Use the functions that provide reduced CPU time

Discover improved scalability and availability

Reduce TCO with more zIIP eligibility
Note: Before using this information and the product it supports, read the information in “Notices” on page xix.
# Contents

**Figures** ........................................................................................................... xi

**Examples** .......................................................................................................... xv

**Tables** ............................................................................................................... xvii

**Notices** ............................................................................................................. xix
**Trademarks** ....................................................................................................... xx

**Preface** .............................................................................................................. xxi
**The team that wrote this book** ........................................................................... xxi
**Become a published author** ............................................................................... xxiv
**Comments welcome** ....................................................................................... xxiv

**Summary of changes** ....................................................................................... xxv
**September 2007, First Edition** .......................................................................... xxv
  **March 2008, First Update** ................................................................................ xxv
  **September 2008, Second Update** .................................................................... xxv
  **March 2009, Third Update** ............................................................................... xxvi
  **April 2009, Fourth Update** ............................................................................... xxvi
  **November 2009, Fifth Update** ....................................................................... xxvii
  **July 2010, Sixth Update** .................................................................................. xxviii
  **July 2011, Seventh Update** ............................................................................. xxviii
  **September 2011, Eighth Update** .................................................................... xxviii
  **September 2012, Ninth Update** ..................................................................... xxviii

**Chapter 1. Overview of DB2 9 for z/OS** .............................................................. 1
**1.1 DB2 9 for z/OS** ............................................................................................ 2
**1.2 SQL enhancements** ..................................................................................... 3
**1.3 XML** ............................................................................................................ 4
**1.4 DB2 subsystem** ............................................................................................ 4
**1.5 Availability and capacity** ............................................................................. 6
**1.6 Utility performance** ..................................................................................... 6
**1.7 Networking and e-business** ......................................................................... 7
**1.8 Data sharing enhancements** ........................................................................ 8
**1.9 Installation and migration** .......................................................................... 8
**1.10 Performance tools** ..................................................................................... 8

**Chapter 2. SQL performance** .......................................................................... 11
**2.1 DISTINCT and GROUP BY enhancements** ............................................... 12
  **2.1.1 Performance with group collapsing** ....................................................... 12
  **2.1.2 Performance for DISTINCT sort avoidance** ....................................... 13
  **2.1.3 Conclusion** ............................................................................................ 13
**2.2 Dynamic prefetch enhancement for regular index access during an SQL call** .... 14
  **2.2.1 Performance** .......................................................................................... 14
**2.3 Global query optimization** .......................................................................... 14
  **2.3.1 Performance** .......................................................................................... 16
**2.4 MERGE and SELECT FROM MERGE** .................................................... 19
  **2.4.1 Performance** .......................................................................................... 19
  **2.4.2 Conclusions** .......................................................................................... 23
### Chapter 4. DB2 subsystem performance

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 CPU utilization in the DB2 9 for z/OS</td>
<td>83</td>
</tr>
<tr>
<td>4.1.1 OLTP processing</td>
<td>83</td>
</tr>
<tr>
<td>4.1.2 Query processing</td>
<td>84</td>
</tr>
<tr>
<td>4.1.3 Column processing</td>
<td>84</td>
</tr>
<tr>
<td>4.2 CPU utilization in the client/server area</td>
<td>84</td>
</tr>
<tr>
<td>4.2.1 Conclusion</td>
<td>85</td>
</tr>
<tr>
<td>4.3 z10 and DB2 workload measurements</td>
<td>85</td>
</tr>
<tr>
<td>4.3.1 z10 performance</td>
<td>87</td>
</tr>
<tr>
<td>4.3.2 DB2 performance with z10</td>
<td>87</td>
</tr>
<tr>
<td>4.3.3 Conclusion</td>
<td>91</td>
</tr>
<tr>
<td>4.4 The IBM zEnterprise System and DB2</td>
<td>91</td>
</tr>
<tr>
<td>4.4.1 Benefits of zEnterprise for DB2</td>
<td>92</td>
</tr>
<tr>
<td>4.4.2 IBM Smart Analytics Optimizer for DB2 for z/OS</td>
<td>93</td>
</tr>
<tr>
<td>4.5 Virtual storage constraint relief</td>
<td>94</td>
</tr>
<tr>
<td>4.5.1 EDM pool changes for static SQL statements</td>
<td>95</td>
</tr>
<tr>
<td>4.5.2 EDM pool changes for dynamic SQL statements</td>
<td>96</td>
</tr>
<tr>
<td>4.5.3 Below-the-bar EDM pool</td>
<td>97</td>
</tr>
<tr>
<td>4.5.4 CACHEDYN_FREELOCAL</td>
<td>97</td>
</tr>
<tr>
<td>4.5.5 Automated memory monitoring</td>
<td>98</td>
</tr>
<tr>
<td>4.5.6 Instrumentation and processes for virtual storage monitoring</td>
<td>99</td>
</tr>
<tr>
<td>4.5.7 Conclusion</td>
<td>100</td>
</tr>
<tr>
<td>4.5.8 Recommendations</td>
<td>100</td>
</tr>
<tr>
<td>4.6 Real storage</td>
<td>101</td>
</tr>
<tr>
<td>4.6.1 Conclusion</td>
<td>103</td>
</tr>
<tr>
<td>4.6.2 Recommendations</td>
<td>103</td>
</tr>
<tr>
<td>4.7 Distributed 64-bit DDF</td>
<td>104</td>
</tr>
<tr>
<td>4.7.1 z/OS shared memory usage measurement</td>
<td>106</td>
</tr>
<tr>
<td>4.7.2 Conclusion</td>
<td>106</td>
</tr>
<tr>
<td>4.7.3 Recommendation</td>
<td>107</td>
</tr>
<tr>
<td>4.8 Distributed address space virtual storage</td>
<td>107</td>
</tr>
<tr>
<td>4.9 Distributed workload throughput</td>
<td>108</td>
</tr>
<tr>
<td>4.9.1 Conclusion</td>
<td>108</td>
</tr>
<tr>
<td>4.10 WLM assisted buffer pool management</td>
<td>108</td>
</tr>
<tr>
<td>4.11 Automatic identification of latch contention and DBM1 below-the-bar virtual storage</td>
<td>110</td>
</tr>
<tr>
<td>4.11.1 Verification</td>
<td>112</td>
</tr>
<tr>
<td>4.12 Latch class contention relief</td>
<td>114</td>
</tr>
<tr>
<td>4.12.1 Latch class 6</td>
<td>116</td>
</tr>
<tr>
<td>4.12.2 Latch class 19</td>
<td>116</td>
</tr>
<tr>
<td>4.12.3 Latch class 24</td>
<td>117</td>
</tr>
<tr>
<td>4.13 Accounting trace overhead</td>
<td>118</td>
</tr>
<tr>
<td>4.13.1 Typical user workload</td>
<td>118</td>
</tr>
<tr>
<td>4.13.2 Tracing relative percentage overhead</td>
<td>119</td>
</tr>
<tr>
<td>4.13.3 Conclusion</td>
<td>120</td>
</tr>
<tr>
<td>4.13.4 Recommendation</td>
<td>121</td>
</tr>
<tr>
<td>4.14 Reordered row format</td>
<td>121</td>
</tr>
<tr>
<td>4.14.1 Reordered row format and compression</td>
<td>123</td>
</tr>
<tr>
<td>4.14.2 Conclusion</td>
<td>123</td>
</tr>
</tbody>
</table>
### Chapter 5. Availability and capacity enhancements

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3 Object-level recovery</td>
<td>145</td>
</tr>
<tr>
<td>5.4 Relief for sequential key insert</td>
<td>145</td>
</tr>
<tr>
<td>5.4.1 Insert improvements with DB2 9 for z/OS</td>
<td>145</td>
</tr>
<tr>
<td>5.4.2 Performance</td>
<td>146</td>
</tr>
<tr>
<td>5.4.3 Conclusions</td>
<td>147</td>
</tr>
<tr>
<td>5.4.4 Recommendations</td>
<td>147</td>
</tr>
<tr>
<td>5.5 Index compression</td>
<td>148</td>
</tr>
<tr>
<td>5.5.1 Performance</td>
<td>148</td>
</tr>
<tr>
<td>5.5.2 Conclusions</td>
<td>149</td>
</tr>
<tr>
<td>5.5.3 Recommendation</td>
<td>149</td>
</tr>
<tr>
<td>5.6 Log I/O enhancements</td>
<td>150</td>
</tr>
<tr>
<td>5.6.1 Conclusion</td>
<td>150</td>
</tr>
<tr>
<td>5.6.2 Recommendations</td>
<td>150</td>
</tr>
<tr>
<td>5.7 Not logged table spaces</td>
<td>151</td>
</tr>
<tr>
<td>5.7.1 Performance</td>
<td>151</td>
</tr>
<tr>
<td>5.7.2 Conclusion</td>
<td>152</td>
</tr>
<tr>
<td>5.7.3 Recommendations</td>
<td>152</td>
</tr>
<tr>
<td>5.8 Prefetch and preformatting enhancements</td>
<td>153</td>
</tr>
<tr>
<td>5.8.1 Performance</td>
<td>153</td>
</tr>
<tr>
<td>5.8.2 Conclusion</td>
<td>154</td>
</tr>
<tr>
<td>5.9 WORKFILE database enhancements</td>
<td>155</td>
</tr>
<tr>
<td>5.9.1 Performance</td>
<td>155</td>
</tr>
<tr>
<td>5.9.2 Conclusions</td>
<td>156</td>
</tr>
<tr>
<td>5.9.3 Recommendations</td>
<td>156</td>
</tr>
<tr>
<td>5.10 LOB performance enhancements</td>
<td>157</td>
</tr>
<tr>
<td>5.10.1 Performance</td>
<td>157</td>
</tr>
</tbody>
</table>
Chapter 6. Utilities .................................................. 199
6.1 Utility CPU reduction ........................................... 200
  6.1.1 CHECK INDEX performance ............................... 200
  6.1.2 LOAD performance ......................................... 201
  6.1.3 REBUILD INDEX performance ............................ 204
  6.1.4 REORG performance ........................................ 205
  6.1.5 RUNSTATS index performance ............................ 208
  6.1.6 Index key generation improvements ...................... 210
6.2 MODIFY RECOVERY enhancements ............................ 212
6.3 RUNSTATS enhancements ...................................... 214
  6.3.1 Histogram statistics ....................................... 214
  6.3.2 CLUSTERRATIO enhancements ............................ 216
6.4 Recovery enhancements ........................................ 218
  6.4.1 BACKUP and RESTORE SYSTEM ......................... 218
  6.4.2 RECOVER utility enhancements for point-in-time recovery 221
6.5 Online REBUILD INDEX enhancements ........................ 223
  6.5.1 Performance ................................................ 224
  6.5.2 Conclusion ................................................ 225
  6.5.3 Recommendations ......................................... 225
6.6 Online REORG enhancement .................................... 226
  6.6.1 Performance ................................................ 227
  6.6.2 Conclusion ................................................ 228
  6.6.3 Recommendations ......................................... 229
  6.6.4 Online LOB REORG ........................................ 229
6.7 Online CHECK DATA and CHECK LOB ......................... 230
  6.7.1 Online CHECK DATA ....................................... 230
  6.7.2 Online CHECK LOB ........................................ 231
  6.7.3 Recommendations ......................................... 232
6.8 TEMPLATE switching ............................................ 232
  6.8.1 Performance ................................................ 232
6.9 LOAD COPYDICTIONARY enhancement ........................ 233
6.10 COPY performance ............................................. 233
6.11 Best practices ................................................ 235
  6.11.1 Recommendations for running the LOAD utility .......... 236
  6.11.2 Recommendations for running the REBUILD INDEX utility 236
  6.11.3 Recommendations for running the REORG utility .......... 236
  6.11.4 Recommendations for running the COPY utility .......... 237
## Contents

**Appendix A. Summary of relevant maintenance** .......................................................... 291  
A.1 Performance enhancements APARs ..................................................................... 292  
A.2 Functional enhancements APARs ..................................................................... 298  
A.3 z/OS APARs ........................................................................................................ 303  

**Appendix B. Statistics report** .............................................................................. 305  
B.1 OMEGAMON XE Performance Expert statistics report long layout .................. 306  
B.2 OMEGAMON XE Performance Expert accounting report long layout .......... 322  

**Appendix C. EXPLAIN tables** ............................................................................... 327  
C.1 DSN_PLAN_TABLE ............................................................................................... 328  
C.2 DSN_STATEMNT_TABLE ...................................................................................... 334  
C.3 DSN_FUNCTION_TABLE ....................................................................................... 335  
C.4 DSN_STATEMENT_CACHE_TABLE .................................................................... 336  
C.5 New tables with DB2 9 for z/OS .................................................................... 341  

**Appendix D. INSTEAD OF triggers test case** ....................................................... 343  
D.1 INSTEAD OF trigger DDL ................................................................................ 344  
D.2 INSTEAD OF trigger accounting .................................................................... 345  

**Appendix E. XML documents** ............................................................................. 351  
E.1 XML document decomposition ....................................................................... 352  
  E.1.1 XML document ............................................................................................. 352  
  E.1.2 XML schema .................................................................................................. 354  
  E.1.3 DDL for decomposition ................................................................................ 368  
E.2 XML index exploitation ...................................................................................... 368  
  E.2.1 XML index definitions .................................................................................. 368  
  E.2.2 EXPLAIN output ........................................................................................... 369  

**Abbreviations and acronyms** .............................................................................. 371  

**Related publications** ............................................................................................ 373  
IBM Redbooks ........................................................................................................... 373  
Other publications .................................................................................................... 374  
Online resources ........................................................................................................ 375  
How to get Redbooks ................................................................................................. 375  
Help from IBM ............................................................................................................ 375  

**Index** .................................................................................................................... 377
## Figures

2-1 Explain table changes comparison incorporating global optimization ........................................ 16
2-2 Global optimization comparison .................................................................................................. 17
2-3 Explain table changes for global optimization ............................................................................. 18
2-4 Comparison of base SQL and MERGE for dynamic SQL .............................................................. 21
2-5 Comparison of base SQL and MERGE for static SQL ................................................................. 21
2-6 Comparison of base SQL and SELECT FROM MERGE for dynamic SQL .................................. 22
2-7 Comparison of base SQL and SELECT FROM MERGE for static SQL ....................................... 23
2-8 Comparison of mass DELETE and TRUNCATE ............................................................................ 28
2-9 INSERT and SELECT using a table with 1 column ........................................................................ 42
2-10 INSERT and SELECT using a table with 20 columns ................................................................. 42
2-11 BINARY and VARBINARY performance of INSERT of one million rows ................................ 44
2-12 BINARY and VARBINARY performance of SELECT of one million rows .............................. 45
2-13 Performance comparison INSERT DECFLOAT versus DECIMAL ........................................... 47
2-14 Performance comparison SELECT of DECFLOAT versus DECIMAL ........................................ 48
2-15 INSERT CPU overhead for index on expression .......................................................................... 54
2-16 Gaps in ranges ................................................................................................................................ 55
2-17 Sparse, dense, and nonexistent values ....................................................................................... 56
3-1 XML standards ............................................................................................................................. 64
3-2 DB2 V9 new XML configuration .................................................................................................. 65
3-3 XML nodes ...................................................................................................................................... 67
3-4 Implicitly and explicitly created objects for an XML column definition .................................... 68
3-5 Batch INSERT performance ........................................................................................................ 70
3-6 Results of the cost of indexing when parsing ............................................................................. 71
3-7 Decomposition results .................................................................................................................. 73
3-8 Update performance .................................................................................................................... 75
3-9 Fetch performance ....................................................................................................................... 76
3-10 Index exploitation performance ................................................................................................ 78
3-11 XML compression performance ................................................................................................ 79
4-1 OLTP workload improvement DB2 9 versus DB2 V8 ................................................................. 83
4-2 Comparison of CPU per commit .................................................................................................. 85
4-3 The System z10 ............................................................................................................................... 86
4-4 CPU time improvement ratio from z9 to z10 ............................................................................... 89
4-5 The TPC-D like average times with 3 CPs ................................................................................... 91
4-6 The zEnterprise System ................................................................................................................. 92
4-7 Benefits of System zEnterprise to DB2 ...................................................................................... 93
4-8 IBM Smart Analytics Optimizer .................................................................................................... 94
4-9 DB2 V8 and DB2 V9 real storage usage with test distributed workload ....................................... 103
4-10 Example of shared memory addressing .................................................................................... 104
4-11 DB2 V8 and DB2 V9 DIST address space usage below-the-bar comparison ............................ 107
4-12 DISPLAY THREAD(\*) TYPE SYSTEM output ......................................................................... 110
4-13 Sample DSNV508I, DSNV510I, and DSNV512I messages ......................................................... 111
4-14 DISPLAY THREAD(\*) SERVICE STORAGE output ................................................................. 111
4-15 Accounting classes % CPU overhead for 60 query application ............................................... 119
4-16 Accounting classes % CPU overhead for 100 package query application ............................. 120
4-17 The basic row format ................................................................................................................ 121
4-18 The reordered row format ........................................................................................................ 122
4-19 SQL PL comparison .................................................................................................................... 130
4-20 DB2 index structure .................................................................................................................... 132
4-21 Index and data structure .............................................. 133
4-22 Preformatting improvements ...................................... 134
4-23 Comparison of complex select and hardware support ........ 138
4-24 DRDA redirect using RMF ........................................... 140
4-25 DRDA redirect using OMEGamon Performance Expert ......... 141
4-26 RMF report showing a zIIP redirect% estimate from PROJECTCPU=YES .......................... 141
4-27 Tivoli OMEGamon DB2PE Accounting Report for utility workload zIIP redirect ............... 142
4-28 DB2 V9 synergy with new I/O hardware ......................... 143
4-29 DB2 sequential prefetch ........................................... 147
4-30 Synchronous I/O .................................................... 147
4-31 List prefetch (microseconds) ...................................... 148
4-32 Optimization Service Center statistics collection overhead. 150
5-1 Index split roughly 50/50 ............................................ 155
5-2 Asymmetric index page splits ...................................... 156
5-3 Number of index entries per leaf page for the non-clustered partitioning index .......... 159
5-4 Number of getpages and buffer updates for the non-clustered partitioning index .......... 160
5-5 Number of index entries per leaf page for the clustered non-partitioned index ........... 161
5-6 Number of getpages and buffer updates for the clustered non-partitioned index ........... 161
5-7 Number of getpages and buffer updates at application level ................................ 162
5-8 Class 3 suspensions in a non-data sharing environment ....... 164
5-9 Number of index entries per leaf page for the non-clustered partitioning index .......... 165
5-10 Number of index entries per leaf page for the clustered non-partitioned index .......... 165
5-11 Number of index entries per leaf page for the DSI index ........................................ 165
5-12 Latch class 6 contention in a two-way data sharing ............ 166
5-13 Coupling facility CPU utilization ................................ 168
5-14 Class 2 CPU time .................................................... 168
5-15 Class 2 elapsed time in a two-way data sharing environment . 169
5-16 Class 3 suspensions time in a two-way data sharing environment ............................ 170
5-17 NOT LOGGED scenario ............................................. 174
5-18 Sequential prefetch throughput .................................... 177
5-19 Preformat throughput ............................................... 177
5-20 Elapsed time for SQL with heavy sort activities - sort cost ............................................. 179
5-21 CPU time for SQL with heavy sort activities - sort cost ..................................................... 180
5-22 Insert records into a declared global temporary table .......... 180
5-23 SELECT COUNT(*) from a declared global temporary table .............................................. 181
5-24 LOB insert performance class 2 elapsed and CPU time and class 3 wait time ............. 184
5-25 LOB insert performance class 1 elapsed and CPU time ....................................................... 184
5-26 LOB select performance class 1 and 2 elapsed times .............. 185
5-27 LOB performance select of varying size ........................................ 185
5-28 Plan and package performance DB2 V7 versus DB2 V8 versus DB2 V9 ....................... 190
5-29 Positioned updates and deletes with optimistic concurrency control ......................... 191
5-30 Optimistic locking class 2 CPU time ................................ 192
5-31 REBIND PACKAGE() PLANMGMGT(OFF) vs. REBIND PACKAGE() ................................. 196
5-32 REBIND PACKAGE() PLANMGMGT(BASIC) vs. REBIND PACKAGE() ............................ 196
5-33 REBIND PACKAGE() PLANMGMGT(EXTENDED) vs. REBIND PACKAGE() ......................... 197
5-34 REBIND PACKAGE() SWITCH(PREVIOUS) vs. REBIND PACKAGE() .............................. 197
5-35 REBIND PACKAGE() SWITCH(ORIGINAL) vs. REBIND PACKAGE() ............................ 197
6-1 CHECK INDEX SHRLEVEL REFERENCE results ...................... 201
6-2 CPU improvement on LOAD utility ................................ 202
6-3 CPU improvement of the LOAD utility with dummy input ........ 203
6-4 CPU improvement on REBUILD INDEX ........................... 205
6-5 CPU improvement on REORG utility ................................ 206
6-6 CPU improvement on REORG INDEX ............................... 208
# Examples

2-1 Simple SQL statement demonstrating the new GROUP BY functionality. .......................... 12
2-2 SQL statement illustrating the new DISTINCT processing. ............................................ 13
2-3 Query example illustrating the new optimization. ......................................................... 16
2-4 Correlated subquery ................................................................................................. 18
2-5 Base case SQL flow and program logic MERGE equivalent ........................................ 20
2-6 MERGE case SQL flow. ......................................................................................... 20
2-7 Base case program logic SELECT FROM MERGE equivalent .................................... 22
2-8 SELECT FROM MERGE flow. .................................................. 22
2-9 Base case - SQL for V9 conversion mode ................................................................. 24
2-10 New case - SQL for V9 new-function mode .............................................................. 24
2-11 SQL example for ORDER BY and FETCH FIRST n ROWS in subselect .................... 27
2-12 Query example for sparse index - DEGREE ANY .................................................. 30
2-13 Query example for sparse index - DEGREE 1 ........................................................ 31
2-14 INSTEAD OF trigger SQL ............................................................. 39
2-15 Relevant program logic (PL/I) for insertion of rows into the view ......................... 40
2-16 BIGINT examples .................................................................................. 41
2-17 Restriction on DECFLOAT key column. ................................................................. 46
2-18 Inserting into DECFLOAT data type .................................................................... 47
2-19 Selecting with DECFLOAT conversion ............................................................... 48
3-1 Fetching XML data .................................................................................. 76
3-2 Statements for test cases ............................................................................... 77
4-1 Storage monitor messages ............................................................................... 98
4-2 SDSNMACS(DSNDQISE) ...................................................................... 99
4-3 Statistics report sample. .................................................................................. 100
4-4 z/OS DISPLAY VIRTSTOR,HVSHARE output .......................................... 105
4-5 Virtual storage layout above the bar ............................................................. 106
4-6 DIS THREAD(*) TYPE(SYSTEM) ...................................................... 112
4-7 DIS THREAD(*) SERVICE(WAIT) ..................................................... 113
4-8 DIS THREAD(*) SERVICE(STORAGE) ............................................. 113
4-9 Class 3 suspension report ....................................................................... 115
4-10 Latch classes report .............................................................................. 115
4-11 New resource unavailable information ......................................................... 126
4-12 Sample sequential processing program to access the data ......................... 133
4-13 Simple testcase with single DECFLOAT(16) <-> DECFLOAT (34) casting ...... 137
4-14 Complex testcase with multiple DECFLOAT(16) <-> DECFLOAT (34) castings ... 137
5-1 Sample DDL for the partition-by-range table .................................................... 157
5-2 Sample DDL for a non-clustered partitioning index ........................................ 158
5-3 Sample DDL for a non-partitioned index .......................................................... 158
5-4 Sample DDL for DPSI ........................................................................ 158
5-5 Finding existing package copies ................................................................. 195
6-1 LOAD COPYDICTIONARY example ..................................................... 233
7-1 Creating and invoking a SOAP UDF ............................................................. 247
9-1 DSNFIAU - LOBFILE option .............................................................. 260
9-2 Query for shadow data sets ........................................................................ 271
B-1 Statement for the statistics report ............................................................... 306
B-2 Sample of the statistics report long layout .................................................... 306
B-3 Statement for the accounting report ............................................................ 322
B-4 Sample of the accounting report long layout ............................................ 322
<table>
<thead>
<tr>
<th>C-1 Creating DSN_STATEMENT_CACHE_TABLE</th>
<th>337</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-1 Create table space, table, index and view</td>
<td>344</td>
</tr>
<tr>
<td>D-2 INSTEAD OF trigger.</td>
<td>345</td>
</tr>
<tr>
<td>D-3 PL/I logic.</td>
<td>345</td>
</tr>
<tr>
<td>D-4 Accounting Trace Long for INSTEAD of TRIGGER</td>
<td>345</td>
</tr>
<tr>
<td>E-1 XML document</td>
<td>352</td>
</tr>
<tr>
<td>E-2 XML schema.</td>
<td>354</td>
</tr>
<tr>
<td>E-3 DDL for the INSERT test case.</td>
<td>368</td>
</tr>
<tr>
<td>E-4 DDL for index definition</td>
<td>368</td>
</tr>
</tbody>
</table>
## Tables

2-1 Measurement for the new GROUP BY sort ............................................. 12
2-2 Measurement for new DISTINCT process ........................................... 13
2-3 Comparison of global query optimization improvements ....................... 17
2-4 Global optimization improvements .................................................... 18
2-5 Comparison of multiple SQL statements and a single SQL statement for SELECT
FROM UPDATE or DELETE - Singleton SELECT ........................................ 25
2-6 Comparison of multiple SQL statements and a single SQL statement for SELECT
FROM UPDATE or DELETE - Multiple rows ............................................ 25
2-7 Class 2 CPU comparison of mass DELETE and TRUNCATE ....................... 28
2-8 Comparison of sparse index query - DEGREE ANY ................................. 30
2-9 Comparison for sparse index query - DEGREE 1 .................................. 31
2-10 New BW workload (100 queries) .................................................... 33
2-11 Existing BW workload (100 queries) ............................................... 33
2-12 New columns in DSN_STATEMENT_CACHE_TABLE ............................... 38
2-13 Explicit object CREATE/DROP ....................................................... 49
2-14 IMPLICIT object CREATE/DROP .................................................... 49
2-15 Index on expression CPU comparison .............................................. 53
2-16 Without index on expression ........................................................ 53
2-17 With index on expression ............................................................. 53
2-18 Performance improvement using file reference variables during INSERT .... 58
2-19 Performance improvement using file reference variables during SELECT .... 58
3-1 Test environment details ............................................................... 70
3-2 The UNIFI payment messages ....................................................... 71
3-3 Insert without and with validation .................................................. 72
4-1 Workload characteristics .............................................................. 89
4-2 V8 and V9 virtual storage comparison under workload ......................... 96
4-3 CACHEDYN_FREELOCAL settings .................................................... 98
4-4 V8 versus V9 real storage comparison under workload ....................... 102
4-5 DB2 V9 and DB2 V9 64-bit CPU usage with test distributed workload .......... 108
4-6 Tabulated data for relative CPU% increase for 100 package query application 119
5-1 Class 2 CPU time DB2 V8 versus DB2 V9 - Ascending index key order .... 163
5-2 Class 2 CPU time DB2 V8 versus DB2 V9 - Random index key order ...... 163
5-3 Insert rate (ETR) in two-way data sharing - Ascending index key order ... 170
5-4 Insert rate (ETR) in two-way data sharing - Random index key order ...... 171
5-5 SELECT COUNT(*) of 100,000 rows using index scan .......................... 172
5-6 REBUILD INDEX 100,000 rows with a key size of 20 bytes .................. 173
5-7 Workload using logged versus not logged ....................................... 175
5-8 LOB operations and locks ............................................................ 182
5-9 LOB select performance class 1 and class 2 CPU times in seconds ......... 185
5-10 Class 1 elapsed time - CLOB versus varchar ................................... 186
6-1 Details of the CHECK INDEX SHRLEVEL REFERENCE measurements .... 200
6-2 Details of workload 1 - LOAD ...................................................... 202
6-3 Details of workload 2 - LOAD PART REPLACE .................................. 203
6-4 Details of workload - REBUILD INDEX .......................................... 204
6-5 Details of workload - REORG TABLESPACE .................................... 206
6-6 Details of workload - REORG INDEX .............................................. 207
6-7 Details of RUNSTATS (index) measurements .................................... 209
6-8 Details of the utility test cases ...................................................... 210
6-9  Performance of frequency and histogram statistics collection .......................... 215
6-10  Details of the workload used for online REBUILD INDEX ................................. 224
6-11  Online REORG with 1, 2, and 5 NPIs defined ................................................. 228
6-12  Elapsed time comparison between V9 OLR and V8 OLR with REORG NPI or NPIs 229
6-13  Summary of COPY TABLESPACE measurements .................................................. 234
6-14  Best practices for REORG PART with SHRLEVEL CHANGE .............................. 237
9-1   Growth of DB2 catalog ....................................................................................... 264
A-1   DB2 V9 current performance-related APARs ....................................................... 292
A-2   DB2 V9 current function-related APARs .............................................................. 298
A-3   z/OS DB2-related APARs ................................................................................... 303
C-1   PLAN_TABLE columns by release ...................................................................... 328
C-2   PLAN_TABLE contents and brief history ............................................................... 329
C-3   EXPLAIN enhancements in DSN_STATEMENT_TABLE ................................. 335
C-4   DSN_FUNCTION_TABLE extensions .................................................................. 336
C-5   Contents of DSN_STATEMENT_CACHE_TABLE ............................................. 339
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
Trademarks

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

AIX®
BladeCenter®
CICS®
Cube Views®
DB2 Connect™
DB2®
Distributed Relational Database Architecture™
Domino®
DRDA®
DS8000®
Enterprise Storage Server®
ESCON®
eServer™
FICON®
FlashCopy®
HiperSockets™
HyperSwap®
i5/OS®

IBM®
IMS™
Informix®
iSeries®
Lotus®
MQSeries®
MVS™
OMEGAMON®
OS/390®
pureXML®
QMF™
Query Management Facility™
RACF®
Rational®
Redbooks®
Redbooks (logo) ©
Resource Measurement Facility™

RETAIN®
RMF™
Solid®
Sysplex Timer®
System Storage®
System z10®
System z9®
Tivoli®
WebSphere®
z/Architecture®
z/OS®
z/VM®
z/VSE™
z10™
z9®
zSeries®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

SAP NetWeaver, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

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

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

DB2® 9 for z/OS® is an exciting new version, with many improvements in performance and little regression. DB2 V9 improves availability and security, as well as adds greatly to SQL and XML functions. Optimization improvements include more SQL functions to optimize, improved statistics for the optimizer, better optimization techniques, and a new approach to providing information for tuning. V8 SQL procedures were not eligible to run on the IBM® System z9® Integrated Information Processor (zIIP), but changing to use the native SQL procedures on DB2 9 makes the work eligible for zIIP processing. The performance of varying length data can improve substantially if there are large numbers of varying length columns. Several improvements in disk access can reduce the time for sequential disk access and improve data rates.

The key DB2 9 for z/OS performance improvements include reduced CPU time in many utilities, deep synergy with IBM System z® hardware and z/OS software, improved performance and scalability for inserts and LOBs, improved SQL optimization, zIIP processing for remote native SQL procedures, index compression, reduced CPU time for data with varying lengths, and better sequential access. Virtual storage use below the 2 GB bar is also improved.

This IBM Redbooks® publication provides an overview of the performance impact of DB2 9 for z/OS, especially performance scalability for transactions, CPU, and elapsed time for queries and utilities. We discuss the overall performance and possible impacts when moving from version to version. We include performance measurements that were made in the laboratory and provide some estimates. Keep in mind that your results are likely to vary, as the conditions and work will differ.

In this book, we assume that you are somewhat familiar with DB2 V9. See DB2 9 for z/OS Technical Overview, SG24-7330, for an introduction to the new functions.

In this book we have used the new official DB2 9 for z/OS abbreviated name for IBM Database 2 Version 9.1 for z/OS as often as possible. However, we have used the old DB2 V9 notation for consistency when directly comparing DB2 V9 to DB2 V8 or previous versions.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), San Jose Center.

Paolo Bruni is a DB2 Information Management Project Leader at the ITSO, San Jose Center. He has authored several IBM Redbooks documents about DB2 for z/OS and related tools. He has conducted workshops and seminars worldwide. During Paolo's many years with IBM, in development, and in the field, his work has been mostly related to database systems.

Kevin Harrison is a Certified Senior I/T Architect with IBM Software Group: Information Management. He is responsible for pre- and post-sales technical consulting for DB2 on System z in the America's west region. He has 27 years of involvement with large systems and DB2, with 18 of those years at IBM as a DB2 technical specialist. He has served as a DB2 educator, led many DB2 design and implementation reviews, as well as DB2 and System z performance and tuning engagements, serves as an IBM liaison to several regional DB2 user groups, and is certified in several disciplines for DB2 on System z. He is currently a
member of the IBM North American IT Architect Certification board, IBM zChampions, and IBM Americas Data Management technical competency. Kevin holds a degree in Organic Chemistry from Southwest Missouri State University.

Garth Oldham is a Senior Systems Management Integration Specialist in IBM Australia. He is currently specializing in the DB2 systems programming area. He has 28 years of systems programming experience in a number of z/OS related fields. Garth holds a degree in Biology from the University of York in the United Kingdom.

Leif Pedersen is a Certified IT Specialist in IBM Denmark. He has 20 years of experience in DB2. His areas of expertise include designing and developing business applications running against DB2, DB2 system administration, DB2 performance, disaster recovery, DB2 external security, and DB2 for z/OS in a distributed database environment. As a DB2 instructor, he also teaches many of the DB2 for z/OS classes. Leif previously coauthored the book DB2 for z/OS and OS/390 Version 7 Performance Topics, SG24-6129.

Giuseppe Tino is a DB2 System Programmer with IBM Global Technology Services, Securities Industry Services (SIS) in Toronto, Canada. He has been part of the SIS organization for the last five years providing support for DB2 on z/OS at both the system and application levels. His areas of focus include the installation, maintenance, and performance analysis of multiple DB2 data sharing environments. In addition, Giuseppe holds a degree in Electrical Engineering from Ryerson University.

The authors from left to right: Garth Oldham, Paolo Bruni, Giuseppe Tino, Kevin Harrison, and Leif Pedersen
Special thanks to Catherine Cox, manager of the DB2 Performance Department in IBM Silicon Valley Lab, for making this project possible.

Thanks to the following people for their contributions to this project:

Rich Conway
Emma Jacobs
Bob Haimowitz
Deanna Polm
Sangam Racherla
International Technical Support Organization

Jeffrey Berger
Meg Bernal
Frank Butt
John Campbell
Ying Chang
Hsiuying Cheng
Catherine Cox
Paramesh Desai
Marko Dimitrijevic
David Dossantos
Willie Favero
James Guo
Akiko Hoshikawa
Gopal Krishnan
Laura Kunioka-Weis
Allen Lebovitz
Ching Lee
Chao-Lin Liu
Claire McFeely
Roger Miller
Todd Munk
Mai Nguyen
Mary Petras
Vivek Prasad
Terry Purcell
Akira Shibamiya
Bryan Smith
Yumi K. Tsuji
Frank Vitro
Maryela Weihrauch
Dan Weis
Chung Wu
Li Xia
David L. Zhang
Guogen Zhang
Silicon Valley Laboratory
Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- Use the online Contact us review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an e-mail to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400
Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7473-00
for DB2 9 for z/OS Performance Topics
as created or updated on September 28, 2012.

September 2007, First Edition

The revisions of this First Edition, first published on September 11, 2007, reflect the changes and additions described below.

March 2008, First Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

Changed information
- Corrected the surname of an author under the team photo (sorry Garth!)
- Corrected the description of Figure 2-9 on page 42 and Figure 2-10 on page 42.
- Replaced 7.3.2 Developers Workbench with 7.3.2, “IBM Data Studio” on page 248.
- Update information on performance related APARs in Appendix A, “Summary of relevant maintenance” on page 291.
- Updated the referenced bibliography at “Related publications” on page 373.

New information
- Added section 4.3, “z10 and DB2 workload measurements” on page 85.
- Added section 5.14, “Package stability” on page 192.
- Added PTF numbers and text at 7.2, “MQ Messaging Interfaces user-defined function” on page 244.
- Added performance and new function related APARs in Appendix A, “Summary of relevant maintenance” on page 291.

September 2008, Second Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

Changed information
- Updated section 5.14, “Package stability” on page 192.
- Updated performance and new function related APARs in Appendix A, “Summary of relevant maintenance” on page 291.
- Updated the bibliography in “Related publications” on page 373.
New information

- Added a restrictions paragraph in 2.13.3, “DECFLOAT” on page 45.
- Added information on z1™ performance in 4.3, “z10 and DB2 workload measurements” on page 85.
- Added MVS™ APAR in 4.10, “WLM assisted buffer pool management” on page 108.
- Added a comment on DGTT in 4.16.1, “Workfile sizing” on page 126.
- Added the restriction that a DECFLOAT column cannot be used in an index in 4.20.3, “Hardware support for the DECFLOAT data type” on page 136.
- Added a small section on AMP in 4.20.5, “DASD improvements” on page 142.
- Added information on the data clustering new methods of RUNSTATS in 6.3.2, “CLUSTERRATIO enhancements” on page 216.
- Added APAR PK62027 in 8.7, “Reduction in LOB locks” on page 255.
- Added a Note in “DB2 9 modes: An overview” on page 262 to inform that the DB2 term compatibility mode has been changed to conversion mode. This change has not been applied throughout the book yet to avoid unnecessary change bars.
- Added DB2 performance and new function related APARs in Appendix A, “Summary of relevant maintenance” on page 291.
- Added a new z/OS APARs table in Appendix A, “Summary of relevant maintenance” on page 291.

March 2009, Third Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

Changed information

- Updated performance and new function related APARs in Appendix A, “Summary of relevant maintenance” on page 291.
- Updated the bibliography in “Related publications” on page 373.
- Changed compatibility mode to conversion mode without change bars.

New information

- Added a footnote in 1.1, “DB2 9 for z/OS” on page 3.
- Added the new maximum limit for the number of implicit databases in 2.14, “Autonomic DDL” on page 49.
- Added section 2.15, “APPEND YES option in DDL” on page 50.
- Added section “HyperPAV” on page 143.
- Added section “Solid-state drives” on page 146.
- Added a pointer to recent APARs in 5.9, “WORKFILE database enhancements” on page 178.
- Added some information in 5.14, “Package stability” on page 192.
- Added section 6.9, “LOAD COPYDICTIONARY enhancement” on page 233.
- Added information for LASTUSED column in 8.8, “Index improvements” on page 255.
- Added section 9.3.6, “To rebind or not to rebind” on page 267.
- Added performance and new function related APARs in Appendix A, “Summary of relevant maintenance” on page 291.

April 2009, Fourth Update

This revision reflects the addition, deletion, or modification of new and changed information described below.
**Changed information**
- Removed the previous section 2.4 “Optimization for a complex query” to reflect APARs PK74778, PK77060, and PK79236.
- Updated “Premigration work” on page 266 to state 10000 archive log volumes (not data sets) are supported.
- Updated APARs in Appendix A, “Summary of relevant maintenance” on page 291.

**New information**
- Added a note in 2.14, “Autonomic DDL” on page 49 to mention UNLOAD APAR PK60612 to help with simple table space deprecation.
- Added a note in 5.14.4, “Performance” on page 195 on SPT01 compression APAR PK80375.

**November 2009, Fifth Update**
This revision reflects the addition, deletion, or modification of new and changed information described below.

**Changed information**
- Deleted text about tracing at 2.18.2, “FETCH CONTINUE” on page 58.
- Updated Figure 5-29 on page 191.
- Updated APARs in Appendix A, “Summary of relevant maintenance” on page 291.

**New information**
- Added z/OS PTF and DB2 APAR for “WLM assisted buffer pool management” on page 108.
- Added a paragraph on DSNZPARM SPRMRRF and LOAD REPLACE and REORG TABLESPACE ROWFORMAT parameter at 4.14, “Reordered row format” on page 121.
- Added reference to DSNZPARM SPRMRRF in 5.1, “Universal table space” on page 152 and listed recommended APARs.
- Added a Note in 5.10.3, “Recommendations” on page 187.
- Added DSNZPARMs OPTIOWGT, OPTIXOPREF, and SPRMRRF in 9.4, “DSNZPARM changes” on page 273.
- Added section “DSNZPARM change for the DSN6FAC macro” on page 276 to discuss the new PRIVATE_PROTOCOL DSNZPARM.
- Added APARs in Appendix A, “Summary of relevant maintenance” on page 291.
- Added a note in Appendix C, “EXPLAIN tables” on page 327 about old EXPLAIN table formats being deprecated.

**July 2010, Sixth Update**
This revision reflects the addition, deletion, or modification of new and changed information described below.

**Changed information**
- Updated section 5.1, “Universal table space” on page 152 by removing charts and changing recommendations.
- Updated APARs in Appendix A, “Summary of relevant maintenance” on page 291.

**New information**
- Added section 4.4, “The IBM zEnterprise System and DB2” on page 91.
- Added a note for DSN1COPY APAR PM03691 at the end of 4.14, “Reordered row format” on page 121.
- Added considerations to section 5.1, “Universal table space” on page 152.
- Added INSERT APARs in 5.4.4, “Recommendations” on page 171.
- Added APARs in Appendix A, “Summary of relevant maintenance” on page 291.

### July 2011, Seventh Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

#### Changed information
- Updated section 2.11.2, “Conclusion” on page 38 changing REOPT AUTO to be available in conversion mode.
- Updated section 2.17.2, “Recommendation” on page 56 changing histogram statistics to be available in conversion mode.
- Updated section 4.7, “Distributed 64-bit DDF” on page 104 to remove mention of ECSA.
- Updated section 6.3.1, “Histogram statistics” on page 214 changing to conversion mode.
- Updated APARs in Appendix A, “Summary of relevant maintenance” on page 291.

#### New information
- Added APARs in Appendix A, “Summary of relevant maintenance” on page 291.

### September 2011, Eighth Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

Change bars reflect these last updates.

#### Changed information
- Updated section 5.12, “Package performance” on page 189 to clarify the measurements.
- Updated APARs in Table A-1, Table A-2, and Table A-3 of Appendix A, “Summary of relevant maintenance” on page 291.

#### New information
- Added MINSTOR to the changed DSNZPARAMs in “DSNZPARAM changes for the DSN6SPRM macro”. on page 274.
- Added APARs in Table A-1, Table A-2, and Table A-3 of Appendix A, “Summary of relevant maintenance” on page 291.

### September 2012, Ninth Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

Change bars reflect these last updates.

#### Changed information
- Updated section 2.10.5, “Conclusion” on page 36 to clarify the scope of the EXCEPT enhancement.
Overview of DB2 9 for z/OS

In this chapter, we briefly review the features of DB2 9 for z/OS as they are presented in this book. We also anticipate comparisons and recommendations that we present later in the book. The topics that we cover in this chapter include:

- DB2 9 for z/OS
- SQL enhancements
- XML
- DB2 subsystem
- Availability and capacity
- Utility performance
- Networking and e-business
- Data sharing enhancements
- Installation and migration
- Performance tools
1.1 DB2 9 for z/OS

The official name of the thirteenth version of DB2 for z/OS is *DB2 Version 9.1 for z/OS*, program number 5635-DB2. Notice that Universal Database (nor UDB) is no longer in the name, and 9.1 indicates version and release as for the other platforms. The abbreviated name is DB2 9 for z/OS. However, we often used the old format of DB2 V9 in this book, especially when comparing to V8. You will find V9.1 in the new DB2 manuals.

For more introductory information about the functions that are presented in this book, see *DB2 9 for z/OS Technical Overview*, SG24-7330, and *DB2 Version 9.1 for z/OS What's New?*, GC18-9856. Other references are listed in “Related publications” on page 373. In addition, you can find a wealth of reference material for DB2 9 on the Web at the following address:

http://www.ibm.com/software/data/db2/zos/

DB2 for z/OS V8 delivered loads of new SQL functionality, converted the catalog to a native UNICODE format, and restructured the engine for 64-bit addressing. It also introduced the two-step migration process, which involves the compatibility mode (now called *conversion mode*) and the *new-function mode*. This process is also used to migrate from V8 to V9.

DB2 9 for z/OS concentrates on user requirements in four major areas:

- Enabling high volume processing for the next wave of Web applications
- Enhancing the traditional DB2 for z/OS strengths of reliability, availability, scalability, and performance
- Reducing total cost of ownership (TCO)
- Enhancing DB2 for data warehousing

SQL capabilities have expanded to help the productivity and portability of application developers and vendor packages. The new SQL MERGE statement can be used to insert or update DB2 data more easily and efficiently. Other SQL capabilities include SELECT FROM UPDATE/DELETE/MERGE, a new APPEND option for fast INSERT, a new SQL statement called TRUNCATE, and FETCH CONTINUE for retrieving large objects (LOBs) faster. New data types include DECIMAL FLOAT, BIGINT, BINARY, and VARBINARY. New Data Definition Language (DDL) statements, such as an implicit CREATE TABLE, help you run DDL unchanged when porting between platforms.

DB2 9 adds a native XML data type that allows applications to store XML documents in a native, hierarchical format. This technology, called pureXML®, greatly helps DB2 customers who are moving to service-oriented architecture (SOA) environments. pureXML technology is common across the IBM DB2 family. DB2 9 for z/OS is a hybrid data server for both XML and relational data. DB2 9 stores XML data in a binary encoded format in a natural hierarchy that is different from relational data. This is native XML, with indexing, query syntax, and schema validation built in.

DB2 for z/OS has traditionally supported reliability, availability, scalability, and performance. Online schema changes now allow you to rename columns, indexes, or schemas. You can quickly replace one copy of a table with another copy that is cloned online. A new universal tablespace organization combines the benefits of segmented and partitioned table spaces. Furthermore DB2 has the option to dynamically add partitions as the table grows.

A new REORG option removes the BUILD2 phase from online reorganization, and an online REBUILD INDEX function is provided.
Regulatory compliance issues can be costly. It helps if functions are provided by DB2. DB2 9 for z/OS adds a new capability for a trusted context and database roles, so that an application server’s shared user ID and password can be limited to work only from a specific physical server. It also allows better auditing and accounting from client systems. Auditing has improved with more granularity in the audit traces that are collected.

DB2 9 can use IBM System z\textsuperscript{1} and tape controllers to encrypt the data that resides on these devices, and System z has advanced capabilities to centrally manage all of the encryption keys. By offloading the encryption work to the storage devices, you can save a lot of server processing power. DB2 9 also encrypts data with Secure Sockets Layer (SSL), by eliminating a point of data interception on the wire.

System-level backup and recovery, which uses volume-level FlashCopy\textsuperscript{®}, was introduced in V8. It has been expanded with V9 to add the ability to use tape for backups and to restore at the object level from volume-level backups.

Specialty engines (processors that free up general computing capacity to improve price/performance), such as IBM Integrated Facility for Linux\textsuperscript{®} (IFL), IBM Internal Coupling Facility (ICF), and System z Application Assist Processor (zAAP) for Java\textsuperscript{™} processing, have seen the addition of System z9 Integrated Information Processor (zIIP). zIIPs targeted three types of work with DB2 V8:

\begin{itemize}
  \item Parallel processing for large complex queries
  \item DB2 utilities for index maintenance
  \item Requests that come into DB2 via TCP/IP and Distributed Relational Database Architecture\textsuperscript{™} (DRDA\textsuperscript{®})
\end{itemize}

In DB2 9, SQL procedures become eligible for zIIPs when called from DRDA.

For data warehousing applications, DB2 9 SQL enhancements include INTERSECT, EXCEPT, RANK, caseless comparisons, cultural sort, and FETCH FIRST in a fullselect. In addition, index compression can save disk space, especially in warehousing systems where typically more and larger indexes are defined.

There are query optimization improvements, and a new capability called Optimization Service Center, which provides a full set of functions to monitor and tune query performance. Query Management Facility\textsuperscript{™} (QMF\textsuperscript{™}) DB 2 9 customers can do drag-and-drop querying, as well as use executive dashboards, data visualization, and enhanced online analytical processing (OLAP) with DB2 Cube Views\textsuperscript{®}.

Most of the new functions provided by DB2 9 were measured in a laboratory environment to verify that they perform as expected and compared to V8 analogous support, when existent. We have grouped these functions, sometimes arbitrarily, in the sections that follow.

### 1.2 SQL enhancements

The SQL enhancements improve queries for data warehousing and reporting and provide large consistency across the DB2 family. More queries can be expressed in SQL with new SQL enhancements. The set operators INTERSECT and EXCEPT clauses make SQL easier

\textsuperscript{1} IBM introduced the industry’s first self-encrypting enterprise tape drive, the IBM System Storage\textsuperscript{®} TS1120, in 2006, followed by Linear Tape Open (LTO) self-encrypting drives that support a wide range of lower-cost tape environments. The IBM System Storage DS8000\textsuperscript{®} with Full Disk Encryption extends this market-proven encryption model to enterprise disk systems to support the security requirements of demanding enterprise environments in a practical and cost-effective manner. See: \url{http://www.ibm.com/jct03001c/systems/storage/solutions/data_encryption/}
to write. OLAP extensions for RANK, DENSE_RANK, and ROW_NUMBER add new capabilities. Two new SQL data manipulation statements, MERGE and TRUNCATE, offer opportunities for improved performance. The new data types DECIMAL FLOAT, BIGINT, BINARY, and VARBINARY can provide better accuracy and portability for your data. Improvements in LOBs provide new function, more consistent handling, and improved performance.

Data volumes continue to increase, while the SQL statements grow more complex. The SQL enhancements provide more opportunities for optimization, and DB2 9 adds optimization enhancements to improve query and reporting performance and ease of use. Improved data is provided for the optimizer, with improved advisory algorithms and a rewritten approach to handling performance information for tuning and for exceptions. Histogram statistics provide better information about non-uniform distributions of data when many values are skewed, rather than just a few. Improved algorithms widen the scope of optimization.

Out of the many new functions detailed in Chapter 2, “SQL performance” on page 11, you can look at the following functions to obtain performance improvements from your existing applications after you migrate to the DB2 9 new-function mode:

- DISTINCT and GROUP BY enhancements
- Dynamic prefetch enhancement for regular index access during an SQL call
- Global query optimization
- Optimization for complex query
- Generalized sparse indexes and in memory data caching
- Dynamic index ANDing for star join query
- LOB performance

1.3 XML

DB2 9 transforms the way XML information is managed by integrating XML with relational data. The DB2 new pureXML feature has revolutionized support for XML data by combining the management of structured and unstructured data to allow you to store XML data in its native format.

In Chapter 3, “XML” on page 61, we first describe the infrastructure of DB2 objects that support this new data type. Then we report and draw conclusions from measurements on XML documents of different sizes. As expected, the document size is the primary gauging factor for good performance, both when retrieving and inserting a document.

Compression of XML documents for efficient storage and transmission is fully supported and recommended for space savings and manageability.

1.4 DB2 subsystem

System z synergy is one of the key factors in improving performance. DB2 uses the latest improvements in hardware and operating system to provide better performance, improved value, more resilience, and better function. Faster fiber channels and improved IBM System Storage DS8000 performance provide faster data access, and DB2 9 makes adjustments to improve I/O performance more. FlashCopy can be used for DB2 backup and restore. DB2 makes unique use of the z/Architecture® instruction set, which has new long-displacement instructions and better performance for the long-displacement instructions on the latest processors. DB2 continues to deliver synergy with data and index compression. The z/OS Workload Manager (WLM) improvements will help manage DB2 buffer pools.
When you first move to DB2 9, we expect overall performance to improve for customers who are running System z9 990 (z990) and 890 (z890) processors. Utility performance improvements contribute as soon as you migrate. Take advantage with reorganization and collecting improved histogram statistics. Then REBIND your primary packages and adjust DSNZPARMs. Larger improvements come when you move to new-function mode and make design changes, such as new indexing options for compression, index on expression, and larger page sizes. Native SQL procedures, added use of zIIP, and improved SQL continue the improvements.

Insert performance increases substantially through a wide range of improvements. Logging performance is improved with latching improvements and striped archive logging. The newer disk and channel changes, such as DS8000 Turbo, 4 Gigabit per second channels, and MIDAW, improve data rates substantially. Indexes are improved with larger page sizes to reduce the number of page splits and with a better page split. Where performance should be optimized for inserts, rather than for later retrieval, the append option can be used. If the data needs to be randomized to avoid insert hot spots, the new randomized index key is useful.

Memory improvements continue the work from V8, with memory shared above the bar between the distributed data facility (DDF) and DBM1 address spaces. Shared memory can be used to avoid moving data from one address space to the other. More data structures from the environmental descriptor manager (EDM) pool and dynamic statement cache are moved above the bar. We anticipate about 10% to 15% memory improvements or 200 MB to 300 MB in below-the-bar space, but customers need to monitor and manage still.

In Chapter 4, “DB2 subsystem performance” on page 81, we discuss several topics that are related to enhancements that affect DB2 subsystem performance:

- CPU utilization in the DB2 engine and in the client/server area with DB2 9
  Most of the workloads benefit from CPU and elapsed time improvement.
- Virtual storage constraint relief
  DB2 9 has extended VSCR.
- Changes in real storage utilization patterns in DB2 9
  We explain what has changed and the impact that the changes can have on your processing environment.
- Distributed 64-bit DDF
  VSCR in DB2 9 is extended to include the DIST address space.
- Improved throughput of the distributed workload
- WLM assisted buffer pool management
  This is a new promising technique, but is not yet consolidated, to manage buffer pool sizes via WLM depending on utilization.
- A monitor to checks for automatic identification of latch contention and DBM1 virtual storage excessive usage below the bar
  The monitor provides messages based on different thresholds.
- A number of common latch contention problems addressed in DB2 9
- Reduction of accounting trace overhead
  Such overhead can be reduced by choosing a better granularity, which allows accounting-level detail that is appropriate for your environment.
- Reordered row format, which provides CPU savings when you use variable length columns
Several Buffer Manager enhancements and the merge of workfile and TEMP databases that beneficially affect your system

The creation of SQL procedures as native SQL procedures, which can be redirected to the zIIP if coming from DRDA

DB2 changes on index look-aside and enhanced preformatting

These changes improve data and index access performance.

New data types, which take advantage of hardware enhancements

The Optimization Service Center monitor, which is light in CPU utilization

1.5 Availability and capacity

DB2 9 for z/OS continues to bring changes that improve availability, keeping up with the explosive demands of e-business, transaction processing, and business intelligence. DB2 9 also delivers increased capacity and scalability with more functions to allow changes online and to reduce resource utilizations. The segmented space structure is more efficient in many situations. Therefore, adding a new segmented space structure for partitioned table spaces helps DB2 scale more efficiently. The new partition-by-growth table space is the default table space in DB2 9. Changes to allow DB2 to create the needed databases and table spaces permit DB2 to scale more effectively.

In Chapter 5, “Availability and capacity enhancements” on page 151, we discuss these improvements that are implemented to reduce inhibitors to the full exploitation of faster and more powerful hardware. In particular, there are several improvements in the area of INSERT, UPDATE, and DELETE intensive workloads.

This chapter discusses the following concepts:

- Universal table space
- Clone table
- Object-level recovery
- Relief for sequential key insert
- Index compression
- Log I/O enhancements
- Not logged table spaces
- Prefetch and preformatting enhancements
- WORKFILE database enhancements
- LOB performance enhancements
- Spatial support
- Package performance
- Optimistic locking
- Package stability

1.6 Utility performance

There have been many improvements to the utilities in DB2 9. First, the utilities have all been enhanced to support all new functions in DB2 9, such as universal table spaces, XML data type, CLONE table spaces, compressed indexes, NOT LOGGED table spaces, and so on.

Utility CPU reduction is the first improvement that customers will notice immediately in DB2 9. We have seen substantial performance improvements in the utilities, with some early
customers noting 20% to 30% reductions in CPU time. The primary improvements are in index processing. In general, you can probably obtain larger improvements if you have more indexes.

Here are examples of our measurements:

- 0 to 15% Copy Tablespace
- 5 to 20% in Recover Index, Rebuild Index, and Reorg Tablespace/Partition
- 5 to 30% in Load
- 20 to 60% in Check Index
- 35% in Load Partition
- 30 to 50% in Runstats Index
- 40 to 50% in Reorg Index
- Up to 70% in Load Replace Partition with NPIs and dummy input

One exception to the CPU time improvement is online reorganization for a partition with non-partitioning indexes. Eliminating the BUILD2 phase provides a dramatic improvement in availability, but can increase the CPU time and the elapsed time when one or a few partitions are reorganized. For this process, the non-partitioning indexes are copied to the shadow data set. There is a smaller percentage improvement than those shown in the previous bullets for LOAD, REORG and REBUILD if the work is using a zIIP on both V8 and V9.

Besides the CPU reduction, in Chapter 6, “Utilities” on page 199, we describe the performance measurements for the functional improvements that are specific to the individual utilities. The chapter includes a description of the following utility-related topics:

- MODIFY RECOVERY enhancements
- RUNSTATS enhancements
- Recovery enhancements
- Online REBUILD INDEX enhancements
- Online REORG enhancement
- Online CHECK DATA and CHECK LOB
- TEMPLATE switching
- COPY performance

We also add a section on best practices, which is generally applicable to V8 and V9 of DB2.

### 1.7 Networking and e-business

Technology allows traditional host systems to run Web applications that are built in Java and accommodate the latest business requirements. Java Web applications need the services of a driver to access the database server. DB2 Version 8 provided a universal driver that uses the DRDA protocol to access data on any local or remote server. The universal driver has enhanced usability and portability. DB2 9 for z/OS continues to add functions that are related to better alignment with the customer's strategic requirements and general connectivity improvements.

Chapter 7, “Networking and e-business” on page 241, provides a description of the following performance-related topics:

- Network trusted context
- MQ Messaging Interfaces user-defined function (UDF)
- SOA

Customers who are working on the Web and SOA can benefit from using DB2 9 for z/OS.
1.8 Data sharing enhancements

No specific workload measurements for data sharing were implemented. However several individual improvements are provided by DB2 9 for z/OS. The main focus is on high availability. Chapter 8, “Data sharing enhancements” on page 251, describes these improvements to data sharing. The high-availability improvements are achieved by a combination of performance, usability, and availability enhancements.

The following topics are significant availability improvements and mostly relate to restarting a failed DB2 member:

- Data sharing logging improvement
- Reduction in LOB locks
- Improved group buffer pool write performance
- Improved WLM routing based on DB2 health
- Improved workload balancing within the same logical partition (LPAR)
- Group buffer pool dependency removal by command
- Open data set ahead of use via command
- Enhanced messages when unable to get physical locks (P-locks)

Data sharing also takes advantage of the enhancement that is related to index management:

- Index compression and greater than 4 KB pages for indexes
- Sequential key insert performance improvement
- Ability to randomize index keys that give less contention

1.9 Installation and migration

The two-step migration, with the conversion mode and new-function mode, is in place for V8 to V9 migration. In Chapter 9, “Installation and migration” on page 259, we provide the major performance related issues for migration. We include process timings for the steps that are necessary to move from conversion mode through to the new-function mode.

We discuss the following topics:

- Installation verification procedure (IVP) sample program changes
- Installation
- Migration
- Catalog consistency and integrity checking
- DSNZPARM changes

1.10 Performance tools

DB2 Tools for z/OS help reduce manual tasks, automatically generate utility jobs, capture and analyze performance data, and make recommendations to optimize queries. In addition, they maintain high availability by sensing and responding to situations that could result in database failures and system outages. For information about DB2 tools, see the IBM DB2 and IMS™ Tools Web page:

http://ibm.com/software/data/db2imstools/

In Chapter 10, “Performance tools” on page 279, we briefly introduce the enhanced and new performance tools that are made available with DB2 9 for z/OS:
- IBM Tivoli® OMEGAMON® XE for DB2 Performance Expert on z/OS
- IBM Optimization Service Center and Optimization Expert for z/OS
Chapter 2. SQL performance

DB2 9 for z/OS accelerates the progress in SQL, with many new functions, statements, and clauses. For example, there are new SQL data manipulation statements in MERGE and TRUNCATE. There are new data types with DECFLOAT, BIGINT, BINARY, and VARBINARY types. Improvements in large objects (LOBs) provide more consistent handling and improved performance. Data definition consistency and usability are improved. DB2 V9 is a major leap in DB2 family consistency and in the ability to port applications to DB2 for z/OS.

In this chapter, we look at performance aspects of the SQL enhancements. This chapter contains the following sections:

- DISTINCT and GROUP BY enhancements
- Dynamic prefetch enhancement for regular index access during an SQL call
- Global query optimization
- MERGE and SELECT FROM MERGE
- SELECT FROM UPDATE or DELETE
- FETCH FIRST and ORDER BY in subselect and fullselect
- TRUNCATE SQL statement
- Generalized sparse indexes and in-memory data caching
- Dynamic index ANDing for star join query
- INTERSECT and EXCEPT
- REOPT AUTO
- INSTEAD OF triggers
- BIGINT, VARBINARY, BINARY, and DECFLOAT
- Autonomic DDL
- APPEND YES option in DDL
- Index on expression
- Histogram statistics over a range of column values
- LOB performance
2.1 DISTINCT and GROUP BY enhancements

DB2 9 delivers enhancements for both DISTINCT and GROUP BY performance under the following situations:

- Sort enhancement for both DISTINCT and GROUP BY with no column function
  
  In DB2 9, a sort processing change allows group collapsing to occur in the sort input phase. Group collapsing removes the duplicates from the tree in the input phase and eliminates the requirement for a subsequent merge pass in the sort.
  
  This enhancement was already available for GROUP BY with the column function.

- DISTINCT sort avoidance with a non-unique index
  
  Prior to DB2 9, DISTINCT can only use a unique index to avoid a sort, where GROUP BY can also use a duplicate index for sort avoidance.
  
  In DB2 9, DISTINCT can use a duplicate index for duplicate removal without performing a sort.

2.1.1 Performance with group collapsing

In V8, grouping is done after sort input processing. In V9, DB2 applies the group collapsing optimization for the GROUP BY query without a column function and for DISTINCT. The result is fewer workfile getpages and less CPU time.

The SQL statement in Example 2-1 performs a tablespace scan of 1.65 million rows followed by a sort for the GROUP BY into nine different groups.

**Example 2-1** Simple SQL statement demonstrating the new GROUP BY functionality

```
SELECT COVGTYPE FROM COVERAGE GROUP BY COVGTYPE
```

The performance measurements in Table 2-1 show the improvements for the query with GROUP BY in Example 2-1 due to the new sort avoidance and group collapsing enhancements.

**Table 2-1** Measurement for the new GROUP BY sort

<table>
<thead>
<tr>
<th></th>
<th>V8</th>
<th>V9</th>
<th>Percent improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getpages (work file)</td>
<td>26051</td>
<td>6</td>
<td>99</td>
</tr>
<tr>
<td>CPU (seconds)</td>
<td>9.0</td>
<td>5.8</td>
<td>36</td>
</tr>
</tbody>
</table>

As shown in this example, and confirmed by other tests in the lab, the new GROUP BY sort reduces an overall query CPU time by 35% to 40% when the number of groups is such that all can fit in a single tournament sort tree, thereby eliminating any subsequent merge pass. When the number of groups is large enough to spill over into work files, around a 10% improvement may be observed based upon lab measurements.

The following rules of thumb apply:

- For fewer groups (that is, more duplicates), a higher percent of CPU reduction is obtained.
- For more groups (that is, fewer duplicates), a lower percent of CPU reduction is obtained.
2.1.2 Performance for DISTINCT sort avoidance

In V8, a DISTINCT query cannot use a non-unique index to avoid sort for DISTINCT. Only a unique index can be used. In V9, a DISTINCT query can have sort avoidance via a non-unique index.

The SQL statement in Example 2-2 performs a tablespace scan in V8 of 624,000 rows followed by a sort of 24,984 values for the DISTINCT. V9 is able to exploit the non-unique index on column SANM.

*Example 2-2  SQL statement illustrating the new DISTINCT processing*

```sql
SELECT DISTINCT SANM FROM POLICY
```

The measurement results in Table 2-2 show the improvements for the query with DISTINCT in Example 2-2 due to the new sort avoidance.

*Table 2-2  Measurement for new DISTINCT process*

<table>
<thead>
<tr>
<th></th>
<th>V8</th>
<th>V9</th>
<th>Percent improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getpages (work file)</td>
<td>22234</td>
<td>2772</td>
<td>87.5</td>
</tr>
<tr>
<td>CPU (seconds)</td>
<td>4.93</td>
<td>3.7</td>
<td>25</td>
</tr>
</tbody>
</table>

As a second example, consider the following query with a non-unique index on P_PARTKEY:

```sql
SELECT DISTINCT(P_PARTKEY) FROM PART
ORDER BY P_PARTKEY
FETCH FIRST 10 ROWS ONLY;
```

In V8, a non-matching index only scan via a non-unique index, followed by a sort for DISTINCT, results in six million distinct values. In V9, a non-matching index only scan via a non-unique index results in no sort being performed.

Measurement results for this query show a CPU reduction of 2800 times versus V8. It also shows a 100% workfile getpage reduction (0 versus 71,364 in V8) because a sort is avoided.

This query is able to avoid the sort in V9 and to take advantage of the FETCH FIRST n ROWS ONLY enhancement where the workfile allocation is avoided for the FETCH FIRST clause if the result can fit within a single page.

2.1.3 Conclusion

New functionality for index usage and sort improvements provides significant savings in both CPU and getpages for queries that contain GROUP BY or DISTINCT.

*Note:* The DISTINCT and GROUP By enhancements are available in conversion mode. REBIND is required to obtain a sort avoidance benefit.
2.2 Dynamic prefetch enhancement for regular index access during an SQL call

Sequential prefetch is chosen when data access is clustered and the number of pages exceeds a certain threshold. Both of these conditions are not easy to measure. Sequential prefetch, therefore, may be kicked off due to inaccurate estimation. When this occurs, unneeded pages are scheduled, and therefore, there is wasted I/O and CPU time. Sequential prefetch cannot provide optimal performance as dynamic prefetch does in several situations:

- Sequential prefetch is determined at bind time and cannot be adjusted at execution time. Dynamic prefetch is based on sequential detection at run time and is based on access pattern.
- Sequential prefetch does not provide support for backward prefetch, while dynamic prefetch provides support for both forward and backward directions.
- Sequential prefetch is driven by hitting a triggering page, which is an even multiple of 32 pages in an SQL call. If a prefetch triggering page is skipped, the next set of pages must be read one by one until the next triggering page is reached.

There is no triggering page for dynamic prefetch. Dynamic prefetch can switch automatically between multi-page prefetch read and single-page synchronous read as needed. This is effective when the cluster ratio of index or indexes that are used prior to data access is less than 100%.

- Two dynamic prefetch engines can be run in parallel.

Notice the DB2 9 for z/OS has also improved the way the data clustering information is collected by RUNSTATS, see 6.3.2, “CLUSTERRATIO enhancements” on page 216 for details.

By switching to dynamic prefetch, DB2 9 avoids wasteful prefetch I/Os for disorganized indexes and for skip sequential index and/or data page access.

2.2.1 Performance

The dynamic prefetch enhancement is applicable to single table access, multi-table join, outer join, subquery, and union. This enhancement affects the prefetch operations during index scan or table access via index scan in an SQL call. Utilities already used dynamic prefetch.

Lab measurements indicate a query performance with an elapsed time improvement between 5% and 50% for queries that use index scan. There can be up to a 10% reduction of synchronous I/O for data pages and a possible reduction of up to 75% of synchronous I/O for index pages. There is some reduction in CPU time.

**Note:** The dynamic prefetch functionality is available in conversion mode after a REBIND.

2.3 Global query optimization

The purpose of the enhancement to global query optimization is to improve query performance by allowing the DB2 V9 optimizer to generate more efficient access paths for queries that involve multiple parts. The changes are within the DB2 optimizer and the DB2 runtime components.
There is no external function. However, DB2 provides details for the way in which a query that involves multiple parts is performed. Also, since the way in which a query with multiple parts is performed is no longer fixed to the way in which the query was coded, the EXPLAIN output is modified to make it easier to tell what the execution sequence is for these types of queries.

Global query optimization addresses query performance problems that are caused when DB2 breaks a query into multiple parts and optimizes each of those parts independently. While each of the individual parts may be optimized to run efficiently, when these parts are combined, the overall result may be inefficient. This enhancement may be beneficial to several types of applications including enterprise resource planning (ERP).

For example, consider the following query:

```sql
SELECT * FROM T1 WHERE EXISTS (SELECT 1 FROM T2, T3 WHERE T2.C2 = T3.C2 AND T2.C1 = T1.C1);
```

Prior to V9, DB2 breaks this query into two parts: the correlated subquery and the outer query. Each of these parts is optimized independently. The access path for the subquery does not take into account the different ways in which the table in the outer query may be accessed and vice versa.

DB2 may choose to do a table scan of T1, which would result in significant random I/O when accessing T2, while a non-matching index scan of T1 would avoid the random I/O on T2. In addition, DB2 does not consider reordering these two parts. The correlated subquery is always performed after accessing T1 to get the correlation value. If T1 is a large table, and T2 is a small table, it may be much more efficient to access T2 first and then T1, especially if there is no index on T2.C1, but there is an index on T1.C1.

Global query optimization allows DB2 to optimize a query as a whole rather than as independent parts. This is accomplished by allowing DB2 to:

- Consider the effect of one query block on another
- Consider reordering query blocks

Subquery processing is changed due to the new consideration for cross-query block optimization. All subqueries are now processed by the DB2 optimizer differently than before, and the new processing is summarized as follows:

- The subquery itself is represented as a “virtual table” in the FROM clause that contains the predicate with the subquery.
- This “virtual table” may be moved around within the referencing query in order to obtain the most efficient sequence of operations.
- Predicates may be derived from the correlation references in the subquery and from the subquery SELECT list.
- These predicates can be applied to either the subquery tables or the tables that contain the correlated columns depending on the position of the “virtual table”.
- When determining the access path for a subquery, the context in which the subquery occurs is taken into consideration.
- When determining the access path for a query that references a subquery, the effect that the access path has on the subquery is taken into consideration.
2.3.1 Performance

Example 2-3 illustrates a query with an embedded subquery, in bold, where new optimization techniques can be considered.

Example 2-3  Query example illustrating the new optimization

```sql
SELECT O_ORDERSTATUS, COUNT(*) AS ORDER_COUNT
FROM ORDER
WHERE O_ORDERPRIORITY = '1-URGENT'
  AND O_TOTALPRICE <= 17500
  AND O_ORDERKEY IN (SELECT
                       DISTINCT O_ORDERKEY
                       FROM LINEITEM, ORDER
                       WHERE L_ORDERKEY = O_ORDERKEY
                       AND O_ORDERDATE >= DATE('1998-01-01')
                       AND O_ORDERDATE < DATE('1998-01-01') + 1 DAY
                       AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERSTATUS
ORDER BY O_ORDERSTATUS;
```

In this query example, DB2 considers the query as a whole instead of separately. Previously the bold section of the query would have been considered in a query block (QB) by itself.

Figure 2-1 illustrates the changes in the Explain table when a separate query block in V8 becomes part of a join with parent query block in V9. The arrow points to the additional row for a subquery that has correlated and non-correlated forms. DSNWFQB(nn) is the table name (workfile query block). In this case, 02 is the query block number that is associated with the subquery.

The additional column PARENT_PLANNO is used together with PARENT_QBLOCKNO to connect a child query block to the parent miniplan for global query optimization. Instrumentation facility component identifier (IFCID) 22 reflects the value of this new column.

```
+-------------------------------------------------------------------------+
| V8 | QB | PNO | M  | TNAME         | AT | MC | ACNAME       | PQ | PP | QBTYP  |
+-------------------------------------------------------------------------+
| 1_ |  1 |   1 |  0 | ORDER         |  N |  1 | PXO@OKODCKSPOP | 0 | 0 | SELECT |
| 2_ |  1 |   2 |  3 |              |  0 |  0 |             | 0 | 0 | SELECT |
| 3_ |  2 |   1 |  0 | ORDER         |  I |  1 | UXO@CKOKDSEP  | 1 | 1 | NCOSUB |
| 4_ |  2 |   2 |  1 | LINEITEM      |  I |  1 | PXL@OKSDRFSKEPD | 1 | 1 | NCOSUB |
| 5_ |  2 |   3 |  3 |              |  0 |  0 |             | 1 | 1 | NCOSUB |
+-------------------------------------------------------------------------+
```

```
+-------------------------------------------------------------------------+
| V9 | QB | PNO | M  | TNAME         | AT | MC | ACNAME       | PQ | PP | QBTYP  |
+-------------------------------------------------------------------------+
| 1_ |  1 |   1 |  0 | DSNWFQB(02)   |  R |  0 |             | 0 | 0 | SELECT |
| 2_ |  1 |   2 |  4 | ORDER         |  I |  1 | PXO@OKODCKSPOP | 0 | 0 | SELECT |
| 3_ |  1 |   3 |  3 |              |  0 |  0 |             | 0 | 0 | SELECT |
| 4_ |  2 |   1 |  0 | ORDER         |  I |  1 | UXO@CKOKDSEP  | 1 | 1 | NCOSUB |
| 5_ |  2 |   2 |  4 | LINEITEM      |  I |  1 | PXL@OKSDRFSKEPD | 1 | 1 | NCOSUB |
| 6_ |  2 |   3 |  3 |              |  0 |  0 |             | 1 | 1 | NCOSUB |
+-------------------------------------------------------------------------+
```

Figure 2-1  Explain table changes comparison incorporating global optimization

Table 2-3 illustrates the performance improvement for global query optimization with comparison of V8 to V9 and V9 with parallelism as compared to V8 base. In V8, due to the nature of the query, and the inability to incorporate the subquery, parallelism is not applicable. Parallelism was not employed by query block 1 because the optimizer cannot determine the
page ranges and drive query parallelism from a work file. (The result of the subquery is used to probe the outer table.) V9 supports the parallel degrees on the inner table.

Similar considerations apply to INSERT, DELETE, and UPDATE statements with similar types of subqueries with no parallelism.

**Note:** The global query optimization functionality is available in conversion mode and requires a REBIND.

### Table 2-3   Comparison of global query optimization improvements

<table>
<thead>
<tr>
<th></th>
<th>V8</th>
<th>V9</th>
<th>V9 parallel</th>
<th>Percent improvement (V9/V9parallel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed time (seconds)</td>
<td>404.7</td>
<td>220</td>
<td>29</td>
<td>54/717</td>
</tr>
<tr>
<td>CPU time (seconds)</td>
<td>14.3</td>
<td>13.7</td>
<td>15</td>
<td>4.7/-4.8</td>
</tr>
</tbody>
</table>

Figure 2-2 graphically illustrates the savings in Table 2-3 that were incurred when global optimization transforms the same queries and considers the subquery.

![Figure 2-2   Global optimization comparison](image-url)
A second global optimization query example involves a correlated subquery that is not transformed to a join in V8 when parallelism is enabled. In this example, the filtering comes from the subquery table. See Example 2-4.

**Example 2-4  Correlated subquery**

```sql
SELECT *
FROM ACCOUNTS A
WHERE EXISTS
  (SELECT 1
   FROM CUSTOMERS C
   WHERE A.CUSTNO = C.CUSTNO
   AND   C.ZIPCODE = 99999
   AND   C.STATUS = 'N');
```

The V8 execution accesses the outer table first, and then accesses the subquery for each outer row, as outlined in the Explain table output in Figure 2-3.

**Figure 2-3  Explain table changes for global optimization**

The V9 Explain table output shows that the query has been converted to a non-correlated subquery in query block 2. The output from query block 2 is DSNWFQB(02), which is accessed first in query block 1. The subquery result is used to access the ACCOUNTS table.

Therefore, in V9, DB2 is able to convert the query to access the most filtering table first. Table 2-4 highlights the performance improvement for this query due to global optimization in V9.

**Table 2-4  Global optimization improvements**

<table>
<thead>
<tr>
<th></th>
<th>V8</th>
<th>V9</th>
<th>Percent improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed time (seconds)</td>
<td>8</td>
<td>.1</td>
<td>99</td>
</tr>
<tr>
<td>CPU time (seconds)</td>
<td>6.04</td>
<td>.07</td>
<td>99.99</td>
</tr>
</tbody>
</table>

Note that this query is not eligible for parallelism in V9 because it is an extremely short running query.

Queries in V8 that are already able to choose an efficient access path may not see any improvement in V9. However, as demonstrated by the previous examples, global optimization
in V9 is able to consider alternate join sequences and join methods for queries that previously were fixed in their sequence due to V8 subquery to join limitations.

In situations where the preferred sequence was not available, then considerable performance improvements can be achieved with global optimization. This is true of workloads that are heavy users of subqueries, which is common among ERP vendors, since manually rewriting the query is not an option to improve performance.

See Appendix A.1, “Performance enhancements APARs” on page 292, for recent maintenance on this function.

2.4 MERGE and SELECT FROM MERGE

The MERGE statement combines the conditional UPDATE and INSERT operation in a single statement. This provides the programmer with ease of use in coding SQL. It also reduces the amount of communication between DB2 and the application as well as network traffic.

The SELECT FROM MERGE statement combines an UPDATE, followed by an INSERT or an UPDATE, followed by an INSERT with a subsequent SELECT of updated or inserted values that return the possible delta changes.

The statements can be either dynamic or static.

A typical database design has a table that contains knowledge of a domain and transaction data. The transaction data can contain updates to rows in the table or new rows that should be inserted into the table. Prior to V9, applying the changes from the transaction data to the table requires two separate SQL statements: an UPDATE statement for those rows that are already in existence in the table and an INSERT statement for those rows that do not exist.

DB2 V9 has the capability, with a single SQL statement for source data that is represented as a list of host variable arrays, to match rows in the target and perform the UPDATE or to find rows with no match and then INSERT them.

Refer to the DB2 Version 9.1 for z/OS SQL Reference, SC18-9854, for a detailed explanation about the MERGE SQL syntax. Refer to DB2 Version 9.1 for z/OS Application Programming and SQL Guide, SC18-9841, for usage examples.

2.4.1 Performance

For this performance measurement, the environment consisted of the following items:

- For static SQL
  - z/OS 1.7 System z9 processor
  - PL/I
  - Index access
- For dynamic SQL
  - z/OS 1.7 System z9 processor
  - Dynamic SQL using JDBC
  - JDK Version 1.5 [run-time env]
  - Java Common Connectivity (JCC) T4 Driver on z/OS Version 3.3.14 [Java driver]
  - Index access
MERGE
Example 2-5 illustrates a base case SQL flow using SQL to perform the same DML operations that are equivalent to MERGE.

Example 2-5  Base case SQL flow and program logic MERGE equivalent

UPDATE TABLE NAME SET VAL1=:HV_VAL1, VAL2=:HV_VAL2
If record not found (if SQL code >0) then...
INSERT INTO TABLE_NAME (VAL1, VAL2) VALUES (:HV_VAL1, :HV_VAL2)

Example 2-6 illustrates the new SQL flow for using MERGE.

Example 2-6  MERGE case SQL flow

MERGE INTO TABLE-NAME AS A USING (VALUES (:HV_VAL1, :HV_VAL2) FOR N ROWS AS T
  (VAL1, VAL2)
  ON A.VAL1=T.VAL1
  WHEN MATCHED THEN UPDATE SET VAL2 = A.VAL2 + T.VAL2
  WHEN NOT MATCHED THEN INSERT (VAL1, VAL2) VALUES (T.VAL1,T.VAL2);
Figure 2-4 illustrates the performance of dynamic SQL where a base case set of SQL and program logic is compared to the new MERGE functionality. Note that the environments for static and dynamic are different. A comparison across the two environments should not be inferred.

![Dynamic SQL](image)

Figure 2-4   Comparison of base SQL and MERGE for dynamic SQL

Figure 2-5 illustrates the performance of static SQL where a base case set of SQL and program logic is compared to the new MERGE functionality.

![Static SQL](image)

Figure 2-5   Comparison of base SQL and MERGE for static SQL
**SELECT FROM MERGE**

Example 2-7 illustrates a base case SQL flow using SQL to perform the same DML operations that are equivalent to SELECT FROM MERGE.

<table>
<thead>
<tr>
<th>Example 2-7</th>
<th>Base case program logic SELECT FROM MERGE equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Cursor</td>
<td></td>
</tr>
<tr>
<td>Fetch (before values) of the row that will be changed</td>
<td></td>
</tr>
<tr>
<td>Loop for Update - Rec not found</td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td></td>
</tr>
<tr>
<td>End loop</td>
<td></td>
</tr>
<tr>
<td>Fetch (after value)</td>
<td></td>
</tr>
<tr>
<td>Close cursor</td>
<td></td>
</tr>
</tbody>
</table>

Example 2-8 illustrates the new SELECT FROM MERGE flow DML operation.

<table>
<thead>
<tr>
<th>Example 2-8</th>
<th>SELECT FROM MERGE flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT FROM MERGE CASE</td>
<td></td>
</tr>
<tr>
<td>DCL C1 SCROLL CURSOR WITH ROWSET POSITIONING FOR SELECT COL1,COL2,DELTA FROM FINAL TABLE (MERGE INTO TABLE-NAME.......</td>
<td></td>
</tr>
<tr>
<td>OPEN CURSOR</td>
<td></td>
</tr>
<tr>
<td>FETCH NEXT ROWSET</td>
<td></td>
</tr>
<tr>
<td>CLOSE CURSOR</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2-6 illustrates the performance of SQL where a base case SQL and program logic flow is compared to the new SELECT FROM MERGE functionality for dynamic SQL. Note that the environments for static and dynamic are different. A comparison across the two environments should not be inferred.

![Figure 2-6](chart.png)  
*Figure 2-6  Comparison of base SQL and SELECT FROM MERGE for dynamic SQL*
Figure 2-7 illustrates the performance of SQL where a base case SQL and program logic flow is compared to the new SELECT FROM MERGE functionality for static SQL.

![Static SQL Performance Chart]

The overall results of using the new functionality compared to the base case indicate that:

- The static MERGE statement performs 13% better.
- The static SELECT FROM MERGE statement performs 19% better.
- The dynamic MERGE statement performs 20% better.
- The dynamic SELECT FROM MERGE statement performs 9% better.

### 2.4.2 Conclusions

The usage of MERGE and SELECT FROM MERGE provides a powerful and more efficient way of coding SQL than the traditional way of using multiple SQL statements to provide the same functionality. The usage of this new SQL syntax provides better performance, less interaction across the network, less DB2 and application interaction, and less SQL to code.

**Notes:**

- The MERGE operation and the access of the intermediate result table from the MERGE operation does not use parallelism.
- The MERGE and SELECT FROM MERGE operations are available in new-function mode.
2.4.3 Recommendations

Use the new MERGE and SELECT FROM MERGE capability in conjunction with GET DIAGNOSTICS and multiview operations. By doing so, you will benefit from a better performing set of SQL when performing a target row insert or update match against a DB2 table or view.

2.5 SELECT FROM UPDATE or DELETE

The SELECT FROM UPDATE or DELETE closes the gap for DB2 family compatibility:

- DB2 for Linux, UNIX®, and Microsoft® Windows®: SELECT FROM INSERT/UPDATE/DELETE
- DB2 for z/OS V8: SELECT FROM INSERT
- DB2 for z/OS V9: SELECT FROM UPDATE/DELETE

The new syntax for V9 allows SELECT from a searched UPDATE or searched DELETE.

A single statement can now be used to determine the values before or after records are updated or deleted. Work files are used to materialize the intermediate result tables that contain the modified rows.

**Note:** The SELECT FROM UPDATE or DELETE functionality is available in new-function mode.

2.5.1 Performance

Lab measurements were performed with both a simple table space and segmented table space. There was one unique index on the table. Measurement scenarios included singleton SELECT and SELECT with multiple rows returned.

We illustrate an example with DB2 V8 and V9 in conversion mode, which requires the two separate SQL statements as shown in Example 2-9. In this case, we select the old salary and then update it.

**Example 2-9  Base case - SQL for V9 conversion mode**

```
SELECT EMPNO, SALARY FROM EMPLOYEE WHERE EMPNO = 'EMPN0005';
UPDATE EMPLOYEE SET SALARY = SALARY *1.1
     WHERE EMPNO = 'EMPN0005';
```

Example 2-10 illustrates the equivalent example of SQL in V9 where the new SQL functionality is exploited within the confines of a single SQL statement.

**Example 2-10  New case - SQL for V9 new-function mode**

```
SELECT EMPNO, SALARY FROM OLD TABLE
     (UPDATE EMPLOYEE SET SALARY = SALARY *1.1
     WHERE EMPNO = 'EMPN0005');
```

After the execution of this SQL, the final result from the overall SQL query consists of the rows of the target table before all changes were applied.
Table 2-5 illustrates the CPU differences between a set of base SQL statements that are required to perform a set of operations versus the same operation that can be done in new-function mode with a single SQL statement. The CPU values were measured on System z9.

Table 2-5  Comparison of multiple SQL statements and a single SQL statement for SELECT FROM UPDATE or DELETE - Singleton SELECT

<table>
<thead>
<tr>
<th>SQL</th>
<th>CPU in seconds for Example 2-9</th>
<th>CPU in seconds for Example 2-10</th>
<th>Percent change</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT FROM OLD TABLE, UPDATE</td>
<td>0.078</td>
<td>0.099</td>
<td>+27%</td>
</tr>
<tr>
<td>SELECT FROM FINAL TABLE, UPDATE</td>
<td>0.085</td>
<td>0.100</td>
<td>+17%</td>
</tr>
<tr>
<td>SELECT FROM FINAL TABLE, UPDATE, INCLUDE</td>
<td>0.113</td>
<td>0.101</td>
<td>-11%</td>
</tr>
<tr>
<td>SELECT FROM OLD TABLE, DELETE</td>
<td>0.104</td>
<td>0.115</td>
<td>+17%</td>
</tr>
</tbody>
</table>

Table 2-6 illustrates the CPU differences between a set of base SQL statements that are required to perform a set of operations versus the same operation that can be done in new-function mode with a single SQL statement and with multiple rows returned.

Table 2-6  Comparison of multiple SQL statements and a single SQL statement for SELECT FROM UPDATE or DELETE - Multiple rows

<table>
<thead>
<tr>
<th>SQL</th>
<th>V9 Base (conversion mode) CPU in seconds</th>
<th>V9 New (new-function mode) CPU in seconds</th>
<th>Percent change</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT FROM OLD TABLE, UPDATE</td>
<td>0.143</td>
<td>0.168</td>
<td>+18%</td>
</tr>
<tr>
<td>SELECT FROM FINAL TABLE, UPDATE</td>
<td>0.150</td>
<td>0.168</td>
<td>+12%</td>
</tr>
<tr>
<td>SELECT FROM FINAL TABLE, UPDATE, INCLUDE</td>
<td>0.195</td>
<td>0.171</td>
<td>-12%</td>
</tr>
<tr>
<td>SELECT FROM OLD TABLE, DELETE</td>
<td>0.196</td>
<td>0.242</td>
<td>+23%</td>
</tr>
<tr>
<td>SELECT FROM OLD TABLE, DELETE all</td>
<td>0.232</td>
<td>0.543</td>
<td>+134%</td>
</tr>
</tbody>
</table>

The following nomenclature is used in the measurement tables:

- **OLD TABLE** means the original table prior to any action.
- **FINAL TABLE** means the table after any action has occurred.

The include-column clause (INCLUDE column name or names) introduces a list of additional columns that are to be included in the result table of the change (DELETE/UPDATE/INSERT) statements. The included columns are available only if the change statement is nested in the FROM clause of a SELECT statement or a SELECT INTO statement.
A WHERE clause in the change statement cannot contain correlated references to columns that are outside of the statement. The target of the change statement must be a base table, a symmetric view, or a view where the view definition has no WHERE clause. If the searched change statement is used in the SELECT statement and the change statement references a view, the view must be defined using the WITH CASCADED CHECK OPTION clause. AFTER triggers that result in further operations on the target table cannot exist on the target table.

Important: Use SELECT FROM DELETE with segmented table space with caution because mass delete is disabled in SELECT FROM DELETE.

2.5.2 Conclusions

The SELECT FROM UPDATE or DELETE SQL enhancement is a usability feature of DB2 V9 and cannot be compared to DB2 V8 in terms of equivalence. Additional CPU is required to manipulate intermediate results in the work file. SELECT FROM UPDATE or DELETE provides a mechanism to simplify coding and let DB2 handle the logic of the operation that is required instead of programming it in the application or with multiple SQL statements. It can also provide some relief in the number of trips across the network in certain types of applications. It is a usability feature that provides ease of use.

There are other instances where it is possible to have improved performance, for instance, all situations where an UPDATE or DELETE statement requires a table space scan. In V8, issuing two separate SQL statement would repeat the table space scan. We have seen that combining two or more statements into one with SELECT FROM adds workfile overhead. However, in all cases where the workfile overhead is less than the overhead of issuing multiple statements, this enhancement also provides a performance improvement.

2.6 FETCH FIRST and ORDER BY in subselect and fullselect

In DB2 V8, the ORDER BY and FETCH FIRST n ROWS ONLY clauses are only allowed at the statement level as part of a SELECT statement or a SELECT INTO statement. You can specify the clauses as part of a SELECT statement and write:

```
SELECT * FROM T ORDER BY c1 FETCH FIRST 1 ROW ONLY
```

However, you cannot specify the clauses within the fullselect and write:

```
INSERT INTO T1
(SELECT * FROM T2 ORDER BY c1 FETCH FIRST 1 ROW ONLY)
```

Assume that you have a very large table of which you want only the first 2000 rows sorted in a particular order. You would code a SELECT statement using the FETCH FIRST and ORDER BY clauses. The constraining issue is that the sort is done before the FETCH. This would cause a very large sort for no reason. The alternative is to code SQL using a temp table, which is considerable more work than a simple SELECT.

For FETCH FIRST N ROWS, DB2 sort implements a new function to help improve performance by doing the sorting in memory. Only the final results at the end are written to the work file. Normally for FETCH FIRST N ROWS, the user requests a small number of rows to be returned. To take advantage of this in-memory sort, the number of rows multiplied by the size of the data, plus the key, must fit within a 32K page ((data size plus key size) x n rows less than or equal to 32704). Otherwise, the normal tournament sort path is chosen. DB2 V9 allows specification of these new clauses as part of subselect or fullselect.
Example 2-11 shows a sample SQL statement where new functionality is used. The lines in bold indicate the new SQL capability.

**Example 2-11  SQL example for ORDER BY and FETCH FIRST n ROWS in subselect**

```
SELECT EMP_ACT.EMPNO, PROJNO
FROM EMP_ACT
WHERE EMP_ACT.EMPNO IN
  (SELECT EMPLOYEE.EMPNO
   FROM EMPLOYEE
   ORDER BY SALARY DESC
   FETCH FIRST 3 ROWS ONLY)
```

Prior to V9, the following statements were not supported:

- ORDER BY in a subselect
- FETCH FIRST n ROWS in subselect

### 2.6.1 Conclusion

In DB2 V9, equivalent ORDER BY performance in a subselect or a fullselect is observed. Lab measurements show that there can be up to two times reduction in CPU time or elapsed time observed for queries that use FETCH FIRST N ROWS clause. If the query is I/O bound, then either CPU or I/O may improve but generally not both.

**Note:** The FETCH FIRST and ORDER BY functions are available in conversion mode.

### 2.7 TRUNCATE SQL statement

To empty a table, you have to either do a mass delete, using DELETE FROM table name without a WHERE clause, or use the LOAD utility, with REPLACE REUSE and LOG NO NOCOPYPEND. If there is a delete trigger on the table, using the DELETE statement requires you to drop and subsequently recreate the delete trigger to empty the table without firing the trigger. LOAD REPLACE works on a table-space level instead of on a table level. You cannot empty a specific table if the table space that belongs contains multiple tables.

The TRUNCATE statement addresses these problems. The TRUNCATE statement deletes all rows for either base tables or declared global temporary tables. The base table can be in a simple table space, a segmented table space, a partitioned table space, or a universal table space. If the table contains LOB or XML columns, the corresponding table spaces and indexes are also truncated.

TRUNCATE is an effective way to delete all the rows in a designated table without activating delete triggers nor altering the current table attributes in the catalog. DB2 transforms the TRUNCATE table statement into a mass delete operation. By doing so, it takes advantage of the current optimized design and provides greater flexibility for the user to deactivate existing delete triggers and then harden the results of the TRUNCATE without a COMMIT. This performance improvement can be gained only on a table that has triggers defined. Otherwise, it performs as it has previously.
2.7.1 Performance

The TRUNCATE operation can run in a normal or fast way. The way that is chosen depends on the table type and its attributes. Users cannot control which way the TRUNCATE statement is processed.

Eligible table types are simple, partitioned, or segmented. Table attributes that determine the way in which the table is truncated are change data capture (CDC), multiple-level security (MLS), or VALIDPROC.

The normal way of processing implies that the TRUNCATE operation must process each data page to physically delete the data records from that page. This is true in the case where the table is either simple or partitioned regardless of table attributes or a table has CDC, MLS, or a VALIDPROC.

The fast way of processing implies that the TRUNCATE operation deletes the data records without physically processing each data page. This is true in the case where a table is either segmented or in a universal table space with no table attributes.

Table 2-7 lists the class 2 CPU times for mass DELETE and TRUNCATE.

<table>
<thead>
<tr>
<th>Table 2-7</th>
<th>Class 2 CPU comparison of mass DELETE and TRUNCATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mass DELETE (class 2 CPU seconds)</td>
</tr>
<tr>
<td>Partitioned</td>
<td>47.40</td>
</tr>
<tr>
<td>DPSI</td>
<td>47.25</td>
</tr>
<tr>
<td>Segmented</td>
<td>0.045</td>
</tr>
</tbody>
</table>

Figure 2-8 illustrates the equivalence of TRUNCATE and mass DELETE.
2.7.2 Conclusion

TRUNCATE provides equivalent performance to a mass delete operation on tables. TRUNCATE also allows this operation on tables that are defined with a delete trigger without having to drop the trigger.

Note: The TRUNCATE SQL statement is available in new-function mode.

2.8 Generalized sparse indexes and in-memory data caching

DB2 V8 replaced the sparse index with in-memory data caching for star schema queries, with runtime fallback to the sparse index when enough memory is not available. The characteristics of star schema sparse indexes are:

- In-memory index occupying up to 240 KB
- Probed through an equal-join predicate
- Binary search for the target portion of the table
- Sequential search within the target portion if it is sparse

The characteristics of in-memory data caching (also known as in-memory work file) are:

- Memory pool size controlled by DSNZPARM SJMXPOOL
- Entire work file in memory (thus is not sparse)
- Searched using binary search (as per sparse index)

In DB2 V8, in-memory work files are stored in a new dedicated global storage pool that is called a star join pool. The DB2 DSNZPARM SJMXPOOL specifies its maximum size, which defaults to 20 MB (maximum 1 GB). It resides above the 2 GB bar and is in effect only when star join processing is enabled through DSNZPARM STARJOIN. When a query that exploits star join processing finishes, the allocated blocks in the star join pool to process the query are freed.

In DB2 V9, in-memory data caching is extended to joins other than just the star join. DB2 V9 uses a local pool above the bar instead of a global pool. Data caching storage management is associated with each thread and, therefore, potentially reduces storage contention. A new DSNZPARM, MXDTCACH, specifies the maximum size in MB (default 20 MB) of memory for data caching of each thread.

All tables that lack an appropriate index or enough statistics can benefit from sparse index in-memory data caching:

- Base tables
- Temporary tables
- Table expressions
- Materialized views

This new access method and join context compete with other options, and then the most efficient access path is chosen. In-memory data caching access costing is not done in access path selection since data caching is a runtime decision. Instead, optimizer costs the usage of sparse index because this is what is chosen if enough memory is not available for in-memory data cache. If in-memory data caching can be used, it is a bonus because it is better performing than sparse index access.

DB2 V9 supports sparse index with multi-column keys. DB2 V8 only supported single column sparse index (and in-memory data cache), which was inefficient if the join contained multiple
keys. A sparse index or in-memory data caching search may be more efficient when there is more than one join predicate between two tables, because of the support for multi-column keys.

A new IFCID 27 tracks the memory utilization for data caching.

### 2.8.1 Performance

Sparse indexing can now be used for a base table, temporary table, table expression, materialized view, and so on. Sparse index access is now externalized and treated as a general access method (PRIMARY_ACCESTYPE = 'T' in the PLAN_TABLE).

Nested loop join with sparse index (or in-memory data cache) on the inner table is a new alternative for optimizer to consider when there is no viable index on the inner table. Previously, optimizer would consider a nested loop join, with a table space scan of inner, or a merge scan join, which would require a sort of both the new and composite tables. Instead of building a sparse index, in-memory data caching can be built if enough space is available in the agent local storage pool above the bar.

Consider the following examples for sparse index performance improvements. Example 2-12 shows a sample SQL query where the predicates (shown in bold) benefit from the capability of in memory data caching and query parallelism that is used.

**Example 2-12  Query example for sparse index - DEGREE ANY**

```sql
SELECT SUM(C_ACCTBAL * 0.01), MAX(P_RETAILPRICE + P_SIZE)
FROM CUSTOMER, PART
WHERE C_ACCTBAL = P_RETAILPRICE
  AND C_ACCTBAL < 1500.00
  AND C_MKTSEGMENT IN ("MACHINERY", "FURNITURE");
```

Table 2-8 shows the results and performance gains for sparse index improvements and parallelism used. In this example, BP0 contains the tables, BP1 contains the work file, and BP2 contains the indexes.

**Table 2-8  Comparison of sparse index query - DEGREE ANY**

<table>
<thead>
<tr>
<th>DB2</th>
<th>Elapsed time</th>
<th>CPU time</th>
<th>Getpages BP0</th>
<th>Getpages BP1</th>
<th>Getpages BP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>V8</td>
<td>43.86</td>
<td>73.49</td>
<td>404572</td>
<td>187523</td>
<td>0</td>
</tr>
<tr>
<td>V9</td>
<td>21.13</td>
<td>49.22</td>
<td>404572</td>
<td>4886</td>
<td>0</td>
</tr>
</tbody>
</table>

This comparison of the queries shows a relational scan on the PART table with a sort merge join and a relational scan on CUSTOMER (involved a sort composite and sort new) in V8. By contrast in V9, a relational scan is performed on the PART table with a nested loop join, and a relational scan is performed on CUSTOMER (with a sort new) and PRIMARY_ACCESTYPE="T". The measured results show an elapsed time reduction of 52%, a CPU reduction of 33%, and a reduction in BP1 getpages of 97%.
Example 2-13 shows a sample SQL query where the predicates (shown in bold) benefit from the capability of in-memory data caching and no query parallelism is used.

Example 2-13  Query example for sparse index - DEGREE 1

```sql
SELECT COUNT(*)
FROM PART, PARTX
WHERE P_PARTKEY < 1000
  AND P_BRAND = PX_BRAND
  AND P_CONTAINER = PX_CONTAINER_V10;
```

Table 2-9 shows the results and performance gains for sparse index improvements, and parallelism is not used. In this example, BP0 contains the tables, BP1 contains the work file, and BP2 contains the indexes.

Table 2-9  Comparison for sparse index query - DEGREE 1

<table>
<thead>
<tr>
<th>DB2</th>
<th>Elapsed time</th>
<th>CPU time</th>
<th>Getpages BP0</th>
<th>Getpages BP1</th>
<th>Getpages BP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>V8</td>
<td>73.92</td>
<td>37.91</td>
<td>141245</td>
<td>184609</td>
<td>30</td>
</tr>
<tr>
<td>V9</td>
<td>38.19</td>
<td>15.39</td>
<td>141245</td>
<td>59424</td>
<td>30</td>
</tr>
</tbody>
</table>

The comparison of the query results show a relational scan on PARTX with sort, merge, join (SMJ), an index scan on PART (with sort composite and sort new), and merge join cols = 2 for V8. By contrast for V9, an index scan is performed on PART with a nested loop join and a relational scan is performed on PARTX with sort new and PRIMARY_ACCESSSTYPE='T'. The results for V9 show an elapsed time reduction of 48%, a CPU reduction of 59%, and a reduction in getpages for BP1 of 68%.

2.8.2 Conclusion

Queries where there is no supporting index for the join can benefit greatly from the extended usage of in-memory data cache and sparse index in V9. The sparse index is not shared across multiple queries, and therefore, must be built for each execution. Therefore, the usage of a sparse index by the optimizer may identify opportunities for a permanent index to be created. It is important to note that a sparse index is not considered if a user index exists on the join column. However, a sparse index is built after the application of local predicates. Therefore, a preferred index to be created may contain columns from both local and join columns.

Note: Generalized sparse indexes and in-memory data caching are available in conversion mode after a REBIND.

2.9 Dynamic index ANDing for star join query

The new capability of dynamic index ANDing for a star join query implements a new kind of star join methodology, pair-wise join with join back, to check the semantic correctness. It consists of three phases:

- Phase 1: Pair-wise join with join back phase (no rids overflow occurred)
- Phase 2: Fall back plan (rids overflow occurred)
- Phase 3: Runtime optimization
The challenge is to consolidate filtering first, because we could find that the fact table has hundreds of millions of rows. If we are off by a factor of 10%, then this can equate a large number.

Customers want efficient star join logic that uses multi-column indexes in order to fully exploit star join access. Ad hoc data warehouse filtering can come from anywhere. Therefore, customers have difficulty in designing for optimal indexes. Customers also need to keep statistics up-to-date and to actively monitor and tune runaway queries.

Prior to V9, DB2 for z/OS uses the following methodology to support star join:

- Cartesian join on the dimension tables prior to a join with the fact table
- Multi-column index access on the fact table with index feedback

The problems with the current method are:

- The cartesian join can become the cause of degradation if the dimensions that are being joined are still large.
- Users need to create a multi-column index on a fact table with the index columns to support joining from different dimension tables. It is difficult to create suitable multi-column indexes unless the combination of filtering dimensions that are included in the queries are known.

The enhancement with V9 introduces a different join approach within a star join group, called a pair-wise join. It requires a one-column index on a fact table to support each dimension table join column. The new join method relies heavily on the resource of a rid pool.

The pair-wise join process enables parallel execution in pair-wise phase for current DEGREE(1), due to the parallel nature of the pair-wise phase. This allows the exploitation of runtime optimization when current DEGREE(1). DB2 is also able to qualify more queries for a pair-wise join due to single dimension filtering and filtering from a snowflake join instead of a local predicate.

Star schema access path determination is now performed using a set of heuristic rules that compares cost model outcomes. From the cost model, a selection is determined that looks at each of three categories:

- Pushdown = star join (JOIN_TYPE=S)
- Non-pushdown = non-star join (JOIN_TYPE=blank)
- Pair-wise join = JOIN_TYPE=P

The result of applying these rules picks one of these three types of plans.

### 2.9.1 Performance

Based on the queries from the new and existing workloads, performance measurements are realized that show a significant improvement in elapsed time and addition System z9 Integrated Information Processor (zIIP) CPU redirect. Comparison of the CPU between the old and new Business Warehouse (BW) workloads should be considered in the context of the zIIP redirect, increased parallelism capability, and reduction in elapsed time.
Table 2-10 illustrates the improvement in the new BW workload where the new star join processing is used.

**Table 2-10  New BW workload (100 queries)**

<table>
<thead>
<tr>
<th></th>
<th>DB2 V8</th>
<th>DB2 V9</th>
<th>Percent improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total elapsed time</td>
<td>70638</td>
<td>8100</td>
<td>88</td>
</tr>
<tr>
<td>(seconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total CPU (seconds)</td>
<td>7211</td>
<td>7126</td>
<td>1.1</td>
</tr>
<tr>
<td>CPU zIIP eligible</td>
<td>1127 (15.63%) projected</td>
<td>5543 (77.8%) projected</td>
<td></td>
</tr>
</tbody>
</table>

This new BW workload is defined as:
- SAP BW database populated with SAP benchmark BW 3.5 toolkits
- Fact table size: 58.4 million rows, 8 indexes and 1 multi-column index, others are single column
- Dimension tables: 8 (2 ~ 99326 rows)
- Snowflakes: Five were added to increase query complexity

One hundred queries are defined and developed by SVL DB2 development and performance. These are based on DB2 V8 BW workload. New queries were added to better reflect the query performance challenges learned from V8. Additional queries that should benefit from the new pair-wise join method are also in this workload; therefore, they use more parallelism and have more CPU time eligible for zIIP redirect. These workloads represent customer workloads without adequate (multi-column) index support. This is typical in customer environments.

Table 2-11 illustrates the improvement in the existing BW workload where the new star join processing is used.

**Table 2-11  Existing BW workload (100 queries)**

<table>
<thead>
<tr>
<th></th>
<th>DB2 V8</th>
<th>DB2 V9</th>
<th>Percent improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total elapsed time</td>
<td>21320</td>
<td>12620</td>
<td>41</td>
</tr>
<tr>
<td>(seconds)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total CPU (seconds)</td>
<td>6793</td>
<td>5351</td>
<td>21</td>
</tr>
<tr>
<td>CPU zIIP eligible</td>
<td>2568 (37.4%) projected</td>
<td>3328 (62%) projected</td>
<td></td>
</tr>
</tbody>
</table>

This old workload is defined as a set of databases that are populated with SAP benchmark BW 3.5 toolkits:
- Fact table size: ~15,000,000 rows
- Dimensions: 8

One hundred queries are developed by SVL DB2 development and performance. Eighty of these queries are based on an existing standard benchmark workload. This old workload represents a well-tuned workload with adequate index support. It also contains queries that use the V8 star join method. It contains additional new queries that better reflect some of the challenges learned in V8. This workload represents a customer workload with a well-tuned index design, which is not typical in the field due to the cost of an index and the lack of tuning skills.
2.9.2 Conclusion

Key messages for this enhancement are:

- V9 performs better than V8 for BW types of workloads that have a well-tuned index design.
- V9 outperforms V8 for BW workloads with multi-column indexes. See the new BW workload for details.
- Increased parallelism results in greater zIIP offload in V9 for BW workload, and some parallelism is possible even for DEGREE(1).
- V9 improves the solution that is ready to use, which reduces the burden of index design and query tuning from users.
- Overall, V9 offers a significant total cost of ownership (TCO) reduction when running a BW type of workload.

Note: Dynamic index ANDing for a star join query is available in conversion mode and requires a REBIND.

2.10 INTERSECT and EXCEPT

The UNION, EXCEPT, and INTERSECT clauses specify the set operators union, difference, and intersection. UNION is already supported. For DB2 family compatibility, DB2 9 for z/OS introduces EXCEPT and INTERSECT. To combine two or more SELECT statements to form a single result table, use one of the following key words:

- **UNION**
  This clause returns all of the values from the result table of each SELECT statement. If you want all duplicate rows to be repeated in the result table, specify UNION ALL. If you want redundant duplicate rows to be eliminated from the result table, specify UNION or UNION DISTINCT.

- **EXCEPT**
  This clause returns all rows from the first result table (R1) that are not also in the second result table (R2). If you want all duplicate rows from R1 to be contained in the result table, specify EXCEPT ALL. If you want redundant duplicate rows in R1 to be eliminated from the result table, specify EXCEPT or EXCEPT DISTINCT.

EXCEPT and EXCEPT ALL are an alternative to using subqueries to find orphan rows, that is, rows that are not picked up by inner joins.

- **INTERSECT**
  This clause returns rows that are in the result table of both SELECT statements. If you want all duplicate rows to be contained in the result table, specify INTERSECT ALL. If you want redundant duplicate rows to be eliminated from the result table, specify INTERSECT or INTERSECT DISTINCT.

INTERSECT DISTINCT and EXCEPT DISTINCT are functionally equivalent to WHERE EXISTS and WHERE NOT EXISTS respectively. There is no comparative function for INTERSECT ALL and EXCEPT ALL.
In the following examples, we compare the performance of the new functions in DB2 V9 and the V8 way of coding SQL.

2.10.1 INTERSECT DISTINCT versus WHERE EXISTS (table space scan)

The following example shows the new function:

DCL C1 CURSOR FOR (SELECT COL1 FROM R1) INTERSECT DISTINCT (SELECT COL1 FROM R2);
DO WHILE (SQLCODE=0)
FETCH NEXT
Base case example:
DCL C1 CURSOR FOR SELECT DISTINCT COL1 FROM R1 WHERE EXISTS (SELECT COL1 FROM R2
WHERE R1.COL1=R2.COL1);
DO WHILE (SQLCODE=0)
FETCH NEXT

There is a result set of four rows for each case. The comparison of these SQL coding styles results in an improvement of 13% when using the new functionality of INTERSECT DISTINCT.

2.10.2 INTERSECT DISTINCT versus WHERE EXISTS (IX ACCESS)

The following example shows the new function:

DCL C1 CURSOR FOR (SELECT COL1 FROM R1) INTERSECT DISTINCT (SELECT COL1 FROM R2);
DO WHILE (SQLCODE=0)
FETCH NEXT

Here is an example with the base case:
DCL C1 CURSOR FOR SELECT DISTINCT COL1 FROM R1 WHERE EXISTS (SELECT COL1 FROM R2
WHERE R1.COL1=R2.COL1);
DO WHILE (SQLCODE=0)
FETCH NEXT

There is a result set of four rows for each case. The new function SQL results in an improvement of 34% when using the new functionality of INTERSECT DISTINCT and an index access is used.

2.10.3 EXCEPT DISTINCT versus WHERE NOT EXISTS (TS SCAN)

The following example shows the new function:

DCL C1 CURSOR FOR (SELECT COL1 FROM R1) EXCEPT DISTINCT (SELECT COL1 FROM R2);
DO WHILE (SQLCODE=0)
FETCH NEXT

Here is an example with the base case:
DCL C1 CURSOR FOR SELECT DISTINCT COL1 FROM R1 WHERE NOT EXISTS (SELECT COL1 FROM R2
WHERE R1.COL1=R2.COL1);
DO WHILE (SQLCODE=0)
FETCH NEXT

Both examples return a result set of two rows. In this scenario, the new function SQL yields a +2.4% regression for the usage of the EXCEPT DISTINCT versus WHERE NOT EXISTS when a table space scan is performed.
2.10.4 EXCEPT DISTINCT versus WHERE NOT EXISTS (IX ACCESS)

The following example shows the new function:

```
DCL C1 CURSOR FOR (SELECT COL1 FROM R1) EXCEPT DISTINCT (SELECT COL1 FROM R2);
DO WHILE (SQLCODE=0)
  FETCH NEXT
```

Here is an example with the base case:

```
DCL C1 CURSOR FOR SELECT DISTINCT COL1 FROM R1 WHERE NOT EXISTS (SELECT COL1 FROM R2 WHERE R1.COL1=R2.COL1);
DO WHILE (SQLCODE=0)
  FETCH NEXT
```

Both examples return a result set of two rows. In this scenario, the new function SQL yields a 34% improvement for the usage of the EXCEPT DISTINCT versus WHERE NOT EXISTS when there is index access.

2.10.5 Conclusion

Consider the usage of the new INTERCEPT and EXCEPT functionality in DB2 9 SQL when queries are required to combine two or more SELECT statements to form a single result set. The performance for these new operators can be much better than the existing way when a large percentage of the result set needs to be processed or there is no good index to support the usage of a correlated subquery. An example is the usage of EXCEPT rather than a correlated NOT EXISTS subquery when there is no good index to support the lookup to the correlated NOT EXISTS subquery.

Note: The new INTERCEPT and EXCEPT functionality is available in new-function mode.

2.11 REOPT AUTO

For dynamic SQL statements, DB2 determines the access path at run time, when the statement is prepared. This can make the performance worse than that of static SQL statements. However, if you execute the same SQL statement often, you can use the dynamic statement cache to decrease the number of times that those dynamic statements must be prepared.

Despite all the enhancements in DB2 optimization, the host variable impact on dynamic SQL optimization and execution is still visible and can result in less than efficient access paths. Customers require DB2 to come up with the optimal access path in the minimum number of prepares.

The BIND option REOPT specifies if DB2 will determine the access path at run time by using the values of SQL variables or SQL parameters, parameter markers, and special registers.

For dynamic SQL, DB2 V8 has REOPT(NONE), REOPT(ONCE) and REOPT(ALWAYS).

- **NONE** specifies that DB2 does not determine the access path at run time by using the values of SQL variables or SQL parameters, parameter markers, and special registers. NONE is the default.
- **ONCE** specifies that DB2 determine the access path for any dynamic SQL statements only once, at the first time the statement is opened. This access path is used until the
prepared statement is invalidated or removed from the dynamic statement cache and need to be prepared again.

- ALWAYS specifies that DB2 always determines the access path at run time each time an SQL statement is run.

Just as a reminder, static SQL only supports REOPT(NONE) and REOPT(ALWAYS).

The option WITH KEEP DYNAMIC specifies that DB2 keeps dynamic SQL statements after commit points. If you specify WITH KEEP DYNAMIC, the application does not need to prepare an SQL statement after every commit point.

If you specify WITH KEEP DYNAMIC, you must not specify REOPT(ALWAYS). WITH KEEP DYNAMIC and REOPT(ALWAYS) are mutually exclusive. However, you can specify WITH KEEP DYNAMIC and REOPT(ONCE).

DB2 V9 introduces the new reoptimization option REOPT(AUTO).

REOPT(AUTO) can specify whether dynamic SQL queries with predicates that contain parameter markers are to be automatically reoptimized when DB2 detects that one or more changes in the parameter marker values renders dramatic selectivity changes. The newly generated access path replaces the current one and can be cached in the statement cache. Consider using this functionality especially when queries are such that a comparison can alternate between a wide range or a narrow range of unknown host variable values.

REOPT(AUTO) can reduce the number of prepares for dynamic SQL (both short and full) when used for queries that have host variable range changes. It may also improve execution costs in a query such as:

```
SELECT ... FROM T1 WHERE C1 IN (?, ?, ?) ...
```

Here T1 is a large table that uses different host variables for successive executions with host variable values that provide good filtering.

If you specify REOPT(AUTO), DB2 automatically determines if a new access path is required to further optimize the performance of a statement for each execution. REOPT(AUTO) applies only to dynamic statements that can be cached. If dynamic statement caching is turned off and DB2 executes a statement that is bound with REOPT(AUTO), no reoptimization occurs.

For dynamic SQL queries with parameter markers, DB2 automatically re-optimizes the statement when DB2 detects that the filtering of one or more of the predicates changes dramatically. The newly generated access path replaces the current one and is cached in the statement cache. DB2 reopts at the beginning and then monitors runtime values supplied for parameter markers.

REOPT(AUTO) decides at OPEN (DEFER PREPARE), depending on the content of host variables, whether to re-prepare a statement. The first optimization is the same as REOPT(ONCE). Based on host variable contents, REOPT(AUTO) may:

- Re-prepare a statement and insert into the global cache (full PREPARE)
- Remain in AUTO mode, checking at each OPEN (short PREPARE), BIND_RO_TYPE = 'A'
- Change to optimum mode, with no more checking at OPEN
Table 2-12 illustrates the new columns for REOPT(AUTO) in the DSN_STATEMENT_CACHE_TABLE.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIND_RA_TOT</td>
<td>Total number of REBINDs that have been issued for the dynamic statement due to REOPT(AUTO)</td>
</tr>
<tr>
<td>BIND_RO_TYPE</td>
<td>'N' REOPT(NONE) or its equivalent, '1' REOPT(ONCE) or its equivalent, 'A' REOPT(AUTO), 'O' Current plan is deemed as optimal: no need for further REOPT(AUTO)</td>
</tr>
</tbody>
</table>

Emphasis should be placed on the fact that re-optimization checking, due to REOPT(AUTO) and the statement in auto mode (BIND_RO_TYPE = 'A'), will result in extra overhead at statement execution regardless of whether the statement is re-optimized. You should consider this cost against the potential total cost of the statement execution. If the statement execution cost is relatively high, then REOPT(AUTO) may be a good candidate. Also note that when in auto mode, if re-optimization occurs, then a full PREPARE is done that has even more added cost at the time of statement execution. Since REOPT(AUTO) applies at the package level, consider isolating the statements that benefit from this function to packages that would be bound with this option while other statements are in packages without this option.

For any other mode, ('O','N', and so on), the REOPT(AUTO) cost at statement execution is negligible.

For statements for which REOPT(AUTO) may result in frequent re-optimization, note that the current implementation has a 20% limitation of re-optimization of the total executions on an individual statement. This means that, if every statement execution would result in re-optimization because of the different disparate host variable values, re-optimization will not necessarily occur due to this limit. For this kind of processing, use REOPT(ALWAYS) to avoid this limitation.

2.11.1 Performance

The performance measurements are impacted by the changes provided by APARs PK47318 and PK49348. Details will be provided when the PTFs will be made available and new measurements will be implemented to show a comparison of CPU overhead for BIND REOPT options.

The reason for the new REOPT function is reported through a field in IFCID 0022 records. The number of the predicate that triggers REOPT is recorded. IFCID 0003 records the number of REOPT times due to the parameter marker value changes at the thread level. The number of REOPTs caused by REOPT(AUTO) is recorded in a new field of IFCID 0316 records.

You can monitor these performance parameters to determine the effective use of REOPT AUTO for your specific environment. This helps to determine the effectiveness, given your specific set of DB2 objects and accompanying statistics and query types for those objects.

2.11.2 Conclusion

Consider using the REOPT(AUTO) bind option to achieve a better balance between the costs of reoptimization and the costs of processing a statement. You might use the REOPT(AUTO)
bind options for many statements for which you can choose either the REOPT(ALWAYS) or REOPT(NONE) bind options, in the following situations:

- The statement is a dynamic SQL statement and can be cached.
- The SQL statement sometimes takes a relatively long time to execute, depending on the values of referenced parameter markers, especially when parameter markers refer to non-uniform data that is joined to other tables. For such SQL statements, the performance gain from a new access path that is chosen based on the input variable values for each SQL execution, may or may not be greater than the performance cost of reoptimization when the statement runs.

There is increase overhead compared to REOPT(ALWAYS). The optimum result of REOPT(AUTO) reverts to the same performance as the query bound with literals, with no checking at each OPEN.

Monitor these performance parameters to determine the effective use of REOPT AUTO for your specific environment. This helps to determine the effectiveness, given your specific set of DB2 objects, accompanying statistics and query types for those objects.

**Notes:** The REOPT AUTO functionality is available in conversion mode. See APARs PK49348 and PK47318 (UK31630) for additional improvements to REOPT AUTO.

### 2.12 INSTEAD OF triggers

INSTEAD OF triggers provide a mechanism to unify the target for all read/write access by an application while permitting separate and distinct actions to be taken for the individual read and write actions. INSTEAD OF triggers are processed instead of the update, delete, or insert operation that activates the trigger.

Unlike other forms of triggers that are defined only on tables, INSTEAD OF triggers can only be defined on views. Views are not deletable, updatable, or insertable if they are read-only. INSTEAD OF triggers provide an extension to the updatability of views. Using INSTEAD OF triggers, the requested INSERT, UPDATE, or DELETE operation against the view is replaced by the trigger logic, which performs the operation against the table on behalf of the view. This happens transparently to the application, which believes all operations are performed against the view.

For additional information, refer to *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854.

#### 2.12.1 Performance

Consider the following example for the definition and execution of the SQL where you might use this new functionality. Details are in Appendix D, “INSTEAD OF triggers test case” on page 343. Example 2-14 illustrates the creation of an INSTEAD OF trigger.

**Example 2-14   INSTEAD OF trigger SQL**

**View Statement:**

```sql
CREATE VIEW EMPLOYEE_VIEW (EMPEMPLN,EMPDEPTN,EMPLEVEL,
    EMPLOYEE_VIEWS,EMPXALRY,EMPLPROJ) AS
SELECT EMPEMPLN,EMPDEPTN,EMPLEVEL,
    EMPLOYEE_VIEWS,EMPXALRY,EMPLPROJ
FROM EMPLOYEE01 WHERE EMPLOYEE01.EMPDEPTN > '077';
```
INSTEAD OF trigger statement:

```
CREATE TRIGGER EMPV_INSERT INSTEAD OF INSERT ON EMPLOYEE_VIEW
REFERENCING NEW AS NEWEMP
FOR EACH ROW MODE DB2SQL
INSERT INTO EMPLOYEE01
  VALUES ('A',NEWEMP.EMPEMPLN,'A',NEWEMP.EMPDEPTN,
    NEWEMP.EMPLEVEL,
    'A',NEWEMP.EMPLTEAM,NEWEMP.EMPSALRY,NEWEMP.EMPLPROJ,
    1,1,1,'A',1,1,'A','A');
```

Example 2-15 illustrates the relevant program logic for insertion of rows into the view.

```
Example 2-15   Relevant program logic (PL/I) for insertion of rows into the view

EMPEMPLN = 'EMPN2000';
DO I = 2 TO 10;
  SUBSTR(EMPEMPLN,8,1) = SUBSTR(EMPTABLE,I,1);
EXEC SQL
  INSERT INTO EMPLOYEE_VIEW
    ( EMPEMPLN,EMPDEPTN,EMPLEVEL,EMPLTEAM,EMPSALRY,EMPLPROJ)
  VALUES (:EMPEMPLN,'078',4,146,75000.00,4);
END;
```

The table that EMPLOYEE _VIEW is based on is created in BP1. The index for the table is in BP2. In the view that is created, 462 rows qualify. Execution of the program logic results in nine rows being inserted into the table and view.

See D.2, “INSTEAD OF trigger accounting” on page 345, which refers to the accounting report for the execution of the INSTEAD OF trigger. NINSERT2 is the base program for execution of the SQL used in the previous examples. Note that DML counts show a total of 18 INSERTS. This indicates an INSERT of nine rows into the base table EMPLOYEE01 and nine rows into the view EMPLOYEE_VIEW. INSTEAD OF trigger EMPV_I#0ER shows as a class 7 consumer. Detailed information in the package accounting section of the report shows the execution characteristics of the INSTEAD OF trigger.

**Note:** The INSTEAD OF trigger employs the use of a work file. While providing new functional capability, some overhead is involved in this type of trigger usage.

### 2.12.2 Conclusion

INSTEAD OF triggers provide a mechanism to simplify coding and let DB2 handle the logic of the operation that is required instead of programming it in the application. It can also provide relief in the number of trips across the network in certain types of applications. It is a usability feature that provides ease of use when requirements exist to execute logic against a view, and it can be handled by a trigger, and therefore, is transparent to the application program.

**Note:** The INSTEAD OF trigger functionality is available in new-function mode.
2.13 BIGINT, VARBINARY, BINARY, and DECFLOAT

For optimal performance, we recommend that you map the Java data types that are used in applications to the DB2 column data types. The main reason is to provide for efficient predicate processing. It also minimizes data type conversion cost.

In DB2 V8, usage of the decimal data type was required for large integers. Some of the issues that need to be addressed are that, in Java, longs are primitive types, which are stack allocated. BigDecimals are heap allocated, which is more expensive; they take up a lot more storage and need to be garbage collected.

Native support of the decimal floating point (DECFLOAT) data type in DB2 9 enables DECFLOAT data to be stored or loaded into DB2 tables; it also allows for manipulation of DECFLOAT data. These data types provide portability and compatibility to the DB2 family and platform.

2.13.1 BIGINT

The SQL BIGINT (8 byte) data type is introduced in DB2 V9 for compatibility with the Java BIGINT type. Mapping the Java application data types to DB2 column data types renders better application performance. The main reason is to provide efficient predicate processing. It also minimizes data type conversion cost.

BIGINT is an exact numeric data type that is capable of representing 63-bit integers. It extends a set of currently supported exact numeric data types (SMALLINT and INTEGER) and is compatible with all numeric data types.

The BIGINT scalar function returns a big integer representation of a number or string representation of a number. Example 2-16 shows the return values that are expected when using BIGINT functions.

Example 2-16  BIGINT examples

```
SELECT BIGINT(12345.6)
FROM SYSIBM.SYSDUMMY1;
---Returns 12345

SELECT BIGINT('00123456789012')
FROM SYSIBM.SYSDUMMY1;
---Returns 123456789012
```

The existing scalar functions CHAR, DIGITS, LENGTH, MOD, MULTIPLY_ALT, POWER®, and VARCHAR have been extended to support BIGINT data type.

Performance

Figure 2-9 shows the DB2 CPU and elapsed time comparison for inserting one million rows of various data types. BIGINT is compared with INTEGER and DECIMAL(19,0). In this case the row consists of only one column.
For INSERT, the elapsed and CPU times are very similar for all three data types.

For SELECT, BIGINT shows good performance, better than DECIMAL(19,0) by 5% for elapsed time, and very similar to INTEGER. For CPU time, BIGINT shows performance better than DECIMAL(19,0) by about 2% for elapsed time, and very similar to INTEGER.

Figure 2-10 shows the DB2 CPU and elapsed time comparison for inserting one million rows of various data types. BIGINT is compared with INTEGER and DECIMAL(19,0). In this case the row consists of 20 columns.
BIGINT performs better than DECIMAL(19,0) in terms of CPU time in both tests of SELECT and INSERT of 1 million rows. On a per column processing basis, BIGINT takes 1% less CPU time for SELECT and 3% less CPU time in the case of INSERT.

2.13.2 BINARY and VARBINARY

BINARY is a fixed-length binary string (1 to 255 bytes) and VARBINARY is a variable-length binary string (1 to 32704 bytes). BINARY and VARBINARY data types extend current support of binary strings (BLOB) and are compatible with BLOB data type.

They are not compatible with character string data types:
- There is an improvement for binary data over the character string FOR BIT DATA.
- The equivalent V8 data type for BINARY is CHAR FOR BIT DATA.
- The equivalent V8 data type for VARBINARY is VARCHAR FOR BIT DATA.

The BINARY scalar function returns a BINARY (fixed-length binary string) representation of a string. The following examples assume EBCDIC encoding of the input literal strings:
- Returns a fixed-length binary string with a length attribute of 1 and a value of BX’00’:
  ```sql
  SELECT BINARY(‘‘,1)
  FROM SYSIBM.SYSDUMMY1
  ```
- Returns a fixed-length binary string with a length attribute of 5 and a value of BX’D2C2C80000’:
  ```sql
  SELECT BINARY(‘KBH’,5)
  FROM SYSIBM.SYSDUMMY1
  ```

The VARBINARY scalar function returns a VARBINARY (varying-length binary string) representation of a string. The following examples assume EBCDIC encoding of the input literal strings:
- Returns a varying length binary string with a length attribute of 1, an actual length of 0, and a value of empty string:
  ```sql
  SELECT VARBINARY(‘‘)
  FROM SYSIBM.SYSDUMMY1
  ```
- Returns a varying length binary string with a length attribute of 5, an actual length of 3, and a value of BX’D2C2C8’: 
  ```sql
  SELECT VARBINARY(‘KBH’,5)
  FROM SYSIBM.SYSDUMMY1
  ```

The existing scalar functions INSERT, LEFT, LTRIM, POSSTR (POSITION does not support), REPEAT, REPLACE, RIGHT, RTRIM, STRIP, and SUBSTR have been extended to support BINARY and VARBINARY data types.

Figure 2-11 shows the CPU and elapsed time comparison for an INSERT of one million rows. This comparison is performed showing results for BINARY versus CHAR FOR BIT DATA with
a length of 100, and VARBINARY versus VARCHAR FOR BIT DATA with lengths of 10 and 1000.

Figure 2-11  BINARY and VARBINARY performance of INSERT of one million rows
Figure 2-12 shows the CPU and elapsed time comparison for a SELECT of one million rows. This comparison is performed showing results for BINARY versus CHAR FOR BIT DATA with a length of 100, and VARBINARY versus VARCHAR FOR BIT DATA with lengths of 10 and 1000.

![Figure 2-12: BINARY and VARBINARY performance of SELECT of one million rows](image)

INSERT and SELECT of one million rows of BINARY and VARBINARY columns are measured to be less than 3% of CPU difference when compared to that of CHAR FOR BIT DATA and VARCHAR FOR BIT DATA respectively.

### 2.13.3 DECFLOAT

A decimal floating-point constant is a DECFLOAT signed or unsigned number within the range of DECFLOAT. It has one of the following characteristics:

- A number that is specified as two numbers that are separated by an E with either of the following characteristics:
  - Excluding leading zeros, the number of digits in the first number exceeds 17 (precision).
  - The exponent is outside of the range of double floating-point numbers (smaller than -79 or larger than 75).

If a decimal floating-point constant contains an E, the first number can include a sign and a decimal point. The second number can include a sign but not a decimal point. The value of the constant is the product of the first number and the power of 10 specified by the second number. It must be within the range of a DECFLOAT(34). Excluding leading zeros, the number of digits in the first number must not exceed 34 and the number of digits in the second must not exceed 4. The number of characters in the constant must not exceed 42.
A number that does not contain an E, but has more than 31 digits

All numbers have a sign, a precision, and a scale. If a column value is zero, the sign is positive. Decimal floating point has distinct values for a number and the same number with various exponents, for example: 0.0, 0.00, 0.0E5, 1.0, 1.00, and 1.0000. The *precision* is the total number of binary or decimal digits excluding the sign. The *scale* is the total number of binary or decimal digits to the right of the decimal point. If there is no decimal point, the scale is zero.

DECFLOAT (or a distinct type based on DECFLOAT) cannot be used for primary key, unique key, a foreign key or parent key, an IDENTITY column, a column in the partitioning key (PARTITION BY RANGE), a column used for index on expression, and a column that has FIELDPROC.

The scalar functions COMPARE_DECFLOAT, DECFLOAT, DECFLOAT_SORTKEY, NORMALIZE_DECFLOAT, QUANTIZE, and TOTALORDER have been introduced.

DECFLOAT is currently supported in the Java, Assembler, and REXX languages.

With the introduction of these data types, the numeric data types are categorized as follows:

- **Exact numerics: binary integer and decimal**
  
  Binary integer includes small integer, large integer, and big integer. Binary numbers are exact representations of integers. Decimal numbers are exact representations of real numbers. Binary and decimal numbers are considered exact numeric types.

- **Decimal floating point**
  
  Decimal floating point numbers include DECFLOAT(16) and DECFLOAT(34), which are capable of representing either 16 or 34 significant digits.

- **Approximate numerics: floating point**
  
  Floating point includes single precision and double precision. Floating-point numbers are approximations of real numbers and are considered approximate numeric types.

Native support for decimal floating point (DECFLOAT) in DB2 is similar to both packed decimal and floating point, but can represent the number exactly versus approximation. DECFLOAT can represent much bigger and smaller numbers than DECIMAL.

The formats are a length of 17 for DECFLOAT(34) and a length of 9 for DECFLOAT(16). When describing a DECFLOAT host variable, make sure that the return length is 8 or 16 and not 9 or 17.

**Restrictions**

A DECFLOAT column cannot be a PARTITION KEY, UNIQUE KEY, PRIMARY KEY, FOREIGN KEY, or CHECK constraint, nor is it indexable.

If you try to define an index with a DECFLOAT column, you get an error as shown in Example 2-17.

*Example 2-17  Restriction on DECFLOAT key column*

```sql
CREATE UNIQUE INDEX XLITXT
ON LITXT(INT_COL, DECFLOAT_COL)
USING STOGROUP SYSDEFLT
    PRIQTY 14400 SECQTY 28800
CLUSTER
FREEPAGE 0
PCTFREE 0
BUFFERPOOL BP2
```
CLOSE NO;
SQLERROR ON CREATE COMMAND, EXECUTE FUNCTION
RESULT OF SQL STATEMENT:
  DSNT408I SQLCODE = -350, ERROR: DECFLOAT_COL WAS IMPLICITLY OR EXPLICITLY
  REFERENCED
  IN A CONTEXT IN WHICH IT CANNOT BE USED
SQLSTATE = 42962 SQLSTATE RETURN CODE
SQLERRP = DSIIKY SQL PROCEDURE DETECTING ERROR
SQLERRD = 66 0 0 -1 0 0 SQL DIAGNOSTIC INFORMATION
SQLERRD = X'00000042' X'00000000' X'00000000' X'FFFFFFFF' X'00000000' X'
  INFORMATION

SPUFI currently does not allow you to select data from DECFLOAT type columns. APAR
PK43861 adds that capability.

**Performance**

Assuming that you might want to convert from DECIMAL data type to DECFLOAT because
your application needs precision and large numbers, we compare the data types for INSERT
and SELECT on a System z9 processor with hardware support. You can see the benefits of
this support in 4.20.3, “Hardware support for the DECFLOAT data type” on page 136.

Example 2-18 shows the statements of a test that inserted one million rows into a table that
was created to test the DECFLOAT performance as compared to a DECIMAL data type. The
inserted rows contain either 15 DECFLOAT(16) or 15 DECIMAL(16, 0) columns with a
primary key defined on an INTEGER column.

**Example 2-18  Inserting into DECFLOAT data type**

```sql
INSERT INTO USRT001.TESTAB1
  ( ROWNUM,DEC01,DEC02,DEC03,DEC04...DEC15)
VALUES  ( :ROWNUM1, :DEC01, :DEC02, ...:DEC15);
```

Figure 2-13 shows the results. The insert performance of DECFLOAT(16) data type is much
higher than DECIMAL(16, 0) because of external to internal format conversion.

![Figure 2-13  Performance comparison INSERT DECFLOAT versus DECIMAL](image)
Example 2-19 shows the statements for a test that selects one million rows using an index and using System z9 hardware support for the arithmetic sum operation. This test requires the conversion of the DECIMAL or DECFLOAT into CHAR variable.

Example 2-19 Selecting with DECFLOAT conversion

```sql
SELECT HEX(DEC01 + DEC02) INTO :VAR01
FROM USRT001.TESTAB1 WHERE ROWNUM = :ROWNUM1;
```

Figure 2-14 shows the results. The DECFLOAT SELECT performance was 55.5% of the DECIMAL select measured in DB2 class 2 CPU time.

Overall some overhead is involved in using the new DECFLOAT data type, but there is an advantage in having the actual representation of the number versus an approximation.

Some restrictions regarding DECFLOAT can impact the performance of particular queries:
- No index is supported on a DECFLOAT column; therefore, a predicate on DECFLOAT is not indexable.
- A predicate associated with DECFLOAT is not treated as a stage-1.
- Predicate generation through transitive closure is not applied to a predicate that is associated with DECFLOAT data types.

**Note:** BIGINT, BINARY, VARBINARY, and DECFLOAT are available in new-function mode. DECFLOAT has additional requirements that are documented in *Program Directory for IBM DB2 9 for z/OS*, GI10-8737.
2.14 Autonomic DDL

DB2 9 NFM implicitly creates a database if the user does not specify an IN clause or database name on a CREATE TABLE statement. The names of implicit databases depend on the value of the sequence SYSIBM.DSNSEQ_IMPLICITDB:\[1\]

DSN00001, DSN00002, DSN00003, ..., DSN09999, and DSN10000

Also, if the containing table space is implicitly created, DB2 creates all the system-required objects for the user. Examples of implicit creation are: enforcing a primary key index, enforcing a unique key index, and enforcing a ROWID index on which the column is defined as ROWID GENERATED BY DEFAULT and LOB objects. The simple table space is deprecated: DB2 9 no longer implicitly creates, or allows you to create, simple table spaces. However, DB2 still supports simple table spaces that were created prior to DB2 9.

Note: PK60612 (UK35132) has modified to allow UNLOAD from ICOPY of a table space which was non-segmented even though now it is defined as segmented. This can help in case of accidental drop of a simple table space.

Autonomic DDL or implicit creation of objects simplifies the porting of other database management system (DBMS) databases and applications to the System z platform. It also reduces the number of tasks that a skilled DBA has to deal with by automatically creating system required objects and increases concurrency when creating objects in database.

2.14.1 Performance

Table 2-13 lists the measurements when defining and then dropping the objects in a traditional sequence of DDL.

<table>
<thead>
<tr>
<th>Explicit tables</th>
<th>Elapsed time (seconds)</th>
<th>CPU time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create 200 databases</td>
<td>~1</td>
<td>0.52</td>
</tr>
<tr>
<td>Create 200 tablespaces</td>
<td>24</td>
<td>0.32</td>
</tr>
<tr>
<td>Create 200 tables</td>
<td>5</td>
<td>0.41</td>
</tr>
<tr>
<td>Drop 200 tablespaces</td>
<td>8</td>
<td>0.56</td>
</tr>
<tr>
<td>Drop 200 databases</td>
<td>~0</td>
<td>0.27</td>
</tr>
</tbody>
</table>

Table 2-14 lists the measurements when defining and then dropping the objects in a sequence of DDL which implicitly defines databases and table spaces.

<table>
<thead>
<tr>
<th>Implicit tables</th>
<th>Elapsed time (seconds)</th>
<th>CPU time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create 200 tables</td>
<td>20</td>
<td>0.76</td>
</tr>
<tr>
<td>Drop 200 tables</td>
<td>45</td>
<td>21.5</td>
</tr>
</tbody>
</table>

\[1\] The default maximum number of databases that can be created implicitly has been lowered from 60000 to 10000 with APAR PK62178 (PTF UK44489). Once the limit is reached, object are assigned cyclically to the existing implicit databases, following from the start the sequence of the previously created implicit databases. The user is allowed to set the number of implicit databases they want to have up to the limit.
2.14.2 Conclusion

Autonomic DDL is a usability feature of DB2 V9. It allows the implicit creation of objects without involvement from customer administrative personnel and improves the ability to port DB2 objects from other platforms and DB2 family.

Notes:
- In DB2 9 conversion mode, DB2 implicitly creates a segmented table space with SEGSIZE 4, LOCKSIZE ROW.
- In DB2 9 new-function mode, new system parameters are introduced. The parameters determine how implicit objects are created. For an additional discussion about the new system parameters TBSPOOL, IDXBPPOOL, TBSBPLOB, TBSBPXML, IMPDSDEF, and IMPTSCMP, refer to DB2 Version 9.1 for z/OS Installation Guide, GC18-9846.

2.15 APPEND YES option in DDL

At INSERT or LOAD RESUME time, DB2 tries to make adequate reuse of the available free space and follow the sequence dictated by a clustering index or MEMBERCLUSTER option. This can slowdown the performance of INSERT while DB2 goes searches the appropriate place for the insert.

There are instances, however, when the application might not really care where the new row will be located. For instance, when a REORG is planned anyway for the table at the completion of a large batch insert process or when the data is always randomly accessed.

DB2 9 has added the new keyword APPEND YES/NO to the CREATE and ALTER TABLE DDL statements:

- YES
  Requests data rows to be placed into the table by disregarding the clustering during SQL INSERT and online LOAD operations. Rather than attempting to insert rows in cluster-preserving order, rows are appended at the end of the table or appropriate partition.

- NO
  Requests standard behavior of SQL INSERT and online LOAD operations, namely that they attempt to place data rows in a well clustered manner with respect to the value in the row's cluster key columns. NO is the default option.

A new column, APPEND, in SYSIBM.SYSTABLES records the chosen option.

Because you can ALTER this attribute on and off, you can switch it on (YES) for a massive insert batch job you run once a month and always follow with REORG/RUNSTATS anyway, then switch it back off (NO) for your normal online insert processing. REORG and LOAD
REPLACE are unaffected by the APPEND option, so you can re-establish the clustering sequence while allowing a faster insert.

The APPEND is valid for all tables except those created in XML and work files table spaces\(^2\).

### 2.16 Index on expression

Index on expression allows you to create an index on a general expression. You can enhance your query performance if the optimizer chooses the index that is created on the expression. Use index on expression when you want an efficient evaluation of queries that involve a column expression. In contrast to simple indexes where index keys are based only on table columns, indexes on expression is not exactly the same as the values in the table columns. The values have been transformed by the expressions that you specify, for instance arithmetic calculations, built-in functions, and so on.

Some of the keywords and semantics that are used for simple indexes do not apply to this type of index. If the index is defined as unique, the uniqueness is enforced against the values that are stored in the index, and not the original column values.

#### 2.16.1 Performance

In this type of index, the expressions are evaluated at DML (INSERT/DELETE/UPDATE) or utility index maintenance (create, rebuild index, reorg table space) time and kept in the index. At query time, the predicate is evaluated directly against the values as they are stored in the index if the optimizer chooses to use that index. There is no runtime performance overhead when using this type of index. If this kind of index is used in the query, the predicates are evaluated as matching or screening predicates. Otherwise the predicates are evaluated in stage 2. DB2 also uses expression matching logic as it does in materialized query tables (MQTs). For instance, an index defined on Salary + Bonus is used even though the predicate may ask for Bonus + Salary.

In the following examples, we illustrate the benefit of using index on expression compared to using a built-in function or arithmetic expression during SQL execution.

Indexes were created on the example table as follows:

- **On an integer column**
  - Index I1 as C_CUSTKEY
  - Index I2 as C_CUSTKEY*2

- **On a varchar column**
  - Index V1 as C_NAME
  - Index V2 as UPPER(C_NAME, 'locale')
  - Index V3 as SUBSTR(C_NAME,10,9)

- **Locale**: EN_US (EBCDIC) and UNI (UNICODE)

Four different cases were measured to show the performance improvements when index on expression is used with column functions and arithmetic expressions in the index definition. All measurements were made on V9, with or without the use of index on expression.

\(^2\) APAR PK65220 (PTF UK41212) has added this option to LOB auxiliary table spaces. The APPEND clause on the base table is extended to be specified on the LOB table via CREATE AUX TABLE DDL statement or ALTER TABLE DDL statement in DB2 9 NFM.
Case 1: The query executed for UPPER in SELECT and predicate

```sql
SELECT UPPER(C_NAME,'EN_US') FROM CUSTOMER
WHERE UPPER(C_NAME,'EN_US')
    = 'CUSTOMER#003999999';
```

Table 2-15 on page 53 contains the measurement results for CPU. The comparison is for results when index V1 is chosen versus index V2. The column function UPPER is used in both the SELECT and the predicate.

Prior to this enhancement with an index on C_NAME, the optimizer would have performed an index scan with 0 matching columns and would have searched the entire index tree. After the enhancement, with the index on UPPER(C_NAME), the optimizer is able to choose index only access with one matching column.

Case 2: The query executed for UPPER in predicate only

```sql
SELECT C_CUSTKEY FROM CUSTOMER
WHERE UPPER(C_NAME,'EN_US')
LIKE 'CUSTOMER#0039999%9'
```

Table 2-15 on page 53 contains the measurement results for CPU. The comparison is for results when index V1 is chosen versus index V2. The column function UPPER is used in the predicate only.

Prior to this enhancement, this query with an index on C_NAME would have chosen a table space scan on CUSTOMER. With the new functionality, this query now chooses an index scan with one matching column plus the selected data pages.

Case 3: The query executed for SUBSTR in SELECT and predicate

```sql
SELECT SUBSTR(C_NAME,10,9),COUNT(*)
FROM CUSTOMER
WHERE C_CUSTKEY > 149,990 AND
SUBSTR(C_NAME,10,9) LIKE '0039999%9'
GROUP BY SUBSTR(C_NAME,10,9);
```

Table 2-15 on page 53 contains the measurement results for CPU. The comparison is for results when index V1 is chosen versus index V3. The column function SUBSTR is used in both the SELECT and the predicate.

Prior to this enhancement, with an index on C_NAME, the query would have performed an index scan on the entire index tree and a sort for the GROUP BY. With the enhancement in place, the query now uses the index with one matching column. A sort is avoided.

Case 4: The query executed for an arithmetic expression

```sql
SELECT C_NAME FROM CUSTOMER WHERE C_CUSTKEY*2 < 2000007 AND C_CUSTKEY*2 > 2000001;
```

Table 2-15 on page 53 contains the measurement results for CPU. The comparison is for results when index I1 is chosen versus index I2 using an index that is created with an arithmetic expression C_CUSTKEY*2 and the arithmetic expression is in the predicate.

Prior to this enhancement, DB2 would have chosen a relational scan and sequential prefetch. With the index on expression of C_CUSTKEY*2, the access is an index scan with one matching column and then access to the data pages.
Table 2-15 shows the results of all four queries where index on expression was used with indexes that are defined with either a column function or an arithmetic expression.

**Table 2-15  Index on expression CPU comparison**

<table>
<thead>
<tr>
<th>Case</th>
<th>Query type</th>
<th>CPU without index on expression</th>
<th>CPU with index on expression</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UPPER on SELECT and predicate</td>
<td>12.89</td>
<td>0.01</td>
<td>1289 X</td>
</tr>
<tr>
<td>2</td>
<td>UPPER on predicate only</td>
<td>13.45</td>
<td>0.012</td>
<td>1120 X</td>
</tr>
<tr>
<td>3</td>
<td>SUBSTR on SELECT and predicate</td>
<td>8.47</td>
<td>0.015</td>
<td>564 X</td>
</tr>
<tr>
<td>4</td>
<td>Arithmetic expression</td>
<td>10.57</td>
<td>0.04</td>
<td>262 X</td>
</tr>
</tbody>
</table>

Table 2-16 shows the getpages and access paths that were chosen without the use of index on expression.

**Table 2-16  Without index on expression**

<table>
<thead>
<tr>
<th>Case</th>
<th>Query type</th>
<th>AT</th>
<th>MC</th>
<th>IX</th>
<th>BP0 Getpages</th>
<th>BP3 Getpages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UPPER on SELECT and predicate</td>
<td>I</td>
<td>0</td>
<td>Y</td>
<td>3</td>
<td>37,503</td>
</tr>
<tr>
<td>2</td>
<td>UPPER on predicate only</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>192,670</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>SUBSTR on SELECT and predicate</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>189,463</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Arithmetic expression</td>
<td>R</td>
<td>-</td>
<td>-</td>
<td>192,517</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2-17 shows the getpages and access paths that were chosen with the use of index on expression.

**Table 2-17  With index on expression**

<table>
<thead>
<tr>
<th>Case</th>
<th>Query type</th>
<th>AT</th>
<th>MC</th>
<th>IX</th>
<th>BP0 Getpages</th>
<th>BP3 Getpages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UPPER on SELECT and predicate</td>
<td>I</td>
<td>1</td>
<td>Y</td>
<td>61</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>UPPER on predicate only</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>69</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>SUBSTR on SELECT and predicate</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>21</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Arithmetic expression</td>
<td>I</td>
<td>1</td>
<td>N</td>
<td>15</td>
<td>3</td>
</tr>
</tbody>
</table>

In the previous two tables, the following nomenclature is used:

- **AT** = Access type, **MC** = Matching columns, and **IX** = Index only.
- **BP0** contains the table CUSTOMER and the DB2 catalog tables.
- **BP3** contains the index on CUSTOMER.

The percentage of improvement on queries varies depending on the size of the table. The examples shown here are queries against a 4.5 million row table with 60 partitions. Some CPU cost is for the evaluation of the expression.
Overall overhead for usage of index on expression is introduced into the evaluations done at DML (INSERT/DELETE/UPDATE) and index maintenance time. CPU overhead may be introduced to the following SQL and utilities for an index defined with expression:

- INSERT
- UPDATE ON KEY VALUE
- LOAD
- REBUILD INDEX
- REORG TABLESPACE
- CHECK INDEX

There is no impact on REORG INDEX.

Figure 2-15 shows the CPU overhead for INSERT when using index on expression of various types.

![Graph showing CPU overhead for INSERT](image)

**Figure 2-15** INSERT CPU overhead for index on expression

### 2.16.2 Conclusion

Index on expression shows significant improvements in query performance. Laboratory test results show dramatic improvement when index on expression is used. There is a significant reduction in overall CPU and DB2 getpages.

Although there is CPU cost on the evaluation of the expression, the CPU overhead is considered reasonable. These operations are LOAD, CREATE INDEX, REBUILD INDEX, REORG TABLESPACE, and CHECK INDEX.

**Note:** The index on expression functionality is available in new-function mode.
2.17 Histogram statistics over a range of column values

RUNSTATS can collect frequency statistics for a single-column index or multi-column indexes. Catalog space and bind time performance concerns make the collection of these types of statistics on every distinct value found in the target column or columns impractical. Such frequency statistics are commonly collected only on the most frequent or least frequent, and therefore most biased, values. These types of limited statistics often do not provide an accurate prediction of the value distribution because they require a rough interpolation across the entire range of values. In some cases, distribution statistics for any single value cannot help DB2 to estimate predicate selectivity, other than by uniform interpolation of filter factors over the uncollected part of the value range. The result of such interpolation might lead to inaccurate estimation and undesirable access path selection. You can improve access path selection by specifying the histogram statistics option HISTOGRAM in RUNSTATS.

Gaps occur in column values when logically there are no valid values for a range of predicate arguments that could be specified for a predicate value. An example of this is a date range for a column of type INTEGER that is used to contain a date field of the format yyyyymm. When the predicate specifies a value between 200512 and 200601, there are values that do not correspond to actual yyyyymm data, for example, 200513-200600. These values leave gaps in the distribution of the data that can lead to problems for optimization interpolation.

Figure 2-16 shows an example where gaps in ranges may cause poor performance in queries.

Example #1: When gaps exist in ranges
Application uses INTEGER (or worse, VARCHAR) to store YEAR-MONTH data.

There are 12 values in 200501~200512, but zero values in 200513~200600.

**Query:** SELECT * FROM T WHERE T.C1 between ‘a skipped range’
SELECT * FROM T WHERE T.C1 between ‘a non-skipped range’

Optimizer assumes
BETWEEN 200512 AND 200601

Returns more rows than
T.C1 between 200501 and 200512;

90 valid numerics, but only 2 valid dates

12 valid numerics, and 12 valid dates

*Figure 2-16   Gaps in ranges*

Sales data presents another example where distribution of sales information can be dense or sparse dependent on geographical or national holidays and events. For example, sales data in a table where rows are inserted by day or week, in the United States, would have a significantly greater density of information from the day after Thanksgiving holiday until Christmas Day. Then sales data density would decline after Christmas and experience a much lower density (or sparseness). This data would typically be dense prior to a significant event or holiday and then become sparse afterwards.
Figure 2-17 shows an example where sparse or dense and nonexistent values can cause poor performance problems.

<table>
<thead>
<tr>
<th>Example #2: Sparse and dense ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SALES_DATE BETWEEN '2005-12-11' AND '2005-12-24'</strong> returns significantly more rows than a two-week range in a single month</td>
</tr>
</tbody>
</table>

**Query:**
- SELECT * FROM T WHERE T.C1 between 'a sparse range'
- SELECT * FROM T WHERE T.C1 between 'a dense range'

<table>
<thead>
<tr>
<th>Example #3: Non-existent values out of [lowkey, highkey] range</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 only records the second highest or lowest value for a column. Hard to detect any out-of-range value.</td>
</tr>
</tbody>
</table>

**Query:**
- SELECT * FROM T WHERE T.C1 = 'non-existent value'

The V9 RUNSTATS utility produces an equal-depth histogram. Each interval (range) has about the same number of rows but not the same number of values. The maximum number of intervals is 100. The same value stays in the same interval. NULL values have their own interval. There are possible skipped gaps between intervals, and there is a possibility of an interval containing a single value only.

Histogram statistics are also used to more accurately determine equality of degree for parallel child tasks.

### 2.17.1 Performance

Measurements in the lab resulted in up to two times the elapsed time or CPU time improvement for several queries and better join sequences that reduce data, index, and workfile getpages. If the query is I/O bound, either CPU or I/O may improve but generally not both.

For information about RUNSTATS HISTOGRAM utility performance, refer to 6.3, “RUNSTATS enhancements” on page 214.

### 2.17.2 Recommendation

Use histogram statistics to evaluate predicate selectivity in RANGE/LIKE/BETWEEN predicates for all fully qualified intervals, plus interpolation of partially qualified intervals. Histogram statistics also help in the following situations:

- **EQ, ISNULL, INLIST**
  - A single-value interval matched the searching literal or interpolation within the covering interval.

- **COL op COL**
  - Pair up two histogram intervals that satisfy operator OP. You must gather histogram statistics for intervals on both sides of the operator.
Histogram statistics improve filter factor estimation. Histogram statistics, like frequency statistics, require knowledge of the literal values for accurate filter factor estimation. The exception is exclusion of NULLs in filter factor estimation.

Note: The histogram statistics over a range of column values are available in conversion mode with a REBIND required.

2.18 LOB performance

There are two enhancements in LOB performance that can have a significant impact on overall performance of queries that process LOBs:

- **LOB file reference**
  Applications can efficiently insert a big LOB into a DB2 table and write a big LOB into a file without having to acquire any application storage. Undoubtedly, with file reference enhancement, DB2 for z/OS not only maintains its family compatibility, but also improves its LOB performance.

- **FETCH CONTINUE**
  FETCH CONTINUE allows an application to do a FETCH against a table that contains LOB or XML columns, using a buffer that might not be large enough to hold the entire LOB or XML value. If any of the fetched LOB or XML columns do not fit, DB2 returns information about which column or columns were truncated and the actual length. To enable this behavior on the FETCH, the application must add the WITH CONTINUE clause. The application is then able to use that actual length information to allocate a larger target buffer, to execute a FETCH statement with the CONTINUE clause, and finally to retrieve the remaining data for those columns.

For detailed information about the implementation of LOBS in DB2, refer to *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270.

2.18.1 LOB file reference variable

The purpose of file reference variables is to import or export data between an LOB column and an external file outside of the DB2 system. In the past, if you used a host variable to materialize the entire LOB in the application, your application would need adequate storage. It would incur poor performance, because the file I/O time would not be overlapped with any DB2 processing or network transfer time.

Locator variables used in conjunction with the SUBSTR function can be used to overlap the file I/O time with DBM1 processing time or network transfer time and to avoid materializing the whole LOB in the application. However, there is still some CPU overhead to transfer pieces of the LOB between DBM1 and the application.

LOB file reference variables accomplish the same function using less CPU time and avoiding the use of any application storage for the LOBs. LOB file references are also easier to use than locator variables. LOB file reference variables are supported within a DB2 for z/OS system or in a distributed configuration between DB2 for z/OS subsystems.
With this new functionality, all the SQL statements using LOB file reference experience a performance gain. In particular, the following statements are where LOB file reference is expected to play a big role:

- `INSERT INTO LOBT1(CLOB1) VALUES(:FRHostVar)`
- `UPDATE LOBT1 SET CLOB1 = :FRHostVar`
- `SELECT CLOB1 INTO :FRHostVar FROM LOBT1`
- `FETCH cursor TO :FRHostVar`

**Performance**

File reference variables perform at device speed, and all input/output is overlapped. Unlike the concatenation operator with locator variables, file reference variables scale well as the LOB size increases.

Table 2-18 lists the measurements when using file reference variables during INSERT.

<table>
<thead>
<tr>
<th>Insert a 1 GB LOB</th>
<th>CPU (seconds)</th>
<th>Elapsed time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>File reference variable</td>
<td>2.7</td>
<td>26</td>
</tr>
<tr>
<td>Concatenation of 80 locator variables and using one locator variable</td>
<td>39</td>
<td>39</td>
</tr>
<tr>
<td>Concatenation of 80 locator variables and using two locator variables</td>
<td>150</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 2-19 lists the measurements when using file reference variables during SELECT.

<table>
<thead>
<tr>
<th>Select 1 GB LOB</th>
<th>CPU (seconds)</th>
<th>Elapsed time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>File reference variable</td>
<td>2.7</td>
<td>12</td>
</tr>
<tr>
<td>Using SUBSTR</td>
<td>1.6</td>
<td>12</td>
</tr>
</tbody>
</table>

**Recommendation**

Use file reference variables where applicable to achieve better performance and greater I/O throughput.

**Note:** The LOB file reference variable functionality is available in new-function mode.

### 2.18.2 FETCH CONTINUE

The FETCH CONTINUE enhancement introduces extensions to the FETCH SQL statement. These extensions to the FETCH statement are the WITH CONTINUE clause and the CONTINUE clause. They provide a convenient method for applications to read from tables that contain LOB or XML columns, when the actual length of the LOB or XML value is not known or is so large that the application cannot materialize the entire LOB in memory. The declared maximum length for an LOB column is frequently much larger than the typical LOBs that are inserted into those columns. Prior to this enhancement, applications that used embedded SQL to read from tables that contain LOB columns typically had to declare or allocate storage that was equal in size to the maximum defined storage size for the LOB column. This could cause a shortage of virtual memory in certain configurations.
LOB locators are one way to avoid having to preallocate space for the entire LOB. However, they have some problems as well, including slower performance in some cases, excessive resource consumption at the server, and more complex coding.

This enhancement allows an application to do a FETCH against a table that contains LOB or XML columns, using a buffer that might not be large enough to hold the entire LOB or XML value.

**Recommendations**
There are two expected common uses of FETCH CONTINUE:

- **FETCH into a moderate sized buffer, into which most values are expected to fit**
  For any values that do not fit, allocate a larger buffer using the actual length, as returned by DB2. Use FETCH CURRENT CONTINUE to retrieve the rest of the data and assemble the value. For this method, the programming language that is used must allow for dynamic storage allocation. The application must also build and manage its own SQLDA and use INTO DESCRIPTOR SQLDA with FETCH and FETCH CURRENT CONTINUE.

- **Streaming the data through a single fixed-size buffer**
  The application fetches the LOB/XML object in pieces using a “small” temporary buffer, for example 32 KB in as many FETCH and FETCH CURRENT CONTINUE operations as necessary, using the same buffer area. As it performs each operation, the application moves the data to another area for assembly, or pipes it to a file, tool, or other application.

For applications that perform “random access” to parts of an LOB, when using such functions as LENGTH, SUBSTR, and POSSTR, or when LOB materialization is to be avoided, we still recommend that you use LOB locators.

**Note:** The FETCH CONTINUE functionality is available in new-function mode.
XML

DB2 9 for z/OS presents a plethora of innovative functions and features. At the top of the list is the newly integrated DB2 XML storage engine that supports pureXML. It gives you the ability to store and query your XML data in its inherent hierarchical format.

Additionally, this release is equipped with hybrid data server support for both relational and pureXML storage, to unite the management of both structured and unstructured data. This support encompasses the seamless integration of XML with existing relational data, the exploitation of both tabular and hierarchical data models, and the flexibility of using SQL and XPath (subset of XQuery) in your applications for e-business.

In this chapter, we discuss the details of XML as well as performance particulars of various XML functions:

- Overview of XML
- pureXML support with DB2 for z/OS
- XML performance
3.1 Overview of XML

In the information management marketplace, XML continues to advance in the exchange of digital information in a multitude of forms across heterogeneous systems. Originally derived from the Standard Generalized Markup Language (SGML), XML was designed to better support the World Wide Web (WWW) and other networked applications across the Internet. Unlike the Hypertext Markup Language (HTML), XML is an open-ended markup language that can easily accommodate future expansions and additions. Hence, XML was created by the World Wide Web Consortium (W3C) to allow the circulation of complex, structured data and documents across various devices and platforms over the Internet.

XML data and documents are becoming an important business asset that contains valuable information. When XML data is exploited by unlocking the value of the information that it contains, it can translate into various opportunities for organizations. Hence, XML industry standards are becoming more and more widespread since there is a drive toward revolutionary service-oriented architecture (SOA) environments. To counterbalance the opportunities that XML presents, it presents challenges to its security, maintenance, and manipulation as XML data becomes more critical to the operations of an enterprise. In addition, XML data might have to be updated, audited, and integrated with traditional data. All of these tasks must be done with the reliability, availability, and scalability afforded to traditional data assets.

In order to unleash the potential of XML data, it requires storage and management services similar to what enterprise-class relational database management systems (DBMS), such as DB2, have been providing for relational data.

Therefore, the technology in DB2 V9 fundamentally transforms the way XML information is managed for maximum return while seamlessly integrating XML with relational data. Additionally, the DB2 V9 new pureXML feature has revolutionized support for XML data by combining the management of structured and unstructured data to allow you to store XML data in its native format.

3.1.1 XML and SOA

The SOA, considered a best practice for over two decades, is finally being embraced by many enterprises that are seeking to increase business agility and decrease the time and cost of implementing new business solutions. In an SOA, discrete business functions or processes are created as independent, loosely coupled services with standard interfaces that can be accessed by other applications, services, or business processes regardless of the platform or programming language. These services can be flexibly combined to support different or changing business processes and functions. SOA supports the creation of composite applications, which can be quickly assembled from existing and new Web services.

Web services

Web services are self-describing, modular applications that expose business logic as services that can be published, discovered, and invoked over the Internet. It is a technology that is well-suited to implement an SOA.

Web services and SOAs are dedicated to reducing or eliminating impediments to interoperable integration of applications, regardless of their operating system platform or language of implementation. The following sections summarize and highlight the most compelling characteristics of Web services and SOA.
**Componentization**

SOA encourages an approach to systems development in which software is encapsulated into components called **services**. Services interact through the exchange of messages that conform to published interfaces. The interface that is supported by a service is all that concerns any prospective consumers; implementation details of the service itself are hidden from all consumers of the service.

**Platform independence**

In an SOA, the implementation details are hidden. Therefore, services can be combined and orchestrated regardless of programming language, platform, and other implementation details. Web services provide access to software components through a wide variety of transport protocols, increasing the number of channels through which software components can be accessed.

**Investment preservation**

As a benefit of componentization and encapsulation, existing software assets can be exposed as services within an SOA using Web services technologies. When existing software assets are exposed in this way, they can be extended, refactored, and adapted into appropriate services to participate within an SOA. This reuse reduces costs and preserves the investment. The evolutionary approach enabled by Web services eliminates the necessity to rip and replace existing solutions.

**Loose coupling**

As another benefit of componentization, the SOA approach encourages **loose coupling** between services, which is a reduction of the assumptions and requirements shared between services. The implementation of individual services can be replaced and evolved over time without disrupting the normal activities of the SOA system as a whole. Therefore, loosely coupled systems tend to reduce overall development and maintenance costs by isolating the impact of changes to the implementation of components and by encouraging reuse of components.

**Distributed computing standardization**

Web services are the focal point of many, if not most, of the current standardization initiatives that are related to the advancement of distributed computing technology. Additionally, much of the computer industry’s research and development effort related to distributed computing is centered on Web services.

**Broad industry support**

Core Web services standards, including SOAP, Web Services Description Language (WSDL), XML, and XML schema, are universally supported by all major software vendors. This universal support provides a broad choice of middleware and tooling products with which to build service-oriented applications.

**Composability**

Web services technologies are planned to enable designers to mix and match different capabilities through composition. For example, systems that require message-level security can leverage the Web services Security standard. Any system that does not require message-level security is not forced to deal with the complexity and overhead of signing and encrypting its messages. This approach to composability applies to all of the various qualities of service, such as reliable delivery of messages and transactions. Composability enables Web services technologies to be applied consistently in a broad range of usage scenarios, such that only the required functionality has to be implemented.
Summary
The technology of Web services is the most likely connection technology of SOA. Web services essentially use XML to create robust relationships. Based on XML standards, Web services can be developed as loosely-coupled application components using any programming language, any protocol, or any platform. This mode of development facilitates the delivery of business applications as a service that is accessible to anyone, anytime, at any location, and using any platform. Hence, through its revolutionary pureXML support, DB2 V9 embodies the immense flexibility that XML provides to assist your business in getting your SOA initiative off the ground.

For more details about data servers and SOA, see Powering SOA with IBM Data Servers, SG24-7259.

3.1.2 XML data access

In 2003, the first edition of the SQL/XML standard was published by the International Organization for Standardization (ISO). The second edition of SQL/XML was published in 2006. W3C currently makes the SQL/XML standard, and the technology that it includes, more highly visible to interested parties through their Web site. The Web site makes it possible for the SQL and XML communities to follow the development of SQL/XML as it occurs and to readily ascertain the current status of the standard's progression (see Figure 3-1). The Web site is located at the following address:

http://www.w3.org/XML/

The DB2 9 for z/OS engine processes SQL, SQL/XML, and XPath in an integrated manner since DB2 treats both SQL and a subset of XQuery as independent primary query languages. Applications can continue to use SQL and additionally SQL/XML extensions that allow publishing of relational data in XML format. XPath is a subset of XQuery and is typically used to access the native XML store. Optionally XPath may use SQL statements to combine XML...
data with SQL data. The functionality of XQuery is currently available only on DB2 for Linux, UNIX, and Windows, but you can expect to see it in the future for z/OS as well.

### 3.2 pureXML support with DB2 for z/OS

Prior to DB2 V9, decomposition methods were used to store XML data in order to comply with the relational model. Here are two options that were available to store XML data:

- **CLOB/VARCHAR**: Stored as linear text in a column of a table with the CLOB or VARCHAR data type
- **Shredding**: Decomposed into rows and columns to fit the relational model

Performance can become an issue when the application continues to use XML as an interface to the data after the XML data has been decomposed to relational. Otherwise there should not be any performance issue. However, storing the XML document as a CLOB or a VARCHAR in an XML column prevents XML parsing during insertion. Furthermore, reconstruction of the document from decomposed data is a rather complex and costly process. Hence, it does not make sense to decompose the document since it would require reconstruction in order for it to be used. Similarly, XML extenders work well but have been proven to experience performance setbacks in robust applications.

DB2 V9 leverages the XML phenomenon in applications for e-business through its pureXML support. The XML and relational services in DB2 V9 are tightly integrated through the exploitation of both tabular and hierarchical data models, thereby offering the industry’s first pureXML and relational hybrid data server.

The XML data is parsed by DB2 and stored as interconnected nodes on fixed size database pages. Non-validating XML parsing occurs within a newly integrated element of z/OS 1.8 called **z/OS XML System Services** (z/OS XML). z/OS XML is a system-level XML parser that is integrated with the base z/OS operating system and is designed to deliver an optimized set of services for parsing XML documents. z/OS XML has also been made available on z/OS V1.7. In addition, IBM United States Announcement 107-190, dated 18 April 2007, indicates the enabling of the z/OS XML component to take advantage of System z Application Assist Processors (zAAPs) and System z9 Integrated Information Processors (zIIPs).

The ability to support native XML in DB2 is accomplished through new storage management, indexing, and optimization techniques. Although XML and relational data are stored separately, both are under the control of the same DB2 engine. Figure 3-2 illustrates this new implementation.

![Figure 3-2  DB2 V9 new XML configuration](image-url)
Therefore, DB2 V9 is equipped with pureXML technology and includes the following capabilities:

- pureXML data type and storage techniques for efficient management of hierarchical structures that are common in XML documents.
- pureXML indexing technology to speed searches of subsets of XML documents
- New query language support (for XPath and SQL/XML) based on industry standards and new query optimization techniques
- Industry-leading support for managing, validating, and evolving XML schemas
- Comprehensive administrative capabilities, including extensions to popular database utilities
- XMLPARSE and XMLSERIALIZE functions to convert an XML value from its internal tree format to the corresponding textual XML (and vice versa)
- Integration with popular application programming interfaces (APIs) and development environments
- Enterprise proven reliability, availability, scalability, performance, security, and maturity that you have come to expect from DB2
- Leading-edge standards-based capabilities to enable SOA requirements

For more details about the new pureXML feature in DB2 V9, see *DB2 9 for z/OS Technical Overview*, SG24-7330, and *DB2 Version 9.1 for z/OS XML Guide*, SC18-9858.

### 3.2.1 XML structure

Each XML document has both a logical and a physical structure. As a logical structure, an XML document is made up of the markup. The markup consists of declarations, elements, attributes, comments, character references, and processing instructions. The physical structure of XML is made up by entities or files.

As a logical model, the XML document is best represented as a tree of nodes. The root node, also referred to as the document node, is the root of the tree, and it does not occur anywhere else in the tree. There are six nodes types in XML (see Figure 3-3):

- Document nodes
- Element nodes
- Text nodes
- Attribute nodes
- Processing instruction nodes
- Comment nodes
The DB2 V9 pureXML feature introduces the new XML data type in the CREATE TABLE statement to define a column as type XML.

Note: DB2 allows you to store only well-formed XML documents in columns with data type XML according to the XML 1.0 specification. See DB2 9 for z/OS Technical Overview, SG24-7330, for more information.

Each column of type XML can hold one XML document for every row of the table. Even though the XML documents are logically associated with a row, XML and relational columns are stored differently.

For example, a CREATE TABLE statement could look like this:

```sql
CREATE TABLE PEPPXML1 (C1 CHAR(10), XMLCOL XML)
```
The result of this statement is a table that consists of two columns: character column C1 and column XMLCOL with the data type XML. Also, several other objects have also been created implicitly by DB2 to support the XML column. Figure 3-4 illustrates all of these objects that are created in this definition.

![Diagram of objects created for an XML column definition]

The objects illustrated in Figure 3-4 include:

- A column called DB2GENERATEDDOC_ID_FOR_XML, which we refer to as the DocID column hereafter.
  DocID uniquely represents each row. (This column is hidden.) No matter how many XML columns are defined for one table, DB2 only needs one DocID column. The DocID is defined as "generated always", which means that you cannot update the DocID column.

- A unique index on the DocID column that is defined as NOT NULL.
  This index is known as a document ID index.

- An XML table space.
  The implicitly created table space has the following characteristics:
  - Partitioned by growth, if the base table space is not partitioned.
  - Partitioned by range, if the base table space is partitioned.
  Even though it is partitioned in this case, the XML table space does not have limit keys. The XML data resides in the partition number that corresponds to the partition number of the base row.
3.3 XML performance

As XML technology is increasingly embraced by enterprises for their on demand applications for e-business, management of XML data becomes a necessity. Hence, the integration of pureXML support in DB2 9 for z/OS delivers a vast amount of opportunities when managing your XML data.

XML is new technology, several performance and functional enhancements are provided via the maintenance stream. Some APARs are listed in Appendix A, “Summary of relevant maintenance” on page 291. A good starting point for current with maintenance is the information APAR II14426.

Performance analysis has been conducted to assess DB2 9 ability to store and manage XML data. Since XML is a totally new component in DB2 9, we are at the preliminary stages of its usage, and there is a lot of room for performance analysis in more realistic environments.

A recent whitepaper on DB2 and z/OS XML System Services is available at:

In the following sections, we first look at inserting, updating, and fetching XML documents. Then we look at index exploitation and compression.

3.3.1 INSERT performance

To insert data into an XML column, use the SQL INSERT statement. The data that you insert into the XML column must be a well-formed XML document, as defined in the XML 1.0 specification. The application data type can be XML (XML AS BLOB, XML AS CLOB, or XML AS DBCLOB), character, or binary.

If you insert an XML document into a DB2 table with validation, you must register an XML schema document. Similarly, a decomposition stored procedure is provided so that you can extract data items from an XML document and store those data items in columns of DB2 tables, using an XML schema. For information about validation, decomposition, and adding an XML schema to the DB2 XML schema repository (XSR), see DB2 Version 9.1 for z/OS XML Guide, SC18-9858.

In order to evaluate the performance of INSERT, four separate test cases were conducted to insert an XML document into a DB2 table. Table 3-1 lists the details of the environments that were used for the four test cases.
In the following sections, we describe the details and analysis of the test cases.

**Test case 1: Batch INSERT performance**

The new data type XML presented in DB2 V9 offers you the ability to insert (parse) an XML document in its native format. To assess the ability to perform a batch INSERT of XML documents into a DB2 table, the following statement was executed:

```sql
EXEC SQL INSERT INTO USRT001.X01KTB
VALUES (:hv1, 'ABCDEFGHIJ', 'abcdefghijklm', XMLPARSE(DOCUMENT :buffer));
```

See [DB2 Version 9.1 for z/OS XML Guide, SC18-9858](#), for details about using the XMLPARSE function. Figure 3-5 illustrates the results of this test case.

![Figure 3-5 Batch INSERT performance](image)

We observe a pleasing trend when parsing an XML document into a DB2 table. When parsing one million 1 KB XML documents, the CPU time is 168 seconds, and the elapsed time is 262 seconds. By contrast, parsing sets of 10 KB, 100 KB, 1 MB, or 10 MB (each case consuming 1 GB of storage), XML documents yield an average of 62 seconds CPU time and 75 seconds elapsed time.

**Conclusion**

The ability to parse an XML document into a DB2 table yields good performance in DB2 V9. Similar to retrieval performance, inserting XML documents into DB2 tables is scalable, and CPU time is dependent on the XML document size. As a result, the more nodes you have in your XML document, the more expensive the parsing is.

**Test case 2: INSERT performance with XML indexes defined**

To assess the ability to perform a batch INSERT of XML documents into a DB2 table with one, two, or no XML indexes defined, the following statement was run (same as test case 1):

```sql
EXEC SQL INSERT INTO USRT001.X01KTB
VALUES (:hv1, 'ABCDEFGHIJ', 'abcdefghijklm', XMLPARSE(DOCUMENT :buffer));
```

See the [DB2 Version 9.1 for z/OS SQL Reference, SC18-9854](#), for details about using the XMLPARSE function.
CPU and elapsed times tend to get larger as more indexes are defined on the table. The average CPU and elapsed times were calculated, and the subsequent ratios were computed to determine the cost of the XML indexes as shown in Figure 3-6.

![Insert (with Indexes) Performance](image)

*Figure 3-6  Results of the cost of indexing when parsing*

By looking at Figure 3-6, it is evident that one XML user index adds approximately 15-20% of CPU time. Similarly, an XML user index can add 10-15% in elapsed time.

**Conclusion**

In general, as mentioned in the previous test case, inserting XML documents into DB2 tables is scalable. As a result, the CPU time is dependent on the XML document size and on the number of XML indexes that are defined on the table. Hence, the more nodes you have in your XML document, the more expensive the parsing is.

Although there is an expected cost when parsing, you can create an index on an XML column for efficient evaluation of XPath expressions to improve performance during queries on XML documents.

**Test case 3: INSERT with validation performance**

You can validate an XML document against the schema that is registered in the XML schema repository (XSR) tables. New stored procedures (SYSPROC.XSR_REGISTER, SYSPROC.XSR_ADDSCHEMADOC, and SYSPROC.XSR_COMPETE) are provided to register the schema to XSR tables. After the schema is registered, you can invoke the user-defined function (UDF) SYSFUN.DSN_XMLVALIDATE in an XMLPARSE function. Input of DSN_XMLVALIDATE is a BLOB, a CLOB, or a VARCHAR data type.

To test the performance of inserting an XML document into a DB2 table with schema validation, three Universal Financial Industry (UNIFI) payment messages were used, as listed in Table 3-2.

<table>
<thead>
<tr>
<th>Document</th>
<th>Message name</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-bank direct debit collection</td>
<td>FIToFICustomerDirectDebitV01</td>
<td>pacs.003.001.01</td>
</tr>
<tr>
<td>Inter-bank payment return</td>
<td>PaymentReturnV01</td>
<td>pacs.004.001.01</td>
</tr>
<tr>
<td>Payment status report</td>
<td>PaymentStatusReportV02</td>
<td>pain.002.001.02</td>
</tr>
</tbody>
</table>
You can view the three respective XML schemas used for the messages in Table 3-2 on page 71 on the ISO Web site at the following address:

http://www.iso20022.org/index.cfm?item_id=59950

The following statement was executed to test the performance of INSERT without validation:

```sql
EXEC SQL
INSERT INTO TBTA0101 (XML1)
VALUES
(XMLPARSE(DOCUMENT :clobloc1));
```

The following statements were executed to test the performance of INSERT with validation:

```sql
EXEC SQL
INSERT INTO TBTA0101 (XML1)
VALUES
(XMLPARSE(DOCUMENT SYSFUN.DSN_XMLVALIDATE(:clobloc1,
'SYSXSR.pacs311')));
EXEC SQL
INSERT INTO TBTA0101 (XML1)
VALUES
(XMLPARSE(DOCUMENT SYSFUN.DSN_XMLVALIDATE(:clobloc1,
'SYSXSR.pacs411')));
EXEC SQL
INSERT INTO TBTA0101 (XML1)
VALUES
(XMLPARSE(DOCUMENT SYSFUN.DSN_XMLVALIDATE(:clobloc1,
'SYSXSR.pacs212')));
```

See *DB2 Version 9.1 for z/OS XML Guide*, SC18-9858, for details about using the `XMLPARSE` and `DSN_XMLVALIDATE` functions. The two previous statements were executed 50000 times in a single thread for the three message types. Table 3-3 shows the results of a performance comparison.

**Table 3-3  Insert without and with validation**

<table>
<thead>
<tr>
<th>XML schema</th>
<th>Time</th>
<th>INSERT without VALIDATION</th>
<th>INSERT with VALIDATION</th>
<th>Percentage difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-bank direct debit collection</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class 2 CPU time (seconds)</td>
<td>10.69</td>
<td>18.17</td>
<td>+69.97%</td>
<td></td>
</tr>
<tr>
<td>Class 2 elapsed time (seconds)</td>
<td>22.83</td>
<td>30.48</td>
<td>+33.51%</td>
<td></td>
</tr>
<tr>
<td>Inter-bank payment return</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class 2 CPU time (seconds)</td>
<td>18.07</td>
<td>34.85</td>
<td>+92.86%</td>
<td></td>
</tr>
<tr>
<td>Class 2 elapsed time (seconds)</td>
<td>37.52</td>
<td>56.52</td>
<td>+50.64%</td>
<td></td>
</tr>
<tr>
<td>Payment status report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class 2 CPU time (seconds)</td>
<td>15.10</td>
<td>33.88</td>
<td>+124.37%</td>
<td></td>
</tr>
<tr>
<td>Class 2 elapsed time (seconds)</td>
<td>32.46</td>
<td>54.99</td>
<td>+69.41%</td>
<td></td>
</tr>
</tbody>
</table>
It is evident that the CPU cost approximately doubles when inserting XML documents with schema validation in comparison to inserting such documents without validation. Similarly, an average of 50% cost in elapsed time is also experienced when using the validation feature.

**Conclusion**

It is evident that inserting XML documents into a DB2 table is fast and efficiently stores large XML documents. However, although there are benefits with the validation feature in DB2 9, there is an associated cost in both CPU and elapsed times.

**Test case 4: INSERT using decomposition performance**

 Decomposition, which is sometimes referred to as shredding, is the process of storing content from an XML document into columns of relational tables. After it is decomposed, the data has the SQL type of the column into which it is inserted. DB2 9 always validates the data from XML documents during decomposition. If information in an XML document does not comply with its specification in an XML schema document, DB2 does not insert the information into the table.

To test the performance of decomposing an XML document into a DB2 table, the following statement was executed:

EXEC SQL CALL SYSPROC.XDBDECOMPXML(:xmlschema:p1ind, :xmlname:p2ind, :clobloc1:p3ind, :docid:p4ind);

See DB2 Version 9.1 for z/OS XML Guide, SC18-9858, for details about invoking the stored procedure SXDBDECOMPXML. The previous statement produced the results shown in Figure 3-7. Refer to E.1, “XML document decomposition” on page 352, to see the document, schema, and Data Definition Language (DDL) of the table that is used in the schema.

![Figure 3-7  Decomposition results](image)

To decompose this XML document, the results indicate an average CPU time of 5.60 seconds and an average elapsed time of 10.33 seconds. The insert and insert with validation are shown for reference.

**Conclusion**

The performance of XML document decomposition is as expected since schema validation must be included in the actual decomposition.
General recommendations for INSERT

We recommend that you insert XML data from host variables, rather than from literals, so that the DB2 database server can use the host variable data type to determine some of the encoding information. In addition, to adjust the response time when running INSERT, it may be necessary to perform LOG I/O tuning.

However, in general, if you have a choice, use the LOAD utility instead of the SQL INSERT statement. When using the SQL INSERT statement, our measurements show an increase of about 30% in elapsed time and an increase of roughly 40% in CPU time.

For XML documents that are greater than 32 KB, an XML file reference variable can be used in a LOAD utility.

The LOAD and INSERT comparison is the same regardless of using the file reference variable.

Note: APAR PK47594 for XML LOAD improves performance for large documents.

3.3.2 UPDATE performance

To update data in an XML column, use the SQL UPDATE statement. The entire column value is replaced when executing this statement. The input to the XML column must be a well-formed XML document, as defined in the XML 1.0 specification. The application data type can be XML (XML AS BLOB, XML AS CLOB, or XML AS DBCLOB), character, or binary.

Performance measurement

In order to verify the performance of updating an XML document, a test case was constructed. The test case was executed in the following environment:

▶ System z9 processor
▶ z/OS 1.8
▶ ESS M800

Test case

In order to update the XML document, the following SQL UPDATE statement was executed:

EXEC SQL
UPDATE USRT001.X10KTB SET HIST_XML= XMLPARSE(DOCUMENT :buffer)
WHERE HIST_INT1 = :hv1;

When updating an XML document, an SQL UPDATE statement is equivalent to the combination of performing an SQL DELETE statement with an SQL INSERT statement since the whole document is replaced. As a result, here are the statements that were executed:

EXEC SQL
DELETE FROM USRT001.X10KTB
WHERE HIST_INT1 = :hv1;

EXEC SQL INSERT INTO USRT001.X01KTB
VALUES (:hv1, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz', XMLPARSE(DOCUMENT :buffer));
The statements produced the results that are shown in Figure 3-8.

We observe that the SQL UPDATE requires 144 seconds of CPU time and 83 seconds of elapsed time. An SQL DELETE followed by an SQL INSERT requires 141 seconds of CPU time and 121 seconds of elapsed time.

**Conclusion**

Since there is no subdocument level of updating that takes place in V9, the entire XML document is replaced. As a result, the performance of SQL UPDATE is nearly equivalent to the performance of SQL DELETE plus SQL INSERT.

**Recommendations**

If there is a frequent need to update certain XML documents, we recommend that you consider storing sets of subdocuments rather than storing one large document.

### 3.3.3 XML retrieval and XML serialization

*Serialization* is the process of converting an object into a form that can be readily transported. With DB2, you use XMLSERIALIZE to retrieve an XML document in its tree format into an LOB. That is, you can use XML serialization to convert internal XML data to the serialized string format that the application can process. The XMLSERIALIZE function can convert the XML data type to CLOB, BLOB, or DBCLOB.

The FETCH statement positions a cursor on a row of its result table. It can return zero, one, or multiple rows and assigns the values of the rows to host variables if there is a target specification. Hence, when you fetch an entire XML document, you retrieve the document into an application variable. At this point, the XML document is said to be in its serialized format.

You can retrieve XML data that is stored in an XML column by using an SQL SELECT statement or by using an XMLQUERY function.
Performance measurement

In order to test the performance of the FETCH statement, a query was run to repeatedly fetch different sizes of lab developed XML documents. All measurements were performed in the following environment:

- System z9 processor
- z/OS 1.7
- IBM System Storage DS8300

Example 3-1 shows the statement that was performed to assess the FETCH performance.

Example 3-1   Fetching XML data

EXEC SQL DECLARE SER CURSOR FOR
    SELECT HIST_INT1, HIST_XML
    FROM USRT001.X01KTB WHERE HIST_INT1 > 0
FOR FETCH ONLY ;

EXEC SQL OPEN SER;
DO I=1 to TOTAL;
    EXEC SQL FETCH SER INTO :HIST_INT1,:HIST_CLOB;
END;
EXEC SQL CLOSE SER;

Figure 3-9 illustrates the result of executing the statement shown in Figure 3-9.

![Graph illustrating fetch performance](image)

We observe that the smallest document size (1 KB) takes 73 seconds of elapsed time and 72 seconds of CPU time when fetching one million XML documents. As the size of the XML document increases by a factor of 10 (and the number of documents fetched decreases by a factor of 10, to effectively retrieve a total of 1 GB), the CPU and elapsed times decrease to an average of 36 seconds.

Conclusion

Thus, retrieving XML documents from a DB2 table is scalable and delivers a reasonable performance return. The CPU and elapsed times are dependent on the XML document size. Hence, as the number of nodes in your XML document increases, the more expensive the serialization will be.
Chapter 3. XML

Recommendations
For best performance, we strongly recommend that you choose a proper XML document size. The XML document size should be chosen based on your application requirements.

3.3.4 Index exploitation

An XML index can be used to improve the efficiency of queries on XML documents that are stored in an XML column. In contrast to traditional relational indexes, where index keys are composed of one or more table columns that you specify, an XML index uses a particular XML pattern expression to index paths and values in XML documents stored within a single column. The data type of that column must be XML. Instead of providing access to the beginning of a document, index entries in an XML index provide access to nodes within the document by creating index keys based on XML pattern expressions. Because multiple parts of a XML document can satisfy an XML pattern, DB2 might generate multiple index keys when it inserts values for a single document into the index.

**Note:** XML indexing is supported only for the XMLEXISTS predicate.

The XMLEXISTS predicate can be used to restrict the set of rows that a query returns, based on the values in XML columns. The XMLEXISTS predicate specifies an XPath expression. If the XPath expression returns an empty sequence, the value of the XMLEXISTS predicate is false. Otherwise, XMLEXISTS returns true. Rows that correspond to an XMLEXISTS value of true are returned.

Performance measurement
A series of five statements were created to evaluate the performance of XML indexes. All measurements were performed using a System z9 processor running z/OS 1.8 using System Storage DS8300 disks.

Test cases
The five statements listed in Example 3-2 were strategically constructed to ensure that the DB2 optimizer chooses the desired access path. The XML index definitions are listed in E.2.1, “XML index definitions” on page 368.

**Example 3-2  Statements for test cases**

Statement 1 -- Ensures XML index is chosen over XML table space scan:

```
SELECT C_NAME, C_XML FROM USRT005.TPCD01 WHERE 
  XMLEXISTS ('/customer/customer_xml[@CUSTKEY="601"]' PASSING C_XML);
```

Statement 2 -- Ensures XML index is chosen with range predicate:

```
SELECT C_NAME, C_XML FROM USRT005.TPCD01 WHERE 
  XMLEXISTS ('/customer/customer_xml[@CUSTKEY>="800" and @CUSTKEY<="810"]' 
    PASSING C_XML)
```

Statement 3 -- Ensures an exact match takes place:

```
SELECT C_NAME, C_XML FROM USRT005.TPCD01 WHERE XMLEXISTS 
  ('/customer/customer_xml/order[@orderkey=583149028]' 
    PASSING C_XML);
```

Statement 4 -- Ensures partial filtering:

```
SELECT C_NAME, C_XML FROM USRT005.TPCD01 WHERE XMLEXISTS 
  ('/customer/customer_xml/order[@orderkey=583149028]/price' 
    PASSING C_XML);
```
Statement 5 -- Ensures a multiple index scan takes place:
SELECT C_NAME, C_XML FROM USRT005.TPCD01 WHERE XMLEXISTS
  ('/customer/customer_xml/order[@orderkey=583149028]' PASSING C_XML)  AND
  XMLEXISTS ('/customer/customer_xml[@CUSTKEY="9601"]' PASSING C_XML);

A comparative analysis was conducted in order to recognize the effect of XML indexes.
Figure 3-10 shows the result of this comparison.

Figure 3-10 presents a 99% reduction in elapsed time when using an XML index in
statements 1, 3, 4, and 5. Similarly, statement 2 delivers an 83% reduction in elapsed time.
You can view the output of the EXPLAIN statement in E.2, “XML index exploitation” on
page 368, to verify the access paths that were chosen for each query.

Conclusion
Undoubtedly, having a good XML index is critical to query performance. DB2 supports
path-specific value indexes on XML columns so that elements and attributes frequently used
in predicates and cross-document joins can be indexed.

Recommendations
To ensure best performance, we make the following recommendations regarding matching
between an XPath query and an XML pattern:
  ▶ Matching without // step is better than with // step.
  ▶ Matching without a wildcard (*) is better than with a wildcard.
  ▶ Matching with more steps is better than matching with fewer steps.
  ▶ Containment: An index with /c can be used to match the predicate for /a/b/c, but an index
    on /a/b/c cannot be used to match predicate //c.
  ▶ Data type: Data types have to match to trigger index exploitation.

In addition, in order to maximize performance, we strongly recommend that you execute the
RUNSTATS utility on both the base table space and corresponding XML table space with
INDEX(ALL) to keep access path statistics current.
### 3.3.5 Compression

XML documents tend to be quite large in comparison to other forms of data representation. However, as mentioned earlier in this chapter, the ubiquitousness of XML in the IT industry far outweighs its physical dimension. Therefore, compression of XML documents for efficient storage and transmission is required for the utmost manageability. DB2 V9 delivers XML compression capabilities to assist you in saving on storage.

**Important:** DB2 does not perform any specified compression on the XML table space until the next time that you run the REORG utility on this data.

#### Performance measurement

DB2 V9 XML compression method was tested and analyzed in order to assess its performance. All measurements were performed using a System z9 processor running z/OS 1.8 using Storage System DS8300 disks.

A sample set of UNIFI (International Standard ISO 20022) XML documents (for a total of 114 KB) was stored in a DB2 table using both compression options (STRIP WHITESPACE and PRESERVE WHITESPACE). In addition, a user XML document (10 KB) was stored using the same options. Figure 3-11 shows the results of this test.

![Graph showing XML compression performance](image)

Figure 3-11 presents a 68% space savings when compressing the UNIFI XML document with the STRIP WHITESPACE option. Furthermore, compressing the UNIFI document with the PRESERVE WHITESPACE option produces a storage reduction of 71%. Similarly, a user XML document exhibits a storage savings of 82% with the STRIP WHITESPACE option and 84% with the PRESERVE WHITESPACE option.
Conclusion
As a result, it is evident that there is a good compression ratio on an XML table space. Even with the PRESERVE WHITESPACE option, there are also significant disk storage savings. The CPU behavior is similar to the relational model. However, there is a significant CPU impact if you select many documents (DocScan).

Recommendations
Thus, to maximize your space savings, we recommend that you use the default option (STRIP WHITESPACE) for the XML document. In addition, using this default option assists in the performance of the serialization process.
DB2 subsystem performance

In this chapter, we discuss several topics that are related to enhancements that affect DB2 subsystem performance. In this chapter, we present the following sections:

- CPU utilization in the DB2 9 for z/OS
  We provide a comparison between equivalent workloads on DB2 V8 and DB2 9. We show that most of the workloads benefit from CPU and elapsed time improvement.

- CPU utilization in the client/server area
  We provide a comparison between the same workload on DB2 V8 and DB2 9. We show that most of the workloads benefit from CPU improvement.

- z10 and DB2 workload measurements
  We introduce the IBM z10 Enterprise Class and a set of early workload measurements to highlight the possible range of CPU improvements.

- The IBM zEnterprise System and DB2
  We mention the new zEnterprise System and some of the expected improvements coming from its hosting DB2 subsystems.

- Virtual storage constraint relief
  DB2 V8 introduces support for 64-bit storage in the DBM1 and IRLM address spaces. This can provide significant virtual storage constraint relief (VSCR). DB2 9 has extended VSCR. We first explain what has changed in V9 and then evaluate the impact of these enhancements on virtual storage usage.

- Real storage
  Real storage utilization patterns have changed in DB2 9. We explain what has changed and the impact these changes can have on your processing environment.

- Distributed 64-bit DDF
  Distributed storage distributed data facility (DDF) has changed to use 64-bit processing, which extends VSCR in DB2 9 to include the DIST address space. We first explain what has changed in DDF processing and evaluate the impact of the changes.

- Distributed address space virtual storage
Distributed address space virtual storage usage has changed due to the DDF 64-bit processing. We show the change in the DDF address space virtual storage usage.

- Distributed workload throughput
  Distributed workload throughput has improved due to the DDF 64-bit processing. We show the improvement gained by the 64-bit enhancement.

- WLM assisted buffer pool management
  A new autonomic technique to manage buffer pool sizes via Workload Manager (WLM) depending on utilization has been introduced with DB2 V9.

- Automatic identification of latch contention and DBM1 below-the-bar virtual storage
  A monitor checks for an abnormal CPU and virtual storage situation and provides messages based on different thresholds.

- Latch class contention relief
  We describe a number of common latch contention problems and explain how these problems have been addressed in DB2 9.

- Accounting trace overhead
  We show the measured overhead with various accounting traces running. This helps you select the accounting level detail that is appropriate for your environment.

- Reordered row format
  We describe the changes and how these changes can affect the performance.

- Buffer manager enhancements
  A number of changes have occurred in the Buffer Manager. We describe how these changes can help your environment.

- WORKFILE and TEMP database merge
  WORKFILE and TEMP databases have merged. We describe the changes and how they affect your system.

- Native SQL procedures
  SQL procedures are now able to be run natively in DB2 9. We describe this change and the performance improvements that it brings.

- Index look-aside
  Index look-aside has been enhanced to reduce the amount of effort that is needed to process an index. We explain how this works and the benefit it can bring.

- Enhanced preformatting
  Enhanced preformatting reduces the amount of time that is spent waiting for preformat actions to complete.

- Hardware enhancements
  We show how DB2 9 is exploiting its synergy with the IBM System z processors.

- Optimization Service Center support in the DB2 engine
  We describe how the Optimization Service Center monitor can affect CPU utilization.
4.1 CPU utilization in the DB2 9 for z/OS

We compared a number of workloads that were run against DB2 9 to test the DB2 engine that is used for processing. We split the workloads so that we can measure the throughput for certain types of workloads. In this section, we look at three performance areas: OLTP, query processing, and column processing.

4.1.1 OLTP processing

We ran and measured workloads that emulate an online transaction processing (OLTP) workload against DB2 V8. The results were compared to the same workload on DB2 9. The comparison showed that, for this OLTP workload test, we realized an overall CPU utilization improvement for both data sharing and non-data sharing.

![OLTP workload improvement DB2 9 versus DB2 V8](image)

General OLTP CPU utilization improvements of 1.4% were measured for non-data sharing environments. Data sharing CPU utilization improvement was measured at 3.3%. The factors involved in the general OLTP improvements were the enhancements to the index manager, the change in the declaim processing (see 4.15, “Buffer manager enhancements” on page 124), and some streamlining in the DB2 instrumentation area.

The data sharing improvements are a combination of the general improvements that are already detailed for the non-data sharing environment and two data sharing-specific changes. One change is the reduction in log-latch contention, in this case a reduction of ten times. For more information, and for details about latch class 19, see 4.12, “Latch class contention relief” on page 114. The other change is the elimination of cross invalidations for secondary group buffer pools. For group buffer pools that are duplexed, DB2 9 eliminates cross invalidations as a result of the secondary being updated.
4.1.2 Query processing

We ran and measured a query workload to compare the DB2 V8 and DB2 V9 elapsed times for a suite of queries. This benchmark consisted of a suite of 140 business-oriented queries and concurrent data modifications. These queries are generally considered to be more complex than most OLTP transactions.

Out of the 140 queries, 95 of them (68%) had access path changes in DB2 9 compared to DB2 V8. When the complete suite was run, there was an overall 15.8% improvement in elapsed times and a 1.4% CPU utilization improvement.

4.1.3 Column processing

We compared the same batch workload that we ran in a DB2 V8 environment to a DB2 9 environment. This batch workload included column processing functions in input and output. The output column and input column processing values are an average of processing seven data types. This is a better reflection of the normal type of workload processing since it is not concentrating on one particular data type.

Most of the column processing functions in DB2 9 show a reduction in CPU utilization when compared to the same DB2 V8 workload, on average, about 13% in output and 7% in input.

4.2 CPU utilization in the client/server area

A number of standard OLTP distributed workloads were tested against a DB2 V8 server configuration and then the same workload was run against a DB2 9 server on the same hardware. Testing was done using the IBM Relational Warehouse Workload distributed workload, which simulates a typical customer workload against a warehouse database. The Relational Warehouse Workload is described in DB2 for MVS/ESA V4 Data Sharing Performance Topics, SG24-4611.

The measurements were taken on the DB2 server side and plotted as a graph of the workload (CPU per commit) in Figure 4-2. These measurements include the TCP/IP communication cost as well as the cost for the six workloads that were measured. This reflects the overall workload better. The workloads were done in a logical partition (LPAR) with no System z9 Integrated Information Processors (zIIPs) available. Doing this gave us a better measurement without having to take into account the zIIP processing.

The measurements of the CPU time per commit for the distributed Relational Warehouse Workload remained approximately the same. The largest increase was 1.1% and the largest decrease was 5.4%, with four out of the six workloads showing a decrease in CPU usage per distributed commit. The SQLCLI and JDBC workloads showed a slight increase in CPU utilization per commit. In all the other workloads, the CPU usage remained approximately static or decreased.

The bars in Figure 4-2 should be compared as pairs. Each pair is the DB2 V8 and DB2 V9 measurements with the DB2 V9 percentage change noted above the DB2 V9 column.
The following legend explains the workloads shown in Figure 4-2:

- SQLCLI was an ASCII DB2 Connect™ Client using call-level interface (CLI).
- SQLEMB was an ASCII DB2 Connect Client using Embedded SQL.
- STCLI was an ASCII DB2 Connect Client using CLI Stored Procedure calls.
- STEM B was an ASCII DB2 Connect Client using Embedded Stored Procedure calls.
- SQLJ was a UTF-8 Client using Java Common Connectivity (JCC) Type4 Driver and SQLJ.
- JDBC was a UTF-8 Client using JCC Type4 Driver and JDBC.

The same Relational Warehouse Workload was measured on a system with one zIIP enabled to compare the DB2 V8 and DB2 V9 distributed workloads from a Distributed Relational Database Architecture (DRDA) zIIP redirect perspective. The measurements showed that there was no significant difference between the DB2 V8 and DB2 V9 workloads using the zIIP redirect. The expected percentages for the zIIP redirect were achieved with no throughput degradation.

4.2.1 Conclusion

There is an overall small percentage reduction in CPU usage comparing the same distributed workload between DB2 V8 and DB2 V9. You should see equivalent CPU usage and throughput between DB2 V8 and all modes of DB2 V9 for a comparable distributed workload.

4.3 z10 and DB2 workload measurements

On February 26th 2008, IBM has announced the System z10® Enterprise Class (z10 EC). See Figure 4-3. This mainframe has a processor speed of 4.4 GHz, has up to 1.5 TB of memory, new open connectivity options and delivers performance and capacity growth drawing upon the heritage of the z/Architecture servers. The z10 EC is a general purpose
server for compute intensive workloads (such as business intelligence) and I/O intensive workloads (such as transaction and batch processing).

The z10 EC supports up to 60 Logical Partitions (LPARs): each one can run any of the supported operating systems: z/OS, z/VM®, z/VSE™, z/TPF, and Linux on System z. You can configure from a 1-way to a 64-way Symmetrical Multiprocessor (SMP) processing power.

The System z10 Enterprise Class offers five models: E12, E26, E40, E56 and E64. The names represent the maximum number of configurable processors (CPs) in the model.

The z10 EC continues to offer all the specialty engines available with its predecessor z9:

- **ICF - Internal Coupling Facility**
  Used for z/OS clustering. ICFs are dedicated for this purpose and exclusively run Coupling Facility Control Code (CFCC).

- **IFL - Integrated Facility for Linux**
  Exploited by Linux and for z/VM processing in support of Linux. z/VM is often used to host multiple Linux virtual machines (called guests.)

- **SAP - System Assist Processor**
  Offloads and manages I/O operations. Several are standard with the z10 EC. More may be configured if additional I/O processing capacity needed.

- **zAAP - System z10 Application Assist Processor**
  Exploited under z/OS for designated workloads which include the IBM JVM and some XML System Services functions.

- **zIIP - System z10 Integrated Information Processor**
  Exploited under z/OS for designated workloads which include some XML System Services, IPSec off-load, part of DB2 DRDA, complex parallel queries, utilities, global mirroring (XRC) and some third party vendor (ISV) work.

The z10 EC also continues to use the Cryptographic Assist Architecture first implemented on z990. Further enhancements have been made to the z10 EC CPACF.
Refer to the following Redbooks, for more information:

- IBM System z10 Enterprise Class Technical Introduction, SG24-7515
- IBM System z10 Enterprise Class Technical Guide, SG24-7516
- IBM System z10 Enterprise Class Configuration Setup, SG24-7571
- IBM System z10 Enterprise Class Capacity on Demand, SG24-7504

### 4.3.1 z10 performance

The z10 EC Model E64 offers approximately 1.5 times more capacity than the z9 EC Model S54 system, however, compared to previous processors, the z10 CPU time is more sensitive to the type of workload. Based on customer and laboratory measurements a range of 1.2 to 2.1 times capacity improvements has been observed.

The z10 EC has been specifically designed to focus on new and emerging workloads where the speed of the processor is a dominant factor in performance. The result is a jump in clock speed from the z9 1.7 GHz to the z10 4.4 GHz. The storage hierarchy design of the z10 EC is also improved over z9 EC, however, the improvement is somewhat limited by the laws of physics so the latencies have increased relative to the clock speed.

Workloads that are CPU-intensive tend to run above average (towards 2.1) while workloads that are storage-intensive tend to run below average (towards 1.2).

IBM continues to measure the systems' performance using a variety of workloads and publishes the result on the Large Systems Performance Reference (LSPR) report. The LSPR is available at:

http://www.ibm.com/servers/eserver/zseries/lspr/

The MSU ratings are available on the Web:


The LSPR workload set comprises the following workloads:

- Traditional online transaction processing workload OLTP-T
- Web enabled online transaction processing workload OLTP-W
- A heavy Java based online stock trading application WASDB
- Batch processing, represented by the CB-L (commercial batch with long running jobs)
- A new ODE-B Java batch workload, replacing the CB-J workload.

The LSPR provides performance ratios for individual workloads as well as for the ‘default mixed workload’ which is composed of equal amounts of the five workloads described above.

The LSPR Web site also contains a set of answers for frequently asked questions, several on the performance of z10, at:


zPCR is the Processor Capacity Reference for IBM System z to capacity plan for IBM System z and eServer™ zSeries® processors. Capacity results are based on IBM's LSPR data supporting all IBM System z processors (including the System z10). For getting started with zPCR, refer to:

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS1381

### 4.3.2 DB2 performance with z10

DB2 9 for z/OS can potentially benefit from several of the enhanced functions of the z10:
- The new z10 has faster processors and more processors. Note that z/OS 1.9 is needed for 64-way in a single LPAR.
- Larger memory: DB2 users can potentially see higher throughput with more memory used for DB2 buffer pools, EDM pools or SORT pools. Note that z/OS 1.8 is needed for >256 GB in a single LPAR.
- Improved IO: Improvements in the catalog and allocation can make the large number of datasets much faster and easier to manage. Disk IO times and constraints can be reduced.
- Substantial improvements in XML parsing can result from use of the zIIP and zAAP specialty engines. The z10 zIIP and zAAP engines are much faster than z9 at no additional cost. The zIIP processors can be used for XRC processing.
- Other functions of interest:
  - Infiniband CF links
  - New OSA-Express3, 10 GbE for faster remote applications
  - HiperDispatch
  - Hardware Decimal Floating Point facility
  - Extended Addressing Volumes up to 223 GB/volume
  - z/OS XML performance enhancements
  - TCP/IP performance enhancements
  - HiperSockets™ Multiple Write Facility for better performance for larger message sizes (DRDA)
  - Basic Hyperswap2 to remove DASD controller as a single point of failure
  - zIIP assisted Global Mirroring (XRC)
- Functions of future interest for DB2 exploitation
  - 1 MB page size
  - 50+ instructions added to improve compiled code efficiency

Detailed DB2 workload measurements comparing z10 with z9 are under way in the lab to delineate the ranges depending on workloads and set the right expectations.

The preliminary results are summarized in Figure 4-4.
Figure 4-4 shows the ratio in CPU time reduction between z10 and z9. As a reference, 2 times improvement corresponds to 50% CPU reduction, 1.43 times corresponds to 30% CPU reduction. The Ln values have been measured in laboratory environment, the Cn values are customers measurements.

The workloads are clarified in Table 4-1.

**Table 4-1 Workload characteristics**

<table>
<thead>
<tr>
<th>Identifier in Figure 4-4</th>
<th>Workload</th>
<th>CPU improvement</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>Query</td>
<td>TPC-D like</td>
<td>2.00</td>
</tr>
<tr>
<td>L2</td>
<td>OLTP</td>
<td>online brokerage</td>
<td>1.68</td>
</tr>
<tr>
<td>L3</td>
<td>Simulated OLTP</td>
<td>1.67</td>
<td></td>
</tr>
<tr>
<td>L4</td>
<td>IRWW distributed (CLI)</td>
<td>1.62</td>
<td></td>
</tr>
<tr>
<td>L5</td>
<td>Open/Close intensive</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>L6</td>
<td>Utility</td>
<td>REORG TABLESPACE</td>
<td>1.42</td>
</tr>
<tr>
<td>L7</td>
<td>Utility</td>
<td>RUNSTATS</td>
<td>2.13</td>
</tr>
<tr>
<td>L8</td>
<td>Utility</td>
<td>TABLESPACE,TABLE,IX</td>
<td></td>
</tr>
<tr>
<td>L9</td>
<td>Data sharing and non-data sharing</td>
<td>SELECT and FETCH</td>
<td>1.9</td>
</tr>
<tr>
<td>L10</td>
<td>Data sharing</td>
<td>Single row INSERT</td>
<td>1.22</td>
</tr>
<tr>
<td>L11</td>
<td>Data sharing</td>
<td>Multi row INSERT</td>
<td>1.01</td>
</tr>
</tbody>
</table>
The observed DB2 values have shown a wide range from about 1 to 2.1.

Normal workloads go from 1.4 for OLTP transactions to 2 for query intensive workload. The lower values are likely if the environment is not well tuned and the DB2 address spaces show a high percentage of CPU utilization in comparison with the total (including the applications) DB2 CPU time.

However, workloads with intensive Open/Close activity show 1.2 ratio due to the higher cost of storage intensive activity.

Intensive INSERT activity in data sharing shows even smaller ratios (.99 to 1.2). Intensive localized INSERT, especially in case of multi-row, suffers from the so-called ‘spinning’ problem described in 4.12.2, “Latch class 19” on page 116 and heightened by the fast z10 processor. This situation is being investigated.

**Service Units on z10 compared to z9**

Service Unit is derived from the formula: Service Unit = CPU time in seconds * SRM constant. SRM constant is a CPU model dependent coefficient used by SRM to convert CPU time to Service Unit. The assumed SRM constant is 1.58 times smaller than z9. This means that, if a given DB2 workload CPU time does not improve by 1.58 times on z10 compared to z9, then the Service Unit on z10 will increase proportionately.

Since the z10 CPU time improvement over z9 can range from 1.2 to 2.1 times depending on the workload, the Service Unit can either increase or decrease depending on the workload.

For DB2 online transaction workload executing short-running SQL calls, the z10 Service Unit tends to increase over z9. On the other hand, for DB2 query workload executing long-running SQL calls, the z10 Service Unit tends to be smaller than z9.

**Details of DB2 query workload measurements**

In this section we provide details of measurements by DB2 development laboratory related to the query workload shown as L1 in Figure 4-4 on page 89.

This DB2 9 for z/OS query workload is a TPC-D like benchmark with a database of 30 GB. It consists of total of 141 internally developed queries (131 parallel and 10 sequential).

The measurements have been done comparing two environments with 3 CPs on z9 and z10.

The TPC-D like query performance comparison in terms of average elapsed time and CPU time (in seconds) is reported in Figure 4-5.

<table>
<thead>
<tr>
<th>Identifier in Figure 4-4</th>
<th>Workload</th>
<th>CPU improvement</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>L12</td>
<td>XML query</td>
<td>1.70</td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>OLTP CICS®</td>
<td>1.51</td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>OLTP CICS</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>High Open/close</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>Data sharing batch SELECT</td>
<td>1.42</td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>INSERT/DELETE</td>
<td>.99 (under study)</td>
<td></td>
</tr>
</tbody>
</table>
The average CPU time is reduced by 50% (a 2 times improvement).

The throughput increases but it is not reported since the I/O configuration was different for the two environments. No issues are present for access path determination.

4.3.3 Conclusion

The synergy of DB2 for z/OS with System z platform continues. The combined improvements provided by z/OS V1.10, the z10 EC, and storage can mean significant scalability, resiliency, security, workload management, and price performance capabilities for your workloads. Existing DB2 for z/OS workloads can gain benefit from many improvements: z/OS V1.10 Contention Management and hashed DSAB searches; EAV; Basic HyperSwap®; HiperDispatch; IBM System Storage DS8000 series AMP (Adaptive Multi-stream Prefetching); and z10 EC CPUs, memory, I/O and network bandwidth.

However, the improvement depends on the type of workloads, with DB2 9 for z/OS new workloads potentially gaining more advantage from z/OS V1.10 additional XML exploitation of the zIIP specialty processor, and the z10 EC server’s increased performance efficiency for the decimal float data type initially enabled in DB2 9.

4.4 The IBM zEnterprise System and DB2

On July 22nd 2010, IBM has announced the IBM zEnterprise™ System (see IBM United States Hardware Announcement 110-170).

The system, see Figure 4-6, consists of the IBM zEnterprise 196 (z196™) central processor complex (CPC), the IBM zEnterprise Unified Resource Manager, and built-in support for the IBM zEnterprise BladeCenter® Extension (zBX).
The IBM zEnterprise 196 is designed with improved scalability, performance, security, resiliency, availability, and virtualization. The z196 Model M80 provides up to 1.6 times the total system capacity of the z10™ EC Model E64, and all z196 models provide up to twice the available memory of the z10 EC. The zBX infrastructure works with the IBM zEnterprise 196 to enhance System z virtualization and management to deliver an integrated hardware platform that spans System z mainframe and POWER7™ technologies. The IBM zEnterprise Unified Resource Manager, delivered with the z196, is designed to deliver end-to-end virtualization and management along with the ability to optimize technology deployment according to individual workload requirements.

The new IBM zEnterprise BladeCenter Extension (zBX) Model 002 is designed to provide a redundant hardware infrastructure that supports the multiplatform environment of z196. The zBX Model 001 allows the management of heterogeneous resources to support the IBM Smart Analytics Optimizer application as a single entity of the System z10. These infrastructures allow z196 and the System z10 to extend System z qualities of service and management to a new set of devices.

### 4.4.1 Benefits of zEnterprise for DB2

Figure 4-7 summarizes the expected benefits that the zEnterprise may provide to DB2 for z/OS systems.
4.4.2 IBM Smart Analytics Optimizer for DB2 for z/OS

On the same day, IBM United States Software Announcement 210-266 has introduced the IBM Smart Analytics Optimizer for DB2 for z/OS. This software function enables a new class of business intelligence and data warehousing workloads for the IBM System z platform.

IBM Smart Analytics Optimizer for DB2 for z/OS, V1.1, together with related hardware, is a high-performance, integrated hardware/software solution for IBM System z, delivering dramatically faster analytic query responses transparently to users.

Figure 4-8 summarizes the characteristics of this solution.
IBM Smart Analytics Optimizer

What is it?
• A special purpose, network-attached appliance that is an add-on to a DB2 for z/OS system
• Offloads typical DW/BI queries resulting in predictable and orders-of-magnitude faster query response times while reducing overall TCO

Business value
• Dramatically lowers the cost for query and reporting on System z
• Advanced in-memory scale-out cluster technologies that keep the complete system centrally managed without having to change any requirements for BI applications
• Complements the many new Data Warehousing features in DB2 9 for z/OS
• Leverages the many new warehousing and business intelligence solutions now available on System z

Expected uses for DB2 for z/OS customers:
• Requirements to accelerate a subset of their warehouse or reporting queries
• Looking for more insight and business intelligence from operational data
• Needs to consolidate datamarts or data stores into one enterprise warehouse

Figure 4-8 IBM Smart Analytics Optimizer

The IBM Smart Analytics Optimizer integrates into an IBM DB2 9 for z/OS (or later versions) data warehouse environment, providing high-performance query software that is based on advanced data in memory technology and executed on an IBM zEnterpise BladeCenter Extension (zBX) attached to and managed by the System z server. This offering helps enable a new class of high-speed business intelligence (BI) queries never before available on System z. It extends System z qualities of service of manageability, security, and availability to analytic applications, while seamlessly combining IBM’s hardware and software worlds.

The following IBM Redbooks publications provide information on these products:
- IBM zEnterprise System Technical Guide, SG24-7833
- IBM zEnterprise BladeCenter Extension Model 001, REDP-4668

4.5 Virtual storage constraint relief

DB2 9 for z/OS has moved more DB2 structures above the bar in the DBM1 address space to provide extra VSCR for the DB2 structures that still need to reside below the bar. Part of the environmental descriptor manager (EDM) pool storage has moved above the bar. The skeleton cursor table (SKCT) and the skeleton package table (SKPT) have moved above the bar. The static SQL sections, cursor table (CT) and package table (PT), have been split into sections that reside both above the bar and below the bar.

EDM storage is composed of these components, each of which is in a separate storage area:
- **EDM RDS pool below**: A below-the-bar pool that contains the part of the cursor tables (CTs) and package tables (PTs) that must be below the bar.
- **EDM RDS pool above**: An above-the-bar pool that contains the part of the PTs and CTs that can be above the bar.
- **EDM DBD pool**: An above-the-bar pool that contains database descriptors (DBDs).
- **EDM statement pool above**: An above-the-bar pool that contains dynamic cached statements.
- **EDM skeleton pool**: An above-the-bar pool that contains SKPTs and SKCTs.

Moving the plan and package skeletons completely above the bar into their separate EDM pools is expected to provide the most VSCR for DB2 subsystems that primarily use plan/package execution. In some cases, this relief can be as much as 200 Mb, or greater, in the DBM1 address space. It frees this storage for other below-the-bar constrained processes.

In DB2 9, we have seen an average reduction of approximately 60% in below-the-bar storage for the EDM pool. However a wide variation in storage reduction has been seen with values as low as 20% and as high as 85%. The amount of EDM pool reduction below the bar in your environment varies and is dependent on your workload mix.

In DB2 9, all of the hash tables, object blocks, and mapping blocks that are associated with the EDM fixed storage pools are moved above the bar. These storage pools contain the small mapping control blocks that map each block of EDM storage (above or below) as well as the larger object identifying control blocks. Moving DB2 V9 usage of these control blocks above the bar allows scaling of the number of EDM objects without affecting below-the-bar storage usage.

A change to the placement of control block structures for dynamic SQL-related storage has moved these to be in above-the-bar storage. In DB2 V8 with APAR PQ96772, two out of the three control blocks that are associated with dynamic statement caching moved above the bar. DB2 V9 has moved the last control block that is associated with each SQL statement to above-the-bar storage. This has the benefit that the EDM statement cache pool can now expand as necessary above the bar without resulting in equivalent below-the-bar storage increases to contain any statement cache control blocks.

Dynamic statements have a larger portion split above the bar than for static SQL bound in DB2 V9 plans or packages. This split has to do with the extra storage that is required in dynamic statement storage for DESCRIBE column information and PREPARE options that do not exist as part of the statement storage for static statements. This DESCRIBE and PREPARE storage is all moved above the bar. As a generalization, individual statement storage is larger for dynamic SQL than for static SQL.

Peak storage usage for bind/dynamic PREPARE has been reduced by moving some short-term control blocks above the bar. The most significant of these is parse tree storage, which is short-term storage that is held during the full prepare of statements. The peak storage usage for parse trees below the bar has reduced by 10% for full prepares. Miniplan storage and some tracing storage have now moved above the bar.

User thread storage, system thread storage, and stack storage values remain about the same as DB2 V8. However some of the relief actions taken in DB2 V9 can be offset by V9 storage increases in other functional areas.

### 4.5.1 EDM pool changes for static SQL statements

DB2 V9 has started the process of moving the plan and package static statement storage above the bar. You will find that each statement bound on V9 now has a below-the-bar portion and an above-the-bar portion. This data relocation work has targeted certain statement types. Some statements have as much as 90% of the statement storage moved above the bar and some show less than 5% moved above the bar.
To achieve the VSCR in the below-the-bar storage for the CT and PT, you need to rebind your plans and packages in DB2 V9 to move the relevant sections to 64-bit addressing. This rebinding causes the plan sections to be eligible for loading into 64-bit addressable storage.

The above-the-bar EDM RDS pool for the CT and PT structures has a default size of 2 GB of above-the-bar storage. This value is not an installed clist changeable value. If you experience problems with this default sizing for this pool, then contact IBM service to understand the options that are available for this pool.

The total storage used for statements below the bar and above the bar in DB2 V9 is significantly larger than it was in V8. This extra storage usage happens only when a REBIND is done on DB2 V9.

The storage usage below-the-bar and above-the-bar information is collected in new IFCID 225 and IFCID 217 fields. These values can be reported on with OMEGAMON; see 4.5.6, “Instrumentation and processes for virtual storage monitoring” on page 99, for more details.

In Table 4-2, we show a virtual storage usage comparison on virtual storage below the bar between a DB2 V8 and DB2 V9 system running the same workload. These DB2 V8 and DB2 V9 test results were taken from a non-data sharing DB2 subsystem that was running with 360 active threads driven by an enterprise resource planning (ERP) distributed workload. The DB2 V8 and DB2 V9 subsystems were both running in new-function mode and the measurements were taken on the same hardware and operating system platform, which was a System z9 processor running z/OS 1.7. The same workload and configuration were used for the real storage measurements that are presented later.

<table>
<thead>
<tr>
<th>Virtual storage below 2 GB</th>
<th>V8</th>
<th>V9</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBM1 below 2 GB used, out of 1774 MB available</td>
<td>1091 MB</td>
<td>819 MB</td>
</tr>
<tr>
<td>Local dynamic statement cache</td>
<td>466</td>
<td>172</td>
</tr>
<tr>
<td>Thread/stack storage</td>
<td>500</td>
<td>528</td>
</tr>
<tr>
<td>EDM</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>DBM1 real storage</td>
<td>1935</td>
<td>2203</td>
</tr>
</tbody>
</table>

### 4.5.2 EDM pool changes for dynamic SQL statements

In DB2 V8, there are 30 global, variable-length storage pools that contain the dynamic SQL statement storage that is used for statement execution located in below-the-bar storage. In DB2 V9, the individual statements have their storage split between above-the-bar and below-the-bar portions. This split has created 30 above-the-bar variable-length pools to contain the above-the-bar portion of the dynamic SQL statement storage. These pools are treated the same way as the below-the-bar pools.

Data collected via the DB2 IFCID 225 storage statistics summary record can be used for checking with the MAXKEEPD DSNZPARM to keep track of how the local statements cache is being used. MAXKEEPD is used to limit the number of dynamic statements that are to be held in the cache. The QW0225LC and QW0225HC fields are the values that can be used for this comparison. IFCID 225 is used for the storage manager pool summary statistics and is produced at every DB2 statistics interval, controlled by the STATIME DSNZPARM.

The QW0225LC field is the number of statements in the cached SQL statements pools. QW0225HC is the high water mark of the number of statements at the period of the highest
storage utilization in the cached SQL statement pools. Both of these values relate to below-the-bar storage.

Using these storage utilization values instead of pool totals is useful for understanding how peaks in cached SQL statement usage drive the total storage usage in the DBM1 address space. Typically, the dynamic SQL execution storage can be extensive and may need close monitoring. It can be one of the largest factors in driving DBM1 below-the-bar virtual storage constraint.

There are no separate IFCID 225 instrumentation fields for the accumulated size of the 30 above-the-bar statement cache pools. This data can be found in the IFCID 217 storage detail record, which is also now cut at the DB2 statistics interval if it is activated. The IFCID 217 statistics collections can be activated with the following command:

-START TRACE(GLOBAL) IFCID(217)

Both the IFCID 217 and IFCID 225 are written as SMF 102 type records. The SMF data can be processed to provide a report on these dynamic SQL storage utilization values. The above-the-bar accumulated pool sizes for the cached SQL statements are backed by real storage as they expand on demand and take storage that is in “first referenced” state.

4.5.3 Below-the-bar EDM pool

Below-the-bar EDM pool now has no least recently used (LRU) objects, which reduces the EDM latch class 24 serialization contention. However because of the removal of the LRU objects, the below-the-bar EDM pool must be sized to contain peak usage plus a cushion for fragmentation. The below-the-bar EDM pool should be sized to accommodate between 110% and 130% of peak usage to take into account pool fragmentation and the need to allocate contiguous storage for any section. Because of the removal of the LRU objects from the below-the-bar EDM pool, it is possible to have an EDM pool full conditions for this storage pool. Because there is no automatic expansion or contraction of this below-the-bar EDM pool, you need to modify this via a DSNZPARM change and activate your change via the SET SYSPARM LOAD. This is the only way to expand, or contract, the below-the-bar EDM pool.

If you do get EDM pool full situations and need a way to identify what the failing requests are then start a trace for IFCID 31. This trace record is only ever produced when an EDM pool full situation is encountered. The IFCID 31 identifies the object type, size, and requestor. When the trace is active, any EDM pool full condition results in this trace record being produced.

You can view the EDM statistics in IFCID00 2 collected from a DB2 statistics trace or from an IFI request. Refer to DSNWMSG in the SDSNSAMP library for field definitions of this trace record (DSNDQWS1 and DSNDQISE in SDSNMACS further define IFCID 2).

4.5.4 CACHEDYN_FREELOCAL

There is the potential that a high number of active dynamic SQL statements can cause a critical below-the-bar storage shortage. This is usually a temporary spike in the number of threads running, and these threads hold an increasing number of SQL statements in the local cache. With increasing statements in the local cache, the below-the-bar storage cushion that is available to DB2 decreases to a critical value causing system problems, 00E200xx storage related abends, or other unpredictable storage-related errors.

To help alleviate this potential below-the-bar storage cushion-related problem, a new DSNZPARM in the DSN6SPRM macro has been introduced. The new subsystem parameter CACHEDYN_FREELOCAL indicates whether DB2 can free cached dynamic statements to
relieve DBM1 below-the-bar storage. CACHEDYN_FREELOCAL applies only when the KEEP_DYNAMIC(YES) bind option is active. The default value is 1, which means that DB2 frees some cached dynamic statements to relieve high use of storage when the cached SQL statement pools have grown to a certain size. If you specify 0, DB2 does not free cached dynamic statements to relieve high use of storage by dynamic SQL caching.

Table 4-3 shows the possible settings of the CACHEDYN_FREELOCAL value and how these values affect DB2 V9 management of the local SQL cache below-the-bar. The settings between 0 and 3 identify the triggering levels that DB2 V9 uses to attempt to reclaim local SQL cache below-the-bar storage.

### Table 4-3  CACHEDYN_FREELOCAL settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>Statement size</th>
<th>Local SQL cache size (below the bar) in MB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 KB</td>
<td>All</td>
</tr>
<tr>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>75</td>
<td>85</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>88</td>
</tr>
<tr>
<td>3</td>
<td>75</td>
<td>88</td>
</tr>
</tbody>
</table>

With CACHEDYN_FREELOCAL set to 1, if the local SQL cache below-the-bar storage exceeds 500 MB, and the percent used of extended private DBM1 below the bar exceeds 75% of the available storage, then statements larger than 100 KB begin to be released at close if possible. If the local SQL cache storage below-the-bar growth continues and the used percent of extended private DBM1 below-the-bar usage exceeds 85%, then all statements are candidates to be released as they are closed.

By varying the value of CACHEDYN_FREELOCAL, you change the triggering level where DB2 V9 tries to free local SQL cache storage below the bar. The CACHEDYN_FREELOCAL DSNZPARM is changeable online via the SET SYSPARM RELOAD command, so the triggering level can be changed without interruption to your DB2 subsystem.

### 4.5.5 Automated memory monitoring

DB2 V9 has implemented a time-based memory monitoring function that can help to alert you when certain storage utilization thresholds are reached. In DB2 V9, the built-in monitor runs from startup to shutdown and checks the health of the system at one-minute intervals. As part of this built-in monitor, the DBM1 storage below the bar is monitored for critical storage increases. When the DBM1 storage below the bar reaches specific thresholds of 88, 92, 96, or 98 percent used of the available storage, messages (DSNV508I, DSNV510I, DSNV511I, and DSNV512I) are issued. These messages report current DBM1 storage consumption and indicate the agents that are consuming the most storage.

Example 4-1 shows the type of output that is written by the built-in DBM1 storage monitor to the console.

### Example 4-1  Storage monitor messages

```
DSNV508I -SE20 DSNVMON - DB2 DBM1 BELOW-THE-BAR 09
STORAGE NOTIFICATION
  91% CONSUMED
  87% CONSUMED BY DB2
```
4.5.6 Instrumentation and processes for virtual storage monitoring

Instrumentation has been added to report on the usage of these new and changed virtual storage pools. The information that is collected and presented in IFCID 225 is summary information. It presents a snapshot of the storage usage in the DBM1 address space at each DB2 statistics interval. The collection of this data has a small overhead in terms of CPU consumption and does not produce large amounts of SMF data. This summary information is available through DB2 Statistics Trace Class 6. We currently recommend that you set the DB2 ZPARMS STATIME value to five and that you set the ZPARMS SYNCVAL to zero.

You can gather more detailed information at the thread level via the IFCID 217 collection. The IFCID 217 information is also now cut at the DB2 statistics interval. The IFCID 217 statistics collections can be activated with the following command:

```
-START TRACE(GLOBAL) IFCID(217)
```

The information in IFCID 217 and IFCID 225 is written out in the SMF type 102 record and can be processed from this source. The use of a reporting tool, such as IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, DB2 Performance Expert, or IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS, can produce reports that show the storage usage.

Example 4-2 shows the changes to the DSNDQISE macro from the IBM-supplied SDSNMACS library. DSNDQISE is the mapping macro for the instrumentation data for mapping the EDM pool statistics. The new fields hold the instrumentation data for the counters for some of the storage location changes. This data is also externalized in IFCID225 where it can be processed by OMEGAMON.

Example 4-2  SDSNMACS(DSNDQISE)

<table>
<thead>
<tr>
<th>DSNDQISE</th>
<th>DSNDQISE</th>
<th>DSNDQISE</th>
<th>DSNDQISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>QISEKFAL DS</td>
<td>QISEKFAL DS</td>
<td>QISEKFAL DS</td>
<td>QISEKFAL DS</td>
</tr>
<tr>
<td>QISEKPGE DS</td>
<td>QISEKPGE DS</td>
<td>QISEKPGE DS</td>
<td>QISEKPGE DS</td>
</tr>
<tr>
<td>QISEKFRE DS</td>
<td>QISEKFRE DS</td>
<td>QISEKFRE DS</td>
<td>QISEKFRE DS</td>
</tr>
<tr>
<td>QISECTA DS</td>
<td>QISECTA DS</td>
<td>QISECTA DS</td>
<td>QISECTA DS</td>
</tr>
<tr>
<td>QISEKTA DS</td>
<td>QISEKTA DS</td>
<td>QISEKTA DS</td>
<td>QISEKTA DS</td>
</tr>
<tr>
<td>QISESFAL DS</td>
<td>QISESFAL DS</td>
<td>QISESFAL DS</td>
<td>QISESFAL DS</td>
</tr>
<tr>
<td>QISESPGE DS</td>
<td>QISESPGE DS</td>
<td>QISESPGE DS</td>
<td>QISESPGE DS</td>
</tr>
<tr>
<td>QISESFRE DS</td>
<td>QISESFRE DS</td>
<td>QISESFRE DS</td>
<td>QISESFRE DS</td>
</tr>
<tr>
<td>QISEKNFN DS</td>
<td>QISEKNFN DS</td>
<td>QISEKNFN DS</td>
<td>QISEKNFN DS</td>
</tr>
<tr>
<td>QISEKNFRA DS</td>
<td>QISEKNFRA DS</td>
<td>QISEKNFRA DS</td>
<td>QISEKNFRA DS</td>
</tr>
</tbody>
</table>
Example 4-3 shows an extract from an OMEGAMON PE statistics report that indicates the EDM pool allocations and their storage location. The level of detail in this report allows you to monitor each pool separately.

### Example 4-3 Statistics report sample

<table>
<thead>
<tr>
<th>EDM POOL</th>
<th>QUANTITY</th>
<th>/SECOND</th>
<th>/THREAD</th>
<th>/COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGES IN RDS POOL (BELOW)</td>
<td>37500.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HELD BY CT</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HELD BY PT</td>
<td>4602.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FREE PAGES</td>
<td>32898.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FAILS DUE TO POOL FULL</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PAGES IN RDS POOL (ABOVE)</td>
<td>524.3K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HELD BY CT</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HELD BY PT</td>
<td>3504.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FREE PAGES</td>
<td>520.8K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FAILS DUE TO RDS POOL FULL</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PAGES IN DBD POOL (ABOVE)</td>
<td>262.1K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HELD BY DBD</td>
<td>67.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FREE PAGES</td>
<td>262.1K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FAILS DUE TO DBD POOL FULL</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PAGES IN STMT POOL (ABOVE)</td>
<td>262.1K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HELD BY STATEMENTS</td>
<td>5.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FREE PAGES</td>
<td>262.1K</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FAILS DUE TO STMT POOL FULL</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PAGES IN SKEL POOL (ABOVE)</td>
<td>25600.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HELD BY SKCT</td>
<td>2.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>HELD BY SKPT</td>
<td>322.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FREE PAGES</td>
<td>25276.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>FAILS DUE TO SKEL POOL FULL</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>

Dumps are another useful tool to monitor storage usage within the DBM1 address space. However, they capture only a point-in-time picture of storage usage and do not capture peaks.

A system dump of the DB2 address spaces can be formatted with the IPCS VERBEXIT command by using the following parameters:

```
VERBEXIT DSNWDMP 'ALL SM=3'
```

The output from the DSNWDMP produces a useful summary of DBM1 storage usage.

### 4.5.7 Conclusion

Most DB2 V9 users see VSCR below the bar in the DBM1 address space. However the amount of storage relief that you get depends on your workload mix.

### 4.5.8 Recommendations

When taking advantage of the virtual storage constraint relief functionality, we recommend that you follow these considerations:
Use packages where possible. By using multiple packages, you can increase the effectiveness of EDM pool storage management by having smaller objects in the pool.

Use RELEASE(COMMIT) when appropriate. Using the bind option RELEASE(COMMIT) for infrequently used packages and plans can cause objects to be removed from the EDM pool sooner.

Understand the impact of using DEGREE(ANY). A plan or package that is bound with DEGREE(ANY) can require 50% to 70% more storage for the CTs and PTs in the EDM pool than one bound with DEGREE(1). If you change a plan or package to DEGREE(ANY), check the change in the column AVGSIZE in SYSPLAN or SYSPACKAGE to determine the increase that is required.

Monitor your virtual storage usage to ensure that you do not over commit your virtual storage resources below the bar. Use the instrumentation data that DB2 V9 provides to monitor long-term trend analysis and short-term storage snapshots.

Size the EDM pool values to ensure that the EDM pool size for below-the-bar storage caters for the peak usage of this storage pool.

### 4.6 Real storage

If inadequate real storage is available on the processor, then workloads start to page. If DB2 V9 is one of the workloads that experiences paging, it is a significant performance impact to the DB2 workload.

You need to make sure that the buffer pools are 100% backed by real storage. DB2 uses an LRU or first in, first out (FIFO) process to reclaim pages in the buffer pools. If there is insufficient real storage, the storage page that contains the buffer for the oldest page could be paged out to disk. When DB2 needs to reclaim this page for reuse, then it has to be brought back into storage. The paging activity caused by the reclaim can have a significant impact on DB2 throughput. Because expanded storage is not exploited by z/OS in the z/Architecture mode, all paging activity goes directly to DASD. It is wise to prevent excessive paging to auxiliary storage. Use Resource Measurement Facility™ (RMF™) Monitor II and III to monitor the real storage frames and auxiliary frames that are used by the DB2 address spaces. It is important to keep in mind that your DB2 subsystem should not page.

Above-the-bar EDM pool storage is not backed by real storage until it is actually used. When storage is first allocated above the bar, it remains in a “first reference” state and is not backed by real storage. When this storage is accessed, or changed, it has to be backed by real or auxiliary storage for the duration that it is allocated.

The above-the-bar DBD, SQL statement cache, and skeleton pools are likely to reach the maximum space size that is specified for the pools, as the creation of cached objects fill these pools over time. You can expect to see real storage usage growth over time as the above-the-bar pools fill until they reach the point where they are fully used.

The DBD pool, particularly for data sharing where cross invalidation can fill the pool with unused DBDs, and the statement cache pool are likely to become fully used on large active systems. The skeleton pool also may show this behavior, but it is less likely as storage becomes freed for reuse.

The above-the-bar CT/PT pool is likely to have only peak allocation backed by real storage (and remain backed) as storage is reused from the low address end of the pool. The CT/PT pool contains only active in-use thread storage at any time. Storage pages are released due to completing threads and are reused first rather than other free pages in a “first reference” state. This reduces the impact of real storage usage by the above-the-bar pools.
Measurements have shown that, on average, real storage usage in DB2 V9 has increased by 10% or less. The pattern of storage usage has changed with most of the variation shown as a decrease in real storage usage in the DIST address space and a corresponding increase in the DBM1 address space. This is due to the move of the DDF storage to 64-bit in the z/OS shared virtual object. The shared virtual objects storage is accounted for by storage statistics by assigning this storage ownership to the DBM1 address space. This means that the reduction in DDF real storage usage is almost balanced by the increase in DBM1 real storage usage.

Table 4-4 shows a real storage usage comparison between a DB2 V8 system and a DB2 V9 system running the same workload. These DB2 V8 and DB2 V9 test results were taken from non-data sharing DB2 subsystems that were running with 360 active threads, driven by an ERP distributed workload. The DB2 V8 and DB2 V9 subsystems were running in new-function mode. The measurements were taken on the same hardware and operating system platform, which was a System z9 processor running z/OS 1.7.

Table 4-4  V8 versus V9 real storage comparison under workload

<table>
<thead>
<tr>
<th>Real storage frames from RMF x1000</th>
<th>V8</th>
<th>V9</th>
<th>Percent delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBM1</td>
<td>497</td>
<td>564</td>
<td>+13</td>
</tr>
<tr>
<td>DDF</td>
<td>171</td>
<td>115</td>
<td>-33</td>
</tr>
<tr>
<td>MSTR</td>
<td>17</td>
<td>16</td>
<td>-1</td>
</tr>
<tr>
<td>IRLM</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>689</td>
<td>699</td>
<td>+1</td>
</tr>
</tbody>
</table>

The RMF report values show the real storage increase in the DBM1 address space due to the storage accounting for the z/OS shared memory object that is being allocated to DBM1. The decrease in the DDF real storage usage is due to the move of the TCP/IP communication buffers and associated control blocks to the z/OS shared memory object. The overall real storage increase for this workload was 1%.

The same workload that was measured for DIST address space virtual storage usage, in 4.5, “Virtual storage constraint relief” on page 94, was used for a real storage comparison at the same time. The total real storage usage of all the DB2 V9 address spaces was measured while this workload was at its peak. The data is plotted in Figure 4-9. The RMF LPAR central storage data was used for the comparison. The results show that the real storage increase between DB2 V8 and DB2 V9 was 3% for one test and 4% for the other test.
Chapter 4. DB2 subsystem performance

4.6.1 Conclusion

Real storage usage in DB2 V9 has grown by up to 10% in measured workloads. The VSCR introduced in DB2 V9, by moving more of the DB2 storage to be backed above the bar, changes the pattern of real storage acquisition because above the bar is initially in "first referenced" state and not backed by real or virtual storage. When the first referenced storage is accessed, real storage is needed. We have stressed several times that adequate real storage is important in DB2 performance.

4.6.2 Recommendations

In regard to real storage, we make the following recommendations:

- Monitor your systems and DB2 V9 usage of real storage by using the tools that are available for this purpose. You can use the DB2 instrumentation data in IFCID 225 and IFCID 217 to see the sizes that the DB2 virtual storage pools are reaching and use this data to see if your estimated real storage requirements are sufficient.
- Wherever possible, ensure that your systems have enough real storage to ensure that the DB2 subsystem does not page.
4.7 Distributed 64-bit DDF

The DB2 DIST address space supports network communications with other remote systems and execution of database access requests on behalf of remote users.

In DB2 V8, DDF ran in 31-bit mode, and all virtual storage accessed by DDF was below the bar. The DDF address space private storage was used for communication buffers and control blocks for the active and inactive threads. This meant that approximately 1.5 GB of address space storage was available for DDF. Several of the DB2 V8 DDF control blocks used an extended common storage area (ECSA), which could cause constraints in this shared system resource. For some customers, the DDF address space was becoming a storage bottleneck, particularly where large numbers of sections were being prepared for each thread.

In DB2 V9, almost all of DDF runs in 64-bit mode and now uses above-the-bar storage in the z/OS Shared Memory Facility. See Figure 4-10 for an example of the virtual storage layout. DDF has moved almost all of the control blocks from ECSA into shared memory, freeing this system’s resource for other users.

Shared memory is a relatively new type of virtual storage that allows multiple address spaces to easily address common storage that is introduced in z/OS 1.5. It is similar to ECSA, since it is always addressable and no address register mode or cross memory moves are needed. It is different from ECSA since it is not available to all address spaces on the system. Only those that are registered with z/OS as being able to share this storage have visibility to this storage. Shared memory resides above the 2 GB bar.

Although 64-bit DDF is a performance enhancement for distributed server processing, it can also provide VCSR for the DIST address space. This VSCR is helpful in scaling large distributed workloads that were previously constrained by 31-bit mode. The shared memory object is created at DB2 startup, and all DB2 address spaces for the subsystem (DIST, DBM1, MSTR, and Utilities) are registered to be able to access the shared memory object.
The shared memory size is defined by the use of the HVSHARE parameter of the IEASYSxx member in the parmlib concatenation. The size that you specify, or let default, for the HVSHARE value determines where the system places the virtual shared area. If you specify a value less than 2 TB for the size, the system obtains storage that straddles the 4 TB line; half of the storage comes from above the 4 TB line, and half of the storage comes from below the 4 TB line. If you specify a value larger than 2 TB for the size, the system obtains storage starting at the 2 TB line. The value that you specify is rounded up to a 64-GB boundary. For more details about how to specify this parameter, see z/OS MVS Initialization and Tuning Reference, SA22-7592.

You can issue the following z/OS command to see the current defined storage and how much of it is currently allocated:

```
DISPLAY VIRTSTOR,HVSHARE
```

Example 4-4 shows the output of this command.

Example 4-4  z/OS DISPLAY VIRTSTOR,HVSHARE output

```
D VIRTSTOR,HVSHARE
IAR019I  17.51.48 DISPLAY VIRTSTOR 075
SOURCE = DEFAULT
TOTAL SHARED = 522240G
SHARED RANGE = 2048G-524288G
SHARED ALLOCATED = 262145M
```

Important: The system must be configured with sufficient shared private to allow every DB2 subsystem to obtain 128 GB of shared memory object at startup time.

The shared virtual memory is not backed at startup time; it is in a "first referenced" state and is not backed until it is referenced. The shared virtual memory is not charged against the MEMLIMIT of the DBM1 address space.

TCP/IP supports asynchronous receive socket operations using common storage buffers to improve performance. Applications that want to take advantage of 64-bit above-the-bar storage for backing data received on a socket are unable to take advantage of the performance gain of using common storage on an asynchronous receive. To address this limitation, TCP/IP has been updated to support use of 64-bit shared memory objects as the common storage buffer for asynchronous receives. DB2 V9 uses this TCP/IP support to move its communications storage usage to above the bar.

This above-the-bar shared memory usage has eliminated most of the data movement and data reformatting between the DBM1 and DIST address spaces. This means that we have reduced the number of CPU cycles that are needed to move data in and out of the system. Also by using only one shared copy of the data, we have reduced total storage usage.

These storage improvements allow more threads to be active at the same time, reduce CPU usage for distributed server requests, and reduce ECSA usage by DB2. These DDF 64-bit improvements are available in all DB2 V9 modes.

The most improvement in performance is seen in the Application Server (AS) function.

To ensure your system can support DB2 V9 DDF and TCP/IP using 64-bit shared memory, use the maintenance information that is supplied in APAR II14203, which details the necessary prerequisite maintenance.
4.7.1 z/OS shared memory usage measurement

New fields have been added to IFCID217 and IFCID225 that can be used to report on the shared memory usage. To collect this instrumentation data, ensure that a statistics class 6 trace is activated to get IFCID225 data and individually specify the IFCID217 trace via the -START TRACE(GLOBAL) IFCID(217) command. When activated, IFCID217 is recorded at the DB2 Statistics interval.

- IFCID217 storage detail record for the DBM1 address space is changed to add:
  QW0217SG TOTAL VIRTUAL SHARED ALLOCATED VSAS ABOVE THE BAR

- IFCID225 storage summary record for the DBM1 address space is changed to add:
  QW0225SF TOTAL FIXED VIRTUAL 64BIT SHARED
  QW0225SG TOTAL GETMAINED VIRTUAL 64BIT SHARED
  QW0225SV TOTAL VARIABLE VIRTUAL 64BIT SHARED

These four instrumentation fields can be reported with an appropriate tool that processes the IFCIDs. We have run an OMEGAMON XE Performance Expert report that shows these fields. Example 4-5 shows the relevant section of the output. Four fields from IFCID217 and IFCID225 are the last four lines shown in the report extract.

Example 4-5  Virtual storage layout above the bar

<table>
<thead>
<tr>
<th>DBM1 STORAGE ABOVE 2 GB</th>
<th>QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED STORAGE</td>
<td>(MB) 4.46</td>
</tr>
<tr>
<td>GETMAINED STORAGE</td>
<td>(MB) 4898.21</td>
</tr>
<tr>
<td>COMPRESSION DICTIONARY</td>
<td>(MB) 0.00</td>
</tr>
<tr>
<td>IN USE EDM DBD POOL</td>
<td>(MB) 0.26</td>
</tr>
<tr>
<td>IN USE EDM STATEMENT POOL</td>
<td>(MB) 0.02</td>
</tr>
<tr>
<td>IN USE EDM RDS POOL</td>
<td>(MB) 13.69</td>
</tr>
<tr>
<td>IN USE EDM SKELETON POOL</td>
<td>(MB) 1.27</td>
</tr>
<tr>
<td>VIRTUAL BUFFER POOLS</td>
<td>(MB) 421.87</td>
</tr>
<tr>
<td>VIRTUAL POOL CONTROL BLOCKS</td>
<td>(MB) 0.28</td>
</tr>
<tr>
<td>CASTOUT BUFFERS</td>
<td>(MB) 0.00</td>
</tr>
<tr>
<td>VARIABLE STORAGE</td>
<td>(MB) 626.15</td>
</tr>
<tr>
<td>THREAD COPIES OF CACHED SQL STMTS</td>
<td>(MB) 1.09</td>
</tr>
<tr>
<td>IN USE STORAGE</td>
<td>(MB) 0.00</td>
</tr>
<tr>
<td>HWM FOR ALLOCATED STATEMENTS</td>
<td>(MB) 0.00</td>
</tr>
<tr>
<td>SHARED MEMORY STORAGE</td>
<td>(MB) 622.97</td>
</tr>
<tr>
<td>TOTAL FIXED VIRTUAL 64BIT SHARED</td>
<td>(MB) 23.82</td>
</tr>
<tr>
<td>TOTAL GETMAINED VIRTUAL 64BIT SHARED</td>
<td>(MB) 1.11</td>
</tr>
<tr>
<td>TOTAL VARIABLE VIRTUAL 64BIT SHARED</td>
<td>(MB) 598.04</td>
</tr>
</tbody>
</table>

We have provided a sample OMEGAMON XE Performance Expert statistics report in the long format, which is shown in Appendix B, “Statistics report” on page 305. The sample shown in Example 4-5 was extracted from that report.

4.7.2 Conclusion

The 64-bit DDF processing using the z/OS Shared Memory Facility can impact the virtual storage usage of your systems. z/OS shared memory usage should be monitored at the initial migration to DB2 V9 and at any time where the DDF processing significantly changes to ensure that you have enough virtual storage allocated.
4.7.3 Recommendation

We make the following recommendations:

- Verify that the parmlib IEASYSxx member is set correctly to ensure your DB2 V9 subsystems can allocate the shared memory object.
- Monitor your virtual and real storage usage to make sure the movement of the DDF function to 64-bit processing, using the z/OS shared memory object, can be achieved without causing workload degradation.
- Monitor your ECSA usage with an appropriate tool, such as RMF. Reduce the allocated size if you can safely recover storage from this common shared system area after a successful migration to DB2 V9.

4.8 Distributed address space virtual storage

A distributed workload was measured on a DB2 V9 system. We compared the virtual storage utilization to the same workload that was run against a DB2 V8 subsystem. The workload consisted of 500 distributed active connections that were held open before committing. Each held a thread, which enabled a storage usage snapshot to be taken with all the connections and threads open. The testing workload was driven by a DB2 connect client that used CLI Open Database Connectivity (ODBC) for one test and an embedded SQL interface for the other test.

The virtual storage usage in the DIST address space for subpools 229/230 below the bar was measured with all connections and threads active and reported with RMF. Storage subpools 229/230 are where the control blocks and buffers reside for DDF. We used RMF for the virtual storage address space usage data for the DIST address space because the DB2 IFCID 225 storage statistics SMF record does not provide this information.

The graph in Figure 4-11 shows that the virtual storage usage in the DIST address space was reduced by 39% for both the CLI and embedded SQL interface.

![Figure 4-11  DB2 V8 and DB2 V9 DIST address space usage below-the-bar comparison](image)

The following legend explains the workloads shown in Figure 4-11:

- SQLCLI was an ASCII DB2 Connect Client using CLI.
- SQLEMB was an ASCII DB2 Connect Client using embedded SQL.
4.9 Distributed workload throughput

Distributed workload throughput has increased in DB2 V9. A test was done using the non-64-bit DDF interface; an equivalent test was done with the 64-bit DDF interface. Three of the workloads that were used in the Relational Warehouse Workload test were used in this test, and another insert intensive workload was added.

The workload used in 4.2, “CPU utilization in the client/server area” on page 84, was also measured to compare the CPU usage for non 64-bit and 64-bit processing with shared memory usage. We measured the class 1 CPU utilization for four types of workload and compared the results. The results shown in Figure 4-5 show that the 64-bit DDF interface and shared memory usage produced a decrease in CPU usage for all workloads that were tested.

<table>
<thead>
<tr>
<th>Distributed workload</th>
<th>Class 1 CPU reduction</th>
<th>Normalized throughput improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL CLI</td>
<td>-2.6%</td>
<td>+2.1%</td>
</tr>
<tr>
<td>SQL embedded</td>
<td>-5.2%</td>
<td>+2.3%</td>
</tr>
<tr>
<td>JDBC</td>
<td>-1.5%</td>
<td>+1.0%</td>
</tr>
<tr>
<td>Insert intensive</td>
<td>-6.3%</td>
<td>+5.1%</td>
</tr>
</tbody>
</table>

The data shows that there was a reduction in class 1 CPU time for all the workloads, which achieved a corresponding increase in the normalized ITR. The insert-intensive workload received the greatest benefit of the performance improvement. All distributed workloads benefited from the performance enhancements that were produced by changing the DIST address space to 64-bit and the use of the shared memory object for data transfer between DBM1 and DIST.

4.9.1 Conclusion

The implementation of 64-bit addressing and the use of the z/OS shared memory object by DDF has provided a reduction in CPU utilization for workloads that use this function. The 64-bit and shared memory facility has enabled DDF to reduce data moves between the DBM1 and DIST address spaces. The shared memory facility means that no cross-memory movement is needed to get execution requests into DBM1 and execution results out to DIST for sending to the remote clients. The reduction in data movement between address spaces has decreased the CPU usage for all tested workloads.

The non-64-bit DDF testing was done on a DB2 V9 system before any of the 64-bit or shared memory enhancements were introduced. The DB2 V8 and DB2 V9 DDF performance was equivalent at this time. With the results showing a decrease in DDF CPU utilization with the 64-bit enhancements, we would expect to see a comparable decrease in CPU utilization between DB2 V8 and DB2 V9.

4.10 WLM assisted buffer pool management

z/OS 1.8 delivers new WLM services that can assist DB2 in making dynamic buffer pool size adjustments based on real-time workload monitoring. In conversion mode, DB2 V9 exploits these new services to allow for dynamic buffer pool size adjustments so that the system’s memory resources can be more effectively used to achieve workload performance goals. This functionality should lead to better usage of existing memory resources for important work and
improve throughput of that work. For example, a buffer pool on a non-critical DB2 subsystem can be shrunk to reassign its storage to a buffer pool on an important DB2 subsystem on the same LPAR if important transactions are not meeting their performance goals.

You can enable or disable this functionality via a new AUTOSIZE(YES/NO) option of the ALTER BUFFERPOOL command at the individual buffer pool level. By default, automatic buffer pool adjustment is turned off. Only the size attribute of the buffer pool is changed.

Automatic management of buffer pool storage entails the following actions:

- DB2 registers the BPOOL with WLM.
- DB2 provides sizing information to WLM.
- DB2 communicates to WLM each time allied agents encounter delays due to read I/O.
- DB2 periodically reports BPOOL size and random read hit ratios to WLM.

DB2 notifies WLM each time that an allied agent encounters a delay caused by a random getpage that has to wait for read I/O. Periodically, DB2 reports to WLM the buffer pool sizes and hit ratio for random reads. WLM maintains a histogram. It plots the size and hit ratio over time and projects the effects of changing the size of the buffer pool. It determines the best course of action to take if the work is not achieving its goals. If WLM determines that buffer pool I/O is the predominant delay, it determines whether increasing the buffer pool size can help achieve the performance goal.

Depending on the amount of storage available, WLM may instruct DB2 to increase the buffer pool or first decrease another buffer pool and then increase the buffer pool in question. If a buffer pool is adjusted, the results will be just as though an ALTER BUFFERPOOL VPSIZE command had been issued. DB2 V9 restricts the total adjustment to +/- 25% the size of the buffer pool at DB2 startup. However, be aware that, for example, if a buffer pool size is changed and later DB2 is shut down and subsequently brought up, the last used buffer pool size is remembered across DB2 restarts; you can potentially change that size by another +/- 25% of the new value.

Keep in mind that there are implications to using this feature. Since DB2 V8, buffer pools have been allocated above the 2 GB bar. For each DB2 system, you can define up to a total of 1 TB of virtual space for your buffer pools. If you subscribe to the DB2 V8 recommendation to page fix your I/O bound buffer pools with low buffer pool hit ratios in order to save the CPU overhead of continually page fixing and freeing those pages, it is now possible that you may add up to 25% more demand on real storage to back those buffer pools.

For example, if you have 800 MB of buffer pools defined and they are page fixed, if they grow 25%, you would need an additional 200 MB of real storage to back them. If you do not have the extra capacity or are already paging to auxiliary storage, you could severely impact the operation of your system. We recommend that you closely monitor your real storage consumption when turning on WLM assisted buffer pool management for buffer pools that are defined with PGFIX(YES).

You can find additional information about WLM assisted buffer pool management in DB2 9 for z/OS Technical Overview, SG24-7330.

Note: DB2 offers some real storage protection for other workloads on the same LPAR from being “squeezed” by page fixing of buffer pools. DB2 will page fix buffer pools up to 80% of the real storage of the z/OS LPAR. However, this is by a DB2 subsystem. If you have more than one DB2 on that LPAR, you can potentially page fix 100% of the real storage.

This function is still under investigation (see z/OS APAR OA18461 recently closed with PTF UA48912 and DB2 PK75626 still OPEN) and measurements are under way. We plan to
update this section to reflect the measurements, when available, with conclusions and recommendations.

4.11 Automatic identification of latch contention and DBM1 below-the-bar virtual storage

DB2 V9 adds a monitor for automated health checking with the intent to improve the reliability, availability, and serviceability (RAS) of your DB2 subsystems.

Two issues that have sometimes impacted DB2 in the past have been processor stalls, which cause latch contention, and DBM1 below-the-bar storage shortages. These issues were identifiable by the following processes:

- DB2 V7 introduced a serviceability command, DISPLAY THREAD(*) SERVICE(WAIT), to help identify CPU stalls.
- The IFCID 225 records identify DBM1 storage constraint issues.

However, these processes were manual, and DB2 did not provide an automated means to identify them.

With DB2 9 in conversion mode, a built-in monitor runs from restart to shutdown and checks the health of the system on one-minute intervals. The built-in monitor identifies CPU stalls (for system, DBAT, and allied agents) that result in latch contention. The monitor attempts to clear the latch contention by a temporary priority boost via WLM services to the latch holder. This should allow customers to run closer to 100% CPU utilization by reducing the chances that less important work can hold a latch for an extended period of time, causing important work to stall.

In addition, DBM1 storage below the 2 GB bar is monitored for critical storage increases, and messages are sent when thresholds are reached.

You can view the health of your system by issuing the following command (see Figure 4-12):

```
DISPLAY THREAD(*) TYPE(SYSTEM)
```

![Figure 4-12 DISPLAY THREAD(*) TYPE(SYSTEM) output](image)

When DBM1 storage below the 2 GB bar reaches a threshold of 88, 92, 96, or 98 percent of the available storage, messages (DSNV508I, DSNV510I, DSNV511I, and DSNV512I) are issued reporting the current DBM1 storage consumption and the agents that consume the most storage. See Figure 4-13 for a sample of the DSNV508I, DSNV510I, and DSNV512I messages.
You can easily write automation based on these messages and take proactive actions to prevent the problem from becoming serious.

Furthermore, the DISPLAY THREAD command is extended to include STORAGE as an option. You can now issue the following command (see Figure 4-14):

```
DISPLAY THREAD(*) SERVICE(STORAGE)
```

The health monitor feature should correct some problems, help provide early warning about other problems, and at least provide additional diagnostic information. This automated monitor should help lower the cost of ownership for DB2 and help ensure that customers maintain healthy DB2 systems.

**Recommendation:** In order to properly detect these CPU stalls, we recommend that you run the started task for DB2 MSTR in the SYSSTC dispatching priority.
To avoid failover, DB2 attaches a monitor task for each system address space in the following order:

1. MSTR
2. DBM1
3. DIST, if present

The tasks are active for intervals and check for agents that are stalled on latches and conditions of below-the-bar cache.

### 4.11.1 Verification

The distributed DRDA SQL CLI Relational Warehouse Workload was measured and the new DISPLAY THREAD monitoring information was collected. The workload was not impacted by any measurable overhead. The display results are shown in the following examples.

The DISPLAY THREAD(*) TYPE(SYSTEM) shows the system agent threads that are deemed useful for serviceability purposes. Example 4-6 shows the monitoring threads and the situation that is related to processors latching contention. During this workload run, no boost of allied threads holding latches was required.

#### Example 4-6  DIS THREAD(*) TYPE(SYSTEM)

```
-D91B DIS THREAD(*) TYPE(SYSTEM)
DSNV401I -D91B DISPLAY THREAD REPORT FOLLOWS -
DSNV497I -D91B SYSTEM THREADS - 867
DB2 ACTIVE
NAME     ST A   REQ ID           AUTHID   PLAN     ASID TOKEN
D91B     N *  0 002.VMON  03 SYSOPR            006F     0
V507-INACT MONITOR, INTERVALS=3, STG=N/A, BOOSTS=N/A, HEALTH=N/A
D91B     N *  0 028.ERMON01 SYSOPR            006F     0
D91B     N *  0 028.RESYNT01 SYSOPR            006F     0
D91B     N *  0 027.PNACBV01 SYSOPR            006F     0
V490-SUSPENDED 07136-15:22:18.35 DSNLQCTL +00000A0E UK22985
D91B     N *  0 027.DDFINT03 SYSOPR            006F     0
V490-SUSPENDED 07136-15:22:18.35 DSNLQCTL +00000A0E UK22985
D91B     N *  0 027.DDFINT04 SYSOPR            006F     0
V490-SUSPENDED 07136-15:22:18.35 DSNLQCTL +00000A0E UK22985
D91B     N *  0 027.SLSTNR02 SYSOPR            006F     0
D91B     N *  0 027.RLSTNR02 SYSOPR            006F     0
D91B     N *  0 027.GORST02 SYSOPR            006F     0
D91B     N *  0 028.RESYNC01 SYSOPR            006F     0
V490-SUSPENDED 07136-15:22:18.44 DSNLQCTL +00000A0E UK22985
D91B     N *  0 010.PMICMS01 SYSOPR            006E     0
V490-SUSPENDED 07136-15:30:03.55 DSNMICMS +000005C0 UK15726
D91B     N *  0 010.PMITMR02 SYSOPR            006E     0
D91B     N *  0 022.SPQMON01 SYSOPR            006E     0
D91B     N *  0 014.RTSTST00 SYSOPR            006E     0
V490-SUSPENDED 07136-15:29:18.28 DSNBITMR +00000B78 09.24
D91B     N *  0 010.PM2PCP01 SYSOPR            006E     0
V490-SUSPENDED 07136-15:29:30.72 DSNBITMR +00000B78 09.24
D91B     N *  0 010.PM2PCK02 SYSOPR            006E     0
V490-SUSPENDED 07136-15:29:48.28 DSNBITMR +00000B78 09.24
D91B     N *  0 002.VMON  02 SYSOPR            006E     0
V507-INACT MONITOR, INTERVALS=4, STG=N/A, BOOSTS=N/A, HEALTH=N/A
D91B     N *  0 023.GSCSNM03 SYSOPR            006C     0
D91B     N *  0 004.JW007  01 SYSOPR            006C     0
V490-SUSPENDED 07136-15:30:03.89 DSNJW107 +0000029E UK22636
D91B     N *  0 004.JTIMR  00 SYSOPR            006C     0
```
Example 4-7 shows no threads waiting.

Example 4-7  DIS THREAD(*) SERVICE(WAIT)

-D91B DIS THREAD(*) SERVICE(WAIT)
DSN401I -D91B DISPLAY THREAD REPORT FOLLOWS -
DSN419I -D91B NO CONNECTIONS FOUND
DSN9022I -D91B DSNVDT ‘-DIS THREAD’ NORMAL COMPLETION

Example 4-8 shows the SERVICE(STORAGE) output, related to virtual storage usage. In this case, the distributed threads are shown under V492 with the amount of storage that they use in DBM1 agent local pools as collected by DB2 in the cursor table. The sum of LONG and VLONG is what it is used below the bar in 31-bit private. After 64 is the amount above the bar in 64-bit private.

Example 4-8  DIS THREAD(*) SERVICE(STORAGE)

-D91B DIS THREAD(*) SERVICE(STORAGE)
DSN401I -D91B DISPLAY THREAD REPORT FOLLOWS -
DSN402I -D91B DSNVDT ‘-DIS THREAD’ NORMAL COMPLETION
4.12 Latch class contention relief

A latch is a DB2 mechanism for controlling concurrent events or the use of system resources. Latches are conceptually similar to locks in that they control serialization. They can improve concurrency because they are usually held for a shorter duration than locks and they cannot "deadlatch". However, latches can wait, and this wait time is reported in accounting trace class 3 data.

Latches are not under user control, and they are generally not described in much detail. However a few of the latch classes are associated with potential performance problems, and their function and analysis are documented.

DB2 V9 has addressed a number of performance issues with the high usage of certain latches and the underlying problem that was causing them. DB2 reports lock and latch
suspension times in IFCID 0056 and IFCID 0057 pairs. This can be reported on by
OMEGAMON in its accounting trace. The report extract in Example 4-9 shows the
accumulated lock and latch suspensions that are reported as class 3 suspension time.

Example 4-9  Class 3 suspension report

<table>
<thead>
<tr>
<th>CLASS 3 SUSPENSIONS</th>
<th>AVERAGE TIME</th>
<th>AV.EVENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCK/LATCH(DB2+IRLM)</td>
<td>0.057264</td>
<td>0.70</td>
</tr>
<tr>
<td>SYNCHRON. I/O</td>
<td>0.003738</td>
<td>3.88</td>
</tr>
<tr>
<td>DATABASE I/O</td>
<td>0.002905</td>
<td>3.38</td>
</tr>
<tr>
<td>LOG WRITE I/O</td>
<td>0.000833</td>
<td>0.50</td>
</tr>
<tr>
<td>OTHER READ I/O</td>
<td>0.000270</td>
<td>0.15</td>
</tr>
<tr>
<td>OTHER WRITE I/O</td>
<td>0.000020</td>
<td>0.00</td>
</tr>
<tr>
<td>SER.TASK SWTCH</td>
<td>0.000221</td>
<td>0.03</td>
</tr>
<tr>
<td>UPDATE COMMIT</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>OPEN/CLOSE</td>
<td>0.000154</td>
<td>0.01</td>
</tr>
<tr>
<td>SYSLGRRNG REC</td>
<td>0.000010</td>
<td>0.01</td>
</tr>
<tr>
<td>EXT/DEL/DEF</td>
<td>0.000046</td>
<td>0.00</td>
</tr>
<tr>
<td>OTHER SERVICE</td>
<td>0.000010</td>
<td>0.01</td>
</tr>
<tr>
<td>ARC.LOG(QUIES)</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>LOG READ</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>DRAIN LOCK</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>CLAIM RELEASE</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>PAGE LATCH</td>
<td>0.000006</td>
<td>0.00</td>
</tr>
<tr>
<td>NOTIFY MSGS</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>GLOBAL CONTENTION</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>COMMIT PH1 WRITE I/O</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>ASYNCH CF REQUESTS</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>TCP/IP LOB</td>
<td>0.000000</td>
<td>0.00</td>
</tr>
<tr>
<td>TOTAL CLASS 3</td>
<td>0.061519</td>
<td>4.76</td>
</tr>
</tbody>
</table>

If the LOCK/LATCH(DB2+IRLM) times become a significant part of the total suspension time,
then analysis of the lock or latch causes should be undertaken.

The latch classes that are used are shown in part of the OMEGAMON XE Performance
Expert output, which reports on the number of latches that DB2 took during the reporting
period (Example 4-10).

Example 4-10  Latch classes report

<table>
<thead>
<tr>
<th>LATCH CNT</th>
<th>/SECOND</th>
<th>/SECOND</th>
<th>/SECOND</th>
<th>/SECOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC01-LC04</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LC05-LC08</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LC09-LC12</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LC13-LC16</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LC17-LC20</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LC21-LC24</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LC25-LC28</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>LC29-LC32</td>
<td>0.00</td>
<td>0.00</td>
<td>0.08</td>
<td>0.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOCKING ACTIVITY</th>
<th>QUANTITY</th>
<th>/SECOND</th>
<th>/THREAD</th>
<th>/COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUSPENSIONS (ALL)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>SUSPENSIONS (LOCK ONLY)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>
DB2 V9 has addressed a number of latch class suspensions that could cause latch suspension times to be a contributor to the class 3 suspension time.

4.12.1 Latch class 6

DB2 latch class 6 is held for an index buffer split in a data sharing group. Index page latches are acquired to serialize changes within a page and check that the page is physically consistent. Acquiring page latches ensures that transactions that access the same index page concurrently do not see the page in a partially changed state. Latch class 6 contention can be a problem in the data sharing group with heavy insert activity across the members of the group because the latch taken for the index manager is a physical lock (P-lock) latch.

In the data sharing environment, each index split causes two forced log writes. The frequency of index splitting can be determined from LEAFNEAR, LEAFFAR, and NLEAF in SYSINDEXES and SYSINDEXPART catalog tables. The index split serialization is reported as latch class 6 contention in statistics collection.

In DB2 V9 new-function mode, the insert pattern in an index is monitored and analyzed. Based on the detected insert pattern, DB2 V9 can split an index page by choosing from several algorithms. If an ever-increasing or ever-decreasing sequential insert pattern is detected for an index, DB2 splits the index pages asymmetrically by using an approximately 90/10 split. See “Asymmetric index page split” on page 154 for more details about this performance improvement in reducing index contention.

Creating an index with the larger index page sizes that are now available also reduces the number of page splits in the index. The larger index page size is beneficial in cases where a frequent index split results from a heavy insert workload. See “Larger index page size” on page 156 for more details about the performance improvements.

Using a randomized index key order also reduces latch class 6 contention by moving the insert activity to different index pages. We also measured the performance improvement gained by using the randomized index key; see “Index key randomization” on page 156 for the details.

4.12.2 Latch class 19

The code that moves log records into the log output buffer takes the log-write latch in latch class 19. A high LOCK/LATCH wait time shown in an OMEGAMON report can occur as a result of high class 19 log latch contention.

When DB2 launched data sharing, it introduced the concept of a log record sequence number (LRSN) for the identification of log records. The LRSN is a value that is derived from the stored clock time stamp and synchronized across the members of a data sharing group by the Sysplex Timer®. This mechanism is used in data sharing to provide the ordering of log records that have been updated by any member in the group, in alternative to the RBA used in a non-data sharing subsystem.

In data sharing, with versions prior to DB2 9, the LRSN value of each log record on an individual member has to be unique at the subsystem level. The LRSN increment process uses the store clock (STCK) instruction and the returned value increments every 16 microseconds. This means that, if two successive log records for the same DB2 data sharing
member have the same LRSN value, then DB2 re-drives the STCK instruction process for the LRSN until a different value is generated for the second log record. This 16 microsecond interval is spent spinning waiting for the STCK increment and the latch is held.

In DB2 9 new-function mode, the LRSN values only have to be unique within a given data or index page, not within a DB2 log record. This means that successive data changes that create logging information within a single data or index page must have unique LRSNs to order the changes as they arrive and are processed. DB2 9 in new-function mode only re-drives the LRSN increment, and accrues spin wait time, when successive changes that require logging into a given page have the same LRSN value.

This enhancement reduces the need to have uniqueness at the log record level. It also removes the need to hold the log latch while the LRSN is incremented. This should provide increased log and transaction throughput and reduced latch class 19 waits.

Initial testing had shown up to nearly two times improvement in logging rate through the reduction of how often DB2 performs the LRSN spin and the removal of the log latch. Other testing done in a data sharing group with the Relational Warehouse Workload showed a 10 time reduction in log latch contention.

However, further changes are now under study for the situation of inserting consecutive rows or multiple rows with rowset positioning. In these cases the inserts have the likelihood of happening within the same data and index pages and ending up with the same LRSN, especially with faster processors such as z10.

Latch class 19 reduction can also be achieved by using the new DB2 9 function of having a not logged table space. By removing logging from a table space, you remove any potential for log records that are associated with this tablespace to contribute to the logging load of the system. See 5.7, “Not logged table spaces” on page 174, for more details about the not logged option.

The reduction in latch class 19 waits can mean that there is potential for an increase in logging activity, which can cause contention for the log buffers due to the increased logging rate. Waits due to unavailable log buffers are captured in IFCID 001 and can be seen in an OMEGAMON long statistics report. The field name is UNAVAILABLE OUTPUT LOG BUFF. If you find that log buffer waits are occurring, you can use DSN6LOGP DSNZPARM LOGBUFF to increase the value.

### 4.12.3 Latch class 24

The EDM pool is another resource that is serialized against, and latch class 24 is used for these serialization functions. In a storage constrained system where EDM pool reuse is high, you can be affected by waiting on latch class 24. DSNZPARM EDMBFIT specifies how free space is to be used for large EDM pools that are greater than 40 MB. If you set DSNZPARM EDMBFIT=YES, the EDM better fit the algorithm that uses latch class 24 to serialize access to the EDM pool while it searches for the best fit.

To reduce latch class 24 contention for the EDM pool, we recommend that you always set DSNZPARM EDMBFIT=NO and increase the EDM pool size. In recent releases, DB2 has split the EDM pool into other functional pools to help reduce contention in these pools.

In DB2 V7, there was only one EDM pool, which had only one LRU Latch. In DB2 V8, the EDM pool was split into the DBD pool and the other EDM pool, which reduced the necessity to wait on latch class 24. In DB2 9, the other EDM pool was split three ways, which further reduces the waits in latch class 24.
We recommend that, in most cases, you set EDMBFIT=NO. It is especially important to set EDMBFIT=NO when high class 24 latch, the EDM LRU latch, contention exists. For example, be sure to set EDMBFIT=NO when class 24 latch contention exceeds 500 contentions per second. Waits due to latch class contentions can be seen in an OMEGAMON long statistics report.

See 4.5, “Virtual storage constraint relief” on page 94, for information about sizing the EDM pool at DB2 V9.

### 4.13 Accounting trace overhead

All accounting traces have some overhead over the base CPU requirement for running a workload. As more detail is needed in the accounting data, more detailed tracing is done, which leads to greater workload. To show the overhead of package accounting at various levels of detail, we ran two types of workload. The first one was typical of a user workload. The second one was an artificial test that was used to show the relative percentage overhead of the level of detail that is used in the tracing.

Both workloads used the measurement of class 1 and class 2 CPU time as the basis for comparison. We measured and then compared the CPU overhead of various multiple accounting trace classes between DB2 V8 and DB2 V9.

Class 1 elapsed time is always present in the accounting record and shows the duration of the accounting interval. It includes time spent in DB2 as well as time spent in the front end. In the accounting reports, it is referred to as application time.

Class 2 elapsed time, produced only if accounting class 2 is active, counts only the time spent in the DB2 address space during the accounting interval. It represents the sum of the times from any entry into DB2 until the corresponding exit from DB2. It is also referred to as the time spent in DB2.

### 4.13.1 Typical user workload

The first test was a single package batch application that contained 60 static queries and was run as a single threaded application. This workload was run as a comparative test for an expected user workload. The data graphed in Figure 4-15, which was obtained by running the workload and the accounting trace classes, is shown in the graph center with the details specified. The range of accounting classes measured varied from class 1, which was taken as the base measurement, to the highest level of detail with multiple concurrent active accounting classes, which consisted of 1, 2, 3, 7, 8, and 10.

The data shows the relative percent of CPU increase for each package accounting class for a 60-query application. This workload consisted of a moderately long running, single package workload with many statements.
4.13.2 Tracing relative percentage overhead

The second test was a multipackage, short running batch application that contained 100 packages. Each package was a single SQL statement. This application was run so that there were 500 interactions of the workload, which ensured that there was enough work in the DB2 subsystem to create a viable test workload.

The data in Figure 4-16 shows the relative percent CPU increase for each package accounting class. This artificial workload consisted of a short running, multiple package workload with a single short running statement. This test was run to show the extreme of running a short running application, with a large package overhead, with the maximum level of accounting detail data that is being collected.

Table 4-6  Tabulated data for relative CPU% increase for 100 package query application

<table>
<thead>
<tr>
<th>Accounting class</th>
<th>CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DB2 V8 CL1</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1, 2</td>
<td>2.87</td>
</tr>
<tr>
<td>1, 2, 3</td>
<td>3.18</td>
</tr>
<tr>
<td>1, 2, 3, 7, 8</td>
<td>6.96</td>
</tr>
<tr>
<td>1, 2, 3, 7, 8, 10</td>
<td>23.29</td>
</tr>
</tbody>
</table>

The tabulated data shown in Table 4-6 is also shown graphically in Figure 4-16.
The measurements show that accounting class 10, package detail, for a short running application can have a large overhead.

To help reduce the CPU overhead that is associated with the accounting trace, DB2 V9 has introduced filtering capabilities. These filtering capabilities allow you to target specific workloads. Prior to DB2 V9, when starting traces, you could qualify the trace by PLAN name, AUTHID, LOCATION, and IFCID, but you could not specify an exclude list. In DB2 V9, you can be more specific on the -START TRACE command by specifying additional include parameters as well as exclude parameters. This gives you the option to dramatically reduce what is being traced and the amount of trace records that are produced. The new filtering keywords that can be used in the -START TRACE for INCLUDE or EXCLUDE are:

- USERID: Client user ID
- WRKSTN: Client workstation name
- APPNAME: Client application name
- PKGLOC: Package LOCATION name
- PKGCOL: Package COLLECTION name
- PKGPROG: PACKAGE name
- CONNID: Connection ID
- CORRID: Correlation ID
- ROLE: End user’s database ROLE

4.13.3 Conclusion

DB2 V9 shows the equivalent or reduced CPU time for accounting trace classes versus DB2 V8. For a single package application that is doing moderate activity, the DB2 V9 overhead of adding classes 7, 8, and 10 is insignificant and is still reduced versus V8. For larger, multi-package application, doing a single or a few short running SQL per package, the overhead of V9 is much reduced versus DB2 V8, but is still significant for classes 7, 8, and 10.
V9 offers several new trace filters that allow you to reduce the overhead that is associated with tracing the complete class. These filters include the ability to include or exclude data with wild card capability. These new tracing capability options can help target more detailed trace classes selectively and reduce overall CPU consumption by DB2 accounting traces.

### 4.13.4 Recommendation

Use the filtering capabilities introduced in DB2 V9 to restrict what you trace if you know a specific workload that you want to target. See the section about the filtering that is available with the START TRACE command in the *DB2 Version 9.1 for z/OS Command Reference*, SC18-9844.

To monitor the basic load of the system, try continually running classes 1, 3, 4, and 6 of the DB2 statistics trace and classes 1 and 3 of the DB2 accounting trace. This monitoring should give you a good indication about the health of your DB2 subsystem without creating a large overhead. In the data you collect, look for statistics or counts that differ from past measurements that allow you to do trend analysis.

### 4.14 Reordered row format

Prior to DB2 9 for z/OS, each variable-length column in a data row was stored with its length preceding it in a two-byte field. A field that follows the variable-length column has a variable offset. This is now called basic row format (BRF). To access a column with a variable offset, DB2 has to scan the row to find the column of interest. This scanning process is started at the first variable-length column until DB2 reaches the column to process. If we update any variable-length column, DB2 logs the changed column and all following columns to the end of the row.

For these reasons, we used to recommend that you place all variable-length columns at the end of the row and place the most frequently changed columns at the end of the row. See Figure 4-17.

---

**Variable-length or compressed-row considerations**

- Variable-length columns at end of row to minimize retrieval cost
- Frequently updated columns at end of row to minimize log volume

| F1 | F2 | V3 | F4 | F5 | V6 |
---|---|---|---|---|---|

- Variable-length row update generally logs from the first changed byte to end of row
- Variable-length row update without length change logs from the first changed byte to the last changed column (V7)
- Fixed-length row update logs from the first changed column to the last changed column

*Figure 4-17 The basic row format*
Nowadays you often do not have the opportunity to control the placement of data columns to optimize your DB2 performance for the data you support. Examples of this are the dynamic ERP and customer relationship management (CRM) applications. To help alleviate the problem of non-optimal row formatting, DB2 has introduced a new format for data rows called reordered row format (RRF), which helps reduce the impact and overhead of processing variable-length columns.

In DB2 9 new-function mode, you do not have to worry about the order of columns within the row. You can specify the order however you want. DB2 automatically reorders the columns within the row to place all the variable-length columns at the end of the physical row within the data page. Instead of storing the columns with their length, we store all of the offsets for the variable-length columns in an area that follows the last fixed-length column and prior to the first variable-length column. DB2 can now directly access each variable-length column and know its length by doing simple calculations with the offsets.

By using this method, DB2 removes any non-optimal row formats and hence reduces row processing effort. The format in which the row is stored in the table is changed to optimize the column location for data retrieval and for predicate evaluation. See Figure 4-18.

To get your current data into the DB2 9 new-function mode reordered row format, you have to use the REORG or LOAD REPLACE utility. These utility functions automatically convert table spaces to the reordered row format. Prior to DB2 V9, table spaces were created in basic row format. The first use of these utilities in V9 NFM converts from basic row format to the new reordered row format. There is no change in any SQL syntax that is needed to access the data in the reordered row format. All table spaces that are created in V9 new-function mode are automatically in the reordered row format. The exception is that, if there is a table in the table space with EDITPROC or VALIDPROC, then the table space remains in the basic row format.

In terms of the amount of log data produced, the V9 reordered row format should result in roughly the same amount. However the exception is when a given table has undergone a significant tuning to reduce the amount of log data. Also Update, rather than Insert and Delete, is the predominant application.
Prior to the reordered row format implementation, if all the columns in a row were defined as variable-length columns, DB2 needed to do offset calculations to access each column. This process could consumes CPU. With the new reordered row format in DB2 V9, this offset calculation processing for variable-length columns is not needed because all the columns are directly addressable.

There is no impact in performance if you do not have any varying-length columns. There is no change for data access for tables that only contain fixed-length columns.

With APAR PK85881, the new DSNZPARM SPRMRRF can enable or disable RRF at subsystem level.

With the same APAR, LOAD REPLACE and REORG TABLESPACE utilities have a new parameter, ROWFORMAT, which allows you to choose the output format (RRF or BRF) of the referenced table space or partition(s). This parameter has no effect on LOB, catalog, directory, XML, and universal table space participating in a CLONE relationship.

APAR PK87348 has enabled the use of basic row format for universal table spaces.

The DB2 catalog and directory are not converted to the reordered row format and will remain in the basic row format when the REORG utility is run against them.

### 4.14.1 Reordered row format and compression

Using KEEPDICTIONARY when doing a REORG conversion from a basic row format to a reordered row format can produce poor compression results. In V9, KEEPDICTIONARY is ignored when converting from the basic row format to the reordered row format, unless it is specified differently in a new HONOR_KEEPDICTIONARY DSNZPARM in the DSNJUZ installation job.

**Attention:** In the case of many small variable length columns with the same length, the conversion to reordered row format may reduce the compression factor. In this specific case, you can convert the table space back to BRF with the LOAD or REORG utilities using the ROWFORMAT parameter or disable RRF at the subsystem level with the new DSNZPARM SPRMRRF.

We have generally recommended to use VARCHAR for lengths more than 18-20 bytes. This is still a valid recommendation.

### 4.14.2 Conclusion

The reordered row format is a DB2 9 new-function mode enhancement that is automatically implemented if the application table space qualifies.

### 4.14.3 Recommendation

Use the REORG and LOAD REPLACE utility functions to convert the basic row format to the reordered row format and experience the performance gains that the new reordered row format can provide to rows with several variable-length columns.

Use the DSNZPARM SPRMRRF to synchronize the activation of RRF in multiple DB2 subsystems where table spaces can be physically transported via DSN1COPY.
4.15 Buffer manager enhancements

Several changes in DB2 9 have improved the overall performance of the buffer manager component. For example, DB2 9 has increased the prefetch and deferred write quantities when dealing with large buffer pools. Large is verified by DB2 as follows:

- For sequential prefetch, if VSEQT*VPSIZE > 160 MB for SQL, 320 MB for utility
- For deferred write, if VPSIZE > 160 MB for SQL, 320 MB for utility

For more details about these functions, see 5.8, “Prefetch and preformatting enhancements” on page 176.

DB2 9 uses a larger preformatting quantity when formatting table spaces. This means that less time is spent switching to preformat the table space and less time is spent updating the ICF catalog for the highly used relative byte address (RBA) value. For more information, see 4.19, “Enhanced preformatting” on page 134.

DB2 9 has implemented a long-term storage page fix on the I/O work area that is used for compressed indexes and castout engine work areas. Measurements have shown that this long-term page fixing has resulted in a 3% reduction in DBM1 address space service request block (SRB) time.

DB2 9 has increased the number of open and closed service tasks from 20 to 40. This allows more parallelism in the open and closed processes.

DB2 9 has removed the increase in CPU time that is used by the hash anchor point processing of the buffer pool. DB2 V8 showed a non-linear increase in the CPU that is used under an SRB when processing the hash chains for large buffer pools, which are those buffer pools that are greater than 5 GB. In DB2 V9, the CPU used under the SRB for hash chain processing is now linear, which reduces the CPU impact of larger buffer pools.

We have reduced the amount of processing time that the declaim process uses. In one DB2 V8 measurement, the declaim process used up to 8% of the total CPU time. In DB2 V8, the declaim process sequentially scans all partitions that are involved in the declaim scope. For example in DB2 V8, if 1000 partitions were involved in a declaim scope, DB2 would scan all 1000 partitions even though it may not have claims on all those partitions. This meant that the transaction path length and CPU utilization for declaim could be large. In DB2 V9, this has been enhanced so that DB2 scans only the partitions that have been claimed.

During testing with 10 partitions and a claim on only three of the partitions, the DB2 V9 declaim path length was shown to be half that of DB2 V8. Therefore, we would expect a 50% CPU reduction for the declaim process. The benefits of the enhancement scale up as the number of partitions increases. The testing also showed that there was no degradation for the claim processing with the declaim enhancement.

The amount of DB2 stack storage that is taken by the buffer manager modules from the DBM1 below-the-bar storage has been reduced. V8 APAR PK21237 made the change where the buffer manager engines were modified to use a single common above-the-bar storage pool, rather than having a separate pool for each engine. For both the castout engines and P-lock engines, excess stack storage is now released before suspending, while awaiting for
more work to do. Additionally, the default maximum for the number of engines for castout or
defered writes was reduced back to 300 in V8 because it increased storage below the bar.
This engine number reduction to reduce storage utilization below the bar has been carried
forward to DB2 V9. The buffer manager trace has also been moved to be above the bar.
These changes contribute to the overall DBM1 VSCR that has been achieved in DB2 9.

4.16 WORKFILE and TEMP database merge

Currently DB2 supports two databases for temporary files and temporary tables: the
WORKFILE database and the TEMP database. The 'AS' clause of the CREATE DATABASE
statement with either the 'WORKFILE' or 'TEMP' subclause indicates that the database to be
created is either a WORKFILE database or a TEMP database. Each DB2 subsystem or data
sharing member has one WORKFILE database and may have one TEMP database.

The WORKFILE database is used by DB2 for storing created global temporary tables and as
storage for work files for processing SQL statements that require temporary working space,
for SORTs, materializing views, triggers, nested table expressions, and others.

The TEMP database is used by DB2 for storing the external (user-defined) declared global
temporary tables and the DB2 declared global temporary tables for static scrollable cursor
implementation.

In DB2 9, the two temporary databases, WORKFILE and TEMP, supported by DB2 are
converged into one database. This allows you to define, monitor, and maintain a single
WORKFILE database to be used as storage for all temporary files and tables. This
convergence preserves the external functionality of the temporary tables as it is today.

This merging of the functions into the WORKFILE database means the TEMP database is no
longer used by DB2, but we suggest that you do not DROP the TEMP database until you are
sure that you will not be falling back to DB2 V8. Otherwise, you will have to recreate the
TEMP database if you fallback to DB2 V8.

After you complete the migration from DB2 V8 and want to reclaim the storage associated
with an existing TEMP database, it is your responsibility to drop the TEMP database and
reclaim the DASD storage to be used for the WORKFILE database or for something else.

The TEMP database supported 4 KB, 8 KB, 16 KB, and 32 KB page sizes, but the
WORKFILE database supports only 4 KB or 32 KB page sizes. If you create table spaces in
the WORKFILE database, then you must specify a 4 KB and 32 KB page size.

The declared global temp table and scrollable cursor implementations now use the
WORKFILE database. To compensate for the removal of the TEMP database, which supports
the SEGSIZE clause for creation of a table space in it, the restriction that SEGSIZE cannot be
specified for creating a table space in the WORKFILE database is removed, in new-function
mode only.

In conversion mode, the SEGSIZE clause continues to be rejected in the CREATE
TABLESPACE statement for creating table spaces in the WORKFILE database. For table
spaces that existed in a WORKFILE database before migration from DB2 Version 8, for those
created during migration, and for those created in conversion mode of DB2 9 for z/OS, the
SEGSIZE column of the catalog table SYSTABLESPACE continues to show 0 as the segment
size. A value of 0 indicates that these table spaces were created prior to the enablement of
the DB2 V9 new-function mode. However, DB2 treats these table spaces as segmented, with
the default segment size of 16, both in conversion mode and new-function mode.
4.16.1 Workfile sizing

DB2 sorts have been optimized to select the best page size for the workfile table space. If the workfile record length is less than 100 bytes, then a 4 KB page size is used. In all other cases, a 32 KB page size is used. This can result in less wasted workfile space and faster I/O, but it can change the balance between the 4 KB and 32 KB workfile usage. If DB2 V9 selects a 32 KB work file, and one is not available, then DB2 uses a 4 KB work file. However, you will lose the performance benefit that the 32 KB page size would have given you. In 4.16.2, “Instrumentation for workfile sizing” on page 127, we show the fields that you can monitor to see if you are not getting optimal usage of the work files.

We recommend that you check the number and size of your current 32 KB work files. We expect that you will need to increase the number of 32 KB work files due to the DB2 V9 increase of using 32 KB work files. There will be a corresponding decrease in the amount of storage that is needed for 4 KB work files. We also recommend that you check the sizes for your 4 KB and 32 KB buffer pools that are used for the work files. We expect that you will need to increase the size of your 32 KB buffer pool and decrease the size of the 4 KB buffer pool.

We recommend that you set the DSNZPARM 'DSVCI' to YES. This enables DB2 to create the DB2 managed data sets with a VSAM control interval that matches the page size for the table spaces.

A new online updatable DSNZPARM, MAXTEMPS, has been added to DSN6SYSP. This DSNZPARM specifies the maximum amount of workfile storage that an agent can use and is specified in units of MBs or GBs. If the DSNZPARM value is specified as 0, then no limit is enforced, which was the default in previous releases.

When the total storage used by an agent exceeds the MAXTEMPS value that you specified, a Resource Unavailable message (DSNT501I) and a Resource Unavailable SQLCODE (-904) are issued; the activity that caused this condition terminates. The -904 message has a new reason code and resource name as shown in Example 4-11.

Example 4-11   New resource unavailable information

<table>
<thead>
<tr>
<th>SQLCODE</th>
<th>-904</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNSUCCESSFUL EXECUTION CAUSED BY AN UNAVAILABLE RESOURCE</td>
<td></td>
</tr>
<tr>
<td>REASON</td>
<td>00C90305</td>
</tr>
<tr>
<td>TYPE OF RESOURCE</td>
<td>'100'x (for database)</td>
</tr>
<tr>
<td>RESOURCE NAME</td>
<td>'WORKFILE DATABASE'</td>
</tr>
<tr>
<td>SQLSTATE</td>
<td>57011</td>
</tr>
</tbody>
</table>

The MAXTEMPS DSNZPARM can protect your workfile from becoming exhausted by runaway queries and declared global temporary tables, however make sure you have enough 32 KB WORKFILE space allocated to avoid -904 resource unavailable failures for large declared global temporary tables (DGTT) since they cannot span workfiles. Pertinent APARs on space reuse and performance for DGTT for INSERT/DELETE are PK62009, PK67301, PK70060, and PM02528. The general recommendation is to allocate workfiles with secondary extents for use by DGTT.

In-memory workfile support is provided when the final sort output data that would have been stored in a work file is less than the 4 KB or 32 KB pagesize of the selected work file. This means that small sorts are not written out to the work file, but the results are presented directly from the WORKFILE 4 KB or 32 KB buffer. This usage of in-memory workfile support provides a performance enhancement. In one test measurement, we achieved a 10 to 30% CPU reduction for small sorts. This enhancement is not available for declared global
temporary tables (both user-defined and used by DB2) as they are always written to the WORKFILE.

In-memory workfile support is expected to be of most benefit for online transactions with relatively short-running SQL calls in which the number of rows that are sorted can be small.

### 4.16.2 Instrumentation for workfile sizing

The counters for monitoring temporary space utilization at the DB2 subsystem level are included in the IFCID 2 (statistics record). Some new counters have been added to the Data Manager Statistics block DSNDQIST to track storage usage in the workfile database. These fields are written out in IFCID 2:

- **QISTWFCU**: Current total storage used, in MB
- **QISTWFMU**: High watermark, maximum space ever used, in MB
- **QISTWFMX**: Maximum allowable storage limit (MAXTEMPS) for an agent, in MB
- **QISTWFNE**: Number of times the maximum allowable storage limit per agent was exceeded
- **QISTWF04**: Current total 4 KB-page table space storage used, in MB
- **QISTWF32**: Current total 32 KB-page table space storage used, in MB
- **QISTW04K**: Current total 4 KB-page table space storage used, in KB
- **QISTW32K**: Current total 32 KB-page table space storage used, in KB
- **QISTWFP1**: The number of times that a 32 KB-page tablespace was used when a 4 KB page tablespace was preferable (but not available)
- **QISTWFP2**: The number of times that a 4 KB-page tablespace was used when a 32 KB-page tablespace was preferable (but not available)

IFCID 342 provides information on workfile and temporary database usage.

A new IFCID 343 trace record will be written when the MAXTEMPS DSNZPARM limit for an agent is exceeded. The new IFCID 343 trace record is classified as trace type PERFM CLASS 3 and STAT CLASS(4).

### 4.16.3 Workfile performance

Maintenance (PK70060/UK46839 and PK67691/UK47354) has differentiated the workfile behavior between normal workfile use and DGTT use.

It is strongly advised that multiple table spaces with zero secondary quantity be defined for work files use in the WORKFILE database. DB2 gives preference to table spaces with zero secondary quantity when allocating space for workfiles. Multiple workfile table spaces help in supporting efficient concurrent read and write I/Os to work files.

If applications use Declared Global Temporary Tables (DGTTs), it is also advised that some table spaces be defined with non-zero secondary quantity in the WORKFILE database. This will minimize contention for space between work files and DGTTs, since DB2, when allocating space for DGTTs, gives preference to table spaces that can grow beyond the primary space allocation (with SECQTY > 0), since DGTTs are limited to one table space and cannot span multiple table spaces as the work files can.

If there will be no DGTTs, then it is better that all table spaces are defined with zero secondary quantity.
The more recent APAR PM02528 (PTF UK56046) introduced a new DSNZPRM in DSN6SPRM: WFDBSEP YES or NO. This parameter specifies whether DB2 should provide an unconditional separation of table spaces in the workfile database based on the table spaces’ allocation attributes.

If the value is YES, DB2 always directs declared global temporary table (DGTT) work only to DB2-managed (STOGROUP) workfile table spaces defined with a non-zero SECQTY and workfile work only to other workfile table spaces (DB2-managed table spaces defined with a zero SECQTY or user-managed table spaces). If no table space with the preferred allocation type is available, DB2 issues an error (message DSNT501I and SQLCODE -904).

If the value is NO, DB2 attempts to direct declared global temporary table (DGTT) work to DB2-managed (STOGROUP) workfile table spaces defined with a non-zero SECQTY and workfile work to any other workfile table space (DB2-managed table spaces defined with a zero SECQTY or user-managed table spaces). If no table space with the preferred allocation type is available, DB2 selects a table space with a non-preferred allocation type. In other words, the existing workfile database behavior as established with APAR PK70060 (PTF UK46839) remains.

The default value is NO. The parameter WFDBSEP is opaque in DB2 9 (documented but not panel-supported) and its value can be changed starting with DB2 9 CM.

Depending on the number of table spaces and the amount of concurrent activity, performance can vary. In general, adding more table spaces improves the performance. You may have to fine-tune the combination of different types of table spaces in the workfile database.

4.17 Native SQL procedures

The difference between a native SQL procedure and an external SQL procedure is whether DB2 creates an external program for it and how the procedure runs. For an external SQL procedure, which was the only type of SQL procedure supported prior to V9, DB2 creates an external program to implement the processing logic. The procedure runs in a WLM-established stored procedures address space (WLM-SPAS). The SQL procedure implementation in DB2 V9 has been enhanced to introduce native SQL procedures that run entirely in the DBM1 address space. A native SQL procedure does not use the WLM stored procedure controlled execution environment.

See DB2 9 for z/OS Technical Overview, SG24-7330, and the DB2 Version 9.1 for z/OS SQL Reference, SC18-9854, for a description of the CREATE PROCEDURE statements.

A native SQL procedure is fully integrated into DB2 V9. It provides improved compatibility with DB2 for Linux, UNIX, and Windows, and DB2 for i5/OS®. It also eliminates the requirement of an external SQL procedure for a C compiler on the z/OS system. This simplifies the deployment of the SQL procedures and ensures portability.

Prior to DB2 V9, the SQL procedure language had to run under the control of the WLM-SPAS. DB2 V9 has changed this process so that the SQL procedure no longer runs in a WLM-SPAS; the SQL procedure now runs natively in the DBM1 address space.

Running SQL procedures in DB2 V8 in a WLM address space incurred the overhead of communications between the WLM-SPAS and the DBM1 address space for each SQL call. For systems that run heavy stored procedure workloads that are composed of many short running stored procedures, at or near 100% CPU utilization, this added overhead could potentially inhibit the amount of work being processed. DB2 V9 provides support for native SQL procedures that run entirely within DB2 and do not require a WLM-SPAS. By running the
SQL procedure in the DBM1 address space, it avoids the stored procedure invocation overhead as well as the round-trip cost of switching between WLM and the DBM1 address space for each SQL call.

Internal throughput rate (ITR) improvements of between 0% and 40% have been observed in comparison to running an external SQL procedure. The SQL procedure workload could be zIIP eligible if it is a DDF request via the DRDA because it runs in the DBM1 address space under a DDF enclave SRB.

The key factor is the ratio of the time that is spent executing SQL statements to the number of SQL statements. We expect to see little or no difference in ITR if it is an SQL procedure with a few long-running SQL statements because the most performance gain is in the removal of the address space switching process for SQL calls. A long running SQL procedure realizes little performance gain because the address space switching and WLM to DBM1 address space overhead are only a small part of the total CPU consumption. On the other end, if there are many short-running SQL statements, it shows a higher difference because more of the overall time is spent going across the API.

For information about native SQL procedure control statements, refer to *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854. For a general description about the new functionalities of SQL procedures, see *DB2 9 for z/OS Technical Overview*, SG24-7330. For considerations on migrating external to native SQL procedures see the technote *Converting an external SQL procedure to a native SQL procedure* available from:


The TEXT column of the SYSIBM.SYSROUTINES table points to the auxiliary table SYSIBM.SYSROUTINESTEXT. This is required to hold the large object (LOB) data. The data held in auxiliary table SYSIBM.SYSROUTINESTEXT is the character large object (CLOB) data that is the source text of the CREATE or ALTER statement with the body for the routine.

We compared the DB2 V8 SQL (external) procedure implementation with the DB2 V9 SQL procedure native and external implementations. We selected three workloads to compare (Figure 4-19 on page 130):

- **Simple workload**: This workload is a stored procedure that executes two simple SELECT statements to give a simple non-complex query comparison across the versions.

- **Relational Warehouse Workload**: This workload is a benchmark that is centered around the principal transactional activities of an order-entry environment. It consists of seven stored procedures in a predefined mix for entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses.

- **Financial workload**: This workload is meant to be representative of a financial company's OLTP environment. There is a representative mix of transaction types of varying complexity and scalability, read-only and read-write functions, and e-business style business-to-business (B2B) and business-to-consumer (B2C) interactions. This test workload did not include a DB2 V9 external procedure comparison test.
The comparison of the three workloads had the following results:

- **For the simple workload**
  - A 19% improvement in external throughput rate (ETR) with DB2 V9 native SQL procedures
  - An 11% improvement in ITR with DB2 V9 native SQL procedures

- **For the Relational Warehouse Workload**
  - A 27% improvement in ETR with DB2 V9 native SQL procedures
  - A 37% improvement in ITR with DB2 V9 native SQL procedures
  - zIIP total redirect eligibility (IIP+IIPCP) increased from 8% to 55%

In the SQL procedure support in DB2 V8, there are cases where SQL functions are converted into SELECT statements when the original SQL procedure source is precompiled. The same SQL functions are converted into SET statements in the DB2 V9 native SQL language support. This occurs in two of the Relational Warehouse Workload transactions and creates the higher performance benefit that is seen.

- **For the financial workload**
  - Equivalent ETR between DB2 V9 native and V8 external SQL procedures
  - An 18% improvement in ITR with DB2 V9 native SQL procedures
  - zIIP total redirect eligibility (IIP+IIPCP) increased from 6% to 50%

### 4.17.1 Conclusion

Significant throughput benefits can be realized by running SQL procedures natively in DB2 V9.
4.17.2 Recommendation

There are several usability advantages when using the new V9 native SQL procedures:

- Greater consistency with SQL procedures that are supported by the DB2 family, which greatly reduces the steps that are needed to test and deploy SQL procedures and increases portability across the DB2 platforms
- Increased functionality, such as new special registers for procedures and use of nested compound statements
- The ability to define multiple versions of a native SQL procedure
- DB2 and DSN commands for native SQL procedures
- Sizable additional zIIP processing eligibility for remote native SQL procedures

We make the following recommendations:

- From a performance point of view, converting your current critical SQL procedures to the new format may take advantage of the native execution in DB2 reducing the usage of the WLM stored procedure address spaces and increasing throughput.
- Define the size of the SYSIBM.SYSROUTINESTEXT auxiliary table so that it is large enough to contain the native SQL procedures in your environment.

4.18 Index look-aside

The objective of the index look-aside technique is to minimize the number of getpage operations that are generated when an individual SQL statement or DB2 process is executed repeatedly and makes reference to the same or nearby pages. Index look-aside results in a significant reduction in the number of index and data page getpage requests when an index is accessed in a sequential, skip-sequential, or hot spot pattern. This happens often with applications that process by ordered values.

DB2 keeps track of the index value ranges and checks whether the required entry is in the leaf page accessed by the previous call. It also checks against the lowest and highest key of the leaf page. If the entry is found, DB2 can avoid the getpage and traversal of the index tree.

If the entry is not within the cached range, DB2 checks the parent non-leaf page's lowest and highest key. If the entry is found in the parent non-leaf range, DB2 has to perform a getpage but can avoid a full traversal of the index tree. If an entry is not found within the parent non-leaf page, DB2 starts an index probe from the index root page.
A *DB2 index* is a set of key data that is organized into a “B-tree” structure. Figure 4-20 shows an example of a three-level DB2 index. At the bottom of the tree are the leaf pages of the index. Each leaf page contains a number of index entries that consist of the index key itself and a pointer or pointers, known as a record identifier (RID), which are used to locate the indexed data row or rows. Each entry in the intermediate non-leaf index page (LEVEL 2) identifies the highest key of a dependent leaf page along with a pointer to the leaf page’s location. At the top of the tree, a single page, called the **root page**, provides the initial entry point into the index tree structure. Like non-leaf pages, the root page contains one entry for each dependent LEVEL 2 page.

Index look-aside consists of DB2 checking whether the required entry is in the leaf page that was accessed by the previous call. If the entry is not found, DB2 checks the range of pages that are addressed by the parent non-leaf page. If the entry is not found there either, DB2 goes to the top of the index tree structure and establishes an absolute position on the required leaf page.
For example, assume that your table has the data and index structure shown in Figure 4-21.

```
Figure 4-21   Index and data structure
```

Assume also that, from a sorted input file, your program reads the key of the rows that you want to access and processes as the pseudocode shows in Example 4-12.

```
Example 4-12   Sample sequential processing program to access the data

READ INPUT FILE
DO WHILE (RECORDS_TO_PROCESS)
SELECT * FROM TABLE
WHERE KEY = INPUT_KEY
UPDATE TABLE
READ INPUT FILE
END DO;
```

We assume that the input file is sorted in the clustering index sequence and the index is clustered. When the program executes, the sequence of getpages is:

- For the first select to a row in D: getpages A, B, C, D
- For the remaining selects to rows in D: no getpages, index look-aside
- For first select to a row in E: getpage E
- For the remaining selects to rows in E: no getpages, index look-aside
- For the first select to a row in H: getpages K, H
- For the first select to a row in Q: getpages A, O, P, Q

Compare the number of getpages that are required for random access in the example table. This is usually four per select: one per index level plus one per data page. A reduced number of getpages leads to reduced CPU time and elapsed time.

Index look-aside was introduced in DB2 V4 for SELECT. In DB2 V8, index look-aside was added for additional indexes, for clustering an index during INSERT. In DB2 V9, it is possible for more indexes to use the index look-aside function for DELETE.
4.18.1 Conclusion

The implementation of index look-asides can be of considerable benefit by reducing the number of index getpage requests and the number of times that index traversal needs to be done. The savings are application-oriented but can be considerable in I/O reduction.

4.19 Enhanced preformatting

The DB2 preformat quantity increased from 2 cylinders at DB2 V8 to 16 cylinders at DB2 V9. This can be 16 cylinders per stripe for an extended format data set. The increase in preformat quantity has the added advantage that updates to the ICF catalog to register the high-used-RBA are reduced in number. Instead of being every 2 cylinders, it is now only every 16 cylinders giving increased throughput. Note that every time the data set extends with a new extent, the acquisition and the first preformat operation are synchronous, so you want to aim to reduce the number of new extents that a data set acquires.

The increase in DB2 preformat quantity is of main benefit to single stream sequential mass inserts and the inserting of large rows or LOBs. Figure 4-22 shows the preformatting improvements for 4 KB control interval (CI) and 32 KB CI sizes. The 4 KB preformatting throughput was increased by 47.2% compared to DB2 V8, and the 32 KB preformatting throughput increased by 45.7%.

![Figure 4-22 Preformatting improvements](image)

4.20 Hardware enhancements

DB2 9 brings improved synergy with System z hardware. DB2 makes unique use of the z/Architecture instruction set, and recent instructions provide improvements in reliability, performance, and availability. DB2 continues to deliver synergy with hardware data...
Chapter 4. DB2 subsystem performance

The latest System z9 processor improvements for DB2 are the zIIP and the new Business Class and Enterprise Class processors. DB2 uses the latest improvements in hardware and operating system to provide better performance, improved value, more resilience, and better function.

DB2 V9 benefits from large real memory support, faster processors, and better hardware compression. DB2 uses parallel access volume (PAV) and multiple allegiance features of the DS8000 and Enterprise Storage Server (ESS) DASD subsystems. FlashCopy is used for DB2 Backup and Restore utilities.

For more details, see Chapter 2, “Synergy with System z” in DB2 9 for z/OS Technical Overview, SG24-7330.

4.20.1 Use of the z/Architecture long-displacement facility

DB2 V9 extensively uses the long-displacement instruction set that is available in z/Architecture mode, which was introduced with 64-bit addressing. These long-displacement facility instructions enable DB2 to use register-constraint relief by reducing the need for base registers, code size reduction by allowing fewer instructions to be used, and additional improved performance through removal of possible address-generation interlocks.

The long-displacement facility provides a 20-bit signed displacement field in 69 previously existing instructions (by using a previously unused byte in the instructions) and 44 new instructions. This 20-bit signed displacement allows relative addressing of up to 524,287 (512K -1) bytes beyond the location that is designated by a base register or base and index register pair and up to 524,288 (512K) bytes before that location. The enhanced previously existing instructions generally are ones that handle 64-bit binary integers. The new instructions generally are new versions of instructions for 32-bit binary integers.

These instructions are available on the System z990, z890, and System z9 models in z/Architecture mode that exploit the long-displacement instruction set. Previous CPU models do not exploit such hardware support and are run in emulation mode. Therefore some penalty in CPU overhead will occur on System z900 and z800 models because the hardware support is simulated by microcode. Running DB2 V9 on a z900 or z800, you can expect to see an average of a 5 to 10% CPU increase. However in a column-intensive processing workload, this CPU increase will be greater than this expected 5% to 10%. Calculations have shown this CPU overhead could be as high as 100% for the long displacement facility emulation for some column-intensive workloads.

4.20.2 DASD striping of archive log files

The ability to now take advantage of DASD striping for the DB2 archive logs allows the redirect process to run at the same speed as the DB2 active log. Prior to DB2 V9, the DB2 archive logs were read with the Basic Direct Access Method (BDAM). The BDAM process had advantages in allowing direct access to a target CI, but limited the use of newer technologies to provide performance enhancements for the archive logs. The ability to use the same technology as the DB2 active logs means that the DB2 archive log offload process can keep up with speed that the DB2 log fills. If it takes longer to write an archive log copy than the active log from which the archive data is coming, then on a busy system that is doing a lot of logging, there is the possibility that logging may be suspended while waiting for the archive log processing to catch up.
To achieve this technology equivalence for the active and archive logs, the DB2 log archive process has been modified to use the Basic Sequential Access Method (BSAM) I/O process for reading the archive log. This use of the BSAM I/O process has allowed the use of striped data sets and other Data Facility Storage Management Subsystem (DFSMS) features that are available for extended format data sets. This means that the DB2 archive log offload process can keep up with speed that the DB2 active log fills.

The ability to use striped data sets for the DB2 archive logs is available in all DB2 V9 modes. The use of extended format data sets should be limited to new-function mode because they are not supported if you fall back to DB2 V8.

Recommendation
Where possible, we recommend that you define the DB2 archive logs to take advantage of DFSMS extended format data set characteristics. Taking advantage of the striping that DFSMS extended format data sets can use can help alleviate any potential archive offload bottlenecks in your system.

The use of extended format data sets can be limited to new-function mode only. With new-function mode, there is no possibility of falling back to a previous DB2 version that does not support extended format data sets.

4.20.3 Hardware support for the DECFLOAT data type

DB2 9 provides support of the decimal floating point (DECFLOAT) data type. DECFLOAT is similar to both packed decimal and floating point (IEEE or Hex) data types. Floating point can only approximate common decimal numbers. DECFLOAT can represent a decimal number exactly and can represent much bigger and smaller numbers than DECIMAL. See 2.13.3, “DECFLOAT” on page 45, for more information and performance comparisons.

Because of the complex architecture of the zSeries processors, an internal code, called millicode, is used to implement many of the functions that are provided by these systems. While the hardware can execute many of the logically less complex and high-performance instructions, millicode is required to implement the more complex instructions, as well as to provide additional support functions related primarily to the central processor.

IBM is implementing hardware decimal floating point facilities in a System z9 processor. The facilities include 4-, 8-, and 16-byte data formats, an encoded decimal (base-10) representation for data, instructions for performing decimal floating point computations, and an instruction that performs data conversions to and from the decimal floating point representation. This is clarified in the 18 April 2007 announcement, which is available on the Web at:

http://www.ibm.com/common/ssi/rep_ca/0/897/ENUS107-190/ENUS107190.PDF

Base 10 arithmetic is used for most business and financial computation. To date, floating point computation, which is used for work that is typically done in decimal arithmetic, has involved frequent necessary data conversions and approximation to represent decimal numbers. This has made floating point arithmetic complex and error prone for programmers who use it in applications where the data is typically decimal data.

Initial software support for hardware decimal floating point is limited to high level Assembler support running in z/OS and z/OS.e on a System z9 processor. z/OS V1.9 provides support for hardware decimal floating point instructions and decimal floating point data types in the C and C++ compilers as a programmer-specified option. Support is also provided in the C Run Time Library and the DBX Debugger. No support is available for machines earlier than the System z9 processor.
For more information about requirements and restrictions for DECFLOAT, refer to Program Directory for IBM DB2 9 for z/OS, GI10-8737. The DECFLOAT format is generated by the hardware instruction on machines that have this instruction available.

**Restriction:** A DECFLOAT column cannot be defined as key column of an index. See also 2.13.3, “DECFLOAT” on page 45.

Chapter 4. DB2 subsystem performance

DB2 9 uses System z9 hardware support for the new DECFLOAT data type. This data type allows you to use decimal floating point numbers with greater precision and to reflect decimal data exactly without approximation. DB2 detects whether the processing complex it is running on has the hardware support to process the DECFLOAT data type. If the hardware support is not installed, then DB2 uses a software emulation to process the DECFLOAT data type. This software emulation in DB2 uses extra CPU cycles, and this extra processing shows in the DB2 class 2 CPU time.

Two tests were done to evaluate the CPU reduction that System z9 hardware support provides for the DECFLOAT data type. The first test was a simple test with a single arithmetic function (see Example 4-13). The second was a more complex test with multiple arithmetic functions (see Example 4-14). The tests were done by selecting one million rows of DECFLOATs with and without hardware support for DECFLOAT. The results showed that the hardware support improves the performance but only for certain operations. The more arithmetic operations and DECFLOAT(16) and DECFLOAT (34) castings we do, the greater the performance improvement is with hardware support enabled.

The performance of the simple query shown in Example 4-13 showed that, with the hardware support for DECFLOAT processing, there was an observed 3% class 2 CPU reduction compared to using software emulation code for the one million row selects.

Example 4-13  Simple testcase with single DECFLOAT(16) <-> DECFLOAT (34) casting

```sql
SELECT HEX(DECFLOAT_COL01 + DECFLOAT_COL02) INTO :CHAR01 FROM TABLE1
```

Example 4-14 illustrates a more complex SQL statement with many DECFLOAT operations using hardware millicode versus software emulation.

Example 4-14  Complex testcase with multiple DECFLOAT(16) <-> DECFLOAT (34) castings

```sql
SELECT HEX(DECFLOAT(DECFLOAT_COL01, 34) + DECFLOAT(DECFLOAT_COL02, 34) +
DECFLOAT(DECFLOAT_COL03, 34) + DECFLOAT(DECFLOAT_COL04, 34) +
DECFLOAT(DECFLOAT_COL05, 34) + DECFLOAT(DECFLOAT_COL06, 34) +
DECFLOAT(DECFLOAT_COL07, 34) + DECFLOAT(DECFLOAT_COL08, 34) +
DECFLOAT(DECFLOAT_COL09, 34) + DECFLOAT(DECFLOAT_COL10, 34) +
DECFLOAT(DECFLOAT_COL11, 34) + DECFLOAT(DECFLOAT_COL12, 34) +
DECFLOAT(DECFLOAT_COL13, 34) + DECFLOAT(DECFLOAT_COL14, 34) +
DECFLOAT(DECFLOAT_COL15, 34) ) INTO :CHAR01 FROM TABLE1
```
The performance of the complex query shown in Example 4-14 showed that, with the hardware support for DECIMAL processing, there was a 46.7% class 2 CPU reduction compared to using software emulation code for the query with multiple arithmetic functions. See Figure 4-23.

Conclusion

Based on the testing that was done, we can expect anywhere between a 3% to 50% DB2 class 2 CPU reduction when hardware support is available for the DECIMAL data type. The amount of CPU reduction depends on how many arithmetic calculations and DECIMAL(16) <-> DECIMAL(34) castings you do. On machines where this instruction is not available, the conversion is done in software. The results from lab measurements show that hardware support does improve the performance, but only for certain operations. The more arithmetic operations and DECIMAL(16) <-> DECIMAL(34) castings that are performed, the greater the performance improvement is with hardware support enabled.

4.20.4 zIIP usage

The special purpose zIIPs have been made available on all System z systems, starting from the System z9 Enterprise Class and z9 Business Class servers. zIIPs are used to redirect specialized processing elements from z/OS, where these elements are initially associated with various types of DB2 processing. For more details about specialty engines that are used by DB2 V9, see DB2 9 for z/OS Technical Overview, SG24-7330.

DB2 V8 moves eligible DRDA, a selected DB2 utility, part of Business Intelligence (BI) workloads, and most index management work of the utilities to a zIIP, reducing software cost and improving available capacity of existing general purpose engines.

DB2 V9 adds remote native SQL procedures execution and more instances of eligible query parallelism (BI) work. Furthermore, IBM has announced that z/OS XML System Services processing will execute on the System z Application Assist Processor (zAAP) and that DB2
can direct the full amount of the z/OS XML System Services processing to zIIPs when it is used as part of any zIIP eligible workload, like DRDA.

The prerequisites for zIIP usage with DB2 V9 are:

- z/OS release V1R7 or later
- System z9 Enterprise Class or z9 Business Class with zIIPs or later (currently z10 and zEnterprise)

zIIP support is incorporated in z/OS V1R8 base and is available as a Web installable function modification identifier (FMID) for z/OS V1R7. zIIP support for z/OS V1R7 (minimum z/OS level for DB2 V9) is available via FMID JBB772S.

The zIIP is for users who need to redirect standard processor work to offset software and hardware costs. The biggest cost reductions are in software, because IBM does not charge for software that runs on the zIIPs. There is also a reduction in hardware cost for a zIIP, which is much less than a standard processor.

The zIIP specialty processor is restricted in the workload that can run on it. The workload that can be scheduled on a zIIP must be running under an enclave SRB. DB2 V9 requests an eligible workload to be scheduled on a zIIP where one is enabled.

If you are not currently running a zIIP, but you want to use a tool, such as IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS (or RMF), to determine how much work can be offloaded to the zIIP, then set the PROJECTCPU control in the SYS1.PARMLIB IEAOPTxx member to YES. PROJECTCPU is used to enable zIIP usage projection without a real zIIP being enabled. This WLM function enables estimation of how much zIIP redirect can occur as reported in zIIP-eligible CPU (IIPCP) time when a zIIP is not available.

When a zIIP is installed, you can measure its workload with the relevant RMF reporting functions. A tool like DB2 Performance Monitor can process the DB2 accounting trace records.

Note the following points for monitoring the DRDA zIIP activity:

- Set up a WLM policy with a service class or classes for SUBSYSTEM TYPE=DDF.
- RMF Monitor 1 Type 70 Record monitors the overall zIIP activity:
  - Logical processor busy as seen by z/OS is reported.
  - Physical processor busy as seen by LPAR is reported.
- RMF Monitor 1 Type 72 Record shows more detail:
  - The amount of time spent executing on zIIPs is reported.
  - Usage and delay sample counts for zIIP eligible work are reported.
- DB2 accounting trace records can provide information about the zIIP because zIIP relevant data is provided in IFCID 3,147,148, 231, and 239.

IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS, DB2 Performance Expert, or IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS can be used to monitor the zIIP information provided in the accounting trace data.

In the batch reporting, the CPU time is now reported as three fields: normal central process (CP) processing, zIIP eligible, and actual zIIP. These fields are named as follows:

- The standard CPU TIME (non-
  zIIP) is named CP CPU TIME.
- zIIP-eligible CPU time is named IIPCP CPU.
- zIIP time is named IIP CPU TIME.
Figure 4-24 shows the formula for calculating the DRDA zIIP redirect percentage when zIIP is being used. The APPL% IIP value indicates processing on the zIIP. The APPL% IIPCP value indicates any zIIP eligible processing that ran on CP because zIIP was busy or not installed. A high non-zero value indicates a need to configure more zIIPs. In this example, the zero value for APPL% IIPCP indicates that there is no need to configure additional zIIPs for this workload.

The DRDA zIIP redirect % can also be calculated using the service times:

Service Times IIP / Service Times CPU

<table>
<thead>
<tr>
<th>RMF Workload Activity Report Showing CLI SQL DRDA zIIP Redirect</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPORT BY: POLICY=DRDAIC1 WORKLOAD=DB2 SERVICE CLASS=DFDFWORK RESOURCE GROUP=*NONE</td>
</tr>
<tr>
<td>INTERVAL: 54 Sec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Report</th>
<th>Service Time</th>
<th>APPL %</th>
<th>IIP</th>
<th>CPU</th>
<th>SRB</th>
<th>AAPCP</th>
<th>CPU %</th>
<th>IIPCP</th>
<th>AAP</th>
<th>CPU %</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG</td>
<td>2.90</td>
<td>29.3</td>
<td>16.2</td>
<td></td>
<td></td>
<td>0.0</td>
<td></td>
<td>0.0</td>
<td></td>
<td>24.20</td>
</tr>
<tr>
<td>MPL</td>
<td>14</td>
<td>SSCHRT</td>
<td>507.2</td>
<td>I/O</td>
<td></td>
<td>831475</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENDED</td>
<td>11384</td>
<td>QUEUED</td>
<td>0.2</td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>END/S</td>
<td>207.84</td>
<td>R/S AFFIN</td>
<td>0.0</td>
<td>DISC</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$SWAPS</td>
<td>0</td>
<td>INELIGIBLE</td>
<td>0.0</td>
<td>Q+PEND</td>
<td>0.1</td>
<td>TOT</td>
<td>831425</td>
<td>HST</td>
<td>0.0</td>
<td>AAP</td>
</tr>
<tr>
<td>EXECBD</td>
<td>0</td>
<td>CONVERSION</td>
<td>0.0</td>
<td>IOSQ</td>
<td>0.0</td>
<td>IDC</td>
<td>15179</td>
<td>AAP</td>
<td>0.0</td>
<td>IIP</td>
</tr>
<tr>
<td>AVG ENC</td>
<td>2.90</td>
<td>STD DEV</td>
<td></td>
<td>ABSRPTN</td>
<td>5243</td>
<td>TRX SERV</td>
<td>5243</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REM ENC</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS ENC</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Service Times : CPU time includes IIP and AAP time

APPL % is % of a single engine.

APPL% IIP = Service Time IIP / Report Interval
APPL% CP = (Service Time CPU+SRB+RCT+IIT-AAP–IIP) / Report Interval

Using WLM Service Class DDFWORK
Redirect % = Service Time IIP / Service Time CPU
= APPL% IIP / (APPL% CP+APPL% IIP)

Figure 4-24  DRDA redirect using RMF
Figure 4-25 shows the DRDA workload zIIP redirect percentage using the DB2 Performance Expert accounting report for Connect Type DRDA. *IIP CPU time* is the CPU time on zIIP. *IIPCP CPU time* shows any zIIP eligible processing that ran on CP because the zIIP was busy or not installed. A high non-zero value indicates a need to configure more zIIPs. In this example, the zero value for IIPCP CPU indicates that there is no need to configure additional zIIPs.

**Figure 4-25 DRDA redirect using OMEGAMON Performance Expert**

The zIIP estimated redirect percentage is calculated by dividing the IIPCP% by the CP% in the APPL% column to the far right of the report. In this case, the values 4.56 and 17.44 show an estimated redirect percentage of 26%.

**Figure 4-26 RMF report showing a zIIP redirect% estimate from PROJECTCPU=YES**

The zIIP estimated redirect percentage is calculated by dividing the IIPCP% by the CP% in the APPL% column to the far right of the report. In this case, the values 4.56 and 17.44 show an estimated redirect percentage of 26%.
Figure 4-27 shows an IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS report of a workload that ran on a system with a zIIP enabled. The report also shows that, where you have eligible zIIP redirect work and a zIIP is not available, this value is reported as well. This can be an indicator that the zIIP is saturated with other DB2 workloads, and adding extra zIIPs may give performance enhancements.

### Conclusion

A zIIP specialty engine may be a cost-effective solution for some DB2 workloads. The actual cost savings depends on the software pricing model used and the workloads that drive the peak CPU usage.

zIIP has an easy implementation with no DB2 application or subsystem changes and no external tuning options. zIIP can be leveraged to grow, develop, or port new distributed and business intelligence applications on DB2 for z/OS in a cost-effective way.

For more information about zIIP, refer to the following Web address:

http://www.ibm.com/systems/z/ziip/

---

### 4.20.5 DASD improvements

DB2 has traditionally exploited DASD functions and integrated them to provide better access solutions. *Disk Storage Access with DB2 for z/OS, REDP-4187,* explores several of the new DASD features and the improved synergy in versions up to DB2 for z/OS Version 8. DB2 V9 continues the trend by taking advantage of the recent new I/O hardware.

#### PAV and multiple allegiance utilization

We have always sought to avoid I/O contention and increase DASD throughput through the I/O subsystem by placing frequently used data sets on fast disk devices and by distributing I/O activity throughout the DASD subsystems. While this still holds true, distributing the I/O activity is less important when you use disk devices with PAV support and multiple allegiance support.
The PAV feature allows multiple concurrent I/Os on a given device when the I/O requests originate from the same system. PAVs make possible the storing of multiple partitions on the same volume with almost no loss of performance. In older disk subsystems, if more than one partition is placed on the same volume (intentionally or otherwise), attempts to read the partitions result in contention, which shows up as I/O subsystem queue (IOSQ) time. Without PAVs, poor placement of a single data set can almost double the elapsed time of a parallel query.

The multiple allegiance feature allows multiple active concurrent I/Os on a given device when the I/O requests originate from different systems. PAVs and multiple allegiance dramatically improve I/O performance for parallel work on the same volume by nearly eliminating IOSQ or PEND time and drastically lowering elapsed time for transactions and queries.

We recommend that you use the PAV and multiple allegiance features for the DASD volumes and subsystems that have the DB2 data stored.

HyperPAV
In November 2006, IBM made available HyperPAV, a disk technology which is a step forward for Parallel Access Volumes (PAVs) for the IBM System Storage DS8000 series (M/T 2107). Made possible through the storage capabilities provided by the IBM System Storage DS8000 subsystem for System z processors, mainframe FICON technology and host OS software, HyperPAV has been designed to drastically reduce the queuing of I/O operations. EAV (Extended Addressable Volumes) complements HyperPAV by enabling z/OS to manage more storage. HyperPAV is especially useful for remote mirroring and for managing solid state disks.

Addresses continue to be pooled by LCU (Logical Control Unit), and there is a limit of 256 addresses per LCU. In theory, each LCU can execute a maximum of 256 concurrent I/Os. However, prior to HyperPAV, several factors contributed to limiting the I/O concurrency. Alias addresses were bound to a specific volume. To rebind an alias to a different volume took time and system resources, because a rebind had to be coordinated throughout the sysplex and with the DS8000. If different systems in the sysplex all contended for the same LCU and used different volumes, they could steal aliases from each other and create a thrashing situation. HyperPAV enables the I/O supervisor to immediately select an alias from the LCU address pool without notifying other systems, because the aliases are no longer bound to a volume. Thus, it is more likely that each system could actually perform 256 concurrent I/Os for each LCU. Only the base addresses used for each volume are, in effect, statically bound to a volume.

I/O concurrency is often limited by the physical limitations of spinning disks. Sometimes HyperPAV will have the effect of shifting the queuing from the UCB queues to the disks. However, HyperPAV is useful for cache friendly workloads and for remote mirroring since a UCB is tied up while data is transmitted over a remote link. In effect, Hyperplane makes it less likely that read performance will be impacted by poor performance for remote write operations. HyperPAV is also important to achieve high throughput for solid state disks.

HyperPAV requires the DS8000 Release 2.4, and z/OS Release 1.8. PTFs are also provided for z/OS 1.6 and 1.7.

FICON channels
The older ESCON® channels have a maximum instantaneous data transfer rate of approximately 17 MB per second. The newer FICON channels currently have a speed of 4 GB per second, and the FICON speed is bidirectional, which theoretically allows 4 GB per second to be sustained in both directions. Channel adapters on the host processor and the storage server limit the actual speed that is achieved in data transfers.
FICON channels in System z9 servers are faster than those in the previous processors and feature MIDAW (which stands for modified indirect address word) improvements.

**MIDAW**

The MIDAW facility was introduced in the System z9 processor to improve FICON bandwidth, especially when accessing IBM DB2 databases. This facility is a new method of gathering data into and scattering data from discontinuous storage locations during an I/O operation. MIDAWs reduce the number of frames and sequences that flow across the FICON link, which makes the channel more efficient.

DB2 9 use of MIDAWs requires it to be run on the System z9 processor and requires use of the IBM z/OS 1.7 operating system.

MIDAWs are implemented by the z/OS Media Manager. To take advantage of Media Manager and MIDAW, users must use data set types that are supported by Media Manager, such as linear data sets and extended format data sets. The most significant performance benefit of MIDAWs is achieved with extended format data sets.

Refer to the IBM Redpaper™ publication *How does the MIDAW Facility Improve the Performance of FICON Channels Using DB2 and other workloads?*, REDP-4201, which describes how the MIDAW facility improves the performance of FICON channels using DB2 and other workloads. This paper was written using DB2 V8 as the reference software level; the same information and recommendations also apply to DB2 V9.

**AMP**

Adaptive Multi-stream Prefetch (AMP) was first introduced in Release 2.4G of the DS8000. It does not require any z/OS changes. AMP typically improves the sequential read throughput by 30%. AMP achieves this by increasing the prefetch quantity sufficiently to meet the needs of the application. For example, if the application is CPU bound, there is no need to prefetch a lot of data from the disk. At the opposite extreme, if the application is I/O bound and the channels are very fast, then AMP will prefetch enough data to enable the “back end” operations keep up with the channel. As more data is prefetched, more disks are employed in parallel. In other words, high throughput is achieved by employing parallelism at the disk level.
DASD hardware evolution
The DS8000 and DS8000 turbo disks with channel performance improvements in fiber channels and in the System z9 processor have made large improvements in data rates for sequential reading. Figure 4-28 shows the functions that have contributed to improve the data transfer speed.

<table>
<thead>
<tr>
<th>Synergy with new I/O hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Data transfer with RAID-5 architecture</td>
</tr>
<tr>
<td>• A data set is spread across as many as eight disks enabling faster prestaging</td>
</tr>
<tr>
<td>• FICON channels are progressively much faster than ESCON channels</td>
</tr>
<tr>
<td>• System z990 introduced FICON Express 2</td>
</tr>
<tr>
<td>• System z9 introduced FICON Express 4</td>
</tr>
<tr>
<td>• DS8000 introduced faster device adapters and host adapters</td>
</tr>
<tr>
<td>• MIDAW and AMP have increased the channel efficiency</td>
</tr>
<tr>
<td>• MIDAW requires System z9 (2094) and z/OS1.6 OA10984, OA13324/13384</td>
</tr>
<tr>
<td>• DB2 9 supports larger pages for indexes, increases preformat and prefetch quantity</td>
</tr>
</tbody>
</table>

*Figure 4-28  DB2 V9 synergy with new I/O hardware*

Significant improvements have been made in I/O throughput using the newer DS8000 and turbo disks. The use of FICON Express and System z9 processors also contribute to the increases in throughput.

With DB2 V8, the data rate is improved from 40 MB/sec, on the ESS model 800, to 69 MB/sec. The use of System z9 processor and MIDAW has further improved the data rate to 109 MB/sec. With two stripes, that configuration can reach 138 MB/sec.

DB2 9 changes in read quantity, write quantity, and prefetch quantity allow the same hardware to deliver 183 MB/sec. in reading and a similar speed for writing. MIDAW has practically eliminated the performance gap between extended format data sets and non-extended format data sets.

The recent performance for sequential reading over current IBM disks has shown a sharp upward trend. The data rates climbed slowly in previous generations of disk. However, changes from a simple disk read to reading from a computer with similar amounts of memory and processing capability to the main computer have improved the speed enormously as the processing, use of memory, and parallel processing bypass the disk constraints.
Figure 4-29 represents the state of the art for DB2 sequential prefetch on current hardware.

<table>
<thead>
<tr>
<th>MB/sec</th>
<th>4 KB</th>
<th>8 KB</th>
<th>16 KB</th>
<th>32 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 V8 From Disk</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 V9 From Disk</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DV2 V8 From Cache</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 V9 From Cache</td>
<td>140</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 V8 AMP</td>
<td>160</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 V9 AMP</td>
<td>180</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 4-29  DB2 sequential prefetch*

The data rate of 160 MB/sec. with 4 KB pages, going up to over 200 MB/sec. with 16 KB pages, is the result of several contributing factors. Such factors include DB2 9 changes in preformat quantity (from 2 to 16 cylinders) and doubling the prefetch quantity. They also include DS8000 Turbo performance and the adoption of an innovative prestaging algorithm called *adaptive multi-streaming prefetching* (AMP), which is available with a 2.4 GB LIC. Even higher rates could be obtained with DSFMS data striping.

**Solid-state drives**

Recent trends in direct access storage devices have introduced the use of NAND flash semiconductor memory technology for solid-state drives (SSDs).

Flash memory is a non volatile computer memory that can be electrically erased and reprogrammed. It is a technology that is primarily used in memory cards and USB flash drives for general storage and transfer of data between computers and other digital products. Flash memory offers good read access times and better kinetic shock resistance than hard disks. Flash memory, once packaged in a memory card, is durable, can withstand intense pressure, extremes of temperature, and even immersion in water.

Because SSDs have no moving parts, they have better performance than spinning disks, or hard disk drives (HDD), and require less energy to operate than HDDs. Like spinning disks, SSDs provide random access and retain their data when powered off. SSDs cost more per unit of storage than HDDs but less than previous semiconductor storage. The industry expects these relationships to remain that way for a number of years although the gap in price should narrow over time. As a result, these technologies will likely coexist for some time.

In February 2009, IBM announced the IBM System Storage DS8000 Turbo series with solid-state drives (SSDs). Earlier, in October 2008, IBM had announced a complementary feature called High Performance FICON for System z (zHPF). zHPF exploits a new channel protocol especially designed for more efficient I/O operations. IBM recommends High Performance FICON for an SSD environment. zHPF requires a z10 processor and z/OS 1.10 (or an SPE retrofitted to z/OS 1.8 or 1.9), as well as a storage server such as the DS8000 that supports it. zHPF provides lower response times when accessing SSDs.

Preliminary measurements with SSDs were implemented by the DB2 performance department.
The measurements were done on a z9 and z10 processors with 4 FICON Express 4 channels connected to a DS8300 LPAR. The DS8300 was configured with lots of HDD disks, but only 16 SSD disks that formed two RAID 5 ranks.

**Synchronous I/O**

The performance chart of Figure 4-30 shows the DB2 synchronous I/O wait times as reported by DB2. They include the CPU time for media manager and IOS to process the I/O. When the disks get busier, the wait times go up. The response time to read a 4K page from HDD depends on the seek distance. These measurements were obtained using a 300 GB HDD drive with 15K RPM. Randomly seeking within an individual data set typically produces a wait time of 4 milliseconds (ms) or less. When the HDD has to frequently seek between from inner extreme to the outer extreme of the disk, the response time produced is 8 ms. The typical response time when the data is evenly distributed across the disk is around 6 ms.

In contrast, SSD does no seeking or rotation and the wait time for the DS8000 server is 838 microseconds, about 7 times faster than what is typical for HDD. So, whereas HDD access is 25 times slower than cache, SSD is only about 3.5 times slower than cache. For a cache unfriendly workload, this translates to up to 7 times faster transactions. A cache unfriendly workload is one that has a very large working set or a very low access density relative to the cache size.

We also observe that zHPF lowers the wait time for a cache hit by 53 microseconds and for a cache miss, zHPF lowers the wait time by 100 microseconds, or 12%. HDDs also get faster by 100 microseconds, which is insignificant if the average wait time is 6000 microseconds. Consequently zHPF enables us to say that SSDs are 8 times faster instead of 7 times faster. zHPF helps to squeeze more value out of SSD in this and other situations.

**List prefetch**

For densely clustered data, dynamic prefetch is still the fastest way to read the pages. How does SSD and HDD compare for sequential I/O? The answer is that sequential performance is not sensitive to HDD performance, because HDDs are good at streaming data. That is, there is very little seek time or rotational delay to reposition the device for sequential access. Furthermore, RAID striping overcomes any potential limitation in the disk speeds. Instead the...
component that gates sequential performance is speed of the path along which the data is transferred, including the channels and the host adapters.

In case of poorly clustered data or sparse data, DB2 uses an index to select the pages and it may elect to sort the RIDs of the pages and then fetch them using list prefetch. Since the pages are sorted, the seek times for HDD are much less than when the pages are not sorted. Furthermore, as the fetched pages become denser, the cost per page goes down, albeit DB2 has to read more pages. The DB2 buffer manager further cuts the cost by scheduling up to two prefetch engines in parallel.

As we can see from the chart in Figure 4-31, with SSD, the cost per page is independent of the number of pages fetched. That cost is only 327 microseconds, which is only 40% of the cost for random access. Part of that reason is the parallelism of list prefetch and the other is due to the fact that list prefetch reads 32 pages per I/O.

HDD actually does quite well compared to SSD when DB2 has to fetch a lot of pages. In fact, they perform exactly the same when DB2 fetches 14% of the pages.

However, if the data is entirely in cache, the cost per page is only 47 microseconds, which is still 7 times faster than SSD.

![List prefetch page cost](image)

**Figure 4-31  List prefetch (microseconds)**

- 47 microseconds from cache
- 327 microseconds from SSD

**Conclusion**

The conclusion from the early measurements is that SSDs:

- Improve DB2 synchronous I/Os and list prefetch
- Lower energy costs
- Potentially could increase the disk durability
- Cost more, but cost is rapidly lowering

SSDs will provide consistent response times over a wider range of workload demand, and they will also simplify the performance tuning because I/O skews across the RAID ranks will have much less effect.
Together, SSDs and zHPF are the foundation for performance enhancements that could dramatically reduce the negative effects of a low cluster ratio or poorly organized indexes. One of the things learned from the early study was that 4 channels was not sufficient to maximize the performance of these 16 drives. Near the end of the project, the processor was upgraded to a z10 processor with 8 channels and finally all of the processor and z/OS upgrades were done to enable zHPF to be used.

Note that SSD technology improves one aspect of the I/O subsystem by removing the bottlenecks imposed by spinning disks. However, the disks are only one component of the I/O subsystem. Every component is important to the overall performance. While perhaps reducing the need for a large cache, SSDs will inevitably expose other bottlenecks in the I/O subsystem, such as channels, host adapters (HA), device adapters and the processor. It is important to remember to consider all of the features that have been introduced in recent years to improve every component: MIDAWs, HyperPAV, AMP, and High Performance FICON, as well as 4 gigabit/second FICON links.

Other measurements
More measurements are taking place in IBM on this interesting new technology. More documentation is being made available. For another set of preliminary performance considerations using SSD with SAP and DB2, see the white paper IBM System z and System Storage DS8000: Accelerating the SAP Deposits Management Workload With Solid State Drives, available at: http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101442

4.21 Optimization Service Center support in the DB2 engine

The IBM Optimization Service Center for DB2 for z/OS is a new tool that is part of the no-charge DB2 accessories suite for z/OS. This product at is aimed at problem query resolution. It collects information from monitoring SQL that can be used for tuning selective SQL activity, both dynamic and static. See IBM DB2 9 for z/OS: New Tools for Query optimization, SG24-7421, for more information about Optimization Service Center and the related tool Optimization Expert.

The insertion of measuring probes into a working system to collect monitoring data always has some performance overhead above the base execution environment. We created a testing environment that we used to measure the overhead of the monitoring that is implemented for the Optimization Service Center product. The testing environment was based around a batch application that runs on a 3CP System z9 processor with data on a DS8300 DASD subsystem. The batch application consisted of a mix of 60 dynamic queries that had elapsed times varying from 0.01 seconds to 80 seconds. This query mix was chosen to simulate a customer environment. The DB2 V9 environment consisted of a single non-data sharing subsystem.

The workload was run several times with varying monitoring levels set. The results were analyzed to show the percentage of overhead in DB2 class 2 CPU time that the monitoring caused. In all the test cases, the Optimization Service Center monitor output was written to a single user-defined output table. A base level was set by running the query workload against the DB2 V9 subsystem with a normal accounting trace set to allow the collection of CPU statistics used for the result analysis.

Testing was then done with an IFCID 318 trace activated. This is the instrumentation class that is used for dynamic statement caching statistics collection. This IFCID 318 instrumentation overhead can be used to compare current statistics collection techniques to
the overhead used by turning on the Optimization Service Center monitor traces. Three Optimization Service Center profiles were used: a base monitor, a monitor with CPUTIME, and a monitor with CPUSPIKE. The level of statistics that was collected was varied against each of the Optimization Service Center profiles to show how statistics collection influences the CPU overhead.

The following legend explains the workloads shown in Figure 4-32:

- Base: Normal accounting trace to which other workloads were compared
- IFCID 318: Dynamic SQL performance monitoring
- Min stats: Minimum statistics for MONITOR
- Min stats CPU: Minimum statistics for MONITOR CPUTIME/SPIKE
- Normal stats: Normal maximum statistics
- All stats: All statistics

### 4.21.1 Conclusion

An Optimization Service Center MONITOR with the minimum statistics recording has no overhead compared to a situation where no Optimization Service Center profile had been started. An Optimization Service Center MONITOR CPUTIME/SPIKE with minimal statistics has approximately a 1% DB2 class 2 CPU overhead. An Optimization Service Center MONITOR with maximum statistics will vary from 4% to 9% DB2 class 2 CPU overhead depending on the kind and amount of statistics that were collected.

### 4.21.2 Recommendation

We recommend that you use the Optimization Service Center documentation to select the level of monitoring statistics that are needed for the analysis that you want to do. Using the analysis provided in this section, estimate the percentage increase in DB2 class 2 CPU time that could occur and use this estimate to ensure you have enough CPU headroom to use the monitoring functions without impacting your workloads.
Availability and capacity enhancements

DB2 9 for z/OS continues to bring changes that improve availability and capacity, keeping up with the explosive demands of e-business, transaction processing, and business intelligence. In this chapter, we discuss the availability and capacity enhancements that are implemented in DB2 9 for z/OS to eliminate or reduce inhibitors to the full exploitation of faster and more powerful hardware.

We discuss the following topics in this chapter:

- Universal table space
- Clone table
- Object-level recovery
- Relief for sequential key insert
- Index compression
- Log I/O enhancements
- Not logged table spaces
- Prefetch and preformatting enhancements
- WORKFILE database enhancements
- LOB performance enhancements
- Spatial support
- Package performance
- Optimistic locking
- Package stability
5.1 Universal table space

Prior to DB2 9 for z/OS, you could not define a table space using both the SEGSIZE and NUMPARTS parameters. These parameters were mutually exclusive. In DB2 9 for z/OS new-function mode, this restriction is removed, and you can combine the benefits of segmented space management with partitioned table space organization.

The table spaces that are both segmented and partitioned are called universal table spaces (UTS). The combined advantages are:

- A segmented space-map page has more information about free space than a partitioned space-map page.
- Mass delete performance is improved because mass delete in a segmented table space organization tends to be faster than in other types of table space organizations.
- All or most of the segments of a table are ready for immediate reuse after the table is dropped or mass deleted.
- Partitioning allows for support of large table spaces and parallelism of accesses.

There are two types of universal table space: partition-by-growth (PBG) and partition-by-range (PBR).

**Partition-by-growth table spaces** are like single-table DB2-managed segmented table spaces. DB2 manages partition-by-growth table spaces and automatically adds a new partition when more space is needed to satisfy an insert. There is no partitioning value binding data to partitions. The table space begins as a single-partition table space and automatically grows, as needed, as more partitions are added to accommodate data growth. DB2 automatically adds a new partition when it needs more space to satisfy an insert. Compression dictionaries are copied across the new partitions. A partition-by-growth table space can grow up to 128 TB, and its maximum size is determined by MAXPARTITIONS, DSSIZE, and page size. This structure is useful to deal with table spaces that increase the overall size on demand.

**Partition-by-range (alias range-partitioned) table spaces** are based on partitioning ranges. They are similar to the traditional data partitioned table spaces with the addition of segmented characteristics. The maximum size of a partition-by-range table space is 128 TB. A partition-by-range table space is segmented. However, it contains a single table, which makes it similar to the regular partitioned table space. partition-by-range table spaces are defined by specifying both keywords SEGSIZE and NUMPARTS on a CREATE TABLESPACE statement. After the table space is created, activities that are already allowed on partitioned or segmented table spaces are allowed. You can specify partition ranges for a range-partitioned universal table space on a subsequent CREATE TABLE or CREATE INDEX statement.

Universal table spaces are identified by values in columns of the DB2 catalog table SYSTABLESPACE:

- The TYPE column value:
  - “R” Range-partitioned universal table space
  - “P” Implicit table space created for pureXML columns.
  - “O” Table space is a LOB table space
  - “G” Partitioned-by-growth table space
- MAXPARTITIONS is a new column showing the maximum number of partitions.
PARTITIONS column contains the number of physical partitions (data sets) that currently exist.

Universal table spaces are created in reordered row format (see 4.14, “Reordered row format” on page 121) by default, unless the DSNZPARM SPRMRRF is set to DISABLE (APAR PK87348).

Details on universal table spaces are listed in DB2 9 for z/OS Technical Overview, SG24-7330 and in the DB2 Version 9.1 for z/OS Administration Guide, SC18-9840.

In the next section we summarize the considerations on using universal table spaces.

5.1.1 Using universal table spaces

Universal table spaces (UTS) are at their first release, new with DB2 9 for z/OS. While the expectation is that their usage will be more pervasive in future releases of DB2, they currently offer both some enhancements and some limitations:

- Only one table per table spaces can be defined.
- No migration path for existing table spaces. Must use the sequence UNLOAD, DROP, CREATE, LOAD, followed by GRANTS and BINDs.
- ALTER of SEGSIZE or DSSIZE is not supported.
- UTS PBR requires table controlled partitioning, index partitioning is not supported.
- MEMBER CLUSTER is not supported for UTS PBR/PBG. This restriction can be a major performance inhibitor for data sharing.
- For UTS PBG:
  - LOAD cannot operate at partition level.
  - REORG PART will not re-establish the defined distributed free space if there is not enough space created by DELETE to fit the data back into the partition.
  - A full REORG could result in empty partitions.

5.1.2 Recommendations

Use the new universal table space, partition-by-growth or partition-by-range, when you want to benefit from the usability improvements that are available with this new type of table space in DB2 9 for z/OS, namely the flexibility offered by partition by growth and the almost continuous availability provided by CLONE tables. DB2 uses UTS for the implicit definition of XML table spaces1.

Keep in mind the restrictions mentioned in 5.1.1, “Using universal table spaces” on page 153. The major advantage, when migrating from traditional data partitioned to UTS PBR in DB2 9, is the faster mass DELETE. Keep using the classic partitioned table space for a more general optimal high volume insert workload which can benefit from MEMBER CLUSTER and its space search algorithm.

See Appendix A, “Summary of relevant maintenance” on page 291 for recent APARs on this function, including PK75149, PK83735, and PM15729.

---

1 See 3.2.1, “XML structure” on page 66.
5.2 Clone table

In DB2 9 for z/OS new-function mode, you can create a clone table on an existing base table. When you create a clone table, you also initiate the creation of multiple additional clone objects such as indexes, constraints, and triggers.

The clone table is created in the same table space as the base table and has the same structure as the base table. This includes, but is not limited to, column names, data types, null attributes, check constraints, and indexes. When ADD CLONE is used to create a clone of the specified base table, the base table must conform to the following rules:

- Reside in a DB2-managed universal table space
- Be the only table in the table space
- Must not be defined with a clone table
- Must not be involved in any referential constraint
- Must not be defined with any after triggers
- Must not be a materialized query table
- If the table space is created with the DEFINE NO clause, already have created data sets
- Must not have any pending changes
- Must not have any active versioning
- Must not have an incomplete definition
- Must not be a created global temporary table or a declared global temporary table

For more information about clone table, see Chapter 6, “Utilities” on page 199, and DB2 9 for z/OS Technical Overview, SG24-7330.

5.3 Object-level recovery

The backups taken by the BACKUP SYSTEM utility (referred to here as system-level backups) in DB2 V8 are used for recovery of the entire DB2 system. DB2 V9 has an enhancement that allows a subset of the data to be recovered from the system-level backups. Recovery is via the RECOVER utility, which is now capable of using the volume of system-level backups for the restore phase, in addition to the various image copies (full, in-line, concurrent, and incremental).

For more information about object-level recovery, see 6.4.2, “RECOVER utility enhancements for point-in-time recovery” on page 221, and DB2 9 for z/OS Technical Overview, SG24-7330.

5.4 Relief for sequential key insert

In this section, we describe the enhancements in DB2 9 for z/OS that help in reducing the hot spots for intensive sequential key insert applications.

5.4.1 Insert improvements with DB2 9 for z/OS

In this section, we discuss the major improvements that are now available for helping with the insert of intensive applications.

Asymmetric index page split
Prior to DB2 9 for z/OS, an index page can be done as a 100/0 split if keys are inserted into the end of an index or as a 50/50 split if keys are inserted in the middle of the key range.
Figure 5-1 shows a simplified representation of key inserts. When using the 50/50 split of an index page, you can experience frequent page splits that leave half of the splitting pages empty.

The effect of page splits after inserts of keys A5 and B5

Figure 5-1   Index split roughly 50/50
In DB2 9 for z/OS new-function mode, the insert pattern in an index is detected. Based on the insert pattern, DB2 9 for z/OS can split an index page by choosing from several algorithms. If an ever-increasing sequential insert pattern is detected for an index, DB2 splits index pages asymmetrically using approximately a 90/10 split. See Figure 5-2. With a 90/10 split, most existing keys stay on the original index page, thereby leaving more space on the new index page for future inserts.

![Asymmetric index page splits](image)

If an ever-decreasing sequential insert pattern is detected in an index, DB2 splits index pages asymmetrically using approximately 10/90 split. With the 10/90 split, most existing keys are moved to the new index page, thereby making room on the original index page for future inserts.

If a random insert pattern is detected in an index, DB2 splits index pages with a 50/50 ratio.

**Larger index page size**
Prior to DB2 9 for z/OS, you can only specify a 4 KB buffer pool on the CREATE INDEX statement for a particular index. In DB2 9 for z/OS new-function mode, you can specify 4 KB, 8 KB, 16 KB, or 32 KB buffer pools on the CREATE INDEX and ALTER INDEX statement.

A larger index page size can minimize the need for DB2 to split index pages, help performance for sequential processing, and better compression ratio.

**Index key randomization**
Index contention, especially on a hot page, can be a major problem in a data sharing environment and can limit scalability. DB2 9 for z/OS new-function mode introduces a randomized index key order. The randomized key order allows DB2 to spread out index keys within the whole index tree, instead of maintaining an ascending or descending order, thereby minimizing index page contention and turning a hot index page into a cool index page.

A randomized index key can reduce lock contention, but can increase the number of getpages, lock requests, and read and write I/Os. Using a randomized index key can produce a dramatic improvement or degradation of performance.
To use a randomized index key, you specify RANDOM after the column name on the CREATE INDEX command or specify RANDOM after the column name of the ALTER INDEX ADD COLUMN command.

The values of the HIGH2KEY and LOW2KEY columns in the SYSCOLSTATS and SYSCOLUMNS catalog tables are not updated by RUNSTATS if the columns are randomized-key columns.

### 5.4.2 Performance

The measurements are done by using a twelve-partition table. A partition by range table was created by using the DDL in Example 5-1.

**Example 5-1 Sample DDL for the partition-by-range table**

```sql
CREATE TABLE XYR1TAB (
   COL01 TIMESTAMP NOT NULL,
   COL02 INTEGER NOT NULL
      GENERATED ALWAYS AS IDENTITY
         (START WITH 1,
          INCREMENT BY 1,
          MINVALUE 1,
          NO MAXVALUE,
          NO CYCLE,
          CACHE 1000000, <- for non-DS
          CACHE 20,      <- for 2-way DS
          NO ORDER),
   COL03 VARCHAR(20) NOT NULL,
   COL04 TIME NOT NULL,
   COL05 DATE NOT NULL,
   COL06 CHAR(7) NOT NULL,
   COL07 CHAR(8) NOT NULL,
   COL08 INTEGER NOT NULL)
PARTITION BY RANGE (COL06 ASC)
   (PARTITION 1 ENDING AT('2007-01'),
    PARTITION 2 ENDING AT('2007-02'),
    PARTITION 3 ENDING AT('2007-03'),
    PARTITION 4 ENDING AT('2007-04'),
    PARTITION 5 ENDING AT('2007-05'),
    PARTITION 6 ENDING AT('2007-06'),
    PARTITION 7 ENDING AT('2007-07'),
    PARTITION 8 ENDING AT('2007-08'),
    PARTITION 9 ENDING AT('2007-09'),
    PARTITION 10 ENDING AT('2007-10'),
    PARTITION 11 ENDING AT('2007-11'),
    PARTITION 12 ENDING AT('2007-12'))
IN DBXYR.TSXYR1;
```
A non-clustered partitioning index was created with the DDL shown in Example 5-2. In the first test, column COL01 was defined by using ascending sort order, and in the second test, it was defined as RANDOM to be used as a randomized index key.

**Example 5-2  Sample DDL for a non-clustered partitioning index**

```sql
CREATE INDEX XYR1IDX2
  ON XYR1TAB(COL06 ASC,
           COLO1 ASC/RANDOM,
           COLO7 ASC)
  PARTITIONED
  USING STOGROUP XYR1STOG PRIQTY 180000 SECQTY 180000
    ERASE YES
    FREEPAGE 0
    PCTFREE 10
    GBPCACHE CHANGED
    DEFINE YES
    BUFFERPOOL BP2
    CLOSE NO
    DEFER NO
    COPY YES;
```

A non-partitioned index was created using the DDL shown in Example 5-3. In the first test column, COL01 was defined by using an ascending sort order, and in the second test, it was defined by using a RANDOM sort order to be used as a randomized index key.

**Example 5-3  Sample DDL for a non-partitioned index**

```sql
CREATE INDEX XYR1IDX3
  ON XYR1TAB(COL01 ASC/RANDOM,
           COLO7 ASC)
  CLUSTER
  USING STOGROUP XYR1STOG PRIQTY 180000 SECQTY 180000
    ERASE YES
    FREEPAGE 0
    PCTFREE 10
    GBPCACHE CHANGED
    DEFINE YES
    BUFFERPOOL BP3
    CLOSE NO
    DEFER NO
    PIECESIZE 2 G
    COPY YES;
```

Also, a data-partitioned secondary index (DPSI) was created using the DDL shown in Example 5-4.

**Example 5-4  Sample DDL for DPSI**

```sql
CREATE INDEX XYR1IDX4
  ON XYR1TAB(COL02 ASC/RANDOM)
  PARTITIONED
  USING STOGROUP XYR1STOG PRIQTY 180000 SECQTY 180000
    ERASE YES
    FREEPAGE 0
    PCTFREE 10
    GBPCACHE CHANGED
```
A low and a high value record were inserted into each partition. This was done to ensure that keys are inserted in the middle of the key range. The application performed 100 inserts per commit.

The tests with varying type of indexes were performed in a non-data sharing environment and in a data sharing environment.

**Non-data sharing**

We examine the impact of the enhancements on the index structures and the system and application performance in a non-data sharing environment.

**Non-clustered partitioning index**

Figure 5-3 summarizes the increase of entries per index leaf page in a non-data sharing environment during a sequential key insert in a non-clustered partitioning index using both ascending and random order together with an index page size of both 4 KB and 32 KB.

![Number of index entries/leaf page (XYR1IDX2) non-data sharing](image)

The measurements show that the asymmetric page split in DB2 V9 has increased the number of index entries per leaf page for the non-clustered partitioning index by 82.7% compared to DB2 V8. Changing the index page size from 4 KB to 32 KB gives approximately an increase of eight times in the number of entries per page. This increase is similar to the increase of the page size compared to an index page size of 4 KB.

Using a randomized index key order with a page size of 4 KB shows, in DB2 V9, an increase of the number of entries per index leaf page by 27.1% compared to DB2 V8. Changing the index page size from 4 KB to 32 KB gives approximately an increase of eight times in the number of entries per index leaf page compared to an index size of 4 KB, which is similar to the increase of the page size.
Figure 5-4 summarizes the reduction of the number of getpages per commit in a non-data sharing environment during sequential key insert in a non-clustered partitioning index using both ascending and random key order together with an index page size of both 4 KB and 32 KB.

![XYR1IDX2 (Non-clustered partitioning index) non-data sharing](image)

**Figure 5-4  Number of getpages and buffer updates for the non-clustered partitioning index**

The measurements show that, for a non-clustered partitioning index using an ascending order and an index page size of 4 KB, the number of getpages in DB2 V9 is reduced by 70.8% compared to DB2 V8. At the same time, the number of buffer updates are reduced by 4.2%. Changing the index page size from 4 KB to 32 KB in DB2 V9 reduced the number of getpages by 75.3% compared to DB2 V8. The number of buffer updates are reduced by 8.7%.

Using the randomized index key order in DB2 V9 shows nearly the same performance as an ascending index in DB2 V8 in terms of number of getpages and buffer updates. The number of getpages is reduced by 1.5%, and the number of buffer updates is reduced by 0.7%. Changing the index page size from 4 KB to 32 KB reduced the number of getpages by 29.5% and the number of buffer updates by 8.3% compared to DB2 V8.

**Clustered non-partitioned index**

Figure 5-5 summarizes the increase of entries per index leaf page in a non-data sharing environment during sequential key insert in a clustered non-partitioned index using both ascending and random order together with an index page size of both 4 KB and 32 KB.
The measurements show that the asymmetric page split in DB2 V9 has increased the number of index entries per leaf page for the clustered non-partitioned index by 89.3% compared to DB2 V8. Using an index page size of 32 KB gives approximately an increase of eight times in the number of entries per page, which is similar to the increase of the page size compared to an index page size of 4 KB.

Using a randomized index key order with a page size of 4 KB shows, in DB2 V9, an increase of the number of entries per index leaf page by 30.1% compared to DB2 V8. Using an index page size of 32 KB gives approximately an increase of eight times in the number of entries per index leaf page compared to an index size of 4 KB, which is similar to the increase of the page size.

Figure 5-6 summarizes the reduction of the number of getpages per commit in a non-data sharing environment. This example occurred during a sequential key insert in a clustered non-partitioned index using both ascending and random key order together with an index page size of both 4 KB and 32 KB.
The measurements show that, for a clustered non-partitioned index using an ascending order and an index page size of 4 KB, the number of getpages in DB2 V9 is reduced by 36.2% compared to DB2 V8. At the same time, the number of buffer updates is reduced by 3.6%. Using an index page size of 32 KB in DB2 V9 reduced the number of getpages by 82.9% compared to DB2 V8. The number of buffer updates is reduced by 7.2%.

Using the randomized index key order in DB2 V9 shows an increase of approximately 17 times in the number of getpages compared to DB2 V8. The number of buffer updates is reduced by 0.8% compared to DB2 V8. Using an index page size of 32 KB shows an increase of approximately 12 times in the number of getpages compared to DB2 V8. The number of buffer updates is reduced by 6.8% compared to DB2 V8.

Index look-aside in DB2 V9 helps a lot to reduce the number of getpages for a clustered index. For more information about index look-aside, see 4.18, “Index look-aside” on page 131.

**Data partitioned secondary index**

The results are similar to those explained in “Non-clustered partitioning index” on page 159, so they are not repeated here. The observations are also similar.

**System and application impact**

Figure 5-7 summarizes the changes of the number of getpages and the changes of buffer updates per commit at application level in a non-data sharing environment. This example occurred during a sequential key insert using both ascending and random key order together with an index page size of both 4 KB and 32 KB.

The measurements show that, at the application level, the total number of getpages for all three indexes is reduced by 66.6% in DB2 V9 when using a page size of 4 KB and an ascending key order compared to DB2 V8. At the same time, the number of buffer updates is reduced by 2.1%.

Changing the index page size from 4 KB to 32 KB using an ascending index key order in DB2 V9 reduced the number of getpages at the application level by 71.2% compared to DB2 V8. The number of buffer updates is reduced by 4.7%.

Using the randomized key order in DB2 V9 shows an increase in the total number of getpages at the application level by 50.4% compared to DB2 V8. The number of buffer updates is reduced by 0.2% compared to DB2 V8.
Changing the index page size from 4 KB to 32 KB using an randomized index key order in DB2 V9 increases the number of getpages at the application level by 17.9% compared to DB2 V8. The number of buffer updates is reduced by 4.4%.

Table 5-1 shows the DB2 class 2 CPU time, class 2 elapsed time, and class 3 suspension time for DB2 V9, using an index page size of both 4 KB and 32 KB and using an ascending index key order. A comparison is made with the V8 base case.

With 4 KB pages, using the ascending key in DB2 V9 reduced the class 2 CPU time by 16.7% and the class 2 elapsed time by 27.1%. With 32 KB pages, using the ascending key in DB2 V9 reduced the class 2 CPU time by 18% and the class 2 elapsed time by 13.6%.

Table 5-1  Class 2 CPU time DB2 V8 versus DB2 V9 - Ascending index key order

<table>
<thead>
<tr>
<th>DB2 V8</th>
<th>DB2 V9 4 KB, ascending</th>
<th>Delta %</th>
<th>DB2 V9 32 KB, ascending</th>
<th>Delta %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 2 CPU time (sec.)</td>
<td>Class 2 CPU time (sec.)</td>
<td>Delta %</td>
<td>Class 2 CPU time (sec.)</td>
<td>Delta %</td>
</tr>
<tr>
<td>6.090</td>
<td>5.074</td>
<td>- 16.7</td>
<td>4.992</td>
<td>- 18.0</td>
</tr>
<tr>
<td>Class 2 Elapsed time (sec.)</td>
<td>Class 2 Elapsed time (sec.)</td>
<td>Delta %</td>
<td>Class 2 Elapsed time (sec.)</td>
<td>Delta %</td>
</tr>
<tr>
<td>Class 3 Suspension time (sec.)</td>
<td>Class 3 Suspension time (sec.)</td>
<td>Delta %</td>
<td>Class 3 Suspension time (sec.)</td>
<td>Delta %</td>
</tr>
<tr>
<td>12.988</td>
<td>8.887</td>
<td>- 31.6</td>
<td>11.757</td>
<td>- 9.5</td>
</tr>
</tbody>
</table>

Table 5-2 shows the DB2 class 2 CPU time, class 2 elapsed time, and class 3 suspension time for DB2 V9, using an index page size of both 4 KB and 32 KB and using a randomized index key order. A comparison is made with the V8 base case.

With 4 KB pages, using the randomized index key order in DB2 V9 in a non-data sharing environment increases the class 2 CPU time by 10.2%, but it reduces the class 2 elapsed time by 13% compared to DB2 V8. With 32 KB pages, using the randomized index key order in DB2 V9 in a non-data sharing environment increases the class 2 CPU time by 4%, but it reduces the class 2 elapsed time by 18.9% compared to DB2 V8.

Table 5-2  Class 2 CPU time DB2 V8 versus DB2 V9 - Random index key order

<table>
<thead>
<tr>
<th>DB2 V8</th>
<th>DB2 V9 4 KB, random</th>
<th>Delta %</th>
<th>DB2 V9 32 KB, random</th>
<th>Delta %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 2 CPU time (sec.)</td>
<td>Class 2 CPU time (sec.)</td>
<td>Delta %</td>
<td>Class 2 CPU time (sec.)</td>
<td>Delta %</td>
</tr>
<tr>
<td>6.090</td>
<td>6.714</td>
<td>+ 10.2</td>
<td>6.374</td>
<td>+ 4.7</td>
</tr>
<tr>
<td>Class 2 Elapsed time (sec.)</td>
<td>Class 2 Elapsed time (sec.)</td>
<td>Delta %</td>
<td>Class 2 Elapsed time (sec.)</td>
<td>Delta %</td>
</tr>
<tr>
<td>19.587</td>
<td>17.036</td>
<td>- 13.0%</td>
<td>15.892</td>
<td>- 18.9%</td>
</tr>
<tr>
<td>Class 3 Suspension time (sec.)</td>
<td>Class 3 Suspension time (sec.)</td>
<td>Delta %</td>
<td>Class 3 Suspension time (sec.)</td>
<td>Delta %</td>
</tr>
<tr>
<td>12.988</td>
<td>8.084</td>
<td>- 37.8</td>
<td>7.431</td>
<td>- 42.8</td>
</tr>
</tbody>
</table>
Figure 5-8 shows the details of the components for the class 3 suspension time in this non-data sharing environment during a sequential key insert using both ascending and random key order together with index page sizes of both 4 KB and 32 KB.

Figure 5-8   Class 3 suspensions in a non-data sharing environment

In DB2 V9, the DB2 class 3 suspension time is reduced for an index size of 4 KB and ascending index key order by 31.6% compared to DB2 V8. When changing the index page size to 32 KB, the class 3 suspension time is reduced by 9.5% compared to DB2 V8.

Using the randomized index key order in DB2 V9, the class 3 suspension time is reduced for an index page size of 4 KB by 37.8% compared to DB2 V8. When changing the index page size to 32 KB then the class 3 suspension time is reduced by 42.8% compared to DB2 V8.

Data sharing
In this section, we show how the impact of the enhancements on the index structures and the system and application performance are especially beneficial in a data sharing environment.

Non-clustered partitioning index
Figure 5-9 summarizes the increase of entries per index leaf page in a two-way data sharing environment during sequential key insert in a non-clustered partitioning index using both ascending and random order together with an index page size of both 4 KB and 32 KB.
The measurements show that the asymmetric page split in DB2 V9 has increased the number of index entries per leaf page for the non-clustered partitioning index by 23.7% compared to DB2 V8. Changing the index page size to 32 KB gives an increase of approximately eight times in the number of entries per page, which is similar to the increase of the page size compared to an index page size of 4 KB.

Using a randomized index key order with a page size of 4 KB shows, in DB2 V9, an increase of the number of entries per index leaf page by 29.4% compared to DB2 V8. Using an index page size of 32 KB gives approximately an increase of eight times in the number of entries per index leaf page compared to an index size of 4 KB, which is similar to the increase of the page size.

Figure 5-10 summarizes the increase of entries per index leaf page in a two-way data sharing environment during a sequential key insert in a clustered non-partitioned index using both ascending and random order together with an index page size of both 4 KB and 32 KB.
DB2 V8. Using an index page size of 32 KB gives an increase of approximately eight times in
the number of entries per page, which is similar to the increase of the page size compared to
an index page size of 4 KB.

Using a randomized index key order with a page size of 4 KB shows, in DB2 V9, an increase
in the number of entries per index leaf page by 31.0% compared to DB2 V8. Changing the
index page size from 4 KB to 32 KB gives an increase of approximately eight times in the
number of entries per index leaf page compared to an index size of 4 KB, which is similar to
the increase of the page size.

**Data partitioned secondary index**

Figure 5-11 summarizes the changes of entries per index leaf page in a two-way data sharing
environment during sequential key insert in a DPSI using both ascending and random index
key order together with an index page size of both 4 KB and 32 KB.

The measurements show that the asymmetric page split in DB2 V9 has increased the number
of index entries per leaf page for the DPSI by 21.0% compared to DB2 V8. Using an index
page size of 32 KB gives an increase of approximately eight times in the number of entries
per page, which is similar to the increase of the page size compared to an index page size of
4 KB.

Using a randomized index key order with a page size of 4 KB shows that, in DB2 V9, the
number of entries per index leaf page is reduced by 18.1% compared to DB2 V8. Changing
the index page size from 4 KB to 32 KB gives an increase of approximately eight times in the
number of entries per index leaf page compared to DB2 V8.
System and application impact

Figure 5-12 summarizes latch class 6 contention in a two-way data sharing environment during sequential key insert using both ascending and randomized index key order and using an index page size of both 4 KB and 32 KB.

These measurements show that latch class 6 contention is increased by 19% for an ascending index key order using a page size of 4 KB compared to DB2 V8. Using a page size of 32 KB reduced latch class 6 contention by 83.1% compared to DB2 V8.

Using a randomized index key order with a page size of 4 KB reduced latch class 6 contention by 97.6% compared to DB2 V8. Using a page size of 32 KB reduced latch class 6 contention by 99.4% compared to DB2 V8.
Figure 5-13 summarizes the coupling facility CPU utilization in a two-way data sharing environment during sequential key insert using both ascending and random index key order and using an index page size of both 4 KB and 32 KB.

The coupling facility CPU utilization is reduced from 4.5% to 3.5% for the primary coupling facility. For the secondary coupling facility, it is reduced from 13.4% to 4.6% in DB2 V9 when using an ascending index key order and a page size of 4 KB. Using a page size of 32 KB reduces the coupling facility CPU utilization for the primary coupling facility to 3.0 and for the secondary coupling facility to 2.5%.

Using the randomized index key order, the coupling facility CPU utilization is increased for the primary coupling facility from 4.5% to 16.4%. For the secondary coupling facility, the CPU utilization is reduced from 13.4% to 12.8% when using a page size of 4 KB. When changing the page size from 4 KB to 32 KB, the CPU utilization in the primary coupling facility is increased to 30.0%, and for the secondary coupling facility, the CPU utilization is increased to 33.2%.

Figure 5-14 summarizes the class 2 CPU time in a two-way data sharing environment during sequential key insert using both ascending and randomized index key order and using an index page size of both 4 KB and 32 KB. The values are reported for the two members of the data sharing group.
Figure 5-14 Class 2 CPU time

The class 2 CPU time is reduced by 46.1% compared to DB2 V8, when using an ascending index key order and a page size of 4 KB. Changing the page size to 32 KB reduced the class 2 CPU time by 55.4%.

When using the randomized index key order and a page size of 4 KB, the class 2 CPU time is reduced by 0.6%, which is similar compared to DB2 V8. When using an index page size of 32 KB, the class 2 CPU time is increased by 30.4% compared to DB2 V8.

Figure 5-15 summarizes the class 2 elapsed time in a two-way data sharing environment during a sequential key insert using both ascending and randomized index key order and using an index page size of both 4 KB and 32 KB. The values are reported for the two members of the data sharing group.

Figure 5-15 Class 2 elapsed time in a two-way data sharing environment
The class 2 elapsed time is reduced by 29.3% compared to DB2 V8, when using an ascending index key order and a page size of 4 KB. Using an index page size of 32 KB reduced the class 2 CPU time by 39.8%.

Using the randomized index key order and a page size of 4 KB reduced the class 2 elapsed time by 39.1% compared to DB2 V8. When changing the page size to 32 KB, the class 2 elapsed time is reduced by 14.0% compared to DB2 V8.

Figure 5-16 shows the details of the components of the class 3 suspension time in the two-way data sharing environment during a sequential key insert using both ascending and randomized index key order and using an index page size of both 4 KB and 32 KB. The values are reported for the two members of the data sharing group.

![Class 3 suspensions time in a two-way data sharing environment](image)

The class 3 suspension time in a two-way data sharing environment is reduced by 28.5% when using an ascending index key order and a page size of 4 KB compared to DB2 V8. When using an index page size of 32 KB, the class 3 suspension time is reduced by 37.3%.

Using the randomized index key order and a page size of 4 KB reduced the class 3 suspension time by 40.1% compared to DB2 V8. When changing the page size to 32 KB, the class 3 suspension time is reduced by 10.0%.

Table 5-3 summarizes the insert rate per second in a two-way data sharing environment during sequential key insert using ascending index key order and using an index page size of both 4 KB and 32 KB.

| Table 5-3 Insert rate (ETR) in two-way data sharing - Ascending index key order |
|---------------------------------|---------------------------------|---------------------------------|
| Insert rate (ETR) per second   | Insert rate (ETR) per second   | Delta                           |
| Insert rate (ETR) per second   | Insert rate (ETR) per second   | Delta                           |
| DB2 V8                         | DB2 V9 4 KB ascending          | DB2 V9 32 KB ascending          |
| 1935.33                        | 2038.67                        | 5.3%                            |
| 2278.00                        | 17.7%                          |                                 |

Figure 5-16  Class 3 suspensions time in a two-way data sharing environment
The insert rate per second is increased in DB2 V9 by 5.3% compared to DB2 V8 when using an ascending index key order and a page size of 4 KB. When changing the page size to 32 KB, the insert rate per second is increased by 17.7% compared to DB2 V8.

Table 5-4 summarizes the insert rate per second in a two-way data sharing environment during a sequential key insert using randomized index key order and using an index page size of both 4 KB and 32 KB.

<table>
<thead>
<tr>
<th>DB2 V8</th>
<th>DB2 V9 4 KB random</th>
<th>DB2 V9 32 KB random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert rate (ETR) per second</td>
<td>Insert rate (ETR) per second</td>
<td>Delta</td>
</tr>
<tr>
<td>1935.33</td>
<td>2541.67</td>
<td>31.3%</td>
</tr>
</tbody>
</table>

When using the randomized index key order in DB2 V9, the insert rate per second is increased by 31.3% compared to DB2 V8. When changing the page size to 32 KB, the insert rate per second is similar to DB2 V8.

### 5.4.3 Conclusions

Depending on an insert pattern, an asymmetric index page split can reduce index page splits by up to 50% for an index page size of 4 KB, which reduces the number of pages that are used for a given index.

Using buffer pools that are larger than 4 KB to increase the page size to 8 KB, 16 KB, or 32 KB is good for heavy inserts and can reduce index page splits up to eight times.

Tests that run a massive insert in the middle of the key range, in a data sharing environment, show that the class 2 CPU time improvement can be up to 55%, and the class 2 elapsed time improvement can be up to 39%.

When running the same tests in a non-data sharing environment, the measurements show that an improvement of the class 2 CPU time can be up to 18% and the improvement of the class 2 elapsed time can be up to 27%.

For a non-clustered index, performance has been improved by index look-aside, which minimizes the need to traverse an index tree.

The new randomized index key order can solve problems with an index hot page and turn it into a cool page. In a data sharing environment, tests that use a randomized index key order with an index page size of 4 KB have shown that the class 2 CPU time is similar to using an ascending index key order in DB2 V8, and the class 2 elapsed time can be reduced by up to 39% compared to DB2 V8.

### 5.4.4 Recommendations

We recommend that you use a larger page size for indexes especially if you have a high latch class 6 contention in a data sharing environment. Each index page split requires two forced log writes in a data sharing environment. In data sharing, it can be a trade off between the page splits and potential higher index page physical lock (P-lock) contention.

You can use IFCID 0057 to determine whether you are experiencing any latch contention problems.
We recommend that you have randomized indexes in a large buffer pool with a high hit ratio to minimize the number of I/O.

Stay current with maintenance. Insert workloads are becoming more and more intensive and DB2 is adding more performance oriented APARs to DB2 9 for z/OS and Version 8 as well. Most current ones are:

- PK76738 for segmented table space
- PK83735 for UTS and segmented table spaces
- PM04272 for non UTS table spaces
- PM12935 for classic partitioned table spaces.

5.5 Index compression

In DB2 9 for z/OS new-function mode, you can compress your indexes. Unlike tablespace (data) compression, the mechanism that is implemented for index compression does not require a compression dictionary. Because DB2 performs dictionary-less compression, newly created indexes can begin compressing their contents immediately.

DB2 9 for z/OS only compresses the data in the leaf pages. The technique used to compress is based on eliminating repetitive strings (compaction) and is similar to what VSAM does with its indexes. Index pages are stored on disk in their compressed format (physical 4 KB index page on disk) and are expanded when read from disk into 8 KB, 16 KB, or 32 KB pages. In order to turn on compression for an index, the index must be defined in an 8 KB, 16 KB or 32 KB buffer pool, and the index must be enabled for compression by specifying COMPRESS YES. If this is done by using the ALTER INDEX command, then the change will not take effect before the index has been stopped and started again. If the index is partitioned, the COMPRESS clause always applies to all partitions.

**Restriction:** Indexes that use a 4 KB buffer pool cannot be compressed.

You might find value in long-term page fixing the buffer pools where uncompressed indexes reside, but it is not useful for the buffer pools of compressed indexes. DB2 does not perform I/O from the buffer pool of a compressed index, but from a 4 KB page I/O work area that is permanently page fixed; it is small compared to the size of the buffer pool. The pages in the work area are used for reading or writing to disk and do not act as a cache for the application.

For more information, see Index Compression with DB2 9 for z/OS, REDP-4345.

5.5.1 Performance

Table 5-5 summarizes a SELECT COUNT(*) statement using an index without compression and with an index page size of 4 KB compared with using an index with compression and an index page size of 16 KB.

**Table 5-5**  SELECT COUNT(*) of 100,000 rows using index scan

<table>
<thead>
<tr>
<th>All times in seconds</th>
<th>DB2 V9 - no index compression</th>
<th>DB2 V9 - index compression</th>
<th>Delta %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1 elapsed time</td>
<td>5.01</td>
<td>4.09</td>
<td>- 18.4</td>
</tr>
<tr>
<td>Class 1 CPU time</td>
<td>4.18</td>
<td>4.00</td>
<td>- 4.3</td>
</tr>
<tr>
<td>DBM1 service request block (SRB) time</td>
<td>1.18</td>
<td>2.28</td>
<td>+ 93.2</td>
</tr>
</tbody>
</table>
For this type of a select statement, the class 1 elapsed time is reduced by 18.4%, and the class 1 CPU time is reduced by 4.3% when compression is enabled.

The DBM1 address space is accounted for the CPU time that is used for decompression of index pages when they are read into the buffer pool. In this case, the CPU time for the DBM1 address space is increased by 93% compared to using an index without compression.

The total use of CPU time (class 1 CPU time + DBM1 SRB time) that is used without index compression is compared with the total use of CPU time that is used with compression. The total CPU time is increased by 17.2%.

Compression overhead due to conversion is minimal for indexes that mostly reside in the buffer pool and are frequently scanned and seldom updated. This is not the case for data compression, where scanning data causes conversion anyway for each row passed to the application program.

Table 5-6 summarizes the REBUILD INDEX utility that rebuilds an index without compression and uses an index page size of 4 KB and that rebuilds an index with compression and uses an index page size of 16 KB.

<table>
<thead>
<tr>
<th>All times in seconds</th>
<th>DB2 V9 - no index compression</th>
<th>DB2 V9 - index compression</th>
<th>Delta %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 3 susp. time</td>
<td>0.62</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>Total CPU time</td>
<td>4.18+1.18=5.36</td>
<td>4.00+2.28=6.28</td>
<td>+17.2</td>
</tr>
</tbody>
</table>

For rebuilding an index with compression compared to rebuilding an index without compression, the class 1 elapsed time is increased by 5.0%, and the class 1 CPU time is increased by 4.6%. The class 3 suspension time is increased by 18.2%.

The DBM1 address space is accounted for in the CPU time used for compressing index pages when they are built. In this case, the CPU time for the DBM1 address space is increased nearly five times when compared to rebuilding an index without compression.

The total use of CPU time (class 1 CPU time + DBM1 SRB time) used without index compression is compared with the total use of CPU time used with compression. The total CPU time is increased by 11.2%.
5.5.2 Conclusions

Enabling compression for large indexes, especially non-partitioned indexes, can save disk space. Measurements have shown a compression ratio from 25% to 75%. On average, you can expect a compression ration of 50%.

In one measurement, we observed that using index compression can increase the total use of CPU time (class 1 CPU time + DBM1 SRB time) by 18% for a SELECT statement and 11% for rebuilding an index.

5.5.3 Recommendation

We recommend that you run the DSN1COMP utility to simulate compression and calculate the compression ratio and the utilization of the various buffer pool page sizes, before you turn on compression for an index. The utilization of the buffer pool helps in deciding the page size to use for the index. Enabling index compression generally increases CPU time (attributed to both the application and the DBM1 address space) but saves disk space. However, if the index keeps staying in the buffer pool, there is no overhead while using it.

5.6 Log I/O enhancements

In DB2 9 for z/OS, the number of active log input buffers is increased from 15 to 120. In addition, archive log files are now read by Basic Sequential Access Method (BSAM) instead of Basic Direct Access Method (BDAM), so that the archive log files can use Data Facility Storage Management Subsystem (DFSMS) striping and compression.

The number of archive log files input buffers is increased from 1 to N. N is proportional to the number of stripes. For each stripe, DB2 uses 10 tracks worth of buffers, regardless of the block size.

5.6.1 Conclusion

An increase in the number of active log buffers to 120 improves performance during the fast log apply phase. Measurements shows that fast log apply throughput can increase up to 100% compared to DB2 V8. Fast log apply throughput is highest when the active log files are striped and most of the log records that are being read are inapplicable to the object or objects that are being recovered or reorganized.

When DB2 is reading archive log files, two BSAM buffers allow double buffering, but BSAM compression also allows for better I/O, especially if the archive log files are striped.

5.6.2 Recommendations

We recommend that you use stripes for both the active and archive log files to benefit for faster I/O during read, write, and offloading of active log files to archive log files.

5.7 Not logged table spaces

In DB2 9 for z/OS new-function mode, you can turn off logging for data that is recreated anyway, like some materialized query tables (MQTs), or temporarily during application
processes of your data, like heavy batch insert or update activity during year-end, for which backups are already taken. See Figure 5-17.

By not logging table spaces, you have the ability to increase the workload without impacting the amount of logging to be performed. After the batch processes are complete, you can turn on logging again and take an image copy of objects that are involved so they are recoverable.

The NOT LOGGED parameter that is specified with either the CREATE TABLESPACE or ALTER TABLESPACE statement applies to all tables that are created in the specified table space and to all indexes of those tables. For LOBs, this is not exactly true. See LOBs with DB2 for z/OS: Stronger and Faster, SG24-7270, for details.

Applying a NOT LOGGED logging attribute to a particular table space suppresses only the logging of undo and redo information. Control records with respect to that table space, such as open and closed page set log records, continue to be logged, regardless of the logging attribute. As a result, there will be a reduction in the amount of recovery log data that is generated, and there may be an increased speed of log reads for RECOVER and DSN1LOGP.

For more information about NOT LOGGED, see DB2 9 for z/OS Technical Overview, SG24-7330.

### 5.7.1 Performance

The measurements in Table 5-7 show the same workload using logging and NOT LOGGED for table spaces.

<table>
<thead>
<tr>
<th></th>
<th>Logged</th>
<th>NOT LOGGED</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed time - class 1</td>
<td>1:00:59</td>
<td>59:48</td>
<td>- 1.7%</td>
</tr>
<tr>
<td>Elapsed time - class 2</td>
<td>48:14</td>
<td>47:09</td>
<td>- 2.2%</td>
</tr>
<tr>
<td>CPU time - class 2</td>
<td>46:12</td>
<td>45:57</td>
<td>- 0.5%</td>
</tr>
<tr>
<td>MSTR CPU time</td>
<td>1:04</td>
<td>0:02</td>
<td>- 97%</td>
</tr>
<tr>
<td>Log write I/O</td>
<td>12.9</td>
<td>0.03</td>
<td>- 99%</td>
</tr>
<tr>
<td>Update commit</td>
<td>16.4</td>
<td>9.0</td>
<td>- 45%</td>
</tr>
</tbody>
</table>

Using the NOT LOGGED option can reduce up to 99% of the log write I/O. At the same time, it only reduces the class 2 elapsed time by 2.2%. The class 2 CPU time is reduced by 0.5%, which is similar to not to turning off logging.
5.7.2 Conclusion

Turning off logging for a particular table space suppresses only the logging of undo and redo information. There is nearly no improvement in terms of performance by turning off logging for this test case. However, if NOT LOGGED is used in a high log write I/O or high log latch contention environment, there can be a big elapse, CPU time reduction, or both.

The ability to avoid logging for a particular table space can help by reducing latch class 19 contention.

5.7.3 Recommendations

We recommended that you turn off logging only if you have a contention problem with the amount of log records that are written for a particular table space or when the recovery of a particular table space is not important. The data in that table space can be easily recreated.

5.8 Prefetch and preformatting enhancements

DB2 9 for z/OS conversion mode uses dynamic prefetch for index scans instead of using sequential prefetch. Dynamic prefetch is more intelligent and more robust.

**PREFETCH column:** The PREFETCH column of the PLAN_TABLE still shows the value of “S” for sequential prefetch, even though DB2 uses dynamic prefetch for index scans.

Sequential prefetch is still used for table space scans and has been enhanced in DB2 9 for z/OS.

Larger prefetch quantity and deferred write quantity are used in V9 for large buffer pools, which are defined as follows:

- For sequential prefetch, if VPSEQT*VPSIZE > 160 MB for SQL, 320 MB for utility
- For deferred write, if VPSIZE > 160 MB for SQL, 320 MB for utility

The maximum prefetch quantity goes from 128 KB (with V8) to 256 KB (with V9) in SQL for a table space scan. It goes from 256 KB (with V8) to 512 KB (with V9) in a utility.

DB2 9 for z/OS conversion mode can now preformat up to 16 cylinders if the allocation sizes are large enough. The change is from preformatting two cylinders at a time in DB2 V8 to as many as 16 cylinders at a time if the allocation quantity is at least 16 cylinders.

5.8.1 Performance

Dynamic prefetch uses two SRB engines, which triggers more pre-staging in the disk control unit. If the index is cache resident and the index scan is I/O bound, then the throughput can increase up to 100% compared to DB2 V8 by using sequential prefetch. If the index is not cache resident and the index scan is I/O bound, the throughput can increase up to 20% when using a DS8000 disk unit.

Figure 5-18 summarizes the increase in throughput of sequential prefetch when performing a table space scan and reading data from both disk and cache.
When reading data from disk, by using sequential prefetch, the throughput is increased by 9.8% for a page size of 4 KB compared to DB2 V8. For a page size of 8 KB, the throughput is increased by 9.5%. For a page size of 16 KB, the throughput is increased by 11.4%, and for a page size of 32 KB, the throughput is increased by 5.6% compared to DB2 V8.

Reading data from cache, by using sequential prefetch, the throughput is increased by 22.0% for a page size of 4 KB compared to DB2 V8. For a page size of 8 KB, the throughput is increased by 24.8%. For a page size of 16 KB, the throughput is increased by 30.6%, and for a page size of 32 KB, the throughput is increased by 30.9% compared to DB2 V8.

Figure 5-19 shows the increase of throughput due to the increase of the size of preformatting.

For the 4 KB control interval (CI) size, the throughput is increased by 47.2% compared to DB2 V8. For the 32 KB CI size, the throughput is increased by 45.7%.
5.8.2 Conclusion

Changing from sequential prefetch to use dynamic prefetch for index scans can reduce the elapsed time by up to 50%.

Using a larger preformatting quantity can reduce the elapsed time by up to 47% for a heavy insert application.

When reading a table space by using sequential prefetch, the throughput rate can increase by up to 11.4% when reading data from DASD. Also, when reading data from cache, the throughput rate can be increased by up to 30.9%.

5.9 WORKFILE database enhancements

DB2 V8 supports two databases for temporary files. The WORKFILE database is used for sort work files and created global temporary tables. The TEMP database is used for declared global temporary tables and static scrollable cursors.

In DB2 9 for z/OS conversion mode, the WORKFILE database and the TEMP database is converged into the WORKFILE database. The WORKFILE database has been optimized to select the best page size when using a workfile table space. If the workfile record length is less than 100 bytes, a table space with a page size of 4 KB is used. Otherwise, a table space with a page size of 32 KB is used. Work file access is often sequential. Therefore, using a larger page size can be more efficient. DB2 9 for z/OS tries to use a 32 KB buffer pool for larger sort record sizes to gain improved performance. Using a larger page size reduces the number of I/O.

Small sorts are now using an in-memory work file to avoid the cost of acquiring and freeing work files. The in-memory work file is used if the data fits into one 4 KB or 32 KB page.

The WORKFILE database consists of workfile table spaces that are now segmented.

See Appendix A.1, “Performance enhancements APARs” on page 292 for recent maintenance on this function.

Prior to DB2 9 for z/OS, it was not possible to control how much temporary work space was used by an individual agent. Large sorts, materialized views, and so on can monopolize the defined temporary space at the expense of other work and cause other work to fail due to unavailable resources.

With DB2 9 for z/OS, it is now possible to control temporary space utilization at the agent level. A new DSNZPARM, MAXTEMPS, is added to DSN6SYSP to specify the maximum amount of space that can be used by a single agent at any single time. You can specify this parameter in either MB or GB. If you specify a value of 0 (the default), then no limit is enforced. If any given agent exceeds MAXTEMPS, a resource unavailable condition is raised, and the agent is terminated. For more information about MAXTEMPS, see DB2 Version 9.1 for z/OS Installation Guide, GC18-9846.

5.9.1 Performance

To understand the performance of the WORKFILE database, we tested the following configuration:

- z/OS 1.7
Using this configuration, we ran the following workloads:

- **Workload for heavy sort activities**
  - SQL of various length of sort records, a selection of 1.5 million rows with and without ORDER BY to obtain the sort cost
  - Workfile buffer pool size:
    - DB2 V8: 4 KB = 4800
    - DB2 V9: 4 KB = 4800 and 32 KB = 600, 1200, 2400, 4800, 9600
  - Five million rows in a table with one index

- **Workload for declared global temporary table measurements**
  - Various lengths of rows for INSERT of 1.5 million rows into a declared global temporary table
  - SELECT COUNT(*) from a declared global temporary table with 1.5 million rows
  - Workfile buffer pool size:
    - V8: 4 KB = 4800
    - V9: 4 KB = 4800 and 32 KB = 600, 1200, 2400, 4800

Figure 5-20 summarizes the elapsed time for sort performance for an SQL statement with heavy sort activities with different record sizes and different sizes of the 32 KB buffer pool.

Lab measurements showed that for DB2 V9 sorting 1.5 million rows with a workfile record size less than 105 bytes had less than 5% elapsed time difference compared to DB2 V8. When sorting 1.5 million rows with a workfile record size greater than 105 bytes, then the 32K work file is used and the elapsed time can be reduced by up to 50% compared to DB2 V8. The size of the 32 KB buffer pool is important for elapsed time reduction. When the record size increases beyond the 105 bytes, increasing the number of 32 KB workfile buffers improves V9 elapsed time.

Figure 5-21 summarizes the CPU time for sort performance for an SQL statement with heavy sort activities with different record sizes and different sizes of the 32 KB buffer pool. Our lab measurements showed that, for DB2 V9, the CPU time for sorting 1.5 million rows with a workfile record size that is less than 105 bytes had less than a 10% difference compared to DB2 V8.

---

**Figure 5-20  Elapsed time for SQL with heavy sort activities - sort cost**

**Figure 5-21  CPU time for SQL with heavy sort activities - sort cost**
DB2 V8. When sorting 1.5 million rows with a workfile record size that is greater than 105 bytes, the 32 KB work file is used and the CPU time is reduced by up to 20% compared to DB2 V8. The lab measurements also showed that, compared to DB2 V8, the CPU time for sorting 1.5 million rows in DB2 V9 can increase by up to 7% if only 4 KB work files are defined. We do not recommend this configuration.

Figure 5-21  CPU time for SQL with heavy sort activities - sort cost

Figure 5-22 summarizes the reduction of elapsed time during insert into a declared global temporary table using different record sizes and different sizes of the 32 KB buffer pool.

Figure 5-22  Insert records into a declared global temporary table

A declared global temporary table uses a TEMP database in V8 and uses workfile database in V9. Lab measurements showed that, for DB2 V9, inserting 1.5 million rows into a declared
global temporary table with a workfile record size that is less than 100 bytes had up to 10% elapsed time reduction compared to DB2 V8. When inserting 1.5 million rows with a workfile record size that is greater than 100 bytes, the 32K work file is used, and the elapsed time is reduced by up to 17% compared to DB2 V8.

By increasing the 32 KB workfile buffer pool, you can improve V9 elapsed time on large insert records (greater than 100 bytes) in a declared global temporary table.

Figure 5-23 summarizes the reduction of elapsed time during a SELECT COUNT(*) statement of 1.5 million rows in a declared global temporary table by using different record sizes and different sizes of the 32 KB buffer pool.

![Figure 5-23 SELECT COUNT(*) from a declared global temporary table](image)

Lab measurements showed that, for DB2 V9 with APAR PK43765, SELECT COUNT(*) from a 1.5 million row table with a workfile record size less than 100 bytes had up to 35% elapsed time reduction compared to DB2 V8. When SELECT COUNT(*) from a 1.5 million row table with a workfile record size greater than 100 bytes, the 32K work file is used, and as a result, the elapsed time can be reduced by up to two times compared to DB2 V8.

By increasing the 32 KB workfile buffer pool, you can improve the elapsed time of DB2 V9 on SELECT COUNT(*) with large records (greater than 100 bytes) in a declared global temporary table.

In DB2 V9, the elapsed time of SELECT COUNT(*) of 1.5 million rows in a declared global temporary table with a record size of 50 bytes is reduced by up to 39.6% compared to DB2 V8.

5.9.2 Conclusions

The improvement of using 32 KB work files results in less workfile space that is needed and faster I/O, even when there is no MIDAW on the channel.
The size of the 32 KB buffer pool can impact performance and the amount of CPU time and elapsed time that can be reduced. Generally, the bigger the buffer pool is, the more the CPU and elapsed time can be reduced. If the 32 KB buffer pool becomes too big, performance regression can occur. The optimal size of the buffer pool depends on the workload.

Using an in-memory work file is beneficial for online transactions that have relatively short running SQL statements in which the number of rows that are sorted is small and can fit on one page, either 4 KB or 32 KB.

The elapsed time can be reduced by up to 50% for SQL statements with heavy sort activities due to the use of 32 KB work files.

CPU time can increase by up to 7% compared to DB2 V8, if DB2 V9 has only 4 KB work files that are allocated. We do not recommend this.

5.9.3 Recommendations

We recommend that you check the number of 32 KB work files. You may need to increase the number of 32 KB work files due to the increase in using 32 KB work files in DB2 9 for z/OS. You may also need to increase the size of your buffer pool used for 32 KB work files.

The buffer pool activity for a 4 KB work file should be significantly less than in DB2 V8. Therefore, consider decreasing the number of 4 KB work files and the size of the 4 KB buffer pool as well.

See the updated workfile instrumentation in 4.16, “WORKFILE and TEMP database merge” on page 125.

5.10 LOB performance enhancements

In DB2 V6 through DB2 V8, DB2 uses two types of locks to ensure the integrity of an LOB: the S-LOB and the X-LOB locks. These locks are similar to the common S-locks and X-locks. There are no U-LOB locks, because of a different update mechanism that takes place for LOBs. Updating an LOB for DB2 means the deallocation of used data pages, and allocating and inserting new data pages that contain the new LOB value.

In DB2 V8, an S-LOB lock is acquired and freed in each FETCH (both non-unit of recovery (UR) and UR readers) and DELETE call. That is, an S-LOB lock is not held until commit. An X-LOB lock that is acquired in INSERT or space allocation is held until commit. See Table 5-8 for changes in LOB locking behavior in DB2 V9.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT and UPDATE</td>
<td>Continues to hold a lock on the base row or page as in V8. However, for inserting or updating the LOB value, an X-LOB lock that is taken is held only until the duration of the insert or update (that is, not until commit).</td>
</tr>
<tr>
<td>DELETE</td>
<td>Continues to hold a lock on the base row or page as in V8. However, for deleting the LOB value, no S-LOB lock nor X-LOB lock is taken.</td>
</tr>
<tr>
<td>SELECT</td>
<td>Continues to hold any locks on the base row or page as in V8. However, no S-LOB lock is taken while building the data descriptor on the LOB data.</td>
</tr>
</tbody>
</table>
The processing of LOBs in a distributed environment with Java Universal Driver on the client side has been optimized for the retrieval of larger amounts of data. This new dynamic data format is available only for the Java Common Connectivity (JCC) T4 driver (Type 4 Connectivity). The call-level interface (CLI) of DB2 for Linux, UNIX, and Windows also has this client-side optimization. Many applications effectively use locators to retrieve LOB data regardless of the size of the data that is being retrieved. This mechanism incurs a separate network flow to get the length of the data to be returned, so that the requester can determine the proper offset and length for SUBSTR operations on the data to avoid any unnecessary blank padding of the value. For small LOB data, returning the LOB value directly instead of using a locator is more efficient. The overhead of the underlying LOB mechanisms can tend to overshadow the resources that are required to achieve the data retrieval.

For these reasons, LOB (and XML) data retrieval in DB2 V9 has been enhanced so that it is more effective for small and medium size objects. It is also still efficient in its use of locators to retrieve large amounts of data. This functionality is known as **progressive streaming**.

For more information about LOBs, see *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270.

### 5.10.1 Performance

We tested the following configuration:

- System z9 LPAR with three CPUs (one zIIP or one zAAP)
- ES800
- z/OS 1.7
- JCC V9 build 3.3.17

Figure 5-24 summarizes the improvement of the class 2 elapsed time, class 2 CPU time, and the class 3 wait time in DB2 V9 when inserting 24,000 LOBs with a size 100 KB each. DB2 V9 uses the new dynamic data format (progressive locator) as a default, and DB2 V8 uses a materialized LOB as a default.
In DB2 V9, the class 2 elapsed time is reduced by 22% when performing an insert of 24,000 LOBs with a size of 100 KB compared to DB2 V8. At the same time, the class 2 CPU time is reduced by 19%, and the class 3 wait time is reduced by 22% compared to DB2 V8.

Figure 5-25 summarizes the improvement of the class 1 elapsed time and class 1 CPU time in DB2 V9 when inserting 24,000 LOBs with a size 100 KB each. DB2 V9 uses a progressive locator, which is the default. DB2 V8 uses a materialized LOB, which is the default.

In DB2 V9, the class 1 elapsed time is reduced by 16% when performing an insert of 24,000 LOBs with a size of 100 KB compared to DB2 V8. At the same time, the class 1 CPU time is reduced by 22%, and the class 1 CPU time that is offloaded to zIIP is reduced by 22% compared to DB2 V8.
Figure 5-26 summarizes the performance improvement of selecting 1,000 LOBs with a size of 100 KB in DB2 V9 compared to DB2 V8. DB2 V9 uses a progressive locator, and DB2 V8 uses a materialized LOB.

![Chart: Select 1000 LOBs of size 100 KB](image)

*Figure 5-26  LOB select performance class 1 and 2 elapsed times*

In DB2 V9, the class 1 elapsed time is reduced by 38%, and the class 2 elapsed time is reduced by 40.6%, compared to DB2 V8 when selecting 1,000 rows with 100 KB of LOBs.

Table 5-9 summarizes the performance improvement of the class 1 and class 2 CPU time when selecting 1,000 LOBs with a size of 100 KB in DB2 V9 compared to DB2 V8.

*Table 5-9  LOB select performance class 1 and class 2 CPU times in seconds*

<table>
<thead>
<tr>
<th></th>
<th>DB2 V8</th>
<th>DB2 V9</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1 CPU time</td>
<td>0.038</td>
<td>0.0097</td>
<td>- 74.5%</td>
</tr>
<tr>
<td>Class 1 zIIP CPU time</td>
<td>0.046</td>
<td>0.0098</td>
<td>- 78.7%</td>
</tr>
<tr>
<td>Class 2 CPU time</td>
<td>0.014</td>
<td>0.0093</td>
<td>- 33.6%</td>
</tr>
</tbody>
</table>

In DB2 V9, the class 1 CPU time is reduced by 74.5%, and the class 1 zIIP CPU time is reduced by 78.7%, compared to DB2 V8 when selecting 1,000 rows with 100 KB of LOBs.

Figure 5-27 summarizes the performance difference when retrieving 1,000 CLOBs using an old locator, materialized LOB, and progressive locator with sizes of the CLOB of 1 KB, 40 KB, and 80 KB. The progressive locator processes the 1 KB LOB and sends it in-line in a query result. For the 40 KB LOB, the progressive locator sends it as chained to a query result. The progressive locator processes the 80 KB LOB via locator since streamBufferSize is set to 70000.
Retrieving 1,000 CLOBs with a size of 1 KB using materialized LOB is 33.5% faster in DB2 V9 compared to using the old locator for retrieving CLOBs. Using a progressive locator improves performance by 48.8% compared to using a materialized LOB. When comparing a progressive locator to the old locator, performance increased by 65.3% for CLOBs with a size of 1 KB.

Retrieving 1,000 CLOBs with a size of 40 KB using a materialized LOB is 39.6% faster in DB2 V9 compared to using the old locator for retrieving CLOBs. Using a progressive locator improved performance by 47.7% compared to materialized LOB. When comparing a progressive locator to the old locator, performance has increased by 68.4% for CLOBs with a size of 40 KB.

When retrieving 1,000 CLOBs with a size of 80 KB, the progressive locator is 1% slower than using the old locator in DB2 V9.

Table 5-10 summarizes the class 1 elapsed time for retrieving 1,000 rows with a 20 KB CLOB versus retrieving 1,000 rows with a 20 KB varchar column.

<table>
<thead>
<tr>
<th>Time in seconds</th>
<th>CLOB</th>
<th>Varchar</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old locator</td>
<td>12.629023</td>
<td>6.464362</td>
<td>+ 95.4%</td>
</tr>
<tr>
<td>Materialized LOB</td>
<td>7.357661</td>
<td>6.548800</td>
<td>+ 12.4%</td>
</tr>
<tr>
<td>Progressive locator</td>
<td>4.676349</td>
<td>6.509943</td>
<td>- 28.2%</td>
</tr>
</tbody>
</table>

When retrieving 1,000 rows with a 20 KB CLOB using the old locator, the class 1 elapsed time nearly doubles compared to retrieving 1,000 rows with a 20 KB varchar. When using a materialized LOB to retrieve 1,000 rows with CLOBs, the class 1 elapsed time is 12.4%
higher compared to retrieving 1,000 rows with varchar. When using a progressive locator to retrieve 1,000 rows with CLOBs, the class 1 elapsed time is 28.2% faster compared to retrieving 1,000 rows with varchar.

5.10.2 Conclusion

In DB2 V9, when inserting LOBs, you benefit from the increase of the preformatting quantity up to 16 cylinders and from the ability of DB2 V9 to trigger preformatting earlier than in DB2 V8. The larger the size of the LOB is, the more the insert will benefit from these changes in DB2 V9.

When you insert LOBs from a distributed environment, you also benefit from the use of shared memory between the distributed data facility (DDF) and DBM1 address space as well as progressive streaming. For more information about DDF using shared memory, see 4.7, “Distributed 64-bit DDF” on page 104.

Using the progressive locator, which is the default in DB2 V9, can significantly improve the class 1 elapsed time performance due to the reduction of network round trips.

For an object size between 12 KB and 32 KB, the application performance is improved in DB2 V9 when storing data in LOBs, compared to storing the data in a VARCHAR, especially if row buffering is used and multiple rows are retrieved.

5.10.3 Recommendations

In a data sharing environment, now there is a need to ensure that changed pages are forced out before the X-LOB lock is released. We recommend that you use GBPCACHE(CHANGED) for better performance.

Note: Recent maintenance on LOBS:
- PTF UK41212 for APAR PK65220 adds LOBs support to the APPEND YES option.
- PTF UK43355 for APAR PK75216 enhances the performance for the LOAD and UNLOAD of LOBs using file reference variables with PDS and HFS/zFS by improving the process for the open and close of data sets.

5.11 Spatial support

IBM Spatial Support for DB2 for z/OS provides a set of spatial data types that you can use to model real-world entities, such as the locations of customers, the boundaries of parks, and the path of cable lines. You can manipulate spatial data by using spatial functions, which you can invoke from within an SQL statement. Also, you can create indexes on spatial data, which can be used by DB2 to optimize spatial query performance.

When you enable a DB2 subsystem for spatial operations, IBM Spatial Support for DB2 for z/OS provides the database with seven distinct geometry types. Six of these geometry types are instantiated, and one is abstract.

Data types ST_Point, ST_LineString, and ST_Polygon are used to store coordinates that define the space that is occupied by features that can be perceived as forming a single unit.

- Use ST_Point when you want to indicate the point in space that is occupied by a discrete geographic feature. The feature might be a small one, such as a water well; a large one,
such as a city; or one of intermediate size, such as a building complex or park. In each case, the point in space can be located at the intersection of an east-west coordinate line (for example, a parallel) and a north-south coordinate line (for example, a meridian). An ST_Point data item includes an X coordinate and a Y coordinate that define such an intersection. The X coordinate indicates where the intersection lies on the east-west line; the Y coordinate indicates where the intersection lies on the north-south line. The data type is VARBINARY.

- Use **ST_Linestring** for coordinates that define the space that is occupied by linear features, for example streets, canals, and pipelines. The data type is BLOB.

- Use **ST_Polygon** when you want to indicate the extent of space that is covered by a multi-sided feature, for example a voting district, a forest, or a wildlife habitat. An ST_Polygon data item consists of the coordinates that define the boundary of such a feature. The data type is BLOB.

In some cases, ST_Polygon and ST_Point can be used for the same feature. For example, suppose that you need spatial information about an apartment complex. If you want to represent the point in space where each building in the complex is located, you use ST_Point to store the X and Y coordinates that define each such point. Otherwise, if you want to represent the area that is occupied by the complex as a whole, you use ST_Polygon to store the coordinates that define the boundary of this area.

Data types ST_MultiPoint, ST_MultiLineString, and ST_MultiPolygon are used to store coordinates that define spaces that are occupied by features that are made up of multiple units.

- Use **ST_MultiPoint** when you are representing features that are made up of units whose locations are each referenced by an X coordinate and a Y coordinate. For example, consider a table whose rows represent island chains. The X coordinate and Y coordinate for each island has been identified. If you want the table to include these coordinates and the coordinates for each chain as a whole, define an ST_MultiPoint column to hold these coordinates. The data type is BLOB.

- Use **ST_MultiLineString** when you are representing features that are made up of linear units, and you want to store the coordinates for the locations of these units and the location of each feature as a whole. For example, consider a table whose rows represent river systems. If you want the table to include coordinates for the locations of the systems and their components, define an ST_MultiLineString column to hold these coordinates. The data type is BLOB.

- Use **ST_MultiPolygon** when you are representing features that are made up of multi-sided units, and you want to store the coordinates for the locations of these units and the location of each feature as a whole. For example, consider a table whose rows represent rural counties and the farms in each county. If you want the table to include coordinates for the locations of the counties and farms, define an ST_MultiPolygon column to hold these coordinates. Data type is BLOB.

A multi-unit is not meant as a collection of individual entities. Rather, it refers to an aggregate of the parts that makes up the whole.

The data type that is abstract, or not instantiated, is ST_Geometry. You can use ST_Geometry when you are not sure which of the other data types to use. An ST_Geometry column can contain the same kinds of data items that columns of the other data types can contain.

When creating a new table or adding a spatial data column to an existing table using a spatial data type based on LOBs, all considerations and recommendations about LOBs apply. See LOBs with DB2 for z/OS: Stronger and Faster, SG24-7270, for details.
Before you query spatial columns, you can create indexes and views that facilitate access to them. Good query performance is related to having efficient indexes defined on the columns of the base tables in a database. Creating an index on a spatial column can dramatically improve performance as indexes can for other data types.

Spatial queries use a type of index called a spatial grid index. The indexing technology in IBM Spatial Support for DB2 for z/OS uses grid indexing, which is designed to index multidimensional spatial data, to index spatial columns. IBM Spatial Support for DB2 for z/OS provides a grid index that is optimized for two-dimensional data on a flat projection of the Earth. A grid index is optimized for two dimensional data. The index is created on the X and Y dimensions of a geometry using the minimum bounding rectangle (MBR) of a geometry.

A spatial grid index divides a region into logical square grids with a fixed size that you specify when you create an index. The spatial index is constructed on a spatial column by making one or more entries for the intersections of each geometry's MBR with the grid cells.

An index entry consists of the grid cell identifier, the geometry MBR, and the internal identifier of the row that contains the geometry. You can define up to three spatial index levels (grid levels). Using several grid levels is beneficial because it allows you to optimize the index for different sizes of spatial data.

For more information about spatial support, see IBM Spatial Support for DB2 for z/OS User’s Guide and Reference, GC19-1145.

### 5.12 Package performance

When migrating from one release of DB2 to another release of DB2, the CPU time for executing plans and packages may increase. When migrating to DB2 V9, the CPU time for executing a plan or package increases due to the addition of new functions.

#### 5.12.1 Performance tests

We tested the following configuration:

- One plan with 101 packages. One package does CONNECT/DISCONNECT through call attach. 100 packages execute only one very short, simple SQL statement.
- One single thread that is run with 500 plan invocations (50,000 package/statements)

Figure 5-28 compares the average class 2 CPU time of the plan for DB2 V7, DB2 V8, and DB2 V9. The average class 2 CPU time of the 100 packages together is also compared for DB2 V7, DB2 V8, and DB2 V9.

The chart is just to demonstrate how a very short running package is affected on each of V7, V8, and V9 when bound on the same or a prior release. V9 has a slight increase over previous releases.
The measurements show that the CPU time increased by 33% in DB2 V8 compared to DB2 V7. In DB2 V9, the CPU time increased by 12% compared to DB2 V8.

### 5.12.2 Conclusion

In DB2 V9, executing packages with a single or a few short running SQL statements shows an increase of CPU time when compared to DB2 V8. The increase of CPU time for executing a plan or a package when migrating to DB2 V9 is less than the increase of CPU time for executing a plan or a package when migrating to DB2 V8. The slight increase can be attributed to some extra allocation work due to the EDM structures of the packages being split above and below the bar with DB2 V9. This increase becomes negligible with more complex statements in the package.
5.13 Optimistic locking

*Optimistic concurrency control* is a concurrency control method that is used in relational databases that does not use locking. This method is commonly referred to as *optimistic locking*. Optimistic locking, which is available in DB2 V9 new-function mode, is faster and more scalable than database locking for concurrent data access. It minimizes the time during which a given resource is unavailable for use by other transactions. When an application uses optimistic locking, locks are obtained immediately before a read operation and are released immediately after the operation. Update locks are obtained immediately before an update operation and held until the end of the transaction. See Figure 5-29.

![Figure 5-29 Positioned updates and deletes with optimistic concurrency control](image)

Optimistic locking uses the RID and a row change token to test whether data has been changed by another transaction since the last read operation. DB2 can determine when a row was changed. It can ensure data integrity while limiting the time that locks are held.

To safely implement optimistic concurrency control, you must establish a row change timestamp column with a CREATE TABLE statement or an ALTER TABLE statement. The column must be defined as NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP or NOT NULL GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP. After you establish a row change timestamp column, DB2 maintains the contents of this column. When you want to use this change token as a condition when making an update, you can specify an appropriate condition for this column in your WHERE clause.
5.13.1 Performance

Figure 5-30 summarizes the class 2 CPU time for inserting 1,000,000 rows into three different tables. The first table is defined with a ROW CHANGE TIMESTAMP column to use optimistic locking. The second table is defined with a TIMESTAMP column as NOT NULL WITH DEFAULT. The third table is defined with a TIMESTAMP column as NOT NULL, and the timestamp is assigned from a host variable. The bars in Figure 5-30 follow the same sequence.

![Figure 5-30](image)

The measurements show that the class 2 CPU time for using ROW CHANGE TIMESTAMP, the generated TIMESTAMP used for concurrency control, is 6.6% faster than using a TIMESTAMP column that is defined with NOT NULL WITH DEFAULT. It also shows that the class 2 CPU time for using ROW CHANGE TIMESTAMP is 11.6% slower than using a TIMESTAMP column and assigning the value from a host variable.

5.13.2 Conclusion

We expected that DB2-assisted ROW CHANGE TIMESTAMP generation would cost slightly more than inserting the predetermined TIMESTAMP, but would be more efficient than calling CURRENT TIMESTAMP to assign a timestamp. Our measurement results have validated the assumption.

5.14 Package stability

New options for REBIND PACKAGE and REBIND TRIGGER PACKAGE allow to preserve multiple package copies and allow users to switch back to a previous copy of the bound package. This function, a first step towards a general access plan stability, has been made available through the maintenance stream after general availability of DB2 9 for z/OS. The relevant APAR is PK52523 (PTF UK31993). The support is for PACKAGE, not PLAN, and includes non-native SQL procedures, external procedures, and trigger packages.

This function will help in those situations where a static SQL REBIND causes unstable performance of queries due to changes in access paths. These situations are typically cases of regression when migrating to a new release, or when applications are bound/rebound after applied maintenance, or changes in data distribution and schema, or changes to the
application. See 9.3.6, “To rebind or not to rebind” on page 267 for recommendations using package stability.

For the known critical transactions, users have had techniques like access path hints to revert to old access paths, however the procedure is not simple and identifying the source of such regressions can take time. package stability provides a fast and relative inexpensive way of reverting to a previous ‘good’ access path improving availability and allowing time for problem determination.

At REBIND PACKAGE, the old copies of the package are saved in the DB2 directory and extra rows per copy are inserted in the catalog tables:

- SYSPACKAGE
- SYSPACKSTMT

A row in SYSPACKDEP reflects the currently active copy.

The new REBIND PACKAGE option is called PLANMGMT and it can be used to control whether and how REBIND PACKAGE saves old package copies. Another REBIND option is called SWITCH and is used to control the switching back to a saved PACKAGE copy.


### 5.14.1 Controlling the new PLANMGMT option for REBIND PACKAGE

The option can be controlled at two levels:

- **Subsystem level via a new DSNZPARM (PLANMGMT)**
- **BIND level with new options for REBIND**

#### Subsystem level

A new system parameter called PLANMGMT for specifying the default setting of PLANMGMT option of REBIND PACKAGE.

Possible settings are: OFF, BASIC and EXTENDED. The default value of this parameter is OFF. To use a setting other than OFF, update your DB2 9 subsystem parameter (DSNZPxxx) modules as follows:

- Edit your customized copy of installation job DSNTIJUZ
- Add the keyword parameter PLANMGMT=<x> -- where <x> is BASIC, EXTENDED, or OFF -- to the invocation of the DSN6SPRM macro in job step DSNTIZA. Make sure to add a continuation character in column 72 if needed.
- Run the first two steps of the DSNTIJUZ job you modified, to assemble and link the load module.
- After the job completes, you must either use the SET SYSPARM command or stop and start DB2 for the change to take effect.

#### REBIND level

It is controlled by new REBIND option also called PLANMGMT. When using BASIC/EXTENDED at the subsystem level, most bind options can still be changed at REBIND.

You can affect packages selectively with three possible settings: OFF, BASIC and EXTENDED.
5.14.2 Controlling the new SWITCH option for REBIND PACKAGE

When regression occurs, a new REBIND PACKAGE option called SWITCH, allows you to roll back to using the access path of the old package, effectively bypassing the current REBIND invocation.

- **SWITCH(PREVIOUS)**
  The PREVIOUS copy is activated
  Switches from the current to the previous copy
  Provides a means of falling back to last used copy

- **SWITCH(ORIGINAL)**
  The ORIGINAL copy is activated. The current copy moves to previous, and the original copy becomes current.
  It provides a mean of falling back to the oldest known package.

5.14.3 Deleting old PACKAGE copies

A new FREE PACKAGE option called PLANMGMTSCOPE allows you to free copies that are no longer necessary.

- **PLANMGMTSCOPE(ALL)**
  To free the entire package including all copies. This is the default.

- **PLANMGMTSCOPE(INACTIVE)**
  To free all old copies only.

The existing DROP PACKAGE () and DROP TRIGGER () commands will drop the specified package and trigger as well as all associated current, previous and original copies.
A simple query to determine the number of copies preserved for a package is listed in Example 5-5.

**Example 5-5  Finding existing package copies**

```sql
SELECT SP.COLLID, SP.NAME, SP.VERSION,
      COUNT(DISTINCT SPD.DTYPE) AS COPYCOUNT
FROM SYSIBM.SYSPACKAGE SP, SYSIBM.SYSPACKDEP SPD
WHERE SP.NAME = SPD.DNAME
GROUP BY SP.COLLID, SP.NAME, SP.VERSION
```

For additional details, refer to *DB2 Version 9.1 for z/OS Installation Guide*, GC18-9846 and other current DB2 product documentation.

### 5.14.4 Performance

The catalog tables involved for the plan copy management are:

- SYSPACKAGE, which reflects the current copy
- SYSPACKDEP, which reflects dependencies of all copies

Other catalog tables will reflect the metadata for all copies.

Note that using the PLANMGMT(BASIC) option can double the disk space allocation for tablespace SPT01 for each package, and using the PLANMGMT(EXTENDED) option can triple it. The extra space is needed to maintain old copies.

**Note:** PTF UK50987 for APAR PK80375 provides SPT01 compression.

Using the BASIC or EXTENDED option adds some CPU overhead to the performance of the REBIND PACKAGE command. For class 2 CPU time overhead at REBIND time see Figure 5-31 to Figure 5-35 on page 198.

These figures show the plan management measurement numbers. That is, they do not show the amount of regression that you can quickly avoid by going back to a ‘good’ package, they are meant to show the cost of investing in this new option.

A single package with either 1, 10 or 50 simple SQL statements in it, is used. For each RUN or REBIND measurement, the package is either run or rebound 100 times. The bars on left refer to the base REBIND PACKAGE() case, the bars on the right refer to the option listed in the title of the chart. All times are in seconds.

Figure 5-31 compares REBIND PACKAGE() PLANMGMT(OFF) to REBIND PACKAGE(). As expected, the overhead is within measurements noise, practically zero.
Figure 5-31  REBIND PACKAGE() PLANMGMT(OFF) vs. REBIND PACKAGE()

Figure 5-32 compares REBIND PACKAGE() PLANMGMT(BASIC) to REBIND PACKAGE(). The RUN overhead does not reach 2%. The REBIND overhead varies between 15 and 17%.

Figure 5-32  REBIND PACKAGE() PLANMGMT(BASIC) vs. REBIND PACKAGE()

Figure 5-33 compares REBIND PACKAGE() PLANMGMT(EXTENDED) to REBIND PACKAGE(). The RUN overhead does not reach 3%. The REBIND overhead varies between 17 and 21%.

Figure 5-33  REBIND PACKAGE() PLANMGMT(EXTENDED) vs. REBIND PACKAGE()
Figure 5-33  REBIND PACKAGE() PLANMGMT(EXTENDED) vs. REBIND PACKAGE()

Figure 5-34 compares REBIND SWITCH(PREVIOUS) to REBIND PACKAGE(). The RUN overhead is almost zero. The REBIND SWITCH to previous shows an improvement between 52 and 55%.

Figure 5-34  REBIND PACKAGE() SWITCH(PREVIOUS) vs. REBIND PACKAGE()

Figure 5-35 compares REBIND SWITCH(ORIGINAL) to REBIND PACKAGE(). The RUN overhead is almost zero. The REBIND SWITCH to the original package shows an improvement between 54 and 63%.

Figure 5-35  REBIND PACKAGE() SWITCH(ORIGINAL) vs. REBIND PACKAGE()
5.14.5 Comments

Laboratory measurements show that basically the run time overhead is negligible, preserving old copies has no impact on query performance.

There is no overhead for PLANMGMT(OFF). The CPU overhead for binds using multiple package copies is a fully justifiable 15-20% in these measurements. REBIND with SWITCH() is very efficient with a 2 to 3 times reduction.

Of course your mileage will vary, depending on the number and complexity of the SQL in the packages.

DB2 users concerned about access path regressions seen after REBIND PACKAGE command can use this new function to preserve and restore old access paths with very good performance in terms of CPU overhead.

However, this is a safety mechanism that, if not used by exception on packages considered critical, but applied generically, will require extra directory (and at a lesser extent catalog) allocation depending on the number of package copies maintained.
Chapter 6. Utilities

DB2 9 for z/OS delivers a number of performance boosting enhancements by way of its Utilities Suite. One of the more remarkable features in this release is the reduction in utility CPU consumption particularly for CHECK INDEX, LOAD, REORG, REBUILD INDEX, and RUNSTATS. DB2 V9 also improves data availability with online processing for utilities such as CHECK DATA, CHECK LOB, REBUILD INDEX, and REORG. In addition to improved recovery capabilities, DB2 V9 provides the ability to collect range distribution statistics through the RUNSTATS utility to better assist the optimizer in access path selection.

In this chapter, we discuss the improvements that have been made to the utilities in DB2 V9 with accentuation on the performance they deliver. Here are the areas of focus:

- Utility CPU reduction
- MODIFY RECOVERY enhancements
- RUNSTATS enhancements
- Recovery enhancements
- Online REBUILD INDEX enhancements
- Online REORG enhancement
- Online CHECK DATA and CHECK LOB
- TEMPLATE switching
- LOAD COPYDICTIONARY enhancement
- COPY performance
- Best practices
6.1 Utility CPU reduction

DB2 V9 takes a major step toward reducing overall CPU utilization for DB2 utilities. This improvement in CPU cost is mostly evident in the CHECK INDEX, LOAD, REBUILD, REORG, and RUNSTATS utilities. This objective has been achieved primarily by the following initiatives:

- There is a reduction in the path length in the index manager.
  - The creation of a block interface to the index manager allows many index keys to be passed in a single call.
  - The overhead of calling the index manager for every key is avoided, resulting in a significant CPU reduction.

- Shared memory objects that are located above the bar are used to avert data movement. The movement of rows between the batch and DBM1 address spaces is avoided, which results in reduced CPU usage.

- Significant improvements have been made to the index key generation component in DB2 V9. The ability has been added to efficiently handle non-padded index keys with varying length columns.

Note: These enhancements take effect immediately in DB2 9 conversion mode.

The performance advances of the CHECK INDEX, LOAD, REBUILD, REORG, and RUNSTATS utilities in DB2 V9 have been measured and are analyzed in this section.

6.1.1 CHECK INDEX performance

The CHECK INDEX utility tests whether indexes are consistent with the data that they index and issues warning messages when it finds an inconsistency. The performance of this utility with default options was tested in order to recognize the CPU reduction that has occurred with DB2 V9. All measurements were performed using a System z9 processor with z/OS 1.8 and DS8300 disks. Table 6-1 provides a summary of the details of the test cases.

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table and index details</th>
<th>Index to be checked</th>
<th>Control statement (with SHRLEVEL REFERENCE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>➤ 10 partitions</td>
<td>One partitioning index</td>
<td>CHECK INDEX (with defaults)</td>
</tr>
<tr>
<td></td>
<td>➤ 26 columns with 118 byte length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 50 million rows</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 1 partitioning index (PI)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 5 non-partitioning indexes (NPIs)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>➤ Same as case 1</td>
<td>One NPI</td>
<td>CHECK INDEX (with defaults)</td>
</tr>
</tbody>
</table>

A comparison of both DB2 V8 and V9 was conducted and the results are shown in Figure 6-1. The measurements show a 36% reduction in both CPU time and elapsed time when performing CHECK INDEX with default options on the partitioning index. Likewise, when running this utility on the NPI, there is a 20% reduction in CPU time and a 39% reduction in...
elapsed time. The elapsed time reduction is attributed to two factors: the reduction in CPU and the parallelism for SHRLEVEL REFERENCE that was added in V9.

![CHECK INDEX Performance](image)

**Figure 6-1** CHECK INDEX SHRLEVEL REFERENCE results

**Conclusion**
The CHECK INDEX utility takes advantage of the various index management enhancements that are introduced in DB2 V9.

**6.1.2 LOAD performance**

The LOAD utility loads records into one or more tables of a table space and builds or extends any indexes that are defined on them. In order to examine the improvements that are made to the utility CPU usage in DB2 V9, the LOAD utility was executed using two different workloads in both DB2 V8 and V9. These two workloads are described in the sections that follow.
Workload 1: Comparison of LOAD in V8 and V9

The CPU time and elapsed time that are required to execute the LOAD utility on a partitioned table space were examined. All measurements were performed using a System z9 processor and DS8300 disks running z/OS 1.8. Table 6-2 summarizes the details of the workload that was used.

Table 6-2  Details of workload 1 - LOAD

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table details</th>
<th>Index details</th>
<th>Input details</th>
<th>Control statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10 partitions</td>
<td>1 PI</td>
<td>50 million rows</td>
<td>LOAD (with defaults)</td>
</tr>
<tr>
<td></td>
<td>26 columns with 118 byte length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Same as Case 1</td>
<td>1 PI</td>
<td>50 million rows</td>
<td>LOAD (with defaults)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 NPI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Same as Case 1</td>
<td>1 PI</td>
<td>50 million rows</td>
<td>LOAD (with defaults)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 NPIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Same as Case 1</td>
<td>1 PI</td>
<td>5 million rows</td>
<td>LOAD PART 1 (with defaults)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 NPIs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A comparison of both DB2 V8 and V9 was conducted; Figure 6-2 shows the results.

Figure 6-2  CPU improvement on LOAD utility

The measurements show a considerable improvement in both the CPU time and elapsed time in all four cases that were tested. The most visible CPU time improvement of 33% is experienced when loading a single partition in comparison to DB2 V8. Similarly, a 29% reduction in elapsed time is also seen. In addition, a 25% improvement in both CPU time and elapsed time can be seen when loading an entire table space with a partitioning index defined. Although there are improvements when loading a table with NPIs defined, the improvements to the CPU and elapsed times are less significant as the number of NPIs increases.
Conclusion
The results indicate that the index enhancements that have been made in the release of DB2 V9 have reduced NPI updates considerably. As a result, the BUILD and RELOAD phases benefit the most from these enhancements. However, as you increase the number of NPIs, the sorting subtasks begin to neutralize the overall improvement.

Workload 2: Comparison of LOAD PART REPLACE in V8 and V9
This test measures the CPU time and elapsed time of the LOAD PART REPLACE utility for a partitioned table space with dummy input. All measurements were performed using a System z9 processor with z/OS 1.8 and DS8300 disks. The LOAD utility was tested; Table 6-3 summarizes the details.

Table 6-3  Details of workload 2 - LOAD PART REPLACE

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table details</th>
<th>Index details</th>
<th>Control statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10 partitions 26 columns with 118 byte length 5 million rows in PART 1</td>
<td>1 PI 1 NPI</td>
<td>LOAD PART 1 REPLACE (with defaults)</td>
</tr>
<tr>
<td>2</td>
<td>10 partitions 26 columns with 118 byte length 0 rows in PART 1</td>
<td>1 PI 1 NPI</td>
<td>LOAD PART 1 REPLACE (with defaults)</td>
</tr>
<tr>
<td>3</td>
<td>10 partitions 26 columns with 118 byte length 18 thousand rows in PART 1</td>
<td>1 PI 1 NPI</td>
<td>LOAD PART 1 REPLACE (with defaults)</td>
</tr>
</tbody>
</table>

This test was performed for both DB2 V8 and V9, and the results were compared. Figure 6-3 shows the results of this test case.

Figure 6-3  CPU improvement of the LOAD utility with dummy input
Figure 6-3 is dominated by a whopping 67% CPU time reduction from V8 when loading dummy input into a 5 million row partition. Correspondingly, a significant 65% reduction in elapsed time is also observed in case 1. When loading dummy input into an empty partition (case 2), there is an average 19% cost savings in both CPU time and elapsed time. Case 3 illustrates a 9% cost in CPU time and a 6% increase in elapsed time compared to DB2 V8.

**Conclusion**

It is evident that loading dummy input into a partition that contains a large number of rows produces an excellent performance return. This improvement is the result of the improvements in index management activity, which takes place mostly during the BUILD phase.

You will notice that there is a slight cost when loading dummy input into a partition that is empty or contains a small number of rows. This is because of the cost that is associated with scanning the whole NPI. This becomes less visible when the partition contains a larger number of rows.

### 6.1.3 REBUILD INDEX performance

The REBUILD INDEX utility reconstructs indexes or index partitions from the table that they reference, extracting the index keys, sorting the keys, and then loading the index keys into the index. The performance of this utility was tested in order to recognize the CPU reduction. The CPU time and elapsed time of the utility was measured for a partitioned table space. All measurements were performed using a System z9 processor with z/OS 1.8 and DS8300 disks. Table 6-4 summarizes the details.

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table and index details</th>
<th>Index to be rebuilt</th>
<th>Control statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>➤ 10 partitions</td>
<td>➤ 1 PI</td>
<td>REBUILD INDEX 1 PI (with defaults)</td>
</tr>
<tr>
<td></td>
<td>➤ 26 columns with 118 byte length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 50 million rows</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 1 partitioning index</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 5 NPIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>➤ Same as Case 1</td>
<td>➤ 1 NPI</td>
<td>REBUILD INDEX 1 NPI (with defaults)</td>
</tr>
<tr>
<td>3</td>
<td>➤ Same as Case 1</td>
<td>➤ 1 NPI part</td>
<td>REBUILD INDEX NPI/PART 1 (with defaults)</td>
</tr>
<tr>
<td>4</td>
<td>➤ Same as Case 1</td>
<td>➤ 1 PI</td>
<td>REBUILD INDEX 1 PI and REBUILD INDEX 1 NPI (with defaults)</td>
</tr>
<tr>
<td></td>
<td>➤ 1 NPI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>➤ Same as Case 1</td>
<td>➤ 1 PI</td>
<td>REBUILD INDEX (ALL) (with defaults)</td>
</tr>
<tr>
<td></td>
<td>➤ 5 NPIs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The performance of this utility was tested, and a comparative analysis was conducted between DB2 V8 and V9. Figure 6-4 presents the results of this test case.
Chapter 6. Utilities

205

Figure 6-4   CPU improvement on REBUILD INDEX

It is evident that the largest CPU reduction of 19% is experienced when rebuilding the single partitioning index. This also results in a 17% reduction in elapsed time. However, rebuilding the two indexes translates into a 15% CPU time reduction and a 13% elapsed time reduction. Similarly, the trend in the reduction in CPU time continues with a 13% decrease in CPU time when rebuilding the six indexes. Yet, a 21% reduction in elapsed time is witnessed when rebuilding the six indexes. Rebuilding a single NPI yields only a 5% improvement in both CPU and elapsed time. Correspondingly, a minimal improvement was observed when rebuilding the logical part of an NPI.

Conclusion

The REBUILD INDEX utility takes full advantage of the enhancements that are made to index management in DB2 V9 to decrease CPU and elapsed times. The more indexes that are rebuilt, the more improvement is experienced in elapsed time due to the parallelism in the index building process. However, as expected, the sorting subtasks will impact the CPU time and the improvements will lessen. Also, no improvement will be observed in the rebuilding of the logical part of an NPI since the new block interface implementation would not be invoked.

6.1.4 REORG performance

The REORG utility reorganizes a table space (REORG TABLESPACE) to improve access performance and to reclaim fragmented space or re-establish free space. In addition, indexes can be reorganized with the REORG INDEX utility so that they are more efficiently clustered or to re-establish free space. These two functions have been examined in greater detail to compare their performance in both DB2 V8 and V9 environments. In order to test the improvements made in DB2 V9, the REORG utility was executed both at the table space and index space levels.
REORG TABLESPACE comparison between V8 and V9

In this test, we measured the CPU time and elapsed time of the REORG utility for a partitioned table space. All measurements were performed using a System z9 processor with z/OS 1.8 and DS8300 disks. The LOAD utility was tested; Table 6-5 summarizes the details.

Table 6-5  Details of workload - REORG TABLESPACE

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table details</th>
<th>Index details</th>
<th>Control statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10 partitions ▶ 26 columns with ▶ 5 million rows per partition</td>
<td>1 partitioning index</td>
<td>REORG TABLESPACE LOG NO (with defaults)</td>
</tr>
<tr>
<td>2</td>
<td>Same as Case 1</td>
<td>1 partitioning index ▶ 1 NPI</td>
<td>REORG TABLESPACE LOG NO (with defaults)</td>
</tr>
<tr>
<td>3</td>
<td>Same as Case 1</td>
<td>1 partitioning index ▶ 5 NPIs</td>
<td>REORG TABLESPACE LOG NO (with defaults)</td>
</tr>
<tr>
<td>4</td>
<td>Same as Case 1</td>
<td>1 partitioning index ▶ 5 NPIs</td>
<td>REORG TABLESPACE PART 1 LOG NO (with defaults)</td>
</tr>
</tbody>
</table>

This test case measures the CPU tie and elapsed time of the REORG utility for a partitioned table space. A comparative analysis was conducted to see the utility’s performance in both DB2 V8 and V9. Figure 6-6 shows the results of this test case.

REORG Performance

![CPU Time](chart)

![Elapsed Time](chart)

We observe a 9% CPU time improvement when reorganizing a table space with a single NPI defined. In addition, a 5% improvement in elapsed time is also seen in this case. When
reorganizing the whole table space with five NPIs defined, the CPU time and elapsed time improvement are 8% and 4% respectively. By reorganizing a single partition of a table space with five NPIs defined, a minimal CPU time improvement of 3% is observed compared to V8. A 5% improvement in elapsed time is noted in this particular case.

**Conclusion**

The block interface to the index manager implementation in DB2 V9 assists the REORG utility when reorganizing a table space with indexes defined. This is evident through the improvements to the CPU and elapsed times in the BUILD phase. However, as you increase the number of indexes that are defined on the table space, the sorting subtasks begin to impact both the CPU and elapsed times in the UNLOAD and SORT phases. Additionally, there is only a minimal improvement in the single partition REORG since the block interface implementation to the index manager is not used on logical partitions (LPARs).

**REORG INDEX comparison between V8 and V9**

This test measures the CPU time and elapsed time of the REORG utility for different indexes. All measurements were performed using a System z9 processor with z/OS 1.8 and DS8300 disks. The REORG INDEX utility was tested; Table 6-6 provides a summary of the details.

*Table 6-6  Details of workload - REORG INDEX*

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table and index details</th>
<th>Index to be reorganized</th>
<th>Control statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>➤ 10 partitions</td>
<td>➤ 1 PI</td>
<td>REORG INDEX (with defaults)</td>
</tr>
<tr>
<td></td>
<td>➤ 26 columns with 118</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>byte length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 5 million rows per</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 1 partitioning index</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 1 NPI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>➤ Same as Case 1</td>
<td>➤ 1 NPI</td>
<td>REORG INDEX (with defaults)</td>
</tr>
<tr>
<td>3</td>
<td>➤ Same as Case 1</td>
<td>➤ 1 PI</td>
<td>REORG INDEX PART 1 (with defaults)</td>
</tr>
</tbody>
</table>
This test case measures the CPU and elapsed time of the REORG utility for an index (PI, NPI, and so on). A comparative analysis was conducted to see the utility's performance in both DB2 V8 and V9. Figure 6-6 presents the results of this test case.

Figure 6-6 shows a monolithic improvement in both CPU time and elapsed time compared to V8. For the three cases that we tested, there is a 46% to 48% improvement in CPU time. Correspondingly, this is complemented with a 45% to 48% improvement in elapsed time.

Conclusion
The REORG INDEX benefits the most from the block interface implementation to the index manager in DB2 V9. This overall reduction in CPU originates from a decreased CPU cost in the UNLOAD and BUILD phases. There is a reduction in overhead since there is no need to cross the component interfaces so many times. In addition, the improvements are attributed to various index enhancements that have been made in this release.

6.1.5 RUNSTATS index performance

The RUNSTATS utility gathers summary information about the characteristics of data in table spaces, indexes, and partitions. The performance of this utility was tested in order to recognize the CPU reduction. The CPU time and elapsed time of the utility were tested for a partitioned table space. All measurements were performed using a System z9 processor with z/OS 1.8 and DS8300 disks. Table 6-7 provides a summary of the details.
Table 6-7  Details of RUNSTATS (index) measurements

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table and index details</th>
<th>Index to be checked</th>
<th>Control statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>➤ 10 partitions</td>
<td>➤ 1 PI</td>
<td>RUNSTATS INDEX (with defaults)</td>
</tr>
<tr>
<td></td>
<td>➤ 26 columns with 118 byte length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 50 million rows</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 1 partitioning index</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>➤ 5 NPIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>➤ Same as above</td>
<td>➤ 1 NPI</td>
<td>RUNSTATS INDEX (with defaults)</td>
</tr>
<tr>
<td>3</td>
<td>➤ Same as above</td>
<td>➤ 1 PI, 5 NPIs</td>
<td>RUNSTATS INDEX (with defaults)</td>
</tr>
</tbody>
</table>

Figure 6-7 presents the results of these three test cases.

The measurements show a 33% reduction in both CPU and elapsed times when performing RUNSTATS on the partitioning index. Similarly, both the CPU and elapsed times are reduced by 45% when running this utility on one NPI. Moreover, in comparison to DB2 V8, there is a 42% reduction in both CPU time and elapsed time when performing RUNSTATS on all six indexes.

**Conclusion**

Similar to other index-intensive utilities, the RUNSTATS (index) utility takes full advantage of the block interface implementation to the index manager in DB2 V9. As a result, there is a reduction in overhead since there is no need to cross the component interfaces so many times.
6.1.6 Index key generation improvements

In general, DB2 uses a unique technique to handle index key generation from a data row. In DB2 V8, this technique was exploited only for index keys that are fixed in length. These index keys also had variable length columns as long as the index was not created with the “NOT PADDED” attribute. Hence, the varying length columns were padded with blanks or zeros to the maximum length. If you created the key with the “NOT PADDED” attribute to save disk space, this technique was not able to handle it, and other techniques were used.

In DB2 9, modifications to this technique have been made to efficiently handle indexes with varying length key columns as well as the “NOT PADDED” attribute (available in conversion mode). In addition, the technique was extended to support the new reordered row format (see 4.14, “Reordered row format” on page 121). Since the data rows are already stored in an improved format, the performance gain is less in comparison to indexes with varying length key columns. Given that the reordered row format is active only in new-function mode, this improvement can only be seen when you are in new-function mode.

To assess the performance of these improvements as seen through the CHECK INDEX, LOAD, REBUILD, REORG, and RUNSTATS utilities, the CPU time and elapsed time of the utility were measured. All measurements were performed by using a System z9 processor with z/OS 1.8 and DS8300 disks. Table 6-8 provides a summary of the details.

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table details</th>
<th>Index details</th>
<th>Utility (with default options)</th>
</tr>
</thead>
</table>
| 1           | 20 million rows  
              One index  
              VARCHAR columns  
              (length varying from 1 to 14) | Two scenarios used:  
              1. Fixed  
              2. VARCHAR (not padded) |  
              CHECK INDEX  
              LOAD  
              REBUILD INDEX  
              REORG TABLESPACE  
              REORG INDEX  
              RUNSTATS INDEX |
| 2           | 20 million rows  
              One index  
              Four VARCHAR columns of length 1 | Two scenarios used:  
              1. Fixed  
              2. VARCHAR (not padded) |  
              CHECK INDEX  
              LOAD  
              REBUILD INDEX  
              REORG TABLESPACE  
              REORG INDEX  
              RUNSTATS INDEX |
The two cases were tested with the different scenarios, and a comparative analysis was performed. Figure 6-8 shows the results for Case 1.

![Percentage Difference V8 to V9 Index Key Generation – Case 1](chart)

It is evident that all utilities take advantage of the improvements that are made to index key generation when the columns vary in length from 1 to 14. For the fixed scenario, there is a reduction of 6% to 47% in both CPU time and elapsed time in comparison to V8. Similarly, the VARCHAR scenario yields an 8% to 66% reduction in both CPU time and elapsed time.
Figure 6-9 shows the results of Case 2.

Similar to the first case, the results shown in Figure 6-9 produce a 7% to 51% reduction in both CPU time and elapsed time for the fixed scenario. The VARCHAR scenario yields a 28% to 67% reduction for the CPU and elapsed times for the utilities that were selected.

**Conclusion**

Therefore, the enhancements that have been made to index key generation in DB2 V9 have significant performance benefits for most utilities. In addition to the reordered row format support, DB2 is now able to effectively support indexes with varying length key columns and indexes that are defined with the “NOT PADDED” attribute.

### 6.2 MODIFY RECOVERY enhancements

The MODIFY utility with the RECOVERY option deletes records from the SYSIBM.SYSCOPY catalog table, related log records from the SYSIBM.SYSLGRNX directory table, and entries from the database descriptor (DBD). This option also recycles DB2 version numbers for reuse.

The MODIFY option allows you to remove outdated information from both SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX to save disk space and increase performance. These tables, particularly SYSIBM.SYSLGRNX, can become large and take up a considerable amount of space.
With versions prior to DB2 V9, you can remove records that were written before a specific date or of a specific age, and you can delete records for an entire table space, partition, or data set. DB2 V9 enhancements simplify the usage MODIFY RECOVERY and make it safer to use by not deleting more than is intended.

In addition, with DB2 V9, the MODIFY RECOVERY utility no longer acquires the “X” DBD lock when removing the object descriptors (OBDs) for tables that had been previously dropped. In DB2 V9, the “U” DBD lock is acquired instead of the X DBD lock to allow availability and flexibility.

The MODIFY RECOVERY utility is updated to allow more control over the range of recovery information that is retained for any given table space, through the use of the RETAIN keyword. New specifications in the MODIFY RECOVERY statement enable you to be more granular when you define your retention criteria. See Figure 6-10.

![Figure 6-10 MODIFY RECOVERY syntax](image)

For details about the syntax, see *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855. The introduction of the keyword RETAIN in DB2 9 (available only in new-function mode) promotes simplification and safety instead of the tricky approach in defining deletion criteria as was the case in previous releases.

**Conclusion and recommendations**

The MODIFY RECOVERY utility now deletes SYSIBM.SYSLGRNX entries even if no SYSIBM.SYSCOPY records are deleted. By deleting outdated information from these tables, you can help improve performance for processes that access data from these tables.

**Note:** Deletion works on dates and not on timestamps. As a result, more entries than requested might be kept. For example, if the five most recent copies were taken on the same day, and RETAIN LAST(2) is specified, then the records for all five copies that were taken on that day are retained in SYSCOPY.

We strongly recommend that you use the MODIFY RECOVERY utility regularly to take advantage of these benefits. In addition, to further improve performance and minimize lock contention, perform a REORG of both SYSIBM.SYSCOPY and SYSIBM.SYSLGRNX on a regular basis as well.

If you are using BACKUP SYSTEM and not regular image copies, REORG still inserts rows in SYSIBM.SYSCOPY. In this case, we recommend that you continue to use MODIFY RECOVERY with DELETE AGE or DELETE DATE.
6.3 RUNSTATS enhancements

The RUNSTATS utility gathers summary information about the characteristics of data in table spaces, indexes, and partitions.

Data sequences and distribution of values help DB2 in determining the best access path and the best technique for I/O.

DB2 9 for z/OS introduces histogram statistics to deal with complex and large variations of value distributions and improves the methodology for collecting data clustering information.

6.3.1 Histogram statistics

Data distribution statistics, which are recorded in the DB2 catalog, are important for query optimization. The DB2 optimizer uses these statistics to calculate the cost in creating the most optimal access path for each SQL statement during the bind process. Prior to DB2 9, frequency statistics that were collected relied on single values, either from a single column or from multiple columns. The existing optimizer technique uses the column frequency and is good in the evaluation of a small amount of values. Histogram statistics are supported in DB2 9 for z/OS conversion mode to provide the optimizer with more meaningful data.

**Note:** RUNSTATS TABLESPACE ignores the HISTOGRAM option when processing XML table spaces and indexes.

Histogram statistics collect the frequency statistics of the distinct values of column cardinality over an entire range. This method results in better selectivity and has the potential for better access path selection. As a result, the RUNSTATS utility (RUNSTATS TABLESPACE / RUNSTATS INDEX) now collects information by quantiles. You can specify how many quantiles DB2 is to use from 1 to 100 per column.

Three new columns have been added to SYSIBM.SYSCOLDIST and SYSIBM.SYSKEYTGTDIST (as well as SYSCOLDIST_HIST, SYSCOLDISTSTATS, SYSKEYTGTDIST_HIST, SYSKEYGTDISTSTATS) catalog tables:

- QUANTILENO
- LOWVALUE
- HIGHVALUE

Since the histogram describes the data distribution over the entire range, predicate selectivity is calculated more accurately if the range matches the boundary of any one quantile or any group of consecutive quantiles. Even if there is no perfect match, predicate selectivity interpolation is now done within one or two particular quantiles. With the interpolation done in much smaller granularity, predicate selectivity is expected to be evaluated with more accuracy.

**Note:** Inline RUNSTATS for histogram statistics is not supported in the REORG and LOAD utilities.

**Performance**

The RUNSTATS utility was tested to evaluate the CPU time to collect frequency statistics and histogram statistics as well as a combination of the two. Table 6-9 shows the results of these measurements. Here we describe the environment and the workload that was used:

The test environment consisted of:
System z9 processor
ESS model E20
z/OS 1.8
DB2 for z/OS 9 (new-function mode)

The workload for measurement consisted of:
- Table with 1.65 M rows
- 1 PI
- 5 NPIs
- 10 partitions

We ran measurements on the following cases:

Case 1:
```
RUNSTATS TABLESPACE PEPPDB.CARTSEG TABLE(OCB.COVRAGE)
COLGROUP(CVRGTYPE) COUNT(9) UPDATE(NONE) REPORT YES
```

Case 2:
```
RUNSTATS TABLESPACE PEPPDB.CARTSEG TABLE(OCB.COVRAGE)
COLGROUP(CVRGTYPE) FREQVAL UPDATE(NONE) REPORT YES
```

Case 3:
```
RUNSTATS TABLESPACE PEPPDB.CARTSEG TABLE(OCB.COVRAGE)
COLGROUP(CVRGTYPE) HISTOGRAM UPDATE(NONE) REPORT YES
```

Table 6-9 shows that collection frequency statistics or histogram statistics add some overhead in CPU (and elapsed times). This is due to the COLUMN ALL default option, which contributes to most of the time by using CPU for collecting statistics on a large number of columns.

Table 6-9 Performance of frequency and histogram statistics collection

<table>
<thead>
<tr>
<th>Case</th>
<th>1 column count</th>
<th>2 column count</th>
<th>3 column count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Count only</td>
<td>1.94</td>
<td>2.06</td>
<td>2.26</td>
</tr>
<tr>
<td>2 - Frequency evaluation</td>
<td>2.06</td>
<td>2.33</td>
<td>2.48</td>
</tr>
<tr>
<td>3 - Histogram</td>
<td>2.2</td>
<td>2.61</td>
<td>2.78</td>
</tr>
</tbody>
</table>

Conclusion
The implementation of collecting histogram statistics by the RUNSTATS utility proves to be beneficial since the extra cost is marginal when compared to the collection of frequency statistics. The DB2 optimizer takes advantage of the new histogram statistics and improves access path selection.

Note: Additional CPU overhead is required and is expected when collecting multi-column statistics.

For more information about the influence that histogram statistics have on access path determination, see 2.17, “Histogram statistics over a range of column values” on page 55.
Recommendations

As a general recommendation, specify RUNSTATS only on columns that are specified in the SQL statements (predicates, order by, group by, having). For histograms, specify 100 (default) for the NUMQUANTILES option and let DB2 readjust to an optimal number. Predicate selectivity is more accurate if the searching range matches the boundary of any one quantile or any group of consecutive quantiles. Lower values can be used when there is a good understanding of the application. For example, if the query ranges are always done on boundaries such as 0-10%, 10-20%, 20-30%, then NUMQUANTILES 10 may be a better choice. Even if there is no perfect match, predicate selectivity interpolation is done with one or two particular quantiles, which results in more accurate predicate evaluation.

You can use histogram statistics to evaluate predicate selectivity. The better filter factor benefits RANGE/LIKE/BETWEEN predicates for all fully qualified intervals and interpolation of partially qualified intervals. It can also help in case of EQ, IS NULL, IN LIST, and COL op COL.

**Note:** Even if the RUNSTATS utility is run on less than 100 column values, histogram statistics are collected.

If RUNSTATS TABLESPACE is executed on columns or column groups, sorting is required. As a result, if FREQVAL statistics are also specified, then they will share the same sort. We recommend that you collect only column statistics on columns that are used in SQL predicates.

**Tip:** Use the Statistics Advisor to determine which statistics to collect. New for V9, the Statistics Advisor automatically recommends when to collect histogram statistics.

If RUNSTATS INDEX is executed for an index with key columns of mixed order, histogram statistics can be collected only on the prefix columns with the same order. Hence, if the specified key columns for histogram statistics are of mixed order, a warning message DSNU633 is issued, and the HISTOGRAM option is ignored.

### 6.3.2 CLUSTERRATIO enhancements

The way CLUSTERRATIO is calculated in V8 and prior releases of DB2 is showing some limitations and has been enhanced with DB2 9 for z/OS to better reflect new requirements and new functions (such as the increased dynamic prefetch use).

With V8, the CLUSTERRATIO formula counts a row as clustered if the next key resides on an equal or forward page of the last RID of the current key. This does not help in determining whether the gap to the next key could benefit from prefetch, based on the current prefetch quantity. The CLUSTERRATIO formula just considers the next key clustered whether in the next page or 1000 pages ahead, although a key with a distance greater than the prefetch quantity would not be found in the buffer pool following the next prefetch request.

This first limitation may have led to the inappropriate usage of sequential prefetch or list prefetch, or even cases of the DB2 optimizer deciding not to use an index based upon an over-estimated CLUSTERRATIO.

With DB2 9, the formula tracks pages within a sliding prefetch window that considers both the prefetch quantity and buffer pool size. The page is considered clustered only if it falls within this window.
Another limitation is that with V8, the current method of tracking keys ahead of the current position does not consider a reverse clustering sequence. Dynamic prefetch can be activated on trigger values in forward or reverse sequence, however the V8 CLUSTERRATIO formula considers only sequential prefetch, which only supports forward access.

With DB2 9, dynamic prefetch is used more widely (see 2.2, “Dynamic prefetch enhancement for regular index access during an SQL call” on page 14) so the sliding window now counts pages as clustered in either direction, since the direction can change from forward to reverse and still be considered clustered.

A third limitation of the DB2 V8 CLUSTERRATIO formula is the lack of information on data density. It calculates that data is sequential, but not how dense it is. From an access point of view it is good to let the optimizer know if retrieving a certain number of rows via a clustered index involves just a few data pages (dense data) or they are scattered across many pages (plain sequential data).

The DB2 9 formula introduces a new SYSINDEXES catalog table column, DATAREPEATFACTORF, which tracks whether rows are found on a page other than the current. Used in conjunction with the CLUSTERRATIOF value, the optimizer can now distinguish between dense and sequential and make better choices.

A fourth limitation with V8 is that only the distinct key values (not the RIDs) are counted, this results in lower cardinality indexes being at a disadvantage since the resultant CLUSTERRATIO was not as high as if all RIDs were counted. Notice that duplicate RID chains are guaranteed to be in sequence anyway. The consequence is that lower cardinality indexes may not be considered by the optimizer due to the expectation of a high percentage of random I/Os, whereas sequential RID chains can benefit from dynamic prefetch.

The DB2 9 CLUSTERRATIO formula has been enhanced to count each RID.

**Activating the enhanced CLUSTERRATIO**

The enhancements to the DB2 9 CLUSTERRATIO formula are activated when STATCLUS subsystem parameter is set to the default value of ENHANCED on the DB2 utilities install panel as shown in Figure 6-11.

See also 9.4, “DSNZPARM changes” on page 273.

The new column DATAREPEATFACTORF is defined as FLOAT and represents the anticipated number of data pages that will be touched following an index key order. It is present in index catalog tables such as SYSINDEXES, SYSINDEXES_HIST, SYSINDEXESSTATS, and SYSINDEXESSTATS_HIST.

**Recommendations**

As a general recommendation, while this is a configurable DSNZPARM, do not disable the default of RUNSTATS collecting the new CLUSTERRATIO and DATAREPEATFACTOR information. They will help the optimizer to make better choices and exploit DB2 access functions.

**Important:** You need to run the IBM RUNSTATS utility after DB2 9 migration and before rebinding static SQL or preparing dynamic SQL to benefit from the DB2 9 optimizer enhancements that expect a more accurate CLUSTERRATIO calculation.

Apply PTF UK47894 for APAR PK84584 to resolve better cluster ratio and sequential detection in RUNSTATS for index and table spaces with pages greater than 4 KB.
The improved formula with DATAREPEATFACTOR and associated optimizer enhancements are available in all modes of DB2 9.

---

**DSNTIP6 MIGRATE DB2 - DB2 UTILITIES PARAMETERS**

```plaintext
DSNTIP6  MIGRATE DB2 - DB2 UTILITIES PARAMETERS
====>

Enter system-level backup options for RESTORE SYSTEM and RECOVER below:
1 SYSTEM-LEVEL BACKUPS ===> NO  As a recovery base: NO or YES
2 RESTORE/RECOVER ===> NO  From dump: NO or YES
3 DUMP CLASS NAME ===> NO  For RESTORE/RECOVER from dump
4 MAXIMUM TAPE UNITS ===> NOLIMIT  For RESTORE SYSTEM: NOLIMIT or 1-255

Enter other DB2 Utilities options below:
5 TEMP DS UNIT NAME ===> VIO  Device for temporary utility data sets
6 UTILITY CACHE OPTION ===> NO  3990 storage for DB2 utility IO
7 STATISTICS HISTORY ===> NONE  Default for collection of stats history
8 STATISTICS ROLLUP ===> NO  Allow statistics aggregation: NO or YES
9 STATISTICS CLUSTERING ===> ENHANCED  For RUNSTATS (ENHANCED or STANDARD)
10 UTILITY TIMEOUT ===> 6  Utility wait time multiplier

PRESS: ENTER to continue  RETURN to exit  HELP for more information
```

---

**Figure 6-11  DSNTIP6**

### 6.4 Recovery enhancements

With the number of objects that are involved in today's robust databases, it has become an overwhelming task to maintain them. As a result, DB2 V9 provides enhanced backup and recovery mechanisms through such utilities as BACKUP SYSTEM, RESTORE SYSTEM, and RECOVER to assist you in this process. In addition, the integration of the incremental FlashCopy feature into these utilities minimizes I/O impact and considerably reduces elapsed time for creation of the physical copy.

#### 6.4.1 BACKUP and RESTORE SYSTEM

Prior to DB2 V9, the online BACKUP SYSTEM utility invokes z/OS DFSMShsm, which allows you to copy the volumes on which the DB2 data and log information reside for either a DB2 subsystem or a data sharing group. All data sets that you want to copy must be storage management subsystem (SMS)-managed data sets. You can subsequently run the RESTORE SYSTEM utility to recover the entire system.

In DB2 V9 for z/OS, the BACKUP SYSTEM utility can be used to manage system-level backups on tape. This capability requires a minimum of z/OS DFSMShsm V1.8 and is available only in new-function mode.
The syntax in Figure 6-12 illustrates this new functionality.

Three options provide the tape control for this utility. See the *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855, for details.

- **FORCE**: The FORCE option allows you to overwrite the oldest DFSMSHsm version of the fast replication copy of the database copy pool. If the copy pool's DFSMSHsm dump classes have been initiated, but are only partially completed, you can still overwrite these copy pools.

- **DUMP**: The DUMP option indicates that you want to create a fast replication copy of the database copy pool and the log copy pool on disk and then initiate a dump to tape of the fast replication copy. The dump to tape begins after DB2 successfully establishes relationships for the fast replication copy. To minimize the peak I/O on the system, the BACKUP SYSTEM utility does not wait for the dump processing to complete.

- **DUMPONLY**: The DUMPONLY option allows you to create a dump on tape of an existing fast replication BACKUP SYSTEM copy (that is currently residing on the disk) of the database copy pool and the log copy pool. You can also use this option to resume a dump process that has failed. Similar to the DUMP option, the BACKUP SYSTEM utility does not wait for the dump processing to complete, which minimizes the peak I/O on the system.
Like the BACKUP SYSTEM utility, the RESTORE SYSTEM utility in DB2 V9 also supports tape control when using system-level backups. Tape control requires a minimum of z/OS DFSMShsm V1.8 and is available only in new-function mode. See Figure 6-13.

![RESTORE SYSTEM syntax](image)

The tape management is controlled by the mutual combination of the two options:

- **FROMDUMP**: The FROMDUMP option indicates that you want to dump only the database copy pool to tape during the restore. Parallelism is exercised here but is limited by the number of distinct tape volumes on which the dump resides.

  **Note**: The FROMDUMP and DUMPCLASS options that you specify for the RESTORE SYSTEMS utility override the RESTORE/RECOVER FROM DUMP and DUMPCLASS NAME installation options that you specify on installation panel DSNTIP6.

- **TAPEUNITS**: The TAPEUNITS option specifies the limit on the number of tape drives that the utility should dynamically allocate during the restore of the database copy pool from dumps on tape.

  **Note**: The default is the option that you specified on installation panel DSNTIP6. If no default is specified, then the RESTORE SYSTEM utility tries to use all of the tape drives in your system.

See the *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855, for details.

**Object-level recovery**

DB2 V9 offers the ability to recover individual table spaces and index spaces (COPY YES indexes) using system-level backups taken by the BACKUP SYSTEM utility. Object-level recovery requires a minimum of z/OS V1R8 and can be performed with the RECOVER utility. You have the ability to recover to a point in time or to a current point. The utility uses the previous system-level backup or image copy.

**Important**: In order to perform object level recovery, you must specify YES for SYSTEM-LEVEL BACKUPS on the DSNTIP6 panel.

**Recommendations**

The BACKUP SYSTEM utility gives quite a bit of flexibility when it comes to tape control. In particular, the FORCE option allows the initiation of a new backup to start even though a previous DUMP has not finished. We recommend that you use only the FORCE option if it is more important to take a new system-level backup than to save a previous system-level backup to tape.
Furthermore, the DUMP option allows FlashCopy and disk-to-tape copies to overlap. As a consequence, these options trigger additional I/O on the system. We recommend that you control these events by verifying the DUMP status through the use of the DFSMS/hsm command LIST COPYPOOL with the DUMPVOLS option.

If you need system-level backups to be dumped to tape, invoke BACKUP SYSTEM twice. The first time that you run the utility, you invoke FlashCopy. Then, you run the utility again with the DUMPONLY option after the physical background copy for the FlashCopy has completed.

For object-level recovery, if underlying data sets are deleted or moved to a different set of volumes, this type of recovery is not possible. We recommend that you take in-line image copies with the REORG utility and then force object-level recovery after moving DB2 page sets.

### 6.4.2 RECOVER utility enhancements for point-in-time recovery

Performance can be considerably improved by enabling the fast log apply function on the DB2 subsystem. The RECOVER utility automatically uses the fast log apply process during the LOGAPPLY phase if the fast log apply function has been enabled. When the recovery completes, all storage used by the fast log apply function is returned to the DBM1 address space. Both copies of the log can be used, and the buffer default for DB2 V9 has been increased from 100 MB to 500 MB. However, we recommend that you do not increase the number of concurrent recovery jobs per member.

The RECOVER utility recovers data to the current state or to a previous point in time by restoring a copy and then applying log records. In the versions previous to DB2 V9, a point-in-time recovery could cause a data inconsistency problem since the point recovered to was not a consistent point. This is because there was no process to back out in-flight units of recovery (UR). As a result, it was recommended to take QUIESCE points to eliminate this problem. However, running the QUIESCE utility yielded the following problems:

- It reduced access to “read-only” on the objects that are being quiesced. This created an immediate impediment to applications that were running on high volume systems.
- Frequently, running this utility produced unwanted overhead on production systems.

In reality, the execution of many point-in-time recoveries are performed to unplanned points in time. This introduces manual intervention since the data that is left in an inconsistent state must be repaired. This error prone method proves to be quite time consuming and requires a much deeper knowledge of DB2.

**Attention:** Use the FORCE option with extreme caution since the oldest version of the database or log copy pool is overwritten with another version before it has a chance to be saved to tape. You will lose the oldest version of the copy pool or pools.
DB2 9 for z/OS takes considerable measures to reduce the time that is required for manual interventions to perform such an operation. The following steps are improved during the RECOVER to a point in time:

1. Automatically detect the uncommitted transactions that are running at the point-in-time recovery point.

2. Roll back changes on the object to be recovered to ensure data consistency after the point-in-time recovery. No fast log apply function is used. Here are the particulars:
   a. Changes made on the recovered objects by URs that are INFLIGHT, INABORT, POSTPONED ABORT during the recovery time are rolled back.
   b. URs that are INDOUBT during the recovery point are treated as INABORT, and their changes on the recovered objects are also rolled back.
   c. INCOMMIT URs are treated as committed, and no rollback occurs.
   d. If a UR changes multiple objects in its life span:
      i. Only changes made by it on the objects that are being recovered are rolled back.
      ii. Changes made by it on other objects are not rolled back.

3. Leave the recovered objects in a consistent state from a transaction point of view.

**Note:** Consistency is not guaranteed when recovering to a specified image copy (TOCOPY, TOLASTCOPY, and TOLASTFULLCOPY of RECOVER options).

DB2 objects can now be recovered to any previous point in time with full data integrity. In addition, these improvements allow you to avoid running the QUIESCE utility regularly so you can reduce the disruption to DB2 users and applications. Recovery to a point in time with consistency is available only in new-function mode to avoid issues with coexistence environments.

**Important:** You must include all associated objects in the same RECOVER utility job to ensure data consistency from the application point of view. If the object or objects are not specified in the same list, the utility sets the appropriate prohibitive pending state. For example, if the parent table space is recovered to a point in time, but its dependent table space is not recovered in the same list, the dependent table space is placed in a CHKP (check pending) state.

**Recommendations**

The ability to recover to any prior point in time is now possible with the enhancements made to the RECOVER utility. As mentioned earlier, there can be an extensive benefit from a performance perspective if the fast log apply function is activated on the DB2 subsystem. The RECOVER utility automatically uses the fast log apply process during the LOGAPPLY phase if the fast log apply function has been enabled. After the recovery completes, all storage that is used by the fast log apply function is returned to the DBM1 address space.

Due to the ability to back out URs when recovering to any point in time, we suggest that you run the QUIESCE utility less frequently to further increase your service availability. The only reasons to run QUIESCE now are:

- To mark in SYSCOPY a point in time that has significance, such as the beginning or end of major batch processing
- To create a common sync-point across multiple systems, for example, across IMS and DB2
In addition, the option RESTOREBEFORE has been added to the RECOVER utility. This option allows you to search for an image copy, concurrent copy, or system-level backup with an relative byte address (RBA) or log record sequence number (LRSN) value earlier than the specified 'X'byte-string' value. We recommend that you exercise the use of this feature. For example, if you know that you have a broken image copy, you can direct the RECOVER utility to a previous full or incremental copy and perform a LOGAPPLY from that point onward.

Incremental FlashCopy

Significant developments have been made to support incremental FlashCopy. In addition to z/OS Version 1 Release 8 DFSMShsm, this function requires APAR OA17314 for z/OS. The incremental in FlashCopy is much different from the incremental in image copies because no merge with the full image copy is required. Note that incremental FlashCopy does not reduce the need for disk volumes, unlike an incremental image copy potentially does. For more details about this enhancement, see DB2 9 for z/OS Technical Overview, SG24-7330.

6.5 Online REBUILD INDEX enhancements

The REBUILD INDEX utility reconstructs indexes or index partitions from the table that they reference. Prior to V9, users often created indexes with DEFER YES and then used REBUILD INDEX to finish the creation of the index to avoid large sorts using the sortwork database and to decrease the outage window.

In DB2 V9 for z/OS, the REBUILD INDEX utility has been extended to support read and write access (new SHRLEVEL CHANGE option) for a longer period of time during the execution of the utility (available in new-function mode). It now has a log phase, and there are DRAIN WAIT options similar to the other utilities such as REORG, CHECK DATA, and so on. As a result, applications have greater access to data while indexes on that data are being rebuilt. This enhancement is complemented with the V8 features in which insert, update, and delete operations are supported on indexes that are non-unique, while the index is in the process of rebuilding.

Note: REBUILD INDEX with the SHRLEVEL CHANGE option is invalid for indexes on XML tables.
6.5.1 Performance

During the execution of REBUILD INDEX in V8, the entire table space was read only and all indexes were unavailable. In DB2 V9, this utility proves to be more resilient in terms of performance and availability compared to V8.

Thus, four cases were tested to recognize the performance benefits that accompany this release. The online REBUILD INDEX utility was tested. Table 6-10 summarizes the details.

Table 6-10  Details of the workload used for online REBUILD INDEX

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table and index details</th>
<th>Index to be rebuilt</th>
<th>Control statements</th>
</tr>
</thead>
</table>
| 1           | 10 partitions           | 1 PI                | 1. REBUILD INDEX (with V8 defaults)  
|             | 26 columns with 118 byte length |                     | 2. REBUILD INDEX SHRLEVEL NONE (with V9 defaults)  
|             | 50 million rows         |                     | 3. REBUILD INDEX SHRLEVEL REFERENCE (with V9 defaults)  
|             | 1 partitioning index    |                     | 4. REBUILD INDEX SHRLEVEL CHANGE (with V9 defaults)  
|             | 5 NPIs                  |                     |                    |
| 2           | Same as Case 1          | 1 NPI               | 1. REBUILD INDEX (with V8 defaults)  
|             |                        |                     | 2. REBUILD INDEX SHRLEVEL NONE (with V9 defaults)  
|             |                        |                     | 3. REBUILD INDEX SHRLEVEL REFERENCE (with V9 defaults)  
|             |                        |                     | 4. REBUILD INDEX SHRLEVEL CHANGE (with V9 defaults)  
| 3           | Same as Case 1          | 1 NPI part          | 1. REBUILD INDEX (with V8 defaults)  
|             |                        |                     | 2. REBUILD INDEX SHRLEVEL NONE (with V9 defaults)  
|             |                        |                     | 3. REBUILD INDEX SHRLEVEL REFERENCE (with V9 defaults)  
|             |                        |                     | 4. REBUILD INDEX SHRLEVEL CHANGE (with V9 defaults)  
| 4           | Same as Case 1          | 1 PI 5 NPIs         | 1. REBUILD INDEX (with V8 defaults)  
|             |                        |                     | 2. REBUILD INDEX SHRLEVEL NONE (with V9 defaults)  
|             |                        |                     | 3. REBUILD INDEX SHRLEVEL REFERENCE (with V9 defaults)  
|             |                        |                     | 4. REBUILD INDEX SHRLEVEL CHANGE (with V9 defaults)  |
Figure 6-14 shows the results of this comparison.

From Figure 6-14, it is interesting to see that, when rebuilding a partitioning index, there is an actual 18% decrease in both CPU time and elapsed time when comparing the “V8 BASE” (equivalent to SHRLEVEL REFERENCE) and the “V9 SHRLEVEL CHANGE”. In the case of rebuilding the NPI, you can also see a 5% decrease in CPU time in the same comparison. When rebuilding all indexes, there is a 12% decrease in CPU time when comparing V8 to V9 SHRLEVEL CHANGE. Furthermore, when rebuilding all indexes, the elapsed time experienced a 7% reduction when comparing V8 to V9 (with the SHRLEVEL CHANGE option).

6.5.2 Conclusion

Not only does the new SHRLEVEL CHANGE option in DB2 V9 allow increased availability, but there is a performance benefit when using the REBUILD INDEX utility compared to V8. The improvements seen here are due to the index block interface improvements. However, as you increase the number of indexes, the improvements are reduced due to the impact of the UNLOAD and SORT phases. The SORT phase initiates the DFSORT subtasks that result from increased CPU time. To counter this effect, a type of parallel processing is initiated (on the same table space) when rebuilding PIs and NPIs that results in a decrease in the size of the sort data set, as well as the total time that is required to sort all the keys.

6.5.3 Recommendations

Significant improvements have been made to the REBUILD INDEX utility to increase availability with the SHRLEVEL CHANGE option. We strongly recommend that you run this utility with the SHRLEVEL CHANGE option during light periods of activity on the table space.
or index. Avoid scheduling REBUILD INDEX with SHRLEVEL CHANGE when critical applications are executing.

The online REBUILD INDEX utility is not well suited for unique indexes and concurrent XML because the index is placed in RBDP while it is being built. Inserts and updates to the index fail with a resource unavailable (SQL CODE -904) because uniqueness checking cannot be done while the index is in RBDP.

**Note:** REBUILD INDEX with the SHRLEVEL CHANGE option is invalid for indexes over XML tables. Also, this option is not supported for tables that are defined with the NOT LOGGED attribute, XML indexes, or spatial indexes.

REBUILD INDEX SHRLEVEL CHANGE should only be used to fix a broken or restricted index, or to build an index after DEFER. You should not use the REBUILD INDEX SHRLEVEL CHANGE utility to move an index to different volumes; instead use the online REORG utility.

**Important:** REBUILD INDEX with the SHRLEVEL CHANGE option cannot be run to rebuild indexes on the same table space concurrently. To circumvent this restriction, REBUILD INDEX can build indexes in parallel by specifying multiple indexes in a single utility statement to allow DB2 to control the parallelism. You are still allowed to concurrently rebuild indexes in different tables spaces, as is the concurrency in rebuilding different partitions of an index in a partitioned table space.

### 6.6 Online REORG enhancement

In today’s fast-paced, business world, important information is at the forefront and must be continuously available in order to meet industry demands. DB2 V9 embodies this requirement through significant enhancements made to its online REORG utility. A substantial improvement has been made with the removal of the BUILD2 phase to reduce the outage time. In addition, a significant reduction in elapsed time is seen in both the UNLOAD and RELOAD phases through the use of parallelism.

The BUILD2 phase affects REORG at a partition level (REORG TABLESPACE PART n SHRLEVEL REFERENCE or CHANGE). Prior to DB2 V9, when you reorganized a table space with SHRLEVEL CHANGE, the data that is undergoing reorganization is not available to applications during the SWITCH and BUILD2 phases. (Data is read-only during the last iteration of the LOGAPPLY phase.) As a result, if you want to run an online REORG at the partition level (does not apply at the table-space level), and you have one or more NPIs, then you have to enter this BUILD2 phase. The BUILD2 phase is at the end because you have to deal with the NPIs, which result in an outage.

The BUILD2 phase has the following characteristics in versions prior to DB2 V9:

- Logical parts of any NPIs are updated from the shadow NPI data sets with the new record identifiers (RIDs) that are generated by REORG.
- The entire NPI is unavailable for write operations.
- The outage is additional to the outage that is required for the SWITCH phase.
In DB2 V9, actions were taken to eliminate the BUILD2 phase to increase availability (not optional). The following elements make up the new REORG utility without the BUILD2 phase:

- A complete copy of each NPI is created so that it can be switched.

  During the UNLOAD phase, the utility attaches subtasks to unload the NPIs and builds the shadow data set which must be as large as its corresponding NPI (not just the LPAR that is being reorganized). These subtasks appear in the output of the -DIS UTIL command with a phase name of ‘UNLOADIX’.

  The index build tasks that started during the RELOAD phase (sort, build, and inline statistics) in versions prior to DB2 V9 are now started during the UNLOAD phase.

- REORG must now handle updates to parts of the NPI that are not being reorganized. Therefore REORG SHRLEVEL REFERENCE (or CHANGE) PART n for a partitioned table space with one or more NPIs now has a LOG phase.
  - The utility attaches one or more subtasks during the LOG phase to speed up processing of the log records. These subtasks appear in the output of the -DIS UTIL command with a phase name of LOGAPPLY.
  - During the last iteration of the LOG phase, all logical parts of the NPI are read-only to applications (UTRO).

  Note: The –ALTER UTILITY command can be used to modify the LOG phase parameters. See the DB2 Version 9.1 for z/OS Command Reference, SC18-9844, for details about using this command.

- During the SWITCH phase, all logical parts of the NPIs will be unavailable (UTUT) to applications.

  This may affect applications that access partitions that are not being reorganized.

## 6.6.1 Performance

In order to realize the improvements that have been made to the online REORG utility in DB2 V9, a test was conducted using three different configurations. This test measures the CPU time and elapsed time of the REORG utility for different indexes. All measurements were performed using a System z9 processor with z/OS 1.8 and DS8300 disks. Each table in the table space that is reorganized contains 50 million rows.
Three separate configurations (varying NPIs) were tested. Table 6-11 shows a summary of the results.

**Table 6-11  Online REORG with 1, 2, and 5 NPIs defined**

<table>
<thead>
<tr>
<th>NPIs</th>
<th>CPU time</th>
<th>Elapsed time</th>
<th>CPU time</th>
<th>Elapsed time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 NPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REORG PART 1 SHRLEVEL CHANGE</td>
<td>-3%</td>
<td>-48%</td>
<td>+96%</td>
<td>+34%</td>
</tr>
<tr>
<td>REORG PART 1:4 SHRLEVEL CHANGE</td>
<td>-22%</td>
<td>-58%</td>
<td>+7%</td>
<td>-32%</td>
</tr>
<tr>
<td>2 NPIs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REORG PART 1 SHRLEVEL CHANGE</td>
<td>-7%</td>
<td>-42%</td>
<td>+113%</td>
<td>+35%</td>
</tr>
<tr>
<td>REORG PART 1:4 SHRLEVEL CHANGE</td>
<td>-30%</td>
<td>-46%</td>
<td>+6%</td>
<td>-31%</td>
</tr>
<tr>
<td>5 NPIs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REORG PART 1 SHRLEVEL CHANGE</td>
<td>-8%</td>
<td>-8%</td>
<td>+126%</td>
<td>+116%</td>
</tr>
<tr>
<td>REORG PART 1:4 SHRLEVEL CHANGE</td>
<td>-39%</td>
<td>-17%</td>
<td>+3%</td>
<td>+13%</td>
</tr>
</tbody>
</table>

The results in Table 6-11 demonstrate a consistent trend when reorganizing a partitioned table space with SHRLEVEL CHANGE. Reorganizing one partition yields an average of a 6% reduction in CPU time for a 10-partitioned table space. In addition, there is an average of a 33% reduction in elapsed time for the same table space. When reorganizing a set of four partitions, there is an average of a 30% reduction in CPU time and an average of a 40% reduction in elapsed time. However, the CPU time and elapsed time for a table space that contains 50 partitions are subjected to a regression when reorganizing a single partition or a set of four partitions.

### 6.6.2 Conclusion

To maximize availability, the online REORG process has been enhanced through the removal of the BUILD2 phase. In addition to the elimination of the BUILD2 phase, the elapsed time reduction is achieved by the parallelism in the UNLOAD and RELOAD phases as well as the parallel tasks that are invoked in the LOGAPPLY phase. However, as the number of partitions for a table space increase, an increase in both CPU time and elapsed time is experienced when few partitions are reorganized. Also, as the size and number of NPIs increase on a table, a degradation in both CPU time and elapsed time is experienced.

Therefore, in consideration of industry requirements, the changes to the online REORG in this release were strategically implemented to grant maximum availability for your critical
e-business applications. In addition, the output of an online REORG yields the reorganization of all of your NPIs that are defined on that table.

6.6.3 Recommendations

To further minimize the elapsed time costs in the online REORG, we strongly recommend that you reorganize a set of contiguous partitions. If contiguous partitions are being reorganized and they are specified in a range in the control card (for example, PART4:8), then parallelism will automatically occur within the utility. However, if you run a REORG TABLESPACE SHRLEVEL CHANGE (or REFERENCE) PART x job concurrently with a REORG TABLESPACE SHRLEVEL CHANGE (or REFERENCE) PART y job for the same table space, and at least one NPI exists, the second job that starts will fail (message DSNU180I). This is largely due to the launching of the parallel tasks in the LOGAPPLY phase. You must run these jobs serially since the entire NPI of the table will be rebuilt.

Since the BUILD2 phase has been removed, you must take into account the additional storage that will be required for the whole NPI or NPIs of the table. In addition, the CPU cost of the REORG will increase as a result of rebuilding of the whole NPI.

Note: Prior to V9, a limit of 254 parts per REORG on a compressed table space existed. In V9, this restriction has been lifted because the storage consumption has been reduced for compressed partitioned table spaces.

The unload and reload of the whole NPI during a REORG SHRLEVEL CHANGE (or REFERENCE) PART is essentially equivalent to a REORG INDEX of the NPI. If you currently run REORG INDEX on all NPIs following a REORG PART, this should no longer be needed. As a result, a V9 online REORG is equivalent to a V8 online REORG plus a REORG INDEX of the NPIs. Table 6-12 shows the measurements for a 100 partition table space. This table illustrates this considerable elapsed time improvement with the number of NPIs that are defined on the table.

Table 6-12  Elapsed time comparison between V9 OLR and V8 OLR with REORG NPI or NPIs

<table>
<thead>
<tr>
<th>Number of NPIs</th>
<th>V8 partition OLR + REORG NPI or NPIs</th>
<th>V9 partition OLR</th>
<th>Delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64.7 sec.</td>
<td>39.3 sec.</td>
<td>-39.3%</td>
</tr>
<tr>
<td>2</td>
<td>131.3 sec.</td>
<td>74.4 sec.</td>
<td>-43.3%</td>
</tr>
<tr>
<td>5</td>
<td>321.6 sec.</td>
<td>168.2 sec.</td>
<td>-47.7%</td>
</tr>
</tbody>
</table>

From Table 6-12, it is apparent that a DB2 9 online REORG performs a reorganization of all of its NPIs. This translates into a colossal 43% improvement in elapsed time when compared to V8. Hence, we strongly recommend that you discontinue the execution of REORG INDEX of your NPIs following an online REORG PART.

6.6.4 Online LOB REORG

In this release, improvements to the REORG utility for LOB table spaces have been made to increase availability. Prior to DB2 9, there was no access to the LOB during the execution of the utility since the equivalent access was that of SHRLEVEL NONE. No space was reclaimed upon completion of the REORG since the existing VSAM data set was used. In addition, LOG NO was not allowed, which resulted in additional logging, particularly for LOB table spaces that were defined as LOG YES.
DB2 V9 overcomes the previous limitations with increased availability through the inception of SHRLEVEL REFERENCE, and physical disk space is now reclaimed. This is possible through the use of shadow data sets because they are required in the same way as any other REORG using SHRLEVEL REFERENCE. This implementation is available in DB2 9 conversion mode.

To allow read access to the LOB during the execution of the REORG utility, the original LOB table space must be drained of all writers. LOBs are implicitly reorganized during the LOB allocation to the shadow data set.

**Note:** Contrary to regular table spaces, no sorting takes place during the copying of the original data set to the shadow data set.

When this operation is complete, all access to the LOB table space is stopped (the readers are drained), while the original data set is switched with the shadow data set. At this point, full access to the new data set is enabled, and an inline copy is taken to ensure recoverability of data.

For more information about reorganizing LOBs, see *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270.

**Recommendations**
Since REORG SHRLEVEL REFERENCE now can reclaim space, we recommend that you run REORG when:

- Syslobstats.Freespace(in KB) / Systablespace.Nactive(#pages)*Pgsize(in KB) >50%

### 6.7 Online CHECK DATA and CHECK LOB

In DB2 V9 for z/OS, considerable changes have been made to increase application availability. In particular, both the CHECK DATA and CHECK LOB utilities are now able to execute using the SHRLEVEL CHANGE option. In V8, you had read-only access only during the duration of the utility. This type of access is now the default in DB2 V9 with the SHRLEVEL REFERENCE option (available only in new-function mode). As a result, this release offers maximum control of access to your data when you run the CHECK DATA and CHECK LOB utilities.

#### 6.7.1 Online CHECK DATA

The CHECK DATA utility ensures that table spaces are not violating referential and table check constraints and reports information about violations that it detects. CHECK DATA checks for consistency between a base table space and the corresponding LOB or XML table spaces.

**Note:** CHECK DATA does not check LOB or XML table spaces.
Here is a breakdown to illustrate the difference between the SHRLEVEL options for the CHECK DATA utility in DB2 V9:

- SHRLEVEL REFERENCE is the default and allows applications to read from, but cannot write to, the index, table space, or partition that is checked.
- SHRLEVEL CHANGE specifies that applications can read from and write to the index, table space, or partition that is checked.

The new SHRLEVEL CHANGE option operates on shadow data sets. These data sets must be preallocated for user-managed data sets prior to executing the utility. By contrast, if a table space, partition, or index resides in DB2-managed data sets and shadow data sets do not already exist when you execute the utility, DB2 automatically creates the shadow data sets. In both cases, the copy is taken by DB2 using the DFSMS ADRDSSU utility.

**Note:** The utility must be able to drain the objects ahead of the copy. DRAIN WAIT options are available if you want to override the IRLMRWT and UTIMOUT subsystem parameters.

At the end of CHECK DATA processing, the DB2-managed shadow data sets are deleted. If inconsistencies are found, no action can be taken since the utility runs on the shadow table space. Therefore, if the utility detects violations with the data that it is scanning, the CHECK-pending state is not set and avoids the removal of access from the whole table space. Instead, the output generates REPAIR utility statements to delete the invalid data in the live table space.

### 6.7.2 Online CHECK LOB

As with the CHECK DATA utility, DB2 V9 introduced the SHRLEVEL REFERENCE and SHRLEVEL CHANGE options for the CHECK LOB utility. This utility identifies any structural defects in the LOB table space and any invalid LOB values. Prior to this release, you ran CHECK LOB with read-only access (equivalent to SHRLEVEL REFERENCE) to the LOB table space. Therefore, the CHECK LOB utility is now equipped with the SHRLEVEL REFERENCE and SHRLEVEL CHANGE options with the following functions:

- SHRLEVEL REFERENCE is the default and allows applications to read from, but not write to, the LOB table space that is to be checked.
- SHRLEVEL CHANGE specifies that applications can read from and write to the LOB table space that is to be checked.

The execution of the CHECK LOB utility with the new SHRLEVEL CHANGE option engages the use of shadow data sets. For user-managed data sets, you must preallocate the shadow data sets before you execute CHECK LOB SHRLEVEL CHANGE. However, if a table space, partition, or index resides in DB2-managed data sets and shadow data sets do not already exist when you execute CHECK LOB, DB2 creates the shadow data sets. In both cases, the copy is taken by DB2 using the DFSMS ADRDSSU utility.

**Note:** The utility must be able to drain the objects ahead of the copy. DRAIN WAIT options are available if you want to override the IRLMRWT and UTIMOUT subsystem parameters.

At the end of CHECK LOB processing, the DB2-managed shadow data sets are deleted. After successful execution of the CHECK LOB utility with the SHRLEVEL CHANGE option, the CHECK-pending (CHKP) and auxiliary-warning (AUXW) statuses are not set or reset.
6.7.3 Recommendations

The ability to execute the CHECK DATA and CHECK LOB utilities with the SHRLEVEL CHANGE option results in increased availability. As a result, we strongly recommend that you use the IBM FlashCopy V2 feature to maximize performance and accentuate availability since the data is available while the utility is performing I/O operations.

**Attention:** Do not to run CHECK DATA on columns that are encrypted via DB2 built-in functions. Since CHECK DATA does not decrypt the data, the utility might produce unpredictable results.

Also, if a table space contains at least one LOB column, run the CHECK LOB utility before the CHECK DATA utility.

If you need to further maximize your availability, you can create the shadow data sets of the objects that are managed by DB2 prior to the execution of the CHECK DATA and CHECK LOB utilities. This minimizes read-only access to the table space at the time of the copy, especially for the data sets of the LPAR of NPIs. See *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855, for details about creating shadow data sets.

6.8 TEMPLATE switching

The TEMPLATE utility control statement lets you allocate data sets, without using JCL DD statements, during the processing of a LISTDEF list. Prior to V9, the TEMPLATE control statement allows you to define the data set naming convention with additional options, such as data set size, location, and attributes, giving you significant flexibility.

In DB2 9 (conversion mode), TEMPLATE is further enhanced by introducing a feature that allows you to switch templates. The new template switching function allows image copies of varying sizes to have different characteristics. It gives you the ability to direct smaller image copy data sets or inline copy data sets to DASD and larger data sets to tape. Additionally, it allows you to switch templates that differ in ways other than just their UNIT attributes, that is hierarchical storage management (HSM) classes (management class, storage class).

In order to exploit this feature, a new keyword, LIMIT, has been added to the TEMPLATE utility control statement. See *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855, for details about the new syntax.

There are two control parameters within the TEMPLATE statement:

- The maximum primary allocation allowed for that template
- The template name to be switched to when the maximum is reached

**Note:** You can switch templates only once per allocation. Multiple switching does not take place. For example, it is not possible to have a small, medium, and large template.

6.8.1 Performance

The TEMPLATE utility, as well as JCL DDs, uses a different interface (large block) for the COPY and RECOVER utilities when using tapes. As a result, the large block interface allows a block size greater than 32,760 bytes for tapes. This translates into up to a 40% reduction in elapsed time for the COPY and RECOVER utilities. In addition, the TEMPLATE utility supports reading large format data sets to allow greater than 65,535 tracks per DASD volume.
making it quite useful when copying large table spaces. However, the TEMPLATE utility does not create data sets with the LARGE format attribute.

In addition, the switching decision is made based on the “estimated” output data set size. This may differ from the actual final size of the output data set. This difference is particularly true for incremental image copies that are estimated at 10% of the space required for a full image copy.

6.9 LOAD COPYDICTIONARY enhancement

New function to enable LOAD COPYDICTIONARY option to allow compression dictionaries to be copied from one partition to another in a classic partitioned or partition-by-range table space.

This option provides a method to copy a compression dictionary to an empty partition that normally wouldn't have a compression dictionary built yet.

The statement to copy a compression dictionary from physical partition 1 to partitions 3 and 5 looks like Example 6-1.

Example 6-1 LOAD COPYDICTIONARY example

```
LOAD RESUME NO COPYDICTIONARY 1
           INTO TABLE PART 3 REPLACE
           INTO TABLE PART 5 REPLACE
```

6.10 COPY performance

The COPY utility has been modified in DB2 V9 to always check the validity of each page of the object. Prior to DB2 V9, the CHECKPAGE option is always recommended to be performed. However, there are two main disadvantages:

- The CPU overhead is relatively expensive with up to 14% for COPY TABLESPACE (DB2 V8).
- If the COPY utility finds a broken page, it places the table space or index space into COPY PENDING state and makes it inaccessible.

With DB2 V9, drastic measures have been taken to reduce the CPU overhead for COPY TABLESPACE with CHECKPAGE to be almost negligible. This was primarily accomplished by improving the communication between the batch and DBM1 address space to help reduce overall consumption. A change to the least recently used (LRU) algorithm in the buffer pool for COPY now prevents the content of the buffer pool from being dominated by the image copy.

In V9, the CHECKPAGE option is always in operation (not optional) for a COPY TABLESPACE to ensure maximum integrity. The COPY utility performs validity checking for each page and issues a message if an error is found to identify the broken page and the type of error. If more than one error exists on a page, only the first error is identified. The utility continues to check the remaining pages in the table space or index space after it finds an error, but no copying takes place. When the error is detected, the table space or index space is not put in COPY pending state. Instead, a return code of 0008 is issued. A new SYSIBM.SYSCOPY record type is written in order to subsequently force a full image copy (since dirty bits may have already been flipped off in the space map pages). See DB2 9 for z/OS Technical Overview, SG24-7330, for more details.
COPY on indexes needs you to specify the CHECKPAGE option explicitly to activate the full page index checking.

**Note:** You will be able to work with the broken object, with the exception of the broken pages. However, we recommend that you fix the problem as soon as possible.

In addition, DB2 V9 delivers SCOPE(PENDING) support, where objects are copied only if they are in COPY pending status or ICOPY pending status.

**Workload: Comparison of COPY TABLESPACE in V8 and V9**

In order to analyze the functionality of the embedded CHECKPAGE option in DB2 V9, the COPY TABLESPACE utility was executed and compared to DB2 V8. The test was conducted using a System z9 processor and DS8300 disks running z/OS 1.8.

Table 6-13 contains a summary of the different test cases that were used.

<table>
<thead>
<tr>
<th>Case number</th>
<th>Table or index description</th>
<th>Control statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10 partitions</td>
<td>COPY TABLESPACE</td>
</tr>
<tr>
<td></td>
<td>26 columns with 118 byte</td>
<td>(with defaults)</td>
</tr>
<tr>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>50 million rows</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 partitioning index</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 NPIs</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Same as Case 1</td>
<td>COPY CHECKPAGE</td>
</tr>
<tr>
<td></td>
<td>(with defaults)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Same as Case 1</td>
<td>COPY SHRLEVEL CHANGE</td>
</tr>
<tr>
<td></td>
<td>(with defaults)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Same as Case 1</td>
<td>COPY SHRLEVEL CHANGE CHECKPAGE</td>
</tr>
<tr>
<td></td>
<td>(with defaults)</td>
<td></td>
</tr>
</tbody>
</table>

The results are shown in Figure 6-15. Figure 6-15 indicates that, in Case 1, the CHECKPAGE option (active by default in V9) results in a savings of 4% CPU time. When CHECKPAGE is specified for both V8 and V9, V9 shows an improvement in CPU of 19%. SHRLEVEL CHANGE (case 3) is similar to case 1. When invoking the CHECKPAGE in both V8 and V9 with a SHRLEVEL CHANGE option, again there is a 19% reduction in CPU time. With V9, there is a slight increase in elapsed time in case 3, but there is a 19% decrease in elapsed time in case 4.
Conclusion
It is evident that you can always check the validity of your data with the integration of the CHECKPAGE option in DB2 V9. In fact, the COPY TABLESPACE utility (with the integrated CHECKPAGE option) in V9 performs equally or slightly better than the COPY TABLESPACE utility (with and without the CHECKPAGE option) in V8.

Recommendations
When the COPY utility detects a broken page, the table space (or index space) is not placed in COPY pending. As a result, we recommend that you check the return code (0008) of your job and fix the underlying problem. In addition, when the problem is fixed, you will be unable to take an incremental image copy. Since the utility found a broken page, SYSIBM.SYSCOPY is updated to reflect the erroneous status of the object. This indicator forces you to take a full image copy of the object.

6.11 Best practices

With the new enhancements that are made from release to release in DB2, it is important to keep up with the new features that are offered. In particular, the utilities are packaged with new options to boost performance and promote simplicity. As a result, the following sections provide an overview of general recommendations that you should exercise.
6.11.1 Recommendations for running the LOAD utility

You should exercise the following general best practices when running the LOAD utility:

- Use the LOG NO option to reduce log volume and, in cases of extremely high logging rates, reduce log latch contention problems. It requires an image copy; inline copy is a good choice.
- Use the KEEPDICTIONARY option, but track the dictionary effectiveness by checking the PAGESAVE column of SYSIBM.SYSTABLEPART.
- Use inline COPY and inline STATISTICS for CPU savings.
- Use index parallelism by using the SORTKEYS option (default as of V8).
  - On LOAD, provide an argument for SORTKEYS only when the input is tape/PDS member.
  - Remove SORTWKxx DD statements and use SORTDEVT/SORTNUM.
- When using DISCARD, avoid having the input on tape.
  Input has to be re-read to discard errant records.
- Avoid data conversion; use matching representation whenever possible.
- Sort data in clustering order, unless data is randomly accessed via SQL, to prevent the need for a subsequent REORG TABLESPACE.

6.11.2 Recommendations for running the REBUILD INDEX utility

Exercise the following general best practices when you run the REBUILD INDEX utility:

- Since index parallelism is invoked through the default SORTKEYS option, remove SORTWKxx DD statements and use SORTDEVT/SORTNUM.
- Use inline COPY and inline STATISTICS for CPU savings.

6.11.3 Recommendations for running the REORG utility

Follow these general best practices when you run the REORG utility:

- Use the LOG NO reduce log volume and, in cases of extremely high logging rates, reduce log latch contention problems. It requires an image copy; inline copy is a good choice.
- Use the KEEPDICTIONARY option but track the dictionary effectiveness by checking the PAGESAVE column of SYSIBM.SYSTABLEPART.
- Use inline COPY and inline STATISTICS for CPU savings.
- Use the NOSYSREC option to avoid I/O, which is always used for SHRLEVEL REFERENCE and CHANGE.
  Use this option with SHRLEVEL NONE only if you are taking a full image copy prior to running the REORG.
- Use the SORTKEYS option to invoke index parallelism if running on DB2 V7.
  Remove SORTWKxx DD statements and use SORTDEVT/SORTNUM.
- We strongly recommend that you use inline COPY and inline STATISTICS since they save an additional read on the data.
- For LOBs:
Prior to V9, REORG SHRLEVEL NONE is the only option possible. It only re-establishes the sequence on chunks (groups of contiguous 16 pages), but there is no space reclamation. We do not recommend this option.

With V9, REORG SHRLEVEL REFERENCE re-establishes the chunks and reclaims space. Consider running REORG of an LOB table space for either of the following real-time statistics (RTS) conditions:

- SYSTABLESPACESTATS.SPACE>2*DATASIZE/1024 in RTS based on available reclaimable free space
- REORGDISORGLOB/TOTALROWS>50% in RTS based on how pages for a given LOB (chunks) are close to each other

### Recommendations for running online REORG

For REORG PART with the SHRLEVEL CHANGE option, Table 6-14 lists our recommendations for achieving optimal performance.

**Table 6-14   Best practices for REORG PART with SHRLEVEL CHANGE**

<table>
<thead>
<tr>
<th>Option</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEOUT TERM</td>
<td>Frees the objects if time-outs that occur when draining</td>
</tr>
<tr>
<td>DRAIN ALL</td>
<td>Better chance of successful drain</td>
</tr>
<tr>
<td>MAXRO = IRLMRWT minus 5 to 10 seconds</td>
<td>Prevents time-outs</td>
</tr>
<tr>
<td>DRAIN_WAIT = IRLMRWT minus 5 to 10 seconds</td>
<td>Prevents time-outs</td>
</tr>
<tr>
<td>RETRY = UTIMOUT (DSNZPARM)</td>
<td>Default</td>
</tr>
<tr>
<td>URLGWTH (DSNZPARM) and activate IFCID313 (Statistics Class 3)</td>
<td>Enable detection of long running readers and report readers that may block commands and utilities from draining</td>
</tr>
</tbody>
</table>

### 6.11.4 Recommendations for running the COPY utility

Consider these best practices when running the COPY utility:

- **PARALLEL** keyword provides parallelism for lists of objects.
- You can speed up your copies by taking incremental copies if:
  - Less than 5% of pages are randomly updated. Typically this means that less than 1% of rows are updated.
  - Less than 50% of pages are sequentially updated such that updated pages are together and separated from non-updated pages.
- Copy indexes on your most critical tables to speed up recovery.

### 6.12 Best practices for recovery

You can find a complete description of best practices for recovery in *Disaster Recovery with DB2 UDB for z/OS*, SG24-6370. Here is an excerpt that summarizes some key areas of interest:
1. The first phase of recovery is the restore of a full image copy followed by any available incremental image copies. There are basically three ways to improve the performance of the restore phase:
   – Increase the speed with which the image copy is read. One example is data striping of the image copy.
   – Decrease the size of the image copy data set via compression.
   – Restore image copies in parallel.

2. The second phase of recovery is to apply any updates to the table space done after the last incremental image copy was taken. The time to read log records can be reduced by:
   – Increasing the log transfer rate
     One example is data striping of the log. You can also stripe the archive logs with V9. Do not stripe the archive log if you are running on V8; DB2 will not be able to read it natively.
   – Reducing the amount of log data to be read in order to get the relevant records
     One example is the DB2 registration of log record ranges in SYSIBM.SYSLGRNX.
   – Reading log records only once while using them for multiple table space or partition recoveries
     One example is fast log apply.

Applying the log records is a question of normal DB2 performance tuning even though some special considerations apply. Several features and options are available to reduce the elapsed time of the recovery. These include:

- Using a LISTDEF for the RECOVER utility
- Restoring image copies in parallel
- Skipping log data sets without relevant data
- Taking image copies of indexes

6.12.1 Recommendations for fast recovery

The following options are available to obtain the fastest possible recovery:

- Allow the Recovery utility to restore table spaces in parallel.
- Place the image copy on disk for faster restore time.
- Take frequent image copies.
- Consider using incremental image copies.
- Consider use the Merge Copy utility when producing incremental image copies.
- Consider using Copy and Recover for large indexes
- Consider running recovery jobs in parallel, but no more than ten per DB2 subsystem or member.
- Exploit fast log apply.
- Exploit parallel index rebuild.
- Archive logs to disk.
- Specify log data set block size as 24576 bytes.
- Maximize data available from active logs.
- Increase accuracy of the DB2 log range registration.
- Consider using I/O striping for the active logs.
- Consider using the DB2 Log Accelerator tool to provide striping and compression support for the archive logs.
- Evaluate data compression for reducing log data volumes.
- Tune your buffer pools.

### 6.12.2 Recommendations for log-based recovery

Follow these recommendations for log-based recovery:

- Make archive logs disk-resident.
- Maximize the availability of log data on active logs.
- Consider the use of DB2 Data Compression.
- Use tools, which can be essential.

As indicated earlier, see *Disaster Recovery with DB2 UDB for z/OS*, SG24-6370, for more information about recovery.
Networking and e-business

DB2 9 for z/OS continues to add functions that are related to better alignment with the customers strategic requirements and general connectivity improvements. Customers who are working on the Web and service-oriented architecture (SOA) can benefit from using DB2 9.

In this chapter, we address the following topics:
- Network trusted context
- MQ Messaging Interfaces user-defined function
- SOA
7.1 Network trusted context

The new-function mode of DB2 9 for z/OS introduces a new database entity called a *trusted context*. It provides a technique to work with other environments more easily than before, improving flexibility and security.

A trusted context addresses the problem of establishing a trusted relationship between DB2 and an external entity, such as a middleware server, for example:

- IBM WebSphere® Application Server
- IBM Lotus® Domino®
- SAP NetWeaver
- Oracle PeopleSoft
- Siebel Optimizer

The definition of trusted connection objects allows for more granular flexibility. When established, connections from specific users through defined attachments (distributed data facility (DDF), Resource Recovery Services (RRS) Attach, and default subsystem name (DSN)) and source servers allow trusted connections to DB2. The relationship between a connection and a trusted context is established when a connection to the server is first created and remains for the life of that connection.

For more information about network trusted context, see *Securing DB2 and Implementing MLS on z/OS*, SG24-6480.

7.1.1 Performance

To measure the performance of using a network trusted context, we used the following configuration:

- System z9 Danu hardware (2094-724)
  - Two dedicated CPUs totaling 1162 millions of instructions per second (MIPS)
  - Without System z9 Integrated Information Processors (zIIPs) and System z Application Assist Processors (zAAPs)
  - 32 GB real storage
- z/OS V1.7
- DB2 9 for z/OS
- Gigabit Ethernet network connection within a private VLAN
- IBM WebSphere Application Server for Windows 32-bit v6.1.0.0 (build number b0620.14)
- Java Common Connectivity (JCC) driver version 3.1.55
- Microsoft Windows XP service pack 2 with 3.5 GB memory

We used the following four scenarios to measure the performance impact of using a network trusted context in DB2 9 for z/OS:

- **Scenario 1**: Calling the disconnect Java method at the end of each Relational Warehouse Workload transaction and reconnecting to DB2 using a different client ID to run another transaction
  
  This is the most costly scenario, but it is the only way to ensure DB2 user accountability with DB2 V8 prior to the introduction of the network trusted context feature in DB2 V9.
Scenario 2: Calling disconnect Java method only at the end of the entire workload run

Each client user is authenticated against the external user registry of WebSphere Application Server (Windows operating system registry), but the client user identity is not propagated to DB2 V9 to trigger trusted context processing. (Data source property propagateClientIdentityUsingTrustedContext is set to false.) This is the base scenario to determine the delta of adding trusted context processing by DB2 DDF.

Scenario 3: Same as scenario 2, except the client user identity is propagated to DB2 (setting propagateClientIdentityUsingTrustedContext to true)

Also a trusted context definition for DB2 is created with the WITHOUT AUTHENTICATION option, which means that DB2 will not require a client identity password to be passed from the WebSphere Application Server.

Scenario 4: Same as scenario 3, except that the trusted context definition in DB2 is specified as WITH AUTHENTICATION, which requires a client password to be passed from WebSphere Application Server and verified by RACF® along with its user ID.

Figure 7-1 shows the internal throughput rate (ITR) for the four scenarios.

Comparing scenario 1, which is the most costly scenario, with scenario 3 shows an increase in the ITR by 134% when using the new network trusted context feature in DB2 V9.

The difference in ITR between scenarios 2 and 3 shows the additional overhead introduced by using the trusted context processing. The performance impact is a 2.1% digression of the ITR when using a trusted context.

The difference in ITR between scenarios 2 and 4 shows the additional overhead that is introduced by using the trusted context processing when verification of a user ID and password is needed. The performance impact is a 2.6% digression of the ITR when using trusted context together with verification of a user ID and password.
7.1.2 Conclusion

For customers who have strict security requirements and are currently using scenario 1 to enforce user accountability, they can gain a lot by using the trusted context function in DB2 9 for z/OS. Even with full security validation (user ID and password), the ITR is increased by 134% when using a trusted context.

7.1.3 Recommendation

We recommend that you use a trusted context for WebSphere applications. WebSphere can benefit from a trusted context by reusing the connection and only switching the user ID if the authorization ID has changed.

7.2 MQ Messaging Interfaces user-defined function

In DB2 9 for z/OS, the MQ Application Messaging Interface (AMI) is no longer available and is replaced by the new MQ Message Queue Interface (MQI). In the AMI, you had two sets of UDFs, one for 1-phase commit and another one for 2-phase commit. In the new MQI, only one set of UDFs is needed. Each UDF now supports both 1-phase and 2-phase commit. The behavior can be controlled by specifying the ‘syncpoint’ property in the policy for the UDFs.

AMI supports the sending messages of type 'Request', but you cannot provide an option to specify the 'ReplyToQmgr' name and 'ReplyToQ' name for the application that is sending the message. It defaults to the same queue manager and queue that was used while sending the request message. In MQI, the UDFs have been enhanced, so you now can specify ‘ReplyToQmgr’ and ‘ReplyToQ’ name for the ‘Request’ type of message and support all other message types provided by WebSphere MQ.

This function has been made available for DB2 V8 and V9 via the maintenance stream. For more information about the DB2 MQ user-defined functions and how to use them, see the topic entitled “WebSphere MQ with DB2” in the DB2 Version 9.1 for z/OS Application Programming and SQL Guide, SC18-9841.

Note: The new MQ UDF MQI interface is introduced by APAR PK37290, with PTF UK30228 for DB2 V8 and PTF UK30229 for DB2 V9.

7.2.1 Performance

Figure 7-2 compares the MQ AMI functions with the new MQ UDF functions using MQI directly. The schema name of DB2MQ2N is used to qualify the AMI functions, and the schema name of DB2MQ is used to qualify the MQI functions.
The measurements show an improvement of up to 6 to 10 times in the class 1 elapsed time and class 1 CPU time for the MQSend() and MQReceive() functions. An application normally uses more MQ functions than just MQSend() and MQReceive(), so in general, the application elapsed time can be reduced by up to 2 to 3 times.

The UDFs for the MQI support both 1-phase and 2-phase commit. Figure 7-3 shows a comparison of using 1-phase commit and 2-phase commit.

Sending a message using 2-phase commit increased the number of instructions by 4% compared to sending a message using 1-phase commit. Receiving a message using 2-phase commit increased the number of instructions by 19% compared to receive a message using 1-phase commit.

Sending 10 messages using 2-phase commit increased the number of instructions by 0.2% compared to sending a message using 2-phase commit. Receiving 10 messages using 2-phase commit increased the number of instructions by 2.3 times compared to receiving a message using 2-phase commit.
You can convert the number of instructions to CPU time by dividing by the number of MIPS for your type of CPU.

7.2.2 Conclusion

Using MQI directly instead of using AMI, which was built on top of MQI, has improved performance significantly.

Sending and receiving ten messages using 2-phase commit is similar in performance as sending or receiving only one message using 2-phase commit. Sending or receiving ten single messages is roughly ten times faster than sending or receiving one message ten times.

7.2.3 Recommendation

We recommend that you install APAR PK37290, so you can benefit from this performance enhancement, by using MQI directly instead of AMI.

When using 2-phase commit, you can benefit from sending more messages than one message at the time for nearly no extra cost in the number of instructions that are executed in the MQ UDF.

The increase in the number of instructions for receiving more than one message is much less than the number of instructions executed to receive the same number of messages one at the time.

7.3 SOA

The SOAP UDF extends the flexibility of DB2 to be efficient in performance and in connectivity to partners by using the SOAP interface. The Developer Workbench is a comprehensive development environment that can help you develop and maintain your SOA applications.

For more information about SOA, see Powering SOA with IBM Data Servers, SG24-7259.

7.3.1 SOAP UDFs

DB2 9 for z/OS can act as a client for Web services, which allows users to consume Web services in their DB2 applications. There are four UDFs that you can use to consume Web services through SQL statements.

The UDFs use SOAP, which is an-XML based format for constructing messages in a transport independent way and a standard on how the message should be handled. SOAP messages consist of an envelope that contains a header and a body. It also defines a mechanism for indicating and communicating problems that occurred while processing the message. These are known as SOAP faults.

The headers section of a SOAP message is extensible and can contain many different headers that are defined by different schemas. The extra headers can be used to modify the behavior of the middleware infrastructure. For example, the headers can include information about transactions that can be used to ensure that actions performed by the service consumer and service provider are coordinated.
The body section contains the contents of the SOAP message. When used by Web services, the SOAP body contains XML-formatted data. This data is specified in the Web Services Description Language (WSDL) that describes the Web service.

When talking about SOAP, it is common to talk about SOAP in combination with the transport protocol used to communicate the SOAP message. For example, SOAP transported using Hypertext Transfer Protocol (HTTP) is referred to as SOAP over HTTP or SOAP/HTTP.

The most common transport used to communicate SOAP messages is HTTP. This is to be expected because Web services are designed to use Web technologies.

The UDFs send a SOAP request over HTTP to access Web services. When the consumer receives the result of the Web service request, the SOAP envelope is stripped, and the actual XML document result is returned. The result data returned from this SQL call can be processed by the application itself or used in variety of other ways including inserting or updating a table, which is sent to XML Extender for shredding or sent to a WebSphere MQ queue.

Example 7-1 shows how to invoke a UDF that sends a SOAP request over HTTP. In this example, we assume that getRate is the operation name that was offered as Web service by the Web service provider that takes two countries as input and returns the current exchange rate as the output.

**Example 7-1   Creating and invoking a SOAP UDF**

```
CREATE FUNCTION getRate (  
country1 VARCHAR(100),  
country2 VARCHAR(100) )  
RETURNS DOUBLE  
LANGUAGE SQL  
EXTERNAL ACTION  
NOT DETERMINISTIC  

-- SQL Scalar UDF that wraps the web service, "getRate", described by  
-- http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl  
--  
RETURN db2xml.extractDouble(  
db2xml.xmlclob(  
   db2xml.soaphttpv('http://services.xmethods.net:80/soap',  
      ' ',  
      varchar('<m:getRate xmlns:m="urn:xmethods-CurrencyExchange" ' ||  
      'SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"> ' ||  
      '<country1 xsi:type="xsd:string">' || country1 || '</country1>' ||  
      '<country2 xsi:type="xsd:string">' || country2 || '</country2>' ||  
      '</m:getRate>'))), '/***');
```

The SOAPHTTPV function returns a VARCHAR representation of XML data that results from a SOAP request to the Web service specified by the first argument. For more information about SOAPHTTPV, see *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854.
Local SQL statements can be replaced by an SQL call that resolves into a Web services call. See Figure 7-4.

```sql
SELECT rate FROM Ratetable
WHERE country1=c1 AND country2= c2;
```

```sql
SELECT rate FROM TABLE (getRate(?));
```

**Figure 7-4  SQL call resolves into a Web Services call**

The UDF getRate() is a wrapper UDF that calls a SOAP UDF. The SOAP UDF reads the XML reply and returns the results as an SQL result.

For more information about XML in DB2 9 for z/OS, see Chapter 3, “XML” on page 61.

### 7.3.2 IBM Data Studio

In DB2 V7.2 for Linux, UNIX, and Windows, IBM introduced tooling support for stored procedures via the Stored Procedure Builder. IBM enhanced this tooling with the follow-on tool, Development Center, in DB2 V8.1 for Linux, UNIX, and Windows. With DB2 9 for Linux, UNIX, and Windows, the Developer Workbench was introduced. Developer Workbench is based on Eclipse technology. The stored procedure tooling for DB2 databases is also consistent with the tooling delivered in WebSphere Application Developer and Rational® Application Developer. On October 30, 2007, IBM announced the IBM Data Studio, which builds upon the tooling support of Developer Workbench and other IBM tooling products as well.

Currently there are three “versions” of IBM Data Studio:

- Data Studio V1.1.0 (free version), which uses the DB2 on Linux, UNIX, and Windows PID
- Data Studio Developer V1.1.1(charged version), program number 5724-U15
- Data Studio pureQuery runtime V1.1.1 (charged version), program number 5724-U16

The IBM Data Studio is a comprehensive data management solution that empowers you to effectively design, develop, deploy and manage your data, databases and database applications throughout the entire application development life cycle utilizing a consistent and integrated user interface. Included in this tooling suite are the tools for developing and deploying DB2 for z/OS stored procedures. Unlike Development Center (DC), which was included in the DB2 V8.1 UDB Application Development Client (ADC) component, the IBM Data Studio is independent of any other product offering and does not require a DB2 client installed.

IBM Data Studio supports the entire family of DB2 servers, as well as Informix®, using the DRDA architecture. It supports Versions 8 and 9 of DB2 for Linux, UNIX and Windows; Versions 7, 8 and 9 of DB2 for z/OS; Versions 5.3 and up of DB2 for iSeries®; and Informix Data Servers. The suite of servers and functions that IBM Data Studio supports are summarized in Figure 27-1.
IBM Data Studio and Rational Application Developer both sit on top of the Eclipse Framework. Thus, both products have a similar “look and feel”. For information on using the IBM Data Studio with stored procedures see DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond, SG24-7604-00.

For more information about IBM Data Studio and its additional feature, visit:

http://www.ibm.com/software/data/studio
DB2 V9 provides a number of significant enhancements to improve availability, performance, and usability of DB2 data sharing.

In this chapter, we explain the following topics:

- Initiating automatic GRECP recovery at the end of restart
- Deferring the updates of SYSLGRNX
- Opening data sets earlier in restart processing
- Allowing table-level retained locks to support postponed abort unit of recovery
- Simplification of special open processing
- Data sharing logging improvement
- Reduction in LOB locks
- Index improvements:
  - Index compression and greater than 4 KB pages for indexes
  - Sequential key insert performance improvement
  - Ability to randomize index keys giving less contention
- Improved group buffer pool write performance
- Improved Workload Manager routing based on DB2 health
- Improved workload balancing within the same logical partition
- Group buffer pool dependency removal by command
- Open data set ahead of use via command
- Enhanced messages when unable to get P-locks
8.1 Initiating automatic GRECP recovery at the end of restart

In DB2 V8, if DB2 total loss of connectivity or group buffer pool (GBP) structure failures occur, DB2 performs automatic recovery when AUTOREC YES is set.

In DB2 V9, DB2 also initiates automatic recovery when a GBP structure has been lost. DB2 initiates automatic group buffer pool RECOVER-pending (GRECP) recovery, after it has restarted. Each DB2 member only initiates recovery for GRECP for group buffer pool-dependent objects that had an update interest at the time it came down (read/write state). AUTOREC is used to control this functionality, so it must be set to YES to have automatic GRECP recovery on restart.

When DB2 initiates automatic GRECP recovery after restart, it tries and acquires a conditional drain for each GRECP object. If DB2 is successful in acquiring the drain, it performs the recovery from the log. Note that all the drained objects are recovered by a single process. The reason for this is two-fold. Firstly, this results in just one scanning of the log. Secondly and more importantly, the objects do not need to be sequenced. That is, the catalog and directory objects do not need to be done first. Fast log apply is always used as this is part of restart.

If the drain of an object is not successful, a DSNI005I message is issued, and the object remains in GRECP. A possible reason for not successfully draining is that there are outstanding retained locks, either because another DB2 is stopped or is in the early stages of restarting. If this is the case, after that DB2 completes its own restart, it too initiates automatic GRECP recovery.

From an operational perspective, issue -START DB2 commands for all failed members as soon as possible. The amount of recovery work that is performed by an individual DB2 member depends on GBP dependencies across the group. The last member could do a significant amount if the workload is not fully balanced across the group.

The following exceptions apply when automatic GRECP recovery is not used:

- DB2 is restarted with the DEFER ALL option.
- DB2 is restarted in system-level PITR mode.
- DB2 is started in tracker-site mode.
- DB2 is restarted in Restart-Light mode.

8.2 Deferring the updates of SYSLGRNX

The closing off of SYSLGRNX records is important for the performance of recoveries. In DB2 V8, non-GBP dependent objects that were opened, but not involved in the restart log apply phases (forward log recovery and backward log recovery), have SYSLNGRNX entries closed off at the end of restart. These updates at the end of restart could contend with other members updates, resulting in extended restart times.

In DB2 V9, the updating of non-GBP dependent object SYSLGRNX entries are deferred beyond restart, therefore allowing restart to complete quicker. The updating of the SYSLGRNX entries is now triggered by the first system checkpoint following restart. Note that this benefits DB2 restart in a non-data sharing environment as well.
8.3 Opening data sets earlier in restart processing

During the forward log recovery phase of restart, any page sets for which log records are found need to be opened. The number of page sets that need to be opened could number in the thousands. DB2 sets a limit of 40 tasks for the parallel open or close of Virtual Storage Access Method (VSAM) data sets, so data set opens can still amount to a significant part of the elapsed time of restart.

During restart, fast log apply is always enabled. Fast log apply involves reading the log and sorting records. Finally the updates are done by one or more apply tasks, with list prefetches being done to read the appropriate pages. Synchronous opens are done as required.

This enhancement identifies page sets that are not yet opened during the reading of the logs and schedules an asynchronous open. As a result of scheduling these asynchronous opens, which are conditional, less time is spent waiting on data set opens. In the case where the open has not occurred by the time the apply task needs access to the data set, a normal open is performed, and the asynchronous open does not happen because it is conditional.

This function also benefits non-data sharing environments.

This feature is always active unless:

- DB2 is restarted with the DEFER ALL option.
- DB2 is restarted in system-level PITR mode.
- DB2 is started in tracker-site mode.
- DB2 is restarted in Restart-Light mode

8.4 Allowing table-level retained locks to support postponed abort unit of recovery

DB2 during system checkpoints store information about each modified object by an uncommitted unit of recovery (UR). This information, which is stored in the UR summary checkpoint record, is kept at the table space and partition level, but not at the table level. The information in the checkpoint is used for two purposes:

- To determine how far back in the log it needs to go in order to completely back out the UR
- To assist in building the retain lock for postponed abort objects

The enhancement in V9 is to now maintain, in the checkpoint, the information at table level so that each table in a segmented table space can be tracked independently. With each table now having its own lock, an application is not blocked from using other tables within a multi-table table space, due to one table being subject to a postponed abort.

Note that, although each retained table lock will be purged as the back out of the last postponed abort is completely done for that table, the AREST state applies to the table space. The AREST state is not changed until all tables in the table space have resolved any postponed aborts.
8.5 Simplification of special open processing

During the forward and backward log phases of restart, data sets are opened if there are log records to apply. During the opening of these data sets, the retained page set physical locks (P-locks) are reacquired and converted to active page set P-locks. These are held until data set closure.

Not every GBP-dependent object at the time of the DB2 failure has log records to apply. Without the presence of log records to apply, these objects are not opened and consequently the retained P-locks remain. To avoid this situation, DB2 processes these objects at the end of restart, in what is referred to as special open processing.

During special open processing, the retained P-locks are reacquired and purged, although the data sets are not actually opened. When DB2 reacquires the page set P-locks, it first takes what is known as a conversion lock, which prevents a P-lock state from being immediately changed by another DB2. These P-lock states are stored by DB2. It is important that states are accurately recorded and the conversion locks perform the serialization to ensure this. In effect, this is a lock around the taking of a lock. The reason this serialization is needed is that the P-lock exit can be triggered the moment the P-lock is reacquired. For example, if the reacquired state is IX, but it is immediately changed to SIX, before the lock state is stored inside DB2, DB2 could end up with the wrong state being stored.

There is lock contention around taking these conversion locks, which can prolong restart times. The enhancement in V9 is to remove the need for conversion locks during the special open. To remove the need for conversion locks, DB2 uses an alternative serialization technique.

When DB2 manipulates the P-locks, the internal resource lock manager (IRLM) provides DB2 with a sequence number. DB2 now stores this sequence number along with the lock state. When DB2 updates the P-lock state, it can avoid doing the update if the sequence number already stored is greater than the one associated with the state that it is trying to update. This way it does not regress the state.

Conversion locks are still used outside of special open processing.

8.6 Data sharing logging improvement

Prior to DB2 9 for z/OS, successive log records produced on the same DB2 member always had different log record sequence numbers (LRSNs). In order to achieve this, DB2 would re-drive the pulling of the store clock (STCK) to produce the LRSN, if it was the same as the previous one. With processor speeds continually increasing, especially with System z9 technology, this became increasingly likely. Each time the STCK was re-driven in this way, CPU cycles were effectively lost. Note that while DB2 is re-driving the pulling of the STCK, the log latch is maintained. Holding the log latch in this way aggravates log latch contention.

With this enhancement, it is now only necessary for a given DB2 member to produce unique LRSNs when the log records are for the same data page. This saves both CPU cycles and reduces log latch contention.
8.7 Reduction in LOB locks

Changes to LOB locking, detailed in 5.10, “LOB performance enhancements” on page 182, mean that LOB data must be externalized for GBP-dependent LOB table spaces, before locks are released. Also the index on the LOB table needs to be externalized if it is GBP-dependent. LOBs do not have to be externalized to disk for non-GBP dependent LOBs. For GBP-dependent LOBs, the externalizations can be to either DASD or the GBP. The latter is usually the faster. Therefore, users should consider using the GBPCACHE CHANGED option for LOB table spaces.

This function is brought in with locking protocol 3. To be enabled, the data sharing group needed to be quiesced and restarted, when it is in new-function mode. However APAR PK62027 (PTF UK38906) removed this requirement.

8.8 Index improvements

The following improvements to indexes are particularly beneficial in a data sharing environment. They are covered else where in this book.

- Index compression and greater than 4 KB pages for indexes; see 5.5, “Index compression” on page 172
- Sequential key insert performance improvement; see 5.4, “Relief for sequential key insert” on page 154
- Ability to randomize index keys giving less contention; see “Index key randomization” on page 156
- A data-partitioned secondary index (DPSI) to support affinity routing and eliminate data sharing overhead on that partition
  
  The potential to do this is now increased because a key can now be unique within the partition.
- Optimizing INSERT/DELETE/UPDATE performance by removing user-defined unused indexes

Some applications are installed with indexes that are not beneficial or are no longer necessary for queries, static or dynamic, that access the tables. To save space and optimize the performance of INSERT, DELETE, and UPDATE statements, these indexes need to be identified and deleted. DB2 9 for z/OS introduces the LASTUSED column as an addition to the RTS table SYSIBM.SYSINDEXSPACESTATS that indicates the day the index was last used to process an SQL statement. This capability allows you to revisit indexes. With RTS detecting index usage at run time, you can clearly determine when an index can be dropped, and, based on time thresholds and application considerations, you may consider discarding it.

8.9 Improved group buffer pool write performance

Prior to DB2 9 for z/OS, batched GBP writes involved copying the pages to be written to the GBP to contiguous storage, the address of which was passed to XES. XES is now passed a list of addresses that correspond to the buffer pool pages. The copying of pages is now avoided, which improves performance.
For group buffer pools that are duplexed, DB2 V9 eliminates cross invalidations as a result of the secondary that is being updated.

8.10 Improved Workload Manager routing based on DB2 health

Each DB2 has a new health monitor to detect if DB2 is becoming storage constrained. Each DB2 periodically informs Workload Manager (WLM) as to the health of DB2. When DB2 is storage constrained, the health is less than 100%. WLM attempts to route work away from subsystems that are less than 100%, hopefully giving them a chance to recover.

8.11 Improved workload balancing within the same logical partition

Due to virtual storage constraints, some installations have grown their data sharing groups. Instead of adding new logical partitions (LPARs) for each additional DB2, they have decided to run a second DB2 on existing LPARs used by the group. When the group attachment is used from the foreground Time Sharing Option (TSO) or batch, the connection is directed to the first available subsystem that is defined in IEFSSNxx member in SYS1.PARMLIB. This means that workload balancing is particularly difficult for batch.

In V9, the behavior is changed so connections are made on a round-robin basis. This applies to DSN, call attachment facility (CAF), RRS and utilities. This functionality can optionally be achieved in both V7 and V8 via USERMOD.

8.12 Group buffer pool dependency removal by command

There may be periods when work against a particular table space is performed only from one member in the group, for example, during batch processing. Under these circumstances, it is desirable not to pay the data sharing overheads. DB2 V9 introduces the following command to allow the removal of group buffer pool dependency for specified objects:

\[-ACCESS DB(dbname) SPACE(spacename) PART(n) MODE(NGBPDEP)\]

You should run this command on the member on which work is to continue or be scheduled.

DB2 performs the following actions:

1. Drains all readers and writers on all members other than that on which the command is entered.
2. If step 1 is successful, then the writers on the system on which the command was entered are drained, assuming that the object is not already pseudo closed.

If the drains fail, the following message is issued:

\[DSN048I mod-name CLAIMERS EXIST FOR DATA BASE dbname, SPACE NAME tsname, PART partno. GROUP BUFFERPOOL DEPENDENCY CANNOT BE REMOVED.\]

Note: STARTDB authority, either explicit or implicit, is required for the -ACCESS command.
The -DISPLAY DATABASE CLAIMERS command can be used to identify who is blocking the drains. If the drains are successful, then the member on which the command was issued converts the object to non-group buffer pool dependent, including the castout of any changed pages. The object is eligible immediately to be group buffer pool dependent, should there be subsequent accesses from the other members.

In addition to table spaces, the command is valid for index spaces.

### 8.13 Open data set ahead of use via command

The first thread to access a given object drives the physical open of the data set. This gives an elongated response time for that thread. For response-sensitive transactions, the following new command is available to pre-open or prime the data set:

```plaintext
-ACCESS DB(dbname) SPACE(spacename) PART(n) MODE(OPEN)
```

The command, which requires STARTDB authority, can be used for both table spaces and index spaces. The command works in both data sharing and non-data sharing. The command is local in scope within data sharing.

**Note:** APAR PK80925 introduces support for subset, range, and wildcarding in the ACCESS DB command.

### 8.14 Enhanced messages when unable to get P-locks

ABEND04E with reason code 00C20255 was a generic “Unable to obtain a P-lock” condition. This is enhanced once in new-function mode to be more specifically “P-lock failures involving another DB2 member holding the lock which has not responded to the P-lock request”.

These additional reason codes have been added to indicate other P-lock issues:

- **00C2026A**: Unable to obtain a P-lock because another DB2 member is in the process of shutting down
- **00C2026B**: Unable to obtain a P-lock because another DB2 member has had logging suspended due to the active logs being full or an outstanding SET LOG SUSPEND command
- **00C2026C**: Unable to obtain a P-lock because another DB2 member holding the lock encountered an abend in its P-lock exit.
Installation and migration

In this chapter, we review factors that can impact the performance of the installation of DB2 9 and any migration issues that you need to be aware of that can affect DB2 9 performance. A more detailed description of the installation and migration process is in the DB2 9 manuals. For a list of the DB2 9 documentation, refer to the documentation listed in "Other publications" on page 374. Specifically refer to DB2 for z/OS Version 9.1 Installation Guide, GC18-9846, and DB2 for z/OS Version 9.1 Data Sharing: Planning and Administration Guide, SC18-9845.

In this chapter, we describe the process for the installation and migration of DB2 9. We include process timings for the steps that are necessary to move from conversion mode to new-function mode.

In this chapter, we discuss the following topics:

- Installation verification procedure sample program changes
- Installation
- Migration
- Catalog consistency and integrity checking
- DSNZPARM changes
9.1 Installation verification procedure sample program changes

DSNTIAUL is the DB2-supplied sample assembler language unload program. DSNTIAD is the DB2-supplied sample assembler dynamic SQL program. DSNTEP2 is the DB2-supplied sample PL/I dynamic SQL program. And DSNTEP4 is a modification of the DB2-supplied DSNTEP2 sample PL/I dynamic SQL program that supports multi-row fetch.

The sample programs DSNTEP2, DSNTEP4, and DSNTIAUL have all been modified to support the following new data types introduced with DB2 9:

- BIGINT
- BINARY
- VARBINARY
- DECFLOAT
- XML

The sample program DSNTIAD has been modified to now support 2 MB SQL statements. The sample programs DSNTEP2 and DSNTEP4 now support a new parameter called PREPWARN. This tells DSNTEP2 or DSNTEP4 to display the PREPARE SQLWARNING message and set the return code to 4 when an SQLWARNING is encountered at PREPARE.

DSNTIAUL now permits you to use LOB file reference variables to extract a full LOB value to an external data set. The output data sets for unloaded LOB data are allocated dynamically. This is done using the LOB file reference SQL_FILE_CREATE option and are associated with a DD statement.

The data set that was created by DSNTIAUL is named with the convention <prefix>.Q<i>.C<j>.R<k>, where:

- <prefix> is the data set name prefix. <prefix> cannot exceed 26 characters.
- Q<i> is the (<i>-1)th query processed by the current DSNTIAUL session, where <i> is in the range from 00 to 99.
- C<j> is the (<j>-1)th column in the current SELECT statement, where <j> is in the range from 000 to 999.
- R<k> is the (<k>-1)th row fetched for the current SELECT statement, where <k> is in the range from 0000000 to 9999999.

The new DSNTIAUL option LOBFILE is used for this; see Example 9-1. If the DSNTIAUL option LOBFILE is not specified, then LOB data is handled as in previous releases. You can unload up to 32 KB of data from an LOB column.

Example 9-1  DSNTIAUL - LOBFILE option

```plaintext
RUN PROGRAM(DSNTIAUL) PLAN(DSNB91)
PARMS('SQL,2,LOBFILE(DSN910.DSNTIAUL)') -
LIB('DSN910.RUNLIB.LOAD')
```

The parameters in Example 9-1 create LOB unload data sets with names that starting with:

DSN910.DSNTIAUL.Q0000000.C0000000.R0000000

The names end with:

DSN910.DSNTIAUL.Q0000099.C0000999.R9999999

The generated LOAD statement from the execution of DSNTIAUL contains the LOB file reference variables that can be used to load data from these dynamically allocated data sets.
For further information about running DSNTIAUL, DSNTIAD, DSNTEP2, and DSNTEP4, see Appendix D in *DB2 Version 9.1 for z/OS Utility Guide and Reference*, SC18-9855.

9.2 Installation

The installation process has been detailed in two publications. See *DB2 Version 9.1 for z/OS Installation Guide*, GC18-9846, and Chapter 12 in *DB2 9 for z/OS Technical Overview*, SG24-7330.

9.2.1 DB2 9 for z/OS function modification identifiers

DB2 9 for z/OS consists of the following function modification identifiers (FMIDs):

- **Required FMIDs:**
  - HDB9910 (contains DB2 Base, msys plug-in, REXX, MQSeries®, MQListener)
  - HIY9910 (IMS Attach - must be installed even if you do not have IMS)
  - HIZ9910 (Subsystem Initialization)
  - HIR2220 (IRLM V2R2)
  - HDRE910 (DB2 RACF Authorization Exit)
  - JDB9914 (DB2 English Panels)

- **Optional FMIDs:**
  - JDB9912 (DB2 JDBC/SQLJ)
  - JDB9917 (DB2 ODBC)
  - JDB991X (DB2 XML Extender)
  - JDB9911 (DB2 Kanji Panels)

This information was taken from the Program Directory for IBM DB2 9 for z/OS V09.01.00 Program Number 5635-DB2. Refer to this publication for any changes to the DB2 9 prerequisites that could affect your installation and migration process.

The DB2 Utilities Suite, program number 5655-N97, is comprised of FMID JDB991K and contains DB2 utilities that are not provided in the DB2 base.

There is also a no charge feature called the *DB2 Accessories Suite*, program number 5655-R14. It is a separate product that is available for DB2 9, but it is not included in the base. It is a collection of tools that extend the current DB2 functionality:

- DB2 Optimization Service Center, FMID H2AG110
- DB2 Spatial Support, FMID J2AG110
- International Components for Unicode for DB2 for z/OS, FMID H2AF110

9.3 Migration

Migration is the process where you move from a DB2 V8 to a DB2 9 environment. In this section, we assume the DB2 V8 product is installed in new-function mode. We provide details about the Migration process. For information about the installation process, see *DB2 Version 9.1 for z/OS Installation Guide*, GC18-9846, and Chapter 12 of *DB2 9 for z/OS Technical Overview*, SG24-7330.
9.3.1 Introduction to migration to DB2 9

You can only migrate to DB2 9 from a DB2 V8 system in new-function mode. DB2 9 has a minimum requirement that z/OS is to be at least at V1.7 to run and at V1.8 if you want to take advantage of volume-level copy, object-level recovery, and tape support. We recommend that you have a stable z/OS V1.7 or later environment before you migrate to DB2 9. If you have to fall back to a z/OS level that is earlier than V1.7, then DB2 9 will not start.

In September 2007, z/OS V1.9 introduced extra functions, such as XML System Services System z Application Assist Processor (zAAP) support and planned System z9 Integrated Information Processor (zIIP) support, Unicode performance enhancements, and DECFOAT hardware support. For details, see the announcement on the Web at the following address:

DB2 9 operates on any processor that supports 64-bit z/Architecture, including zSeries, System z 990 (z990), System z 890 (z890), or a comparable processor. See the Program Directory for IBM DB2 9 for z/OS, GI10-8737, for more information about system requirements.

To prepare for migration to DB2 V9.1 for z/OS, you must apply APAR PK11129. Read the HOLD data for PK11129, and note that, due to the possibility of prerequisite APARs, it may be necessary to acquire additional APARs that are not related to fallback. See information APARs II12423, II4401, II14464 for additional APAR information, and see DB2 for z/OS Version 9.1 Installation Guide, GC18-9846, for details about migration, fallback, and remigration steps.

During the installation or migration to DB2 V9, the system programmer should be aware of how certain system parameters impact performance. Often the system parameters are chosen based on defaults and are carried along, but sometimes they change across versions and might need adjusting.

The migration path from DB2 V8 new-function mode to DB2 V9 new-function mode goes through three modes. These are the three modes that DB2 V8 used to achieve new-function mode. Two extra modes have been added to this path that are informative about the state of a system after a fallback to a previous mode. These new modes are conversion mode (CM*) and enabling-new-function mode (ENFM*), and are described in the following section.

In conversion mode, no new DB2 9 function is available for use which could preclude the fall back to Version 8. To use some functions, the DB2 subsystem or data sharing group must first convert the DB2 catalog to a new-function mode catalog via the ENFM process.

This information was taken from Program Directory for IBM DB2 9 for z/OS, GI10-8737, for IBM DB2 9 for z/OS V09.01.00 Program Number 5635-DB2. Refer to this publication for any changes to the DB2 9 prerequisites that could affect your installation and migration process.

DB2 9 modes: An overview

DB2 supports the following modes:

- Conversion mode: This is the DB2 mode where DB2 V9 has migrated from V8 and is started for the first time. It is still in conversion mode when the migration job DSNTIJTC has been run to perform the CATMAINT function has completed. Some new user function can be executed in conversion mode. Data sharing systems can have V8 new-function mode and V9 conversion mode members in this coexistence mode. Coexistence is recommended to be short, such as a weekend or a week. DB2 can only migrate to conversion mode from V8 new-function mode.
This mode is not so much compatibility, as conversion, the ability to fall back to DB2 V8. We try to move most (but not all) problems for the migration from new-function mode and ENFM to conversion mode, so that fallback to DB2 V8 can be used, if necessary.

**Note:** IBM changed the DB2 term compatibility mode to *conversion mode* to better reflect the characteristics of this transitory product situation.

- **ENFM**: This mode is entered when CATENFM START is executed, which is the first step of migration job DSNTIJEN. DB2 remains in this mode until all the enabling functions are completed. Note that data sharing systems can only have DB2 V9 members in this mode because you cannot mix V8 new-function mode and DB2 V9 ENFM in a data sharing group.

- **New-function mode**: This mode is entered when CATENFM COMPLETE is executed (the only step of job DSNTIJNF). This mode indicates that all catalog changes are complete and all DB2 V9 new functions can be used.

- **ENFM***: This is the same as ENFM but the asterisk (*) indicates that, at one time, DB2 was at new-function mode and has fallen back to ENFM. Objects that were created when the system was at new-function mode can still be accessed, but no new objects can be created. When the system is in ENFM* it cannot fall back to V8 or coexist with the V8 system.

- **CM***: This mode is the same as conversion mode, but the asterisk (*) indicates that, at one time, DB2 was at a higher level. Objects that were created at the higher level can still be accessed. When DB2 is in CM* mode, it cannot fall back to DB2 V8 or coexist with a DB2 V8 system.

The following situations are possible for DB2 V9 mode fallback:

- It is possible to fallback from ENFM to CM*.
- It is possible to fallback from new-function mode to ENFM* or CM*.
- It is possible to fallback from ENFM* to CM*.

**Note:** If you fall back from new-function mode, or ENFM*, objects that were created or new SQL features you have exploited will still function in ENFM* and CM*, but no new objects can be created and no new functions can be exploited until you return to new-function mode.

Migration to conversion mode is achieved by running the CATMAINT UPDATE job DSNTIJTC, which:
DB2 9 for z/OS Performance Topics

9.3.2 Catalog changes

As DB2 has grown over the years, the amount of data stored in the DB2 catalog has increased with each version. Table 9-1 shows the number of DB2 objects in the catalog from the initial Version 1 to the current Version 9.

<table>
<thead>
<tr>
<th>DB2 Version</th>
<th>Table spaces</th>
<th>Tables</th>
<th>Indexes</th>
<th>Columns</th>
<th>Table check constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>11</td>
<td>25</td>
<td>27</td>
<td>269</td>
<td>N/A</td>
</tr>
<tr>
<td>V3</td>
<td>11</td>
<td>43</td>
<td>44</td>
<td>584</td>
<td>N/A</td>
</tr>
<tr>
<td>V4</td>
<td>11</td>
<td>46</td>
<td>54</td>
<td>628</td>
<td>0</td>
</tr>
<tr>
<td>V5</td>
<td>12</td>
<td>54</td>
<td>62</td>
<td>731</td>
<td>46</td>
</tr>
<tr>
<td>V6</td>
<td>15</td>
<td>65</td>
<td>93</td>
<td>987</td>
<td>59</td>
</tr>
<tr>
<td>V7</td>
<td>20</td>
<td>82</td>
<td>119</td>
<td>1206</td>
<td>105</td>
</tr>
<tr>
<td>V8</td>
<td>21</td>
<td>87</td>
<td>128</td>
<td>1286</td>
<td>105</td>
</tr>
<tr>
<td>V9</td>
<td>28</td>
<td>106</td>
<td>151</td>
<td>1668</td>
<td>119</td>
</tr>
</tbody>
</table>
The changes to the DB2 V9 catalog from DB2 version 8 are detailed in Appendix G of *DB2 Version 9.1 for z/OS SQL Reference*, SC18-9854. Refer to this manual for a detailed description of the changes.

### 9.3.3 Summary of catalog changes

The following list is a summary of the DB2 catalog changes. Update your existing image copy jobs to specify the new objects or ensure that your utility templates include these objects. The addition of extra objects extends the time that your utilities process against the catalog. The extra time that is used to process these DB2 catalog entries is site dependent based on the amount of data and techniques that are used.

**For trusted contexts and roles**
- New table space DSNDDB06.SYSCONTX
  - New table SYSIBM.SYSCONTEXT
  - New table SYSIBM.SYSCONTEXTAUTHIDS
- New table DSNDDB06.SYSROLES
  - New table SYSIBM.SYSROLES
  - New table SYSIBM.SYSOBJROLEDEP

**For object dependencies**
- In existing table space DSNDDB06.SYSOBJ
  - New table SYSIBM.SYSDEPENDENCIES
  - New table SYSIBM.SYSENVIRONMENT

**For extended indexes**
- New table space DSNDDB06.SYSTARG
  - New table SYSIBM.SYSKEYTARGETS
- In existing table space DSNDDB06.SYSSSTATS
  - New table SYSIBM.SYSKEYTARGETSTATS
  - New table SYSIBM.SYSKEYTGTDIST
- In existing table space DSNDDB06.SYSHIST
  - New table SYSIBM.SYSKEYTARGETS_HIST
  - New table SYSIBM.SYSKEYTGTDIST_HIST

**For Java**
- In existing table space DSNDDB06.SYSJAVA
  - New table SYSIBM.SYSJAVAPATHS

**For native SQL procedures**
- New table space DSNDDB06.SYSPLUXA
  - New table SYSIBM.SYSROUTINESTEXT

**For real-time statistics**
- New table space DSNDDB06.SYSRSTSTS
– New table SYSIBM.SYSTABLESPACETATS
– New table SYSIBM.SYSINDEXSPACESTATS

For XML
► New table space DSNDDB06.SYSXML
– New table SYSIBM.SYSXMLSTRINGS
– New table SYSIBM.SYSXMLRELS

For XML schema repository
► New table space DSNXR.SYSXSR
– New table SYSIBM.XSRCOMPONENT
– New table SYSIBM.XSROBJECTS
– New table SYSIBM.XSROBJECTCOMPONENTS
– New table SYSIBM.XSROBJECTGRAMMAR
– New table SYSIBM.XSROBJECTHIERARCHIES
– New table SYSIBM.XSROBJECTPROPERTY
– New table SYSIBM.XSRPROPERTY

9.3.4 DB2 9 migration considerations

Migrating to DB2 9 and fallback to DB2 V8 are only supported from DB2 V8 new-function mode with toleration and coexistence maintenance applied. Before attempting to migrate to DB2 9, DB2 V8 fallback SPE must be applied to the subsystem and the subsystem must be started. Without the DB2 V8 fallback SPE and subsystem startup, the attempt to migrate to DB2 9 fails and the migration attempt is terminated. Information APAR II1440 details the DB2 V8 migration/fallback required APARs to/from DB2 9.

At DB2 startup time, the code level of the DB2 subsystem is compared to the code level that is required by the current DB2 catalog. If there is a code-level mismatch between the two, then the DSNX208E message is issued and the DB2 startup is terminated.

The same process is done for DB2 subsystems in a data sharing group with the addition that the code-level check is also done to the participating DB2 subsystems that are started. If the starting system has a code-level mismatch with the catalog or any of the DB2s that are running, then a message is issued and the subsystem startup is terminated. The startup termination message is DSNX208E or DSNX209E.

We recommend that you start only one DB2 subsystem at DB2 9 in a data sharing group for the migration processing.

Premigration work
Ensure that your BSDSs are converted to the new expanded format that became available with DB2 V8 new-function mode. This expanded format allows for support of up to 10000 archive log volumes and up to 93 data sets for each copy of the active log. If you are running DB2 V8 in new-function mode and have not yet converted to the new BSDS format, you can do so by running the supplied utility (DSNJCNVB). Ensure you have enough disk space, 10 MB per boot strap, available for the new format BSDS before converting.

After conversion of the BSDS, the maximum size that you can allocate for the DB2 active log data set is 4 GB. You can have 93 active log data sets and up to 10 000 archive log data volumes.
We recommend that you do a REORG of your catalog prior to migrating to DB2 V9, because this helps to speed up the migration process. It is good business practice to REORG your catalog at least once per year and to check the integrity of your catalog and directory by checking for broken links using DSN1CHKR. You should also check catalog and directory indexes for consistency (see sample job DSNTIJCX).

Make sure that you run premigration job DSNTIJP9 on your DB2 V8 system to do an overall health check prior to migrating to DB2 V9. This job can be run at any time during the premigration planning stages or any time you are interested in learning about the cleanup work that needs to be done to DB2 V8 in order to prepare for migrating to DB2 V9. It is possible that DSNTIJP9 was not delivered with your DB2 V8 system software and the initial installation or migration as DSNTIJP9 was delivered via the service stream in APAR PK31841. We highly recommend that you apply this APAR and run this job during the early stages of DB2 V9 migration planning. The job is called DSNTIJPM when delivered with the DB2 V9 installation materials.

We recommend that you investigate whether you should increase the size of the underlying VSAM clusters for your catalog and directory before you migrate to DB2 V9. You should resize the catalog objects in order to consolidate space, eliminate extents, and provide for additional space for new columns and objects and for general growth. 

Run the tailored DSNTIJIN migration job, which uses IDCAMS, to define data sets for new the catalog table spaces and indexes.

9.3.5 Catalog migration

There are two steps involved in the catalog migration to V9, as there were to V8:

1. Migrate to conversion mode.
   
   As part of the initial stage of migration to V9 conversion mode, you run supplied and tailored job DSNTIJTC. This job runs the DSNUTILB utility program with the parameters CATMAINT UPDATE. CATMAINT has been part of the standard DB2 migration process for many versions.

2. Enable new-function mode.
   
   The job you run is DSNTIJEN. It consists of an online REORG of the catalog table spaces SYSIBM.SYSOBJ and SYSIBM.SYSPKAGE.
   
   You then run DSNTIJNF to switch to new-function mode, and verify the status by entering:

   ```
   DISPLAY GROUP DETAIL MODE(N)
   ```

9.3.6 To rebind or not to rebind

Remember Shakespeare’s Hamlet, act 3, scene 1 as revised by Roger Miller?

The question is not whether to rebind, but rather when to rebind. Some problems can only be resolved by rebinding, such as rebuilding a package after a dropped and recreated table or index. Some improvements like new optimization techniques, improved statistics to provide better information for optimization choices, or improved virtual storage as used by plans and packages moved above the 2 GB bar, are all available after rebind. Also notice that converting from plans with DBRMs (deprecated in DB2 9 for z/OS) to packages requires binding. See the recent DB2 9 for z/OS Packages Revisited, SG24-7688 for the latest information on packages.
With DB2 9 for z/OS, we recommend to rebind at conversion mode (CM) time. The purpose of CM is to allow a phase for migration testing during which fallback to the previous release is available. Once you are in CM, there are several changes to the access path selection, there is a new CLUSTERRATIO algorithm and a new statistic field called DATAREPEATFACTORF which measures density of underlying data for each index (see 6.3, “RUNSTATS enhancements” on page 214), These improvements are picked up by rebind.

**Tip:** Perform RUNSTATS + REBIND on your packages before converting to DB2 9 NFM

DB2 9 for z/OS has introduced new options for REBIND PACKAGE which can help in case of regression, see 5.14, “Package stability” on page 192. This function is available via maintenance on DB2 9, not on earlier versions, so you can start using it with DB2 9 in CM. Use package stability because it provides, in case of regression, an easy way to restore a previous access path.

If you are still concerned about the extra catalog space required to store on disk multiple versions of the runtime execution structures (which are the execution instructions implementing the chosen access path), after compressing SPT01 (APAR PK80375, PTF UK50987), we provide three alternatives here.

- If you have sufficient disk space to temporarily store three times the size of your current SPT01, using REBIND PACKAGE EXTENDED.
- If you do not have sufficient disk space, you can REBIND using REBIND PACKAGE BASIC only on the first REBIND. This stores the DB2 V8 runtime structures and access path in the PREVIOUS version allowing for fallback. Any subsequent REBINDs should be done with REBIND PACKAGE OFF to avoid deleting the DB2 V8 runtime structures and access path.
- If you are really disk space constrained, use the BASIC option, and then perform a more granular rebind (such as, a collection at the time). Once a package is verified, you can use the options to FREE the PREVIOUS and/or ORIGINAL versions of the package to reclaim disk space.

The optimization and RUNSTATS changes are introduced in DB2 9 CM: you will not need to REBIND again in NFM. But you will have to BIND/REBIND in DB2 9 NFM after changes to the applications which are related to new SQL functions (see Chapter 2, “SQL performance” on page 11 for the functions requiring BIND in NFM).

### 9.3.7 Migration steps and performance

In this section, we provide information that relates to the utility performance for the migration of the DB2 catalog from V8 to V9 conversion mode and enabling V9 new-function mode. The measurements in the charts that follow refer to the case study with three different customer scenarios for the timings of the CATMAINT process.

The case study is the same that was used in *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465, so we can also compare the measurements between the two migrations.

The measurement environment for non-data sharing includes:

- DB2 for z/OS V7, V8 and V9
- z/OS V1.7
- 4-way processor 2094 (System z9 processor)
- Enterprise Storage Server (ESS) model 800 DASD
- BP0, BP8K0, BP16K set to 50,000

Tip: Perform RUNSTATS + REBIND on your packages before converting to DB2 9 NFM
The company catalog sizes in Figure 9-2 and Figure 9-3 are:

- Company 1 (C1) catalog size 28.3 GB
- Company 2 (C2) catalog size 15.2 GB
- Company 3 (C3) catalog size 0.698 GB

**Migration to conversion mode**

The CATMAINT utility, in job DSNTIJTC, updates the catalog and you run this utility during migration to a new release of DB2, it effectively runs DDL against the catalog.

We found that the average CATMAINT CPU times (Figure 9-2) were reduced up to half of the DB2 V8 measured times.

We found that the average CATMAINT elapsed time (Figure 9-3) were reduced up to 20% of the DB2 V8 measured times.
The elapsed time for the CATMAINT migration to DB2 V9 conversion mode from V8 is less than the times for DB2 V7 to V8 conversion mode migration. The DB2 V8 migration times were longer because of the work that was done to prepare the catalog for later conversion to Unicode, which implies more getpages and updates in BP0.

Figure 9-4 shows a graph of the comparison data for DB2 V7 to V8 conversion mode and DB2 V8 to V9 conversion mode migration times.

![DB2 Catalog Migration V8 - V9 CM](image)

**Figure 9-4  Comparison of CATMAINT in V8 and V9**

You can use the data in Figure 9-4 to roughly predict the performance and elapsed times for the CATMAINT function on your system. This assumes that your catalog has no major anomalies, and the data is only representative of catalog data sizes less than 30 GB.

**Enabling new-function mode**

The process of ENFM is achieved by running job DSNTIJEN against the catalog. This job performs several functions, and each one must complete successfully. First the CATENFM utility function runs to update the DB2 catalog to start DB2 V9 in ENFM. Next SYSOBJ and SYSPKAGE are converted to new-function mode via an online reorg process.

The job converts the SYSOBJ table space from 8 KB to 16 KB page size via the REORG CONVERT V9 function. The job allocates the shadow data sets that the online reorg uses. If you modified the data set allocations for SYSOBJ, SYSPKAGE, and their related indexes, then check the new allocations to ensure that they are acceptable. The real-time statistics (RTS) tables are then moved into the DB2 catalog, and a CATENFM step is run to switch RTS processing to the newly created DSNDB06 RTS tables.

If you have user-defined indexes on the DB2 catalog tables in table spaces that are modified by the DSNTIJEN job, you need to ensure that correctly sized shadow data sets are available for your indexes. To identify any shadow data sets, you may need to run the SQL shown in Example 9-2.
Example 9-2  Query for shadow data sets

```
SELECT A.NAME, A.TBNAME, B.TSNAME
FROM SYSIBM.SYSINDEXES A,
     SYSIBM.SYSTABLES    B,
     SYSIBM.SYSINDEXPART C
WHERE A.TBNAME    = B.NAME
    AND A.TBCREATOR = B.CREATOR
    AND A.CREATOR   = C.IXCREATOR
    AND A.NAME      = C.IXNAME
    AND B.DBNAME    = 'DSNDB06'
    AND B.CREATEDBY = 'SYSIBM'
    AND A.IBMREQD  <> 'Y'
    AND C.STORTYPE  = 'E'
    AND B.TSNAME IN( 'SYSOBJ','SYSPKAGE' )
ORDER BY B.TSNAME, A.TBNAME, A.NAME;
```

The statistics shown in Figure 9-5 and Figure 9-6 refer to the case study for the timings of the CATENFM process. The CATENFM utility enables a DB2 subsystem to enter DB2 V9 ENFM and then DB2 V9 new-function mode.

In Figure 9-5, we observe that the average CATENFM CPU time is reduced to between one-fourth and one-half of the DB2 V8 measured times.

![V8 – V9 CATENFM CPU Usage Comparison Non-data Sharing](image)

The company catalog sizes are:

- Company 1 (C1) catalog size 28.3 GB
- Company 2 (C2) catalog size 15.2 GB
- Company 3 (C3) catalog size 0.698 GB
In Figure 9-6, we observe that the average CATENFM elapsed time is reduced to about one-fifth of the DB2 V8 measured times. Remember that, in V8, the DSNTIJNE did an online REORG of 18 catalog table spaces.

The graph in Figure 9-7 shows the comparison data for DB2 V8 new-function mode and DB2 V9 new-function mode migration times (CPU and elapsed).

You can use the data in Figure 9-7 to roughly predict the performance and elapsed times for the DSNTIJEN function on your system. This assumes that your catalog has no major anomalies and the data is only representative of catalog data sizes less than 30 GB.
New-function mode

DB2 V9 is switched to running in new-function mode by running the install job DSNTIJNF. This short running job uses the DSNUTILB program with control card option CAFTNFM COMPLETE. When this job is complete, the catalog is fully migrated to DB2 V9 new-function mode.

9.3.8 Summary and recommendations

Migration from DB2 V8 to DB2 V9 conversion mode takes between 1 and 7 minutes for the DSNTIJTC job based on the size of the catalog and directory. Migration from DB2 V9 conversion mode to DB2 V9 new-function mode takes between 7.5 and 15 minutes for the DSNTIJEN job based on the size of the catalog and directory. These results shown are from the testing that we performed on a medium-to-large-sized catalog and directory environments.

We recommend that you do as much cleanup work as possible on your catalog and directory prior to starting the migration process. This cleanup work helps to identify problems before the migration and reduces the amount of data that needs to be processed.

9.3.9 Catalog consistency and integrity checking

We recommend that you use the SDSNSAMP(DSNTESQ) query that is provided to check the consistency of the DB2 catalogs. The pre-migration job DSNTIJP9 should be run on your DB2 V8 system to do an overall health check prior to migrating to DB2 V9. DSNTIJP9 was delivered to DB2 V8 customers via the service stream in APAR PK31841. The equivalent job that shipped with DB2 V9 is DSNTIJPM.

9.4 DSNZPARM changes

In this section, we outline the changes that have occurred in DSNZPARM for the following macros:

- DSN6SPRM
- DSN6SYSP
- DSNHDECM
- DSN6FAC

DSNZPARM changes for the DSN6SPRM macro

The following changes have occurred for the DSN6SPRM macro:

- The ADMTPROC parameter has been added. This parameter specifies the name of the procedure for the Admin Scheduler that is associated with the DB2 subsystem.
- The CACHEDYN_FREELOCAL parameter has been added. This parameter specifies the level at which DB2 can free cached dynamic statements to relieve DBM1 below-the-bar storage.
- The COMPRESS_SPT01 parameter has been added by APAR PTF UK50987. This opaque DB2 subsystem parameter indicates whether the SPT01 directory space should be compressed. Valid settings are NO and YES, the default setting is NO, and the setting can be changed online. The value of COMPRESS_SPT01 affects REORG execution; if it is YES on a member when a REORG is run on SPT01 then SPT01 will be compressed. In DB2 data sharing, all members should use the same setting for COMPRESS_SPT01.
The EDM_SKELETON_POOL value has been added. This value specifies the minimum size in KB of the above-the-bar EDM skeleton pool for skeleton package tables (SKPTs) and skeleton cursor tables (SKCTs). The default is 102400.

The MAX_CONCURRENT_PKG_OPS parameter has been added. This parameter specifies the maximum number of automatic bind requests that can be processed simultaneously. The default is 10.

The MAX_OPT_ELAP value has been removed. This value was the maximum amount of elapsed time in seconds to be consumed by the DB2 Optimizer.

The MINSTOR default value is changed to YES to minimize DBM1 address space virtual storage constraints. When MINSTOR is set to YES, DB2 actively manages storage for individual threads with the goal of minimizing the total storage used by the thread. Laboratory measurements indicate around 1% CPU overhead with MINSTOR=YES, but it is possible for some storage-management-intensive applications to experience higher percentage CPU overhead. If there is no DBM1 address space virtual storage constraint issue, then MINSTOR can be changed to NO to possibly save some CPU time.

The MXDTCH parameter has been added. This value specifies the maximum size, in MB, of memory for data caching that DB2 allocates from above the bar. The default is 20 MB.

The OPTIOGWGT default value is changed to ENABLE. APAR PK61277/PTF UK39140 had added support in the DB2 V9 optimizer for an improved formula for balancing the costs of input/output and CPU speeds. PK75643/.UK42565 changes the default value of OPTIOGWGT from DISABLE to ENABLE

The OPTIXOPREF default value is changed to ON. PK84092 extends OPTIXOPREF to choose INDEX ONLY ACCESS over INDEX+DATA when the index does not support ordering. An index which can provide index-only access and has better overall filtering could have been ignored in favor of an index which needs to access data during access path selection when both indexes do not support any join ordering or ORDER BY / GROUP BY ordering.

The PARAPAR1=YES serviceability option is removed. This option was introduced in DB2 V8 to enable parallelism reduction of APAR PQ87352.

The REOPT(AUTO) value has been added. This values specifies whether to automatically reoptimize dynamic SQL. The default is NO.

The RELCURHL=NO option is removed. This option was used to release, at COMMIT, any page or row locks on which a cursor WITH HOLD is positioned. Check for incompatibility with applications dependent on retaining page or row locks across commits for a WITH HOLD cursor.

The RESTORE_RECOVER_FROM DUMP value has been added. This value specifies for the RESTORE SYSTEM and the RECOVER utilities whether the system-level backup that has been chosen as the recovery base should be used from the disk copy of the system-level backup (NO), or from the dump on tape (YES). The default is NO.

The RESTORE_TAPEUNITS value has been added. This value specifies the maximum tape units that RESTORE SYSTEM can allocate to restore a system-level backup from tape. The default is NOLIMIT.

The SJMXPPOOL value has been removed. This value was the maximum MB of the virtual memory pool for star join queries.

The new SPRMRRF, added via APAR PK85881, can be used to ENABLE or DISABLE the reordered row format at the subsystem level. This can be useful for keeping the format of the table spaces consistent when using DSN1COPY.
The STATCLUS value has been added. This value specifies the default RUNSTATS clustering statistics type. The default is ENHANCED. This parameter should improve access path selection for duplicate and reverse-clustered indexes. For indexes that contain either many duplicate key values or key values that are highly clustered in reverse order, cost estimation that is based purely on CLUSTERRATIOF can lead to repetitive index scans. A new cost estimation formula based on the DATAREPEATFACTORF statistic to choose indexes avoids this performance problem. To take advantage of the new formula, set the STATCLUS parameter to ENHANCED. Otherwise, set the value to STANDARD.

The SUPPRESS_TS_CONV_WARNING has been removed. In Version 9.1, DB2 always operates as though SUPPRESS_TS_CONV_WARNING=NO.

The SYSTEM_LEVEL_BACKUPS value has been added. This value specifies whether the RECOVER Utility should use system-level backups as a recovery base for object-level recoveries. The default is NO.

The TABLES_JOINED_THRESHOLD value has been removed. This value was the minimum number of table joins in a query to cause DB2 to monitor the resources consumed when determining the optimum access path for that query.

The UTILS_DUMP_CLASS_NAME parameter has been added. This parameter specifies the DFSMShsm dump class for RESTORE SYSTEM to restore from a system-level backup dumped to tape. The default is blank.

**DSNZPARM changes for the DSN6SYSP macro**

The following changes have occurred for the DSN6SYSP macro:

- The DBPROTCL=PRIVATE option is removed. This option allowed a default protocol of PRIVATE to be specified. Because PRIVATE is no longer supported as a default, the option has been removed.

- The IDXBPOOL value to specify default buffer pool for CREATE INDEX now accepts 8 KB, 16 KB, and 32 KB page size buffer pools in addition to 4 KB.

- The IMPDSDEF value has been added. This value specifies whether to define the data set when creating an implicit base table space or implicit index space. The default is YES.

- The IMPTSCMP value has been added. This value specifies whether to enable data compression on implicit base table space. The default is NO.

- The MAXOFILR parameter has been added. This specifies the maximum number of data sets that can be open concurrently for processing of LOB file references. The default is 100.

- The MAXTEMSPS value has been added. This value specifies the maximum MB of temporary storage in the workfile database that can be used by a single agent at any given time for all temporary tables. The default is 0 (meaning no limit, the same as previous releases).

- The MGEXTSZ default has changed. This parameter specifies whether to use sliding secondary quantity for DB2-managed data sets. The default has changed from NO to YES.

- The MAXOFILR value has been added. This value specifies the maximum number of open files for LOB file references that are allowed. The default is 100.

- The RESTORE_RECOVER_FROMDUMP parameter has been added. This specifies whether the system-level backup for the RESTORE SYSTEM and the RECOVER utilities is from the disk copy of the system-level backup (NO), or from the dump on tape (YES). The default is NO.
The RESTORE_TAPEUNITS parameter has been added. This parameter specifies the maximum number of tape units or tape drives that the RESTORE SYSTEM utility can allocate when restoring a system-level backup that has been dumped to tape. The default is NOLIMIT.

The STORPROC-procname option is removed. The DB2-established stored procedure address space is no longer supported, and therefore, this option has been removed.

The TBSBP8K value has been added. This value specifies the default 8 KB buffer pool for CREATE TABLESPACE. The default is BP8K0.

The TBSBP16K value has been added. This value specifies the default 16 KB buffer pool for CREATE TABLESPACE. The default is BP16K0.

The TBSBP32K value has been added. This value specifies the default 32 KB buffer pool for CREATE TABLESPACE. The default is BP32K.

The TBSBPLOB value has been added. This value specifies the default buffer pool to use for LOB table spaces that are created implicitly and for LOB table spaces that are created explicitly without the BUFFERPOOL clause. The default is BP0.

The TBSBPXML value has been added. This value specifies the default buffer pool to use for XML table spaces that are created implicitly. The default is BP16K0.

WLMENV has been changed. It specifies the name of the default Workload Manager (WLM) environment for DB2. It now supports a length of 32 characters, which has increased from 18.

The XMLVALA value has been added. This value specifies in KB an upper limit for the amount of storage that each agent can have for storing XML values. The default is 204800 KB.

The XMLVALS value has been added. This value specifies in MB an upper limit for the amount of storage that each system can use for storing XML values. The default is 10240 MB.

**DSNZPARN change for the DSN6FAC macro**

The following change has occurred for the DSN6FAC macro:

- The PRIVATE_PROTOCOL YES/NO option has been added via APAR PK92339. This option is required to prevent any future re-introduction of Private Protocol objects or requests into the subsystem after all packages and plans have been converted from private protocol to DRDA.

In preparation for a future release of DB2 where private protocol capabilities will no longer be available, a new subsystem parameter, PRIVATE_PROTOCOL, a parameter on the DSN6FAC macro, is provided so that the private protocol capabilities can be enabled or disabled at the subsystem level. If no change is made to the existing subsystem parameters (DSNZPxxx) module after applying the fix to this APAR, private protocol capabilities will still be available.

To disable private protocol capabilities in a subsystem, you first create a new DSNZPxxx module, or modify an existing DSNZPxxx module, where the PRIVATE_PROTOCOL parameter of the DSN6FAC macro is set to NO. Once the DSNZPxxx module is created, the DB2 subsystem must be started with the created/modified DSNZPxxx module, or:

- A -SET SYSPARM command must be issued with the LOAD keyword specifying the created/modified DSNZPxxx module, or

- A -SET SYSPARM command must be issued with the RELOAD keyword if the updated module has the same name as the currently-loaded module.

If you need to subsequently re-enable private protocol capabilities on a running subsystem that previously had the capabilities disabled, then you can issue a -SET SYSPARM
command, or restart DB2, specifying a DSNZPxxx module which does not have the
PRIVATE_PROTOCOL parameter set to NO.

Changes for the DSNHDECM macro
For the DSNHDECM macro, the DEF_DECFLOAT_ROUND_MODE value has been added. This value specifies the default rounding mode for decimal floating point data type. The default is ROUND_HALF_EVEN.
Performance tools

DB2 Performance Tools for z/OS help with analyzing performance data, make recommendations to optimize queries, and maintain high availability by sensing and responding to situations that could result in database failures and system outages.

In this chapter, we explain the following topics:

- IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS
- Optimization Service Center and Optimization Expert for z/OS
10.1 IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS

OMEGAMON XE for DB2 Performance Expert on z/OS is a host-based performance analysis and tuning tool for z/OS environments. Its main objective is to simplify DB2 performance management. OMEGAMON XE for DB2 Performance Expert lets you monitor threads, system statistics, and system parameters by using a single tool. It integrates performance monitoring, reporting, buffer pool analysis, and a Performance Warehouse function. It also provides a single overview system that monitors all DB2 subsystems and DB2 Connect gateways in a consistent way.

OMEGAMON XE for DB2 Performance Expert includes the following advanced capabilities:

- Analyzes and tunes the performance of DB2 and DB2 applications
- Provides expert analysis, a real-time online monitor, and a wide range of reports for analyzing and optimizing DB2 applications and SQL statements
- Includes a Performance Warehouse feature for storing performance data and analysis functions, and for collecting report data
- Defines and applies analysis functions, such as rules of thumb and queries, to identify performance bottlenecks
- A starter set of smart features that provide recommendations for system tuning to gain optimum throughput
- An explain feature
- A Reporting function that presents detailed information about DB2 events that involve CPU times, buffer pool usage, locking, I/O activity, and more
- A buffer pool analysis function that collects data and provides reports on related event activity to get information about current buffer pool behavior and simulate anticipated future behavior

It can provide these reports in the form of tables, pie charts, and diagrams.

- Exception reports for common performance problems to help identify and quantify excessive CPU and elapsed time on a plan and package basis
- Monitors connections of remote applications using Performance Expert Agent for DB2 Connect Monitoring

The availability of these functions, however, varies depending on whether you install OMEGAMON XE for DB2 Performance Expert or the stand-alone product DB2 Performance Manager.
Figure 10-1 shows how to monitor the detailed DB2 statistics information by using the OMEGAMON XE for DB2 Performance Expert client.

![DB2 statistics details in OMEGAMON XE for DB2 Performance Expert client](image)

**Figure 10-1  DB2 statistics details in OMEGAMON XE for DB2 Performance Expert client**

<table>
<thead>
<tr>
<th>DBM1 Virtual Storage</th>
<th>DBM1 virtual storage above 2GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average thread footprint (MB)</td>
<td>8.40</td>
</tr>
<tr>
<td>Max. number of threads</td>
<td>3088.13</td>
</tr>
<tr>
<td>Total DBM1 storage (MB)</td>
<td>44</td>
</tr>
<tr>
<td>Total addressable storage (MB)</td>
<td>32</td>
</tr>
<tr>
<td>Total variable storage (MB)</td>
<td>7</td>
</tr>
<tr>
<td>Total fixed storage (MB)</td>
<td>0</td>
</tr>
<tr>
<td>Total SHARED storage (MB)</td>
<td>4</td>
</tr>
<tr>
<td>Total stack storage in use (MB)</td>
<td>3</td>
</tr>
<tr>
<td>Storage cushion (MB)</td>
<td>113</td>
</tr>
<tr>
<td>Fixed storage (MB)</td>
<td>10</td>
</tr>
<tr>
<td>Obtained storage (MB)</td>
<td>2.943</td>
</tr>
<tr>
<td>Total compression dictionary storage (MB)</td>
<td>0</td>
</tr>
<tr>
<td>Cbuff storage (MB)</td>
<td>0</td>
</tr>
<tr>
<td>Variable storage (MB)</td>
<td>113</td>
</tr>
<tr>
<td>Thread copies of cached SQL sths (MB)</td>
<td>18</td>
</tr>
<tr>
<td>In use storage (MB)</td>
<td>0</td>
</tr>
<tr>
<td>HWM for allocated statements (MB)</td>
<td>0</td>
</tr>
<tr>
<td>Shared memory storage (MB)</td>
<td>5</td>
</tr>
<tr>
<td>Total fixed virtual DB shared (MB)</td>
<td>1</td>
</tr>
<tr>
<td>Total addressable virtual DB shared (MB)</td>
<td>2</td>
</tr>
<tr>
<td>Total variable virtual DB shared (MB)</td>
<td>0</td>
</tr>
</tbody>
</table>
The OMEGAMON XE for DB2 Performance Expert client supports the new storage layout of the EDM pool in DB2 V9. See Figure 10-2.

![Figure 10-2 EDM pool information in OMEGAMON XE for DB2 Performance Expert client](image)

The batch reporting facility presents historical information about the performance of the DB2 system and applications in reports and data sets. System-wide performance data shows information about topics such as CPU times, buffer pool usage, locking, log activity and I/O activity. Application data shows how individual programs behave in DB2.

The batch reporting facility uses DB2 instrumentation data to generate performance reports in a form that is easy to understand and analyze.
You can use OMEGAMON XE for DB2 Performance Expert to:

- Determine DB2 subsystem performance and efficiency
- Identify and resolve potential problems
- Tune the DB2 subsystem
- Measure an application’s performance and resource cost
- Tune applications and SQL queries
- Assess an application’s effect on other applications and the system
- Gather information for cost purposes

OMEGAMON XE for DB2 Performance Expert provides information at various levels of detail depending on your needs.

**Note:** OMEGAMON XE for DB2 Performance Expert V4.1 supports DB2 V9 when APAR PK36297 and PK40691 are applied.

**Recent changes on virtual storage analysis**

Some formulas for virtual storage management, based mostly on the content of IFCID 225 records, have changed with recent maintenance:

PK49126 / UK28017 for PE V2
PK50719 / UK28017 for PM V8
PK49139 / UK27963 for OMPE V3
PK49139 / UK27964 for OMPE V4

The changed values are reported in the Statistics Details panel (shown in Figure 10-1 on page 281) and in the batch statistics report.

**Average thread footprint (MB)**

Average thread footprint (MB) now shows the current average memory usage of user threads (allied threads + DBATs). It is calculated as follows:

\[
\frac{\text{TOT}_\text{VAR}_\text{STORAGE} - \text{TOT}_\text{AGENT}_\text{SYS}_\text{STRG}}{\text{ACT}_\text{ALLIED}_\text{THREADS} + \text{ACTIVE}_\text{DBATS}}
\]

**Maximum number of possible threads**

The maximum number of threads is shown as an integer value both in the statistics reports and the classic online interface. It shows the maximum number of possible threads. Its value depends on the storage size and the average memory usage of active user threads. It is calculated as:

\[
\frac{\text{EXT}_\text{REG}_\text{SIZE}_\text{MAX} - (200*1024*1024) - (\text{TOT}_\text{GETM}_\text{STORAGE} + \text{TOT}_\text{GETM}_\text{STCK}_\text{STOR})}{\left(\frac{\text{TOT}_\text{VAR}_\text{STORAGE} - \text{TOT}_\text{AGENT}_\text{SYS}_\text{STRG}}{\text{ACT}_\text{ALLIED}_\text{THREADS} + \text{ACTIVE}_\text{DBATS}}\right)}
\]

In addition, for the batch reports, another pair of values is shown.

**Average thread footprint (TYPE II) (MB)**

The average thread footprint is the current average memory usage of active allied threads and the maximum number of active DBATs that existed. The formula that is used for this value is suited for enterprise resource planning (ERP) systems, such as SAP. The formula is:

\[
\frac{\text{TOT}_\text{VAR}_\text{STORAGE} - \text{TOT}_\text{AGENT}_\text{SYS}_\text{STRG} - \text{LOC}_\text{DYN}_\text{CACHE}_\text{POOL}}{\text{ACT}_\text{ALLIED}_\text{THREADS} + \text{MAX}_\text{ACTIVE}_\text{DBATS}}
\]
Maximum number of possible threads (TYPE II)

The maximum number of possible threads depends on the storage size and average memory usage of active allied threads and the maximum number of active DBATs that existed. The formula is:

\[(\text{EXT\_REG\_SIZE\_MAX} - \text{BIT\_EXT\_LOW\_PRI\_31}) - \text{MIN}((\text{EXT\_REG\_SIZE\_MAX}/8),(200*1024*1024)) - (\text{TOT\_GETM\_STORAGE} + \text{TOT\_FIXED\_STORAGE} + \text{LOC\_DYN\_CACHE\_POOL} + \text{TOT\_AGENT\_SYS\_STRG})/((\text{TOT\_VAR\_STORAGE} - \text{TOT\_AGENT\_SYS\_STRG} - \text{LOC\_DYN\_CACHE\_POOL}))/(\text{ACT\_ALLIED\__THREADS} + \text{MAX\_ACTIVE\_DBATS}))\]

10.2 Optimization Service Center and Optimization Expert

In this section, we explain IBM Optimization Service Center for DB2 for z/OS and DB2 Optimization Expert for z/OS. For more information see IBM DB2 9 for z/OS: New Tools for Query optimization, SG24-7421.

10.2.1 IBM Optimization Service Center

IBM Optimization Service Center for DB2 for z/OS is a workstation tool for monitoring and tuning the SQL statements that run as part of a workload on your DB2 for z/OS subsystem. Optimization Service Center has replaced and enhanced Visual Explain.

When you start Optimization Service Center for the first time, you see the Welcome panel (Figure 10-3).

Figure 10-3   Optimization Service Center welcome panel
From the Welcome panel, you can select to configure a DB2 subsystem, view query activity, tune a single query, view workloads, tune a workload, or view monitor profiles.

- Reactive tuning: Optimization tools for problem SQL queries
  - You can use Optimization Service Center to identify and analyze problem SQL statements and to receive expert advice about statistics that you might gather to improve the performance of problematic and poorly performing SQL statements on a DB2 for z/OS subsystem.
  - Optimization Service Center makes it easy for you to get and implement expert statistics recommendations. After you identify a problem query, you can run the Statistics Advisor, which provides advice about statistics that you should update or collect to improve the performance of a query. Optimization Service Center generates control statements that can be used to collect and update needed statistics with the RUNSTATS utility and allows you to invoke the RUNSTATS utility directly from your workstation.

You can view query activity from a number of sources, including the DB2 catalog (plan or package), statement cache, Query Management Facility (QMF), and so on. You can even import a query from a file or copy and paste the query text into your project. See Figure 10-4.

Figure 10-4  Optimization Service Center view queries
You can quickly “snap the cache” of your DB2 subsystem to understand which dynamic queries have been running and to discover which of those queries might be causing performance problems on your DB2 for z/OS subsystem.

- **Proactive tuning: Optimization tools for monitoring and tuning SQL workloads**
  - You can use Optimization Service Center to identify and analyze groups of statements and receive expert advice about statistics that you can gather to improve the performance of entire SQL workloads.
  - An **SQL workload** is any group of statements that run on your DB2 subsystem. You define a workload by creating a set of criteria that identifies the group of statements that make up the workload. For example, you might create a workload to capture all of the statements in a particular application, plan, or package.
  - When you have optimized the performance of SQL queries that run on your DB2 for z/OS subsystem, you can create monitor profiles that monitor the health of SQL processing on the subsystem and alert you when problems develop and when more tuning activities might be advised.
  - You can create monitor profiles that record information about the normal execution of static and dynamic SQL statements.
  - You can create monitor profiles that record information about the execution of SQL statements when the execution exceeds specific thresholds during processing.
  - You can capture a set of SQL statements from a DB2 for z/OS subsystem. You might capture statements from the statement cache, from the DB2 catalog, from QMF, from a file on your workstation, or from another workload. You can specify filter criteria to limit the number and type of statements that you capture.
  - You can use the workload Statistics Advisor to gain expert advice regarding statistics that you can collect or update to improve the overall performance of the statements that make up an SQL workload. Optimization Service Center generates control statements that can be used to collect and update needed statistics with the RUNSTATS utility and even allows you to invoke the RUNSTATS utility directly from your workstation.

The Statistics Advisor recommends statistics that you should update or collect to improve the performance of a particularly hot query. See Figure 10-5.
Advanced tuning: Optimization tools for experienced database administrators

- Powerful Optimization Service Center optimization tools enable the experienced DBA to understand, analyze, format, and optimize the SQL statements that run on a DB2 for z/OS subsystem.
- By using Optimization Service Center, you can automatically gather information from DB2 EXPLAIN, which is an SQL function that populates tables with information about the execution of SQL statements. The primary use of EXPLAIN is to capture the access paths of your statements. EXPLAIN can help you when you need to perform the following tasks:
  - Determine the access path that DB2 chooses for a query
  - Design databases, indexes, and application programs
  - Determine when to rebind an application
- Optimization Service Center automatically formats an SQL statement into its component query blocks and presents the query so that you can read it. You can click a tab to see how the DB2 optimizer transforms a query for processing.
- Optimization Service Center provides graphical depictions of the access plans that DB2 chooses for your SQL statements. Such graphs eliminate the need to manually interpret plan table output. The relationships between database objects, such as tables and indexes, and operations, such as tablespace scans and sorts, are clearly illustrated in the graphs. You can use this information to tune your queries.
– Optimization Service Center provides a way for experienced DBAs to graphically create plan hints that specify an access method for all or part of the access plan for an SQL statement and deploy those plan hints to the DB2 for z/OS subsystem.

### 10.2.2 DB2 Optimization Expert for z/OS

IBM DB2 Optimization Expert for z/OS has the same Statistics Advisor as Optimization Service Center. However Optimization Expert also offers a comprehensive set of index, query, and access path advisors to improve system performance and lower total cost of ownership.

You can use Optimization Expert to identify and analyze groups of statements and receive expert advice about measures that you can take to improve the performance of entire SQL workloads. Optimization Expert makes it easy for you to get and implement expert tuning recommendations. After you identify a problem query, you can run any or all of the expert advisors.

Optimization Expert generates control statements that you can use to collect and update needed statistics with the RUNSTATS utility and allows you to invoke the RUNSTATS utility directly from your workstation. The RUNSTATS jobs of DB2 Optimization Expert captures statistics beyond those that you get by using RUNSTATS ALL.

The Query Advisor recommends ways that you can rewrite an SQL query to improve performance. See Figure 10-6.

![Optimization Expert Query Advisor](image)

**Figure 10-6   Optimization Expert Query Advisor**
Optimization Expert considers a number of different conditions and recommends best-practice fixes to common query-writing mistakes. The Index Advisor recommends indexes that you might create or modify to enhance the performance of an SQL query. See Figure 10-7.

The Index Advisor also generates the CREATE INDEX statements that you can use to implement the recommendations, and enables you to execute those statements on the DB2 subsystem directly from your workstation.
Summary of relevant maintenance

In this appendix, we look at recent maintenance for DB2 9 for z/OS that generally relates to performance, availability, and new functionality:

- Performance enhancements APARs
- Functional enhancements APARs
- z/OS APARs

These lists of APARs represent a snapshot of the current maintenance for performance, availability, and new functions at the moment of writing. As such, the list will become incomplete or even incorrect at the time of reading. It is here to identify areas of performance or functional improvements.

Make sure that you contact your IBM Service Representative for the most current maintenance at the time of your installation. Also check on RETAIN® for the applicability of these APARs to your environment, as well as to verify pre- and post-requisites.

We recommend that you use the Consolidated Service Test (CST) as the base for service.
A.1 Performance enhancements APARs

We present a list of performance APARs (Table A-1) that are applicable to DB2 9.

Table A-1  DB2 V9 current performance-related APARs

<table>
<thead>
<tr>
<th>APAR #</th>
<th>Area</th>
<th>Text</th>
<th>PTF and notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK29281</td>
<td>Open data sets</td>
<td>Increasing limit to 100,000 objects per subsystem</td>
<td>PK51503</td>
</tr>
<tr>
<td>PK34251</td>
<td>TEMPLATE</td>
<td>Three new TEMPLATE utility keywords, SUBSYS, LRECL, and RECFM, to allow dynamic allocation of a DDNAME to use MVS BATCHPIPES SUBSYSTEM</td>
<td>UK25291 also for V8</td>
</tr>
<tr>
<td>PK37290</td>
<td>MQ user-defined functions</td>
<td>New implementation of DB2 MQ user defined functions based on MQI</td>
<td>UK30229 also for V8</td>
</tr>
<tr>
<td>PK37354</td>
<td>IFC</td>
<td>IFCID 225 to collect buffer manager storage blocks below the bar. It is now written to an SMF type 100 record with a subtype of 4 (was 102) and the ROLE qualifier is allowed for AUDIT traces.</td>
<td>UK25045</td>
</tr>
<tr>
<td>PK38867</td>
<td>Data sharing</td>
<td>DB2 support for z/OS V1R9.0 Sysplex Routing Services. zIIP awareness enhancements</td>
<td>UK35519 also for V8</td>
</tr>
<tr>
<td>PK40878</td>
<td>Sequence</td>
<td>Reducing log write at sequence update</td>
<td>UK29378 also for V8</td>
</tr>
<tr>
<td>PK41165</td>
<td>Star Join</td>
<td>Performance</td>
<td>UK26800</td>
</tr>
<tr>
<td>PK41323</td>
<td>DROP</td>
<td>Improve performance for implicitly created table space</td>
<td>UK24934</td>
</tr>
<tr>
<td>PK41380</td>
<td>Serviceability</td>
<td>Queries with index scan</td>
<td>UK29529 also in V8</td>
</tr>
<tr>
<td>PK41380</td>
<td>Service</td>
<td>Remove serviceability code from DSNIOST2 mainline and put it in error routine</td>
<td>UK29529 also in V8</td>
</tr>
<tr>
<td>PK41711</td>
<td>CHECK</td>
<td>New DSNZPARM (UTIL_TEMP_STORCLAS) that will allow customers to specify a storage class that points to a storage group with non-PPRC volumes</td>
<td>PK41370 also for V8</td>
</tr>
<tr>
<td>PK41878</td>
<td>Queries</td>
<td>CPU overhead for queries on a partitioned table space with DPSI</td>
<td>UK24125</td>
</tr>
<tr>
<td>PK41899</td>
<td>Utilities</td>
<td>SORTNUM elimination - 1</td>
<td>UK33636</td>
</tr>
<tr>
<td>PK42005</td>
<td>Performance</td>
<td>Unnecessary calls for DBET checking</td>
<td>UK27088 also in V7 and V8</td>
</tr>
<tr>
<td>PK42008</td>
<td>Locking</td>
<td>Excessive partition locking</td>
<td>UK24545</td>
</tr>
<tr>
<td>PK42409</td>
<td>Optimizer XML</td>
<td>Possible poor query performance due to I/O or CPU cost estimate being too high for XML index</td>
<td>UK26798</td>
</tr>
<tr>
<td>PK42801</td>
<td>INSERT</td>
<td>Extra save calls in DSNIOPNP resulted in performance degradation</td>
<td>UK29358</td>
</tr>
<tr>
<td>PK43315</td>
<td>Load/Unload XML</td>
<td>Whitespace and CCSID corrections</td>
<td>UK25744</td>
</tr>
<tr>
<td>PK43475</td>
<td>SQL procedures</td>
<td>Native SQL procedures tracing unnecessarily enabled</td>
<td>UK25006</td>
</tr>
<tr>
<td>PK43765</td>
<td>DGTT</td>
<td>Workfile prefetch quantity (coming from TEMP DB) too small for declared global temporary table (DGTT)</td>
<td>UK26678</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>---------</td>
<td>--------------</td>
<td>----------------------------------------------------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>PK44026</td>
<td>Utilities</td>
<td>Enable dynamic prefetch for:</td>
<td>UK25896</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- UNLOAD phase of REORG INDEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- UNLOAD and BUILD phases of REORG TABLESPACE PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- BUILD phase of LOAD REPLACE PART</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- RUNSTATS phase of RUNSTATS INDEX</td>
<td></td>
</tr>
<tr>
<td>PK44133</td>
<td>New RTS</td>
<td>DSNACCOX allows more meaningful REORG thresholds</td>
<td>UK32795</td>
</tr>
<tr>
<td>PK45916</td>
<td>Utilities</td>
<td>SORTNUM elimination - 2</td>
<td>UK33962 V8</td>
</tr>
<tr>
<td>PK46082</td>
<td>Clone tables</td>
<td>Make CLONE table use the same set of statistics and optimization algorithm as base table</td>
<td>UK28249</td>
</tr>
<tr>
<td>PK46687</td>
<td>Optimization</td>
<td>Virtual Index support retrofitted to V8</td>
<td>UK33731 V8</td>
</tr>
<tr>
<td>PK46972</td>
<td>DPSI</td>
<td>System hang or deadlock with DEGREE ANY and DPSI</td>
<td>UK28211</td>
</tr>
<tr>
<td>PK47318</td>
<td>REOPT(AUTO)</td>
<td>Failure to optimize with successive range predicates</td>
<td>UK31630</td>
</tr>
<tr>
<td>PK47594</td>
<td>XML</td>
<td>Performance improvement for large documents LOAD</td>
<td>UK31768</td>
</tr>
<tr>
<td>PK48453</td>
<td>XML</td>
<td>Logical partition reset of XML NPIs changed to use dynamic prefetch</td>
<td>UK28407</td>
</tr>
<tr>
<td>PK48500</td>
<td>Query</td>
<td>With OR predicates, a multi-index plan may not be chosen by DB2 even if it can provide better performance</td>
<td>UK28261</td>
</tr>
<tr>
<td>PK49348</td>
<td>REOPT(AUTO)</td>
<td>Improve performance of feature REOPT(AUTO) with dynamic statement caching with many short prepares</td>
<td>UK43584</td>
</tr>
<tr>
<td>PK49972</td>
<td>SIGN ON</td>
<td>Reduce Getmain and Freemain activity during DB2 SIGN ON processing for recovery coordinators such as CICS or IMS</td>
<td>UK33510 (UK33509 in V8)</td>
</tr>
<tr>
<td>PK50575</td>
<td>XML</td>
<td>Add zAAP information in SMF101 accounting</td>
<td>UK36263</td>
</tr>
<tr>
<td>PK51099</td>
<td>Data sharing</td>
<td>Avoid excessive conditional lock failures of data page P-lock in data sharing inserts</td>
<td>UK29376 also V8</td>
</tr>
<tr>
<td>PK51853</td>
<td>REORG/LOAD</td>
<td>Allow &gt;254 compressed partitions</td>
<td>UK31489 (also V8)</td>
</tr>
<tr>
<td>PK51976</td>
<td>Diagnosis Guide</td>
<td>This APAR provides the DB2 9 Diagnosis Guide and Reference in PDF format: LY37-3218-01. The same version of</td>
<td>UK30713</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this document is available on the latest DB2 for z/OS licensed library collection. Also in DB2 data set DSN910.SDSNIVPD(DSNDR).</td>
<td></td>
</tr>
<tr>
<td>PK52522</td>
<td>BIND</td>
<td>REBIND PACKAGE will preserve multiple package copies, and allow users to switch back to an older copy.</td>
<td>UK31993</td>
</tr>
<tr>
<td>PK52523</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK54327</td>
<td>LOAD</td>
<td>Many locks for DBET UPDATE are obtained in load utility to partition tablespace with NPI</td>
<td>UK31069 also V8</td>
</tr>
<tr>
<td>PK54988</td>
<td>CPU</td>
<td>Reduce excessive CPU time in DSNB1CPS, especially with many partitions</td>
<td>UK31903</td>
</tr>
<tr>
<td>PK55783</td>
<td>XML</td>
<td>Using XML indexes in join</td>
<td>UK46726</td>
</tr>
<tr>
<td>PK55966</td>
<td>XML</td>
<td>Remove page locks for readers</td>
<td>UK33456</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>PK56337</td>
<td>XML</td>
<td>XPath performance</td>
<td>UK38379</td>
</tr>
<tr>
<td>PK66539</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK56356</td>
<td>IFCID 001</td>
<td>It adds z/OS metrics CPU % in four granularities (LPAR, DB2 subsystem, MSTR, DBM1) to IFCID 001 in a new data section (#13)</td>
<td>UK33795 (UK32198 in V8)</td>
</tr>
<tr>
<td>PK57429</td>
<td>Stored procedures</td>
<td>CPU parallelism</td>
<td>UK36966 V8 as well</td>
</tr>
<tr>
<td>PK57786</td>
<td>XML</td>
<td>Performance improvement for XML query with XMLAGG</td>
<td>UK33048</td>
</tr>
<tr>
<td>PK58291</td>
<td>z/OS EAV</td>
<td>Enhancement to DB2-supplied administrative stored procedures and EAV support</td>
<td>UK36131 UK35902 also V8</td>
</tr>
<tr>
<td>PK58914</td>
<td>XML</td>
<td>XMLTABLE performance</td>
<td>UK37755</td>
</tr>
<tr>
<td>PK60956</td>
<td>Utilities</td>
<td>Poor elapsed time in SORTBLD phase of REORG or REBUILD INDEX utilities due to extend processing</td>
<td>UK34808 also in V8</td>
</tr>
<tr>
<td>PK61277</td>
<td>Optimizer</td>
<td>Support for an improved formula for balancing the costs of input/output and CPU speeds</td>
<td>UK39140</td>
</tr>
<tr>
<td>PK61759</td>
<td>Sort in Utilities</td>
<td>Improved processing in the RELOAD phase of LOAD and REORG TABLESPACE utilities, and also to utilities that invoke DFSORT.</td>
<td>UK36306 also in V8</td>
</tr>
<tr>
<td>PK62009</td>
<td>Workfile</td>
<td>SQLCODE904 on DGTT pageset using ON COMMIT DELETE</td>
<td>UK35215</td>
</tr>
<tr>
<td>PK62027</td>
<td>LOBs</td>
<td>Allow for DB2 9 LOB locking enhancements without requiring group-wide outage</td>
<td>UK38906</td>
</tr>
<tr>
<td>PK62214</td>
<td>Asymmetric index page split</td>
<td>Abend or less than optimum space utilization during INSERT in a data sharing environment</td>
<td>UK39357</td>
</tr>
<tr>
<td>PK64430</td>
<td>Workfile</td>
<td>False contentions on workfile database</td>
<td>UK37344</td>
</tr>
<tr>
<td></td>
<td>only V8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK66218</td>
<td>XML</td>
<td>INSERT performance</td>
<td>UK38971</td>
</tr>
<tr>
<td>PK65220</td>
<td>LOBs</td>
<td>Add LOBs to the APPEND YES option</td>
<td>UK41212</td>
</tr>
<tr>
<td>PK66539</td>
<td>XML</td>
<td>The performance of XMLQuery/XMLTable/XMLExists e enhanced to avoid multiple scans and extensive usage of storage.</td>
<td>UK44488</td>
</tr>
<tr>
<td>PK67301</td>
<td>Logging/workfile</td>
<td>Increased elapsed time in V9 when processing many insert/mass delete sequences in same commit scope on Declared Global Temporary Tables (DGTT)</td>
<td>UK38947</td>
</tr>
<tr>
<td>PK67691</td>
<td>Workfile</td>
<td>False contentions on workfile database due to s-lock instead of is-lock granted on workfile table spaces. Forward fit of PK64430 for V8, see also PK70060</td>
<td>UK47354</td>
</tr>
<tr>
<td>PK68246</td>
<td>DPSI</td>
<td>Index look aside is not used when a DPSI index is used</td>
<td>UK40096 also V8</td>
</tr>
<tr>
<td>PK68265</td>
<td>XML</td>
<td>Deadlocking with concurrent updates</td>
<td>UK51619</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>PK68325</td>
<td>LOAD and COMPRESS YES</td>
<td>Running a LOAD SHRLEVEL NONE RESUME YES COMPRESS YES on a partitioned table space with some parts empty will cause excessive GETPAGES during the utilinit phase</td>
<td>UK37623, also V7 and V8</td>
</tr>
<tr>
<td>PK69079</td>
<td>Query</td>
<td>Increasing BPsize causes wrong index chosen in NLJ</td>
<td>UK39139, also in V8</td>
</tr>
<tr>
<td>PK69346</td>
<td>Query</td>
<td>Poor index chosen with matching and screening predicates</td>
<td>UK39559</td>
</tr>
<tr>
<td>PK70060</td>
<td>Workfile</td>
<td>Mass delete performance with Declared Global Temporary Tables (DGTT)</td>
<td>UK46839</td>
</tr>
<tr>
<td>PK70269</td>
<td>USS Pipes</td>
<td>Enhancement to the TEMPLATE control statement to dynamically allocate a z/OS UNIX file (HFS) for the LOAD and UNLOAD utilities.</td>
<td>UK43947</td>
</tr>
<tr>
<td>PK70789</td>
<td>Query</td>
<td>Filter factor for (c1,c2,...,cn) in subquery incorrect when evaluating potential indexes</td>
<td>UK39739</td>
</tr>
<tr>
<td>PK71121</td>
<td>Sort</td>
<td>Distinct performance improvement</td>
<td>UK39796</td>
</tr>
<tr>
<td>PK73454</td>
<td>Global optimization</td>
<td>Incorrupt on a SELECT with an IN subquery predicate or EXISTS subquery predicate</td>
<td>UK43794</td>
</tr>
<tr>
<td>PK74778</td>
<td>Optimizer</td>
<td>Completed removal of V9 enhancement on range optimization for instances of regression</td>
<td>UK42199</td>
</tr>
<tr>
<td>PK74993</td>
<td>COPY</td>
<td>Better performance for stacked data sets to tape with better tape marks handling</td>
<td>UK45791, also V8</td>
</tr>
<tr>
<td>PK75149</td>
<td>UTS</td>
<td>Mass delete incorrupt</td>
<td>UK43199</td>
</tr>
<tr>
<td>PK75216</td>
<td>LOBs FRV LOAD/UNLOAD</td>
<td>Enhance the performance for the LOAD and UNLOAD of LOBs using file reference variables by improving the process for the open and close of PDS datasets.</td>
<td>UK43355</td>
</tr>
<tr>
<td>PK75618</td>
<td>Sparse index</td>
<td>Incorrupt with sparse index and a join column with a length that exceeds 256 bytes.</td>
<td>UK43576, also V8</td>
</tr>
<tr>
<td>PK75626</td>
<td>WLM managed buffer pool</td>
<td>Provide automatic bufferpool management using WLM support (see OA18461)</td>
<td>UK53198</td>
</tr>
<tr>
<td>PK75643</td>
<td>DSZPARM</td>
<td>DB2 subsystem parameter OPTIOWGT set to ENABLE by default</td>
<td>UK42565</td>
</tr>
<tr>
<td>PK76100</td>
<td>Index anding</td>
<td>A new DB2 subsystem parameter, is introduced: EN_PJSJ. If set to ON, it enables the star join method dynamic index ANDing (also called pair wise join).</td>
<td>UK44120</td>
</tr>
<tr>
<td>PK76676</td>
<td>Stored procedures/ WLM</td>
<td>PK57429 made some changes that could result in enclaves running in subsystem DB2 instead of DDF. It involves WLM APAR OA26104.</td>
<td>UK47686 (V8 only)</td>
</tr>
<tr>
<td>PK76738</td>
<td>Insert in segmented table space</td>
<td>High number of getpages during insert into segmented table space</td>
<td>UK46982, also V8</td>
</tr>
<tr>
<td>PK77060</td>
<td>Optimizer</td>
<td>Incorrupt missing data access partitioned key table space with nested and or predicate conditions &amp; host vars / parameter marker.</td>
<td>UK42199</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>PK77184</td>
<td>Triggers</td>
<td>CPU accounting for trigger package executing on zIIP. A small portion of trigger processing does run on zIIP. If a trigger is called on a DIST transaction (enclave SRB) the trigger will potentially run on a zIIP. If the trigger is called from an SP or UDF, it will not be zIIP eligible since the trigger will be run on a WLM TCB.</td>
<td>UK43486 V8 only</td>
</tr>
<tr>
<td>PK77426</td>
<td>Optimizer</td>
<td>OPTIXOPREF default ON</td>
<td>UK51283</td>
</tr>
<tr>
<td>PK79236</td>
<td>Optimizer</td>
<td>Unpredictable incorrect result can be returned when running the problem queries with UK42199 applied</td>
<td>UK44461</td>
</tr>
<tr>
<td>PK80375</td>
<td>SPT01</td>
<td>New function</td>
<td>UK50987 also V8</td>
</tr>
<tr>
<td>PK80735</td>
<td>XML</td>
<td>For XMLQUERY or XMLEXISTS, the performance of the XPath scanning algorithm has been optimized. For XMLEXISTS containing XPath function fn:upper-case(), fn:substring(), fn:starts-with(), fn:exists() or fn:not(), DB2 now exploits XML value indexes.</td>
<td>UK52413</td>
</tr>
<tr>
<td>PK81062</td>
<td>MQT</td>
<td>Allow usage of parameter markers in dynamic SQL and creation of a clustered index for MQTs</td>
<td>UK59314</td>
</tr>
<tr>
<td>PK81471</td>
<td>Space search for APPEND</td>
<td>Favor extend physical space instead of exhaustive search when inserting to the end of the non-segmented table space</td>
<td>UK59782</td>
</tr>
<tr>
<td>PK82360</td>
<td>UTS</td>
<td>Correct the space tracking in singleton delete scenarios on universal table spaces</td>
<td>UK52037</td>
</tr>
<tr>
<td>PK82631</td>
<td>XML</td>
<td>Performance enhancement for XMLEXISTS predicate by avoiding XPATHT evaluation via index entry checking</td>
<td>UK56037</td>
</tr>
<tr>
<td>PK83397</td>
<td>REORG PBG</td>
<td>ABEND04E RC00E40318 during REORG PART x of a Partition By Growth table space introduced new ZPARM REORG_IGNORE_FREESPACE</td>
<td>UK50932</td>
</tr>
<tr>
<td>PK83735</td>
<td>Log writes</td>
<td>Remove the forced log writes when inserting into a newly formatted/allocated page for a GBP dependent segmented or universal table space</td>
<td>UK51613 also V8 (PK17813)</td>
</tr>
<tr>
<td>PK85889</td>
<td>Utilities</td>
<td>Allow zIIP rerout for DFSORT (PK85856)</td>
<td>UK48912</td>
</tr>
<tr>
<td>PK88970</td>
<td>LOAD</td>
<td>A more precise estimation of keys being sorted for each index via NUMRECS parameter</td>
<td>UK56739 also V8</td>
</tr>
<tr>
<td>PK88972</td>
<td>LOAD</td>
<td>Enable the NUMRECS option of LOAD TABLESPACE</td>
<td>UK56758 also V8</td>
</tr>
<tr>
<td>PK88974</td>
<td>UNLOAD</td>
<td>UNLOAD utility is enhanced to generate the NUMRECS parameter instead of SORTKEYS</td>
<td>UK58025 also V8</td>
</tr>
<tr>
<td>PK90032</td>
<td>XML</td>
<td>Add SQL scalar function SYSIBM.DSN.XMLVALIDATE using z/OS XMLSS which can validate up to a maximum length of 2G-1 bytes and use zIIP/zAAP</td>
<td>UK57388</td>
</tr>
<tr>
<td>PK90040</td>
<td>XML</td>
<td>Add SQL scalar function SYSIBM.DSN.XMLVALIDATE using z/OS XMLSS which can validate up to a maximum length of 2G-1 bytes and use zIIP/zAAP</td>
<td>UK58408</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>PM00068</td>
<td>DSMAX</td>
<td>Support up to 100000 open data sets in the DB2 DBM1 address space</td>
<td>UK58205</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>also V8</td>
</tr>
<tr>
<td>PM02528</td>
<td>Workfiles</td>
<td>New offline zparm WFDBSEP for a hard barrier between DGTT and non-DGTT space use in workfile database</td>
<td>UK56046</td>
</tr>
<tr>
<td>PM04272</td>
<td>INSERT</td>
<td>SQL Insert poor performance when the candidate page</td>
<td>UK55684</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is at the end of the non UTS table space</td>
<td>also V8</td>
</tr>
<tr>
<td>PM06626</td>
<td>IFC</td>
<td>IFCID148 IFI READS and IFCID3 accounting records to reflect that a nested activity is invoked (i.e., stored procedure, user-defined function, or trigger)</td>
<td>UK55386</td>
</tr>
<tr>
<td>PM06953</td>
<td>CP parallelism</td>
<td>DB2 now creates only one enclave for all parallel tasks under a query. This allows WLM to manage each query as a whole and properly distinguish between high-consumption queries and low-consumption queries.</td>
<td>UK58871</td>
</tr>
<tr>
<td>PM11509</td>
<td>Pair-wise join</td>
<td>Remove possible suspension when running a pair-wise join query.</td>
<td>UK60190</td>
</tr>
<tr>
<td>PM12256</td>
<td>DRDA</td>
<td>zIIP related performance improvement for DRDA over TCP/IP connections from remote clients.</td>
<td>UK56725</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>also V8</td>
</tr>
<tr>
<td>PM12935</td>
<td>INSERT</td>
<td>Poor performance with SQL inserts in classic partition since too many space map pages searched to find the candidate page</td>
<td>UK57872</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>also V8</td>
</tr>
<tr>
<td>PM14757</td>
<td>Space map</td>
<td>Reduce global contention on the simple and classic partition table spaces space map pages</td>
<td>UK60965</td>
</tr>
<tr>
<td>PM15729</td>
<td>UTS, PBG</td>
<td>Slow performance for a DROP TABLESPACE or when running certain utilities for a partition-by-growth table space which has only one partition allocated</td>
<td>UK58601</td>
</tr>
<tr>
<td>PM16020</td>
<td>Query parallelism</td>
<td>New ZPARM PARA_EFF to control how much cost reduction (0 to 100%) based on parallelism. 50 is the suggested value.</td>
<td>UK59310</td>
</tr>
<tr>
<td>PM17336</td>
<td>Workfiles</td>
<td>DB2 REXX exec DSNTWFG updated to create the WORKFILE database according to the WORKFILE preferences entered on installation panel DSNTIP9</td>
<td>UK69608</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>also V10</td>
</tr>
<tr>
<td>PM17542</td>
<td>OPEN/CLOSE</td>
<td>Support performance functions of z/OS R12 for opening and closing data sets</td>
<td>UK60888</td>
</tr>
<tr>
<td>PM18557</td>
<td>OPEN/CLOSE</td>
<td>Support performance functions of z/OS R12 for opening and closing data sets</td>
<td>UK59888</td>
</tr>
<tr>
<td>PM19584</td>
<td>LOAD</td>
<td>Support for PRESORTED and FORMAT INTERNAL</td>
<td>UK68099</td>
</tr>
<tr>
<td>PM22768</td>
<td>Access path</td>
<td>Remove regression for an SQL statement containing a Boolean term non-correlated subquery predicate after applying PM07464</td>
<td>UK61778</td>
</tr>
<tr>
<td>PM24812</td>
<td>IFCID002 IFCID003 counters</td>
<td>Counters are updated to include only the number of rows that are updated, deleted, or inserted as a direct result of user actions, not needed by DB2 for that action</td>
<td>UK64595</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>PM25525</td>
<td>REORG</td>
<td>New keyword PARALLEL has been added to the REORG TABLESPACE LIST for processing partitions in parallel (sort space vs. performance).</td>
<td>UK64590</td>
</tr>
<tr>
<td>PM27249</td>
<td>RUNSTATS</td>
<td>Fix uneven distribution in quantiles during RUNSTATS collection of HISTOGRAM statistics</td>
<td>UK67817</td>
</tr>
<tr>
<td>PM28625</td>
<td>IRLM</td>
<td>Reduce number high getpage requests per INSERT due to erroneous long duration IRLM locks</td>
<td>UK65362</td>
</tr>
<tr>
<td>PM30619</td>
<td>Data sharing</td>
<td>Reduce notify messages being sent for class castout</td>
<td>UK64918</td>
</tr>
<tr>
<td>PM35824</td>
<td>LOAD</td>
<td>Support for PRESORTED and FORMAT INTERNAL</td>
<td>UK68100</td>
</tr>
<tr>
<td>PM37112</td>
<td>Online Reorg</td>
<td>Enhancement for LOG phase of REORG SHRLEVEL CHANGE at part level</td>
<td>Packaging</td>
</tr>
<tr>
<td>PM37293</td>
<td>REORG</td>
<td>New ZPARM REORG_LIST_PROCESSING to control the default behavior of REORG TABLESPACE LIST partition processing at system level</td>
<td>UK69494</td>
</tr>
<tr>
<td>PM38435</td>
<td>Virtual storage</td>
<td>Account for storage cushion in DB2 monitor thresholds and system health and add to DSNV508I message</td>
<td>UK69817 also V10</td>
</tr>
<tr>
<td>PM45810</td>
<td>Online Reorg</td>
<td>Enhancement for LOG phase of REORG SHRLEVEL CHANGE at part level (see also PM37112)</td>
<td>UK71129 (OPEN)</td>
</tr>
</tbody>
</table>

**A.2 Functional enhancements APARs**

In Table A-2 we present a list of APARs providing additional functional enhancements to DB2 9 for z/OS.

This list is not and cannot be exhaustive, check RETAIN and the DB2 Web site.

---

<table>
<thead>
<tr>
<th>APAR #</th>
<th>Area</th>
<th>Text</th>
<th>PTF and notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>II14219</td>
<td>zIIP</td>
<td>zIIP exploitation &quot;support use&quot; information</td>
<td></td>
</tr>
<tr>
<td>II14334</td>
<td>LOBs</td>
<td>Info APAR to link together all the LOB support delivery APARs</td>
<td></td>
</tr>
<tr>
<td>II14401</td>
<td>Migration</td>
<td>Info APAR to link together all the migration APARs</td>
<td></td>
</tr>
<tr>
<td>II14426</td>
<td>XML</td>
<td>Info APAR to link together all the XML support delivery APARs</td>
<td></td>
</tr>
<tr>
<td>II14441</td>
<td>Incorrout PTFs</td>
<td>Recommended DB2 9 SQL INCORROUT PTFs</td>
<td></td>
</tr>
<tr>
<td>II14464</td>
<td>Migration</td>
<td>Info APAR to link together all the other migration APARs</td>
<td></td>
</tr>
<tr>
<td>PK28627</td>
<td>DCLGEN</td>
<td>Additional DCLGEN option DCLBIT to generate declared SQL variables for columns defined as FOR BIT DATA (COBOL, PL/I, C, and C++).</td>
<td>UK37397</td>
</tr>
<tr>
<td>PK43861</td>
<td>DECFLOAT</td>
<td>SPUFI currently doesn't allow you to select data from DECFLOAT type columns.</td>
<td>UK42269</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>PK41001</td>
<td>BACKUP</td>
<td>Incremental FlashCopy support in the BACKUP SYSTEM utility</td>
<td>UK28089 (and z/OS 1.8 APAR OA17314)</td>
</tr>
<tr>
<td>PK44617</td>
<td>Trusted context</td>
<td>Additional functions</td>
<td>UK33449</td>
</tr>
<tr>
<td>PK45599</td>
<td>Data Studio</td>
<td>DB2 for z/OS Administrative Enablement Stored Procedures</td>
<td>UK32060</td>
</tr>
<tr>
<td>PK46562</td>
<td>DB2-supplied</td>
<td>Three new stored procedures for DB2 Administrative Enablement:</td>
<td>UK32061 DB2 V8</td>
</tr>
<tr>
<td></td>
<td>stored procedures</td>
<td>- SYSPROC.GET_CONFIG</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SYSPROC.GET_MESSAGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SYSPROC.GET_SYSTEM_INFO</td>
<td></td>
</tr>
<tr>
<td>PK47126</td>
<td>Data Studio</td>
<td>DB2 console messages recorded through IFI interface</td>
<td>UK30279</td>
</tr>
<tr>
<td>PK47579</td>
<td>ALTER auditing</td>
<td>To audit all ALTER TABLE statements on an audited table in addition</td>
<td>UK31511</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to auditing ALTER TABLE statements with the AUDIT option specified</td>
<td></td>
</tr>
<tr>
<td>PK47893</td>
<td>Data Studio</td>
<td>Data Server Administrative Task Scheduler</td>
<td>UK32047 also in V8</td>
</tr>
<tr>
<td>PK48773</td>
<td>SOA</td>
<td>New DB2 UDFs to consume Web Services via SQL statements using SOAP</td>
<td>UK31857 also in V8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>request over HTTP/HTTPS</td>
<td></td>
</tr>
<tr>
<td>PK50369</td>
<td>Data Studio</td>
<td>Enable Visual Explain</td>
<td>UK31820</td>
</tr>
<tr>
<td>PK51020</td>
<td>Spatial</td>
<td>OGC compliant functions</td>
<td>UK30714</td>
</tr>
<tr>
<td>PK51571</td>
<td>XML</td>
<td>XTABLE, XCAST</td>
<td>UK19587</td>
</tr>
<tr>
<td>PK51572</td>
<td>XML</td>
<td>XTABLE, XCAST</td>
<td>UK30693</td>
</tr>
<tr>
<td>PK51573</td>
<td>XML</td>
<td>XTABLE, XCAST</td>
<td>UK33493</td>
</tr>
<tr>
<td>PK51979</td>
<td>RESTORE</td>
<td>Allow RESTORE SYSTEM recovery without requiring log truncation</td>
<td>UK31997</td>
</tr>
<tr>
<td>PK54451</td>
<td>Spatial</td>
<td>Additional OGC compliant UDFs</td>
<td>UK31092</td>
</tr>
<tr>
<td>PK55585</td>
<td>XML</td>
<td>XPath function - 1</td>
<td>UK33650</td>
</tr>
<tr>
<td>PK55831</td>
<td>XML</td>
<td>XPath function - 2</td>
<td>UK34342</td>
</tr>
<tr>
<td>PK56392</td>
<td>ALTER DROP DEFAULT</td>
<td>Add function for ALTER TABLE DROP DEFAULT</td>
<td>UK42715</td>
</tr>
<tr>
<td>PK58292</td>
<td>EAV</td>
<td>Add support to allow DB2 BSDSs and active logs to reside on an EAV</td>
<td>UK35902 also V8</td>
</tr>
<tr>
<td>PK60612</td>
<td>UNLOAD</td>
<td>Modified to allow UNLOAD from ICOPY of TS which was non-segmented</td>
<td>UK35132</td>
</tr>
<tr>
<td></td>
<td></td>
<td>even though now it is defined as segmented (deprecation of simple TS)</td>
<td></td>
</tr>
<tr>
<td>PK62161</td>
<td>IFC</td>
<td>IFCID 002 / IFCID 003 enhancements to display the number of rows</td>
<td>UK44050 also V8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fetched/updated/deleted/inserted in rowset operations</td>
<td></td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>PK62178</td>
<td>Implicit database</td>
<td>The name generated for implicit databases follows the naming convention of DSNxxxxx, where xxxxx is a number ranging from 00001 to 60000 and is controlled internally by the system sequence object SYSIBM.DSNSEQ.IMPLICITDB. For new users installing or migrating to V9, the default maximum number of databases that can be created implicitly will be lowered from 60000 to 10000.</td>
<td>UK44489</td>
</tr>
<tr>
<td>PK63325</td>
<td>LOAD</td>
<td>COPYDICTIONARY enhancement for INTO TABLE PART</td>
<td>UK37143</td>
</tr>
<tr>
<td>PK64045</td>
<td>Private Protocol</td>
<td>Deprecation</td>
<td>UK54508</td>
</tr>
<tr>
<td>PK66085</td>
<td>SOA</td>
<td>Provide correct SQLSTATE for DB2 Web Service Consumer UDFs: SOAPHTTPNC and SOAPHTTPNV</td>
<td>UK37104</td>
</tr>
<tr>
<td>PK71816</td>
<td>DSN1COMP</td>
<td>Externalize dictionaries (Encryption tool)</td>
<td>UK41355</td>
</tr>
<tr>
<td>PK72444</td>
<td>DB2 provided</td>
<td>Enhance the SYSPROC.GET_SYSTEM_INFO stored procedure to return Sysplex name, catalog attributes, ICF catalog of DB2 subsystem data sets for the DB2 directory and catalog databases, the default database, and the temporary database, ICF catalog of active and archive log data sets, and ICF catalog associated with each storage group in SYSIBM.SYSTOGGROUP.</td>
<td>UK52835</td>
</tr>
<tr>
<td>PK75149</td>
<td>UTS, PBG</td>
<td>Mass Delete Process skips scanning thru all spacemaps and mistakenly determines the table space is empty</td>
<td>UK43199</td>
</tr>
<tr>
<td>PK75214</td>
<td>Reporting</td>
<td>zIIP performance reporting with thread reuse</td>
<td>UK42863 V8</td>
</tr>
<tr>
<td>PK77426</td>
<td>OPTIXOPREF</td>
<td>Default of opaque zparm OPTIXOPREF from OFF to ON to prefer index-only index over index+data</td>
<td>UK51283</td>
</tr>
<tr>
<td>PK78958</td>
<td>Reordered row format</td>
<td>Disable REORG TABLESPACE and LOAD REPLACE from converting a COMPRESS YES table space or partition from Basic Row Format (BRF) to Reordered Row Format (RRF) in DB2 9 NFM.</td>
<td>UK45353</td>
</tr>
<tr>
<td>PK78959</td>
<td>REORG serviceability</td>
<td>Allow conversions for RRFt</td>
<td>UK45881</td>
</tr>
<tr>
<td>PK79228</td>
<td>Group attach</td>
<td>New DB2 subsystem parameter RANDOMATT=YES/NO</td>
<td>UK44898</td>
</tr>
<tr>
<td>PK79327</td>
<td>Group attach</td>
<td>DB2 9 introduced a new feature: randomization of group attachment requests. This APAR allows to set a DB2 member ineligible for randomized group attach</td>
<td>UK44899</td>
</tr>
<tr>
<td>PK80224</td>
<td>IRLM</td>
<td>Large numbers of waitors threads in DB2 consumption in IRLM resulting in delays in threads to timeout.</td>
<td>UK45701</td>
</tr>
<tr>
<td>PK80225</td>
<td>IRLM</td>
<td>Delays in Timeout processing may cause group-wide slowdown</td>
<td>UK46565 V8</td>
</tr>
<tr>
<td>PK80320</td>
<td>data sharing</td>
<td>Automatic GRECP recovery is delayed for Lock timeouts or a deadlock with another member</td>
<td>UK45000</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>PK80925</td>
<td>ACCESS DATABASE</td>
<td>Command to support subset, range, and wildcards</td>
<td>PK52650</td>
</tr>
<tr>
<td>PK81151</td>
<td>Extended address volumes</td>
<td>Add extended address volumes (EAV) support to DB2 utilities</td>
<td>UK47678 also V8</td>
</tr>
<tr>
<td>PK82635</td>
<td>XML</td>
<td>XML decomposition feature is deprecated. The XMLTABLE function should be used to decompose an XML document instead.</td>
<td>UK53631</td>
</tr>
<tr>
<td>PK83072</td>
<td>ODBC</td>
<td>New 64-bit implementation of the DB2 ODBC for z/OS driver</td>
<td>UK50918</td>
</tr>
<tr>
<td>PK83683</td>
<td>REORG part</td>
<td>Excessive logging and elapsed time during SORTBILD phase of LOAD PART REPLACE and REORG TABLESPACE PART</td>
<td>UK50265</td>
</tr>
<tr>
<td>PK83735</td>
<td>LOG</td>
<td>DB2 performs forced log writes after inserting into a newly formatted/allocated page on a GBP Dependent Segmented or Universal Table Space (UTS).</td>
<td>PK51613</td>
</tr>
<tr>
<td>PK84584</td>
<td>RUNSTATS</td>
<td>Sequential detection does not trigger PREFETCH when ENHANCED CLUSTERRATIO is used for 8 to 32k table index spaces</td>
<td>UK47894</td>
</tr>
<tr>
<td>PK85068</td>
<td>EXPLAIN</td>
<td>Explain table migration to Unicode</td>
<td>UK53250 also V8</td>
</tr>
<tr>
<td>PK85856</td>
<td>zIIP</td>
<td>For DB2 Sort in utilities</td>
<td>UK48912 also V8</td>
</tr>
<tr>
<td>PK85881</td>
<td>Reordered row format</td>
<td>Enable new DSNZPARM SPRMRRF and ROWFORMAT option for LOAD and REORG TABLESPACE</td>
<td>UK50413</td>
</tr>
<tr>
<td>PK87348</td>
<td>Reordered row format</td>
<td>Enable basic row format for universal table spaces</td>
<td>UK50412</td>
</tr>
<tr>
<td>PK87762</td>
<td>REORG</td>
<td>Reorg partition ranges</td>
<td>UK59095</td>
</tr>
<tr>
<td>PK90089</td>
<td>CATMAINT</td>
<td>CATMAINT UPDATE SCHEMA SWITCH job takes an unexpected long time to run</td>
<td>UK49364</td>
</tr>
<tr>
<td>PK90346</td>
<td>Common criteria</td>
<td>The administrator can restrict the maximum number of concurrent sessions for a primary authorization ID</td>
<td>UK56520</td>
</tr>
<tr>
<td>PK91610</td>
<td>DSNHPC7</td>
<td>V7 precompiler like support</td>
<td>UK51891</td>
</tr>
<tr>
<td>PK92339</td>
<td>Private Protocol switch</td>
<td>New PRIVATE_PROTOCOL subsystem parameter to force error in new binding DSN6FAC PRIVATE_PROTOCOL YES/NO</td>
<td>UK51679 also V8</td>
</tr>
<tr>
<td>APAR #</td>
<td>Area</td>
<td>Text</td>
<td>PTF and notes</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>PK93084</td>
<td>Install changes</td>
<td>Corrected V9 installation CLIST to remove extraneous lines when DSNTIJMV is edited for installing data sharing. In the V8 and V9 DSNTIJSG job, the CREATE statements for the SYSPROC.DSNUTILS and SYSPROC.DSNUTILU stored procedures are updated to specify a default WLM environment named WLMUTL1. Install CLIST member DSNTINS1 is adjusted to account for this change in the editing of DSNTIJSG. In the V8 and V9 DSNTIJSG job, the CREATE statement for the stored procedure SYSPROC.WLM_SET_CLIENT_INFO is updated to specify SECURITY DB2.</td>
<td>UK51595 also V8</td>
</tr>
<tr>
<td>PK99119</td>
<td>Encryption</td>
<td>Exploit the z10 CEX3C encryption hardware in IMS and DB2 encryption tools. Integrated Cryptographic Services Facility (ICSF) version HCR7770 supports this CPACF Protected Key (CPK) feature.</td>
<td>UK52176 for Tools</td>
</tr>
<tr>
<td>PM01177</td>
<td>UTS, PBG</td>
<td>SQLCODE151 -151 is issued at DB2 commit due to a bad reason RC00C9007C returned at commit phase one process</td>
<td>UK53551</td>
</tr>
<tr>
<td>PM03691</td>
<td>DSN1COPY</td>
<td>Help prevent abends, data corruption and storage overlays when DSN1COPY is used incorrectly</td>
<td>Closed, future release</td>
</tr>
<tr>
<td>PM06852</td>
<td>SOA</td>
<td>DB2 Web service consumer functions to invoke a Web service which requires the Simple Object Access Protocol (SOAP) version 1.2</td>
<td>UK55251</td>
</tr>
<tr>
<td>PM07944</td>
<td>EDITPROC</td>
<td>Remove editproc restrictions. If the procedure is sensitive to the format in which DB2 stores the rows of the table, the edit procedure is specified using the default EDITPROC WITH ROW ATTRIBUTES clause of the CREATE TABLE statement.</td>
<td>OPEN</td>
</tr>
<tr>
<td>PM12819</td>
<td>Utilities</td>
<td>Enable use of DB2 Sort for z/OS for DB2 utilities sort processing</td>
<td>UK59101</td>
</tr>
<tr>
<td>PM13259</td>
<td>REORG</td>
<td>Reorg disjoint partition ranges</td>
<td>UK59096</td>
</tr>
<tr>
<td>PM17665</td>
<td>DDF</td>
<td>Changes to authorization checking at the server when Private Protocol is disabled</td>
<td>UK65971 also V8 and V10</td>
</tr>
<tr>
<td>PM27962</td>
<td>Load Defer Index Build</td>
<td>A new LOAD option, INDEXDEFER NPI/ALL, indicates to LOAD that the specified index types will be deferred</td>
<td>Packaging also V10</td>
</tr>
<tr>
<td>PM37300</td>
<td>DDF</td>
<td>Change authorization check at server to recognize secondary IDs from DB2 for z/OS requesters when Private Protocol is disabled.</td>
<td>UK67641 also V8 and V10</td>
</tr>
<tr>
<td>PM38417</td>
<td></td>
<td>End part of PM37300</td>
<td>OPEN</td>
</tr>
<tr>
<td>PM42560</td>
<td>LOAD</td>
<td>Enable new LOAD INDEXDEFER option (see also PM27962)</td>
<td>CLOSED also V10</td>
</tr>
</tbody>
</table>
## A.3 z/OS APARs

In Table A-3 we present a list of APARs providing additional enhancements for z/OS.

This list is not and cannot be exhaustive, check RETAIN and the DB2 Web site.

### Table A-3  z/OS DB2-related APARs

<table>
<thead>
<tr>
<th>APAR #</th>
<th>Area</th>
<th>Text</th>
<th>PTF and notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OA03148</td>
<td>RRS exit</td>
<td>RRS support</td>
<td>UA07148</td>
</tr>
<tr>
<td>OA09782</td>
<td>Unicode</td>
<td>Identify substitution character</td>
<td>UA26564</td>
</tr>
<tr>
<td>OA17735</td>
<td>SRM function for blocked workload</td>
<td><a href="http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10609">http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/FLASH10609</a></td>
<td>UA36199 (provided in z/OS 1.9, and rolled back)</td>
</tr>
<tr>
<td>OA18461</td>
<td>WLM</td>
<td>DB2 buffer pool management assist</td>
<td>UA48912</td>
</tr>
<tr>
<td>OA19072</td>
<td>Unicode</td>
<td>Character conversion problem</td>
<td>UA32755</td>
</tr>
<tr>
<td>OA22443</td>
<td>WLM blocked workload</td>
<td>Enabled by default in all current versions of z/OS</td>
<td>UA40609</td>
</tr>
<tr>
<td>OA23828</td>
<td>XML</td>
<td>z/OS 1.10 XML System Services includes additional zIIP exploitation enabling all z/OS XML parsing in enclave SRB mode to be eligible for zIIP. Retrofitted on z/OS V1.8 and V1.9.</td>
<td>UA41773</td>
</tr>
<tr>
<td>OA23849</td>
<td>Backup/Restore</td>
<td>Support for PPRC volumes in the FRBACKUP or FRRECOV functions</td>
<td>UA42372</td>
</tr>
<tr>
<td>OA36101</td>
<td>Dynamic allocation</td>
<td>Overlay of high private storage with a UCB address</td>
<td>UA60221</td>
</tr>
<tr>
<td>OA26104</td>
<td>zIIP for parallel queries DB2 APARs PK76676/UK47686 (V8) PK87913/UK49700 (V9)</td>
<td>New enclave WORKDEPENDENT which can be created by the DBM1 address space running under the callers task which runs under the original enclave of the request. It is an extension to the original enclave so no additional classification by the customer is required and no additional classification under subsystem type DB2.</td>
<td>UA47647</td>
</tr>
<tr>
<td>OA33106</td>
<td>ESQA storage</td>
<td>Decrease ESQA storage usage for an SRB that is using a linkage stack when its status is being saved</td>
<td>UA56174</td>
</tr>
</tbody>
</table>
Statistics report

In this appendix, we present a sample OMEGAMON XE Performance Expert batch statistics report and a sample OMEGAMON XE Performance Expert batch accounting report. These samples are provided as a reference to allow you to see the changed reports and so you can verify the output fields that are referenced from chapters in the book.

Note: For DB2 9 for z/OS support, you need OMEGAMON PE V4.1 with both of the following APARs:

- APAR PK36297 - PTF UK23929/UK23930 base support
- APAR PK40691 - PTF UK23984, which contains the PE Client
B.1 OMEGAMON XE Performance Expert statistics report long layout

The OMEGAMON XE statistics report was produced by processing the SMF data from a DB2 V9 subsystem. The data in this statistics report is referenced in 4.5, “Virtual storage constraint relief” on page 94.

Example B-1 shows the processing parameters that are used.

Example: B-1  Statement for the statistics report

GLOBAL
STATISTICS TRACE DDNAME(SREPORT)
LAYOUT(LONG)
EXEC

Example B-2 shows the output of the report.

Example: B-2  Sample of the statistics report long layout

1 LOCATION: DSND91B OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4) PAGE: 1-1
GROUP: N/P
MEMBER: N/P
SUBSYSTEM: D91B
DB2 VERSION: V9
SCOPE: MEMBER
INTERVAL FROM: 02/28/07 12:59:34.30
INTERVAL TO: 02/28/07 13:00:34.29

---- HIGHLIGHTS ---------------------------------------------------------------------
INTERVAL START: 02/28/07 12:59:34.30    SAMPLING START: 02/28/07 12:59:34.30    TOTAL THREADS: 0.00
INTERVAL END: 02/28/07 13:00:34.29    SAMPLING END: 02/28/07 13:00:34.29    TOTAL COMMITS: 0.00
INTERVAL ELAPSED: 59.994369    OUTAGE ELAPSED: 0.00000077777778    DATA SHARING MEMBER: N/A

SQL DML                          QUANTITY /SECOND /THREAD /COMMIT  SQL DCL                          QUANTITY /SECOND /THREAD /COMMIT
---------------------------------  --------  -------  -------  -------  ---------------------------------  --------  -------  -------  -------
SELECT                             0.00     0.00      N/C      N/C    LOCK TABLE                          0.00     0.00      N/C      N/C
INSERT                             0.00     0.00      N/C      N/C    GRANT                                0.00     0.00      N/C      N/C
UPDATE                             0.00     0.00      N/C      N/C    REVOKE                               0.00     0.00      N/C      N/C
MERGE                              0.00     0.00      N/C      N/C    SET HOST VARIABLE                    0.00     0.00      N/C      N/C
DELETE                             0.00     0.00      N/C      N/C    SET CURRENT SQLID                    0.00     0.00      N/C      N/C
PREPARE                            0.00     0.00      N/C      N/C    SET CURRENT DEGREE                  0.00     0.00      N/C      N/C
DESCRIBE                           0.00     0.00      N/C      N/C    SET CURRENT RULES                   0.00     0.00      N/C      N/C
DESCRIBE TABLE                     0.00     0.00      N/C      N/C    SET CURRENT PATH                    0.00     0.00      N/C      N/C
OPEN                               0.00     0.00      N/C      N/C    SET CURRENT PRECISION               0.00     0.00      N/C      N/C
CLOSE                              0.00     0.00      N/C      N/C    SET CONNECTION                       0.00     0.00      N/C      N/C
FETCH                              0.00     0.00      N/C      N/C    ASSOCIATE LOCATORS                  0.00     0.00      N/C      N/C
RELEASE                            0.00     0.00      N/C      N/C    ALLOCATE CURSOR                      0.00     0.00      N/C      N/C
TOTAL                              0.00     0.00      N/C      N/C    HOLD LOCATOR                         0.00     0.00      N/C      N/C
                                    0.00     0.00      N/C      N/C    FREE LOCATOR                         0.00     0.00      N/C      N/C
                                    0.00     0.00      N/C      N/C    TOTAL                               0.00     0.00      N/C      N/C
STORED PROCEDURES                 QUANTITY /SECOND /THREAD /COMMIT  TRIGGERS                         QUANTITY /SECOND /THREAD /COMMIT
---------------------------------  --------  -------  -------  -------  ---------------------------------  --------  -------  -------  -------
CALL STATEMENT EXECUTED            0.00     0.00      N/C      N/C    STATEMENT TRIGGER ACTIVATED         0.00     0.00      N/C      N/C
PROCEDURE ABENDED                  0.00     0.00      N/C      N/C    ROW TRIGGER ACTIVATED               0.00     0.00      N/C      N/C
CALL STATEMENT TIMED OUT           0.00     0.00      N/C      N/C    SQL ERROR OCCURRED                   0.00     0.00      N/C      N/C
CALL STATEMENT REJECTED            0.00     0.00      N/C      N/C
USER DEFINED FUNCTIONS            QUANTITY /SECOND /THREAD /COMMIT  ROW ID                         QUANTITY /SECOND /THREAD /COMMIT
---------------------------------  --------  -------  -------  -------  ---------------------------------  --------  -------  -------  -------
EXECUTED                          0.00     0.00      N/C      N/C    DIRECT ACCESS                       0.00     0.00      N/C      N/C
ABENDED                           0.00     0.00      N/C      N/C    INDEX USED                          0.00     0.00      N/C      N/C
TIMED OUT                         0.00     0.00      N/C      N/C    TABLE SPACE SCAN USED              0.00     0.00      N/C      N/C
REJECTED                          0.00     0.00      N/C      N/C
### Appendix B. Statistics report

**GROUP:** N/P  **MEMBER:** N/P  **SUBSYSTEM:** D91B  **DB VERSION:** V9  
**STATISTICS REPORT - LONG**  **INTERVAL FROM:** 02/28/07 12:59:34.30  **REQUESTED FROM:** NOT SPECIFIED

---

**INTERVAL START:** 02/28/07 12:59:34.30  **SAMPLING START:** 02/28/07 12:59:34.30  **TOTAL THREADS:** 0.00  
**INTERVAL END:** 02/28/07 13:00:34.29  **SAMPLING END:** 02/28/07 13:00:34.29  **TOTAL COMMITS:** 0.00  
**INTERVAL ELAPSED:** 59.994369  **OUTAGE ELAPSED:** 0.000000  **DATA SHARING MEMBER:** N/A

---

**SQL DDL**  **QUANTITY**  **/SECOND**  **/THREAD**  **/COMMIT**  **SQL DDL CONTINUED**  **QUANTITY**  **/SECOND**  **/THREAD**  **/COMMIT**

---

CREATE TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE GLOBAL TEMP TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
DECLARE GLOBAL TEMP TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE AUXILIARY TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE INDEX 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE VIEW 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE SYNONYM 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE TABLESPACE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE DATABASE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE STOGROUP 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE ALIAS 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE DISTINCT TYPE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE FUNCTION 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE PROCEDURE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE TRIGGER 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE SEQUENCE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
CREATE TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER INDEX 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER VIEW 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER TABLESPACE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER DATABASE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER STOGROUP 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER FUNCTION 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER PROCEDURE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER SEQUENCE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER TRUSTED CONTEXT 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER VIEW 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER TABLESPACE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER DATABASE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER STOGROUP 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER FUNCTION 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER PROCEDURE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER SEQUENCE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER TABLE 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C
ALTER TRUSTED CONTEXT 0.00 0.00 N/C N/C 0.00 0.00 N/C N/C

---

**EOM POOL**  **QUANTITY**  **/SECOND**  **/THREAD**  **/COMMIT**

---

PAGES IN RDS POOL (BELOW) 37500.00 N/A N/A N/A
HELD BY CT 0.00 N/A N/A N/A
HELD BY PT 4602.00 N/A N/A N/A
FREE PAGES 32896.00 N/A N/A N/A
FAILS DUE TO POOL FULL 0.00 0.00 N/C N/C
PAGES IN RDS POOL (ABOVE) 524.3K N/A N/A N/A
HELD BY CT 0.00 N/A N/A N/A
HELD BY PT 3504.00 N/A N/A N/A
FREE PAGES 520.8K N/A N/A N/A
FAILS DUE TO RDS POOL FULL 0.00 0.00 N/C N/C
PAGES IN DBD POOL (ABOVE) 262.1K N/A N/A N/A
HELD BY DBD 67.00 N/A N/A N/A
FREE PAGES 262.1K N/A N/A N/A
FAILS DUE TO DBD POOL FULL 0.00 0.00 N/C N/C
PAGES IN STMT POOL (ABOVE) 262.1K N/A N/A N/A
HELD BY STATEMENTS 5.00 N/A N/A N/A
FREE PAGES 262.1K N/A N/A N/A
FAILS DUE TO STMT POOL FULL 0.00 0.00 N/C N/C
PAGES IN SKEL POOL (ABOVE) 25600.00 N/A N/A N/A
HELD BY SKEL 2.00 N/A N/A N/A
HELD BY SKPT 322.00 N/A N/A N/A
FREE PAGES 25276.00 N/A N/A N/A
FAILS DUE TO SKEL POOL FULL 0.00 0.00 N/C N/C

---

**EDM POOL**  **QUANTITY**  **/SECOND**  **/THREAD**  **/COMMIT**

---

PAGES IN RDS POOL (BELOW) 37500.00 N/A N/A N/A
HELD BY CT 0.00 N/A N/A N/A
HELD BY PT 4602.00 N/A N/A N/A
FREE PAGES 32896.00 N/A N/A N/A
FAILS DUE TO POOL FULL 0.00 0.00 N/C N/C
PAGES IN RDS POOL (ABOVE) 524.3K N/A N/A N/A
HELD BY CT 0.00 N/A N/A N/A
HELD BY PT 3504.00 N/A N/A N/A
FREE PAGES 520.8K N/A N/A N/A
FAILS DUE TO RDS POOL FULL 0.00 0.00 N/C N/C
PAGES IN DBD POOL (ABOVE) 262.1K N/A N/A N/A
HELD BY DBD 67.00 N/A N/A N/A
FREE PAGES 262.1K N/A N/A N/A
FAILS DUE TO DBD POOL FULL 0.00 0.00 N/C N/C
PAGES IN STMT POOL (ABOVE) 262.1K N/A N/A N/A
HELD BY STATEMENTS 5.00 N/A N/A N/A
FREE PAGES 262.1K N/A N/A N/A
FAILS DUE TO STMT POOL FULL 0.00 0.00 N/C N/C
PAGES IN SKEL POOL (ABOVE) 25600.00 N/A N/A N/A
HELD BY SKEL 2.00 N/A N/A N/A
HELD BY SKPT 322.00 N/A N/A N/A
FREE PAGES 25276.00 N/A N/A N/A
FAILS DUE TO SKEL POOL FULL 0.00 0.00 N/C N/C

---
### Dynamic SQL Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare Requests</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Full Prepare</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Implicit Prepare</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Cache Limit Exceeded</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Prep Stmt Purged</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Local Cache Hit Ratio (%)</td>
<td>N/C</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Open/Close Activity

<table>
<thead>
<tr>
<th>Dataset Type</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Datasets - HMM</td>
<td>134.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DSET Converted R/W -&gt; R/O</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>

### Log Activity

<table>
<thead>
<tr>
<th>Log Activity</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Records Created</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Log CI Created</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Log Write I/O (Log1&amp;2)</td>
<td>8.00</td>
<td>0.13</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Log CI Written (Log1&amp;2)</td>
<td>8.00</td>
<td>0.13</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Log Rate for 1 Log (MB)</td>
<td>N/A</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Log Write Suspended</td>
<td>3.00</td>
<td>0.05</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>
### PLAN/PACKAGE PROCESSING

<table>
<thead>
<tr>
<th></th>
<th>QUANTITY</th>
<th>/SECOND</th>
<th>/THREAD</th>
<th>/COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCREMENTAL BINDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PLAN ALLOCATION ATTEMPTS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PLAN ALLOCATION SUCCESS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PACKAGE ALLOCATION ATTEMPT</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PACKAGE ALLOCATION SUCCESS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PLANS BOUND</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>BIND ADD SUBCOMMANDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>BIND REPLACE SUBCOMMANDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>TEST BINDS NO PLAN-ID</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PACKAGES BOUND</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>BIND ADD PACKAGE SUBCOMMAND</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>BIND REPLACE PACKAGE SUBCOMMAND</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>AUTOMATIC BIND ATTEMPTS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>AUTOMATIC BINDS SUCCESS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>AUTO.BIND INVALID RES. IDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>AUTO.BIND PACKAGE ATTEMPTS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>AUTO.BIND PACKAGES SUCCESS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>REBIND SUBCOMMANDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>ATTEMPTS TO REBIND A PLAN</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PLANS REBOUND</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>REBIND PACKAGE SUBCOMMANDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>ATTEMPTS TO REBIND PACKAGE</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PACKAGES REBOUND</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>FREE PLAN SUBCOMMANDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>ATTEMPTS TO FREE A PLAN</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PLANS FREED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>FREE PACKAGE SUBCOMMANDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>ATTEMPTS TO FREE A PACKAGE</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>PACKAGES FREED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>

--- HIGHLIGHTS ---

1. **START DATABASE**
2. **ALTER DATABASE**
3. **DISPLAY DATABASE**
4. **DISPLAY THREAD**
5. **DISPLAY UTILITY**
6. **DISPLAY TRACE**
7. **DISPLAY RLIMIT**
8. **DISPLAY LOCATION**
9. **DISPLAY ARCHIVE**
10. **DISPLAY BUFFERPOOL**
11. **DISPLAY GROUPBUFFERPOOL**
12. **DISPLAY GROUP**
13. **DISPLAY PROCEDURE**
14. **DISPLAY FUNCTION**
15. **DISPLAY LOG**
16. **DISPLAY DDF**
17. **DISPLAY PROFILE**
18. **ALTER BUFFERPOOL**
19. **ALTER GROUPBUFFERPOOL**
20. **ALTER UTLITY**

--- HIGHLIGHTS ---

### DB2 COMMANDS

<table>
<thead>
<tr>
<th></th>
<th>QUANTITY</th>
<th>/SECOND</th>
<th>DB2 COMMANDS CONTINUED</th>
<th>QUANTITY</th>
<th>/SECOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY DATABASE</td>
<td>0.00</td>
<td>0.00</td>
<td>MODIFY TRACE</td>
<td>1.00</td>
<td>0.02</td>
</tr>
<tr>
<td>DISPLAY THREAD</td>
<td>0.00</td>
<td>0.00</td>
<td>CANCEL THREAD</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY UTILITY</td>
<td>0.00</td>
<td>0.00</td>
<td>TERM UTILITY</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY TRACE</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY RLIMIT</td>
<td>0.00</td>
<td>0.00</td>
<td>RECOVER BSDS</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY LOCATION</td>
<td>0.00</td>
<td>0.00</td>
<td>RECOVER INDOUBT</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY ARCHIVE</td>
<td>0.00</td>
<td>0.00</td>
<td>RESET INDOUBT</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY BUFFERPOOL</td>
<td>0.00</td>
<td>0.00</td>
<td>RESET GENERICLU</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY GROUPBUFFERPOOL</td>
<td>0.00</td>
<td>0.00</td>
<td>ARCHIVE LOG</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY GROUP</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY PROCEDURE</td>
<td>0.00</td>
<td>0.00</td>
<td>SET ARCHIVE</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY FUNCTION</td>
<td>0.00</td>
<td>0.00</td>
<td>SET LOG</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY LOG</td>
<td>0.00</td>
<td>0.00</td>
<td>SET SYSPARM</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>DISPLAY DDF</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY PROFILE</td>
<td>0.00</td>
<td>0.00</td>
<td>ACCESS DATABASE</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ALTER BUFFERPOOL</td>
<td>0.00</td>
<td>0.00</td>
<td>UNRECOGNIZED COMMANDS</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>ALTER GROUPBUFFERPOOL</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTER UTILITY</td>
<td>0.00</td>
<td>0.00</td>
<td>TOTAL</td>
<td>3.00</td>
<td>0.05</td>
</tr>
<tr>
<td>START DATABASE</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START TRACE</td>
<td>2.00</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START DB2</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START RLIMIT</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START DDF</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START PROCEDURE</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START FUNCTION</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>START PROFILE</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### LOCKING ACTIVITY

<table>
<thead>
<tr>
<th>Activity</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suspensions (ALL)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Suspensions (LOCK ONLY)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Suspensions (OTHER)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Timeouts</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Deadlocks</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Lock requests</td>
<td>12.00</td>
<td>0.20</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Unlock requests</td>
<td>38.00</td>
<td>0.63</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Query requests</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Change requests</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Other requests</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Lock escalation (shared)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Lock escalation (exclusive)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Drain requests</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Drain requests failed</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Claim requests</td>
<td>7.00</td>
<td>0.12</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Claim requests failed</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>

### DATA SHARING LOCKING

<table>
<thead>
<tr>
<th>Activity</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global contention rate (N/C)</td>
<td>3.00</td>
<td>0.05</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>P/L-LOCKS XES rate (N/C)</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNLOCK requests (P-LOCKS)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>Change requests (P-LOCKS)</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>
### Highlights

**Interval Start:** 02/28/07 12:59:34.30  |  **Sampling Start:** 02/28/07 12:59:34.30  |  **Total Threads:** 0.00

**Interval End:** 02/28/07 13:00:34.29  |  **Sampling End:** 02/28/07 13:00:34.29  |  **Total Commits:** 0.00

**Interval Elapsed:** 59.994369  |  **Outage Elapsed:** 0.000000  |  **Data Sharing Member:** N/A

### Global DDF Activity

<table>
<thead>
<tr>
<th>Segment</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
<th>Query Parallelism</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBAT QUEUED-MAXIMUM ACTIVE</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/A</td>
<td>MAX.DEGREE OF PARALLELISM</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>CONV.DEALOC-MAX.CONNECTED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/A</td>
<td>PARALLEL GROUPS EXECUTED</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>COLD START CONNECTIONS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>RAN AS PLANNED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>WARM START CONNECTIONS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>RAN REDUCED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>RESYNCHRONIZATION ATTEMPTED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>SEQUENTIAL-CURSOR</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>RESYNCHRONIZATION SUCCEEDED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>SEQUENTIAL-NO ESA</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>CUR TYPE 1 INACTIVE DBATS</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>SEQUENTIAL-NO BUFFER</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>TYPE 1 INACTIVE DBATS HW</td>
<td>1.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>SEQUENTIAL-ENCLAVE SER.</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>CUR TYPE 2 INACTIVE DBATS</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>ONE DB2 - COORDINATOR + NO</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>TYPE 2 INACTIVE DBATS HW</td>
<td>500.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>ONE DB2 - DCL TABLE</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>ACC QUEUED TYPE 2 INACT THR</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>MEMBER SKIPPED (%)</td>
<td>N/C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUR QUEUED TYPE 2 INACT THR</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>REFORM PARAL-CONFIG CHANGED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>QUERIED TYPE 2 INACT THR HW</td>
<td>66.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>REFORM PARAL-NO BUFFER</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>CURRENT INACTIVE DBATS</td>
<td>500.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL INACTIVE DBATS</td>
<td>508.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CURRENT DBATS NOT IN USE</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBATS NOT IN USE HW</td>
<td>17.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DBATS CREATED</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POOL DBATS REUSED</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CPU Times

<table>
<thead>
<tr>
<th>Segment</th>
<th>CPU Time</th>
<th>TCB Time</th>
<th>Preempt SRB</th>
<th>Nonpreempt SRB</th>
<th>Total Time</th>
<th>Preempt IOP SRB</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM SERVICES ADDRESS SPACE</td>
<td>0.013922</td>
<td>0.000000</td>
<td>0.000693</td>
<td>0.014615</td>
<td>N/A</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>DATABASE SERVICES ADDRESS SPACE</td>
<td>0.000135</td>
<td>0.000000</td>
<td>0.009658</td>
<td>0.010792</td>
<td>0.000000</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>ILRM</td>
<td>0.000001</td>
<td>0.000000</td>
<td>0.009455</td>
<td>0.009455</td>
<td>0.000000</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>DDF ADDRESS SPACE</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>0.014058</td>
<td>0.000000</td>
<td>0.029806</td>
<td>0.043864</td>
<td>0.000000</td>
<td>N/C</td>
<td></td>
</tr>
</tbody>
</table>

### DB2 Appliance Interface

<table>
<thead>
<tr>
<th>Segment</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
<th>Data Capture</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABENDS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>LOG RECORDS Captured</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>UNRECOGNIZED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>LOG READS PERFORMED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>COMMAND REQUESTS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>LOG RECORDS RETURNED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>READA REQUESTS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>DATA ROWS RETURNED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>READS REQUESTS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>DESCRIBES PERFORMED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>WRITE REQUESTS</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>TABLES RETURNED</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
<tr>
<td>TOTAL</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### IFC Dest.

<table>
<thead>
<tr>
<th>Segment</th>
<th>Written</th>
<th>Not WRTN</th>
<th>Buf.Over</th>
<th>Not ACCP</th>
<th>WRIT.FAIL</th>
<th>IFC Record Counts</th>
<th>Written</th>
<th>Not WRTN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMF</td>
<td>39.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>SYSTEM RELATED</td>
<td>2.00</td>
<td>0.00</td>
</tr>
<tr>
<td>GTF</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>DATABASE RELATED</td>
<td>2.00</td>
<td>0.00</td>
</tr>
<tr>
<td>OP1</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>ACCOUNTING</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>OP2</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>START TRACE</td>
<td>3.00</td>
<td>0.00</td>
</tr>
<tr>
<td>OP3</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>STOP TRACE</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>OP4</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>SYSTEM PARAMETERS</td>
<td>2.00</td>
<td>0.00</td>
</tr>
<tr>
<td>OP5</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>SYSPARMS-BPPOOLS</td>
<td>2.00</td>
<td>0.00</td>
</tr>
<tr>
<td>OP6</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>AUDIT</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>OP7</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP8</td>
<td>0.00</td>
<td>0.00</td>
<td>N/A</td>
<td>0.00</td>
<td>0.00</td>
<td>TOTAL</td>
<td>12.00</td>
<td>0.00</td>
</tr>
<tr>
<td>RES</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>39.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Accounting Rollup

<table>
<thead>
<tr>
<th>Segment</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
<th>Latch Cnt</th>
<th>/Second</th>
<th>/Second</th>
<th>/Second</th>
<th>/Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROLLUP THRESH REC WRITTEN</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td>LC01-LC04</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
DB2 9 for z/OS Performance Topics

--- MISCELLANEOUS -----------------------------------------------------------------------------

BYPASS COL: 0.00  MAX SQL CASCAD LEVEL: 0.00  MAX STOR LOB VALUES: 0.00

1 LOCATION: DSN918  OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4)  PAGE: 1-11
GROUP: N/P  STATISTICS REPORT - LONG  REQUESTED FROM: NOT SPECIFIED
MEMBER: N/P  TO: NOT SPECIFIED
SUBSYSTEM: D91B  INTERVAL FROM: 02/28/07 12:59:34.30
GROUP: N/P  INTERVAL TO: NOT SPECIFIED
REQUESTED FROM: NOT SPECIFIED
SUBSYSTEM: D91B  INTERVAL FROM: 02/28/07 12:59:34.30
GROUP: N/P  INTERVAL TO: NOT SPECIFIED
REQUESTED FROM: NOT SPECIFIED

--- HIGHLIGHTS -----------------------------------------------------------------------------

INTERVAL START : 02/28/07 12:59:34.30  SAMPLING START: 02/28/07 12:59:34.30  TOTAL THREADS : 0.00
INTERVAL END : 02/28/07 13:00:34.29  SAMPLING END : 02/28/07 13:00:34.29  TOTAL комМITS : 0.00
INTERVAL ELAPSED: 59.993469  OUTAGE ELAPSED: 0.000000  DATA SHARING MEMBER: N/A

DBM1 AND MVS STORAGE BELOW 2 GB  CONTINUED

<table>
<thead>
<tr>
<th>DBM1 AND MVS STORAGE BELOW 2 GB</th>
<th>QUANTITY</th>
<th>DBM1 AND MVS STORAGE BELOW 2 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL DBM1 STORAGE BELOW 2 GB</td>
<td>(MB) 312.24</td>
<td>24 BIT LOW PRIVATE (MB) 0.22</td>
</tr>
<tr>
<td>TOTAL GETMAINED STORAGE</td>
<td>(MB) 147.83</td>
<td>24 BIT HIGH PRIVATE (MB) 0.45</td>
</tr>
<tr>
<td>VIRTUAL BUFFER POOLS</td>
<td>N/A</td>
<td>31 BIT EXTENDED LOW PRIVATE (MB) 48.38</td>
</tr>
<tr>
<td>VIRTUAL POOL CONTROL BLOCKS</td>
<td>N/A</td>
<td>31 BIT EXTENDED HIGH PRIVATE (MB) 329.71</td>
</tr>
<tr>
<td>EDM POOL</td>
<td>N/A</td>
<td>EXTENDED REGION SIZE (MAX) (MB) 1682.00</td>
</tr>
<tr>
<td>COMPRESSION DICTIONARY</td>
<td>N/A</td>
<td>EXTENDED CSA SIZE (MB) 256.35</td>
</tr>
<tr>
<td>CASTOUT BUFFERS</td>
<td>N/A</td>
<td>AVERAGE THREAD FOOTPRINT (MB) 0.17</td>
</tr>
<tr>
<td>DATA SPACE LOCKASIDE BUFFER</td>
<td>N/A</td>
<td>MAX NUMBER OF POSSIBLE THREADS 7431.87</td>
</tr>
<tr>
<td>HIPERPOOL CONTROL BLOCKS</td>
<td>N/A</td>
<td>TOTAL VARIABLE STORAGE (MB) 90.76</td>
</tr>
<tr>
<td>DATA SPACE BP CONTROL BLOCKS</td>
<td>N/A</td>
<td>TOTAL AGENT LOCAL STORAGE (MB) 86.37</td>
</tr>
<tr>
<td>TOTAL AGENT SYSTEM STORAGE</td>
<td>(MB) 3.07</td>
<td>TOTAL AGENT STORAGE (MB) 83.30</td>
</tr>
<tr>
<td>NUMBER OF PREFETCH ENGINES</td>
<td>7.00</td>
<td>TOTAL NUMBER OF ACTIVE USER THREADS 500.00</td>
</tr>
<tr>
<td>NUMBER OF DEFERRED WRITE ENGINES</td>
<td>25.00</td>
<td>N/A</td>
</tr>
<tr>
<td>NUMBER OF CASTOUT ENGINES</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>NUMBER OF GBP WRITE ENGINES</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>NUMBER OF P-LOCK/NOTIFY EXIT ENGINES</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>TOTAL AGENT NON-SYSTEM STORAGE</td>
<td>(MB) 83.30</td>
<td>N/A</td>
</tr>
<tr>
<td>TOTAL NUMBER OF FREE USER THREADS</td>
<td>500.00</td>
<td>N/A</td>
</tr>
<tr>
<td>PIPE MANAGER SUB POOL</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>LOCAL DYNAMIC STMT CNTL BLKS</td>
<td>0.99</td>
<td>N/A</td>
</tr>
<tr>
<td>THREAD COPIES OF CACHED SQL STMTS</td>
<td>0.05</td>
<td>N/A</td>
</tr>
<tr>
<td>IN USE STORAGE</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>STATEMENTS CNT</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>STATEMENT CNT AT HWM</td>
<td>0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>DATE AT HWM</td>
<td>02/28/07</td>
<td>N/A</td>
</tr>
<tr>
<td>TIME AT HWM</td>
<td>20:59:48.69</td>
<td>N/A</td>
</tr>
<tr>
<td>BUFFER &amp; DATA MANAGER TRACE TBL</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>TOTAL FIXED STORAGE</td>
<td>0.57</td>
<td>N/A</td>
</tr>
<tr>
<td>TOTAL GETMAINED STACK STORAGE</td>
<td>73.07</td>
<td>N/A</td>
</tr>
<tr>
<td>TOTAL STACK STORAGE IN USE</td>
<td>(MB) 71.31</td>
<td>N/A</td>
</tr>
<tr>
<td>STORAGE CUSHION</td>
<td>(MB) 337.27</td>
<td>N/A</td>
</tr>
</tbody>
</table>

--- HIGHLIGHTS -----------------------------------------------------------------------------

INTERVAL START : 02/28/07 12:59:34.30  SAMPLING START: 02/28/07 12:59:34.30  TOTAL THREADS : 0.00
INTERVAL END : 02/28/07 13:00:34.29  SAMPLING END : 02/28/07 13:00:34.29  TOTAL комМITS : 0.00
INTERVAL ELAPSED: 59.993469  OUTAGE ELAPSED: 0.000000  DATA SHARING MEMBER: N/A

DBM STORAGE ABOVE 2 GB  CONTINUED

<table>
<thead>
<tr>
<th>DBM STORAGE ABOVE 2 GB</th>
<th>QUANTITY</th>
<th>REAL AND AUXILIARY STORAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL STORAGE</td>
<td>(MB) 4.46</td>
<td>REAL STORAGE IN USE (MB) 855.50</td>
</tr>
<tr>
<td>FIXED STORAGE</td>
<td>(MB) 4898.21</td>
<td>AUXILIARY STORAGE IN USE (MB) 0.00</td>
</tr>
<tr>
<td>GETMAINED STORAGE</td>
<td>(MB) 0.00</td>
<td>N/A</td>
</tr>
<tr>
<td>COMPRESSION DICTIONARY</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>IN USE EDM DBD POOL</td>
<td>(MB) 0.26</td>
<td>N/A</td>
</tr>
<tr>
<td>IN USE EDM STATEMENT POOL</td>
<td>0.02</td>
<td>N/A</td>
</tr>
<tr>
<td>IN USE EDM RDS POOL</td>
<td>(MB) 12.69</td>
<td>N/A</td>
</tr>
<tr>
<td>IN USE EDM EULER POOL</td>
<td>(MB) 1.27</td>
<td>N/A</td>
</tr>
<tr>
<td>VIRTUAL BUFFER POOLS</td>
<td>(MB) 421.87</td>
<td>N/A</td>
</tr>
<tr>
<td>VIRTUAL POOL CONTROL BLOCKS</td>
<td>0.28</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Appendix B. Statistics report

<table>
<thead>
<tr>
<th>Category</th>
<th>Quantity</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
<th>Hit Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffers Available</td>
<td>123.00</td>
<td>N/A</td>
<td>N/C</td>
<td>N/C</td>
<td>100.00</td>
</tr>
<tr>
<td>Unavailable Buffers - Pool Full</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Number of Dataset Opens</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Buffers Allocated - Pool</td>
<td>2000.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>VPOOL or HPool Exp. Failure</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Concurrent I/O Streams - HWM</td>
<td>0.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Pref. I/O Streams Reduction</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Parallel Query Requests</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Pref. Query Reduced To 1/2</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Pref. Query Reduced To 1/4</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Write Operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffer Updates</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Pages Written</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Buffer Updates/Pages Written</td>
<td>N/C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous Writes</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Asynchronous Writes</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Pages Written Per Write I/O</td>
<td>N/C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Horiz. Def. Write Threshold</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Verte. Def. Write Threshold</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Dm Threshold</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Write Engine Not Available</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
<td></td>
</tr>
</tbody>
</table>

--- Highlights ---

- **Interval Start**: 02/28/07 12:59:34.30
- **Sampling Start**: 02/28/07 12:59:34.30
- **Total Threads**: 0.00
- **Total Commits**: 0.00
- **Interval Elapsed**: 59.994369
- **Sampling Elapsed**: 0.000000
- **Data Sharing Member**: N/A

--- HIGHLIGHTS ---

- **DB2 Version**: V9
- **Scope**: MEMBER
- **To**: 02/28/07 13:00:34.29
CURRENT ACTIVE BUFFERS 26.00 N/A N/A N/A
UNAVAIL.BUFFER-VPOOL FULL 0.00 0.00 N/C N/C

NUMBER OF DATASET OPENS 0.00 0.00 N/C N/C
BUFFERS ALLOCATED - VPOOL 2000.00 N/A N/A N/A

DFHSM MIGRATED DATASET 0.00 0.00 N/C N/C
DFHSM RECALL TIMEOUTS 0.00 0.00 N/C N/C

VPOOL OR HPOOL EXP.FAILURE 0.00 0.00 N/C N/C
CONCUR,PREF./O STREAMS-HWM 0.00 N/A N/A N/A

PREF.QUANT.REDUCED TO 1/2 0.00 0.00 N/C N/C
PREF.QUANT.REDUCED TO 1/4 0.00 0.00 N/C N/C

BP1 WRITE OPERATIONS QUANTITY /SECOND /THREAD /COMMIT
BUFFER UPDATES 0.00 0.00 N/C N/C
PAGES WRITTEN 0.00 0.00 N/C N/C
BUFF.UPDATES/PAGES WRITTEN N/C
SYNCHRONOUS WRITES 0.00 0.00 N/C N/C
ASYNCHRONOUS WRITES 0.00 0.00 N/C N/C
PAGES WRITTEN PER WRITE I/O N/C

BP1 READ OPERATIONS QUANTITY /SECOND /THREAD /COMMIT
PAGES READ 0.00 0.00 N/C N/C
PAGE-INS REQUIRED FOR WRITE 0.00 0.00 N/C N/C
LOCATION: DSNd91B

GROUP: N/P
MEMBER: N/P
SUBSYSTEM: D91B
DB2 VERSION: V9
SCOPE: MEMBER

----- HIGHLIGHTS -----
Appendix B. Statistics report

--- HIGHLIGHTS

INTERVAL START : 02/28/07 12:59:34.30  TOTAL THREADS : 0.00
INTERVAL END : 02/28/07 13:00:34.29  TOTAL COMMITS : 0.00
INTERVAL ELAPSED: 59.994369  OUTAGE ELAPSED: 0.000000  DATA SHARING MEMBER: N/A

BP2 WRITE OPERATIONS  QUANTITY / SECOND / THREAD / COMMIT

BUFFER UPDATES 0.00 0.00 N/C N/C
BUFF. UPDATES/PAGES WRITTEN N/C
SYNCHRONOUS WRITES 0.00 0.00 N/C N/C
ASYNCHRONOUS WRITES 0.00 0.00 N/C N/C
PAGES WRITTEN PER WRITE I/O N/C

BP3 SORT/MERGE QUANTITY / SECOND / THREAD / COMMIT

BUFFER UPDATES 0.00 0.00 N/C N/C
BUFF. UPDATES/PAGES WRITTEN N/C
ASYNCHRONOUS WRITES 0.00 0.00 N/C N/C
PAGES WRITTEN PER WRITE I/O N/C

PAGE-INS REQUIRED FOR READ 0.00 0.00 N/C N/C
PAGES WRITTEN 0.00 0.00 N/C N/C
BUFFER UPDATES 0.00 0.00 N/C N/C
SYNC. READS 0.00 0.00 N/C N/C
ASYNC. READS 0.00 0.00 N/C N/C
VPOOL OR HPOOL EXP.FAILURE 0.00 0.00 N/C N/C
ASYNCHRONOUS WRITES 0.00 0.00 N/C N/C

PAGE-INS REQUIRED FOR WRITE 0.00 0.00 N/C N/C
PAGES WRITTEN 0.00 0.00 N/C N/C
BUFFER UPDATES 0.00 0.00 N/C N/C
ASYNCHRONOUS WRITES 0.00 0.00 N/C N/C
PAGES WRITTEN PER WRITE I/O N/C

BP3 WRITE OPERATIONS  QUANTITY / SECOND / THREAD / COMMIT

BUFFER UPDATES 0.00 0.00 N/C N/C
BUFF. UPDATES/PAGES WRITTEN N/C
ASYNCHRONOUS WRITES 0.00 0.00 N/C N/C
PAGES WRITTEN PER WRITE I/O N/C

PAGE-INS REQUIRED FOR READ 0.00 0.00 N/C N/C
PAGES WRITTEN 0.00 0.00 N/C N/C
BUFFER UPDATES 0.00 0.00 N/C N/C
ASYNCHRONOUS WRITES 0.00 0.00 N/C N/C
PAGES WRITTEN PER WRITE I/O N/C

HORIZ.DEF.WRITE THRESHOLD 0.00 0.00 N/C N/C
VERTI.DEF.WRITE THRESHOLD 0.00 0.00 N/C N/C
DM THRESHOLD 0.00 0.00 N/C N/C
WRITE ENGINE NOT AVAILABLE 0.00 0.00 N/C N/C

VPOOL EXPANS. OR CONTRACT. 0.00 0.00 N/C N/C
VPOOL OR HPOOL EXP.FAILURE 0.00 0.00 N/C N/C
CONCUR, PREF. I/O STREAMS-HWM 0.00 N/A N/A N/A
PAGES READ VIA DYNS.sort/PREF 0.00 0.00 N/C N/C

PARALLEL QUERY REQUESTS 0.00 0.00 N/C N/C
S.PREF, PAGES READ/S.PREF, READ N/C
PREF. QUANT. REDUCED TO 1/2 0.00 0.00 N/C N/C
LIST PREFETCH REQUESTS 0.00 0.00 N/C N/C

PAGES WRITTEN PER WRITE I/O N/C
HORIZ.DEF.WRITE THRESHOLD 0.00 0.00 N/C N/C
VERTI.DEF.WRITE THRESHOLD 0.00 0.00 N/C N/C
DM THRESHOLD 0.00 0.00 N/C N/C

INTERVAL ELAPSED: 59.994369  OUTAGE ELAPSED: 0.000000  DATA SHARING MEMBER: N/A
INTERVAL END: 02/28/07 13:00:34.29  SAMPLING END: 02/28/07 13:00:34.29  TOTAL COMMITS : 0.00
INTERVAL START: 02/28/07 12:59:34.30  SAMPLING START: 02/28/07 12:59:34.30  TOTAL THREADS : 0.00

GROUP: N/P
SUBSYSTEM: 0918
LOCATION: DSN918
OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4)

DB2 VERSION: V9
SCOPE: MEMBER
REQUESTED FROM: NOT SPECIFIED
INTERVAL FROM: 02/28/07 12:59:34.30
INTERVAL TO: 02/28/07 13:00:34.29

PAGE: 1-16
### Data Reporting

**Interval Start:** 02/28/07 12:59:34.30  
**Sampling Start:** 02/28/07 12:59:34.30  
**Total Threads:** 0.00

<table>
<thead>
<tr>
<th>Dataset Operations</th>
<th>Total</th>
<th>/Second</th>
<th>/Thread</th>
<th>/Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read Operations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Write Operations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Buffer Operations**

- **Current Active Buffers:** 1195.00  
- **BPool Hit Ratio (%):** N/C  
- **Pages Written:** N/C  
- **Pages Written Per Write I/O:** N/C

---

**Highlights**

- **Interval Start:** 02/28/07 12:59:34.30  
- **Sampling Start:** 02/28/07 12:59:34.30  
- **Total Threads:** 0.00

**Statistics Report - Long**

**Requested From:** Not Specified  
**Member:** N/P  
**SubSystem:** D91B  
**Db2 Version:** V9  
**Scope:** Member  
**Interval From:** 02/28/07 12:59:34.30  
**To:** 02/28/07 13:00:34.29
Appendix B. Statistics report
<table>
<thead>
<tr>
<th>BP6 WRITE OPERATIONS</th>
<th>QUANTITY /SECOND /THREAD /COMMIT</th>
<th>BP6 SORT/MERGE</th>
<th>QUANTITY /SECOND /THREAD /COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFFER UPDATES</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGES WRITTEN</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFP.UpdatedAt/PAGES WRITTEN</td>
<td>N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYNCHRONOUS WRITES</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASYNCHRONOUS WRITES</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGES WRITTEN PER WRITE I/O</td>
<td>N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HORIZ.DEF.WRITE THRESHOLD</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERTI.DEF.WRITE THRESHOLD</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DM THRESHOLD</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE ENGINE NOT AVAILABLE</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAGE-INS REQUIRED FOR WRITE</td>
<td>0.00 0.00 N/C N/C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--- HIGHLIGHTS

- INTERVAL START: 02/28/07 12:59:34.30
- SAMPLING END: 02/28/07 13:00:34.29
- TOTAL COMMITS: 0.00

--- HIGHLIGHTS

- INTERVAL ELAPSED: 59.994369
- OUTAGE ELAPSED: 0.000000
- TOTAL THREADS: 0.00

--- HIGHLIGHTS

- MEMBER: N/P
- GROUP: N/P
- SUBSYSTEM: D91B
- INTERVAL FROM: 02/28/07 12:59:34.30
- DB2 VERSION: V9

--- HIGHLIGHTS

- MEMBER: N/P
- GROUP: N/P
- SUBSYSTEM: D91B
- INTERVAL FROM: 02/28/07 13:00:34.29
- DB2 VERSION: V9
## Appendix B. Statistics report

### Interval Start: 02/28/07 12:59:34.30

<table>
<thead>
<tr>
<th>Description</th>
<th>Quantity</th>
<th>Second</th>
<th>Thread</th>
<th>Commit</th>
<th>Description</th>
<th>Quantity</th>
<th>Second</th>
<th>Thread</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval Elapsed</td>
<td>59,994369</td>
<td></td>
<td></td>
<td></td>
<td>Outage Elapsed</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### BP8 General

<table>
<thead>
<tr>
<th>Description</th>
<th>Quantity</th>
<th>Second</th>
<th>Thread</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Active Buffers</td>
<td>16.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Unavail. Buffer - VPool Full</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>

### Interval End: 02/28/07 13:00:34.29

### Total Commits: 0.00

### Total Threads: 0.00

### Details:

- **Inter-arrival Elapsed**: 59.994369
- **Outage Elapsed**: 0.000000
- **Data Sharing Member**: N/A
- **Interval Start**: 02/28/07 12:59:34.30
- **Sampling Start**: 02/28/07 12:59:34.30
- **Interval End**: 02/28/07 13:00:34.29
- **Sampling End**: 02/28/07 13:00:34.29
- **Total Commits**: 0.00
- **Total Threads**: 0.00

---

### BP8K General

<table>
<thead>
<tr>
<th>Description</th>
<th>Quantity</th>
<th>Second</th>
<th>Thread</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Active Buffers</td>
<td>10.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Unavail. Buffer - VPool Full</td>
<td>0.00</td>
<td>0.00</td>
<td>N/C</td>
<td>N/C</td>
</tr>
</tbody>
</table>

---

### Interval Elapsed: 59.994369

### Outage Elapsed: 0.000000

### Data Sharing Member: N/A

---

### Interval Start: 02/28/07 12:59:34.30

### Sampling Start: 02/28/07 12:59:34.30

### Total Threads: 0.00

### Total Commits: 0.00

---

### Interval End: 02/28/07 13:00:34.29

### Sampling End: 02/28/07 13:00:34.29

### VPool: EXPANS. OR CONTRACT.

### VPool or HPool EXP. Failure

---

### Synchro. Reads

---

### Buffers Allocated - VPool

### Buffers Allocated - VPool

---

### Interval Elapsed: 59.994369

---

### Outage Elapsed: 0.000000

---

### Data Sharing Member: N/A

---

### Interval Start: 02/28/07 12:59:34.30

### Sampling Start: 02/28/07 12:59:34.30

### Total Threads: 0.00

---

### Total Commits: 0.00

---

### Interval End: 02/28/07 13:00:34.29

---

### Sampling End: 02/28/07 13:00:34.29

---

### VPool: EXPANS. OR CONTRACT.

### VPool or HPool EXP. Failure

---

### Synchro. Reads

---

### Buffers Allocated - VPool

---

### Buffers Allocated - VPool

---

### Interval Elapsed: 59.994369

---

### Outage Elapsed: 0.000000

---

### Data Sharing Member: N/A

---

### Interval Start: 02/28/07 12:59:34.30

### Sampling Start: 02/28/07 12:59:34.30

### Total Threads: 0.00

---

### Total Commits: 0.00

---

### Interval End: 02/28/07 13:00:34.29

### Sampling End: 02/28/07 13:00:34.29

### VPool: EXPANS. OR CONTRACT.

### VPool or HPool EXP. Failure

---

### Synchro. Reads

---

### Buffers Allocated - VPool

---

### Buffers Allocated - VPool

---

### Interval Elapsed: 59.994369

---

### Outage Elapsed: 0.000000

---

### Data Sharing Member: N/A

---

### Interval Start: 02/28/07 12:59:34.30

### Sampling Start: 02/28/07 12:59:34.30

### Total Threads: 0.00

---

### Total Commits: 0.00

---

### Interval End: 02/28/07 13:00:34.29

### Sampling End: 02/28/07 13:00:34.29

### VPool: EXPANS. OR CONTRACT.

### VPool or HPool EXP. Failure

---

### Synchro. Reads

---

### Buffers Allocated - VPool

---

### Buffers Allocated - VPool

---

### Interval Elapsed: 59.994369

---

### Outage Elapsed: 0.000000

---

### Data Sharing Member: N/A

---

### Interval Start: 02/28/07 12:59:34.30

### Sampling Start: 02/28/07 12:59:34.30

### Total Threads: 0.00

---

### Total Commits: 0.00

---

### Interval End: 02/28/07 13:00:34.29

### Sampling End: 02/28/07 13:00:34.29

### VPool: EXPANS. OR CONTRACT.

### VPool or HPool EXP. Failure

---

### Synchro. Reads

---

### Buffers Allocated - VPool

---

### Buffers Allocated - VPool
DB2 9 for z/OS Performance Topics

PREF.QUANT.REDUCED TO 1/2  0.00  0.00  N/C  N/C  LIST PREFETCH REQUESTS  0.00  0.00  N/C  N/C
PREF.QUANT.REDUCED TO 1/4  0.00  0.00  N/C  N/C  LIST PREFETCH READS  0.00  0.00  N/C  N/C
                       PAGES READ VIA LIST PREFETCH  0.00  0.00  N/C  N/C
                       L.PRF.PAGES READ/L.PRF.READ  N/C
                       DYNAMIC PREFETCH REQUESTED  0.00  0.00  N/C  N/C
                       DYNAMIC PREFETCH READS  0.00  0.00  N/C  N/C
                       PAGES READ VIA DYN.PREFETCH  0.00  0.00  N/C  N/C
                       D.PRF.PAGES READ/D.PRF.READ  N/C
                       PREF.DISABLED-NO BUFFER  0.00  0.00  N/C  N/C
                       PREF.DISABLED-NO READ ENG  0.00  0.00  N/C  N/C
                       PAGE-INS REQUIRED FOR READ  0.00  0.00  N/C  N/C

BPBK WRITE OPERATIONS QUANTITY /SECOND /THREAD /COMMIT  BPBK Sort/Merge QUANTITY /SECOND /THREAD /COMMIT
------------------------------------------------------------------  ----------------------------------------
BUFFER UPDATES  0.00  0.00 N/C N/C  MAX WORKFILES CONCURR. USED  0.00  N/A  N/A N/A
BUFF. UPDATES/PAGES WRITTEN N/C  MERGE PASSES REQUESTED  0.00  0.00 N/C N/C
SYNCHRONOUS WRITES  0.00  0.00 N/C N/C  MERGE PASS DEGRADED-LOW BUF  0.00  0.00 N/C N/C
ASYNCHRONOUS WRITES  0.00  0.00 N/C N/C  WORKFILE REQ-REJCTD-LOW BUF  0.00  0.00 N/C N/C
PAGE-INS REQUIRED FOR WRITE  0.00  0.00 N/C N/C  WORKFILE REQ-ALL MERGE PASS  0.00  0.00 N/C N/C
PAGES WRITTEN PER WRITE I/O N/C

VPOOL OR HPOOL EXP.FAILURE 0.00  0.00 N/C N/C
VPOOL EXPANS. OR CONTRACT. /PPOOL 0.00  0.00 N/C N/C
CONCUR.PREF.1/D STREAMS-HWM 0.00  0.00 N/A N/A
CONCUR.PREF.1/D STREAMS REDUCTION 0.00  0.00 N/C N/C
PARALLEL QUERY REQUESTS  0.00  0.00 N/C N/C  S.PRF.PAGES READ/S.PRF.READ  N/C
PARALLEL QUERY REDUCTION  0.00  0.00 N/C N/C
PREF.QUANT.REDUCED TO 1/2  0.00  0.00 N/C N/C  LIST PREFETCH REQUESTS  0.00  0.00 N/C N/C
PREF.QUANT.REDUCED TO 1/4  0.00  0.00 N/C N/C  LIST PREFETCH READS  0.00  0.00 N/C N/C
                       PAGES READ VIA LIST PREFETCH  0.00  0.00 N/C  N/C
                       L.PRF.PAGES READ/L.PRF.READ  N/C
                       DYNAMIC PREFETCH REQUESTED  0.00  0.00  N/C  N/C
                       DYNAMIC PREFETCH READS  0.00  0.00  N/C  N/C
                       PAGES READ VIA DYN.PREFETCH  0.00  0.00  N/C  N/C
                       D.PRF.PAGES READ/D.PRF.READ  N/C
                       PREF.DISABLED-NO BUFFER  0.00  0.00  N/C  N/C
                       PREF.DISABLED-NO READ ENG  0.00  0.00  N/C  N/C
                       PAGE-INS REQUIRED FOR READ  0.00  0.00  N/C  N/C

TOTAL WRITE OPERATIONS QUANTITY /SECOND /THREAD /COMMIT  TOTAL READ OPERATIONS QUANTITY /SECOND /THREAD /COMMIT
------------------------------------------------------------------  ----------------------------------------
BUFFER UPDATES  0.00  0.00 N/C N/C  MAX WORKFILES CONCURR. USED  0.00  N/A  N/A N/A
BUFF. UPDATES/PAGES WRITTEN N/C  MERGE PASSES REQUESTED  0.00  0.00 N/C N/C
SYNCHRONOUS WRITES  0.00  0.00 N/C N/C  MERGE PASS DEGRADED-LOW BUF  0.00  0.00 N/C N/C
ASYNCHRONOUS WRITES  0.00  0.00 N/C N/C  WORKFILE REQ-REJCTD-LOW BUF  0.00  0.00 N/C N/C
PAGE-INS REQUIRED FOR WRITE  0.00  0.00 N/C N/C  WORKFILE REQ-ALL MERGE PASS  0.00  0.00 N/C N/C
PAGES WRITTEN PER WRITE I/O N/C

--- HIGHLIGHTS ---
INTERVAL START : 02/28/07 12:59:34.30  INTERVAL FROM: 02/28/07 12:59:34.30
INTERVAL END : 02/28/07 13:00:34.29  INTERVAL ELAPSED:  59.994369
TOTAL THREADS :  0.00  DATA SHARING MEMBER: N/A

GROUP: N/P
STATISTICS REPORT - LONG
REQUESTED FROM: NOT SPECIFIED
MEMBER: N/P
TO: NOT SPECIFIED
DB2 VERSION: V9
SCOPE: MEMBER
OEMAG0N XE FOR DBZ PERFORMANCE EXPERT (V4)
PAGE: 1-23
INTERVAL START: 02/28/07 12:59:34.30
INTERVAL FROM: 02/28/07 12:59:34.30
INTERVAL ELAPSED: 0:00:0000
TOTAL THREADS: 0.00
DATA SHARING MEMBER: N/A

GROUP: N/P
STATISTICS REPORT - LONG
REQUESTED FROM: NOT SPECIFIED
MEMBER: N/P
TO: NOT SPECIFIED
DB2 VERSION: V9
SCOPE: MEMBER
OEMAG0N XE FOR DBZ PERFORMANCE EXPERT (V4)
PAGE: 1-23
INTERVAL START: 02/28/07 12:59:34.30
INTERVAL FROM: 02/28/07 13:00:34.29
INTERVAL ELAPSED: 0:00:0000
TOTAL THREADS: 0.00
DATA SHARING MEMBER: N/A
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pages Written per Write I/O</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Horizontal Def. Write Threshold</td>
<td>0.00</td>
<td>N/C</td>
</tr>
<tr>
<td>Vertical Def. Write Threshold</td>
<td>0.00</td>
<td>N/C</td>
</tr>
<tr>
<td>DM Threshold</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Write Engine Not Available</td>
<td>N/C</td>
<td></td>
</tr>
<tr>
<td>Page-Ins Required for Write</td>
<td>0.00</td>
<td>N/C</td>
</tr>
<tr>
<td><strong>Location</strong>: DSN91B</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Group</strong>: N/P</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Subsystem</strong>: D91B</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DB2 Version</strong>: V9</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Scope</strong>: Member</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Appendix B. Statistics report

321
B.2 OMEGAMON XE Performance Expert accounting report long layout

The OMEGAMON XE accounting report was produced by processing the SMF data from a DB2 V9 subsystem. The data in this OMEGAMON XE Performance Expert accounting report is referenced in 4.12, "Latch class contention relief" on page 114. Example B-3 shows the processing parameters that are used.

Example: B-3  Statement for the accounting report

GLOBAL
ACCOUNTING TRACE DDNAME(AREPORT)
LAYOUT(LONG)
EXEC

Example B-4 shows the output of the report.

```
ACCOUNTING REPORT COMPLETE

LOCATION: DSN91B
GROUP: N/P
MEMBER: N/P
SUBSYSTEM: D91B
DB2 VERSION: V9

CONNTYPE: DRDA

ELAPSED TIME DISTRIBUTION
---------------------------
APPL | 31%   CPU | 1%
DB2  |  3%   NOTACC | 1%
SUSP | 66%   SUSP | 96%

CLASS 2 TIME DISTRIBUTION
---------------------------
APPL(CL.1) | DB2 (CL.2) | IPI (CL.5)  | CLASS 3 SUSPENSIONS | AVERAGE TIME | AV.EVENT | HIGHLIGHTS
------------ | ---------- | ---------- | ------------------- | ------------ | -------- | --------------------------
ELAPSED TIME | 0.093075 | 0.064136 | N/P | SYNCHRON. 1/0 | 0.057264 | 0.70 | #OCCURRENCES : 6184
NONNESTED | 0.093075 | 0.064136 | N/A | DATABASE 1/0 | 0.002905 | 3.38 | #ALLIEDS DISTRIB: 0
STORED PROC | 0.000000 | 0.000000 | N/A | LOG WRITE 1/0 | 0.000833 | 5.00 | #DBATS : 6184
TRIGGER | 0.000000 | 0.000000 | N/A | OTHER READ 1/0 | 0.000270 | 0.15 | #DBATS DISTRIB: 0
CP CPU TIME | 0.001371 | 0.000908 | N/P | SER.TASK SWTCH | 0.000221 | 0.03 | #NORMAL TERMINAT: 6184
AGENT | 0.001371 | 0.000908 | N/A | UPDATE COMMIT | 0.000000 | 0.00 | #ABNORMAL TERMINAT: 0
STORED PRC | 0.000000 | 0.000000 | N/A | SYSLGRNG REC | 0.000154 | 0.01 | #CP/X PARALLELISM: 0
UDF | 0.000000 | 0.000000 | N/A | EXT/DEL/DEF | 0.000046 | 0.00 | #INCREMENT. BIND: 0
PAR.TASKS | 0.000000 | 0.000000 | N/A | OTHER SERVICE | 0.000000 | 0.00 | #COMMTS : 6185
1IPCP CPU | 0.000122 | N/A | N/A | DRAIN LOCK | 0.000000 | 0.00 | #SCROLL LOCK: 0
1IP CPU TIME | 0.001391 | 0.000878 | N/A | PAGE LATCH | 0.000006 | 0.00 | MAX SQL CASC LVL: 0
STORED PROC | 0.000000 | 0.000000 | N/A | NOTIFY MSGS | 0.000000 | 0.00 | UPDATE/COMMIT : 7.33
SUSPEND TIME | 0.000000 | 0.061519 | N/A | COMMIT PH1 WRITE 1/0 | 0.000000 | 0.00 | #SYNCH AVG: 0.000964
AGENT | N/A | 0.061519 | N/A | SYSNCH CF REQUESTS | 0.000000 | 0.00 | #SYNCH REQUESTS: 0
PAR.TASKS | N/A | 0.000000 | N/A | TCP/IP LDB | 0.000000 | 0.00 | #TOTAL COMMIT: 0
STORED PROC | 0.000000 | N/A | N/A | TOTAL CLASS 3 | 0.061519 | 4.76 | #TOTAL CLASS 3: 0
UDF | 0.000000 | N/A | N/A

GLOBAL CONTENTION
L-LOCKS | AVERAGE TIME | AV.EVENT
------ | ---------- | -------
0.000000 | 0.00 | P-LOCKS | AVERAGE TIME | AV.EVENT
0.000000 | 0.00

PARENT (DB,T5,TAB,PAR)
0.000000 | 0.00 | PAGESET/PARTITION | 0.000000 | 0.00
```
### Appendix B. Statistics report

<table>
<thead>
<tr>
<th>SQL DML</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>4.40</td>
<td>27217</td>
</tr>
<tr>
<td>INSERT</td>
<td>3.08</td>
<td>19044</td>
</tr>
<tr>
<td>UPDATE</td>
<td>4.02</td>
<td>24838</td>
</tr>
<tr>
<td>DELETE</td>
<td>0.23</td>
<td>1430</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DDL

<table>
<thead>
<tr>
<th>SQL DDL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DROP</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>ALTER</td>
<td>0.00</td>
<td>0</td>
</tr>
</tbody>
</table>

### SQL DCL

<table>
<thead>
<tr>
<th>SQL DCL</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GRANT</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DCL TABLE</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LOGGING</td>
<td>AVERAGE</td>
<td>TOTAL</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>LOG RECORDS WRITTEN</td>
<td>16.20</td>
<td>100195</td>
</tr>
<tr>
<td>TOT BYTES WRITTEN</td>
<td>1766.39</td>
<td>10923346</td>
</tr>
<tr>
<td>LOG RECORD SIZE</td>
<td>109.02</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AVERAGE SU</th>
<th>CLASS 1</th>
<th>CLASS 2</th>
<th>DYNAMIC SQL STMT</th>
<th>AVERAGE</th>
<th>TOTAL</th>
<th>MISCELLANEOUS</th>
<th>AVERAGE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP CPU</td>
<td>38.88</td>
<td>25.76</td>
<td>MAX STOR LOB VALUES</td>
<td>0.00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGENT</td>
<td>38.88</td>
<td>25.76</td>
<td>NOT FOUND IN CACHE</td>
<td>0.00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NONNESTED</td>
<td>38.88</td>
<td>25.76</td>
<td>FOUND IN CACHE</td>
<td>0.00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STORED PRC</td>
<td>0.00</td>
<td>0.00</td>
<td>IMPLICIT PREPARES</td>
<td>0.00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOD</td>
<td>0.00</td>
<td>0.00</td>
<td>PREPARES AVOIDED</td>
<td>0.00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRIGGER</td>
<td>0.00</td>
<td>0.00</td>
<td>CACHE_LIMIT_EXCEEDED</td>
<td>0.00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAR.TASKS</td>
<td>0.00</td>
<td>0.00</td>
<td>PREP_STMT_PURGED</td>
<td>0.00</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| IIPCP CPU | 3.46 | N/A |
| IIP CPU   | 39.46 | 24.90 |

1 LOCATION: DSND91B  
GROUP: N/P  
SUBSYSTEM: D91B  
MEMBER: N/P  
CONNTYPE: DRDA  
DB2 VERSION: V9  
SCOPE: MEMBER

BP0  BPOOL ACTIVITY  AVERAGE  TOTAL
---------------------  --------  --------
BPOOL HIT RATIO (%)  98.94  N/A
GETPAGES  0.37  2272
BUFFER UPDATES  0.26  1613
SYNCHRONOUS WRITE  0.00  0
SYNCHRONOUS READ  0.00  16
SEQ. PREFETCHREQS  0.00  1
LIST PREFETCHREQS  0.00  0
DYN. PREFETCHREQS  0.00  0
PAGES READ ASYNCHR.  0.00  8

BP1  BPOOL ACTIVITY  AVERAGE  TOTAL
---------------------  --------  --------
BPOOL HIT RATIO (%)  100.00  N/A
GETPAGES  0.47  2910
BUFFER UPDATES  0.24  1508
SYNCHRONOUS WRITE  0.00  0
SYNCHRONOUS READ  0.00  0
SEQ. PREFETCHREQS  0.00  0
LIST PREFETCHREQS  0.00  0
DYN. PREFETCHREQS  0.00  0
PAGES READ ASYNCHR.  0.00  0

BP2  BPOOL ACTIVITY  AVERAGE  TOTAL
---------------------  --------  --------
BPOOL HIT RATIO (%)  100.00  N/A
GETPAGES  20.33  125717
BUFFER UPDATES  0.71  4387
SYNCHRONOUS WRITE  0.00  0
SYNCHRONOUS READ  0.00  5
SEQ. PREFETCHREQS  0.00  0
LIST PREFETCHREQS  0.00  0
DYN. PREFETCHREQS  0.12  714
PAGES READ ASYNCHR.  0.00  0

BP3  BPOOL ACTIVITY  AVERAGE  TOTAL
---------------------  --------  --------
BPOOL HIT RATIO (%)  98.53  N/A
GETPAGES  21.98  135953
BUFFER UPDATES  0.00  0
SYNCHRONOUS WRITE  0.00  0
SYNCHRONOUS READ  0.32  1997
SEQ. PREFETCHREQS  0.00  0
LIST PREFETCHREQS  0.00  0
DYN. PREFETCHREQS  0.79  4884
PAGES READ ASYNCHR.  0.00  0

1 LOCATION: DSND91B  
GROUP: N/P  
SUBSYSTEM: D91B  
MEMBER: N/P  
CONNTYPE: DRDA  
DB2 VERSION: V9  
SCOPE: MEMBER

324  DB2 9 for z/OS Performance Topics
### BP4 BPOOL ACTIVITY

<table>
<thead>
<tr>
<th>Activity</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>94.35</td>
<td>N/A</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>9.38</td>
<td>57976</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>3.41</td>
<td>21084</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>0.39</td>
<td>2405</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH REQS</td>
<td>0.23</td>
<td>1444</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>0.14</td>
<td>873</td>
</tr>
</tbody>
</table>

### BP5 BPOOL ACTIVITY

<table>
<thead>
<tr>
<th>Activity</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>89.68</td>
<td>N/A</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>1.61</td>
<td>9980</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>0.99</td>
<td>6133</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>0.17</td>
<td>1030</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>0.23</td>
<td>1430</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>0.14</td>
<td>873</td>
</tr>
</tbody>
</table>

### BP6 BPOOL ACTIVITY

<table>
<thead>
<tr>
<th>Activity</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>66.47</td>
<td>N/A</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>1.98</td>
<td>12253</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>5.03</td>
<td>31121</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>0.36</td>
<td>2232</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>0.04</td>
<td>277</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>0.01</td>
<td>57</td>
</tr>
</tbody>
</table>

### BP7 BPOOL ACTIVITY

<table>
<thead>
<tr>
<th>Activity</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>48.37</td>
<td>N/A</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>1.37</td>
<td>8449</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>0.92</td>
<td>5705</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>0.70</td>
<td>4305</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH REQS</td>
<td>0.00</td>
<td>2</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>0.00</td>
<td>2</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>0.01</td>
<td>57</td>
</tr>
</tbody>
</table>

### BP8 BPOOL ACTIVITY

<table>
<thead>
<tr>
<th>Activity</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>67.49</td>
<td>N/A</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>11.01</td>
<td>66103</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>2.37</td>
<td>14649</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>1.43</td>
<td>8852</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH REQS</td>
<td>0.33</td>
<td>2049</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>2.15</td>
<td>13291</td>
</tr>
</tbody>
</table>

### TOT4K BPOOL ACTIVITY

<table>
<thead>
<tr>
<th>Activity</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>91.28</td>
<td>N/A</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>13.94</td>
<td>86200</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>3.37</td>
<td>20842</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0.56</td>
<td>3479</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>1.18</td>
<td>7321</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>2.60</td>
<td>16106</td>
</tr>
</tbody>
</table>

### BP8K BPOOL ACTIVITY

<table>
<thead>
<tr>
<th>Activity</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>94.35</td>
<td>N/A</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>9.38</td>
<td>57976</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>3.41</td>
<td>21084</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>0.39</td>
<td>2405</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH REQS</td>
<td>0.23</td>
<td>1444</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>0.14</td>
<td>873</td>
</tr>
</tbody>
</table>

---

**Location:** DSNDB91B

**Subsystem:** D91B

**DB2 Version:** V9

**Conntype:** DRDA

**Accounting Report - Long**

**Scope:** Member

**Order:** Conntype

**Interval From:** 02/28/07 12:54:59.17

**To:** 02/28/07 12:57:48.89

**Requested From:** Not Specified

**To:** Not Specified

---

**Appendix B. Statistics report**
### BPOOL Activity Report

<table>
<thead>
<tr>
<th>Metric</th>
<th>Average</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>91.25</td>
<td>N/A</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>68.52</td>
<td>423757</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>13.94</td>
<td>86200</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>3.38</td>
<td>20901</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>0.56</td>
<td>3479</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH REQS</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>1.19</td>
<td>7330</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>2.61</td>
<td>16171</td>
</tr>
</tbody>
</table>

#### DISTRIBUTED ACTIVITY

1. LOCATION: DSND91B
   - PRODUCT ID: COMMON SERV
   - METHOD: DRDA PROTOCOL
   - ORDER: CONNTYPE
   - SCOPE: MEMBER
   - INTERVAL FROM: 02/28/07 12:54:59.17
   - INTERVAL TO: 02/28/07 12:57:48.89

<table>
<thead>
<tr>
<th>Event</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUESTER</td>
<td>::FFFF:9.30.12#1</td>
</tr>
<tr>
<td>TRANSACTIONS RECEIVED</td>
<td>0.06</td>
</tr>
<tr>
<td>MESSAGES SENT</td>
<td>23.62</td>
</tr>
<tr>
<td>MSG. IN BUFFER</td>
<td>5.03</td>
</tr>
<tr>
<td>METHOD</td>
<td>DRDA PROTOCOL</td>
</tr>
<tr>
<td>#COMMIT(1) RECEIVED</td>
<td>6185</td>
</tr>
<tr>
<td>MESSAGES RECEIVED</td>
<td>23.62</td>
</tr>
<tr>
<td>ROWS SENT</td>
<td>12.53</td>
</tr>
<tr>
<td>#ROLLBACK(1) RECEIVED</td>
<td>0</td>
</tr>
<tr>
<td>BYTES SENT</td>
<td>3043.73</td>
</tr>
<tr>
<td>BLOCKS SENT</td>
<td>4.76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONV. INITIATED</td>
<td>0.06</td>
</tr>
<tr>
<td>SQL RECEIVED</td>
<td>22.38</td>
</tr>
<tr>
<td>BYTES RECEIVED</td>
<td>3100.78</td>
</tr>
<tr>
<td>#DDF ACCESSES</td>
<td>6184</td>
</tr>
<tr>
<td>COMMIT(2) RECEIVED</td>
<td>0</td>
</tr>
<tr>
<td>COMMIT(2) RES. SENT</td>
<td>0</td>
</tr>
<tr>
<td>COMMIT(2) RES. SENT</td>
<td>0</td>
</tr>
<tr>
<td>COMMIT(2) PERFORM.</td>
<td>0</td>
</tr>
<tr>
<td>COMMIT(2) PERFORM.</td>
<td>0</td>
</tr>
<tr>
<td>#THREADS IN DOUBT</td>
<td>0</td>
</tr>
<tr>
<td>#COMMIT(2) RES. SENT</td>
<td>0</td>
</tr>
<tr>
<td>#ROLLBACK(2) RES. SENT</td>
<td>0</td>
</tr>
<tr>
<td>#ROLLBACK(2) RES. SENT</td>
<td>0</td>
</tr>
<tr>
<td>LAST AGENT RECEIVED</td>
<td>0</td>
</tr>
</tbody>
</table>

2. LOCATION: DSND91B
   - PRODUCT ID: COMMON SERV
   - METHOD: DRDA PROTOCOL
   - ORDER: CONNTYPE
   - SCOPE: MEMBER
   - INTERVAL FROM: 02/28/07 12:54:59.17
   - INTERVAL TO: 02/28/07 12:57:48.89

<table>
<thead>
<tr>
<th>Event</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUESTER</td>
<td>::FFFF:9.30.12#1</td>
</tr>
<tr>
<td>TRANSACTIONS RECEIVED</td>
<td>0.06</td>
</tr>
<tr>
<td>MESSAGES SENT</td>
<td>23.62</td>
</tr>
<tr>
<td>MSG. IN BUFFER</td>
<td>5.03</td>
</tr>
<tr>
<td>METHOD</td>
<td>DRDA PROTOCOL</td>
</tr>
<tr>
<td>#COMMIT(2) RECEIVED</td>
<td>6184</td>
</tr>
<tr>
<td>MESSAGES RECEIVED</td>
<td>23.62</td>
</tr>
<tr>
<td>ROWS SENT</td>
<td>12.53</td>
</tr>
<tr>
<td>#ROLLBACK(1) RECEIVED</td>
<td>0</td>
</tr>
<tr>
<td>BYTES SENT</td>
<td>3043.73</td>
</tr>
<tr>
<td>BLOCKS SENT</td>
<td>4.76</td>
</tr>
<tr>
<td>CONV. INITIATED</td>
<td>0.06</td>
</tr>
<tr>
<td>SQL RECEIVED</td>
<td>22.38</td>
</tr>
<tr>
<td>BYTES RECEIVED</td>
<td>3100.78</td>
</tr>
<tr>
<td>#DDF ACCESSES</td>
<td>6184</td>
</tr>
<tr>
<td>COMMIT(2) RECEIVED</td>
<td>0</td>
</tr>
<tr>
<td>COMMIT(2) RES. SENT</td>
<td>0</td>
</tr>
<tr>
<td>COMMIT(2) RES. SENT</td>
<td>0</td>
</tr>
<tr>
<td>COMMIT(2) PERFORM.</td>
<td>0</td>
</tr>
<tr>
<td>COMMIT(2) PERFORM.</td>
<td>0</td>
</tr>
<tr>
<td>#THREADS IN DOUBT</td>
<td>0</td>
</tr>
</tbody>
</table>

ACCOUNTING REPORT COMPLETE
EXPLAIN tables

In this appendix, we describe the DB2 EXPLAIN function and show the contents of the following EXPLAIN tables:

- DSN_PLAN_TABLE
- DSN_STATEMENT_TABLE
- DSN_FUNCTION_TABLE
- DSN_STATEMENT_CACHE_TABLE

We describe each of these EXPLAIN tables, including the content that has not changed with V9, in order to make the material more complete and understandable.

**Important:** The EXPLAIN tables formats for DB2 V7 and prior versions are deprecated. See APAR PK85068 for help in migrating to later versions.

EXPLAIN describes the access paths that are selected by DB2 to execute an SQL statement (UPDATE, SELECT, DELETE, and INSERT), obtains information about how SQL statements from a package or plan will execute, and inserts that information into the `package_owner.PLAN_TABLE` or `plan_owner.PLAN_TABLE`. For dynamically prepared SQL, the qualifier is the current SQLID. An EXPLAIN can be executed either statically from an application program or dynamically using Query Management Facility (QMF) or SQL Processor Using File In (SPUFI).

You can use EXPLAIN to:

- Help application program design
- Assist in database design
- Give a description of the access path that is chosen for a query by DB2
- Help you verify if your application needs to be rebound
- Check if the index or table scan is used by a statement
- Check if DB2 plans are using parallelism

Before you start using EXPLAIN, you need to create a PLAN_TABLE to hold the results of the EXPLAIN. The EXPLAIN can use other tables, such as DSN_FUNCTION_TABLE, DSN_STATEMENT_TABLE, and DSN_STATEMENT_CACHE_TABLE. However, unless you need the information they provide, it is not necessary to create them to use EXPLAIN. A description of the content of these tables follows.
EXPLAIN informs you about the access paths that DB2 chooses and can tell you the following details:

- The number of indexes used
- The I/O method used to read the pages (list prefetch or sequential prefetch)
- The number of index columns that are used as search criteria
- Whether an index access or table scan is used
- The join method
- The order in which the tables are joined
- When and why sorts are performed
- If DB2 plans to use multiple concurrent I/O streams to access your data

This information is saved in the PLAN_TABLE. This table has been evolving with each release of DB2 by adding new columns in order to provide new functions. DB2 inserts rows into the PLAN_TABLE whenever a plan or a package is bound, reound, or rebound with the EXPLAIN(YES) option, or when a program or tool explicitly explains the SQL statement.

Emptying the PLAN_TABLE: If you want to empty the PLAN_TABLE, you must use the DELETE statement, just as you would to delete rows from any table. You also can use the DROP TABLE statement to drop the PLAN_TABLE completely. The action of binding, rebounding, or executing another SQL statement does not replace or delete rows from the PLAN_TABLE.

With DB2 V9:

- The DSN_PLAN_TABLE has one additional column defined, PARENT_PLANNO. It is used together with PARENT_QBLOCKNO to connect a child query block to the parent miniplan for global query optimization. We list all columns in Table C-2 on page 329.
- The DSN_FUNCTION_TABLE has the same columns, but two have changed data type. We list all columns in Table C-4 on page 336.
- The DSN_STATEMENT_TABLE has one new column, TOTAL_COST. We list all columns in Table C-5 on page 339.
- For the DSN_STATEMENT_CACHE_TABLE, we show the CREATE of all columns in Example C-1 on page 337.

### C.1 DSN_PLAN_TABLE

The table can be created by the statements that are contained in member name DSNTESC of the DB2 sample library. DB2 tools also create or update your PLAN_TABLE when needed.

Let us look at the evolution of the PLAN_TABLE in Table C-1. It is also a way to see the evolution of DB2.

<table>
<thead>
<tr>
<th>DB2 release</th>
<th>Columns in PLAN_TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>25</td>
</tr>
<tr>
<td>V1R2</td>
<td>28</td>
</tr>
<tr>
<td>V2</td>
<td>30</td>
</tr>
<tr>
<td>V3</td>
<td>34</td>
</tr>
</tbody>
</table>
Starting with V8, several columns have grown in size to accommodate large names. The 59-column format gives you the most information. If you alter an existing PLAN_TABLE to add new columns, in general, specify the columns as NOT NULL WITH DEFAULT, so that default values are included for the rows that are already in the table. However, as you can see in Table C-2, there are some exceptions, meaning that certain columns do allow nulls. Do not specify those columns as NOT NULL WITH DEFAULT.

Table C-2 describes the PLAN_TABLE, with a small description of each column and a brief history of DB2 evolution. For more information about the PLAN_TABLE, refer to *DB2 for z/OS Version 9.1 Performance Monitoring and Tuning Guide*, SC18-9851.

**Note:** There are some objects for which access is not described by EXPLAIN. For example, such types of access include access to LOB values, which are stored separately from the base table, access to parent or dependent tables that are needed to enforce referential constraints, SQL for routines (triggers, functions, or stored procedures), and explicit access to SECLABEL for row-level security.
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>SMALLINT NOT NULL</td>
<td>A number (0, 1, 2, 3, or 4) that indicates the join method used for the step:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0   First table accessed, continuation of the previous table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1   NESTED LOOP JOIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2   MERGE SCAN JOIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3   Sorts needed by ORDER BY, GROUP BY, SELECT DISTINCT, or UNION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4   HYBRID JOIN</td>
</tr>
<tr>
<td>CREATOR</td>
<td>VARCHAR(128) NOT NULL</td>
<td>The creator of the new table accessed in this step; blank if METHOD is 3.</td>
</tr>
<tr>
<td>TNAME</td>
<td>VARCHAR(128) NOT NULL</td>
<td>The name of a table, materialized query table, created or declared temporary table, materialized view, or materialized table expression.</td>
</tr>
<tr>
<td>TABNO</td>
<td>SMALLINT NOT NULL</td>
<td>Values are for IBM use only.</td>
</tr>
<tr>
<td>ACCESTYPE</td>
<td>CHAR(2) NOT NULL</td>
<td>The method of accessing the new table:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DI  By an intersection of multiple DOCID lists to return the final DOCID list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DU  By a union of multiple DOCID lists to return the final DOCID list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DX  By an XML index scan on the index named in ACCESSNAME to return a DOCID list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E   By direct row access using a row change timestamp column.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I   Index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I1  One-fetch index scan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M   Multiple index scan (followed by MX, MI, or MU)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MI  Intersection of multiple indexes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MU  Union of multiple indexes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MX  Index scan on the index named in ACCESSNAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N   Index scan when the matching predicate contains the IN keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P   By a dynamic index ANDing scan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R   Table space scan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RW  Work file scan of the result of a materialized user-defined table function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T   Sparse index (star join work files)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V   Buffers for an INSERT statement within a SELECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>blank  Not applicable to the current row ACCESTYPE</td>
</tr>
<tr>
<td>MATCHCOLS</td>
<td>SMALLINT NOT NULL</td>
<td>For ACCESTYPE I, I1, N, or MX, the number of index keys used in an index scan; otherwise, 0.</td>
</tr>
<tr>
<td>ACCESSCREATOR</td>
<td>VARCHAR(128) NOT NULL</td>
<td>For ACCESTYPE I, I1, N, or MX, the creator of the index; otherwise, blank.</td>
</tr>
<tr>
<td>ACCESSNAME</td>
<td>VARCHAR(128) NOT NULL</td>
<td>For ACCESTYPE I, I1, N, or MX, the name of the index; otherwise, blank.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Content</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INDEXONLY</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether access to an index alone is enough to carry out the step, or whether data too must. Y Yes N No</td>
</tr>
<tr>
<td>SORTN_UNIQ</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether the new table is sorted to remove duplicate rows. Y Yes N No</td>
</tr>
<tr>
<td>SORTN_JOIN</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether the new table is sorted for join method 2 or 4. Y Yes N No</td>
</tr>
<tr>
<td>SORTN_ORDERBY</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether the new table is sorted for ORDER BY. Y Yes N No</td>
</tr>
<tr>
<td>SORTN_GROUPBY</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether the new table is sorted for GROUP BY. Y Yes N No</td>
</tr>
<tr>
<td>SORTC_UNIQ</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether the composite table is sorted to remove duplicate rows. Y Yes N No</td>
</tr>
<tr>
<td>SORTC_JOIN</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether the composite table is sorted for join method 1, 2 or 4. Y Yes N No</td>
</tr>
<tr>
<td>SORTC_ORDER BY</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether the composite table is sorted for an ORDER BY clause or a quantified predicate. Y Yes N No</td>
</tr>
<tr>
<td>SORTC_GROUP BY</td>
<td>CHAR(1) NOT NULL</td>
<td>Whether the composite table is sorted for a GROUP BY clause. Y Yes N No</td>
</tr>
<tr>
<td>TSOLOCKMODE</td>
<td>CHAR(3) NOT NULL</td>
<td>An indication of the mode of lock to be acquired on either the new table, its table space, or table space partitions. If the isolation can be determined at bind time.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>CHAR(16) NOT NULL</td>
<td>The time at which the row is processed, to the last .01 second. If necessary, DB2 adds .01 second to the value to ensure that rows for two successive queries have different values.</td>
</tr>
<tr>
<td>REMARKS</td>
<td>VARCHAR(762) NOT NULL</td>
<td>A field into which you can insert any character string of 762 or fewer characters.</td>
</tr>
</tbody>
</table>

------ 25 COLUMNS FORMAT -----Version 1 -1983--
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
</table>
| PREFETCH         | CHAR(1) NOT NULL WITH DEFAULT | S  Pure sequential prefetch  
|                  |                        | L  Prefetch through a page list  
|                  |                        | D  Optimizer expects dynamic prefetch  
|                  |                        | Blank  Unknown or no prefetch  |
| COLUMN_FN_EVAL   | CHAR(1) NOT NULL WITH DEFAULT | When an SQL aggregate function is evaluated.  
|                  |                        | R  While the data is being read from the table or index  
|                  |                        | S  While performing a sort to satisfy a GROUP BY clause  
|                  |                        | Blank  After data retrieval and after any sorts.  |
| MIXOPSEQ         | SMALLINT NOT NULL WITH DEFAULT | The sequence number of a step in a multiple index operation.  
|                  |                        | 1, 2, ..., N  For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU.)  
|                  |                        | 0  For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank.)  |

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION</td>
<td>VARCHAR(64) NOT NULL WITH DEFAULT</td>
<td>The version identifier for the package.</td>
</tr>
<tr>
<td>COLLID</td>
<td>CHAR(18) NOT NULL WITH DEFAULT</td>
<td>The collection ID for the package.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS_DEGREE</td>
<td>SMALLINT</td>
<td>The number of parallel tasks or operations activated by a query.</td>
</tr>
<tr>
<td>ACCESS_PGROUP_ID</td>
<td>SMALLINT</td>
<td>The identifier of the parallel group for accessing the new table.</td>
</tr>
<tr>
<td>JOIN_DEGREE</td>
<td>SMALLINT</td>
<td>The number of parallel operations or tasks used in joining the composite table with the new table.</td>
</tr>
<tr>
<td>JOIN_PGROUP_ID</td>
<td>SMALLINT</td>
<td>The identifier of the parallel group for joining the composite table with the new table.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORTC_PGROUP_ID</td>
<td>SMALLINT</td>
<td>The parallel group identifier for the parallel sort of the composite table.</td>
</tr>
<tr>
<td>SORTTN_PGROUP_ID</td>
<td>SMALLINT</td>
<td>The parallel group identifier for the parallel sort of the new table.</td>
</tr>
</tbody>
</table>
| PARALLELISM_MODE | CHAR(1)                | The kind of parallelism, if any, that is used at bind time:  
|                  |                        | I  Query I/O parallelism  
|                  |                        | C  Query CP parallelism  
<p>|                  |                        | X  Sysplex query parallelism  |
| MERGE_JOIN_COLS  | SMALLINT               | The number of columns that are joined during a merge scan join (Method=2).  |</p>
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORRELATION_NAME</td>
<td>VARCHAR(128)</td>
<td>The correlation name of a table or view that is specified in the statement.</td>
</tr>
<tr>
<td>PAGE_RANGE</td>
<td>CHAR(1) NOT NULL WITH DEFAULT</td>
<td>The table qualifies for page-range screening, so that plans scan only the partitions that are needed. Y Yes Blank N</td>
</tr>
<tr>
<td>JOIN_TYPE</td>
<td>CHAR(1) NOT NULL WITH DEFAULT</td>
<td>Type of join F, L, S, or blank</td>
</tr>
<tr>
<td>GROUP_MEMBER</td>
<td>VARCHAR(24) NOT NULL WITH DEFAULT</td>
<td>Member for EXPLAIN</td>
</tr>
<tr>
<td>IBM_SERVICE_DATA</td>
<td>VARCHAR(254) NULL WITH DEFAULT</td>
<td>IBM use only</td>
</tr>
<tr>
<td>WHEN_OPTIMIZE</td>
<td>CHAR(1) NOT NULL WITH DEFAULT</td>
<td>-BIND, RUN</td>
</tr>
<tr>
<td>QBLOCK_TYPE</td>
<td>CHAR(6) NOT NULL WITH DEFAULT</td>
<td>For each query block, an indication of the type of SQL operation performed. SELECT, INSERT, UPDATE...</td>
</tr>
<tr>
<td>BIND_TIME</td>
<td>TIMESTAMP NULL WITH DEFAULT</td>
<td>The time at which the plan or package for this statement query block was bound.</td>
</tr>
<tr>
<td>OPPTHINT</td>
<td>VARCHAR(128) NOT NULL WITH DEFAULT</td>
<td>A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path.</td>
</tr>
<tr>
<td>HINT_USED</td>
<td>VARCHAR(128) NOT NULL WITH DEFAULT</td>
<td>If DB2 used one of your optimization hints, it puts the identifier for that hint (the value in OPPTHINT) in this column.</td>
</tr>
<tr>
<td>PRIMARY_ACCESTYPE</td>
<td>CHAR(1) NOT NULL WITH DEFAULT</td>
<td>If direct row access.</td>
</tr>
<tr>
<td>PARENT_QBLOCKNO</td>
<td>SMALLINT NOT NULL WITH DEFAULT</td>
<td>A number that indicates the QBLOCKNO of the parent query block.</td>
</tr>
<tr>
<td>TABLE_TYPE</td>
<td>CHAR(1)</td>
<td>T table, W work, RB, Q, M, F, C, B...</td>
</tr>
</tbody>
</table>

--- 51 Column Format ----Version 7 --2001---
C.2 DSN_STATEMNT_TABLE

EXPLAIN estimates the cost of executing an SQL SELECT, INSERT, UPDATE, or DELETE statement. The output appears in a table called DSN_STATEMNT_TABLE. The columns of this table and their contents are listed in Table C-3. For more information about statement tables, see “Estimating a statement's cost” in the DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide, SC18-9851. The new column is TOTAL_COST, which indicates the estimated cost of the specified SQL statement.
You can use DB2 EXPLAIN to obtain information about how DB2 resolves functions. DB2 stores the information in a table called DSN_FUNCTION_TABLE. DB2 inserts a row in DSN_FUNCTION_TABLE for each function that is referenced in an SQL statement when one of the following events occurs:

- You execute the SQL EXPLAIN statement on an SQL statement that contains user-defined function invocations.
- You run a program whose plan is bound with EXPLAIN(YES), and the program executes an SQL statement that contains user-defined function invocations.

Before you use EXPLAIN to obtain information about function resolution, you need to create the DSN_FUNCTION_TABLE. The contents are listed in Table C-4.

### Table C-3  EXPLAIN enhancements in DSN_STATEMNT_TABLE

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERYNO</td>
<td>INTEGER NOT NULL WITH DEFAULT</td>
<td>A number that identifies the statement that is being explained.</td>
</tr>
<tr>
<td>APPLNAME</td>
<td>VARCHAR(24) NOT NULL WITH DEFAULT</td>
<td>The name of the application plan for the row, or blank.</td>
</tr>
<tr>
<td>PROGNAME</td>
<td>VARCHAR(128) NOT NULL WITH DEFAULT</td>
<td>The name of the program or package that contains the statement that is being explained, or blank.</td>
</tr>
<tr>
<td>COLLID</td>
<td>VARCHAR(128) NOT NULL WITH DEFAULT</td>
<td>The collection ID for the package.</td>
</tr>
<tr>
<td>GROUP_MEMBER</td>
<td>VARCHAR(24) NOT NULL WITH DEFAULT</td>
<td>The member name of the DB2 that executed EXPLAIN, or blank.</td>
</tr>
<tr>
<td>EXPLAIN_TIME</td>
<td>TIMESTAMP</td>
<td>The time at which the statement is processed.</td>
</tr>
<tr>
<td>STMT_TYPE</td>
<td>CHAR (6) NOT NULL WITH DEFAULT</td>
<td>The type of statement that is being explained. SELECT, UPDATE, INSERT, MERGE, SELUPD, DELCUR, UPDCUR or UPDATE.</td>
</tr>
<tr>
<td>COST_CATEGORY</td>
<td>CHAR (1) NOT NULL WITH DEFAULT</td>
<td>Indicates if DB2 was forced to use default values when making its estimates (B) or used statistics (A).</td>
</tr>
<tr>
<td>PROCMS</td>
<td>INTEGER NOT NULL WITH DEFAULT</td>
<td>The estimated processor cost, in milliseconds, for the SQL statement.</td>
</tr>
<tr>
<td>PROCSU</td>
<td>INTEGER NOT NULL WITH DEFAULT</td>
<td>The estimated processor cost, in service units, for the SQL statement.</td>
</tr>
<tr>
<td>REASON</td>
<td>VARCHAR(254) NOT NULL WITH DEFAULT</td>
<td>A string that indicates the reasons for putting an estimate into cost category B.</td>
</tr>
<tr>
<td>STMT_ENCODE</td>
<td>CHAR (1) NOT NULL WITH DEFAULT</td>
<td>Encoding scheme of the statement. A = ASCII E = EBCDIC U = Unicode</td>
</tr>
<tr>
<td>TOTAL_COST</td>
<td>FLOAT</td>
<td>Overall estimate of the cost.</td>
</tr>
</tbody>
</table>

### C.3 DSN_FUNCTION_TABLE

You can use DB2 EXPLAIN to obtain information about how DB2 resolves functions. DB2 stores the information in a table called DSN_FUNCTION_TABLE. DB2 inserts a row in DSN_FUNCTION_TABLE for each function that is referenced in an SQL statement when one of the following events occurs:

- You execute the SQL EXPLAIN statement on an SQL statement that contains user-defined function invocations.
- You run a program whose plan is bound with EXPLAIN(YES), and the program executes an SQL statement that contains user-defined function invocations.

Before you use EXPLAIN to obtain information about function resolution, you need to create the DSN_FUNCTION_TABLE. The contents are listed in Table C-4.
The DSN_STATEMENT_CACHE_TABLE was recently introduced in V8 by PTF UQ89372 for APAR PQ88073. With this enhancement, the new keyword ALL is added to EXPLAIN STMTCACHE, and the new explain table DSN_STATEMENT_CACHE_TABLE is created to hold the output of IFCID 316 and IFCID 318. A brief description follows, as reported in the APAR text.

There are two different sets of information that can be collected from the SQL statements in the dynamic statement cache. Specifying STMTCACHE with the STMTID or STMTTOKEN
keywords causes the traditional access path information to be written to the PLAN_TABLE for the associated SQL statement as well as a single row written to DSN_STATEMENT_CACHE_TABLE if it exists.

However, specifying STMTCACHE with the new ALL keyword causes information to be written to only DSN_STATEMENT_CACHE_TABLE. It consists of one row per SQL statement in the dynamic statement cache for which the current authorization ID is authorized to execute.

The contents of these rows show identifying information about the cache entries, as well as an accumulation of statistics reflecting the executions of the statements by all processes that have executed the statement.

This information is nearly identical to the information returned from the IFI monitor READS API for IFCIDs 0316 and 0317.

Note that the collection and reset of the statistics in these records is controlled by starting and stopping IFCID 318. For more details, see “Controlling collection of dynamic statement cache statistics with IFCID 0318” in Appendix B, “Programming for the Instrumentation Facility Interface (IFI)” of DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide, SC18-9851.

The new EXPLAIN option is:

EXPLAIN STMTCACHE ALL;

SQLCODE -20248 is issued if no statement is found in the cache for the auth ID that is used.

-20248 ATTEMPTED TO EXPLAIN A CACHED STATEMENT WITH STMTID, STMTTOKEN ID-token, OR ALL BUT THE REQUIRED EXPLAIN INFORMATION IS NOT ACCESSIBLE.

Before you can use EXPLAIN STMTCACHE ALL, Statement Cache must be turned on. You must also create the table DSN_STATEMENT_CACHE_TABLE to hold the results of EXPLAIN STMTCACHE ALL. Example C-1 shows the DDL.

Example: C-1  Creating DSN_STATEMENT_CACHE_TABLE

CREATE DATABASE DSNSTMTC;

CREATE TABLESPACE DSNSUMTS IN DSNSTMTC;

CREATE LOB TABLESPACE DSNLOBTS IN DSNSTMTC BUFFERPOOL BP32K1;

CREATE TABLE DSN_STATEMENT_CACHE_TABLE
(  STMT_ID INTEGER NOT NULL,  STMT_TOKEN VARCHAR(240) ,  COLLID VARCHAR(128) NOT NULL,  PROGRAM_NAME VARCHAR(128) NOT NULL,  INV_DROPALT CHAR(1) NOT NULL,  INV_REVOKE CHAR(1) NOT NULL,  INV_LRU CHAR(1) NOT NULL,  INV_RUNSTATS CHAR(1) NOT NULL,  CACHED_TS TIMESTAMP NOT NULL,  USERS INTEGER NOT NULL,  COPIES INTEGER NOT NULL,  LINES INTEGER NOT NULL,  PRIMAUTH VARCHAR(128) NOT NULL,
CURSQLID       VARCHAR(128) NOT NULL,
BIND_QUALIFIER VARCHAR(128) NOT NULL,
BIND_ISO       CHAR(2)      NOT NULL,
BIND_CDATA     CHAR(1)      NOT NULL,
BIND_DYNRL     CHAR(1)      NOT NULL,
BIND_DEGRE     CHAR(1)      NOT NULL,
BIND_SQLRL     CHAR(1)      NOT NULL,
BIND_CHOLD     CHAR(1)      NOT NULL,
STAT_TS        TIMESTAMP    NOT NULL,
STAT_EXEC      INTEGER      NOT NULL,
STAT_GPAG      INTEGER      NOT NULL,
STAT_SYNR      INTEGER      NOT NULL,
STAT_WIT       INTEGER      NOT NULL,
STAT_EROW      INTEGER      NOT NULL,
STAT_PROW      INTEGER      NOT NULL,
STAT_SORT      INTEGER      NOT NULL,
STAT_INDEX     INTEGER      NOT NULL,
STAT_RSCN      INTEGER      NOT NULL,
STAT_PGRP      INTEGER      NOT NULL,
STAT_ELAP      FLOAT        NOT NULL,
STAT_CPU       FLOAT        NOT NULL,
STAT_SUS_SYNIO FLOAT        NOT NULL,
STAT_SUS_LOCK  FLOAT        NOT NULL,
STAT_SUS_SMIT  FLOAT        NOT NULL,
STAT_SUS_GLCK  FLOAT        NOT NULL,
STAT_SUS_OTHR  FLOAT        NOT NULL,
STAT_SUS_OTHW  FLOAT        NOT NULL,
STAT_RIDLIMIT  INTEGER      NOT NULL,
STAT_RIDSTOR   INTEGER      NOT NULL,
EXPLAIN_TS     TIMESTAMP    NOT NULL,
SCHEMA         VARCHAR(128) NOT NULL,
STMT_TEXT      CLOB(2M)     NOT NULL,
STMT_ROWID     ROWID        NOT NULL  GENERATED ALWAYS
BIND_RA_TOT    INTEGER      NOT NULL WITH DEFAULT,
BIND_RO_TYPE   CHAR(1)      NOT NULL WITH DEFAULT )
IN DSNSTMC.DSNSUMTS CCSID EBCDIC;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX1
ON DSN_STATEMENT_CACHE_TABLE (STMT_ID ASC) ;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX2
ON DSN_STATEMENT_CACHE_TABLE (STMT_TOKEN ASC) CLUSTER;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX3
ON DSN_STATEMENT_CACHE_TABLE (EXPLAIN_TS DESC) ;

CREATE AUX TABLE DSN_STATEMENT_CACHE_AUX IN
DSNSTMC.DSNLOBTS STORES DSN_STATEMENT_CACHE_TABLE COLUMN STMT_TEXT;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_AUXINX
ON DSN_STATEMENT_CACHE_AUX;
Table C-5 describes the contents of this EXPLAIN table.

Table C-5  Contents of DSN_STATEMENT_CACHE_TABLE

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMT_ID</td>
<td>INTEGER</td>
<td>Statement ID, EDM unique token.</td>
</tr>
<tr>
<td>STMT_TOKEN</td>
<td>VARCHAR(240)</td>
<td>Statement token. User-provided identification string.</td>
</tr>
<tr>
<td>COLLID</td>
<td>VARCHAR(128) NOT NULL</td>
<td>Collection ID value is DSNDYNAMICSQLCACHE.</td>
</tr>
<tr>
<td>PROGRAM_NAME</td>
<td>VARCHAR(128) NOT NULL</td>
<td>Program name, name of package, or DBRM that performed the initial PREPARE.</td>
</tr>
<tr>
<td>INV_DROPALT</td>
<td>CHAR(1) NOT NULL</td>
<td>Invalidated by DROP/ALTER.</td>
</tr>
<tr>
<td>INV_REVOKE</td>
<td>CHAR(1) NOT NULL</td>
<td>Invalidated by REVOKE.</td>
</tr>
<tr>
<td>INV_LRU</td>
<td>CHAR(1) NOT NULL</td>
<td>Removed from cache by LRU.</td>
</tr>
<tr>
<td>INV_RUNSTATS</td>
<td>CHAR(1) NOT NULL</td>
<td>Invalidated by RUNSTATS.</td>
</tr>
<tr>
<td>CACHED_TS</td>
<td>TIMESTAMP NOT NULL</td>
<td>Timestamp when the statement was cached.</td>
</tr>
<tr>
<td>USERS</td>
<td>INTEGER NOT NULL</td>
<td>Number of current users of the statement. These users have prepared or executed the statement during their current unit of work.</td>
</tr>
<tr>
<td>COPIES</td>
<td>INTEGER NOT NULL</td>
<td>Number of copies of the statement that are owned by all threads in the system.</td>
</tr>
<tr>
<td>LINES</td>
<td>INTEGER NOT NULL</td>
<td>Precompiler line number from the initial PREPARE.</td>
</tr>
<tr>
<td>PRIMAUTH</td>
<td>VARCHAR(128) NOT NULL</td>
<td>User ID - Primary authorization ID of the user that did the initial PREPARE.</td>
</tr>
<tr>
<td>CURSQLID</td>
<td>VARCHAR(128) NOT NULL</td>
<td>CURRENT SQLID of the user that did the initial prepare.</td>
</tr>
<tr>
<td>BIND_QUALIFIER</td>
<td>VARCHAR(128) NOT NULL</td>
<td>Bind Qualifier, object qualifier for unqualified table names.</td>
</tr>
<tr>
<td>BIND_ISO</td>
<td>CHAR(2) NOT NULL</td>
<td>ISOLATION BIND option:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UR Uncommitted Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS Cursor Stability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RS Read Stability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RR Repeatable Read</td>
</tr>
<tr>
<td>BIND_CDATA</td>
<td>CHAR(1) NOT NULL</td>
<td>CURRENTDATA BIND option:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y CURRENTDATA(YES)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N CURRENTDATA(NO)</td>
</tr>
<tr>
<td>BIND_DYNRL</td>
<td>CHAR(1) NOT NULL</td>
<td>DYNAMICRULES BIND option:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B DYNAMICRULES(BIND)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R DYNAMICRULES(RUN)</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Content</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BIND_DEGRE</td>
<td>CHAR(1)</td>
<td>CURRENT DEGREE value:</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td>A  CURRENT DEGREE = ANY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  CURRENT DEGREE = 1</td>
</tr>
<tr>
<td>BIND_SQLRL</td>
<td>CHAR(1)</td>
<td>CURRENT RULES value:</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td>D  CURRENT RULES = DB2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S  CURRENT RULES = SQL</td>
</tr>
<tr>
<td>BIND_CHOLD</td>
<td>CHAR(1)</td>
<td>Cursor WITH HOLD bind option</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td>Y  Initial PREPARE was done for a cursor WITH HOLD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N  Initial PREPARE was not done for a cursor WITH HOLD</td>
</tr>
<tr>
<td>STAT_TS</td>
<td>TIMESTAMP</td>
<td>Timestamp of stats when IFCID 318 is started.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_EXEC</td>
<td>INTEGER</td>
<td>Number of executions of a statement. For a cursor statement, this is the</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td>number of OPENs.</td>
</tr>
<tr>
<td>STAT_GPAG</td>
<td>INTEGER</td>
<td>Number of getpage operations that are performed for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_SYNR</td>
<td>INTEGER</td>
<td>Number of synchronous buffer reads that are performed for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_WRIT</td>
<td>INTEGER</td>
<td>Number of buffer write operations that are performed for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_EROW</td>
<td>INTEGER</td>
<td>Number of rows that are examined for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_PROW</td>
<td>INTEGER</td>
<td>Number of rows that are processed for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_SORT</td>
<td>INTEGER</td>
<td>Number of sorts that are performed for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_INDEX</td>
<td>INTEGER</td>
<td>Number of index scans that are performed for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_RSCN</td>
<td>INTEGER</td>
<td>Number of table space scans that are performed for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_PGRP</td>
<td>INTEGER</td>
<td>Number of parallel groups that are created for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_ELAP</td>
<td>FLOAT</td>
<td>Accumulated elapsed time that is used for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_CPU</td>
<td>FLOAT</td>
<td>Accumulated CPU time that is used for a statement.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_SUS_SYNIO</td>
<td>FLOAT</td>
<td>Accumulated wait time for synchronous I/O.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_SUS_LOCK</td>
<td>FLOAT</td>
<td>Accumulated wait time for a lock and latch request.</td>
</tr>
<tr>
<td></td>
<td>NOT NULL</td>
<td></td>
</tr>
<tr>
<td>STAT_SUS_SWIT</td>
<td>FLOAT</td>
<td>Accumulated wait time for a synchronous execution unit switch.</td>
</tr>
</tbody>
</table>
Several other tables have been added for health monitor support and the Optimization Service Center support. The other tables are:

- **DSN_PREDICAT_TABLE**
- **DSN_STRUCT_TABLE**
- **DSN_PGROUP_TABLE**
- **DSN_PTASK_TABLE**
- **DSN_FILTER_TABLE**
- **DSN_DETCOST_TABLE**
- **DSN_SORT_TABLE**
- **DSN_SORTKEY_TABLE**
- **DSN_PRANGE_TABLE**
- **DSN_VIEWREF_TABLE**
- **DSN_QUERY_TABLE**
- **DSN_VIRTUAL_INDEXES**

### C.5 New tables with DB2 9 for z/OS

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAT_SUS_GLCK</td>
<td>FLOAT NOT NULL</td>
<td>Accumulated wait time for global locks.</td>
</tr>
<tr>
<td>STAT_SUS_OTHR</td>
<td>FLOAT NOT NULL</td>
<td>Accumulated wait time for read activity that is done by another thread.</td>
</tr>
<tr>
<td>STAT_SUS_OTHW</td>
<td>FLOAT NOT NULL</td>
<td>Accumulated wait time for write activity that is done by another thread.</td>
</tr>
<tr>
<td>STAT_RIDLIMT</td>
<td>INTEGER NOT NULL</td>
<td>Number of times that an RID list was not used because the number of RIDs would have exceeded one or more DB2 limits.</td>
</tr>
<tr>
<td>STAT_RIDSTOR</td>
<td>INTEGER NOT NULL</td>
<td>Number of times that a RID list was not used because not enough storage was available to hold the list of RIDs.</td>
</tr>
<tr>
<td>EXPLAIN_TS</td>
<td>TIMESTAMP NOT NULL</td>
<td>When a statement cache table is populated.</td>
</tr>
<tr>
<td>SCHEMA</td>
<td>VARCHAR(128) NOT NULL</td>
<td>CURRENT SCHEMA value.</td>
</tr>
<tr>
<td>STMT_TEXT</td>
<td>CLOB(2M) NOT NULL</td>
<td>Statement text.</td>
</tr>
<tr>
<td>STMT_ROWID</td>
<td>ROWID NOT NULL GENERATED ALWAYS</td>
<td>Statement ROWID.</td>
</tr>
<tr>
<td>BIND_RA_TOT</td>
<td>INTEGER NOT NULL WITH DEFAULT</td>
<td>Total number of REBIND commands issued because of REOPT(AUTO).</td>
</tr>
<tr>
<td>BIND_RO_TYPE</td>
<td>CHAR(1) NOT NULL WITH DEFAULT</td>
<td>REOPT option</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N  NONE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1  ONCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A  AUTO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0  No need</td>
</tr>
</tbody>
</table>
INSTEAD OF triggers test case

In this appendix, we provide details about the Data Definition Language (DDL) and accounting report for the measurements that are mentioned in 2.12, “INSTEAD OF triggers” on page 39.
D.1 INSTEAD OF trigger DDL

Example D-1 shows the DDL for table space, table, index, and view.

Example: D-1  Create table space, table, index and view

CREATE TABLESPACE TSEMPL01
   IN DBITRK02
     USING VCAT DSNC910
   SEGSIZE 32
   BUFFERPOOL BP1
   LOCKSIZE ROW
   CLOSE NO;

CREATE TABLE EMPLOYEE01(EMPTABLN   CHAR(8),
   EMPEMPLN    CHAR(8),
   EMPLOSEX    CHAR(1),
   EMPDEPTN    CHAR(3),
   EMPLEVEL    SMALLINT,
   EMPSHIFT    CHAR(1),
   EMLTEAM    DECIMAL(3),
   EMPSALRY    DECIMAL(7,2),
   EMLPROJ    DECIMAL(2),
   EMPOTIMS    DECIMAL(5),
   EMPOTIME    DECIMAL(5),
   EMPOTIMA    INTEGER,
   EMPLQUAL    CHAR(6),
   EMPLJOIN    DECIMAL(5),
   EMPLPROM    DECIMAL(5),
   EMLNAME    CHAR(22),
   EMPFILL1    CHAR(7))
   IN DBITRK02.TSEMPL01;
.............................2000 rows

CREATE TYPE 2 INDEX XEMPEM01
   ON EMPLOYEE01(EMPEMPLN)
   USING VCAT DSNC910
   BUFFERPOOL BP2
   CLOSE NO;

CREATE VIEW EMPLOYEE_VIEW (EMPEMPLN,EMPDEPTN,EMPLEVEL,
   EMLTEAM,EMPSALRY,EMLPROJ) AS
   SELECT EMPEMPLN,EMPDEPTN,EMPLEVEL,
   EMLTEAM,EMPSALRY,EMLPROJ
   FROM EMPLOYEE01 WHERE EMPLOYEE01.EMPDEPTN > '077';
.........................462 rows qualify
Example D-2 shows the DDL for the INSTEAD OF trigger.

**Example: D-2**  
INSTEAD OF trigger

```
CREATE TRIGGER EMPV_INSERT INSTEAD OF INSERT ON EMPLOYEE_VIEW
REFERENCING NEW AS NEWEMP
FOR EACH ROW MODE DB2SQL
INSERT INTO EMPLOYEE01
    VALUES ('A',NEWEMP.EMPEMPLN,'A',NEWEMP.EMPDEPTN,
    NEWEMP.EMPLEVEL,
    'A',NEWEMP.EMPLTEAM,NEWEMP.EMPSALRY,NEWEMP.EMPLPROJ,
    1,1,1,'A',1,1,'A','A');
```

Example D-3 shows the relevant PL/I program logic that is used in the tests.

**Example: D-3**  
PL/I logic

```
DCL EMPTABLE     CHAR(10) INIT('0123456789');

.............EMPEMPLN is defined as char (4char,followed by 4 numeric, so to
increment, substrings are used to increment the last numeric SUBSTR(EMPEMPLN,8,1)
picking up the numerics from EMPTABLE

EMPEMPLN = 'EMPN2000';
DO I = 2 TO 10;
    SUBSTR(EMPEMPLN,8,1) = SUBSTR(EMPTABLE,I,1);
EXEC SQL
    INSERT INTO SYSADM.EMPLOYEE_VIEW
    VALUES (:EMPEMPLN,'078',4,146,75000.00,4);
END;
```

D.2 INSTEAD OF trigger accounting

Example D-4 shows the accounting trace long.

**Example: D-4**  
Accounting Trace Long for INSTEAD of TRIGGER

```
LOCATION: DSNC910  OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V4)  PAGE: 1-9
GROUP: N/P  ACCOUNTING TRACE - LONG  REQUESTED FROM: NOT SPECIFIED
---- IDENTIFICATION -------------------------------------------------------------------------------------------------------- ----
ACCT TSTAMP: 04/18/07 14:26:30.08  PLANNAME: NINSERT2  WLM SCL: 'BLANK'  CICS NET: N/A
BEGIN TIME : 04/18/07 14:26:29.98  PROD TYP: N/P  CICS LUN: N/A
END TIME   : 04/18/07 14:26:30.08  PROD VER: N/P  LUW NET: DSNC  CICS INS: N/A
REQUESTER  : DSNC910  LUW LUN: DSNC910  MAINPACK   : NINSERT2  CORRNMBR: 'BLANK'
PRIMAUTH   : SYSADM  CORRNAME: RUNPGM  CONNTYPE: TSO  LUW SEQ:            1  TRANSACT: 'BLANK'
ORIGAUTH   : SYSADM  CONNECT : BATCH  WNAME  : 'BLANK'

MVS ACCOUNTING DATA   : 'BLANK'
ACCOUNTING TOKEN(CHAR): 'BLANK'
ACCOUNTING TOKEN(HEX) : 'BLANK'

ELAPSED TIME DISTRIBUTION            CLASS 2 TIME DISTRIBUTION
------------------------------------  ----------------------------
APPL   | 1%                     | CPU  | 1%|----------------  19%| NOTACC | 9%|----------------  72%| SUSP
DB2    | 28%|---------------- | 72%| NOTACC | 9%|---------------- | 72%| SUSP
SUSP   |---------------- | 72%| NOTACC | 9%|---------------- | 72%| SUSP

TIMES/EVENTS  APPL(CL.1)  DB2 (CL.2)  IFI (CL.5)  CLASS 3 SUSPENSIONS  ELAPSED TIME  EVENTS  HIGHLIGHTS
----------------  ------------  ----------  ----------  -------------------  -------------  ------  ---------------
ELAPSED TIME  0.097415  0.096698  N/P  LOCK/LATCH(DB2+IRLM)  0.000000  0  THREAD TYPE : ALLIED
NONNESTED  0.004863  0.004145  N/A  SYNCHRON. 1/0  0.004051  7  TERMCONDITION: NORMAL
```
**Subject:** Performance Monitoring

**Date:** 04/18/07 14:24:10.10

**Location:** DSNC910

**Action:** Omegamon XE for DB2 Performance Expert (V4)

**Page:** 1-11

<table>
<thead>
<tr>
<th>Event Description</th>
<th>N/A</th>
<th>N/A</th>
<th>N/A</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU Usage</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Task</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Disk</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Load Average</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>System Memory</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Swap</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Virtual Memory</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Additional Information:**

- **Not Accounted Time (N/A):** 0.000000
- **DB2 Ent/Exit:** 0.000000
- **Ex/Std/Ext Proc (N/A):** 0.000000
- **Dcap.Descr.:** N/A
- **DB2 Data:** N/A
- **Log Extract:** N/A

**Identification:**

- **Group:** N/P
- **Member:** N/P
- **Subsystem:** DSNC
- **Actual From:** 04/18/07 14:24:10.10

**DB2 Version:** V9

**SQL DML:**

- **Total SQL DML:** 0.000000
- **SQL DCL:** 0.000000
- **SQL DDL:** 0.000000
- **Create:** 0.000000
- **Drop:** 0.000000
- **Alter:** 0.000000
- **Locking:** 0.000000
- **Data Sharing:** 0.000000

**Miscellaneous:**

- **Comment ON:**
- **Comment OFF:**
- **INDEX USE:** 0.000000
- **PAGE LATCH:** 0.000000
- **TOTAL SUSPENSIONS:** 0.000000
- **SQL ERROR:** 0.000000

**RID List:**

- **Total:** 0.000000
- **Rowid:** 0.000000
- **Stored Proc:** 0.000000
- **Udf:** 0.000000
- **Triggers:** 0.000000

**Notes:**

- **SQL Command:**
- **SQL Statement:**
- **SQL Execution:**
- **SQL Response:**
- **SQL Transaction:**
- **SQL Execution Time:**
- **SQL Error:**

**DB2 for z/OS Performance Topics**
### Appendix D. INSTEAD OF triggers test case

#### IDENTIFICATION

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATION</td>
<td>DSNC910</td>
</tr>
<tr>
<td>DB2 VERSION</td>
<td>V9</td>
</tr>
<tr>
<td>ACCOUNTING TRACE</td>
<td>LONG</td>
</tr>
<tr>
<td>GROUP</td>
<td>N/P</td>
</tr>
<tr>
<td>MEMBER</td>
<td>N/P</td>
</tr>
<tr>
<td>SUBSYSTEM</td>
<td>DSNC</td>
</tr>
<tr>
<td>ACTUAL FROM</td>
<td>04/18/07 14:24:10.10</td>
</tr>
<tr>
<td>REQUESTED FROM</td>
<td>NOT SPECIFIED</td>
</tr>
</tbody>
</table>

#### Accounting Trace - Long

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATION</td>
<td>DSNC910</td>
</tr>
<tr>
<td>DB2 VERSION</td>
<td>V9</td>
</tr>
<tr>
<td>ACCOUNTING TRACE</td>
<td>LONG</td>
</tr>
<tr>
<td>GROUP</td>
<td>N/P</td>
</tr>
<tr>
<td>MEMBER</td>
<td>N/P</td>
</tr>
<tr>
<td>SUBSYSTEM</td>
<td>DSNC</td>
</tr>
<tr>
<td>ACTUAL FROM</td>
<td>04/18/07 14:24:10.10</td>
</tr>
<tr>
<td>REQUESTED FROM</td>
<td>NOT SPECIFIED</td>
</tr>
</tbody>
</table>

#### SELECT STATEMENTS

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATION</td>
<td>DSNC910</td>
</tr>
<tr>
<td>DB2 VERSION</td>
<td>V9</td>
</tr>
<tr>
<td>ACCOUNTING TRACE</td>
<td>LONG</td>
</tr>
<tr>
<td>GROUP</td>
<td>N/P</td>
</tr>
<tr>
<td>MEMBER</td>
<td>N/P</td>
</tr>
<tr>
<td>SUBSYSTEM</td>
<td>DSNC</td>
</tr>
<tr>
<td>ACTUAL FROM</td>
<td>04/18/07 14:24:10.10</td>
</tr>
<tr>
<td>REQUESTED FROM</td>
<td>NOT SPECIFIED</td>
</tr>
</tbody>
</table>

#### Dynamic SQL Statement

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCATION</td>
<td>DSNC910</td>
</tr>
<tr>
<td>DB2 VERSION</td>
<td>V9</td>
</tr>
<tr>
<td>ACCOUNTING TRACE</td>
<td>LONG</td>
</tr>
<tr>
<td>GROUP</td>
<td>N/P</td>
</tr>
<tr>
<td>MEMBER</td>
<td>N/P</td>
</tr>
<tr>
<td>SUBSYSTEM</td>
<td>DSNC</td>
</tr>
<tr>
<td>ACTUAL FROM</td>
<td>04/18/07 14:24:10.10</td>
</tr>
<tr>
<td>REQUESTED FROM</td>
<td>NOT SPECIFIED</td>
</tr>
</tbody>
</table>
### DB2 9 for z/OS Performance Topics

<table>
<thead>
<tr>
<th>BPOOL ACTIVITY</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BP2</strong></td>
<td></td>
</tr>
<tr>
<td>BPOOL HIT RATIO (%)</td>
<td>71</td>
</tr>
<tr>
<td>GETPAGES</td>
<td>14</td>
</tr>
<tr>
<td>BUFFER UPDATES</td>
<td>9</td>
</tr>
<tr>
<td>SYNCHRONOUS WRITE</td>
<td>0</td>
</tr>
<tr>
<td>SYNCHRONOUS READ</td>
<td>4</td>
</tr>
<tr>
<td>SEQ. PREFETCH REQS</td>
<td>0</td>
</tr>
<tr>
<td>LIST PREFETCH REQS</td>
<td>0</td>
</tr>
<tr>
<td>DYN. PREFETCH REQS</td>
<td>0</td>
</tr>
<tr>
<td>PAGES READ ASYNCHR.</td>
<td>0</td>
</tr>
</tbody>
</table>

| **BP10**        |       |
| BPOOL HIT RATIO (%) | 100  |
| GETPAGES         | 36    |
| BUFFER UPDATES   | 36    |
| SYNCHRONOUS WRITE | 0     |
| SYNCHRONOUS READ | 0     |
| SEQ. PREFETCH REQS | 0  |
| LIST PREFETCH REQS | 0     |
| DYN. PREFETCH REQS | 0     |
| PAGES READ ASYNCHR. | 0     |

### BP8K

| **BP8K**         |       |
| BPOOL HIT RATIO (%) | 100  |
| GETPAGES         | 3     |
| BUFFER UPDATES   | 0     |
| SYNCHRONOUS WRITE | 0     |
| SYNCHRONOUS READ | 0     |
| SEQ. PREFETCH REQS | 0  |
| LIST PREFETCH REQS | 0     |
| DYN. PREFETCH REQS | 0     |
| PAGES READ ASYNCHR. | 0     |

### TOT4K

| **TOT4K**         |       |
| BPOOL HIT RATIO (%) | 88    |
| GETPAGES         | 61    |
| BUFFER UPDATES   | 55    |
| SYNCHRONOUS WRITE | 0     |
| SYNCHRONOUS READ | 7     |
| SEQ. PREFETCH REQS | 0  |
| LIST PREFETCH REQS | 0     |
| DYN. PREFETCH REQS | 0     |
| PAGES READ ASYNCHR. | 0     |

### TOTAL

| **TOTAL**         |       |
| BPOOL HIT RATIO (%) | 89    |
| GETPAGES         | 64    |
| BUFFER UPDATES   | 55    |
| SYNCHRONOUS WRITE | 0     |
| SYNCHRONOUS READ | 7     |
| SEQ. PREFETCH REQS | 0  |
| LIST PREFETCH REQS | 0     |
| DYN. PREFETCH REQS | 0     |
| PAGES READ ASYNCHR. | 0     |

### PROGRAM NAME

<table>
<thead>
<tr>
<th>PROGRAM NAME</th>
<th>CLASS 7 CONSUMERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NINSERT2</td>
<td>4%</td>
</tr>
<tr>
<td>EMPV_I#0ERT</td>
<td>96%</td>
</tr>
</tbody>
</table>

---

**LOCATION**: DSNC910  
**ACCOUNTING TRACE - LONG**  
**REQUESTED FROM**: NOT SPECIFIED  
**TO**: NOT SPECIFIED  
**SUBSYSTEM**: DSNC  
**DB2 VERSION**: V9

---

**IDENTIFICATION**

<table>
<thead>
<tr>
<th>ACCT TSTAMP</th>
<th>PLANNAME</th>
<th>WLM SCL</th>
<th>CICS NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/18/07 14:26:30.08</td>
<td>NINSERT2</td>
<td>'BLANK'</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BEGIN TIME</th>
<th>PROD TYP</th>
<th>CICS LUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/18/07 14:26:29.98</td>
<td>N/P</td>
<td>N/A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>END TIME</th>
<th>PROD VER</th>
<th>LUW NET</th>
<th>CICS INS</th>
<th>LUW LUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/18/07 14:26:30.08</td>
<td>N/P</td>
<td>DSNC</td>
<td>N/A</td>
<td>DSNC910</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REQUESTER</th>
<th>CORRNAME</th>
<th>LUW SEQ</th>
<th>LUW INS</th>
<th>ENDUSER</th>
<th>WSNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNC910</td>
<td>RUNPGM</td>
<td>I</td>
<td>D077ADABC09</td>
<td>'BLANK'</td>
<td>'BLANK'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRIMAUTH</th>
<th>CONNTYPE</th>
<th>LUN SEQ</th>
<th>LUW LUN</th>
<th>LUW NET</th>
<th>LUW LUN</th>
<th>LUW LUN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSADM</td>
<td>TSO</td>
<td>C077ADABC09</td>
<td>DSNC</td>
<td>DSNC</td>
<td>DSNC910</td>
<td></td>
</tr>
</tbody>
</table>

---

**PROGRAM NAME**

<table>
<thead>
<tr>
<th>PROGRAM NAME</th>
<th>CLASS 7 CONSUMERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NINSERT2</td>
<td>4%</td>
</tr>
<tr>
<td>EMPV_I#0ERT</td>
<td>96%</td>
</tr>
</tbody>
</table>

---

348  
**DB2 9 for z/OS Performance Topics**
### Appendix D. INSTEAD OF triggers test case

349
XML documents

In this appendix, we provide the extended details about some of the test cases that were carried out in order to evaluate the performance of XML in DB2 V9.

In E.1, “XML document decomposition” on page 352, we present the details of the documents that were used for the decomposition case that is described in “Test case 4: INSERT using decomposition performance” on page 73:

- XML document
- XML schema
- DDL for decomposition

In E.2, “XML index exploitation” on page 368, we present the details for the index exploitation case that is described in 3.3.4, “Index exploitation” on page 77:

- XML index definitions
- EXPLAIN output

Notice that all Universal Financial Industry (UNIFI) schemas and XML that are used for “Test case 3: INSERT with validation performance” on page 71, are available on the International Organization for Standardization (ISO) Web site at the following address:

http://www.iso20022.org/index.cfm?item_id=59950
E.1 XML document decomposition

In this section, we list the XML document, XML schema, and DDL for decomposition that were used in "Test case 4: INSERT using decomposition performance" on page 73.

E.1.1 XML document

Example E-1 shows the XML document that was used for the decomposition performance analysis.

Example: E-1 XML document

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<msg dbName="SAMPLE" version="1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="mqcap.xsd">
  <trans cmitLSN="0000:0000:0000:0234:7432" cmitTime="2005-06-15T21:30:59.000002" isLast="1" segmentNum="1">
    <updateRow srcName="EMPLOYEE" srcOwner="ADMINISTRATOR" subName="EMPLOYEE0001">
      <col name="BIRTHDATE">
        <date>
          <afterVal>1933-08-24</afterVal>
        </date>
      </col>
      <col name="BONUS">
        <decimal>
          <beforeVal>1000.0</beforeVal>
          <afterVal>1500.0</afterVal>
        </decimal>
      </col>
      <col name="COMM">
        <decimal>
          <afterVal>4220.0</afterVal>
        </decimal>
      </col>
      <col name="EDLEVEL">
        <smallint>
          <afterVal>18</afterVal>
        </smallint>
      </col>
      <col name="EMPNO">
        <char>
          <afterVal>000010</afterVal>
        </char>
      </col>
      <col name="FIRSTNAME">
        <varchar>
          <afterVal>CHRISTINE</afterVal>
        </varchar>
      </col>
      <col name="HIREDATE">
        <date>
          <afterVal>1965-01-01</afterVal>
        </date>
      </col>
      <col name="JOB">
        <char>
          <afterVal>PRES</afterVal>
        </char>
      </col>
      <col name="LASTNAME">
        <varchar>
          <afterVal>HAAS</afterVal>
        </varchar>
      </col>
    </updateRow>
  </trans>
</msg>
```
<updateRow srcName="EMPLOYEE" srcOwner="ADMINISTRATOR" subName="EMPLOYEE0001">
<col name="BIRTHDATE">
  <date>
    <afterVal>1933-08-24</afterVal>
  </date>
</col>
<col name="BONUS">
  <decimal>
    <beforeVal>1500.0</beforeVal>
    <afterVal>1000.0</afterVal>
  </decimal>
</col>
<col name="COMM">
  <decimal>
    <afterVal>4220.0</afterVal>
  </decimal>
</col>
<col name="EDLEVEL">
  <smallint>
    <afterVal>18</afterVal>
  </smallint>
</col>
<col name="EMPNO">
  <char>
    <afterVal>000010</afterVal>
  </char>
</col>
<col name="FIRSTNME">
  <varchar>
    <afterVal>CHRISTINE</afterVal>
  </varchar>
</col>
<col name="HIREDATE">
  <date>
    <afterVal>1965-01-01</afterVal>
  </date>
</col>
<col name="JOB">
  <char>

</col>
</updateRow>
E.1.2 XML schema

Example E-2 shows the XML schema for the decomposition performance analysis.

Example: E-2 XML schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>xdb</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
    <xs:annotation>
      <xs:documentation xml:lang="en">
        XML Schema of messages sent by Q Capture to a subscriber.
      </xs:documentation>
    </xs:annotation>
    <xs:annotation>
      <xs:documentation xml:lang="en">
        XML Schema of messages sent by Q Capture to a subscriber.
      </xs:documentation>
    </xs:annotation>
  </xs:annotation>
</xs:schema>
```
<!-- Message type definition -->
<!-- TEST can rowSet be used as Attribute and rowSetMapping as Element at the same time-->
<xsd:element name="msg" type="msgType"/>

<xsd:complexType name="msgType">
  <xsd:choice>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
    <xsd:element name="trans" type="transType"/>
  </xsd:choice>
  <xsd:attribute name="version" type="xs:string" use="required"/>
  <xsd:attribute name="dbName" type="xs:string" use="required"/>
</xsd:complexType>

<!-- Transaction type definition -->
<xsd:complexType name="transType">
  <xsd:choice maxOccurs="unbounded">
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
    <xsd:element name="insertRow" type="singleValRowType"/>
  </xsd:choice>
</xsd:complexType>
<xs:complexType name="singleValRowType">
  <xs:choice>
    <xs:element name="deleteRow" type="singleValRowType"/>
    <xs:element name="updateRow" type="updateRowType"/>
  </xs:choice>
</xs:complexType>

<xs:element name="deleteRow" type="singleValRowType">
  <xs:complexType>
    <xs:choice>
      <xs:element name="deleteRow" type="singleValRowType"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="updateRow" type="updateRowType">
  <xs:complexType>
    <xs:choice>
      <xs:element name="deleteRow" type="singleValRowType"/>
      <xs:element name="updateRow" type="updateRowType"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<!-- the error on registration is because &lt; is used in the attribute annotation rowSet-->
<xs:attribute name="isLast" type="xs:boolean" use="required"/>
<xs:attribute name="segmentNum" type="xs:positiveInteger" use="required"/>
<xs:attribute name="cmitLSN" type="xs:string" use="required"/>
<xs:attribute name="cmitTime" type="xs:dateTime" use="required"/>
<xs:attribute name="authID" type="xs:string"/>
<xs:attribute name="correlationID" type="xs:string"/>
<xs:attribute name="planName" type="xs:string"/>
</xs:complexType>

<!-- LOB type definition -->
<xs:complexType name="lobType">
  <xs:choice>
    <xs:element name="blob" nillable="true">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="blob"/>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>
<xs:complexType>
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="col" type="singleValColType"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attributeGroup ref="opAttrGroup"/>
</xs:complexType>

<xs:complexType name="updateRowType">
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="col" type="updateValColType"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attributeGroup ref="opAttrGroup"/>
</xs:complexType>

<!-- Column types and their common attribute group definition -->

<!-- Row types and their common attribute group definition -->

<xs:complexType name="singleValRowType">
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="col" type="singleValColType"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attributeGroup ref="opAttrGroup"/>
</xs:complexType>

<xs:complexType name="rowOpType">
  <xs:choice>
    <xs:element name="insertRow" type="singleValRowType"/>
    <xs:element name="deleteRow" type="singleValRowType"/>
    <xs:element name="updateRow" type="updateRowType"/>
  </xs:choice>
  <xs:attribute name="isLast" type="xs:boolean" use="required"/>
  <xs:attribute name="cmitLSN" type="xs:string" use="required"/>
  <xs:attribute name="cmitTime" type="xs:dateTime" use="required"/>
  <xs:attribute name="authID" type="xs:string"/>
  <xs:attribute name="correlationID" type="xs:string"/>
  <xs:attribute name="planName" type="xs:string"/>
</xs:complexType>

<!-- Row operation type definition -->

<!-- Column types and their common attribute group definition -->

<xs:complexType name="updateValColType">
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attributeGroup ref="opAttrGroup"/>
</xs:complexType>

<xs:complexType name="singleValColType">
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attributeGroup ref="opAttrGroup"/>
</xs:complexType>
<xs:complexType name="singleValColType">
    <xs:choice>
        <xs:element name="smallint" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="smallint"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="integer" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="integer"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="bigint" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="bigint"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="float" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="float"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="real" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="real"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="double" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="double"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="decimal" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="decimal"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="date" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="date"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="time" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="time"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="timestamp" nillable="true">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="timestamp"/>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
    </xs:choice>
</xs:complexType>
<xs:complexType>
  <xs:simpleContent>
    <xs:extension base="timestamp"/>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="char" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="char"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="varchar" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="varchar"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="longvarchar" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="longvarchar"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="bitchar" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="bitchar"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="bitvarchar" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="bitvarchar"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="bitlongvarchar" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="bitlongvarchar"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="graphic" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="graphic"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="vargraphic" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="vargraphic"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="longvargraphic" nillable="true">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="longvargraphic"/>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
</xs:complexType>
</xs:element>
<xs:element name="real">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="real" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="real" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="double">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="double" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="double" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="decimal">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="decimal" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="decimal" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="date">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="date" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="date" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="time">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="time" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="time" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="timestamp">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="timestamp" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="timestamp" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="char">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="char" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="char" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="beforeVal" type="char" nillable="true"/>
</xs:sequence>
<xs:element name="afterVal" type="char" nillable="true"/>
</xs:complexType>
</xs:element>
<xs:element name="varchar">
<xs:complexType>
<xs:sequence>
<xs:sequence minOccurs="0">
<xs:element name="beforeVal" type="varchar" nillable="true"/>
</xs:sequence>
<xs:element name="afterVal" type="varchar" nillable="true"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="varchar">
<xs:complexType>
<xs:sequence>
<xs:sequence minOccurs="0">
<xs:element name="beforeVal" type="varchar" nillable="true"/>
</xs:sequence>
<xs:element name="afterVal" type="varchar" nillable="true"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="longvarchar">
<xs:complexType>
<xs:sequence>
<xs:sequence minOccurs="0">
<xs:element name="beforeVal" type="longvarchar" nillable="true"/>
</xs:sequence>
<xs:element name="afterVal" type="longvarchar" nillable="true"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="bitchar">
<xs:complexType>
<xs:sequence>
<xs:sequence minOccurs="0">
<xs:element name="beforeVal" type="bitchar" nillable="true"/>
</xs:sequence>
<xs:element name="afterVal" type="bitchar" nillable="true"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="bitvarchar">
<xs:complexType>
<xs:sequence>
<xs:sequence minOccurs="0">
<xs:element name="beforeVal" type="bitvarchar" nillable="true"/>
</xs:sequence>
<xs:element name="afterVal" type="bitvarchar" nillable="true"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="bitlongvarchar">
<xs:complexType>
<xs:sequence>
<xs:sequence minOccurs="0">
<xs:element name="beforeVal" type="bitlongvarchar" nillable="true"/>
</xs:sequence>
<xs:element name="afterVal" type="bitlongvarchar" nillable="true"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="graphic">
<xs:complexType>
<xs:sequence>
<xs:sequence minOccurs="0">
<xs:element name="beforeVal" type="graphic" nillable="true"/>
</xs:sequence>
<xs:element name="afterVal" type="graphic" nillable="true"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="vargraphic">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="vargraphic" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="vargraphic" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="longvargraphic">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="longvargraphic" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="longvargraphic" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="rowid">
  <xs:complexType>
    <xs:sequence>
      <xs:sequence minOccurs="0">
        <xs:element name="beforeVal" type="rowid" nillable="true"/>
      </xs:sequence>
      <xs:element name="afterVal" type="rowid" nillable="true"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="blob">
  <xs:complexType>
  </xs:complexType>
</xs:element>

<xs:element name="clob">
  <xs:complexType>
  </xs:complexType>
</xs:element>

<xs:element name="dbclob">
  <xs:complexType>
  </xs:complexType>
</xs:element>

<xs:choice>
  <xs:attributeGroup ref="colAttrGroup"/>
</xs:choice>

<!-- Attribute group definition -->

<xs:attributeGroup name="commonAttrGroup">
  <xs:attribute name="subName" type="xs:string" use="required"/>
  <xs:attribute name="srcOwner" type="xs:string" use="required"/>
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:rowSetMapping>
        <db2-xdb:rowSet>cs_trans</db2-xdb:rowSet>
        <db2-xdb:column>srcOwner</db2-xdb:column>
      </db2-xdb:rowSetMapping>
    </xs:appinfo>
  </xs:annotation>
</xs:attributeGroup>
<db2-xdb:column>srcName</db2-xdb:column>
</db2-xdb:rowSetMapping>
</xs:appinfo>
</xs:annotation>
</xs:attribute>
</xs:attributeGroup>

<xs:attributeGroup name="colAttrGroup">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="isKey" type="xs:boolean" default="false"/>
</xs:attributeGroup>

<xs:attributeGroup name="opAttrGroup">
  <xs:attribute name="rowNum" type="xs:positiveInteger"/>
  <xs:attribute name="hasLOBCols" type="xs:boolean" default="false"/>
</xs:attributeGroup>

<!--[CDATA[
<!-- Subscription deactivated type definition -->
<xs:complexType name="subDeactivatedType">
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attribute name="stateInfo" type="xs:string" use="required"/>
</xs:complexType>

<!--[CDATA[
<!-- Load done received type definition -->
<xs:complexType name="loadDoneRcvdType">
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attribute name="stateInfo" type="xs:string" use="required"/>
</xs:complexType>

<!--[CDATA[
<!-- Heartbeat type definition -->
<xs:complexType name="heartbeatType">
  <xs:attribute name="sendQName" type="xs:string" use="required"/>
  <xs:attribute name="lastCmitTime" type="xs:dateTime"/>
</xs:complexType>

<!--[CDATA[
<!-- Error Report type definition -->
<xs:complexType name="errorRptType">
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attribute name="errorMsg" type="xs:string" use="required"/>
</xs:complexType>

<!--[CDATA[
<!-- Schema type definition -->
<xs:complexType name="subSchemaType">
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="col" type="colSchemaType"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttrGroup"/>
  <xs:attribute name="sendQName" type="xs:string" use="required"/>
  <xs:attribute name="allChangedRows" type="xs:boolean" default="false"/>
  <xs:attribute name="beforeValues" type="xs:boolean" default="false"/>
  <xs:attribute name="changedColsOnly" type="xs:boolean" default="true"/>
  <xs:attribute name="hasLoadPhase" type="loadPhaseEnumType" default="none"/>
  <xs:attribute name="dbServerType" type="dbServerTypeEnumType" use="required"/>
  <xs:attribute name="dbRelease" type="xs:string" use="required"/>
  <xs:attribute name="dbInstance" type="xs:string" use="required"/>
  <xs:attribute name="capRelease" type="xs:string" use="required"/>
</xs:complexType>

364 DB2 9 for z/OS Performance Topics
<xs:complexType>

<!-- Add column definition -->
<xs:complexType name="addColumnType">
    <xs:sequence maxOccurs="1">
        <xs:element name="col" type="colSchemaType"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonAttrGroup"/>
</xs:complexType>

<!-- Load phase enumeration type definition -->
<xs:simpleType name="loadPhaseEnumType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="none"/>
        <xs:enumeration value="external"/>
    </xs:restriction>
</xs:simpleType>

<!-- DB2 server type enumeration type definition -->
<xs:simpleType name="dbServerTypeEnumType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="QDB2"/>
        <xs:enumeration value="QDB2/6000"/>
        <xs:enumeration value="QDB2/HPUX"/>
        <xs:enumeration value="QDB2/NT"/>
        <xs:enumeration value="QDB2/SUN"/>
        <xs:enumeration value="QDB2/LINUX"/>
        <xs:enumeration value="QDB2/Windows"/>
    </xs:restriction>
</xs:simpleType>

<!-- Column schema type definition -->
<xs:complexType name="colSchemaType">
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="type" type="dataTypeEnumType" use="required"/>
    <xs:attribute name="isKey" type="xs:boolean" default="false"/>
    <xs:attribute name="len" type="xs:unsignedInt"/>
    <xs:attribute name="precision" type="xs:unsignedShort"/>
    <xs:attribute name="scale" type="xs:unsignedShort"/>
    <xs:attribute name="codepage" type="xs:unsignedInt" default="0"/>
</xs:complexType>

<!-- Data type enumeration type definition -->
<xs:simpleType name="dataTypeEnumType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="smallint"/>
        <xs:enumeration value="integer"/>
        <xs:enumeration value="bigint"/>
        <xs:enumeration value="float"/>
        <xs:enumeration value="real"/>
        <xs:enumeration value="double"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="smallint">
    <xs:restriction base="xs:short">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="integer">
    <xs:restriction base="xs:integer">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="bigint">
    <xs:restriction base="xs:long">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="float">
    <xs:restriction base="xs:float">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="real">
    <xs:restriction base="xs:float">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="double">
    <xs:restriction base="xs:double">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="decimal">
    <xs:restriction base="xs:decimal">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="date">
    <xs:restriction base="xs:date">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="time">
    <xs:restriction base="xs:time">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="rowid">
    <xs:restriction base="xs:rowid">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="blob">
    <xs:restriction base="xs:blob">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="clob">
    <xs:restriction base="xs:clob">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="dbclob">
    <xs:restriction base="xs:dbclob">
        <!-- Data type definitions -->
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="timestamp">
   <xs:restriction base="xs:dateTime">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="char">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="varchar">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="longvarchar">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="bitchar">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="bitvarchar">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="bitlongvarchar">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="graphic">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="vargraphic">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="longvargraphic">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="rowid">
   <xs:restriction base="xs:hexBinary">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="blob">
   <xs:restriction base="xs:hexBinary">
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="clob">
   <xs:restriction base="xs:string">
   </xs:restriction>
</xs:simpleType>
E.1.3 DDL for decomposition

Example E-3 shows the DDL that was used for the INSERT performance using the decomposition method.

Example: E-3    DDL for the INSERT test case

<table>
<thead>
<tr>
<th>SQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>drop table &quot;xdb&quot;.&quot;cs&quot;;</td>
</tr>
<tr>
<td>commit;</td>
</tr>
<tr>
<td>create table &quot;xdb&quot;.&quot;cs&quot; (&quot;DBName&quot; VARCHAR(10) NOT NULL,</td>
</tr>
<tr>
<td>&quot;msg_head&quot; VARCHAR(100)) in XMLTEST.XMLTEST3;</td>
</tr>
<tr>
<td>drop table &quot;xdb&quot;.&quot;cs_trans&quot;;</td>
</tr>
<tr>
<td>commit;</td>
</tr>
<tr>
<td>create table &quot;xdb&quot;.&quot;cs_trans&quot; (&quot;DBName&quot; VARCHAR(20) NOT NULL,</td>
</tr>
<tr>
<td>&quot;cmitLSN&quot; VARCHAR(50), &quot;cmitTime&quot; VARCHAR(50), &quot;srcName&quot; VARCHAR(20),</td>
</tr>
<tr>
<td>&quot;srcOwner&quot; VARCHAR(20), &quot;cs_trans&quot; LONG VARCHAR) in XMLTEST.XMLTEST3;</td>
</tr>
</tbody>
</table>

E.2 XML index exploitation

In this section, we list the index definitions and EXPLAIN output for 3.3.4, “Index exploitation” on page 77.

E.2.1 XML index definitions

The DDL listed in Example E-4 is used for the XML indexes that were defined for the index exploitation performance analysis.

Example: E-4    DDL for index definition

<table>
<thead>
<tr>
<th>SQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE INDEX V2CUSTKEY</td>
</tr>
<tr>
<td>ON TPCD02(C.XML)</td>
</tr>
<tr>
<td>GENERATE KEY USING XMPPATTERN</td>
</tr>
<tr>
<td>'/customer/customer_xml/@CUSTKEY' AS SQL VARCHAR(8)</td>
</tr>
<tr>
<td>BUFFERPOOL BP3</td>
</tr>
<tr>
<td>DEFER YES</td>
</tr>
<tr>
<td>CLOSE NO;</td>
</tr>
<tr>
<td>CREATE INDEX V2ORDERKEY</td>
</tr>
<tr>
<td>ON TPCD02(C.XML)</td>
</tr>
<tr>
<td>GENERATE KEY USING XMPPATTERN</td>
</tr>
<tr>
<td>'/customer/customer_xml/order/@orderkey' AS SQL DECIMAL</td>
</tr>
<tr>
<td>BUFFERPOOL BP3</td>
</tr>
<tr>
<td>defer yes</td>
</tr>
<tr>
<td>CLOSE NO;</td>
</tr>
</tbody>
</table>
E.2.2 EXPLAIN output

The EXPLAIN output in Figure E-1 shows the access path that was selected for the five queries in the index exploitation performance analysis.

Figure E-1  EXPLAIN output
## Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>autonomic computing</td>
</tr>
<tr>
<td>ACS</td>
<td>automatic class selection</td>
</tr>
<tr>
<td>AIX®</td>
<td>Advanced Interactive eXecutive from IBM</td>
</tr>
<tr>
<td>AMI</td>
<td>Application Messaging Interface</td>
</tr>
<tr>
<td>AMP</td>
<td>adaptive multi-streaming prefetching</td>
</tr>
<tr>
<td>APAR</td>
<td>authorized program analysis report</td>
</tr>
<tr>
<td>API</td>
<td>application programming interface</td>
</tr>
<tr>
<td>AR</td>
<td>application requester</td>
</tr>
<tr>
<td>AS</td>
<td>Application Server</td>
</tr>
<tr>
<td>ASCII</td>
<td>American National Standard Code for Information Interchange</td>
</tr>
<tr>
<td>AUXW</td>
<td>auxiliary warning</td>
</tr>
<tr>
<td>B2B</td>
<td>business-to-business</td>
</tr>
<tr>
<td>B2C</td>
<td>business-to-consumer</td>
</tr>
<tr>
<td>BCDS</td>
<td>DFSMS/hsm backup control data set</td>
</tr>
<tr>
<td>BCRS</td>
<td>business continuity recovery services</td>
</tr>
<tr>
<td>BDAM</td>
<td>Basic Direct Access Method</td>
</tr>
<tr>
<td>BI</td>
<td>Business Intelligence</td>
</tr>
<tr>
<td>BLOB</td>
<td>binary strings</td>
</tr>
<tr>
<td>BSAM</td>
<td>Basic Sequential Access Method</td>
</tr>
<tr>
<td>BSDS</td>
<td>bootstrap data set</td>
</tr>
<tr>
<td>BW</td>
<td>Business Warehouse</td>
</tr>
<tr>
<td>CAF</td>
<td>call attachment facility</td>
</tr>
<tr>
<td>CDC</td>
<td>change data capture</td>
</tr>
<tr>
<td>CHKP</td>
<td>CHECK pending</td>
</tr>
<tr>
<td>CI</td>
<td>control interval</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
</tr>
<tr>
<td>CLI</td>
<td>call-level interface</td>
</tr>
<tr>
<td>CLOB</td>
<td>character large object</td>
</tr>
<tr>
<td>CM</td>
<td>conversion mode</td>
</tr>
<tr>
<td>CP</td>
<td>central process</td>
</tr>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>CRM</td>
<td>customer relationship management</td>
</tr>
<tr>
<td>CST</td>
<td>Consolidated Service Test</td>
</tr>
<tr>
<td>CT</td>
<td>cursor table</td>
</tr>
<tr>
<td>CVS</td>
<td>Concurrent Versions System</td>
</tr>
<tr>
<td>DASD</td>
<td>direct access storage device</td>
</tr>
<tr>
<td>DBCS</td>
<td>double byte character set</td>
</tr>
<tr>
<td>DBD</td>
<td>database descriptor</td>
</tr>
<tr>
<td>DBMS</td>
<td>database management system</td>
</tr>
<tr>
<td>DDF</td>
<td>distributed data facility</td>
</tr>
<tr>
<td>DDL</td>
<td>Data Definition Language</td>
</tr>
<tr>
<td>DECFLOAT</td>
<td>decimal floating point</td>
</tr>
<tr>
<td>DFSMS</td>
<td>Data Facility Storage Management Subsystem</td>
</tr>
<tr>
<td>DGTT</td>
<td>declared global temporary table</td>
</tr>
<tr>
<td>DPSI</td>
<td>data-partitioned secondary index</td>
</tr>
<tr>
<td>DRDA</td>
<td>Distributed Relational Database Architecture</td>
</tr>
<tr>
<td>DSN</td>
<td>default subsystem name</td>
</tr>
<tr>
<td>ECSA</td>
<td>extended common storage area</td>
</tr>
<tr>
<td>EDM</td>
<td>environmental descriptor manager</td>
</tr>
<tr>
<td>ENFM</td>
<td>enabling-new-function mode</td>
</tr>
<tr>
<td>ERP</td>
<td>enterprise resource planning</td>
</tr>
<tr>
<td>ESS</td>
<td>Enterprise Storage Server</td>
</tr>
<tr>
<td>ETR</td>
<td>external throughput rate</td>
</tr>
<tr>
<td>FIFO</td>
<td>first in, first out</td>
</tr>
<tr>
<td>FMID</td>
<td>function modification identifier</td>
</tr>
<tr>
<td>GBP</td>
<td>group buffer pool</td>
</tr>
<tr>
<td>GRECP</td>
<td>group buffer pool</td>
</tr>
<tr>
<td>HSM</td>
<td>hierarchical storage management</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICF</td>
<td>IBM Internal Coupling Facility</td>
</tr>
<tr>
<td>IDE</td>
<td>integrated development environment</td>
</tr>
<tr>
<td>IFCID</td>
<td>instrumentation facility component identifier</td>
</tr>
<tr>
<td>IFI</td>
<td>Instrumentation Facility Interface</td>
</tr>
<tr>
<td>IFL</td>
<td>Integrated Facility for Linux</td>
</tr>
<tr>
<td>IOSQ</td>
<td>I/O subsystem queue</td>
</tr>
<tr>
<td>IRLM</td>
<td>internal resource lock manager</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ITR</td>
<td>internal throughput rate</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
</tr>
<tr>
<td>IVP</td>
<td>Installation verification procedure</td>
</tr>
</tbody>
</table>
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks" on page 375. Note that some of the documents referenced here may be available in softcopy only.

- IBM zEnterprise System Technical Guide, SG24-7833
- IBM zEnterprise BladeCenter Extension Model 001, REDP-4668
- DB2 9 for z/OS: Resource Serialization and Concurrency Control, SG24-4725-01
- DB2 9 for z/OS: Buffer Pool Monitoring and Tuning, REDP-4604-00
- DB2 9 for z/OS Distributed Functions, SG24-6952-01
- Ready to Access DB2 for z/OS Data on Solid-State Drives, REDP-4537-00
- 50 TB Data Warehouse on System z, SG24-7674-00
- Leveraging Cognos for Linux on z, SG24-7812
- Best Practices for SAP BI using DB2 9 for z/OS, SG24-6489-01
- Enhancing SAP by Using DB2 9 for z/OS, SG24-7239-00
- Securing and Auditing Data on DB2 for z/OS, SG24-7720-00
- IBM Data Studio V2.1: Getting Started with Web Services on DB2 for z/OS, REDP-4510-00
- DB2 9 for z/OS: Packages Revisited, SG24-7688
- DB2 9 for z/OS: Deploying SOA Solutions, SG24-7663
- DB2 9 for z/OS: Backup and Recovery I/O Related Performance Considerations, REDP-4452
- Enterprise Data Warehousing with DB2 9 for z/OS, SG24-7637
- DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond, SG24-7604
- DB2 9 for z/OS Technical Overview, SG24-7330
- DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More, SG24-6079
- DB2 UDB for z/OS Version 8 Performance Topics, SG24-6465
- Disaster Recovery with DB2 UDB for z/OS, SG24-6370
- DB2 9 for z/OS Data Sharing: Distributed Load Balancing and Fault Tolerant Configuration, REDP-4449
- DB2 for z/OS: Considerations on Small and Large Packages, REDP-4424
- Index Compression with DB2 9 for z/OS, REDP-4345
- Disk Storage Access with DB2 for z/OS, REDP-4187
How does the MIDAW Facility Improve the Performance of FICON Channels Using DB2 and other workloads?, REDP-4201

LOBs with DB2 for z/OS: Stronger and Faster, SG24-7270

DB2 for z/OS: Data Sharing in a Nutshell, SG24-7322

Powering SOA with IBM Data Servers, SG24-7259

Securing DB2 and Implementing MLS on z/OS, SG24-6480

DB2 for MVS/ESA V4 Data Sharing Performance Topics, SG24-4611

IBM System z10 Enterprise Class Technical Introduction, SG24-7515

IBM System z10 Enterprise Class Technical Guide, SG24-7516

IBM System z10 Enterprise Class Configuration Setup, SG24-7571

IBM System z10 Enterprise Class Capacity on Demand, SG24-7504

Other publications

These publications are also relevant as further information sources:

DB2 Version 9.1 for z/OS Introduction to DB2 for z/OS, SC18-9847-04

DB2 Version 9.1 for z/OS Administration Guide, SC18-9840-05

DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration, SC18-9845-03

DB2 Version 9.1 for z/OS Reference for Remote DRDA Requesters and Servers, SC18-9853-02

DB2 Version 9.1 for z/OS Installation Guide, GC18-9846-08

DB2 Version 9.1 for z/OS Command Reference, SC18-9844-04

DB2 Version 9.1 for z/OS Application Programming and SQL Guide, SC18-9841-05

DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide, SC18-9851-06

DB2 Version 9.1 for z/OS SQL Reference, SC18-9854-07

DB2 Version 9.1 for z/OS Internationalization Guide, SC19-1161-03

DB2 Version 9.1 for z/OS XML Guide, SC18-9858-05

DB2 Version 9.1 for z/OS ODBC Guide and Reference, SC18-9850-03

DB2 Version 9.1 for z/OS What's New?, GC18-9856-02

DB2 Version 9.1 for z/OS Utility Guide and Reference, SC18-9855-04

DB2 Version 9.1 for z/OS, Reference for Remote DRDA Requesters and Servers, SC18-9853-03

DB2 Version 9.1 for z/OS Application Programming Guide and Reference for Java, SC18-9842-05

DB2 Version 9.1 for z/OS Codes, GC18-9843-06

DB2 Version 9.1 for z/OS Messages, GC18-9849-06

IRLM Messages and Codes for IMS and DB2 for z/OS, GC19-2666-02

IBM OmniFind Text Search Server for DB2 for z/OS Installation, Administration, and Reference Version 1 Release 1, GC19-1146-01

IBM Spatial Support for DB2 for z/OS User’s Guide and Reference Version 1 Release 2, GC19-1145-02
Online resources

These Web sites are also relevant as further information sources:

- DB2 for z/OS
  

- DB2 for z/OS Technical Resources (the DB2 for z/OS Library page)
  
  [http://www.ibm.com/support/docview.wss?rs=64&uid=swg27011656#db29-pd](http://www.ibm.com/support/docview.wss?rs=64&uid=swg27011656#db29-pd)

- zIIP reference information
  

- Extensible Markup Language (XML) architecture
  
  [http://www.w3.org/XML/](http://www.w3.org/XML/)

- Data encryption
  

- Whitepaper: DB2 9 and z/OS XML System Services Synergy Update
  

- Whitepaper: Maximizing offload to zIIP processors with DB2 9 for z/OS native SQL stored procedures
  

- Whitepaper by A. Hoshjkawa and J. Ruby-Brown: DB2 9 and z/OS XML System Services Synergy Update
  

- Whitepaper by A. Hoshjkawa and J. Ruby-Brown: DB2 9 and z/OS XML System Services synergy for distributed workloads
  

- White paper IBM System z and System Storage DS8000: Accelerating the SAP Deposits Management Workload With Solid State Drives, available at:
  

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

Help from IBM

IBM Support and downloads
## Index

### Symbols
- 'AS' clause 125

### Numerics
- 00C20255 257
- 00C2026A 257
- 00C2026B 257
- 00C2026C 257
- 5655-N97, DB2 Utilities Suite 261
- 64-bit DDF 104

### A
- access method 30
- access path hints 193
- accounting report long 322
- accounting trace
  - class 3 data 114
  - overhead 118
- adaptive multi-streaming prefetching (AMP) 146
- ALTER BUFFERPOOL 109
- ALTER TABLE 191
- AMP (adaptive multi-streaming prefetching) 146
- ANDing for star join query 31
- APAR
  - PK36297 305
  - PK40691 305
  - PQ96772 95
- API (application programming interface) 66
- application programming interface (API) 66
- Application Server (AS) 105
- application time 118
- archive log copy 135
- archive log file 135
- AREST state 253
- AS (Application Server) 105
- ascending index key order 163
- ASCII 85, 107, 334
- assisted buffer pool management 108
- asymmetric index page split 154
- autonomic DDL 49
- auxiliary table 129
- auxiliary warning (AUXW) 231
- AUXW (auxiliary warning) 231

### B
- BACKUP 213, 218–221
- BACKUP SYSTEM 154, 218–220
- base 10 arithmetic 136
- base table 26–27, 30, 40, 68, 78, 154, 230, 275, 329
- batch accounting report 305
- batch statistics report 305
- batch workload 84

- best practices 237
- BIGINT 4, 11, 41, 260, 264
- BINARY 41, 43
- BIT 43, 45, 312
- BLOB 43, 74, 188
- BPOOL 109, 313, 347
- broken page 233
- BDS 264, 266
- buffer 70
- buffer manager enhancements 124
- buffer pool 4, 101, 171
  - group 83
  - write performance 256
  - index 172
  - management 108–109
- BUILD2 7, 226–229
- business logic 62

### C
- CACHE 99, 157, 218, 347
- CACHEDYN_FREELOCAL 97
- call-level interface (CLI) 85, 107, 112, 183
- cartesian join 32
- CASTOUT 106, 312
- catalog migration 267
- CATEGORY 335
- CATENFM 263–264, 270–271
- CATMAINT 262–263, 267–269
- CCSID 334
  - corrections 292
  - set 334
- CDC (change data capture) 28
- change data capture (CDC) 28
- character large object (CLOB) 65, 74, 129, 185–186, 338
- CHECK constraint 46
- check constraints 154, 230, 264
- CHECK DATA 199, 223, 230–232
- CHECK INDEX 54, 200
- CHECK LOB 199, 230–232
- CHECK pending (CHKP) 231
- CHECKPAGE 233–234
- CHKP (CHECK pending) 231
- CI size 177
- CICS 345
- class 28, 62, 232, 245, 275
- CLI (call-level interface) 85, 107, 112, 183
- CLI ODBC 107
- CLOB (character large object) 65, 74, 129, 185–186, 338
- CLOB/VARCHAR 65
- clone table 154
- CLONE tables 153
- CLOSE NO 158–159, 344
- CLUSTER 158, 338
- cluster 267
DDF 5, 81–82, 102, 104–105, 107, 187, 242–243
DDL 49, 73, 157–158, 269, 337, 343, 351
INSTEAD OF trigger 344
DECFLOAT 41, 45–48, 136–138, 260, 368
decimal 41, 136, 277
DECIMAL FLOAT 11
decimal floating point 46
declam process 124
declared global temporary table 178
declared global temporary tables 126
declared temporary table 330
decomposition 69, 73
  methods 65
DEGREE 101, 340, 346
DELETE 17, 74, 133, 182, 327, 346
delete 24, 26–28, 39, 152, 213, 223, 231, 328
DELETE AGE 213
DELETE DATE 213
DES 347
DETERMINISTIC 247
Developer Workbench 246
DFSMS 136, 174, 231
DFSMSShsm 218–221, 223, 275
DFSORT 225
DGTT 126
dictionary-less compression 172
DISPLAY THREAD 110–111
DIST address space 81, 102, 104, 107–108
DISTINCT 13
DISTINCT sort avoidance 13
distributed 64-bit DDF 104
distributed computing standardization 63
DocID column 68
DocScan 80
document ID index 68
document node 66
double-byte character set (DBCS) 334
DPSI 28, 158, 166, 255, 294
DRAIN ALL 237
DRAIN WAIT 223
DRDA 85, 112, 129, 138, 297, 322–323
DS8000 4, 135, 176
DS8000 turbo 145
DSMAX 297
DSN_FUNCTION_TABLE 335
DSN_PLAN_TABLE 328
DSN_STATEMENT_CACHE_TABLE 336
DSN_STATEMENT_TABLE 334
DSN1COMP 174
DSN1LOGP 175
DSN6SPRM 97, 273
DSN6SYSP 126, 178, 275
DSNDB06 265, 270–271
DSNDQIST 127
DSN005I message 252
DSNJCNVB 266
DSNTEP2 260
DSNTEP4 260–261
DSNTESQ 273
DSNTIAUL 260
DSNTIJEN 263–264, 267, 270, 273
DSNTIJIN 267
DSNTIJUE 272
DSNTIJNF 263–264, 267, 273
DSNTJJP 267, 273
DSNTJPM 267, 273
DSNTJTC 262–263, 267, 269, 273
DSNTIP6 218, 220
DSNU633 216
DSNUTILB 267
DSNZPARM 29, 117, 273
DSVCI 126
MAXTEMPS 126–127
DSSIZE 152
DSVCI 126
DUMP 219
dump classes 219
DUMPONLY 219
dynamic index ANDing 31
dynamic prefetch 176, 216
  performance 176
dynamic SQL 36
DYNAMICRULES 339
E
EAV 301
e-business 61
ECSA (extended common storage area) 104–105, 107
EDITPROC 122
EDM
  DBD pool 94
  RDS pool above 94
  RDS pool below 94
  skeleton pool 95
  statement pool above 95
  storage components 94
EDM pool 94–97, 99, 101
  above-the-bar 101
  full condition 97
EDMPOOL 5
efficiency 77, 283
encoding scheme 334–335
ENDING AT 157
ENFM 262–264, 270
ENFM* 262
environment 19, 70, 74, 76, 81, 214, 246, 252, 261, 291
ERP 19
ESCON 143
EXCEPT 3, 34–36
EXCLUDE 120
EXISTS 15, 34–36
EXPLAIN 15, 78, 287, 327, 333, 335, 351, 368
  output 369
  table 16, 327
expression 30, 77, 330
extended common storage area (ECSA) 104–105, 107
extensions 58, 66, 336
EXTRACT 322, 346
F
fact table 32–33
fallback mode 263
fast log apply 238, 252–253
  function 221–222
  phase 174
FETCH 22, 26–27, 58–59, 75–76, 182, 323, 330, 346, 350
FETCH CONTINUE 57, 59
FETCH CURRENT 59
FETCH FIRST n ROWS ONLY 26
FICON 135
  channels 143
FIELDPROC 46
file reference variables 57, 260
FINAL TABLE 22
financial workload 129
Flash memory 146
FlashCopy 4, 135, 218, 221, 223, 232
  incremental 223
FMID 139
FMID H2AF110 261
FMID H2AG110 261
FMID J2AG110 261
FMID JDB991K 261
FOR BIT DATA 43–45
FORCE 219–220
FOREIGN KEY 46
FREQVAL 215–216
FROMDUMP 220
function 70, 244, 253, 327

G
GBPCACHE 158, 187, 255
GENERATED ALWAYS 157, 338
GENERATED BY DEFAULT 49
GET DIAGNOSTICS 24
Gl10-8737-00 262
global optimization 18
global query optimization 14
GRECP (group buffer pool RECOVER-pending) 252
  grid indexing 189
Group attach 300
group attachment 256
group buffer pool 83
  write performance 256
group buffer pool RECOVER-pending (GRECP) 252
GROUP BY 12
group collapsing 12

H
health check 267, 273
health monitor 256
High Performance FICON for System z 146
HISTOGRAM 55–56, 214–216
histogram statistics 4, 56–57, 214
host variable 36–37, 46, 57, 74–75, 192
HTML 62
HyperPAV 143

I
IBM Data Studio 248
IBM Relational Warehouse Workload 84
IBM Smart Analytics Optimizer 93
IBM Spatial Support for DB2 for z/OS 187, 189, 261
IBMREQD 271
ICF 124, 134
IDCAMS 267
IDXBPOOL 50
IEAOPTxx 139
IEASYSxx 107
IFCID 106, 337
0003 38
0057 171
147 139
148 139
2 97
2 (statistics record) 127
217 96–97, 99, 103, 106
22 16
225 96–97, 99, 106–107
231 139
239 139
27 30
3 115, 139
31 97
316 336
318 149, 336–337, 340
343 127
II14203 105
II14219 298
II14334 298
II1440 266, 298
II14426 69, 298
II14441 298
II14464 298
image copy 175, 220, 223, 232–233, 238, 265
IMPDSDSF 50, 275
IMPPLICIT 347
IMPTSCMP 50
IMS 261
incremental FlashCopy 223
index
ANDing 31
Compression 4, 172–174, 251, 255
  new-function mode 172
contention 156
document ID 68
key generation 210
key randomization 156
leaf page 159
look-aside 82, 131
manager 200
page size 116, 156, 159–161
page split, asymmetric 154
INDEX ON expression 51
indexing options 5
industry support 63
in-memory work file 29
INSERT 17, 19–20, 24, 26, 40, 43, 47, 54, 58, 69–70, 74,
133, 179, 182, 327, 330–331, 333–335, 345–346, 368
INSTEAD OF trigger DDL 344
INSTEAD OF triggers 39–40
instrumentation 99, 103
  for workfile sizing 127
workfile sizing 127
International Components for Unicode 261
International Organization for Standardization (ISO) 64
International Standard ISO 20022 79
INTERSECT 3, 34–35
investment preservation 63
IPCS 100
IRLM 81, 115–116, 254, 261, 310, 345–346
IS NULL 216
ISO (International Organization for Standardization) 64
ISOLATION LEVEL 311
IX 53, 254
IX ACCESS 35–36

J
Java 41, 46, 183, 242–243, 265
JCC 19, 85, 183, 242
JDBC 19, 84–85, 261
JOIN_TYPE 333

K
KEEPDYNAMIC 98

L
LANGUAGE SQL 247
latch class 19 83, 116
latch class 24 97, 117
latch class 6 116
  contention 167
leaf page 131–132, 160
least recently used (LRU) 97, 101, 339
LENGTH 41, 59
LIKE 52, 56, 216
LIST 216, 221, 346
list prefetch 148, 328
LISTDEF 232
  column 58, 232, 260
  data 183, 255, 260
  file reference 57
  file reference variable 57
  file reference variables 57
  locators 59
  locking 255
  processing 183
  REORG 237
table 229–231, 255, 276
LOB lock 182, 255
S-LOB 182
U-LOB 182
X-LOB 182, 187
LOB table space 230–231
location 64
locator 57–59, 183–187
lock 114, 182, 191
  LOB 255
  S 182
  S-LOB 182
  table-level retained 253
  U-LOB 182
  X 182
  X-LOB 182
locking 191–192, 255, 280, 282
LOG 27, 74, 115, 206, 227, 229, 236, 257, 346
log copy 219, 221
log data sets 238
log record sequence number (LRSN) 116, 223, 254, 264
log-based recovery 239
LOGAPPLY 221–223, 226–229
log-based recovery 239
LOGGED 175–176, 226
logging 5, 117, 174–176, 229, 254, 257
  attribute 175
long-displacement facility 135
LOOP 330
loose coupling 63
LRSN (log record sequence number) 116, 223, 254, 264
LRU (least recently used) 97, 101, 339
LSPR 87

M
maintenance 62–63, 105, 266, 291
map page 152
mass delete 26–29, 152
materialization 59
materialized query table (MQT) 330
MAXKEEPD 96
MAXOFILR 275
MAXPARTITIONS 152
MAXTEMPS 126–127, 178, 275
Media Manager 144
MEMBER CLUSTER 153
MEMLIMIT 105
memory monitoring 98
memory usage 105–106
MERGE 4, 11, 19–23, 330
MGEXTSZ 275
MIDAW 144
migration 261, 263
  process 261
  steps performance 268
millicode 136
MLS (multiple-level security) 28
MODIFY 212–213
MODIFY RECOVERY 213
modular 62
MQT (materialized query table) 330
MSTR 175
multiple allegiance 135, 142–143
multiple-level security (MLS) 28
multi-unit 188
MXDTCACH 29, 274

N
native SQL procedure 128
usability advantages 131
native SQL stored procedures 265
nested loop join 30
network trusted context 242
data sharing system 262
index compression 172
reordered row format 123
node 67
document 66
root 66
types 66
XML 66
non-clustered partitioning index 159
non-leaf page 131–132
non-partitioning indexes 7
NORMALIZE_DECFLOAT 46
NOSYSREC 236
NOT EXISTS 35
NOT LOGGED 175
NOT LOGGED logging attribute 175
NOT NULL GENERATED ALWAYS 191
NOT NULL GENERATED BY DEFAULT 191
NOT NULL WITH DEFAULT 329, 332
NOT PADDED attribute 210
NPI 200, 202–206, 208
NULL 56, 68, 157, 192, 216, 329, 368
NUMPARTS 152
NUMQUANTILES 10 216

O
OA03148 303
OA09782 303
OA17314 223, 299
OA17735 303
OA18461 109, 295, 303
OA19072 303
OA22443 303
OA23828 303
OA23849 303
OA26104 295, 303
OA33106 303
OA36101 303
OBD (object descriptor) 213
object descriptor (OBD) 213
object-level recovery 220
ODBC (Open Database Connectivity) 107, 261
OLAP 4
OLTP processing 83
OMEGAMON XE for DB2 Performance Expert on z/OS 280

OMEGAMON XE Performance Expert
accounting report long layout 322
batch accounting report 305
batch statistics report 305
statistics report long layout 306
online REORG 226
Open Database Connectivity (ODBC) 107
OPTIOWGT 274, 295
OPTIXOPREF 274
ORDER 16, 26–27, 157, 179, 238, 271, 322
ORDER BY 16, 26–27, 330–331
orphan row 34
overhead 63

P
package stability 193
package table (PT) 94, 96, 99–101
page set 175
page size 116, 125–126, 152, 156, 159–160, 264, 270, 275
index 156
paging 101
pair-wise join 31–32
with join back 31
parallel access volume (PAV) 135, 142
parallelism 16, 23, 34, 124, 205, 225–226, 327, 332
PARENT_PLANNO 16
PARTITION 46, 157, 238, 322, 346, 349
partition 7, 124, 143, 172, 200, 202–204, 207, 209, 214, 331
partition-by-growth table space 6, 152
partition-by-range table space 152, 157
PARTITIONED 158
partitioned table space 27, 152, 202
partitioning 7, 46, 152, 158–160, 164–165, 200, 202–203, 225
PARTITIONS 153
PAV (parallel access volume) 135, 142
PG 152
PBR 152
PERFM CLASS 3 127
CHECK INDEX utility 200
COPY 233
dynamic prefetch 176
fast log apply function 221
group buffer pool write 256
LOAD utility 201
REBUILD INDEX 204
REORG 205
RUNSTATS index 208
workload paging 101
Performance Warehouse function 280
PGFIX 109
physical lock (P-lock) 254
PIECESIZE 158
PK11129 262
PK17813 296
PK21237 124
PK28627 298
PK29281 292
PK31841 267
PK34251 292
PK36297 283, 305
PK37290 244, 292
PK37354 292
PK38867 292
PK40691 283, 305
PK40878 292
PK41001 299
PK41165 292
PK41323 50, 292
PK41370 292
PK41380 292
PK41711 292
PK41878 292
PK41899 292
PK42005 292
PK42008 292
PK42409 292
PK42801 292
PK43315 292
PK43475 292
PK43765 181, 292
PK43861 47, 298
PK44026 293
PK44133 293
PK44617 299
PK45599 299
PK45916 293
PK46082 293
PK4656 299
PK46687 293
PK46972 293
PK47126 299
PK47318 38–39, 293
PK47579 299
PK47594 74, 293
PK47649 294
PK47893 299
PK48453 293
PK48500 293
PK48773 299
PK49348 39, 293
PT (package table) 94, 96, 99–101
PTF UK23929/UK23930 305
PTF UK23984 305
pureXML 2, 61–62, 64
  storage 61
  support with DB2 for z/OS 65

Q
QISTW04K 127
QISTW23K 127
QISTWF40 127
QISTWF32 127
QISTWFVIC 127
QISTWFU0 127
QISTWFMX 127
QISTWFNE 127
QISTWFNP 127
QISTWFPP 127
QMF 286
QMF (Query Management Facility) 327
QUALIFIER 327
quantiles 214, 216
QUANTIZE 46
QUERY 310, 346
query 4, 61, 66, 84, 154, 214, 260, 285, 327
  parallelism 332
  performance 14, 78, 187
  processing 84
Query Management Facility (QMF) 327
query workload 90
QUERYNO 329, 335–336
QUIESCE 221–222, 349

R
RACF 243, 261
RANDOM 157, 313
randomize index keys 251
randomized index key order 163
range-partitioned table space 152
range-partitioned universal table space 152
RBA 124, 134, 223, 264
RBDP 226
RDS 96, 100
  pool above 94
  pool below 94
READS 337
real storage 81, 101–102
  monitoring 103
real-time statistics (RTS) 230, 264, 270, 293
REBIND 14
REBIND PACKAGE() PLANMGMT 195
REBUILD INDEX 54, 173, 199–200, 204–205, 209, 223
record identifier (RID) 132, 191, 226, 341, 346
RECOVER 154, 175, 218, 220–223, 232, 238, 274–275
Redbooks Web site 375
Contact us xxiv
Relational Warehouse Workload 84, 129
relative percentage overhead 119
RELCURHL 274

REMARKS 331
remote native SQL procedures 138
REOPT 36–37, 274, 341
REOPT(AUTO) 37, 341
reordered row format 121–123
REORG 7, 54, 79, 122–123, 199–200, 205–208, 267, 270, 293
REORG INDEX 205, 207–209
REORG TABLESPACE 205–206, 210, 226, 229
REPAIR 231
REPEAT 43
REPORT 110–111, 215
requirements 40, 63, 66, 103, 137, 228, 241, 244, 262
Resource Recovery Services (RRS) 242
resource unavailable condition 178, 226
Resource Unavailable SQLCODE (-904) 126
RESTORE 218, 220, 238, 274–275
RESTORE SYSTEM 218, 220, 275–276
RESTOREBEFORE 223
return 35, 62, 75–76, 204, 260, 330
RID (record identifier) 132, 191, 226, 341, 346
RLF 323, 347
RMF 102–103, 107, 139, 141
ROLLBACK 322, 346
root node 66
root page 132
row format 121
  reordered 121
ROWFORMAT xxvii, 123
ROWID 49, 324, 338, 341, 346
RRF 122
RRS (Resource Recovery Services) 242
RRSAF 111
RTS (real-time statistics) 230, 264, 270, 293
  index performance 208
runtime component 14

S
SAP 33, 242
SBCS 334
scale 46
SCHEMA 336, 338, 341, 349
schema 63, 69, 244, 266
schema document 73
SECOTY 158
segmented table space 24, 152, 253
SELECT CLOB 1 58
SELECT FROM MERGE 19
SELECT from MERGE 23
SELECT from UPDATE 24
SELECT list 15
SELECT SUBSTR 52
serialization 75
service-oriented architecture (SOA) 62, 241
services 63
SET 24, 58, 74, 97–98, 238, 257, 346
shared memory 104
shared memory object 102
SYSTABLEPART  236
SYSTABLESPACE
  TYPE column  152
System z Application Assist Processor (zAAP)  65, 138, 242, 262
System z10  85
System z9 Integrated Information Processor (zIIP)  65, 84, 242, 262
System z9 processor improvements  135
system-level backup  154

T
  table space
    compression  172
    partition-by-growth  152
    partition-by-range  152
    range partitioned  152
    scans  176, 287, 340
  table-level retained lock  253
  tables  15, 69–71, 94, 201, 253, 264, 280, 327
  TABLES_JOINED_THRESHOLD  275
  TAPEUNITS  220
  TBNAME  271
  TSBPLOB  50
  TSBPXMLE  50
  TBSPOOL  50
  TCP/IP  102, 322
  TEMP database  125
    merge with WORKFILE database  125
  TEMPLATE  232
    switching  232
  temporary table  30
  TEXT  129, 292, 298, 303, 336
  TIME  115, 139, 157, 238, 311, 345
  time spent in DB2  118
  TIMESTAMP  157, 191–192, 331, 333, 335–338
  traces  82, 118, 120–121, 150
    relative percentage overhead  119
  transitive closure  48
  tree structure  132
  triggers  26–27, 38–39, 125, 154, 187, 329, 343
  TRUNCATE  4, 11, 27–28
  trusted context  242
  TS  35, 322, 346
  TYPE  110, 139, 338, 344–345
  Type 4  183

U
  UDDB lock  213
  UA07148  303
  UA26564  303
  UA32755  303
  UA36199  303
  UA40609  303
  UA41774  303
  UA42372  303
  UA47647  303
  UA48912  109, 303
  UA56174  303
  UA60221  303
  UCB  143
  UDFs  244–246
  UK24125  292
  UK24545  292
  UK24934  50, 292
  UK25006  292
  UK25044  292
  UK25291  292
  UK25744  292
  UK25896  293
  UK26678  292
  UK26798  292
  UK26800  292
  UK27088  292
  UK28089  299
  UK28211  293
  UK28249  293
  UK28261  293
  UK28407  293
  UK29358  292
  UK29376  293
  UK29378  292
  UK29529  292
  UK29587  299
  UK30228  244
  UK30229  244, 292
  UK30279  299
  UK30693  299
  UK30713  293
  UK30714  299
  UK31069  293
  UK31092  299
  UK31151  299
  UK31630  39, 293
  UK31768  293
  UK31820  299
  UK31857  299
  UK31903  293
  UK31993  192, 293
  UK31997  299
  UK32047  299
  UK32060  299
  UK32061  299
  UK32795  293
  UK33048  294
  UK33449  299
  UK33456  293
  UK33493  299
  UK33510  293
  UK33636  292
  UK33650  299
  UK33731  293
  UK33795  294
  UK33962  299
  UK34342  299
  UK34808  294
  UK35132  49, 299
  UK35215  294
  UK35519  292
UNION  34
UNION ALL  34
UNIQUE  46
unique key index  49
unit of recovery (UR)  182
Universal Driver  183
universal table space  27–28, 152, 154
UNLOAD  49, 207–208, 225–228
UPDATE  17, 19–20, 24, 54, 58, 74–75, 115, 133, 191, 215, 263, 267, 327, 346
UQ89372  336
UR (unit of recovery)  182
usability advantages, native SQL procedures  131
USER  264, 306
USING VCAT  344
UTF-8  85
Utilities Suite  199
UTS  152
V
V8  294
VALIDPROC  28, 122
VALUE  54, 326, 349
VALUES  20, 40, 47, 58, 70, 72, 74, 312, 324, 330–331, 334–336, 345, 347
VARBINARY  4, 11, 41, 43, 45, 188, 260
VARCHAR  41, 44–45, 65, 157, 210, 247, 264, 329, 368
VARIABLE  106
variable  74, 96, 192, 210
VERSION  332, 346–347
virtual storage constraint relief (VSCR)  81, 94
virtual storage monitoring  99, 103
Visual Explain  284
VPSIZE  109
VSCR (virtual storage constraint relief)  81, 94
W
WARM  311
Web services  62
  componentization  63
  composability  63
  distributed computing standardization  63
  industry support  63
  investment preservation  63
  loose coupling  63
  platform independence  63
WebSphere  242–244, 247
well-formed XML  69, 74
WFDBSEP  128
WITH  22, 26, 58, 72, 157, 192, 243, 274, 329
WLM (Workload Manager)  4, 82, 108, 129, 135, 139, 345–346
WLM-established stored procedures address space
  (WLM-SPAS)  128
WLM-SPAS (WLM-established stored procedures address space)  128
work file  13, 26, 40, 178, 269
  query block  16
  sizing  126
sizing instrumentation  127
storage  126
workfile  294
WORKFILE and TEMP database merge  125
workload
  financial  129
  paging  101
  Relational Warehouse Workload  129
  simple  129
Workload Manager
  assisted buffer pool management  108
Workload Manager (WLM)  4, 129, 256
workload Statistics Advisor  286
write performance  255
X
X DBD lock  213
XES  255, 310, 346
X-LOB lock  182, 187
X-lock  182
  AS BLOB  69
  AS CLOB  69
  AS DBCLOB  69
  column  57–59, 65, 68–69, 71, 77–78
  data access  64
  data type  67
  documents  66–67, 73, 76
  index  77, 368
  nodes  66
  retrieval  75
  schema  69
  serialization  75
  structure  66
XML Extender  247, 261
XML schema repository (XSR)  69
XML System Services  65, 139
XMLEXISTS  77
xmlns  247
XMLPARSE  66, 70, 72, 74
XMLSERIALIZE  66
XPATH  61, 64
XPath  64, 71
  expression  77
XPath expression  77
XQuery  61, 64
XSR (XML schema repository)  69
Z
z/Architecture
  instruction set  4, 134
    long-displacement facility  135
z/OS 1.7  70, 76, 144, 178
z/OS XML  65
z/OS XML System Services  65
z10
  CPU time reduction 89
  Service Units 90
z10 performance 87
z800 135
z890 5, 135, 262
z900 135
z990 5, 135, 262
zAAP (System z Application Assist Processor) 65, 138, 242, 262
zBX 92
zEnterprise 92
zHPF 146
zIIP 32, 183
  usage 138
zIIP (System z9 Integrated Information Processor) 65, 84, 242, 262
zSeries 136, 262
DB2 9 for z/OS
Performance Topics

DB2® 9 for z/OS® is an exciting new version, with many improvements in performance and little regression. DB2 V9 improves availability and security, as well as adds greatly to SQL and XML functions. Optimization improvements include more SQL functions to optimize, improved statistics for the optimizer, better optimization techniques, and a new approach to providing information for tuning. V8 SQL procedures were not eligible to run on the IBM® System z9® Integrated Information Processor (zIIP), but changing to use the native SQL procedures on DB2 V9 makes the work eligible for zIIP processing. The performance of varying length data can improve substantially if there are large numbers of varying length columns. Several improvements in disk access can reduce the time for sequential disk access and improve data rates.

The key DB2 9 for z/OS performance improvements include reduced CPU time in many utilities, deep synergy with IBM System z® hardware and z/OS software, improved performance and scalability for inserts and LOBs, improved SQL optimization, zIIP processing for remote native SQL procedures, index compression, reduced CPU time for data with varying lengths, and better sequential access. Virtual storage use below the 2 GB bar is also improved.

This IBM Redbooks® publication provides an overview of the performance impact of DB2 9 for z/OS, especially performance scalability for transactions, CPU, and elapsed time for queries and utilities. We discuss the overall performance and possible impacts when moving from version to version. We include performance measurements that were made in the laboratory and provide some estimates. Keep in mind that your results are likely to vary, as the conditions and work will differ. In this book, we assume that you are somewhat familiar with DB2 V9. See DB2 9 for z/OS Technical Overview, SG24-7330, for an introduction to the new functions.