value of a numeric value, which is the numeric value of the number regardless of sign.

**Syntax**

```
ABS (value)
```

**Invocation Names**

- ABS
- @ABS
- ABS-VAL
- ABSOLUTE-VALUE

**Parameter**

- **(value)**
  Specifies the numeric value whose absolute value is calculated.

**Arc Cosine**

The arc cosine functions return the arc cosine (inverse sign) of a numeric value that represents an angle in either degrees or radians.

The single floating point operand returned expresses the angle accurate to decimal places in the range zero to 180 for degrees and zero to \( \pi \) for radians.

**Syntax**

```
ACOSR (value)
```

```
ARCCOSRAD (value)
```

```
ACOSD (value)
```

```
ARCCOSDEG (value)
```
Invocation Names

- ACOSD
- ARCCOSDEG
- ARCCOSINE-DEGREES
- ACOSR
- ARCCOSRAD
- ARCCOSINE-RADIANS

Parameters

- **(value)**
  Specifies the numeric value representing the angle, in degrees or radians, whose arc cosine is calculated. Value must be a value ranging from -1 to +1.

Example

This example uses the arc cosine (degrees) function to calculate the cosine of -0.5:

```
compute 'Arc Cosine' = acosd(-0.5)
Arc Cosine = 120
```

Arc Sine

The arc sine functions return the arc sine of a numeric value that represents an angle in either degrees or radians.

The single operand returned expresses the angle accurate to 10 decimal places in the range -90 to +90 for degrees and -\(\pi/2\) to +\(\pi/2\) for radians.

Syntax

```
►►─┬─ ASIND ────────┬─
  ├─ ARCSINDEG ──────┤
  └─ ARCSINE-DEGREES ┘

►►─┬─ ASINR ────────┬─
  ├─ ARCSINRAD ──────┤
  └─ ARCSINE-RADIANS ┘
```

Invocation Names

- ASIND
- ARCSINDEG
- ARCSINE-DEGREES
- ASINR
- ARCSINRAD
- ARCSINE-RADIANS
Parameter

- **(value)**
  Specifies the numeric value representing the angle, in degrees or radians, whose arc sine is calculated. *Value* must be a value ranging from -1 to +1.

Example

This example calculates the arc sine (in degrees) of 0.8660:

```
compute 'Arc Sine' = asind(0.8660)
Arc Sine = 5.99971
```

Arc Tangent

The arc tangent functions return the arc tangent of a numeric value that represents an angle in either degrees or radians.

The single operand that is returned expresses an angle accurate to 10 decimal places in the range +90 to -90 for degrees and -p/2 to +p/2 for radians.

Syntax

```
►►─┬─ ATAND ────────────┬─ ARCTANDEG ────────┐
    │ ┌─── ARCTAN-DEGREES │
    └─► └────────────────┘

►►─┬─ ATANR ────────────┬─ ARCTANRAD ────────┐
    │ ┌─── ARCTAN-RADIANS │
    └─► └────────────────┘
```

Invocation Names

```
ATAND       ATANR
ARCTANDEG   ARCTANRAD
ARCTAN-DEGREES  ARCTAN-RADIANS
```

Parameter

- **(value)**
  Specifies the numeric value representing the angle, in degrees or radians, whose arc tangent is calculated.
Example

This example calculates the arc tangent (in degrees) of 1.7321:

```
compute 'Arc Tangent' = atand(1.7321)
ARC TANGENT = 6.0007
```

Average

The average function returns the average (mean) value of a specified group of numeric values, based upon changes in the resultant value of the GROUP BY clause.

Syntax

```
AVG (value)
```

Invocation Names

```
AVG
AVE
AVERAGE
```

Parameter

- `(value)`
  Specifies the numeric value or values whose average is being calculated.

Example

This example determines the average salary for each department. The total salary for the employees in DEPT1 = 100000. The number of employees in DEPT1 = 6.

```
compute ave-sal = ave(salary) group by department
AVE-SAL = 16666.66
```

Capitalization

Returns the string that results when the first letter of each word in the specified source string is capitalized and all other characters in the string are converted to lowercase.
Syntax

```
WORDSAP (string)
```

Invocation Name

WORDCAP

Parameter

- **string**
  Specifies the string to be converted.
  *String* can be:
  - A string literal, enclosed in single quotation marks
  - The name of a user-defined variable data field containing the string

The first letter in each word is capitalized and all other characters are converted to lowercase.

Example

In the example below, the word cap function is used on employees' last names:

Initial value:
```
EMP-LNAME: 'O'HEARN
```
Statement:
```
compute new-emp-lname = wordcap(emp-lname)
```
Returned string:
```
'O'Hearn
```

Concatenate

The concatenate function returns the concatenation of a specified list of string values.

Syntax

```
(CON | CONCAT | CONCATENATE) ( string )
```

Invocation Names

CON
CONCAT
CONCATENATE
Parameters

- **(string)**
  Specifies one or more string values that are concatenated to form a single string value.

Examples

**Concatenate fields**
You can concatenate EMP-FIRST-NAME-0415 (PIC X(15)) and EMP-LAST-NAME-0415 (PIC X(15)) so the first name precedes the last name. EMP-FIRST-NAME-0415 is 'ELMER'. EMP-LAST-NAME-0415 is 'OTT'.

```sql
select concatenate(emp-first-name-0415, emp-last-name-0415) from employee
'ELMER          OTT'
```

**Concatenate and extract**
You can use the concatenate function in conjunction with the extract function to concatenate EMP-FIRST-NAME-0415 (PIC X(15)), up to but not including the first blank, with a blank and then with EMP-LAST-NAME-0415 (PIC X(15)). The value of EMP-FIRST-NAME-0415 is 'ELMER' and the value of EMP-LAST-NAME-0415 is 'OTT'.

```sql
select con(extract(emp-first-name-0415), ' ', emp-last-name-0415) from employee
'ELMER   OTT'
```

Cosine

The cosine functions return the cosine of a numeric value that represents an angle in either degrees or radians.

**Syntax**

```
►►─┬─ COSD ──────────────────────►◄
     | COSDEG ──────────►
     └─ COSINE-DEGREES ─┘
```

```
►►─┬─ COSR ──────────────────────►◄
     | COSRAD ──────────►
     └─ COSINE-RADIANS ─┘
```

**Invocation Names**

- COSD
- COSDEG
- COSINE-DEGREES
- COSR
- COSRAD
- COSINE-RADIANS
Parameters

- **(value)**
  Specifies the numeric value representing the angle, in degrees or radians, whose cosine is calculated.

Example

This example calculates the cosine (in degrees) of 60:

```sql
compute 'Cosine' = cosd(60)
COSINE = 5.000
```

Count

The count function returns the number of occurrences of the value identified by the GROUP BY clause.

Syntax

```
COUNT (*)
```

Invocation Names

- `COU`  
- `NUM`  
- `NUMBER`

Example

This example counts the number of employees in a department:

```sql
select count(*) from employee
12
```

Date Change

The date change functions return the conversion of a specified date from one format (Gregorian, calendar, European, or Julian) to another.

This article describes the following information:
• Syntax (see page 194)
• Parameters (see page 194)
• Examples (see page 195)

Syntax

```
Syntax
```

```
Invocation names:
```

```
Parameters
```

• **date**
  Specifies the six-byte object date for the date change function.
  Date can be:

  • A user-supplied numeric literal (enclosed within quotes)
  • The name of a user-defined variable data field

• **century-date**
  Specifies the eight-byte object date (containing the century) for the date change function.
  Century date can be:

  • A user-supplied numeric literal (enclosed within quotes)
  • The name of a user-defined variable data field
'input-date-format'
Specifies the format of the date specified by date or century-date.
Input date can be:

- 'G' -- Gregorian
- 'C' -- Calendar
- 'E' -- European
- 'J' -- Julian

The name of a user-defined variable data field containing the date format

'output-date-format'
Specifies the format to which date or century-date is to be converted.
Output date can be:

- 'G' -- Gregorian
- 'C' -- Calendar
- 'E' -- European
- 'J' -- Julian

The name of a user-defined variable data field containing the date format

Examples

Example 1
In this example, the date change function is used to convert January 28, 1958, from Gregorian to calendar format:

```shell
compute calendar = datechg(580128,'g','c') group by all
012858
```

Example 2
In this example, the date change function is used to convert November 12, 1991, from Julian to Gregorian format. The returned date contains the century portion of the year:

```shell
compute calendar = jgdate(91316,'j','g')
911112
```

Example 3
In this example, the date change function is used to convert January 28, 1958, from Gregorian to calendar format. The returned date contains the century portion of the year:

```shell
compute calendar = datechgx(19580128,'g','c') group by all
01281958
```
Date Difference

The date difference function returns the number of days between two specified dates.

Syntax

►►─── DATEDIF (gregorian-date1, gregorian-date2) ──────────────────────────────►◄

Invocation Name

DATEDIF

Parameters

- **gregorian-date1**
  Specifies the date, in Gregorian format, from which the second date is subtracted to derive the difference in days.

- **gregorian-date2**
  Specifies the date, in Gregorian format, subtracted from the first date to derive the difference in days.

Example

This example determines the number of days between January 28, 1958, and August 11, 1955:

```
select datedif(580128,550811) as 'older' from table1
OLDER = 901
```

Note: If the dates were supplied in reverse order, the value -901 would have been returned.

Date Offset

The date offset function returns the date, in Gregorian format, resulting from adding a specified number of days to a specified date.
Syntax

```
DATEOFF (gregorian-date, offset)
[DATEOFFX]
```

Invocation Name

DATEOFF

Parameters

- **gregorian-date**
  Specifies the date, in Gregorian format, to which the offset is added.

- **offset**
  Specifies the offset, in days, that is added to the specified date. Offset can be a numeric variable data field, a numeric literal, a built-in function that returns a numeric value, or an arithmetic expression. Offset can be negative.

Example

This example determines that date that results from adding four days to January 28, 1958:

```
compute newdate = dateoff(580128, 4)
NEWDATE = 580201
```

Extract

The extract function returns the string that results from removing leading and trailing spaces from a string value.

Syntax

```
EXT (string)
```

Invocation Names

- EXT
- EXTRACT
Parameter

- **(string)**
  Specifies the string value on which the `extract` function is performed.

Example

This example removes leading and trailing spaces from the string contained in EMP-LAST-NAME-0415. The value of EMP-LAST-NAME-0415 is 'VON BUREN'.

```sql
select extract(emp-last-name-0415) from employee
'VON BUREN'
```

Fix

Returns a fixed-length string of 20, 40, 60, or 80 characters.

FIX pads with blanks or truncates the string to the appropriate size. FIX is useful for making SELECT statements UNION compatible.

Syntax

```
FIX20 (string)
FIX40 (string)
FIX60 (string)
FIX80 (string)
```

Invocation Names

- FIX20
- FIX40
- FIX60
- FIX80

Parameter

- **string**
  Specifies the string value on which the `fix` function is performed.
  
  String can be:
  
  - A string literal, enclosed in single quotation marks
  - The name of a user-defined variable data field containing the string
Example

In the example below, FIX40 (in OLQ access mode) is used to create a formatted mailing address report:

```sql
select fix40(concat(substr(emp-fname,1,9),',',emp-lname)) emp-lname, emp-id, 1 from employee
union
select fix40(street), emp-lname, emp-id, 2 from employee
union
select fix40(concat(substr(city,1,10),',state','zip-code), emp-lname, emp-id, 3 from employee
union
select fix40(' ') as detail, emp-lname, emp-id, 9 as line from employee order by 2, 3, 4
```

JOHN MULLHOLLAND
114 LAUREL LANE
SHELBRUNGE FALLS, MA  01210

Index

The index function returns the starting position of a specified string within a string value. If the specified string is not found, a zero is returned.

Syntax

```sql
INDEX (string, search-string)
```

Invocation Names

INDEX
INDEX
STRING-INDEX

Parameters

- **string**
  Specifies the string that is searched.

- **search-string**
  Specifies the string that the index function searches for within `string`. **Search-string** cannot be longer than `string`.

Example

This example lists invoices where the product code contains the string 'ABC':

```sql
select * from invoice where index(prod-code,'abc') ne 0
```
The above select returns all rows whose product code contains the literal, 'abc', somewhere in the string. For instance, the search strings 'ABCDEF' and 'XXXABC' produce success; 'CBAXXX' does not.

**Initial Uppercase**

Returns the string that results when the first letter in the specified source string is capitalized and all other characters in the string are converted to lowercase.

**Syntax**

```
INITCAP (string)
```

**Invocation Name**

`INITCAP`

**Parameter**

- `string`
  
  Specifies the string whose first letter is to be capitalized.

  *String* can be:

  - A string literal enclosed in single quotation marks
  - The name of a user-defined variable data field containing the string

**Example**

In the example below, the initial cap function is used on employees' last names:

Initial value:

```
EMP.LNAME: 'O'HEARN
```

Statement:

```
compute new-emp-lname = initcap(emp-lname)
```

Returned string:

```
'O'hearn
```

**Insert (Function)**

The insert function returns the string that results from a specified string being inserted into a string value starting at a specified position.
Syntax

\[
\text{INS} \quad (\text{string, insertion-string, starting-position})
\]

Invocation Names

\[
\begin{align*}
\text{INS} \\
\text{INSERT}
\end{align*}
\]

Invert Sign

The sign inversion function reverses the sign of a numeric value. A positive numeric value becomes negative; a negative numeric value becomes positive.

Syntax

\[
\text{INV} \quad (\text{value})
\]

Invocation Names

\[
\begin{align*}
\text{INV} \\
\text{INVERT} \\
\text{INVERT-SIGN}
\end{align*}
\]

Parameter

- \( \text{(value)} \)
  Specifies the numeric value whose sign is to be reversed.

Example

This example changes 453.29 from a positive to a negative value: The value of OPER1 is 453.29.

\[
\begin{align*}
\text{select inv(oper1) as 'inverted-value' from table1} \\
\text{INVERTED-VALUE = -453.29}
\end{align*}
\]

Left Justify

The left-justify function returns the string that results from:

- Removing leading blanks from the left side of a string value
- Shifting the remainder of the string value to the left side
- Filling the right side with as many blanks as were removed from the left side

Syntax

```
  ┌───┬─┐
  │LEFT│─┤
  │LEFJUS │
  │LEFT-JUST │
  │LEFT-JUSTIFY ┘
  (string)      ┘
```

Invocation Names

- LEFT
- LEFJUS
- LEFT-JUST
- LEFT-JUSTIFY

Parameter

- (string)
  Specifies the string value on which the left-justify function is performed.

Example

This example left justifies the value contained in EMP-LAST-NAME-0415 (PIC X(15)): The value of EMP-LAST-NAME-0415 is 'OTT'.

```
select lefjus(emp-last-name-0415) from employee
'OTT'
```

Length

The length function returns the length (number of characters) of a string value.

Syntax

```
  ┌───┬─┐
  │SLEN│─┤
  │SLENGTH │
  │STRING-LENGTH ┘
  (string)      ┘
```

Invocation Names

- SLEN
- SLENGTH
- STRING-LENGTH
Parameter

- **(string)**
  Specifies the string value whose length is determined.
To calculate the length of a string value, excluding leading and trailing spaces, use the length function in conjunction with the extract function.

Example

This example determines the length of a name contained in EMP-FIRST-NAME-0415 (PIC X(10)). The value of EMP-FIRST-NAME is 'ELMER'.

```sql
select slength(extract(emp-first-name-0415)) from employee
'ELMER'
5
```

Logarithm

The logarithm functions return the common (base 10) or natural (base E) logarithm of a numeric value.

Syntax

```
LOG10 (value)
@LOG
COMLOG
LOGCOM
LOG-BASE-10

LOGE (value)
@LN
NATLOG
LOGNAT
LOG-BASE-E
```

Invocation Names

```
@LOG @LN
LOG10 LOGE
COMLOG NATLOG
LOGCOM LOGNAT
LOG-BASE-10 LOG-BASE-E
```

Parameter

- **(value)**
  Specifies the numeric value whose logarithm is calculated. *Value* must be greater than zero.
Lowercase

Returns the string that results from converting all characters to lowercase.

Syntax

```
TOLOWER (string)
```

Invocation Names

TOLOWER

Parameter

- **string**
  Specifies the string value on which the lowercase function is performed.
  
  *String* can be:
  
  - A string literal, enclosed in single quotation marks
  - The name of a user-defined variable data field containing the string

Example

In the example below, the lowercase function is used to convert all characters in the last name to lowercase:

Initial value:

```
EMP-LNAME: 'LANCHESTER'
```

Statement:

```
compute new-emp-lname = tolower(emp-lname)
```

Returned string:

```
'lanchester'
```

Maximum

The maximum function returns the highest value in the specified column for all occurrences with the same GROUP BY value.

Syntax

```
MAX (value)
```

MAXIMUM

HIV

HIVAL
Invocation Names

MAX
MAXIMUM
HIV
HIVAL

Parameter

* (value)
The column or expression for which the maximum value is calculated.

Example

This example determines the maximum salary in each department:

```
select max(salary) as 'top salary' from employee
TOP SALARY = '75000'
```

Minimum

The minimum function returns the lowest value in the specified column for all occurrences with the same GROUP BY value.

Syntax

```
MIN
MINIMUM
LOV
LOVAL
```

Invocation Names

MIN
MINIMUM
LOV
LOVAL

Parameter

* (value)
The column or expression for which the minimum value is calculated.

Example

This example calculates the lowest salary in each department.

```
select min(salary) as 'low salary' from employee
LOW SALARY = '17000'
```
Modulo

The modulo function returns the modulus (remainder) of one numeric value divided by another.

Syntax

\[
\text{MOD} \quad \text{MODULO}
\]

(\text{dividend}, \text{divisor})

Invocation Names

- MOD
- @MOD
- MODULO

Parameters

- \text{dividend}
  - Specifies the numeric value that is divided by \text{divisor}.

- \text{divisor}
  - Specifies the numeric value that is divided into \text{dividend}.

Example

This example determines the remainder resulting from the division of two numeric values. The initial value of \text{OPER1} is 43, and the initial value of \text{OPER2} is 10.

```sql
SELECT MOD(oper1, oper2) AS 'remainder' FROM table1
```

REMAINDER = 3

Next Integer Equal or Higher

The next integer equal or higher function returns the smallest integer that is equal to or greater than a numeric value.

Syntax

\[
\text{NEXIH} \quad \text{NEXTINTEH} \quad \text{NEXT-INT-EQHI}
\]

(value)
Invocation Names

@CEIL
NEXIH
NEXTINTEH
NEXT-INT-EQHI

Parameter

- (value)
  Specifies the numeric value to be rounded up to the next integer.

Example

This example raises a balance due amount to the next higher dollar value. The initial value of BALANCE-DUE is 453.29.

```sql
select nexih(balance-due) as 'Balance-due' from invoice
BALANCE-DUE = 454
```

Next Integer Equal or Lower

The next integer equal or lower function returns the largest integer that is equal to or less than a numeric value.

Syntax

```
NEXIL
@INT
@TRUNC
NEXTINEL
NEXT-INT-EQLO (value)
```

Invocation Names

@INT
NEXIL
@TRUNC
NEXTINEL
NEXT-INT-EQLO

Parameter

- (value)
  Specifies the numeric value to be rounded down to the next integer.
Example

This example rounds a balance due amount to the next lower dollar value. The initial value of BALANCE-DUE is 453.29.

```
select nexil(balance-due) as 'balance-due' from invoice
BALANCE-DUE = 453
```

Product

Returns the product of all values in a column.

Syntax

```
PRODUCT (value)
```

Invocation Names

PROD
PRODUCT

Parameter

- **column-name**
  Specifies the list of values on which the product is calculated.

Example

In this example, the compounded interest rate for 2 mutual funds is calculated:

```
select product(interest-rate) from mutual-funds
```

Random Number

The random number function returns a pseudo-random number based on a seed numeric value. The returned random number is greater than zero and less than one, and has a length of nine decimal places.

We suggest you define the seed value with a picture of 9(9) and that you move the result of the function to a variable with a picture of V9(9).
Syntax

.................................................................
\( \text{RAN} \quad (\text{random-number-seed}) \) .................................................................
\( \text{@RAND} \quad \text{RANDOM} \quad \text{RANDOM-NUMBER} \)

Invocation Names

- RAN
- @RAND
- RANDOM
- RANDOM-NUMBER

Parameter

- \((\text{random-number-seed})\)
  Specifies the numeric variable data field that contains the seed value from which the pseudo-random number is calculated. \textit{Random-number-seed} cannot be zero.

Example

This example calculates a random number from the seed value of 13549:

\[
\text{compute 'number' = random (13549)}
\]
\[
\text{NUMBER} = 0.627847111
\]

Right Justify

The right-justify function returns the string that results from:

- Removing blanks on the right side of a string value
- Shifting the remainder of the string value to the right side
- Filling the left side with as many blanks as were removed from the right side

Syntax

.................................................................
\( \text{RIGHT} \quad (\text{string}) \) .................................................................
\( \text{RIGHTJUS} \quad \text{RIGHT-JUST} \quad \text{RIGHT-JUSTIFY} \)
Invocation Names

RIGHT
RIGHTJUS
RIGHT-JUST
RIGHT-JUSTIFY

Parameter

- **(string)**
  Specifies the string value that is right justified.

Example

This example right justifies EMP-LAST-NAME-0415 (PIC X(15)). The initial value of EMP-LAST-NAME-0415 is 'OTT'.

```
select rightjus(emp-last-name-0415) from employee
   'OTT'
```

Sign Value

The sign value function returns a +1, 0, or -1, depending on whether a numeric value is positive, zero, or negative, respectively.

Syntax

```
├─ SIGV
└─ SIGN-VAL ──► SIGN-VALUE ──► (value) ───► 
```

Invocation Names

SIGV
SIGN-VAL
SIGN-VALUE

Parameter

- **(value)**
  Specifies the numeric value whose sign is determined.
Example

This example moves a zero to the SIGN column if an amount is negative, a one to the column if the amount is positive, or a zero to the column if the amount is zero. The initial value of OPER1 is -453.29.

```
select sigv(oper1) as 'sign' from table1
SIGN = -1
```

Sine

The sine functions return the sine of a numeric value that represents an angle in either degrees or radians.

Syntax

```
SIND (value)
SINEDEG
SINE-DEGREES
SINR (value)
SINERAD
SINE-RADIANS
```

Invocation Names

- SIND
- SINEDEG
- SINE-DEGREES
- SINR
- SINERAD
- SINE-RADIANS

Parameter

- (value)
  Specifies the numeric value representing the angle, in degrees or radians, whose sine is calculated.

Example

This example calculates the sine of -60 (in degrees):

```
compute 'sine' = sind(-60)
SINE = -0.86602
```

Square Root

The square-root function returns the square root of a numeric value.
The square-root function returns the square root of a numeric value.

Syntax

```
(\(\text{value}\))
```

Invocation Names

```
SQRT
@SQRT
SQUARE-ROOT
```

Parameter

- `(value)`
  Specifies the numeric value whose square root is calculated. Value cannot be a negative number.

Example

This example calculates the square root of OPER1. The initial value of OPER1 is 256.

```
select sqrt(oper1) as 'square root' from table1
```

SQUARE_ROOT = 16

Standard deviation

Returns the standard deviation of the values in column-name (the square root of the average differences between the values and their arithmetic mean).

The standard deviation function uses the \(n-1\) method (sample standard deviation).

This function calculates standard deviation using the formula shown below:

\[
\sqrt{n-1} \left( \sum_{i=1}^{n} x_i^2 - \left( \sum_{i=1}^{n} x_i \right)^2 \right)
\]

Standard deviation
Syntax

STD (column-name)

Invocation Name

STD

Parameter

- column-name
  Specifies the list of values on which the standard deviation is calculated.

Example

This example displays all rows where the standard deviation of SALARY is greater than $5,000:

```
select std(salary) as 'SALARY STAND-DEV' from employee group by dept-id having std (salary) > 5000
```

Standard deviation population

Returns the standard deviation population of the values in column-name (the square root of the average differences between the values and their arithmetic mean).

The standard deviation population (STDP) function uses the $n$ method ($absolute$ standard deviation).

This function calculates standard deviation using the formula shown below:

$$
\sqrt{\frac{\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}{n}}
$$

Standard deviation population
Syntax

```
STDP  (column-name)
```

Invocation Name

STDP

Parameter

- `column-name`
  Specifies the list of values on which the standard deviation is calculated.

Example

This example displays all rows where the standard deviation population of SALARY is greater than $5,000:

```
select stdp(salary) as 'SALARY STAND-DEV' from employee group by dept-id having stdp (salary) > 5000
```

Substring

The substring function returns the substring of a string value, starting from a specified position and continuing for a specified length.

Syntax

```
SUBS  (string, starting-position, length)
```

Invocation Names

- SUBS
- SUBSTR
- SUBSTRING

Parameters

- `string`
  Specifies the string value from which the substring is taken.
• **starting-position**
  Specifies the numeric starting position of the substring within the string value. **Starting-position** must be a positive number and not greater than the length of the string value.

• **length**
  Specifies the numeric length of the substring within the string value (optional). The sum of **starting-position** and **length**, minus one, cannot be greater than the length of the string value. If **length-evn** is not specified, the substring is taken from the specified starting position to the end of the string value.

**Example**

This example extracts the first seven characters of each employee's last name.

```sql
select substr(emp-last-name-0415,1,7) from employee
```

**EMPLOYEE REPORT**

09/15/96

LITERAT
HEAROWI
PAPAZEU
HEDGEHO
MCDougA

**Sum**

The sum function returns the sum of the specified numeric column with the same GROUP BY value.

**Syntax**

```
SUM (value)
```

**Invocation Names**

SUM  TOTAL  TOT

**Parameter**

• **(value)**
  Specifies the column or columns to be totaled.
Example

In this example, you want to find out the total amount of money being spent on salaries in each department:

```sql
select sum(salary-amount-0420) as 'total salaries' from emposition
```

EMPOSITION REPORT  
09/22/96  
TOTAL SALARIES  
3187500.00

END OF REPORT

Tangent

The tangent functions return the tangent of a numeric value that represents an angle in either degrees or radians.

Syntax

```sql
TAND (value)  
TANDEG  
TANGENT-DEGREES  
TANR (value)  
TANRAD  
TANGENT-RADIANS
```

Invocation Names

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Invocation Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>(value)</td>
<td>TAND</td>
</tr>
<tr>
<td></td>
<td>TANDEG</td>
</tr>
<tr>
<td></td>
<td>TANGENT-DEGREES</td>
</tr>
<tr>
<td></td>
<td>TANR</td>
</tr>
<tr>
<td></td>
<td>TANRAD</td>
</tr>
<tr>
<td></td>
<td>TANGENT-RADIANS</td>
</tr>
</tbody>
</table>

Parameter

- (value)
  - Specifies the numeric value representing the angle, in degrees or radians, whose tangent is calculated.
  - The following rules apply to using value:
    - For TANGENT-DEGREES, value cannot be a value equal to the following expression, where n is any integer:
      \[(n \times 180) + 90\]
    - Thus, value cannot equal values such as -270, +270, -90, or +90 for this function.
For TANGENT-RADIANS, value cannot be a value equal to the following expression:

\((n \times X) + X/2\)

Thus, value cannot equal values such as -X/2 or +X/2 for this function.

Example

This example calculates the tangent of 60 (in degrees):

```
compute 'tangent' = tand(60)
TANGENT = 1.7321
```

Today

Returns today's date in the format requested.

Syntax

```
TODAY TODAYX ('output-date-format')
```

Invocation Names

```
TODAY TODAYX
```

Parameters

- **TODAY/TODAYX**
  Invokes the today function. TODAYX returns a date that contains the century portion of the year.

- **(output-date-format)**
  Specifies the output date format. Output-date-format can be:
  - 'C' for calendar
  - 'E' for European
  - 'G' for Gregorian
  - 'J' for Julian
  - The name of a user-defined variable data field containing the date format
Examples

Example 1

In the example below, the today function is used to display today’s date in the calendar format (where today is March 17, 1996):

Statement:
   compute sys-date = today('c')
Returned value: 031796

Example 2

In the example below, the today function is used to display today’s date in the calendar format (where today is March 17, 1996). The returned date contains the century portion of the year:

Statement:
   compute sys-date = todayx('c')
Returned value: 03171996

Note: In the example above, SYS-DATE must contain the century portion of the year.

Tomorrow

Returns tomorrow’s date in the format requested.

Syntax

<table>
<thead>
<tr>
<th>Tomorrow</th>
<th>TOMMORROWX</th>
</tr>
</thead>
<tbody>
<tr>
<td>('output-date-format')</td>
<td></td>
</tr>
</tbody>
</table>

Invocation Names

TOMORROW
TOMORROWX

Parameters

- TOMORROW/TOMMORROWX
  Invokes the tomorrow function. TOMMORROWX returns a date that contains the century portion of the year.

- 'date-format'
  Specifies the output date format. Output-date-format can be:
The name of a user-defined variable data field containing the date format

Examples

Example 1

In the example below, the tomorrow function is used to display tomorrow’s date in the calendar format (where today is March 17, 1996):

Statement:
   compute sys-date = tomorrow('c')
Returned value: 031896

Example 2

In the example below, the tomorrowx function is used to display tomorrow’s date in the calendar format (where today is March 17, 1996). The returned date contains the century portion of the year:

Statement:
   compute sys-date = tomorrowx('c')
Returned value: 03181996

Note: In the example above, SYS-DATE must contain the century portion of the year.

Translate

The translate function returns a string that results from:

- Extracting characters from an original string that match a selection string
- Replacing the extracted characters with corresponding characters in a substitution string

Characters in a selection string correspond positionally to characters in a substitution string. The first character in the selection string corresponds to the first character in the substitution string, the second corresponds to the second, and so forth.

For example, if the selection string contains the letter A in its tenth position, each occurrence of A in the original string is replaced by the tenth character in the substitution string.
Syntax

TRANS (string, substitution-string ,selection-string )

Invocation Names

TRANS
TRANSLATE

Parameters

- **string**
  Specifies the variable on which the translate function is performed.

- **substitution-string**
  Specifies the substitution string.

- **selection-string**
  Specifies the selection string (optional):
  - If selection-string is longer than substitution-string, the excess characters correspond to blanks.
  - If selection-string specifies the same character more than once, the translate function uses the first occurrence of the character.
  - If selection-string is not specified, the 256-character EBCDIC table is used, consisting of hexadecimals 00 through FF.

Example

This example translates all occurrences of 1, 2, and 3 in course ID values to A, B, and C, respectively. The initial value of COURSE-ID is '321'.
```
select trans(course-id,'123','abc') from course-list
```

Uppercase

Returns the string that results from converting all characters to uppercase.

Syntax

TOUPPER (string)
Invocation Names

**TOUPPER**

Parameter

- **string**
  Specifies the string value on which the uppercase function is performed.
  
  *String* can be:
  
  - A string literal, enclosed in single quotation marks
  - The name of a user-defined variable data field containing the string

Example

In the example below, the uppercase function is used to convert all characters in the last name to uppercase:

Initial value:

EMP-LNAME: ‘Lanchester’

Statement:

```plaintext
compute new-emp-lname = toupper(emp-lname)
```

Returned string:

‘LANCHESTER’

Variance

Returns the variance of the values in *column-name* (the square of the standard deviation).

The variance function uses the *n-1, sample*, method.

This function calculates variance using the formula shown below:

\[
\text{Variance} = \frac{\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}{n-1}
\]

Syntax

```
VAR (column-name)
```
Invocation Name
VAR

Parameter
- column-name
  Specifies the list of values on which the variance is calculated.

Example
The variance function in this example displays the variance between the rows in SALARY-MANAGER:

```
select var(salary-manager) as 'VARIANCE SALARY' from employee group by job-id
```

Variance population

Returns the variance population of the values in column-name (the square of the standard deviation population).

The variance population function uses the n, absolute, method.

This function calculates variance using the formula shown below:

\[
\frac{\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}{n}
\]

Syntax

```
VARP (column-name)
```

Invocation Name
VARP
Parameter

- column-name
  Specifies the list of values on which the variance population is calculated.

Example

The variance function in this example displays the variance population between the rows in SALARY-MANAGER:

```
select varp(salary-manager) as 'SALARY VARP' from employee group by job-id
```

Verify

The verify function returns the position of the first character in a string value that does not occur in a verification string. If every character in the object string value occurs in the verification string, a zero is returned.

Syntax

```
VER (string, verification-string)
```

Invocation Names

VER
VERIFY

Parameters

- string
  Specifies the string value on which the verify function is performed.

- verification-string
  Specifies the string value against whose characters the string value's characters are verified.

Example

This example uses the verify function to return EMP-ID-0415 values that contain numeric values. The initial value of EMP-ID-0415 is 02B6.

```
select emp-last-name-0415 from employee where verify(emp-id-0415,'1234567890') ne 0
```
If EMP-ID-0415 contains only numbers and blanks, the verify function returns a zero. If EMP-ID-0415 contains any other characters, the function returns the position of the first character that is not in the verification string.

Weekday

The weekday functions return the name of the day that corresponds to a specified date.

Syntax

```plaintext
WEEKDAY (date, 'input-date-format')
GWEEKDAY (date)
CWEKDAY
EWEKDAY
JWEEKDAY

WEEKDAYX (century-date, 'input-date-format')
GWEEKDAYX (century-date)
CWEKDAYX
EWEKDAYX
JWEEKDAYX
```

Invocation Names

<table>
<thead>
<tr>
<th>WEEKDAY</th>
<th>GWEEKDAY</th>
<th>CWEKDAY</th>
<th>EWEKDAY</th>
<th>JWEEKDAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEEKDAYX</td>
<td>GWEEKDAYX</td>
<td>CWEKDAYX</td>
<td>EWEKDAYX</td>
<td>JWEEKDAYX</td>
</tr>
</tbody>
</table>

Parameters

- **date**
  Specifies the six-byte object date for the weekday function.
  Date can be:
  - A user-supplied numeric literal (enclosed within quotes)
  - The name of a user-defined variable data field

- **century-date**
  Specifies the eight-byte object date (containing the century) for the weekday function.
  Century date can be:
  - A user-supplied numeric literal (enclosed within quotes)
  - The name of a user-defined variable data field

- **'input-date-format'**
  Specifies the format of the date specified by date or century-date.
  Input date can be:
  - 'G' -- Gregorian
'C' -- Calendar

'E' -- European

'J' -- Julian

The name of a user-defined variable data field containing the date format

Examples

Example 1

This example determines the weekday on which January 28, 1958, fell; the date is provided in calendar format:

```
compute weekday = weekday(012858,'c')
'TUESDAY'
```

Example 2

This example determines the weekday on which November 12, 1991 fell; the date is provided in Julian format:

```
compute weekday = jweekday(91316)
'TUESDAY'
```

Example 3

This example determines the weekday on which January 28, 1958, fell; the date is provided in calendar format. The input date contains the century portion of the year:

```
compute weekday = weekdayx(01281958,'c')
'TUESDAY'
```

Yesterday

Returns yesterday's date in the format requested.

Syntax

```
YESTERDAY ("output-date-format")
```

Invocation Names

YESTERDAY
YESTERDAYX
Parameters

- **YESTERDAY/YESTERDAYX**
  Invokes the yesterday function. YESTERDAYX returns a date that contains the century portion of the year.

- **'output-date-format'**
  Specifies the output date format. Output date can be expressed using:
  - 'C' for calendar
  - 'E' for European
  - 'G' for Gregorian
  - 'J' for Julian
  - The name of a user-defined variable data field that contains the date format

Examples

*Example 1*

In the example below, the yesterday function is used to display yesterday's date in the calendar format (where today is March 17, 1996):

Statement:
```plaintext
compute sys-date = yesterday('c')
```
Returned value: 031696

*Example 2*

In the example below, the yesterdayx function is used to display yesterday's date in the calendar format (where today is March 17, 1996). The returned date contains the century portion of the year:

Statement:
```plaintext
compute sys-date = yesterdayx('c')
```
Returned value: 03161996

⚠️ **Note:** In the example above, SYS-DATE must contain the century portion of the year.
Tailoring the CA OLQ Environment for Ease of Use

You can tailor CA OLQ to suit each user’s needs and facilitate use by predefining your CA OLQ environment as much as possible in the data dictionary through Integrated Data Dictionary (IDD) and during system generation.

This section presents the options available for tailoring the CA OLQ environment for ease of use.

- Data Retrieval (see page 227)
- Using qfiles (see page 229)
- Executing qfiles (see page 230)
- Defining Report Headers (see page 236)
- Synonyms (see page 237)
- Code Tables (see page 237)
- Date Option (see page 238)
- External Pictures (see page 239)

Data Retrieval

In order to facilitate data retrieval in CA OLQ, we encourage the use of:

- The SELECT (IDMS access mode) statement -- If you have the CA IDMS/DB SQL Option
- The SELECT (OLQ access mode) statement -- Automatically generates paths
- Logical records -- Predefine paths through logical records to dictate user access

SELECT (IDMS access mode)

Contents

- SELECT (OLQ access mode) (see page 228)

In order to use SELECT to access SQL tables, you must set the access switch to idms.

How to specify the access switch:

The access switch can be set in the following ways:

1. At system generation time
2. For an individual user in the Dictionary (IDD)
3. For the session, interactively (or until the switch is changed)
Allowing MRR:

Multiple Record Retrieval (MRR) must be allowed before you can use the SELECT (IDMS access mode) statement.

MRR is set in the Dictionary (IDD) with the USER statement.

Note: For more information about the USER statement, see the CA IDMS IDD Quick Reference Guide.

SELECT (OLQ access mode)

Retrieving

The SELECT (OLQ access mode) statement allows you to retrieve multiple records with one command and also to sort and group the information during retrieval.

To enable use of SELECT, set the multiple record retrieval option in IDD. The syntax for the IDD statement is:

```
ADD/MODIFY USER user-name ...
MRR IS ALLOWED
```

Sorting

The ORDER BY clause of the SELECT statement allows you to sort information during retrieval instead of after retrieval. Sorting during retrieval eliminates the need for entering sort commands after the report is built.

To use the ORDER BY clause you must allow sorts through IDD. If the sort option in IDD is not allowed, your users cannot use the ORDER BY clause of the SELECT statement. The syntax for the IDD statement is:

```
ADD/MODIFY USER user-name ...
SORT IS ALLOWED
```

Note: The DISTINCT option implies that CA OLQ does a SORT
Logical Records

With the Logical Record Facility (LRF), you can code logical records (combinations of database records) to decrease the number of calls made to CA IDMS/DB by a CA OLQ query.

With LRF you can eliminate the need for users to choose fields. You can code a logical record subschema and associate it with certain users. The subschema may contain only logical records and dictate what the user can retrieve.

Additionally, you can code many different paths to retrieve records from the database.

Example:

Without a logical record and path, a user would generate an employee salary report with:

```sql
select salary-amount-0420, emp-id-0415, emp-name-0415, dept-id-0410, dept-name-0410
from emposition, employee, department
where emp-emposition and dept-employee
```

With a logical record and path coded by the DBA, the user has two choices for generating the report:

- **In command mode:**
  ```sql
  select * from salary-report
  ```
  Where 'salary-report' is the name of the logical record, and the * (asterisk) stands for all fields within the logical record.

- **In menu mode:**
  Select the logical record from the Record Select screen.

More:

Using Logical Record Facility (https://docops.ca.com/display/IDMS19/Using+Logical+Record+Facility)

Using qfiles

A qfile is a sequence of CA OLQ commands used to build reports. Qfiles can be invoked both online and in batch. Qfiles are stored in the data dictionary and can contain almost any CA OLQ command. Commands not valid for the current CA OLQ environment are flagged or ignored by CA OLQ during execution. With qfiles, you can set up defaults for the CA OLQ operating environment, as well as construct reports.
Building qfiles

You can build qfiles either of two ways:

- Manually through IDD
- Automatically in CA OLQ

⚠️ **Note:** For the syntax required to build qfiles in IDD, see the *CA IDMS IDD Quick Reference Guide.*

To build qfiles in CA OLQ you need only create a report in menu mode or command mode, and then issue a SAVE QFILE `qfile-name` command. CA OLQ automatically creates a qfile containing the commands necessary to rebuild that report.

When you save a qfile with the SAVE QFILE command, the qfile is stored in the data dictionary.

In CA OLQ, you can replace, delete, and save a qfile. The commands that allow you to maintain qfiles in command mode include:

- `SAVE QFILE qfile-name`
- `SAVE QFILE qfile-name REPLACE`
- `DELETE QFILE qfile-name`

In IDD, you can add, modify, and delete qfiles. The commands that allow you to maintain qfiles in IDD include:

- `ADD QFILE NAME IS qfile-name VERSION IS version-number`
- `MODIFY QFILE NAME IS qfile-name VERSION IS version-number`
- `DELETE QFILE NAME IS qfile-name VERSION IS version-number`

Executing qfiles

Qfiles are executed by issuing the `QFILE qfile-name` command. Qfiles can be executed only after being saved.

To execute a qfile named EMP-REPORT, issue the command:

```
qfile emp-report
```

The keyword `QFILE` is optional.
Reporting on qfiles

You can generate reports that list each qfile and its associated source statements (CA OLQ commands) with the CA IDMS/DB dictionary report utility (IDMSRPTS). You can also display the definitions of all qfiles stored in the data dictionary with the Data Dictionary Reporter (DDR).

Generating qfile reports

Qfile definitions are output on the module reports. An alternative method for displaying the qfile source is to use the DDDL command:

```
DISPLAY qfile-name AS SYNTAX
```

**Note:** For the job control language required to execute DDR, see Reporting (https://docops.ca.com/display/IDMS19/Reporting).

**Note:** For more information about using the dictionary to set up qfiles, see Integrated Data Dictionary Options (see page 284).

Special Uses of qfiles

There are three types of qfiles automatically invoked in the following order by CA OLQ at signon:

- **PF key module** -- Defines the defaults for PF keys in command mode, and can contain default settings for other CA OLQ options. The PF key module is associated with the CA OLQ task code and is invoked first.

- **Signon profile qfile** -- Establishes a CA OLQ environment for a user upon entry to CA OLQ. The signon profile qfile is associated with a user ID and is invoked after the pfkey module. The options set in this qfile override those set in the PF key module.

- **Signon qfile** -- Automatically defines a user’s access to a specific subschema at signon to that subschema. The signon qfile is associated with a subschema (which is in turn associated with a user). The signon qfile is invoked after the signon profile qfile and only when security for CA OLQ is on. The options set in this qfile override those set in the signon profile qfile.

**PF Key Module**

A PF key module is executed each time a user issues the CA OLQ task code. The PF key module is not associated with specific users, but with the CA OLQ task code.

**How to create a PF key module**
You create a PF key module as a qfile with IDD, save it in the dictionary, and define it as the pfkey module during system generation:

ADD/MODIFY OLQ ...
PFKEY MODULE IS qfile-name

You can also include global settings in the PF key module. For example, a PF key module might look like:

SET FUNCTION 1 = 'HELP'
SET FUNCTION 2 = 'HELP COMMANDS'
SET FUNCTION 3 = 'QUIT'
SET FUNCTION 4 = 'HELP COMMANDS'
SET FUNCTION 5 = 'DISPLAY'
SET FUNCTION 6 = 'MENU'
SET FUNCTION 7 = 'PAGE BACK'
SET FUNCTION 8 = 'PAGE FORWARD'
SET FUNCTION 9 = 'SWAP'
SET DATE 'Month DD, YYYY'

Signon Profile qfiles

Contents
- Signon qfiles (see page 232)

A signon profile qfile is a set of commands automatically executed upon user signon to CA OLQ. Signon profile qfiles are user specific and can contain PF key assignments, CA OLQ option settings, SET commands, and other global CA OLQ settings.

How to create a signon profile qfile

Create signon profile qfiles through the IDD, and save the sequence of commands as a named qfile in the data dictionary.

You can use IDD to associate a signon profile qfile with a particular user:

ADD/MODIFY/REPLACE USER ...
INCLUDE SIGNON PROFILE IS qfile-nameVERSION IS versionLANGUAGE IS OLQ

Signon qfiles

The signon qfile is associated with a user and a subschema. Signon qfiles execute automatically when a user signs on to a subschema.

A signon qfile allows you to make the users' jobs easier; you can state the field lists for records, and set the CA OLQ report default options to format reports automatically.

How to create a signon qfile

You create a signon qfile through IDD, and save the sequence of commands as a named qfile in the data dictionary. You then associate the qfile with a subschema through IDD.

Once the qfile is saved, associate the qfile with a particular subschema and define it as the signon qfile for a particular user:

ADD/MODIFY/REPLACE USER ...
INCLUDE ACCESS TO SUBSCHEMA ss-nameOF SCHEMA schema-nameVERSION IS version .
SIGNON QFILE IS qfile-name
VERSION IS version

Below is an example of what a signon qfile might look like:

FIELDS FOR EMPLOYEE ARE EMP-ID-0415,
EMP-LAST-NAME-0415, SS-NUMBER-0415
FIELDS FOR DEPARTMENT ARE DEPT-ID-0410, DEPT-HEAD-ID-0410
FIELDS FOR EMPOSITION ARE SALARY-AMOUNT-0420, SKILL-NAME-0420
EDIT EMPLOYEE.SS-NUMBER-0415 PICTURE=99-999-9999

Keep in mind that this qfile does not limit retrieval; if users wish to modify the field lists after
retrieval, they can using an additional FIELDS FOR command.

Including Parameters in qfiles

When building qfiles, you can include parameters and nest qfiles:

- Parameters enable you to produce variations of a report with a single qfile.
- Nested qfiles provide the same type of flexibility as subroutines in computer programs. To switch
  execution from one qfile to another, embed QFILE=qfile-name commands within a qfile. These
  commands invoke previously stored qfiles.

Access mode: You cannot use parameters in qfiles dictionary when the access switch is set to IDMS.

Parameters

Contents

- Nesting qfiles (see page 235)

You can produce variations of a report using a single qfile. You define parameters in a CA OLQ
command into which a user-defined text string can be inserted at execution time.

Access mode: You cannot use parameters in qfiles dictionary when the access switch is set to IDMS.

Example:

You can tailor a qfile to produce standard summary reports for each department. The department ID
is identified as the parameter. You can change the definition of the parameter to reflect the
department you want to report on.

How Parameters work:

You include a parameter in a qfile where you want to change the value of something. For instance;
replace a field name in order to change the field retrieved, or a qfile name in order to change the
next qfile invoked.

Keep in mind that these parameters are positional. The order in which you specify them in the qfile is
the exact order you must specify them when executing the qfile.

In the following example, the department ID value (6666) is replaced with a parameter in order to
change the department being reported on.
Original SELECT:

```sql
select * from department, employee, emposition -
where dept-id-0410 = 6666
```

SELECT with variable parameter:

```sql
select * from department, employee, emposition -
where dept-id-0410 = &dept-no.
```

You define the parameter in the first line of the qfile, and if you want to change the default, you specify the value at execution:

```sql
&dept-no. = 6666
select * from department, employee, emposition -
where dept-id-0410 = &dept-no.
```

If you wanted to override the default, your QFILE command would look like:

```sql
qfile emp-report 2364
```

(Where 2364 is the department number)

Defining Parameters:

Once you have your parameters in the qfile, you must define them in the beginning of the qfile. You do this on the first line of the qfile. If you don't have any parameters to define, you must leave the first line of the qfile blank. The syntax of the parameter line is:

```sql
& parameter-name = parameter-value
```

The `parameter-value` you specify acts as the default. You can override the default by specifying a value at execution of the qfile:

```sql
QFILE qfile-name parameter-value
```

Guidelines for defining parameter statements follow:

- The parameter statement must be the first statement in the qfile.
- The parameters specified in the parameter statement must be in the order in which they appear in the qfile.
- If there are no parameters in the qfile, you must leave the first line of the qfile blank.

Multiple parameters in one command:

You can use more than one variable parameter in a command statement.

Example:

```sql
select &emp-field-name. &dept-field-name. from employee, department
```

The parameter statement that would appear as the first line in the qfile would read:

```sql
&emp-field-name. = EMP-LAST-NAME-0415 -&dept-field-name. = DEPT-ID-0410
```
Nesting qfiles

Nesting qfiles allows you to suspend processing of one qfile to execute another qfile, which, upon completion returns execution back to the first qfile.

**Access mode:** You cannot use parameters in qfiles dictionary when the access switch is set to IDMS.

**What is nesting used for?**

Using this method, you can execute a qfile that retrieves records, calls another qfile to format those records, and returns to the first qfile to then save the formatted records in a report file.

This would be especially helpful if you had one report you wanted formatted in different ways; for example, to format a department report to highlight the employee information, and format the same report to highlight the department information. You could have three qfiles:

- RETRIEVE-DATA -- To retrieve the records
- FORMAT-EMP-REPORT -- To format a report that highlights the employee information
- FORMAT-DEPT-REPORT -- To format a report that highlights the department information

By nesting the qfiles, you could use the same qfile to build the report and then use a variable parameter to call a certain qfile, depending upon how you wanted the report formatted.

**Example:**

RETRIEVE-DATA retrieves records:

```sql
&format='. ' 
select * from department, employee, emposition -
where dept-employee and emp-emposition &format. 
```

In the example above, the first line (&format=' . ') is the parameter definition statement. This statement dictates that the value of the parameter will be supplied at execution time. The last statement (&format). is the parameter that, when specified, calls another qfile.

At execution time, you can issue the command:

```
retrieve-data format-emp-report
```

The FORMAT-EMP-REPORT will be substituted in the parameter &FORMAT. at execution of the qfile and will result in the execution of qfile FORMAT-EMP-REPORT upon the completion of the qfile RETRIEVE-DATA.

**How to nest qfiles**

To nest qfiles, just enter a QFILE qfile-name command in the qfile where you want to transfer execution to the second qfile. When the nested qfile completes execution, control is returned to the first qfile at the command immediately after the last executed QFILE command.
Examples of Nesting qfiles

Nest to any depth:

You can nest qfiles to any depth, as long as the qfiles are not recursively called. If QFILE-1 calls QFILE-2, then within the same nest of qfiles, QFILE-2 cannot call QFILE-1.

Each nested qfile is invoked in turn until the last qfile call is encountered:

```sql
select * from department, employee
save report dept-employee
qfile format-report
```

At the QFILE FORMAT-REPORT command, the qfile named FORMAT-REPORT begins to execute:

```sql
sort on dept-id-0410 in ascending order -
and on emp-id-0415 in ascending order
compute name-max=max(emp-name-0415)
  group by dept-id-0410
```

When the called qfile (FORMAT-REPORT) finishes executing, control is returned to the first qfile at the next command statement:

```sql
save report dept-employee
```

Parameters across qfiles:

You can use parameters across nested qfiles. If you define a parameter in the first qfile, and use the same parameter in the second qfile, the definition stays the same unless you redefine the parameter in the second qfile.

If you do redefine the parameter in the second qfile, the parameter is automatically returned to its first definition when control is returned to the first qfile.

Defining Report Headers

Access mode: CA OLQ does not use headers associated with the desired record in the dictionary when the access switch is set to IDMS.

You can use IDD to define report headers for specific record fields. By defining report headers in the data dictionary, each user can generate meaningful reports without formatting the report headers each time the report is generated.

Use the IDD RECORD ELEMENT substatement:

```sql
REMOVE/REPLACE RECORD ELEMENT IS element-name ... NEW ELEMENT IS new-header
```

Example:

You could change the CA OLQ header from EMP-ID-0415 to EMPLOYEE ID:

```sql
replace record element is emp-id-0415
edit olq header is employee
```
The resulting report contains the column EMPLOYEE instead of the column EMP-ID-0415. The contents of the column are the same, only the headers have changed.

⚠️ **Note:** For more information about using the RECORD ELEMENT statement in IDD, see the *CA IDMS IDD Quick Reference Guide*.

### Synonyms

**Access mode:** CA OLQ does not use synonyms associated with the desired record in the dictionary when the access switch is set to IDMS.

You can use IDD to define synonyms for records and record elements (fields). Menu mode uses these synonyms instead of the record and record element names on appropriate screens. Command mode recognizes these synonyms as the names of records and record elements.

To define synonyms for records, use the IDD statement:

```
ADD/MODIFY RECORD ...
INCLUDE RECORD NAME SYNONYM IS synonym-name ...
LANGUAGE IS OLQ
```

⚠️ **Note:** Note that LANGUAGE IS OLQ is required only for menu mode, in conjunction with the option SYNONYM/NOSYNONYM. Command mode uses any synonym, regardless of language.

To define synonyms for record elements, use the IDD statement:

```
REMOVE/REPLACE RECORD ELEMENT ...
ELEMENT NAME SYNONYM ...
IS element-synonym name
```

### Code Tables

**Access mode:** CA OLQ does not use code tables in the dictionary with records containing fields that are tightly coupled when the access switch is set to IDMS.

You can define a small field to hold a value, and then use a code table to translate that value to the appropriate string upon display of the report.

**Example:**

In a grocery store, each type of apple has a code for the type:
In the record, you can use the code value to identify the item. When you want to print the report, however, you want to see the name of the item. Associate a code table with the record, and when you display the record, the code table translates the code into the appropriate string. A sample code table would look like:

01   MACINTOSH
02   GRAN.SMITH
03   CORTLAND
04   GOLD.DEL
05   RED.DEL

To associate a code table with a field, use the IDD statement:

REMOVE/REPLACE RECORD ELEMENT ...
INCLUDE CODE TABLE IS LIST/encode-value decode-value/NULL

Date Option

You can specify different date formats with the SET DATE command. You can set the format for an entire CA OLQ session by including the SET DATE command in the signon profile qfile or the PF key module. You can also specify a code table to be used when generating the month names.

ADD TABLE NAME IS OLQFRNCH VERSION IS 1
  LANGUAGE IS TABLE
  PUBLIC ACCESS IS ALLOWED FOR ALL
  TYPE IS CODE
  SEARCH IS LINEAR
  ENCODE DATA IS ALPHANUMERIC
  TABLE IS UNSORTED
  DUPLICATES ARE NOT ALLOWED
  VALUES ARE (
    01   JANVIER
    02   FEVRIER
    03   MARS
    04   AVRIL
    05   MAI
    06   JUIN
    07   JUILLET
    08   AOUT
    09   SEPTEMBRE
    10  OCTOBRE
    11  NOVEMBRE
    12  DECEMBRE

To use a codetable specify the SET CODETABLE command.

Including the date in reports:

You can include the current date in reports by specifying $DATE within a report page header or footer. The current date will be substituted upon display or printing of the report.
External Pictures

**Access mode:** CA OLQ does **not** use external pictures associated with the desired record in the dictionary when the access switch is set to IDMS.

You can use IDD to define external pictures for specific report fields. By defining external pictures in the dictionary, each user can generate meaningful reports without formatting the report fields each time the report is generated.

Use the IDD RECORD ELEMENT substatement:

```
REMOVE/REPLACE RECORD ELEMENT ...
EXTERNAL PICTURE IS picture/NULL
```
Using CA OLQ Efficiently

See the following topics to learn how to use CA OLQ more efficiently:
- Controlling Data Retrieval (see page 240)
- Controlling Resource Consumption (see page 246)

Controlling Data Retrieval

Controlling data retrieval involves:
- Using qfiles and logical records to define data retrieval paths
- Using the DML user exit
- Setting the interrupt option and count
- Using a db-key list after a FIND command
- Allowing multiple record retrieval with the SELECT (OLQ access mode) statement

Qfiles

You can predefine data retrieval paths with qfiles, thus defining the information that users are allowed to retrieve. You can further define the information a user is allowed to access by limiting their retrieval to using qfiles only. If you specify that a user can retrieve with qfiles only, it prevents them from using command mode retrieval commands. Unless you specifically prohibit the use of menu mode, the user will be able to retrieve through menu mode.

The syntax required to limit a user's access to CA OLQ to qfiles only is the IDD statement:

```
ADD/MODIFY USER ...
QFILE IS ONLY
```

If you want a user to access CA OLQ through qfiles only, we recommend that you then disallow access to menu mode. To disallow access to menu mode, use the IDD statement:

```
ADD/MODIFY USER ...
MENU MODE IS DISALLOWED
```

Logical records

You can predefine navigation paths and logical records for data retrieval with LRF. This allows you to define the records and fields a user retrieves.
OLQ DML User Exit

OLQ can invoke a user exit prior to issuing any native DML command. This facility enables users to examine, modify, or disallow navigational access to data from within OLQ. This might be used for security enforcement, statistics gathering, or checking for special data values.

To use this feature, the user-written exit must have an entry point of OLQDMLX. To enable the user-written exit, it must be linked with IDMSUXIT and the resulting module is named IDMSUXIT.

The exit program is called in user mode. The registers on entry are as follows:

- R1 -- points to a parameter list that is the same as that generated for a native DML request. This is documented in the CA IDMS Programmer’s Quick Reference Guide.
- R13 -- points to a 16-word save area in which the user exit should save the caller’s registers.
- R14 -- contains the address to which control should be returned.
- R15 -- points to the OLQDMLX1 entry point within the user exit.

The user exit determines if the DML command should or should not be executed and it must set the error status field in the IDMS communications block appropriately. For example, if the user exit determines the current user is allowed access to a specific EMP-ID-0415 record, it would then allow the OBTAIN CALC EMPLOYEE DML command to be executed and set ERRSTAT=0000 in the SSCRTL block. If the current user was denied access to the EMP-ID-0415 record then the user exit could set ERRSTAT=0399 indicating a security error.

See Sample Exit (see page 241) as a model for writing an OLQ DML exit.

Sample Exit

```
OLQDMLX1 TITLE 'SAMPLE USER-WRITTEN DML EXIT FOR OLQ'
***********************************************************************
** THIS PROGRAM IS A TEMPLATE TO BE USED AS AN EXAMPLE FOR PROVIDING Entry INTO AND EXIT FROM AN OLQ USER-WRITTEN DBMS EDIT MODULE FOR RELEASE 18.0 AND LATER.**
***********************************************************************
** THIS IS A SAMPLE ONLY AND NO GUARANTEE IS GIVEN AS TO FUNCTIONALITY, ACCURACY, COMPLETENESS, OR PERFORMANCE.**
***********************************************************************
EJECT

OLQDMLX1 - USER-WRITTEN EXIT FOR DML COMMANDS IN OLQ
```
OLQDMLX1 allows user-defined editing of DML commands before they are issued by OLQ. The edit routine can be used for things such as validating security, keeping statistics, looking for special data values, etc.

If certain records, DML commands, or AREAs are to be selected for editing, an IDMS database procedure should be used.

If many records or many subschemas are to be edited during OLQ processing, this exit should be used.

OLQDMLX1 will be automatically called by OLQ before every DML command if this user exit is linked with module IDMSUXIT with an entry point of OLQDMLX.

REGISTER USGAGE -

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R12</td>
<td>BASE REGISTER</td>
</tr>
<tr>
<td>R13</td>
<td></td>
</tr>
<tr>
<td>R14</td>
<td>RETURN ADDRESS FOR SUBROUTINES</td>
</tr>
<tr>
<td>R15</td>
<td>A(DB/DC INTERFACE)</td>
</tr>
<tr>
<td>R0</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>A(PARAMETER LIST) AT ENTRY AND DURING CALLS</td>
</tr>
<tr>
<td>R2</td>
<td>A(SECURITY REQUEST BLOCK)</td>
</tr>
<tr>
<td>R3</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>WORK REGISTER</td>
</tr>
<tr>
<td>R6</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>A(OLQ GLOBAL WORK AREA)</td>
</tr>
<tr>
<td>R9</td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td></td>
</tr>
</tbody>
</table>

EJECT

DMLXSTG DSECT

* Any user-required storage is defined here

EJECT

WORKAREA DC CL80' '
DMLXSTGL EQU *-DMLXSTG
SSCTRLDS DSECT @SSCTRL

EJECT

User exit entry code is defined here

---------------------------------------------------------------------

MOPT CSECT=OLQDMLX1,ENV=USER
MODE MODE=IDMSDC,WORKREG=R0,QUOTES=YES,DEBUG=YES
USING OLQDMLX,R12
ENTRY OLQDMLX
DC 0F '0',CL8 'OLQDMLX'

OLQDMLX DS 0F

STM R0,R15,0(R13) SAVE OLQ'S REGISTERS
LA R13,16*4(.R13) ADJUST STACK POINTER
LR R12,R15 ADDRESSIBILITY
#GETSTG TYPE=(USER,SHORT),LEN=DMLXSTGL,ADDR=(R11), X PLIST=*,STGID='USER',INIT=X'00'
USING DMLXSTG,R11

LR R5,R13 STACK POINTER
SH R5,=AL2(16*4) A(OLQ'S REGISTERS)
L R14,56(.R5) OLQ'S RETURN REGISTER
CLC =AL2(28),0(R14) IF NOT A DML COMMAND
BNE DMLXEXIT CONTINUE DC PROCESSING

L R1,4(.R5) RESTORE A(OLQ'S PARM LIST)
L R5,4(.R1) A(SSCIDBCM+4)
LA R5,5(.R5)
SR R5,R1 IDBMSCOM (DML verb number)
EJECT
Sample checks for DML commands. The user exit should determine how to proceed for each DML command. Each DML command has an associated DML verb number (https://docops.ca.com/pages/viewpage.action?pageId=380831496). The DML verb number can be checked and analyzed in the subsequent routines.

* Check for 'Bind Run Unit' DML verb number 59

```
DMLX1000 DS 0H
CH R5,=H'59' IF NOT 'BIND RUN UNIT'
BNE DMLX2000 SEE IF THIS IS 'OBTAIN'
* Code 'BIND RUN UNIT' pre-processing here
B DMLXEXIT PERFORM THE 'BIND'
EJECT
```

* Check for 'OBTAIN CALC EMPLOYEE' DML verb number 32

```
DMLX2000 DS 0H
CH R5,=H'32' IF NOT 'OBTAIN CALC'
BNE DMLX3000 SEE IF THIS IS 'FINISH'
L R5,81(R1) A(RECORD NAME)
CLC =&CL16'EMPLOYEE',0(R5) IF NOT 'OBTAIN CALC EMPLOYEE'
BNE DMLXEXIT PERFORM THE 'OBTAIN CALC'
* Code 'OBTAIN CALC EMPLOYEE' processing here
L R5,160(R8) A(RECORD IO BUFFER)
CLC =C'0048',0(R5) IF EMP-ID-0415 NOT = 0048
BNE DMLXEXIT PERFORM THE 'OBTAIN CALC'
ABND2000 L R5,0(R1) A(SSCTRL)
USING SSCTRLDS,R5
MVC ERRSTAT,=C'0399' 'OBTAIN CALC EMPLOYEE' is not allowed
MVC ERRORREC,=C'SECURITY ERROR ' RETURN TO OLQ
B DMLXRETN RETURN TO OLQ
DROP R5
EJECT
```

* Check for 'FINISH' run unit DML VERB number 2

```
DMLX3000 DS 0H
CH R5,=H'2' IF NOT 'FINISH RUN UNIT'
BNE DMLXEXIT PERFORM DML
* Code 'FINISH RUN UNIT' processing here
B DMLXEXIT PERFORM THE 'FINISH RUN UNIT'
EJECT
* User exit 'Exit code' is defined here
* *
* DMLXEXIT - Pass control to IDMS to execute the DML command. IDMS returns control to OLQ after DML command execution.
```

```
DMLXEXIT L R15,=V(IDCSACON) A(CSA)
SH R13,=H'64' POINT TO OLQ'S STACK
LM R8,R14,0(R13) RESTORE OLQ'S Registers
BR R15 EXECUTE REQUESTED COMMAND
* DMLXRETN - The DML command is not executed and control returns to OLQ
```

```
DMLXRETN SH R13,=H'64' POINT TO OLQ'S STACK
LM R8,R14,0(R13) RESTORE OLQ'S Registers
LA R14,2(R14) A(NEXT INSTRUCTION)
BR R14 RETURN TO OLQ
DROP R11
LTORG
#BALI
END OLQDMLX
```
Assembly and Link Edit (z/OS)

// *----------------------------------------------------------------
// * ASSEMBLER IEV90 JOB STREAM
// *----------------------------------------------------------------
//ASMSTEP EXEC PGM=IEV90,
// PARM='ALIGN,XREF,PUNCH,NODECK',
// REGION=2048K
//SYSLIB DD DSN=yourHLQ.CAGJMAC,DISP=SHR
// DD DSN=yourHLQ.CAGJSRC,DISP=SHR
// DD DSN=os390.maclib,DISP=SHR
//SYSUT1 DD DSN=&.&SYSUT1.,UNIT=VIO,SPACE=(1700,(600,100))
//SYSUT2 DD DSN=&.&SYSUT2.,UNIT=VIO,SPACE=(1700,(600,100))
//SYSUT3 DD DSN=&.&SYSUT3.,UNIT=VIO,SPACE=(1700,(600,100))
//SYSPUNCH DD SYSUT1=*
//SYSPUNCH DD DSN=&.&OBJECT.,
// DISP=(NEW,PASS),
// UNIT=SYSDA,
// SPACE=(80,(500,1000))
//SYSIN DD *
OLQ DML Exit program
// *----------------------------------------------------------------
//*r LINK IEWL
// *----------------------------------------------------------------
//LINK EXEC PGM=IEWL,
// PARM='LET,LIST,XREF,RENT',
// REGION=128K,
// COND=(8,LT,ASMSTEP)
//SYSLMOD DD DSN=idms.loadlib,DISP=SHR
//SYSPRINT DD SYSUT1=*
//SYSPRINT DD DSN=yourHLQ.CAGJLOAD,DISP=SHR
//IN1 DD DSN=yourHLQ.CAGJMAC,DISP=SHR
//IN2 DD DSN=yourHLQ.CAGJSRC,DISP=SHR
//SYSLIN DD DDNAME=SYSIN
//SYSLIN DD *
INCLUDE IN1(OLQSDMLE)
INCLUDE IN1(IDMSBALI)
ENTRY ENTRY
MODE AMODE(31),RMODE(ANY)
NAME OLQSDMLE(R)

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yourHLQ.CAGJLOAD</td>
<td>data set name of the CA IDMS SMP/E distribution load library</td>
</tr>
<tr>
<td>idms.loadlib</td>
<td>data set name of the CA IDMS load library</td>
</tr>
<tr>
<td>yourHLQ.CAGJMAC</td>
<td>data set name of the CA IDMS macro library</td>
</tr>
<tr>
<td>yourHLQ.CAGJSRC</td>
<td>data set name of the CA IDMS source library</td>
</tr>
<tr>
<td>os390.maclib</td>
<td>data set name of the z/OS system macro library</td>
</tr>
</tbody>
</table>

Assembly and Link Edit (z/VSE)

// DLBL idmslib,
// EXTENT ,nnnnnn
// LIBDEF *,SEARCH=(idmslib.sublib)
// LIBDEF PHASE,CATALOG=idmslib.sublib
// OPTION CATAL
// EXEC {cf1\cgrid0 ASMA90},SIZE=128K
OLQ DML Exit program
/*
INCLUDE OLQSDMLE
INCLUDE IDMSBALI

15-Jan-2018  244/294
## Interrupt count

### What is the interrupt count?

The interrupt count is the number of CA OLQ database requests allowed before execution is interrupted. When interruption occurs, you can choose to resume execution or to halt execution. This interruption allows you to monitor processing during retrieval.

When processing is interrupted, CA OLQ displays this message:

```
OLQ 098006 00 50 whole lines and 0 partial lines in report.
OLQ 098007 00 50 records read. 50 records selected.
OLQ 098009 00 Continue (yes/no)?
```

### Primary function:

The primary use for interrupt count is to specify the number of CA OLQ database requests after which CA OLQ interrupts the processing. This enables you to limit CA OLQ database requests.

### Additional use:

Another use for interrupt count is to determine the number of rows CA OLQ saves in a table before it performs a commit. (When CA OLQ performs a commit, database locks are released.)

![Note: Keep in mind that, when any problems occur with saving data, CA OLQ rolls back to the last commit. You could end up with a partial table. (This applies to both SQL and ASF tables.)](image)

### Setting interrupt:

You initially set the interrupt count during system generation with the ADD/MODIFY OLQ statement:

```
INTERRUPT COUNT IS nnn
MAXIMUM INTERRUPT COUNT IS nnn
```

These statements set the default interrupt count and the default maximum interrupt for CA OLQ. In this way, you can limit the number of database calls performed on each retrieval request. The system default set at installation is 100. You can, however, make this number as large or as small as you wish. To keep CA OLQ from performing commits while saving tables, set the interrupt count to zero.
A user can retrieve records without specifying an interrupt count; the interrupt count set during system generation acts as the default. If a user uses the SET INTERRUPT COUNT (in CA OLQ) to specify a count that is higher than the maximum set during system generation, the count defaults to the count set with the MAXIMUM INTERRUPT COUNT IS statement during system generation. To prevent CA OLQ from performing commits, set the interrupt count to zero. Note that even when the interrupt option is off, the interrupt count is still set.

Using db-keys for retrieval

CA OLQ can keep db-key lists to facilitate database retrieval following a FIND command. Since keeping a list of db-keys can use additional resources, it should be done only if OPTIONS=DBKEY has been specified during the CA OLQ session. The default option setting is OPTIONS=NODBKEY.

Controlling Resource Consumption

Controlling resource consumption involves:

- Allowing sorts
- Saving reports (size and retention period)
- Allowing use of db-key lists
- Allowing SAVE QFILES

Sorts

Sorting retrieved records is a resource-consuming operation. You can limit the ability to sort records to those users who need it.

You can control use of resources by controlling the use of sorts. There are two methods of sorting in CA OLQ:

- Using SORT after a GET/FIND command
- Using the ORDER BY clause of SELECT (OLQ access mode)

Using SORT after building a report with a GET/FIND command is costly. The ORDER BY clause of SELECT is more efficient because SELECT sorts the data while retrieving it.

When using SELECT (OLQ access mode), you can choose to disallow sorts altogether (this also disallows sorting with the ORDER BY clause of SELECT). To disallow sorting, use the IDD statement:

```
ADD/MODIFY USER ...
SORT IS NOT ALLOWED
```

Access mode: The SORT IS NOT ALLOWED clause is only valid when the access switch is set to OLQ.
Sort and the scratch area

CA OLQ sorts records in memory. The maximum amount of memory CA OLQ uses for any particular sort is determined by the SET MAXIMUM SORT SIZE system generation option. When sorting more records than fit into the maximum allowed, CA OLQ continues the sort by paging records in and out of scratch.

Specifying a large maximum sort size decreases the time a single sort takes and impacts other DC system users by:

- Using more storage
- Not releasing control to DC until the sort has completed

⚠️ **Note:** If you want to use your own operating system sort facility when running a batch job, set up the proper job control language as described in num=11.Batch Processing (see page 256).

Saved reports

You can set default values to keep a limit on the size and retention period of saved reports. You can set default values for specifications users are allowed to make when saving reports:

- **Report retention period** -- The amount of time a report remains saved if the user doesn't specify a retention period or specifies one less than the maximum. When the retention period expires, the report is automatically deleted.

- **Maximum retention period** -- The maximum amount of time a report remains saved if the user specifies a retention period greater than the default retention period mentioned above.

- **Report dictionary name** -- The name of the dictionary in which the catalog information is stored. CA OLQ assigns a catalog entry and passkeys to each saved report.

- **Report size** -- The page size and number of pages allowed per report.

- **Maximum number of reports** -- The maximum number of reports allowed per user. If the maximum is exceeded, a user must delete some reports in order to store new ones.

**Report retention period:**

If a user saves a report without specifying how many days it is to remain saved, the report is saved for the number of days specified by the report retention. This default is set during system generation with:

```
ADD/MODIFY OLQ ...
REPORT RETENTION IS retention-period
```

Keep in mind when using the batch facility, reports saved through batch do not honor the defaults set during system generation. You must specify these options explicitly when saving a batch report.
Maximum report retention:

You can define the maximum report retention (the maximum length of time in days) that a user can specify when saving a CA OLQ report in the queue area. If the user specifies a retention period greater than the default report retention period, the retention period defaults to this maximum. Use the system generation statement:

```
ADD/MODIFY OLQ ...
MAXIMUM REPORT RETENTION IS retention-period
```

Report dictionary name:

Since CA OLQ stores reports through the catalog, each saved report has a catalog entry and passkeys assigned. You can, during system generation, specify in which dictionary this catalog information is stored with:

```
ADD/MODIFY OLQ ...
REPORT DICTNAME IS dictionary-name
```

The specified report dictionary is also the dictionary in which CA OLQ looks to find the job control language needed for batch job submission.

Report size:

You can better estimate the size of the scratch and queue areas if you limit the size of CA OLQ reports retrieved.

To monitor the size of CA OLQ reports retrieved into scratch and saved into queue, you can specify the page size, the number of pages retrieved from the database, and the number of reports allowed per user.

Define the report page size during system generation with:

```
ADD/MODIFY OLQ ...
REPORT FILE PAGE SIZE IS report-file-page-size
```

The specified page size must be at least large enough to accommodate the largest database or logical record to be included in the report.

When a CA OLQ report exceeds the maximum report pages, as set with this option, system ends the retrieval process and stops writing report pages and issues the message:

```
OLQ 097004 00 Maximum report size has been exceeded
```

The incomplete report is retrieved into scratch, is available for viewing, and can be sorted.

Define the number of pages allowed per report during system generation with:

```
ADD/MODIFY OLQ ...
MAXIMUM REPORT PAGES IS max-report-pages
```

Define the number of reports allowed per user during system generation with:

```
ADD/MODIFY OLQ ...
MAXIMUM REPORT COUNT IS max-report-count
```
This way you can control the quantities of reports saved and better estimate the size of the queue area. Use the SELECT (OLQ access mode) statement to retrieve the records. With SELECT (OLQ access mode) you can sort and group the records while retrieving them.

**Db-key list**

You can control resources by limiting the use of db-key lists. Whenever the db-key option in CA OLQ is **ON**, CA OLQ builds a db-key list during retrieval. This means that for every record retrieved, CA OLQ stores a db-key in a list. This can be extremely costly in terms of resources if you perform large retrievals.

We recommend that you set the db-key option in CA OLQ **OFF**. If a user needs to build a db-key list for retrieval purposes, that user can turn the option back on for the time required.

**Saving qfiles**

In order to control resource consumption, you can limit the ability to save qfiles. This prevents users from creating redundant qfiles and also from cluttering up the dictionary with unnecessary qfiles.

To disallow the saving of qfiles, use the IDD DDDL statement:

```
ADD/MODIFY USER ...
QFILE SAVE IS NOT ALLOWED
```
Security

CA OLQ provides security options that enable you to govern the accessibility of the database.

- Assigning Authority to Access CA OLQ (see page 250)
- Limiting Access through Central Security (see page 250)
- Initiating CA OLQ Dictionary Security (see page 250)
- Securing User Access to CA OLQ Components (see page 251)
- Using LRF to Secure The Database (see page 253)
- Security for ASF tables (see page 254)
- Security for Saved Reports (see page 254)

When a security option is changed, CA OLQ enforces the change the next time the user signs on. CA OLQ uses the DC/UCF signon ID and password to determine what the current user can access.

Assigning Authority to Access CA OLQ

Access to CA OLQ can be limited by defining options in:

- Central Security -- Define the category to which task OLQ is associated
- Dictionary Security -- Define individual users that can access OLQ and valid functions can be defined uniquely for each user in each dictionary.

Limiting Access through Central Security

The centralized security administrator can assign the task OLQ to a category. Only users granted execution privilege in the category can access CA OLQ.

⚠️ Note: For more information on category and security, see your security administrator.

Initiating CA OLQ Dictionary Security

You can turn security for CA OLQ on and off with the IDD SET OPTIONS statement in the default dictionary. The full statement for initiating security is:

```
SET OPTIONS FOR DICT SECURITY FOR OLQ IS ON
```

This will allow only those USERs who are defined in the default dictionary to access OLQ.
Securing User Access to CA OLQ Components

You can specify additional CA OLQ security options for users in each dictionary through IDD with the statement:

```plaintext
SET OPTIONS FOR DICT SECURITY FOR OLQ IS ON
ADD/MODIFY/DELETE USER
```

The options associated with this statement are:

- **AUTHORITY FOR UPDATE IS OLQ**
- Access to subschemas
- Access to qfiles
- Access to menu mode
- Securing retrieval interruption
- Allow retrieval of multiple record occurrences
- Allow access to SORT

Subschema Access

CA OLQ uses the DC/UCF signon ID and password to determine which subschemas are available to the current user. You can assign individual users authority to access specific subschemas with the following clause of the IDD ADD USER statement:

```plaintext
ACCESS TO SUBSCHEMA subschema-name
```

This clause specifies that the user has access to the named subschema. Subschema security is enforced on a dictionary-by-dictionary basis.

It is with this statement that you also assign a **signon qfile** to a user. The signon qfile executes when a user signs on to the associated subschema. The clause of the ADD USER statement that associates a signon qfile is:

```plaintext
SIGNON QFILE IS qfile-name
```
Note: For more information about signon qfiles, see Special Uses of qfiles (see page 231).

For the complete syntax and syntax rules of the ADD USER statement, see the CA IDMS IDD Quick Reference Guide.

Qfile Access

You can apply security to qfiles in CA OLQ. By specifying restrictions on CA OLQ qfiles in IDD, you can:

- Allow/disallow a specific user to execute qfiles
- Allow/disallow a specific user to save qfiles
- Limit a user to accessing CA OLQ through qfiles only

These restrictions are defined in IDD, but take effect only if CA OLQ security is turned on in the primary dictionary, as described earlier in this chapter.

Sharing qfiles:

In CA OLQ, only the user who creates a qfile has access to that qfile. You can allow users to share qfiles through IDD. It is more efficient to share qfiles than to have duplicate qfiles for many users. Use the following statement to assign a user access to a particular qfile:

ADD/MODIFY/REPLACE QFILE ...
  INCLUDE USER user-name ...
  REGISTERED FOR PUBLIC ACCESS ...
  PUBLIC ACCESS IS ALL

Note: For more information about this statement, see the CA IDMS IDD Quick Reference Guide.

Executing qfiles:

You can allow and disallow qfile execution on a user-by-user basis through IDD with the statement:

ADD/MODIFY USER ...
  QFILE IS ALLOWED/NOT ALLOWED

Saving qfiles:

You can allow and disallow saving qfiles on a user-by-user basis through IDD with the statement:

ADD/MODIFY USER ...
  QFILE SAVE IS ALLOWED/NOT ALLOWED

Accessing through qfiles only:
You can limit a user's access to CA OLQ to only executing qfiles with the IDD statement:

ADD/MODIFY USER ...
QFILE IS ONLY

**Accessing menu mode:**

If your intention is to allow a user to only access CA OLQ through qfiles, you must explicitly deny that user access to CA OLQ menu mode. You disallow access to menu mode through IDD:

ADD/MODIFY USER ...
MENU MODE IS NOT ALLOWED

### Securing Retrieval Interruption

You can use IDD to secure interruption of processing in CA OLQ. IDD enables you to specify whether a user can choose **not** to interrupt data retrieval, or whether data retrieval is automatically interrupted after a certain number of records have been accessed.

You set this security in IDD, but it takes effect only if CA OLQ security is turned on in the primary dictionary, as described earlier in this chapter.

During system generation, you can specify a maximum interrupt count. This maximum provides a ceiling for the user-specified interrupt count.

⚠️ **Note:** For instructions on how to set these options, see num=12.Setting Defaults (see page 281).

### Using LRF to Secure The Database

You can use LRF to establish security for the database. By coding paths and logical records, you can dictate what a user is allowed to access through CA OLQ.

LRF enables you to:

- **Increase data integrity** -- You can write all database navigational instructions in the subschema. This enables you to predefined paths that dictate how a user accesses data through CA OLQ.

- **Secure data**
  - You can restrict the database record occurrences and fields viewed by the application program.
  - You can restrict the operations that the application program can perform on records and fields.
Security for ASF tables

CA OLQ secures ASF tables regardless of the status of CA OLQ security. To ensure table security, we recommend the following:

- IDs use the same passwords across all dictionaries
- The IDMSNWKQ and IDMSNWKS subschemas use the same page ranges

**Passkeys:**

CA OLQ has automatic security for ASF tables. When a user tries to access a table, CA OLQ uses catalog security to make sure the user has proper authority assigned through passkeys.

The table functions and the passkeys required by the Automatic System Facility (ASF) are presented in Table 10. Passkeys affect ASF tables and reports.

**Table 10. How Passkeys affect CA OLQ Table Processing**

<table>
<thead>
<tr>
<th>To Perform</th>
<th>User Default Passkey Required</th>
<th>Data Access Passkey Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>COPY</td>
<td>COPY</td>
</tr>
<tr>
<td>Create</td>
<td>CREATE</td>
<td>N/A</td>
</tr>
<tr>
<td>Append</td>
<td>ADD</td>
<td>ADD</td>
</tr>
<tr>
<td>Replace</td>
<td>ADD</td>
<td>ADD</td>
</tr>
<tr>
<td>Delete</td>
<td>ERASE</td>
<td>ERASE</td>
</tr>
<tr>
<td>Show tables</td>
<td>BROWSE</td>
<td>N/A</td>
</tr>
<tr>
<td>Show directory</td>
<td>BROWSE</td>
<td>N/A</td>
</tr>
</tbody>
</table>

A user default passkey is a passkey given to a user that allows the user to perform a specific function on any ASF table.

A data access passkey is a passkey associated with a specific table. It allows a specific user to perform a function on a specific table.

In addition to supporting passkey security, OLQ supports row-level security.

⚠️ **Note:** For more information about row-level security, see the CA IDMS Performance Monitor System Administration Guide.

Security for Saved Reports

**Saving a report:**
When you save a report in CA OLQ, information about that report is stored in the CA IDMS catalog. Passkeys are automatically assigned to the user for the report. If you want others to be able to access the report, you must go into CA IDMS and assign those users the appropriate passkeys. The appropriate passkeys and associated functions are presented in Table 10 (see page 254).
Batch Processing

**What batch does:**

The CA OLQ batch facility allows you to perform the same functions as command mode CA OLQ. Batch enables you to retrieve information from a CA IDMS/DB database and sequential files, and to build reports using that information. You can run the jobs offline and schedule them at times of the day when there are less demands on computer resources.

**Commands you can use in batch:**

CA OLQ batch supports the same set of processing commands as command mode CA OLQ except for the subset presented in Table 11.

**Table 11. Commands Not Supported in CA OLQ Batch**

<table>
<thead>
<tr>
<th>Environment</th>
<th>Invalid Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central version</td>
<td>SHOW TABLES</td>
</tr>
<tr>
<td>(CV CA OLQ batch)</td>
<td>SHOW DESTINATIONS</td>
</tr>
<tr>
<td></td>
<td>SUSPEND</td>
</tr>
<tr>
<td></td>
<td>SWAP</td>
</tr>
<tr>
<td></td>
<td>SWITCH</td>
</tr>
<tr>
<td></td>
<td>YES/NO</td>
</tr>
<tr>
<td>Local mode</td>
<td>SAVE REPORT</td>
</tr>
<tr>
<td>(Local CA OLQ batch)</td>
<td>DISPLAY REPORT</td>
</tr>
<tr>
<td></td>
<td>DELETE REPORT</td>
</tr>
<tr>
<td></td>
<td>SHOW DIRECTORY</td>
</tr>
<tr>
<td></td>
<td>SWAP</td>
</tr>
<tr>
<td></td>
<td>SWITCH</td>
</tr>
<tr>
<td></td>
<td>SUSPEND</td>
</tr>
<tr>
<td></td>
<td>YES/NO</td>
</tr>
</tbody>
</table>

**How batch works:**

With CA OLQ batch, you can:

- Read input commands one by one from a command file.
1. Retrieve information from one or more input files (sequential files or databases).

2. Build a report in the output file. This file can contain the data retrieved by the SELECT statement (OLQ mode).

**Creating a command file:**

A **command file** is a file that contains the CA OLQ commands to be executed during the batch job. You can store a command file in a sequential data set or list the commands in the job control language (JCL).

In batch, a line is the delimiter for entering commands. If a single command spans two or more lines, you must include continuation characters at the end of the continued lines to indicate to CA OLQ that the command continues on the next line. The default continuation character at system installation is the hyphen (-).

You must include a **SYSIPT** statement in your JCL. The SYSIPT statement defines the data set name of the file that contains the CA OLQ commands to be executed.

**Executing jobs:**

You can execute CA OLQ batch jobs either under the CA IDMS/DB central version or in local mode.

- **CV batch job streams** execute in the batch region and use the CA IDMS/DB central version for database access. To specify that your batch job is running under the central version, include a SYSCTL card in your z/OS JCL, or an IDMSOPTI card in your z/VSE and CMS JCL.

- **Local mode batch job streams** execute and handle database access within the batch region. Running a local mode batch job requires additional file assignments in your JCL as described later in this chapter.

**How to run a batch job:**

There are two main steps to using the CA OLQ Batch facility:

1. Run IDMSBCF -- IDMSBCF sets up the batch scratch area. This must be run before the CA OLQ local mode batch job step.

2. Run OLQBATCH -- The JCL for this job step is presented below.

**Note:** You can include the IDMSBCF statement in the JCL, but it must be run prior to the OLQBATCH job step.

To set up batch processing in CA OLQ, you:

- Tailor the job control language supplied in this chapter to your environment

- Set the batch class for batch job control language submission
Set default options normally set during system generation

JCL for z/OS and CMS Commands for OLQBATCH

The sections that follow contain sample JCL for the following operating systems running central version or local mode:

- z/OS
- CMS
- z/VSE

z/OS JCL (central version)

Here is sample z/OS JCL to execute CA OLQ batch jobs, when running central version:

```
//FORMAT EXEC PGM=IDMSBCF,REGION=1024K
//STEPLIB DD DSN=idsms.dbal.loadlib,DISP=SHR
// SYSLIB DD DSN=idsms.loadlib,DISP=SHR
//dms msg DD DSN=idsms.sysmsg.dms msg,DISP=SHR
//dms cr DD DSN=dsms.msg.dms msg,DISP=(NEW, PASS),
// UNIT=SYSDA,SPACE=(b b b, n n n n),
// DCD=(RECFM=F,LRECL=l l l l, BLKSIZE=b b b)
//SYSLST DD SYSLST=A
//SYSIDMS DD *
//DMCL=dmcl-namePut other SYSIDMS parameters, as appropriate, here/*
//SYSIPT DD *
//SYSLST DD SYSLOC.dms cr;
//*/
//*/
```

Note: The SORTWK01 through SYSOUT statements are optional. If these statements are included, CA OLQ uses the installation’s sort utility. When these are omitted, CA OLQ uses its own internal sort facility.
z/OS JCL (local mode)

Here is sample z/OS JCL to execute CA OLQ batch jobs in local mode:

CA OLQ Batch (local mode) (z/OS)

```
//FORMAT EXEC PGM=IDMSBCF,REGION=1024K
//STEPLIB DD DSN=idsms.db.loadlib,DISP=SHR
// DSN=idsms.loadlib,DISP=SHR
//dmmsg DD DSN=idsms.sysmsg.dmmsg,DISP=SHR
//dclscr DD DSN=&.dclscr.,DISP=(NEW,PASS),
// UNIT=SYSDA,SPACE=(bbbb,nnnn),
// DCB=(RECFM=F,LRECL=llll,BLKSIZE=cccc)
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-namePut other SYSIDMS parameters, as appropriate, here/*
//SYSIPT DD *
FORMAT FILE SYSLOC.dclscr;
/*
//*/
//OLQBatch EXEC PGM=OLQBatch,REGION=1024K
//STEPLIB DD DSN=idsms.db.loadlib,DISP=SHR
// DSN=idsms.loadlib,DISP=SHR
//dictdb DD DSN=idsms.appdict.ddldml,DISP=SHR
//dldo db DD DSN=idsms.appdict.ddldml,DISP=SHR
//dcmsg DD DSN=idsms.sysmsg.dmmsg,DISP=SHR
//userdb DD DSN=idsms.userdb,DISP=SHR
//dclscr DD DSN=&.dclscr.,DISP=(OLD,DELETE)
//sysjrnl DD DUMMY
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-namePut other SYSIDMS parameters, as appropriate, here/*
//SYSIPT DD *Put CA OLQ commands here/*
//*/

<table>
<thead>
<tr>
<th><strong>idms.db.loadlib</strong></th>
<th>Data set name of the load library containing the DMCL and database name table load modules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>idms.loadlib</strong></td>
<td>Data set name of the load library containing the CA IDMS executable modules</td>
</tr>
<tr>
<td><strong>dcmsg</strong></td>
<td>DDname of the system message (DDLDCMSG) area</td>
</tr>
<tr>
<td><strong>idms.sysmsg.dmmsg</strong></td>
<td>Data set name of the system message (DDLDCMSG) area</td>
</tr>
<tr>
<td><strong>dclscr</strong></td>
<td>DDname of the local mode system scratch (DDLOCSCR) area</td>
</tr>
</tbody>
</table>

**Note:** See z/OS Local Mode Considerations (see page 260), below, for more information.

| **&.dclscr.** | Temporary data set name of the local mode system scratch (DDLOCSCR) area |

**Note:** See z/OS Local Mode Considerations (see page 260), below, for more information.

<table>
<thead>
<tr>
<th><strong>bbbb</strong></th>
<th>Block size (page size)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nnnn</strong></td>
<td>Number of pages in the area</td>
</tr>
<tr>
<td><strong>llll</strong></td>
<td>Logical record length (should be the same as block size)</td>
</tr>
<tr>
<td><strong>dmcl-name</strong></td>
<td>Specifies the name of the DMCL load module</td>
</tr>
<tr>
<td><strong>dictdb</strong></td>
<td>DDname of the application dictionary definition area</td>
</tr>
</tbody>
</table>
idms.appldict.ddldml | Data set name of the application dictionary definition (DDLDML) area
---|---
dloddb | DDname of the application dictionary definition load area
idms.appldict.ddldclod | Data set name of the application dictionary definition load (DDLDCLOD) area
userdb | DDname of the user database file
user.userdb | Data set name of the user database file
sysjrnl | DDname of the tape journal file

**Note:** See z/OS Local Mode Considerations (see page 260), below, for more information.

sysctl | DDname of the SYSCTL file
idms.sysctl | Data set name of the SYSCTL file

**Note:** If both the system and local scratch areas are defined in the DMCL, CA IDMS only uses the local scratch area (SYSLOC.DDLOCSCR, dclscr).

### z/OS Local Mode Considerations

#### Scratch areas:

If both scratch areas are defined in the DMCL, CA IDMS only uses the local scratch area (SYSLOC.DDLOCSCR, dclscr). When running CA OLQ Batch with a DMCL that only defines the system scratch area (SYSTEM.DDLSCR), you must format this area with the following statement:

```
FORMAT FILE SYSTEM.dclscr
```

#### Accessing SQL Tables:

When accessing SQL tables, you must add the following statements:

```
//sqldd DD DSN=idms.syssql.ddlcata,DISP=SHR
//sqlxdd DD DSN=idms.syssql.ddlcatax,DISP=SHR
```

sqldd | DDname of the SQL catalog (DDLCAT) area
idms.syssql.ddlcata | Data set name of the SQL catalog (DDLCAT) area
sqlxdd | DDname of the SQL catalog index (DDLCATX) area
idms.syssql.ddlcatax | Data set name of the SQL catalog index (DDLCATX) area

#### When Creating Tables:

When using CA OLQ Batch to create tables, change:

```
//sysjrnl DD DUMMY
```
To:

//sysjrnl DD DSN=idms.tapejrnl,DISP=(NEW,CATLG),UNIT=tape

Where:

<table>
<thead>
<tr>
<th>DDname</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysjrnl</td>
<td>DDname of the tape journal file</td>
</tr>
<tr>
<td>idms.tapejrnl</td>
<td>Data set name of the tape journal file</td>
</tr>
<tr>
<td>tape</td>
<td>Symbolic device name for tape file</td>
</tr>
</tbody>
</table>

**CMS commands (central version)**

Here are sample CMS commands to execute CA OLQ batch jobs, when running central version:

```markdown
CA OLQ Batch (central version) (CMS)
FILEDEF DCLSCR DISK scratch file a (RECFM F LRECL 111 BLKSIZE bbb)FILEDEF SYSIDMS DISK sysidms parms a (RECFM F LRECL 111 BLKSIZE bbb)FILEDEF SYSIPT DISK sysipt input a (RECFM F LRECL 111 BLKSIZE bbb)EXEC IDMSFD
FILEDEF userdb DISK user userdb a (RECFM FB LRECL ppp BLKSIZE ppp)FILEDEF SYSIDMS DISK sysidms parms a (RECFM F LRECL 111 BLKSIZE bbb)FILEDEF SYSIPT DISK sysipt input a (RECFM F LRECL 111 BLKSIZE bbb)OSRUN IDMSBCF
```

<table>
<thead>
<tr>
<th>scratch file a</th>
<th>File ID of the temporary scratch file for the local mode system scratch (DDLOCSCR) area</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Logical record length of the user input data file</td>
</tr>
<tr>
<td>bbb</td>
<td>Block size of the user input data file</td>
</tr>
<tr>
<td>sysidms parms a</td>
<td>File ID of the file containing SYSIDMS parameters</td>
</tr>
<tr>
<td>sysipt input a</td>
<td>File ID of the file containing the IDMSBCF or OLQBATCH input parameters</td>
</tr>
<tr>
<td>IDMSFD</td>
<td>Exec which defines all FILEDEFs, TXTLIBs, and LOADLIBs required by the system</td>
</tr>
<tr>
<td>userdb</td>
<td>DDname of the user database file</td>
</tr>
<tr>
<td>user userdb a</td>
<td>File ID of the user database file</td>
</tr>
<tr>
<td>ppp</td>
<td>Page size of the user database file</td>
</tr>
</tbody>
</table>

**Usage**

**SYSIDMS file:**

To run IDMSBCF or OLQBATCH, you should include these SYSIDMS parameters:

- `DMCL=dmcl-name`, to identify the DMCL
- `DBNAME=dictionary-name`, to identify the dictionary whose catalog component contains the database definitions
How to create the SYSIDMS file:

To create the SYSIDMS file of SYSIDMS parameters:

1. On the CMS command line, type:
   
   XEDIT sysidms parms a (NOPROF

2. Press [Enter]

3. On the XEDIT command line, type:
   
   INPUT

4. Press [Enter]

5. In input mode, type in the SYSIDMS parameters

6. Press [Enter] to exit input mode

7. On the XEDIT command line, type:
   
   FILE

8. Press [Enter]

⚠️ Note: For documentation of SYSIDMS parameters, see the CA IDMS Database Administration Guide.

How to create the SYSIPT file:

To create the SYSIPT file of IDMSBCF and OLQBATCH input parameters:

1. On the CMS command line, type:
   
   XEDIT sysipt input a (NOPROF

2. Press [Enter]

3. On the XEDIT command line, type:
   
   INPUT

4. Press [Enter]

5. In input mode, type in the IDMSBCF or OLQBATCH input parameters

6. Press [Enter] to exit input mode

7. On the XEDIT command line, type:
   
   FILE
8. Press [Enter]

**DCLSCR scratch file:**

DCLSCR is a scratch file, which resides on a temporary mini disk. To allocate this disk, enter the following commands:

```
DEF t3380 cuu CYL nnn FORMAT cuu fm
```

<table>
<thead>
<tr>
<th>t3380</th>
<th>Disk device type</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuu</td>
<td>Virtual address of the temporary minidisk</td>
</tr>
<tr>
<td>nnn</td>
<td>Space allocated in CYLS</td>
</tr>
<tr>
<td>fm</td>
<td>Filemode for the temporary minidisk</td>
</tr>
</tbody>
</table>

**CMS commands (local mode)**

To specify that OLQBATCH is executing in local mode, perform one of the following:

- Link OLQBATCH with an IDMSOPTI program that specifies local execution mode
- Specify *LOCAL* as the first input parameter of the filename, type and mode identified by `sysipt2` in the OLQBATCH exec.
- Modify the OSRUN statement:
  ```
  OSRUN OLQBATCH PARM='*LOCAL*'
  ```

**Note:** This option is valid only if the OSRUN command is issued from a System Product interpreter or an EXEC2 file.

**z/VSE JCL (central version)**

Here is sample z/VSE JCL to execute CA OLQ batch jobs, when running central version:

```
// JOB  OLBATCH
// DLBL idmslib,'idmslib.library'// EXTENT sysxxx,vvvvv,,,ssss,tttt// ASSGN sysxxx
// DISK,VOL=vvvvv,SHR
// LIBDEF /*,SEARCH=CA IDMS load libraries// EXEC PROC=IDMSLBLS
// EXEC IDMSBCF,SIZE=1048K
// DMCL=dmcl-name Put other SYSIDMS parameters, as appropriate, here/
// FORMAT FILE SYSLOC.dclscr;
// *
// EXEC PROC=IDMSLBLS
// EXEC PROC=sysctl// EXEC OLBATCH,SIZE=1048K
// DMCL=dmcl-name Put other SYSIDMS parameters, as appropriate, here/
// Put CA OLQ commands here/
```

`idmslib` Dtfname of the CA IDMS library
### 'idmslib.library'
Data set name of CA IDMS load libraries, as established during installation

<table>
<thead>
<tr>
<th>sysxxx</th>
<th>SYS number</th>
</tr>
</thead>
<tbody>
<tr>
<td>vvvvvv</td>
<td>Volume serial number</td>
</tr>
<tr>
<td>ssss</td>
<td>Starting extent</td>
</tr>
<tr>
<td>tttt</td>
<td>Number of tracts</td>
</tr>
</tbody>
</table>

### CA IDMS load libraries
The CA IDMS load libraries, as established during installation

<table>
<thead>
<tr>
<th>dmcl-name</th>
<th>Name of the DMCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>dclscr</td>
<td>Dtfname of the local mode system scratch (DDLOCSCR) area</td>
</tr>
<tr>
<td>sysctl</td>
<td>Dtfname of the SYSCTL file</td>
</tr>
</tbody>
</table>

⚠️ **Note:** For more information about the IDMSLBLS procedure, see **IDMSLBLS Procedure (see page 265)**, later in this chapter.

---

### z/VSE JCL (local mode)

Here is sample z/VSE JCL to execute CA OLQ batch jobs in local mode:

```jcl
// JOB OLQLOCAL
// DLBL idmslib,'idmslib.library'// EXTENT sysxxx,vvvvvv,,,ssss,tttt// ASSGN sysxxx
,DISK,VOL=vvvvvv,SHR
// LIBDEF *,SEARCH=CA IDMS load libraries// EXEC PROC=IDMSLBLS
// EXEC IDMSBCF,SIZE=1024K
DMCL=dmcl-name * Put other SYSIDMS parameters, as appropriate, here/*
FORMAT FILE SYSLOC.dclscr;
/*
// EXEC PROC=IDMSLBLS
// ASSGN sysxxx,DISK,VOL=vvvvvv,SHR
// DLBL userdb.'user.userdb'
// EXTENT sysxxx,vvvvvv,,,tttt,lill// EXEC OLQBATCH,SIZE=1024K
DMCL=dmcl-name * Put other SYSIDMS parameters, as appropriate, here/*
Put CA OLQ commands here/*
```

<table>
<thead>
<tr>
<th>idmslib</th>
<th>Dtfname of the CA IDMS library</th>
</tr>
</thead>
<tbody>
<tr>
<td>'idmslib.library'</td>
<td>Data set name of CA IDMS load libraries, as established during installation</td>
</tr>
<tr>
<td>sysxxx</td>
<td>SYS number</td>
</tr>
<tr>
<td>vvvvvv</td>
<td>Volume serial number</td>
</tr>
<tr>
<td>ssss</td>
<td>Starting extent</td>
</tr>
<tr>
<td>tttt</td>
<td>Number of tracts</td>
</tr>
</tbody>
</table>

### CA IDMS load libraries
The CA IDMS load libraries, as established during installation

<table>
<thead>
<tr>
<th>dmcl-name</th>
<th>Name of the DMCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>dclscr</td>
<td>Dtfname of the local mode system scratch (DDLOCSCR) area</td>
</tr>
</tbody>
</table>
IDMSLBLS Procedure

What is the IDMSLBLS procedure?

IDMSLBLS is a procedure provided during a CA IDMS z/VSE installation. It contains file definitions for the CA IDMS components listed below. These components are provided during installation:

- Dictionaries
- Sample databases
- Disk journal files
- SYSIDMS file

Tailor the IDMSLBLS procedure to reflect the filenames and definitions in use at your site and include this procedure in z/VSE JCL job streams.

The sample z/VSE JCL provided in this document includes the IDMSLBLS procedure. Therefore, individual file definitions for CA IDMS dictionaries, sample databases, disk journal files, and SYSIDMS file are not included in the sample JCL.

IDMSLBLS procedure listing
// EXTENT SYSnnn,nnnnnn,,ssss,68
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL dcsr,' idms.system.ddldcscr ',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,135
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL dcmsg,' idms.sysmsg.ddldcmsg ',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,281
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL dclscr,'idms.sysloc.ddldcscr',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL dirldb,'idms.sysdirl.ddldml',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL dirllod,'idms.sysdirl.ddldclod',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL empldem,'idms.sqldemo.empldemo',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,11
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL insdemo,'idms.insdemo1',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL infodem,'idms.sqldemo.infodemo',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL orgdemo,'idms.orgdemo1',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL projdem,'idms.projseg.projdemo',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL indxdem,'idms.sqldemo.indxdemo',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,6
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL secdd,'idms.sysuser.ddlsec',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,26
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL dldodb,'idms.appldict.ddldml',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,51
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL sqldd,'idms.syssql.ddlcat',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,101
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL sqlld,'idms.syssql.ddlcatl',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,51
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL sqlxdd,'idms.syssql.ddlcatx',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,26
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL asfdml,'idms.asfdict.ddldml',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,201
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL asflod,'idms.asfdict.asflod',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,401
// ASSIGN SYSnnn,DISK, VOL=nnnnnn, SHR
// DLBL asfdata,'idms.asfdict.asfdata',1999/365, DA
// EXTENT SYSnnn,nnnnnn,,ssss,101
// DLBL  j1jrnl,'idms.j1jrnl',1999/365,DA
// EXTENT SYSn,,nnnnnn,,ssss,54
// ASSGN  SYSn,DISK, VOL=nnnnnnn, SHR
// DLBL  j2jrnl,'idms.j2jrnl',1999/365,DA
// EXTENT SYSn,,nnnnnn,,ssss,54
// ASSGN  SYSn,DISK, VOL=nnnnnnn, SHR
// DLBL  j3jrnl,'idms.j3jrnl',1999/365,DA
// EXTENT SYSn,,nnnnnn,,ssss,54
// ASSGN  SYSn,DISK, VOL=nnnnnnn, SHR
// DLBL  SYSIDMS,'#SYSIPT',0,SD
/+ */

<table>
<thead>
<tr>
<th>dccat</th>
<th>Filename of the system dictionary catalog (DDLCAT) area</th>
</tr>
</thead>
<tbody>
<tr>
<td>idms.system.dccat</td>
<td>File-ID of the system dictionary catalog (DDLCAT) area</td>
</tr>
<tr>
<td>SYSnnn</td>
<td>Logical unit of the volume for which the extent is effective</td>
</tr>
<tr>
<td>nnnnnn</td>
<td>Volume serial identifier of appropriate disk volume</td>
</tr>
<tr>
<td>ssss</td>
<td>Starting track (CKD) or block (FBA) of disk extent</td>
</tr>
<tr>
<td>dccatl</td>
<td>Filename of the system dictionary catalog load (DDLCATLOD) area</td>
</tr>
<tr>
<td>idms.system.dccatlod</td>
<td>File-ID of the system dictionary catalog load (DDLCATLOD) area</td>
</tr>
<tr>
<td>dccatx</td>
<td>Filename of the system dictionary catalog index (DDLCATX) area</td>
</tr>
<tr>
<td>idms.system.dccatx</td>
<td>File-ID of the system dictionary catalog index (DDLCATX) area</td>
</tr>
<tr>
<td>dcdml</td>
<td>Filename of the system dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>idms.system.ddldml</td>
<td>File-ID of the system dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>dclod</td>
<td>Filename of the system dictionary definition load (DDLDCLOD) area</td>
</tr>
<tr>
<td>idms.system.ddldclod</td>
<td>File-ID of the system dictionary definition load (DDLDCLOD) area</td>
</tr>
<tr>
<td>dclog</td>
<td>Filename of the system log area (DDLDCLOD) area</td>
</tr>
<tr>
<td>idms.system.ddldclog</td>
<td>File-ID of the system log (DDLDCLOD) area</td>
</tr>
<tr>
<td>dcrun</td>
<td>Filename of the system queue (DDLDCRUN) area</td>
</tr>
<tr>
<td>idms.system.ddldcrun</td>
<td>File-ID of the system queue (DDLDCRUN) area</td>
</tr>
<tr>
<td>dcscr</td>
<td>Filename of the system scratch (DDLDSCR) area</td>
</tr>
<tr>
<td>idms.system.ddldscr</td>
<td>File-ID of the system scratch (DDLDSCR) area</td>
</tr>
<tr>
<td>dcmgs</td>
<td>Filename of the system message (DDLDCMSG) area</td>
</tr>
<tr>
<td>idms.systmsd.ddldcmgs</td>
<td>File-ID of the system message (DDLDCMSG) area</td>
</tr>
<tr>
<td>dclscr</td>
<td>Filename of the local mode system scratch (DDLOCSCR) area</td>
</tr>
<tr>
<td>idms.sysloc.ddlocscr</td>
<td>File-ID of the local mode system scratch (DDLOCSCR) area</td>
</tr>
<tr>
<td>dirldb</td>
<td>Filename of the IDMSDIRL definition (DDLDML) area</td>
</tr>
<tr>
<td>idms.syssdirl.ddldml</td>
<td>File-ID of the IDMSDIRL definition (DDLDML) area</td>
</tr>
<tr>
<td>dirldod</td>
<td>Filename of the IDMSDIRL definition load (DDLDCLOD) area</td>
</tr>
<tr>
<td>idms.sysdirl.dirldod</td>
<td>File-ID of the IDMSDIRL definition load (DDLDCLOD) area</td>
</tr>
<tr>
<td>empdemo</td>
<td>Filename of the EMPDEMO area</td>
</tr>
<tr>
<td>idms.empdemo1</td>
<td>File-ID of the EMPDEMO area</td>
</tr>
<tr>
<td>insdemo</td>
<td>Filename of the INSDEMO area</td>
</tr>
<tr>
<td>File-ID or Filename</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>idms.insdemo1</td>
<td>File-ID of the INSDEMO area</td>
</tr>
<tr>
<td>orgdemo</td>
<td>Filename of the ORGDEMO area</td>
</tr>
<tr>
<td>idms.orgdemo1</td>
<td>File-ID of the ORDDEMO area</td>
</tr>
<tr>
<td>empldem</td>
<td>Filename of the EMPLDEMO area</td>
</tr>
<tr>
<td>idms.sqldemo.empldemo</td>
<td>File-ID of the EMPLDEMO area</td>
</tr>
<tr>
<td>infodem</td>
<td>Filename of the INFODEMO area</td>
</tr>
<tr>
<td>idms.sqldemo.infodemo</td>
<td>File-ID of the INFODEMO area</td>
</tr>
<tr>
<td>projdem</td>
<td>Filename of the PROJDEMO area</td>
</tr>
<tr>
<td>idms.projseg.projdemo</td>
<td>File-ID of the PROJDEMO area</td>
</tr>
<tr>
<td>indxdem</td>
<td>Filename of the INDXDEMO area</td>
</tr>
<tr>
<td>idms.sqldemo.indxdemo</td>
<td>File-ID of the INDXDEMO area</td>
</tr>
<tr>
<td>sysctl</td>
<td>Filename of the SYSCTL file</td>
</tr>
<tr>
<td>idms.sysctl</td>
<td>File-ID of the SYSCTL file</td>
</tr>
<tr>
<td>secdd</td>
<td>Filename of the system user catalog (DDLSEC) area</td>
</tr>
<tr>
<td>idms.sysuser.ddlsec</td>
<td>File-ID of the system user catalog (DDLSEC) area</td>
</tr>
<tr>
<td>dictdb</td>
<td>Filename of the application dictionary definition area</td>
</tr>
<tr>
<td>idms.appldict.ddldml</td>
<td>File-ID of the application dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>dloddb</td>
<td>Filename of the application dictionary definition load area</td>
</tr>
<tr>
<td>idms.appldict.ddldclod</td>
<td>File-ID of the application dictionary definition load (DDLDCLOD) area</td>
</tr>
<tr>
<td>sqldd</td>
<td>Filename of the SQL catalog (DDLCAT) area</td>
</tr>
<tr>
<td>idms.syssql.ddlcat</td>
<td>File-ID of the SQL catalog (DDLCAT) area</td>
</tr>
<tr>
<td>sqllod</td>
<td>Filename of the SQL catalog load (DDLCATL) area</td>
</tr>
<tr>
<td>idms.syssql.ddlcatl</td>
<td>File-ID of SQL catalog load (DDLCATL) area</td>
</tr>
<tr>
<td>sqlxd</td>
<td>Filename of the SQL catalog index (DDLDCATX) area</td>
</tr>
<tr>
<td>idms.syssql.ddlcax</td>
<td>File-ID of the SQL catalog index (DDLDCATX) area</td>
</tr>
<tr>
<td>asfdml</td>
<td>Filename of the asf dictionary definition (DDLDMML) area</td>
</tr>
<tr>
<td>idms.asfdict.ddldml</td>
<td>File-ID of the asf dictionary definition (DDLDML) area</td>
</tr>
<tr>
<td>asflod</td>
<td>Filename of the asf dictionary definition load (ASFLOD) area</td>
</tr>
<tr>
<td>idms.asfdict.asflod</td>
<td>File-ID of the asf dictionary definition load (ASFLOD) area</td>
</tr>
<tr>
<td>asfdata</td>
<td>Filename of the asf data (ASFDATA) area</td>
</tr>
<tr>
<td>idms.asfdict.asfdata</td>
<td>File-ID of the asf data area (ASFDATA) area</td>
</tr>
<tr>
<td>ASFDEFN</td>
<td>Filename of the asf data definition (ASFDEFN) area</td>
</tr>
<tr>
<td>idms.asfdict.asfdefn</td>
<td>File-ID of the asf data definition area (ASFDEFN) area</td>
</tr>
<tr>
<td>j1jrnl</td>
<td>Filename of the first disk journal file</td>
</tr>
<tr>
<td>idms.j1jrl</td>
<td>File-ID of the first disk journal file</td>
</tr>
<tr>
<td>j2jrl</td>
<td>Filename of the second disk journal file</td>
</tr>
</tbody>
</table>
### Setting Defaults for Batch Processing

Since CA OLQ batch processing does not have access to the defaults set during system generation, you must explicitly define these defaults in batch.

Table 12 presents the options available, the batch defaults, and the syntax necessary to override the defaults.

**Table 12. Batch Options, Defaults, and Syntax**

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Override Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print line size</td>
<td>80</td>
<td>SET PRINT LINE SIZE</td>
</tr>
<tr>
<td>Print line count</td>
<td>60</td>
<td>SET PRINT LINE COUNT</td>
</tr>
<tr>
<td>Internal storage page size</td>
<td>1920</td>
<td>SET USER ... INTERNAL STORAGE PAGE SIZE</td>
</tr>
<tr>
<td>Report page size</td>
<td>4000</td>
<td>SET USER ... REPORT PAGE SIZE</td>
</tr>
<tr>
<td>Menu mode</td>
<td>Disallowed</td>
<td>N/A</td>
</tr>
<tr>
<td>Continuation character</td>
<td>-</td>
<td>SET CONTINUATION CHARACTER</td>
</tr>
<tr>
<td>Separator character</td>
<td>!</td>
<td>SET SEPARATOR CHARACTER</td>
</tr>
<tr>
<td>Comment character</td>
<td>;</td>
<td>SET COMMENT CHARACTER</td>
</tr>
<tr>
<td>Report retention</td>
<td>1</td>
<td>SAVE REPORT ... RETENTION PERIOD</td>
</tr>
<tr>
<td>Maximum report retention</td>
<td>32767</td>
<td>N/A</td>
</tr>
<tr>
<td>Maximum report pages</td>
<td>32767</td>
<td>N/A</td>
</tr>
<tr>
<td>Terminal line size</td>
<td>132</td>
<td>SET LINE SIZE</td>
</tr>
<tr>
<td>Terminal line count</td>
<td>60</td>
<td>SET LINE COUNT</td>
</tr>
<tr>
<td>Report dictionary database name</td>
<td>blanks</td>
<td>SET REPORT DICTNAME SET USER ... REPORT DICTNAME</td>
</tr>
<tr>
<td>Maximum sort space in K bytes</td>
<td>384</td>
<td>SET MAXIMUM SORT SIZE=</td>
</tr>
</tbody>
</table>

### Defining files

CA OLQ batch retrieves records from the file you specify in the JCL. (If retrieving only from the database, you do not have to specify an input file in the JCL.) The access switch must be set to `olq` to read or write sequential files.
You can define the characteristics of each input file the CA OLQ batch job stream accesses by using the **DEFINE FILE** statement. DEFINE FILE associates a file name and file characteristics with IDD record and element entities.

**Example:**

If your batch job accesses an input file called **INFILE**, you would include the following DEFINE FILE statement in your input data stream:

```
DEFINE FILE INFILE RECORD EMPL-2 (3) DICT TESTDICT
RECFM=F BLKSIZE=80 LRECL=80
```

- The **INFILE** file is used to access input data from the EMPL-2 record.
- The record definition for the **EMPL-2** record is stored in the **TESTDICT** dictionary.
- The number in parenthesis, (3), is the version number. If no version number is specified, the default is 1.

When using DEFINE FILE, the corresponding job control language statement must name the file in the DD name:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>JCL Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>//INFILE DD DSN= infile, DISP= SHR</td>
</tr>
<tr>
<td>z/VSE</td>
<td>// DLBL INFILE, 'infile'</td>
</tr>
<tr>
<td></td>
<td>// EXTENT sysnnn, nnnnnn,sسس,زززز</td>
</tr>
<tr>
<td></td>
<td>// ASSIGN sysnnn, DISK, VOL= nnnnnnnn, SHR</td>
</tr>
<tr>
<td>CMS</td>
<td>FILEDEF INFILE DISK filename filetype filemode (RECFM=F BLKSIZE= nn LRECL= nn)</td>
</tr>
</tbody>
</table>

**Defining input files:**

You can also use DEFINE FILE to define the following characteristics for each input file:

- Record format (required for z/VSE)
- Block size (required for z/VSE)
- Logical record length (required for z/VSE)
- File type
- Device type
- Logical unit (z/VSE only)

**Defining output files:**

The output file is the file the report data is written to. You use the DEFINE FILE statement to direct the output to a file:
DEFINE FILE OUTFILE OUTPUT

This command marks the file OUTFILE as containing the report output.

The data set is written out in its unformatted condition to the output file.

If you want to use the output file as an input file, you must do so in a subsequent job step.

Defining sort work files:

During SORT requests and SELECT with GROUP BY requests, CA OLQ attempts to use the operating system sort utility. If you want CA OLQ to use the operating system sort, you must define appropriate sort work files in the batch job stream. If these sort work files are not in the job stream, CA OLQ uses its internal sort routines.

⚠️ Note: CA OLQ always attempts the operating system sort. If the sort attempt fails, CA OLQ switches to the internal sort. An operating system sort utility is required for CMS users. If no operating sort utility is available, contact CA Technical Support for assistance.

Signing on in batch

The SET command in batch allows you to identify:

- The user, maximum internal page size, report file page size, and report dictionary -- SET USER
- The maximum sort size -- SET MAXIMUM SORT SIZE
- The report dictionary name -- SET REPORT DICTNAME

Specifying user name:

The SET USER statement must be the first statement in your input data stream.

For online batch submission, you can use a substitution string in the batch job control language to specify the user ID. This allows you to share JCL between users without changing the user ID in the JCL. Online CA OLQ performs the substitution during online submission.

The string $USER is changed to the current user ID. This is a straight string substitution. For example:
//$USERBAT JOB . . .

When XYZ is signed on and submits a CA OLQ batch job, this statement becomes:
//XYZBAT JOB . . .
OLQBatch Notification

You can use OLQBNOTE to notify a DC user that a batch job has completed. If the specified user is not signed on at the time, the notification will be discarded. OLQBNOTE runs central version and accepts one or more control cards of the format:

```
USER= user-id,MESSAGE= message,BEEP
```

- **MESSAGE** can be shortened to **MSG** or **MES**.
- **USER** can be shortened to **USE**.
- **BEEP** sounds the terminal alarm, is optional, and should not be used against a non-3270 terminal.
- **Message** should be enclosed in quotation marks (" ") if it contains blanks and double quotation marks (" " ) if it contains single quotation marks.

For online batch submission, the user ID may be specified as $USER. CA OLQ substitutes the DC user ID in its place upon job submission.

Setting up notification:

To use the notification facility, you must define the following during **system generation**:

- **OLQQNOTE** as a queue
- **OLQTNOTE** as a task
- **OLQSNOTE** as a program

OLQBNOTE Example for z/OS

```
z/OS JCL:

OLQBNOTE (z/OS)

//NOTIFY EXEC PGM=OLQBNOTE,REGION=1024K
//STEPLIB DD DSN=idms.dba.loadlib,DISP=SHR
//   DD DSN=idms.loadlib,DISP=SHR
//sysctl DD DSN=idms.sysctl,DISP=SHR
//dmsg DD DSN=idms.sysmsg.ddldcmsg,DISP=SHR
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DMCL=dmcl-namePut other SYSIDMS parameters, as appropriate, here/*
//SYSIPT DD *
USER=$USER MESSAGE='OLQ Batch completed'
/*
```

<table>
<thead>
<tr>
<th>idms.dba.loadlib</th>
<th>Data set name of the load library containing the DMCL and database name table load modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>idms.loadlib</td>
<td>Data set name of the load library containing the CA IDMS executable modules</td>
</tr>
</tbody>
</table>
CA IDMS Reference - 19.0

<table>
<thead>
<tr>
<th>sysctl</th>
<th>DDname of the SYSCTL file</th>
</tr>
</thead>
<tbody>
<tr>
<td>idms.sysctl</td>
<td>Data set name of the SYSCTL file</td>
</tr>
<tr>
<td>dcmg</td>
<td>DDname of the system message (DDLDCMSG) area</td>
</tr>
<tr>
<td>idms.sysmsg.</td>
<td>Data set name of the system message (DDLDCMSG) area</td>
</tr>
<tr>
<td>ddldcmsg</td>
<td></td>
</tr>
<tr>
<td>dmcl-name</td>
<td>Specifies the name of the DMCL load module</td>
</tr>
</tbody>
</table>

⚠️ **Note:** $USER only works when OLQBNOTE is submitted from CA OLQ.

## OLQBNOTE example for CMS

### CMS commands:

**OLQBNOTE (CMS)**

```
FILEDEF SYSDMS DISK sysidms parms a
FILEDEF SYSIPT DISK olqbnote input a (RECFM F LREC L lll BLKSIZE bbb)
EXEC IDMSFD
OSRUN OLQBNOTE
```

- **sysidms parms a**  File ID of the file containing SYSDMS parameters
- **olqbnote input a**  File ID of the file containing the OLQBNOTE input parameters
- **lll**  Logical record length of the user input data file
- **bbb**  Block size of the user input data file
- **IDMSFD**  Exec which defines all FILEDEFs, TXTLIBs, and LOADLIBs required by the system

### SYSDMS file:

To run OLQBNOTE, you should include these SYSDMS parameters:

- **DMCL=dmcl-name**, to identify the DMCL load module
- **Any other SYSDMS parameters**

### How to create the SYSDMS file:

To create the SYSDMS file of SYSDMS parameters:

1. On the CMS command line, type:
   
   ```
   XEDIT sysidms parms a (NOPROF
   ```
2. Press [Enter]
3. On the XEDIT command line, type:
   
   ```
   INPUT
   ```
4. Press [Enter]
5. In input mode, type in the SYSIDMS parameters
6. Press [Enter] to exit input mode
7. On the XEDIT command line, type:
   FILE
8. Press [Enter]

⚠️ **Note:** For documentation of SYSIDMS parameters, see the *CA IDMS Database Administration Guide.*

**SYSIPT file:**

To include SYSIPT, you should add this statement:

```
USER=$USER MESSAGE='OLQ Batch completed'
```

To create the SYSIPT file of OLQBNOTE input parameters:

1. On the CMS command line, type:
   ```
   XEDIT sysipt input a (NOPROF)
   ```
2. Press [Enter]
3. On the XEDIT command line, type:
   ```
   INPUT
   ```
4. Press [Enter]
5. In input mode, type in the OLQBNOTE input parameters
6. Press [Enter] to exit input mode
7. On the XEDIT command line, type:
   ```
   FILE
   ```
8. Press [Enter]

**OLQBNOTE example for z/VSE**

**z/VSE JCL:**

```
OLQBNOTE (z/VSE)
```
// JOB  OLQBNOTE
// DLBL  idmslib,'idmslib.library'// EXTENT sysxxx,vv,vv,ss,ttt// ASSGN sysxxx
// DISK,VOL=vvvvv,SHR
// LIBDEF *,SEARCH=CA IDMS load libraries// EXEC PROC=IDMSLBLS
// EXEC PROC=sysctl// EXEC OLQBNOTE,SIZE=1024K
// USER=$USER MESSAGE='OLQ Batch completed'
/*

* 

idmslib Dtfname of the CA IDMS library

idmslib.library Data set name of CA IDMS load libraries, as established during installation

sysxxx SYS number

vvvvv Volume serial number

ssss Starting extent

tttt Number of tracts

CA-IDMS load libraries The CA IDMS load libraries, as established during installation

sysctl Procedure name containing the SYSCTL file

dmcl-name Name of the DMCL

Batch Class Specification

The CA OLQ system generation BATCH CLASS must be non zero to enable CA OLQ batch submission from online.

As with all users exits, you should ensure the exit meets your site's standards.

⚠️ Note: See the CA IDMS System Operations Guide, for a discussion on installing user exits.

Except for the z/OS environment, user exits are required to submit batch jobs. Exits provided are:

- RHDCUX26 for z/VSE
- RHDCUX21 for CMS

For the z/OS environment, turn off "banner" output for the batch print class.

Operating System Dependent Installation Instructions

To run CA OLQ batch jobs efficiently, tailor the job control language according to your operating system.
z/OS:

1. Change CA OLQ system generation parameter BATCH CLASS to a number between 1 and 64.

2. Create a DC printer defined to go to the JESRDR (or HASPRDR) for the print class specified above. For example:
   
   //JESRDR DD SYSOUT=(A,INTRDR),
   DCB=(RECFM=F,LRECL=80,BLKSIZE=80)

3. Ensure that the last card in the IDD JCL module(s) is:
   
   /*EOF

   If this is not done, submitted jobs will remain on the reader until another job is submitted.

CMS:

1. Change CA OLQ system generation parameter BATCH CLASS to a number between 1 and 64.

2. Modify RHDCUX21 (source on tape) and assemble:
   
   a. Change BATCLASS to same value as above.
   b. Change WKUSER to desired z/VM batch machine.

3. Modify RHDCUXIT to add user exit 21:
   
   a. Add #DEFXIT MODE=SYSTEM,CALL=DC,NAME=RHDCUX21

4. The FILEDEF corresponding to the printer definition for the BATCH CLASS should go to PUNCH and not PRINT. For example:
   
   FILEDEF SYSPRT PUNCH (RECFM F LRECL 80 BLKSIZE 80.

5. Ensure that the PTE associated with the print class has the batch print class value as the only allowable print class.

   **Note:** When running CA OLQ directly on CMS (that is, not under DC), if you intend to do any CA OLQ request that require a sort, you must have an external sort available at run time.

z/VSE:

1. Change CA OLQ system generation parameter BATCH CLASS to a number between 1 and 64. CA OLQ does not use this value other than to determine if job submission is enabled in z/VSE.

2. Modify RHDCUXIT to add user exit 26:
   
   Add #DEFXIT MODE=USER,CALL=IBM,NAME=RHDCUD26

3. Ensure RHDCU26B is defined to your DC system as:
ASSEMBLER NONREENTRANT NOPROTECT

4. Ensure that the last card in the JCL module(s) is:

/*EOF
The CA provided exit requires this to detect the end of the job stream. This card is not submitted to power.

Examples of Batch

Displaying reports:

This example displays those employees whose employee ID numbers are less than or equal to 40.

Input:
SET USER ABC PASS ABCPASS
SET ACCESS OLO
SIGNON DICT=TESTDICT SS=EMPSS01 DBN=EMPDEMO
SELECT * FROM EMPLOYEE WHERE EMP-ID-0415 LE 40 ORDER BY EMP-LAST-NAME-0415
DISPLAY COLS EMP-ID-0415 EMP-LAST-NAME-0415 Output page 1: OLQ RELEASE 19.0

Output page 1:
DATE: 96.267 TIME: 11:00 PAGE: 0001
Copyright (C) yyyy CA, Inc.
OLQ 107017 00 CA OLQ Release 19.0
OLQ 107019 00 Copyright(C) yyyy CA, Inc.
SIGNON SS EMPSS01 DBN=EMPDEMO DICTNAME TESTDICT
OLQ 100021 00 Ready to retrieve data from subschema IDMSNWKA
OLQ 100022 00 Schema: EMPSCHM Version: 1
OLQ 100023 00 Database name: EMPDEMO
OLQ 100025 00 Dictionary name: TESTDICT
OLQ 098006 00 18 Whole lines and 0 partial lines in report.
OLQ 098007 00 18 Records read. 18 Records selected.
DISPLAY COLS EMP-ID-0415 EMP-LAST-NAME-0415 Output page 2: EMPLOYEE.REPORT

EMP-ID-0415 EMP-LAST-NAME-0415
7 BANK
40 CRANE
24 DOUGH
32 FERNDALE
29 GALLWAY
3 GARFIELD
28 GRANGER
27 HEAROWITZ
30 HENDON
20 JACOBI
19 JENSEN
11 JENSON
16 KLEWELLEN
31 LIPSICH
35 LITERATA
15 MAKER
23 O'HEARN
12 PEOPLES
21 TYRO
Creating Multiple Reports in One Job

To create several reports in one batch run, enter the CA OLQ processing statements sequentially in your input data stream. The DISPLAY command places the start of each report on a new page, and resets the page count to 1.

Example:

This batch run creates two reports: one listing the names of employees living in Boston and one listing the names of employees living in Medford:

```
DEFINE FILE INFILE RECORD EMPLOYEE (1) DICT TESTDICT
SET OPTION SPARSE
GET ALL SEQ INFILE.EMPLOYEE WHERE EMP-CITY-0415 EQ BOSTON
DIS COLS EMP-NAME-0415 EMP-CITY-0415
GET ALL SEQ INFILE.EMPLOYEE WHERE EMP-CITY-0415 EQ MEDFORD
DIS COLS EMP-NAME-0415 EMP-CITY-0415
```

Wide Reports

To display wide reports in CA OLQ batch, specify the keywords RIGHT or LEFT with your DISPLAY command.

Example:

If you specify DISPLAY RIGHT, your output report lists the next 132 characters (from left to right) of the report, for as many pages as there are in the report. The display shifts right each time you issue a DISPLAY RIGHT.

Creating a Report with SELECT

This example uses a SELECT statement to retrieve information about departments and employees. It also creates a report (rather than writing to disk). To display your output in a report, issue a DISPLAY command in your input data stream:

```
INPUT: SET USER ABC PASS ABCPASS
SET ACCESS OLQ
OLQ 092032 00 PROCESSING MODE CHANGED TO OLQ.
SELECT * FROM DEPARTMENT, EMPLOYEE WHERE DEPARTMENT.DEPT-HEAD-ID-0410 EQ EMPLOYEE.EMP-ID-0415
DISPLAY COLS DEPT-NAME-0410 DEPT-HEAD-ID-0410 EMP-LAST-NAME-0415
```

Output:

```
OLQ RELEASE 19.0 DATE: 96.267 TIME: 11:00 PAGE: 0001
Copyright (C) yyyy CA, Inc.
OLQ 107017 00 CA OLQ Release 19.0
OLQ 107019 00 Copyright(C) yyyy CA, Inc.
SET USER ABC PASS ?????????
SIGNON SS EMPSS01 DBN=EMPDEMO DICTNAME TESTDICT
OLQ 100021 00 Ready to retrieve data from subschema IDMSNWKA
```
Writing to a Disk File with SELECT

This example routes the outcome of a SELECT statement to a disk file. To route your output to an output file, you must:

1. Set the access switch to `olq`

2. Specify the keyword `OUTPUT` and the name of the output file in your SELECT statement

Example:

To route your output to the OUTFILE output file, issue the following statements:

```
set access olq
select * from employee
  where emp-id-0415 eq 4500 output outfile
```

This example creates a sequential file containing departments and the name of each department's manager:

**Input:** SET USER ABC PASS ABCPASS
SIGNON SS EMPS01 DBN=EMPDEMO DICTNAME TESTDICT
SELECT * FROM DEPARTMENT, EMPLOYEE WHERE
  DEPARTMENT.DEPT-HEAD-ID-0410 EQ EMPLOYEE.EMP-ID-0415
OUTPUT OUTFILE

**Output:**

```
OLQ RELEASE 19.0           DATE: 96.267 TIME: 11:00 PAGE: 0001
Copyright (C) yyyy CA, Inc.
```

```
OLQ 100023 00 Database name: EMPDEMO
OLQ 100025 00 Dictionary name: TESTDICT
SELECT * FROM DEPARTMENT, EMPLOYEE WHERE
OLQ 092022 00 Continuation line accepted
DEPARTMENT.DEPT-HEAD-ID-0410 EQ EMPLOYEE.EMP-ID-0415
OLQ 098006 00 9 whole lines and 0 partial lines in report.
OLQ 098007 00 18 records read. 18 records selected.
DISPLAY COLS DEPT-NAME-0410 DEPT-HEAD-ID-0410 EMP-LAST-NAME-0415
DEPARTMENT/EMPLOYEE REPORT
9/19/91
DEPT-NAME-0410       DEPT-HEAD-ID-0410   EMP-LAST-NAME-0415
EXECUTIVE ADMINISTRATION   30     HENDON
ACCOUNTING AND PAYROLL      11     JENSON
PERSONNEL                  13     PEOPLES
INTERNAL SOFTWARE         3       GARFIELD
BLUE SKIES                 321     MOON
COMPUTER OPERATIONS        4       CRANE
PUBLIC RELATIONS           7       BANK
BRAINSTORMING              15     MAKER
THERMOREGULATION         349     WILCO
END OF REPORT
```

```
OLQ RELEASE 19.0           DATE: 96.267 TIME: 11:00 PAGE: 0003
Copyright (C) yyyy CA, Inc.
```

```
OLQ 100029 00 Signoff accepted - OLQ session terminated.
```
OLQ 100022 00 Schema: EMPSCHM Version: 1
OLQ 100023 00 Database name: EMPDEMO
OLQ 100025 00 Dictionary name: TESTDICT
SELECT * FROM DEPARTMENT, EMPLOYEE WHERE -
OLQ 092022 00 Continuation line accepted
DEPARTMENT.DEPT-HEAD-ID-0410 EQ EMPLOYEE.EMP-ID-0415 -
OLQ 092022 00 Continuation line accepted
OUTPUT OUTFILE
OLQ 149018 00

<table>
<thead>
<tr>
<th>Field name</th>
<th>Offset</th>
<th>Size</th>
<th>Dec no</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTFILE DEPT-ID-0410</td>
<td>0000</td>
<td>0004</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE DEPT-NAME-0410</td>
<td>0004</td>
<td>0045</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE DEPT-HEAD-ID-0410</td>
<td>0049</td>
<td>0004</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE EMP-ID-0415</td>
<td>0053</td>
<td>0004</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE EMP-FIRST-NAME-0415</td>
<td>0057</td>
<td>0010</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE EMP-LAST-NAME-0415</td>
<td>0067</td>
<td>0015</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE EMP-STREET-0415</td>
<td>0082</td>
<td>0020</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE EMP-CITY-0415</td>
<td>0102</td>
<td>0015</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE EMP-STATE-0415</td>
<td>0117</td>
<td>0002</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE EMP-ZIP-FIRST-FIVE-0415</td>
<td>0119</td>
<td>0005</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE EMP-ZIP-LAST-FOUR-0415</td>
<td>0124</td>
<td>0004</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE EMP-PHONE-0415</td>
<td>0128</td>
<td>0010</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE STATUS-0415</td>
<td>0138</td>
<td>0002</td>
<td></td>
<td>CHARACTER</td>
</tr>
<tr>
<td>OUTFILE SS-NUMBER-0415</td>
<td>0140</td>
<td>0009</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE START-YEAR-0415</td>
<td>0149</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE START-MONTH-0415</td>
<td>0151</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE START-DAY-0415</td>
<td>0153</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE TERMINATION-YEAR-0415</td>
<td>0155</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE TERMINATION-MONTH-0415</td>
<td>0157</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE TERMINATION-DAY-0415</td>
<td>0159</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE BIRTH-YEAR-0415</td>
<td>0161</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE BIRTH-MONTH-0415</td>
<td>0163</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
<tr>
<td>OUTFILE BIRTH-DAY-0415</td>
<td>0165</td>
<td>0002</td>
<td>0000</td>
<td>UNSIGNED ZONE</td>
</tr>
</tbody>
</table>
Setting Defaults

The option defaults described here can be overridden with the CA OLQ SET and OPTIONS commands. If these options are not specified during a CA OLQ session, the options take on the default values as described in this chapter.

You set these options during system generation or with IDD.

System Generation Options

Table 14 presents the options available during system generation and the statements that control the options. The system generation statement is ADD/MODIFY/DELETE OLQ.

Table 14. CA OLQ Options Set During System Generation

<table>
<thead>
<tr>
<th>Option</th>
<th>Clause of the Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLQ</td>
<td>INCLUDE/EXCLUDE AUTHORITY IS OLQ</td>
</tr>
<tr>
<td>Access to ASF tables, logical and database records, and sequential files</td>
<td>OLQ ACCESS TO OLQ</td>
</tr>
<tr>
<td>Access to SQL tables</td>
<td>OLQ ACCESS TO IDMS</td>
</tr>
<tr>
<td>PF key module</td>
<td>PFKEY MODULE IS qfile-name</td>
</tr>
<tr>
<td>Interrupt count</td>
<td>INTERRUPT COUNT IS 100/interrupt-count</td>
</tr>
<tr>
<td>Maximum interrupt count</td>
<td>MAXIMUM INTERRUPT COUNT IS 32767/maximum-interrupt-count</td>
</tr>
<tr>
<td>Menu mode</td>
<td>MENU MODE IS ALLOWED/DISALLOWED/ONLY</td>
</tr>
<tr>
<td>Report retention</td>
<td>REPORT RETENTION IS 1/retention-period/FOREVER</td>
</tr>
<tr>
<td>Maximum report retention</td>
<td>MAXIMUM REPORT RETENTION IS 5/max-report-retention/FOREVER</td>
</tr>
<tr>
<td>Size of report pages (in bytes)</td>
<td>REPORT FILE PAGE SIZE IS 4000/report-file-page-size</td>
</tr>
<tr>
<td>Maximum number of reports</td>
<td>MAXIMUM REPORT COUNT IS 5/maximum-report-count</td>
</tr>
<tr>
<td>Number of pages per report</td>
<td>MAXIMUM REPORT PAGES IS 5/maximum-report-pages</td>
</tr>
<tr>
<td>Maximum storage for sorts</td>
<td>MAXIMUM SORT STORAGE IS 100/max-sort-storage-size</td>
</tr>
<tr>
<td>Number of input lines</td>
<td>INPUT LINE SIZE IS 4/input-line-size</td>
</tr>
<tr>
<td>Continuation character</td>
<td>CONTINUATION CHARACTER IS -/continuation-character</td>
</tr>
<tr>
<td>Separator character</td>
<td></td>
</tr>
</tbody>
</table>
### Option

<table>
<thead>
<tr>
<th>Clause of the Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEPARATOR CHARACTER IS '!'/separator-character</td>
</tr>
<tr>
<td>COMMENT CHARACTER IS ';'/comment-character</td>
</tr>
<tr>
<td>REPORT DICTNAME IS dictionary-name</td>
</tr>
<tr>
<td>BATCH CLASS IS 0/batch-class</td>
</tr>
</tbody>
</table>

Special considerations for the options available during system generation are presented below.

**Access to ASF tables:**

You can create, replace, modify, and delete ASF tables.

You use the IDD DDDL source statement USER to set the access switch to **OLQ**:

```plaintext
0LQ  ACCESS  IS  0LQ
```

OLQ is the default.

The OLQ setting also allows you to use the SELECT statement with the following entities:

- ASF tables
- Logical records
- Database records
- Sequential files (batch only)

**Access to SQL tables:**

You can create, replace, modify, and delete tables which are associated with an SQL schema.

You use the IDD DDDL source statement USER to set the access switch to **IDMS**:

```plaintext
0LQ  ACCESS  IS  IDMS
```

**Note:**

- For information on the USER statement, see the *CA IDMS IDD DDDL Reference Guide*.
- For more information about System Generation, see the *CA IDMS System Generation Guide*.

**PF key module:**
You use the system generation statement ADD OLQ to define a qfile as the PF key module to be executed for each user issuing the CA OLQ task code. The intended use of the PF key module is to contain a series of CA OLQ SET FUNCTION commands that assign values to each of the PF keys in CA OLQ.

The specific clause of the ADD OLQ statement is:

```
PFKEY MODULE IS qfile-name
```

**How to set interrupt:**

The interrupt count interrupts processing after the specified number of CA OLQ database requests. You can set:

- A **default interrupt count** -- To specify the interrupt count should the user not specify one
- A **maximum interrupt count** -- To specify the interrupt count should the user specify a count greater than the default interrupt count mentioned above

The system generation statements that set these defaults are:

```
ADD/MODIFY OLQ ...
INTERRUPT COUNT IS interrupt-count
MAXIMUM INTERRUPT COUNT IS max-interrupt-count
```

These statements set the default interrupt count and the default maximum interrupt for CA OLQ.

**Access to menu mode:**

Specify a user's access to menu mode with:

```
MENU MODE IS ALLOWED/NOT ALLOWED/ONLY
```

Users are automatically allowed to access command mode with the ALLOWED and NOT ALLOWED options. **ONLY** specifies that the user is allowed to access CA OLQ through menu mode only.

**Maximum sort storage:**

You can set the maximum amount of storage, in K bytes, that CA OLQ can use for sort operations. Specifying too low a value prevents CA OLQ from performing sort operations. Specifying too large a value degrades CV performance.

The system generation statement that sets this option is:

```
ADD/MODIFY OLQ ...
MAXIMUM SORT STORAGE IS 100/max-sort-storage-size
```

**Number of input lines:**

You can define the number of lines on the screen reserved for command input. The maximum number of lines defined by input-line-size is limited to the total number of lines on the screen minus the number of lines allocated for output.

Keep in mind that a large line size may be needed to comfortably code SELECT statements or multiple commands.
The system generation statement that sets this option is:

```
ADD/MODIFY OLQ ...
INPUT LINE SIZE IS 4
```

**Continuation character:**

To set the default continuation character during system generation, use the statement:

```
ADD/MODIFY OLQ ...
CONTINUATION CHARACTER IS -
```

Use the continuation character in qfiles and batch when the length of a CA OLQ command exceeds one line. You can also use the continuation character to continue commands (or a series of commands) across a pseudo-converse. Specify the continuation character at the end of each line to be continued.

**Separator character:**

The separator character is used to separate commands in the command input area. To set the default separator character during system generation, use the statement:

```
ADD/MODIFY OLQ ...
SEPARATOR CHARACTER IS !
```

**Comment character:**

The comment character is used to separate commands from comments. Anything typed in after the comment character is considered to be a comment and is ignored by CA OLQ. Entering comments can be useful in documenting qfiles.

To set the default comment character during system generation, use the statement:

```
ADD/MODIFY OLQ ...
COMMENT CHARACTER IS ;
```

**Batch class:**

Batch class is used to activate the CA OLQ batch interface. You can specify the print class used by CA OLQ when submitting batch jobs under z/OS. To specify the default print class during system generation, use the statement:

```
ADD/MODIFY OLQ ...
BATCH CLASS IS 0
```

You should ensure that the specified class has been assigned to an internal reader.

### Integrated Data Dictionary Options

**Table 15** presents the options available through IDD and the statements that control the options.

**Table 15. CA OLQ Options Set With the IDD ADD/MODIFY/DELETE USER Statement**

<table>
<thead>
<tr>
<th>Option</th>
<th>Clause of the Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLQ</td>
<td>INCLUDE/EXCLUDE AUTHORITY IS OLQ</td>
</tr>
</tbody>
</table>
Access to ASF tables, logical and database records, and sequential files | OLQ ACCESS TO OLQ
Access to SQL tables | OLQ ACCESS TO IDMS
Subschema access | INCLUDE/EXCLUDE ACCESS TO SUBSCHEMA

**rule = no.**

**Signon qfile**

| ss-name OF SCHEMA schm-name USER user-name |
| SIGNON QFILE IS qfile-name |

**Signon profile qfiles**

| SIGNON PROFILE IS qfile-name LANGUAGE IS OLQ |

**Access to qfiles**

| INCLUDE/EXCLUDE ACCESS TO QFILE qfile-name |

**Execution of qfiles**

| QFILE IS ALLOWED/NOT ALLOWED/ONLY |

**Menu mode**

| MENU MODE IS ALLOWED/NOT ALLOWED/ONLY |

**Saving qfiles**

| QFILE SAVE IS ALLOWED/NOT ALLOWED |

**Retrieving multiple records**

| MRR IS ALLOWED/NOT ALLOWED |

**Interrupt count**

| MANDATORY/OPTIONAL INTERRUPT |

**Sorts**

| SORT IS ALLOWED/NOT ALLOWED |

**Default options**

| DEFAULT OPTIONS ARE ... |

Special considerations for the options available through IDD are presented below.

⚠️ **Note:** For complete syntax and syntax rules, see the [CA IDMS IDD DDDL Reference Guide](#).

**OLQ:**

You can allow users to distribute authority in IDD with the IDD USER statement. USER statement clauses are used to control access to CA OLQ qfiles and subschema views and to assign CA OLQ command authorities and processing and reporting options when the default processing options for the session include SECURITY FOR OLQ IS ON.

If OLQ is specified, the keyword UPDATE must be specified in the FOR clause of the ADD USER statement.

**Access to ASF tables:**

You can create, replace, modify, and delete ASF tables.

You use the IDD DDDL source statement USER to set the access switch to **OLQ**:

```plaintext
0LQ ACCESS IS OLQ
```
OLQ is the default.

The OLQ setting also allows you to use the SELECT statement with the following entities:

- ASF tables
- Logical records
- Database records
- Sequential files (batch only)

**Access to SQL tables:**

You can create, replace, modify, and delete tables which are associated with an SQL schema.

You use the IDD DDDL source statement USER to set the access switch to **IDMS**:

```
OLQ ACCESS IS IDMS
```

**Note:** For information on the USER statement, see the *CA IDMS IDD DDDL Reference Guide*.

**Subschema access:**

You can specify whether the user does or doesn't have access to a particular subschema. CA OLQ uses the DC/UCF signon ID and password of the user to determine which subschemas are available. Subschema security is enforced on a dictionary-by-dictionary basis.

**Signon profile qfiles and users:**

You can associate a signon qfile with a subschema. The signon qfile executes when a user signs on to the subschema.

You use the IDD DDDL source statement entity USER to associate a signon profile with a specific user. The specific clause of the ADD USER statement is:

```
SIGNON PROFILE IS module-name LANGUAGE IS OLQ
```

*Module-name* refers to the name of the saved qfile.

**Access to qfiles:**

You can allow users to access only certain qfiles by naming the qfiles in the user’s ADD USER statement, using the clause:

```
INCLUDE/EXCLUDE ACCESS TO QFILE qfile-name
```

**Executing qfiles:**

Use IDD to specify whether or not specific users are allowed to execute qfiles in general. The IDD statement that applies to this option is the following clause of the ADD USER statement:
QFILE IS ALLOWED/NOT ALLOWED/ONLY

With the ONLY option of the above clause, you can specify that the named user can access CA OLQ only through qfile execution. If you want the user to access CA OLQ only through qfiles, we recommend that you also specify MENU MODE IS DISALLOWED to keep the user from retrieving through menu mode.

Menu mode:

This security option assigns or denies access to the CA OLQ menu mode facility as follows:

- **ALLOWED** (default) authorizes the CA OLQ user to access CA OLQ in command mode and menu mode.
- **NOT ALLOWED** authorizes the CA OLQ user to access CA OLQ in command mode only.
- **ONLY** authorizes the CA OLQ user to access CA OLQ in menu mode only.

The IDD statement that applies to this option is:

```
MENU MODE IS ALLOWED/NOT ALLOWED/ONLY
```

Saving qfiles:

You can specify whether or not you want the user to be able to save qfiles after creating them with the following clause of the ADD USER statement:

```
QFILE SAVE IS ALLOWED/NOT ALLOWED
```

Retrieving multiple records:

This security option specifies whether the user can retrieve multiple record occurrences with a single CA OLQ command. To use the SELECT (IDMS access mode) statement, the user must be assigned authority to retrieve multiple record occurrences.

The IDD statement that applies to this option is:

```
MRR IS ALLOWED/NOT ALLOWED
```

Interrupt count:

Each user in CA OLQ has the ability to set an INTERRUPT COUNT during a CA OLQ session. The interrupt count is the number of records that can be retrieved before the system interrupts processing. As the DBA, you can specify whether the user is allowed to select the NO INTERRUPT option with the IDD option:

```
MANDATORY/OPTIONAL INTERRUPT
```

If you set MANDATORY INTERRUPT, the user cannot choose the NO INTERRUPT option in CA OLQ. If you set OPTIONAL INTERRUPT, the user can choose the NO INTERRUPT option in CA OLQ. When the NO INTERRUPT option is in effect, the default interrupt count set during system generation is used to perform commit checkpoints.

Sorts:

When you specify:
SORT IS NOT ALLOWED

Keep in mind that you are prohibiting sorting in OLQ SELECT commands.

Setting CA OLQ default options:

The CA OLQ default options allow you to tailor the CA OLQ environment. These options come into effect at user signon regardless of the security status. The CA OLQ default options include:

- **HEADER/NO HEADER** specifies whether CA OLQ report files contain a header line. This option has no effect on single-record-occurrence retrieval displays.

- **ECHO/NO ECHO** specifies whether a user-entered command will be repeated by CA OLQ on the output device.

- **ALL/NONE** specifies whether the default internal field list for all records retrieved during the user's CA OLQ session will contain all or none of the fields. Menu mode always defaults to none; no fields are preselected.

- **FILLER/NO FILLER** specifies whether filler field values are displayed.

- **INTERRUPT/NO INTERRUPT** specifies whether the processing interrupt feature for multiple record retrievals is enabled or disabled.

  **Note:** The MANDATORY INTERRUPT specification takes precedence over NO INTERRUPT.

  If INTERRUPT is specified, CA OLQ breaks processing after the specified number of records has been retrieved. If NO INTERRUPT is specified (as long as OPTIONAL INTERRUPT has been specified), the interrupt count is used for commit checkpoints.

- **WHOLE/PARTIAL** specifies the content of displayed path retrieval report lines. WHOLE displays only those lines containing a retrieved occurrence for every record type in a path definition. PARTIAL displays all lines, whether or not they contain data for every path record type. SELECT (OLQ access mode), and therefore menu mode, doesn't build partial lines.

- **FULL/SPARSE** specifies the format of displayed path retrieval report lines. FULL displays data associated with a record type once for each retrieved occurrence. SPARSE, used with a SELECT command, displays only the first of a repeating data value; SPARSE, used with a path command, displays only the first of a repeating record type.

- **OLQ HEADER/NO OLQ HEADER** specifies whether CA OLQ is to use predefined headers as columns headers in the report. This option has no effect on single-record-occurrence retrieval displays.

- **COMMENTS/NO COMMENTS** specifies whether comments accompany the output from HELP RECORDS, HELP SUBSCHEMAS, and HELP QFILE requests.

- **CODETABLE/NO CODETABLE** specifies whether CA OLQ accesses a code table to encode and decode data.
PATHSTATUS/NO PATHSTATUS specifies the conditions under which CA OLQ will retrieve a logical record. NO PATHSTATUS requests CA OLQ to retrieve a logical record only when the path status of LR-FOUND is returned. PATHSTATUS requests CA OLQ to retrieve a logical record when any DBA-defined path status is returned.

EXTERNAL PICTURE/NO EXTERNAL PICTURE specifies whether CA OLQ will use external pictures for displaying data.

VERBOSE/TERSE controls the amount of information displayed following record and field-level breaks.
OLQ Coding Considerations

This article describes considerations when entering commands in CA OLQ command mode.

- Commands (see page 290)
- Defining Session Characters (see page 291)
- Entering Data (see page 292)

Commands

Considerations for commands are as follows:

Using Abbreviations

Three-letter abbreviations are valid for most commands and keywords.

Ending Command Strings

You can end a command string with a comment character or a separator character.

Comment characters signal to CA OLQ that everything following is to be ignored. You can use comment characters to document qfiles. Use separator characters for separating commands in the input field. Even when commands are one to a line in the input field, they must be separated by a separator character.

Issuing Multiple Commands

You can specify multiple commands in a single pseudo-converse by using a separator character. The separator character is defined at system generation as the exclamation point (!). When you use separators with the SELECT statement (IDMS mode), the separators must precede the statement. CA IDMS/DB does not accept a syntax if the separator comes after the SELECT statement:

The following example shows a valid command line that includes multiple commands and the SELECT statement:

```
delete table employee.job!select all from emp_id
```

Invoking Function Keys

To invoke a predefined function from a terminal that does not have function keys, the user can enter the following commands:
Continuation Character

You can use a continuation character to continue commands across pseudo-converses, and continue
batch or qfile commands on a following line.

Defining the comment character:

The default comment character is the semi-colon (;), as defined at system generation. You can
change the definition at any time in command mode CA OLQ by issuing a SET COMMENT CHARACTER
command.

Access mode: You cannot use comment characters with SELECT (IDMS mode) statements.

Defining the separator character:

You can change the definition of the separator character with the SET SEPARATOR CHARACTER
command.

Defining Session Characters

Considerations for defining session characters are as follows:

Comment Character

The default comment character is the semi-colon, as defined at system generation. You can change
the definition at any time in CA OLQ command mode by issuing a SET COMMENT CHARACTER
command. You cannot use comment characters with the SELECT statement.

Separator Character

You can change the definition of the separator character with the SET SEPARATOR CHARACTER
command.
Continuation Character

You can change the definition of the continuation character with the SET CONTINUATION CHARACTER command.

Entering Data

The following input considerations should be noted:

- **Commas as decimal points**: Any numeric value that contains an embedded comma can be interpreted as a decimal number. The ability to recognize commas as decimal points is enabled at system generation.

- **Record names and CA OLQ keywords**: If a database record name is the same as an OLQ keyword, the name should be enclosed in quotation marks.

Specifying Data Values

CA OLQ commands sometimes require user-supplied data values, such as a CALC key, sort key, database key, or user-defined item. The following types of data values are recognized by CA OLQ:

- **Integer**: A whole number in the range -32767 to +32767.

- **Binary**: A binary number.

- **Real number**: A decimal number.

- **Hexadecimal number**: A string of up to 64 hexadecimal digits enclosed in single or double quotation marks and preceded by an X. An even number of digits must be specified; for example, X'00AFD6'.

- **Simple character string**: A 1- to 64-character alphanumeric value that consists of alphabetic letters, numeric digits, and the following special characters: @, $, #, -.  

- **Complex character string**: A 1- to 256-character alphanumeric value enclosed in single or double quotation marks. If a quotation mark appears within a string, it must appear twice or be distinguishable from the enclosing quotation marks. The following examples illustrate acceptable formats for specifying quotation marks within complex character strings:

  "LEE"S"

  'LEE"S'

  'LEE"S'
Enclosing quotation marks are not evaluated by the OLQ command processor. If the value of a complex character string matches a field name specified in the same command, the results are unpredictable.

- **Graphics literal**: A string of up to 32 double-byte characters string (DBCS) characters enclosed with shiftstrings and single quotation marks and prefixed by a G, for example, G'\<DBCS characters>'.

  A mix of EBCDIC and DBCS characters is allowed as long as the correct shiftin/shiftout sequences identifies the code set, for example, 'ABC\<DBCS characters>UVW'.

- **Group value**: A series of values specified in any of the forms described above, separated by commas or blanks. Group values cannot exceed 64 characters in length, and can be enclosed in parentheses; for example, ('8 Cedar St', 'Worcester, MA', 02312).

- **Database key**: A decimal page and line number that define the actual location of the record in the database; For example, 10023:14, where 10023 is the database page number and 14 is the line number of the specified record.

- **Floating point constant**: A representation of a real number as a fixed-point value with an exponent— value E+/-integer-value, for example, 1.2E-22, -3E2.

Numeric and alphanumeric character strings for database fields are interpreted according to their formats in the subschema view. The user need not supply leading zeros or trailing blanks. CA OLQ automatically pads values, as follows:

- **Alphanumeric**: Left-justified, padded with blanks

- **Numeric**: Decimal-aligned, sign-extended (default is plus), zero-filled to the right of the decimal

### Specifying Subscripts

Subscripts are used to identify specific field entries in an array.

The following table shows the array of values for a record that contains sales figures for two divisions of a company for each quarter in the last two years:

<table>
<thead>
<tr>
<th>Year 1</th>
<th>Quarter 1</th>
<th>Div-sales₁</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Div-sales₂</td>
</tr>
<tr>
<td></td>
<td>Quarter 2</td>
<td>Div-sales₁</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div-sales₂</td>
</tr>
<tr>
<td></td>
<td>Quarter 3</td>
<td>Div-sales₁</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div-sales₂</td>
</tr>
<tr>
<td></td>
<td>Quarter 4</td>
<td>Div-sales₁</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div-sales₂</td>
</tr>
</tbody>
</table>
Year 2 | Quarter 1 | Div-sales₁
|-------|----------|---------|
|       |          | Div-sales₂
|-------|----------|---------|
|       | Quarter 2 | Div-sales₁
|       |          | Div-sales₂
|-------|----------|---------|
|       | Quarter 3 | Div-sales₁
|       |          | Div-sales₂
|-------|----------|---------|
|       | Quarter 4 | Div-sales₁
|       |          | Div-sales₂

**Examples**

The following example specifies the fourth quarter of the first year:

QUARTER(1, 4)

The following example specifies third quarter sales for the first division in the second year:

DIV-SALES(2, 3, 1)