IBM DATABASE 2 Version 2.1.0 Performance Report

December 22, 1995

DB2 Common Server Performance Group

IBM Software Solutions Toronto Laboratory
Comments/Questions to DB2TECH at TOROLAB2

Preface

This report describes the various performance measurements that have been carried out in support of DB2 V2.1.0 and DDCS V2.3. Our thanks to staff in the Santa Teresa Lab for the DDCS performance results, and in the Almaden Research Centre for some query results.

Comments or questions on this report can be submitted via IBM personnel as designated on the cover page.

The information contained in this report has not been submitted to any formal IBM test and is distributed as-is. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customers′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

All performance data contained in this report was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this report should verify the applicable data for their specific environment.

Version Notice

This is **version 1.0** of this document.

Summary of Changes

Future changes to this document, if required, will be summarized at this section in reverse chronological sequence. Revision bars (|) will highlight the text changed in new document versions.

Contents

1.0 Introduction . . . 9
1.1 Who should read this document? . . . 9
1.2 What is in this report? . . . 9
1.3 Overview of document format . . . 10
1.4 Report Highlights . . . 11
1.5 Preview of V2.1.1 Improvements . . . 13
1.6 Additional Information . . . 13

2.0 Performance Benchmark Results . . . 14
2.1 OLTP Benchmarks . . . 14
2.1.1 TPC-C Benchmark . . . 14
2.2 SPEC Benchmark . . . 19
2.3 RQW Benchmark . . . 28
2.4 Customer Benchmarks . . . 30
2.4.1 Customer Benchmark 1 (CB1) . . . 30
2.4.1.1 CB1 Query Analysis . . . 30
2.4.1.2 Miscellaneous Definitions . . . 30
2.4.1.3 CB1 Results . . . 33
2.4.2 Customer Benchmark 2 (CB2) . . . 34

3.0 Performance of Queries . . . 37
3.1 Table Scans . . . 37
3.1.1 General Overview . . . 37
3.1.2 Improvements . . . 37
3.1.3 Considerations . . . 37
3.1.4 Performance Tips . . . 40
3.2 Index Scans . . . 41
3.2.1 General Overview . . . 41
3.2.2 Improvements . . . 41
3.2.3 Considerations . . . 41
3.2.4 Performance Tips . . . 42
3.3 Performance of Queries on OS/2 . . . 44
3.3.1 DB2/2 Table Scans . . . 44
3.3.2 DB2/2 Index Scans . . . 45
3.4 SQL Methods and Implementations . . . 46
3.4.1 Dynamic SQL Caching . . . 46
3.4.2 Compound SQL . . . 50
3.4.3 Static vs. Dynamic vs. CLI . . . 51
3.4.4 db2batch . . . 53
3.5 Optimization Classes . . . 54
3.6 Sorting . . . 57

4.0 Performance of Utilities . . . 58
4.1 Import . . . 58
4.1.1 General Overview . . . 58
4.1.2 Improvements . . . 58
4.1.3 Considerations . . . 58
4.1.4 Performance Tips . . . 60
4.2 Load . . . 61
4.2.1 General Overview . . . 61
4.2.2 Improvements . . . 61

4.2.3 Considerations 61
4.2.4 Performance Tips 64
4.3 DB2/2 Load, Create Index, RUNSTATS 65
4.4 Load vs. Import 67
4.4.1 General Overview 67
4.4.2 Improvements 67
4.4.3 Considerations 68
4.4.4 Performance Tips 69
4.5 Create Index 70
4.5.1 General Overview 70
4.5.2 Improvements 70
4.5.3 Considerations 70
4.5.4 Performance Tips 72
4.6 RUNSTATS 73
4.6.1 General Overview 73
4.6.2 Improvements 73
4.6.3 Considerations 73
4.6.4 Performance Tips 74
4.7 Backup/Restore 75
4.7.1 General Overview 75
4.7.2 Improvements 75
4.7.3 Considerations 75
4.7.4 Performance Tips 77
4.8 Reorg 78
4.8.1 General Overview 78
4.8.2 Improvements 78
4.8.3 Considerations 78
4.8.4 Performance Tips 79

5.0 Miscellaneous Observations 81
5.1 Crash Recovery 81
5.1.1 General Overview 81
5.1.2 Improvements 81
5.1.3 Considerations 81
5.1.4 Performance Tips 82
5.2 CONNECT 83
5.2.1 General Overview 83
5.2.2 Improvements 83
5.2.3 Considerations 84
5.2.4 Performance Tips 87
5.3 Data Management 88
5.3.1 Performance Tips 88
5.4 Configuration Parameters 90
5.4.1 OLTP 90
5.4.2 Complex Queries 90
5.4.3 Simple Queries 91
5.4.4 Large Numbers of Users 91
5.4.5 Heavy Loading of Data 91
5.4.6 REORG 91

6.0 Performance of DDCS 92
6.1.1 General Overview 92
6.1.2 Improvements 92
6.1.3 Considerations 92
6.1.3.1 Nature of Queries 93

6.1.3.2 Communication Subsystem RU Size . . . 93
6.1.3.3 RQRIOBLK . . . 93
6.1.4 Performance Measurement Results . . . 93
6.1.4.1 Client with DDCS --> DB2/MVS Measurements . . . 94
6.1.4.2 OS/2 Client --> DDCS --> DB2/MVS Measurements . . . 95
6.1.4.3 Windows** Client --> DDCS --> DB2/MVS Measurements . . . 98
6.1.5 Performance Tips . . . 99

Appendix A. Benchmark and Database Specifications . . . 101
A.1 SPEC . . . 101
A.1.1 SPEC Queries . . . 101
A.1.2 SPEC Database: Data Characteristics . . . 101
A.1.3 SPEC Statistics on Database Tables and Indexes . . . 101
A.1.4 SPEC DBM and Database Configurations on AIX . . . 101
A.1.5 SPEC DBM and Database Configurations on OS/2 . . . 101
A.1.6 SPEC Explain Output . . . 101
A.2 RQW . . . 101
A.2.1 RQW Query Descriptions . . . 101
A.2.2 Catalog Statistics for RTW Database . . . 102
A.3 Internal Benchmark/Database 1 (IB1) . . . 102
A.4 Internal Database 2 (ID2) . . . 102
A.5 Internal Database 3 (ID3) . . . 102
A.6 Customer Benchmark 1 . . . 102
A.6.1 CB1 Descriptions . . . 102
A.6.2 CB1 Table Descriptions . . . 102
A.6.3 CB1 DBM and Database Configurations on AIX . . . 102
A.7 Customer Benchmark 2 . . . 102
A.7.1 CB2 Query Descriptions . . . 102
A.7.2 CB2 Table Descriptions . . . 102
A.7.3 CB2 Statistics on Database Tables and Indexes . . . 103
A.7.4 CB2 DBM and Database Configurations . . . 103

Appendix B. Hardware Specifications . . . 104
B.1 Configuration for machine SERVER 1 (AIX) . . . 104
B.2 Configuration for machine SERVER 2 (AIX) . . . 104
B.3 Configuration for machine SERVER 3 (AIX) . . . 104
B.4 Configuration for machine SERVER 4 (OS/2) . . . 104
B.5 Configuration for machine SERVER 5 (OS/2) . . . 104
B.6 Configuration for CB1 benchmark machine (AIX) . . . 104
B.7 Configuration for CB2 benchmark machine (AIX) . . . 104

Appendix C. DDCS Measurement Configuration . . . 105
C.1.1.1 OS/2 or Windows Client . . . 105
C.1.1.2 OS/2 DDCS Gateway (or Local) . . . 105
C.1.1.3 AIX DDCS Gateway (or Local) . . . 105
C.1.1.4 MVS Host . . . 105
C.1.1.5 LAN . . . 105

Figures

1. Highlights of DB2 V2.1.0 vs DB2 V1.2 on AIX and OS/2 . . . 11
2. RS/6000 TPC-C Revision 3.0 results as of November 3, 1995 . . . 15
3. Executive Summary (page 1) for DB2 TPC-C on 4-way RS/6000 J30 . . . 16
4. Executive Summary (page 1) for DB2 TPC-C on 8-way RS/6000 J30 . . . 17
5. Executive Summary (page 1) for DB2 TPC-C on RS/6000 R24 . . . 18
6. Summary of SPEC Benchmark Queries . . . 19
7. LOCAL SPEC Measurements for AIX and OS/2 (elapsed times in seconds) . . . 20
8. CLIENT/SERVER SPEC Measurements for AIX (elapsed times in seconds) . . . 21
9. SPEC query performance V1.2 and V2.1 . . . 22
10. V1.2 Explain for SPEC query 8. . . . 22
11. V2.1 Explain for SPEC query 8. . . . 23
12. V1.2 Explain for SPEC query 12. . . . 24
13. V2.1 Explain for SPEC query 12. . . . 24
14. V1.2 Explain for SPEC query 19. . . . 25
15. V2.1 Explain for SPEC query 19. . . . 26
16. RQW Results . . . 29
17. CB1 Query Analysis . . . 31
18. CB1 Results for DB2 V1.2 vs. V2.1 . . . 33
19. CB2 Results for DB2 V1.2 vs. V2.1 . . . 35
20. Table Scan Measurements . . . 38
21. Table Scan Measurements for Varying Database Configurations and Prefetch Quantities . . . 39
22. Index Scan Measurements for Varying Database Configurations and Prefetch Quantities . . . 42
23. OS/2 Table Scan Measurements for Varying Database Configurations and Prefetch Quantities . . 44
24. OS/2 Index Scan Measurements for Varying Database Configurations and Prefetch Quantities . . 45
25. Caching measurements. . . . 47
26. Compound SQL Performance . . . 50
27. Compound SQL Performance . . . 50
28. Elapsed times for different Application Coding methods . . . 51
29. db2batch vs. CLP . . . 53
30. Prepare Time for Optimization classes 5,3,1,0 . . . 54
31. Run Time for Optimization classes 0,1,3,5 . . . 55
32. Prepare Times . . . 55
33. Run Times . . . 56
34. Sorting costs . . . 57
35. Import Performance . . . 59
36. Comparison of Import Performance with Varying Configurations . . . 59
37. DB2/6000 Load Scenarios . . . 62
38. Comparison of Different Data Loading Scenarios . . . 62
39. LOAD Performance over Varying Database Sizes . . . 63
40. LOAD - build index vs. # of Sort Directories . . . 63
41. DB2/2 Load Scenarios . . . 65
42. DB2/2 Comparison of Different Data Loading Scenarios . . . 65
43. LOAD vs IMPORT, no indexes. . . . 68
44. LOAD vs IMPORT: no indexes . . . 68
45. LOAD vs IMPORT: 1 index . . . 69
46. LOAD vs IMPORT: 2 indexes . . . 69
47. Appending data using Load and Import . . . 69
48. Create Index per buffpage . . . 71
49. Create Index per sortheap . . . 71
50. Impact of SORTHEAP and BUFFPAGE on CREATE INDEX . . . 72
51. RUNSTATS elapsed times in various scenarios . . . 74

52. Backup/Restore . . . 76
53. Backup and Restore Performance over Varying Database Sizes . . . 76
54. Comparison of Different Backup Scenarios . . . 76
55. Performance results for Reorg (elapsed times in seconds) . . . 79
56. Version 1.2 - Version 2.1 comparison of recovery and update times . . . 81
57. Recovery as affected by workload . . . 82
58. Recovery as affected by I/O cleaners . . . 82
59. Connect Times on a Local Server . . . 84
60. Connect Times on a Local Server . . . 84
61. Connect Times on a Client/Server, <AUTHENTICATION> = CLIENT . . . 85
62. Connect Times on a Client/Server, <AUTHENTICATION> = CLIENT . . . 85
63. Connect Times on a Client/Server, <AUTHENTICATION> = SERVER . . . 86
64. Connect Times on a Client/Server, <AUTHENTICATION> = SERVER . . . 86
65. Connect Times on a Local Server for Version 1.2 . . . 87
66. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (Local) . . . 94
67. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (Local) . . . 94
68. DDCS/2 V1.2 vs. DDCS/2 V2.3 (Local) . . . 95
69. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (OS/2 APPC Client) . . . 96
70. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (OS/2 APPC Client) . . . 96
71. DDCS/2 V1 vs. DDCS/2 V2.3 (OS/2 NetBIOS Client) . . . 97
72. DDCS/6000 V2.3 (OS/2 TCP/IP Client) . . . 97
73. DDCS/2 V2.3 (OS/2 TCP/IP Client) . . . 98
74. DDCS/2 V1.2 vs. DDCS/2 V2.3 (Windows** NetBIOS Client) . . . 98
75. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (Windows** TCP/IP Client) . . . 99
76. DDCS/2 V2.3 (Windows** TCP/IP Client) . . . 99

1.0 Introduction

This report presents performance test results for DB2 Common Server (DB2 C/S) V2.1.0 and DDCS V2.3 in AIX and OS/2 environments.

Most of the results were generated using DB2 for AIX. Because the vast majority of DB2 C/S code is common to all platforms it runs on, the DB2 for AIX results are similar to those for other platforms, subject to the performance differences among the underlying hardware and operating systems.

Most of the results in this document were NOT optimized; instead, they reflect typical or default configurations. These results should not be interpreted as being representative of the results that all prospective users of DB2 V2.1.0 or DDCS V2.3 can expect to attain in their own processing environments. The measurement run characteristics (workload, concurrency level, etc.) may be meaningful for some but not necessarily for all users or for all system configurations. The performance of individual users′ systems will vary with the options and variables they select in configuring their system environment.

Note: This is a TECHNICAL document and a technical knowledge of DB2 V2.1.0 is necessary for complete understanding of the results contained herein and their applicability to specific environments.

1.1 Who should read this document?

This document is intended for people working with DB2. The purpose of this document is to:

a) Demonstrate the performance of DB2 V2.1.0 in many of its key functions.

b) Show the performance improvements of DB2 V2.1.0 vs. its predecessor, DB2 V1.2.0. For a quick summary page see 1.4, "Report Highlights" on page 11.

c) Provide tuning tips.

Therefore the document has a wide audience, encompassing both Database Administrators who wish to increase the performance of their databases, and IBM and customer personnel who want to understand the performance advantages of upgrading/switching to DB2 V2.1.0.

1.2 What is in this report?

This report summarizes the performance of DB2 V2.1.0. You will find in this report several benchmarks (TPC-C, SPEC, RQW, and customer benchmarks) and general performance descriptions of queries, utilities, and miscellaneous functions. The queries which we investigated include index scans and table scans. The utilities and miscellaneous functions include Import, Load, Create Index, Backup/Restore, Runstats, Reorg, crash recovery, and Connect.

It is beyond the scope of this report to provide exhaustive details for all reported benchmarks. For a reader to fully understand what′s behind the performance of a query, for example, requires such information as query syntax, Explain output, answer set size, configuration parameter values, full database characteristics (logical and physical), database monitor snapshot output, etc. Most of this information is provided for the SPEC benchmark, but only a subset is provided for the others.

Many of the multiple-query benchmarks in this document have their performance compared between V1.2 and V2.1 in terms of the following metrics with differing attributes:

Arithmetic Mean The total of N measurements, divided by N. One drawback of arithmetic mean is that large differences in the times for some of the longer measurements mask differences in shorter measurements. On the other hand, differences in arithmetic mean are highly relevant to scenarios like batch execution of N queries within a limited time period.

Geometric Mean The Nth root of the product of N measurements. With geometric mean, all queries have the same weighting. One problem is that a query that degrades from 0.1 seconds to 0.2 seconds, for example, will have the same percentage effect on the geometric mean as a query that degrades from 10 minutes to 20 minutes; the latter is probably of much more concern to users.

1.3 Overview of document format

This document is organized into these major chapters:

- 2.0, "Performance Benchmark Results" on page 14: Here each benchmark is briefly introduced, the queries are described, and the results are given. For more information on the hardware and database configurations we used, please refer to the appendix.
- 3.0, "Performance of Queries" on page 37: Analysis of table scans, index scans, sorting, and the various aspects of dynamic SQL queries. As much as possible, each discussion is laid out in the following format:
  - General Overview: a brief description of when and where this query type is used, and what other commands are related.
  - V2 Improvements: new enhancements since Version 1.
  - Considerations: Various performance factors; rules of thumb; useful graphs.
  - Performance tips and things to avoid.
- 4.0, "Performance of Utilities" on page 58: import, load, create index, backup, etc... The format for each utility is much like the one used in the section on query performance.
- 5.0, "Miscellaneous Observations" on page 81: discussions of various topics, organized much like the preceding ones.
- 6.0, "Performance of DDCS" on page 92: DDCS results, improvements, considerations and tips.
- The appendices describe the benchmarks, databases, and machine configurations used in the various measurements. Within this document, only very basic details are provided. The companion document, *DB2 Version 2.1.0 Performance Report - Appendices*, provides more details (being 91 pages long, it was made a separate document to reduce paper usage by those just intereste d in the basics). The appendix section numbers in the two documents are identical.

1.4 Report Highlights

DB2 V2.1 provides vast improvements over V1.2 in most areas, including OLTP, query, and utility performance. This section summarizes the key contents of this report.

Figure 1. Highlights of DB2 V2.1.0 vs DB2 V1.2 on AIX and OS/2. Most of the numbers shown are percentage reductions in elapsed time; e.g., a 50% reduction means "twice as fast."

Function	Improvement on AIX	Improvement on OS/2	Reference
OLTP Performance	up to 23%	up to 40%	2.1, "OLTP Benchmarks" on page 14.
SPEC benchmark (42 simple queries)	19-53%	39-42%	2.2, "SPEC Benchmark" on page 19.
RQW benchmark (30 fairly complex queries)	27-46%	-	2.3, "RQW Benchmark" on page 28.
Customer benchmark 1 (30 very complex queries)	65-83%	-	2.4.1, "Customer Benchmark 1 (CB1)" on page 30.
Customer benchmark 2 (203 simple dynamic queries)	44%	-	2.4.2, "Customer Benchmark 2 (CB2)" on page 34.
Table scans	41%	39%	3.1, "Table Scans" on page 37 and 3.3.1, "DB2/2 Table Scans" on page 44.
Index scans	-	16-68%	3.3.2, "DB2/2 Index Scans" on page 45.
Sorting	79%	-	3.6, "Sorting" on page 57.
Import	77%	-	4.1, "Import" on page 58, and 4.4, "Load vs. Import" on page 67.
Load	1.7GB/hr	1.4GB/hr	4.2, "Load" on page 61, and 4.3, "DB2/2 Load, Create Index, RUNSTATS" on page 65.
Create Index	16-40%	67%	4.5, "Create Index" on page 70, 4.3, "DB2/2 Load, Create Index, RUNSTATS" on page 65, and 4.2, "Load" on page 61.
RUNSTATS	39%	31%	4.3, "DB2/2 Load, Create Index, RUNSTATS" on page 65, 4.2, "Load" on page 61, and 4.6, "RUNSTATS" on page 73.
Backup/Restore	14GB/hour+	9GB/hour+	4.7, "Backup/Restore" on page 75.
Reorg	55-58%	47%	4.8, "Reorg" on page 78.
DDCS	up to 49%	up to 48%	6.0, "Performance of DDCS" on page 92

Here is brief elaboration on some of the items in Figure 1 on page 11, as well as some other performance topics discussed in this report:

- Dynamic SQL caching: Improvements of about 3 times were observed from the caching of dynamic SQL plans in V2.1. See 3.4.1, "Dynamic SQL Caching" on page 46.
- Compound SQL: Although not a new feature of V2.1, Compound SQL enabled elapsed time reductions of 37-64% in a batch update scenario. See 3.4.2, "Compound SQL" on page 50.
- "SQL implementation": Measurements were performed showing the relative performance of static SQL, dynamic SQL, and CLI, with and without parameter markers. See 3.4.3, "Static vs. Dynamic vs. CLI" on page 51.
- db2batch tool: This performance benchmarking tool is new in V2.1 and can execute some queries many times faster than the db2 command. See 3.4.4, "db2batch" on page 53.
- Optimization classes: These classes were defined to allow an appropriate amount of optimizer effort to be applied to the access plan selection process. See 3.5, "Optimization Classes" on page 54.
- Crash Recovery: A 62% reduction in recovery time with V2.1 was measured on AIX, and page cleaning was seen to reduce recovery time by over 23%. See 5.1, "Crash Recovery" on page 81.
- Data Management: Guidance is provided on the question of whether to use database-managed or system-managed storage for DB2 V2.1 databases. See 5.3, "Data Management" on page 88.
- Configuration Parameters: Many new configuration parameters were added in V2.1. Guidance is provided on setting the parameters which are most significant in various workloads. See 5.4, "Configuration Parameters" on page 90.
- DDCS: Improvements of 26-49% were observed in measurements in DDCS standalone and gateway environments using OS/2 and AIX clients. See 6.0, "Performance of DDCS" on page 92.

1.5 Preview of V2.1.1 Improvements

Several key performance improvements will be shipped in a refresh of V2.1.0, and many have been measured as of this report date. The improvements include:

- ACTIVATE DATABASE command. The DBA will be able to activate a database prior to the first CONNECT to it. As a result, the first CONNECT will be much faster (taking the same time as other CONNECTs) and the desirability of a ″dummy CONNECT″ will be eliminated.
- Index prefetch. For DMS tablespaces in particular, this will drastically improve the elapsed time of index scans, and utilities that perform index scans (RUNSTATS, REORG, REORGCHK).
- BACKUP and RESTORE. These are several times faster due to increased parallelism.
- Query performance. Improvements have been made in such areas as sort memory consumption and path selection, with order of magnitude reductions in elapsed time for some queries.

1.6 Additional Information

Numerous files related to DB2 V2.1.0 are available in DB2INFO for IBM employees, and on CompuServe for others. The PERFORM PACKAGE in DB2INFO contains performance information on V2.1 and previous releases.

A good way to start with DB2INFO is to run the following on VM:

```
REQUEST DB2INFO FROM DB2INFO AT TOROLAB2
```

After receiving the returned EXEC and FILELIST files, invoke DB2INFO to see a list of, and be able to request, particular files or packages.

The following are additional sources of information on DB2 and DDCS performance:

- Details on the benchmarks in this report are in the separate appendix document: *DB2 Version 2.1.0 Performance Report - Appendices*. Available in PERFORM PACKAGE.
- *Administration Guide for DB2 Common Servers*, S20H-4580
- Various DB2 TPC full disclosure reports and executive summaries. Available in PERFORM PACKAGE.
- Summary of TPC Results for DB2 C/S. (DB2TPC TEXT in PERFORM PACKAGE).
- *DB2/6000 V1.2 Performance Report*. Available in PERFORM PACKAGE.
- *DB2/2 & CAE V1.2 Performance Report*. Available in PERFORM PACKAGE.
- *DB2/2 V1.2 SMP Performance Report*. Available in PERFORM PACKAGE.

2.0 Performance Benchmark Results

2.1 OLTP Benchmarks

Major improvements in OLTP performance have occurred in DB2 V2.1.0. Throughput increases of up to 40% have been seen using an internal benchmark similar to the TPC-B industry standard on OS/2, and up to 23% on AIX. Customer benchmarks should typically see improvements in the 15-30% range (or more, to the extent that V2.1 enhancements can be exploited).

Some of the most significant V2.1 OLTP enhancements were:

- Ability to spread data over multiple logical drives in DB2/2, thus minimizing I/O contention.
- Extensive rewriting of locking, logging, and index manager components to reduce pathlength and response times.
- Exploitation of SMP machines by minimizing internal contention for DB2 data areas.
- Sort optimization, for both small and large sorts.

2.1.1 TPC-C Benchmark

TPC-C is an industry-standard OLTP benchmark consisting of 5 moderately complex transactions.

Two DB2 V2.1 TPC-C results, based on the RISC System/6000 J30 and R30 8-way SMP machines, were announced on June 19, 1995. These are outstanding results both for performance and price/performance, as the J30/R30 tops all TPC-C results filed for 8-way 32-bit systems in both metrics.

The DB2 RS/6000 J30 result can be compared to the Informix 7.1 result on the Bull ESCALA D401/8: 2660 tpmC at 530 $/tpmC. The D401/8 is also an 8-way PowerPC 601 machine and had the same amount of memory (2GB) as the J30. The DB2 result represents 17% higher throughput and 34% better price/performance.

Here is a list of all RS/6000 TPC-C Revision 3.0 results as of November 3, 1995:

Machine	tpmC	$/tpmC	DBMS
RISC 6000/R24 c/s	1470.06	$666	DB2 for AIX V2.1
RISC 6000/390 c/s	768.28	$711	Sybase SQLServer 10.0.1
RISC 6000/390 c/s (*)	901.57	$652	Sybase SQLServer 10.0.1
RISC 6000/C10 c/s	404.92	$707	Sybase SQLServer 10.0.1
RISC 6000/C10 c/s (*)	485.88	$654	Sybase SQLServer 10.0.1
RISC 6000/E20 c/s	735.27	$378	Sybase SQLServer 10.0.2
RISC 6000/J30 4way cs	1438.04	$492	DB2 for AIX V2.1
RISC 6000/R30 4way cs	1438.04	$510	DB2 for AIX V2.1
RISC 6000/J30 4way cs	1298.10	$691	Sybase SQLServer 10.0.3
RISC 6000/J30 8way cs	3119.16	$349	DB2 for AIX V2.1
RISC 6000/R30 8way cs	3119.16	$355	DB2 for AIX V2.1
ESCALA D401/8 8way cs	2660.03	$530	Informix Online 7.1
ESCALA D201/8 8way cs	2660.03	$560	Informix Online 7.1

(*) These results were based on a 1MB L2 cache.

Bull ESCALA results are included because they were based on the same processors and amount of memory as the equivalent RS/6000.

Figure 2. RS/6000 TPC-C Revision 3.0 results as of November 3, 1995

Further information on TPC-C results for DB2 V2.1.0, as well as the specification for TPC-C, are available in the PERFORM PACKAGE in DB2INFO; refer to 1.6, “Additional Information” on page 13.

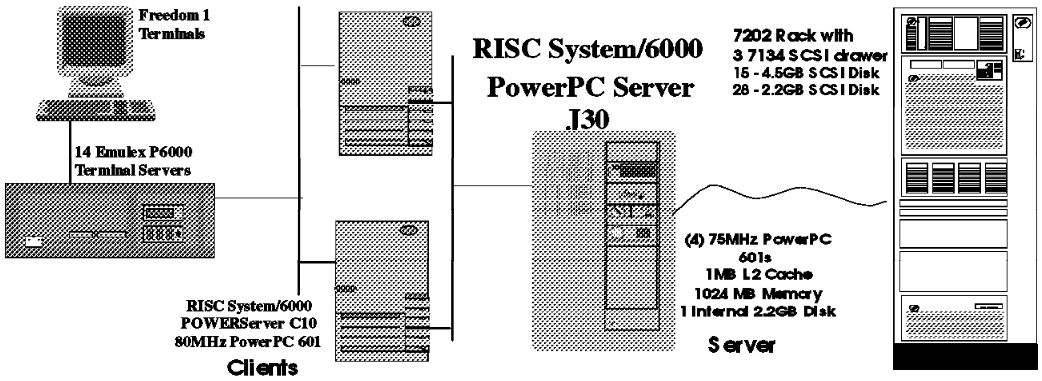
The following figures contain the first pages of TPC-C executive summaries for DB2 V2.1.0 V2.1 on 4-way and 8-way RS/6000 SMP machines and the R24 uniprocessor. It should be noted that enhancements to DB2 and AIX, and the use of more real memory, contributed to the throughput more than doubling between the 4-way and 8-way results. This superlinear scalability is not the norm, even for DB2!

IBM	RISC System/6000 PowerPC Server J30 c/s	TPC-C Rev. 3.0 Report Date: June 5,1995

Total System Cost	TPC-C Local Throughput	Price/Performance	Availability Date:
$707,470	1,438.04 tpmC	$492/tpmC	now except: DB2 for AIX:12/15/95

Processors	Data Base Manager	Operating System	Other Software	Number of users
(4) 75 Mhz PowerPC 601	DB2 for AIX Version 2.1	AIX 4.1.1.2	IMC Tuxedo EPT 4.2.1 DB2 Client Application Enabler for AIX (Included with DB2 for AIX)	1220

Priced System Configuration



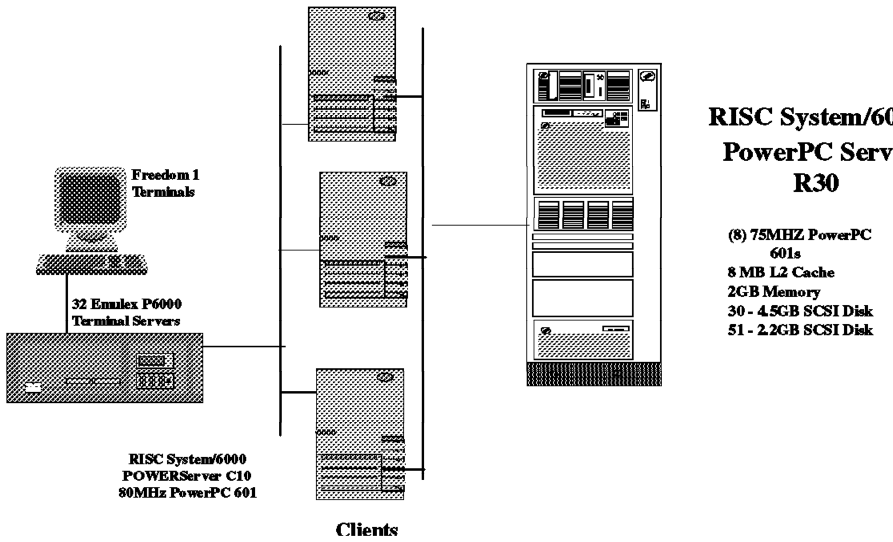
System Components	Qty	Client Description	Qty	Server Description
Processors	2	IBM RS/6000 C10	1	IBM RS/6000 J30
Memory		256 MB each client	1	1,024 MB Main Memory
Disk Controllers		Integrated SCSI	6	SCSI-2 Adapter
Disk Drives	2	1.0 GB SCSI Disks	15	4.5 GB SCSI-2 Disks
	1	2.0 GB SCSI Disk	29	2.2GB SCSI-2 Disks
Total GB of Storage		2 GB each client		123.76 GB formatted capacity
Tape Drives				
Terminals	2	Liberty Freedom One consoles	1	Console
Terminal Connectivity	14	Emulex Terminal Server		Direct Connect

Figure 3. Executive Summary (page 1) for DB2 TPC-C on 4-way RS/6000 J30

IBM	RISC System/6000 PowerPC Server R30 c/s	TPC-C Rev. 3.0 Report Date: June 19, 1995	
Total System Cost	**TPC-C Local Throughput**	**Price/Performance**	**Availability Date:**
$1,106,716	**3,119.16tpmC**	**$355/tpmC**	now except: DB2 for AIX:12/15/95 cpu 5-8: 8/25/95

Processors	Data Base Manager	Operating System	Other Software	Number of users
(8) 75 Mhz PowerPC 601	DB2 for AIX Version 2.1	AIX 4.1.2	IMC Tuxedo EPT 4.2.1 DB2 Client Application Enabler for AIX (Included with DB2 for AIX)	2,750

Priced System Configuration



System Components	Qty	Client Description	Qty	Server Description
Processors	3	IBM RS/6000 C10	1	IBM RS/6000 R30
Memory	3	256 MB	4	512 MB Main Memory
Disk Controllers		Integrated SCSI	10	SCSI-2 Adapter
Disk Drives	3	2.0 GB SCSI Disks	30 51	4.5 GB SCSI-2 Disks 2.2GB SCSI-2 Disks
Total GB of Storage		2 GB each client		232.84 GB
Tape Drives				
Terminals	4	Liberty Freedom 1 console		
Terminal Connectivity	32	Emulex Terminal Server		Direct Connect

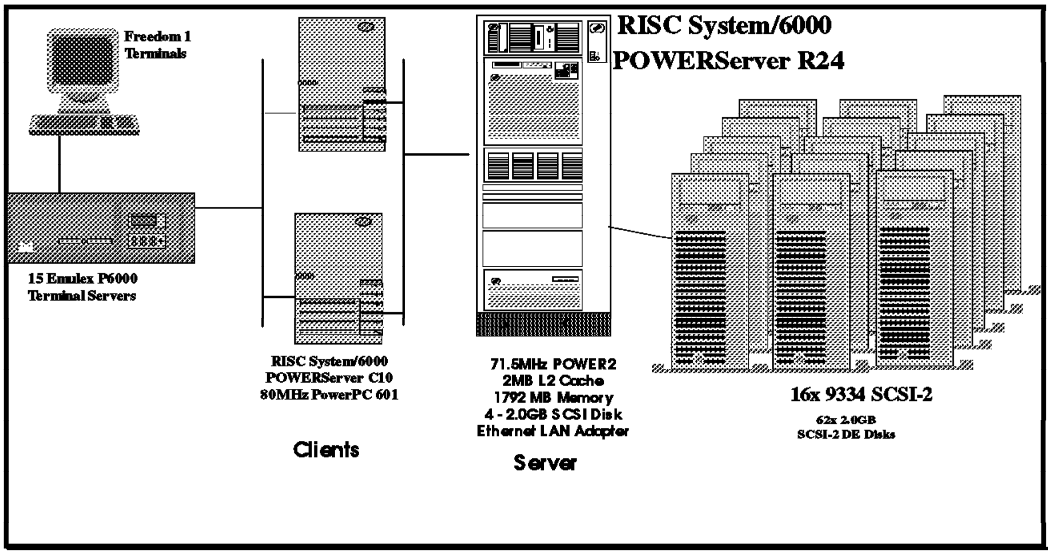
Figure 4. Executive Summary (page 1) for DB2 TPC-C on 8-way RS/6000 J30

IBM	RISC System/6000 POWERServer R24 c/s	TPC-C Rev. 3.0 Report Date: June 5, 1995

Total System Cost	TPC-C Local Throughput	Price/Performance	Availability Date:
$979,231	1,470.06 tpmC	$666/tpmC	Available now except DB2 for AIX available Dec 15, 1995

Processors	Data Base Manager	Operating System	Other Software	Number of users
RISC System/6000 POWERServer R24	DB2 for AIX Version 2.1	AIX 3.2.5	IMC Tuxedo EPT DB2 Client Application Enabler for AIX (Included with DB2 for AIX)	1280

Priced System Configuration



System Components	Qty	Client Description	Qty	Server Description
Processors	2	IBM RS/6000 C10	1	IBM RS/6000 R24
Memory		256 MB	1	1,792 MB Main Memory
Disk Controllers		Integrated SCSI	14	SCSI-2 Adapter
Disk Drives	2	1.0 GB SCSI Disks	66	2.0 GB SCSI-2 Disks
Total GB of Storage	2	1 GB each client		123.75 GB (formatted capacity)
Tape Drives			1	8 mm 5GB Cartridge Tape
Terminals	2	Liberty Freedom One consoles	1	console
Terminal Connectivity	17	Emulex Terminal Server(with spare)		Direct Connect

Figure 5. Executive Summary (page 1) for DB2 TPC-C on RS/6000 R24

2.2 SPEC Benchmark

SPEC[1] is a IBM internal single-user performance benchmark. The benchmark can be run in either stand-alone or client/server environments. It contains 42 queries, which are run one at a time via static SQL.

The following contain SPEC results and descriptions:

- Figure 6 summarizes the queries and rows processed by each.
- Figure 7 contains the results in various scenarios with a local client, and OS/2 results.
- Figure 8 on page 21 contains the results in various scenarios with a client/server configuration.
- A.1, "SPEC" on page 101 contains query syntax, database characteristics, DBM and DB configuration parameters, and Explain output.

The AIX measurements were taken on machine SERVER 3 (see B.3, "Configuration for machine SERVER 3 (AIX)" on page 104) and the client, when used, was SERVER 2 (see B.2, "Configuration for machine SERVER 2 (AIX)" on page 104).

The OS/2 measurements were taken on machine SERVER 4 (see B.4, "Configuration for machine SERVER 4 (OS/2)" on page 104), and the results are in the last two columns of Figure 7.

	Query	Rows Processed
1	Select Last Record	1
2	Select Last Record using index	1
3	Select And Sort	20
4	Select And Sort using index	20
5	Select Complex	20
6	Select Complex using index	20
7	Join Two Tables	29991
8	Join Two Tables using index	29991
9	Select From View	20
10	Select From View using index	20
11	Select Count	1, COUNT=20
12	Select Count using index	1, COUNT=20
13	Select And Sort 99	99
14	Simple Select IN Predicate	5
15	Select/Sort IN Pred, Subquery	10
16	Select/Sort, field1 IN Predicate	1001
17	Select/Sort, field2 IN predicate	1000
18	Select/Sort 3 table Join	9999
19	Sel/Sort, fld1, IN Pred 3-tbl join using index	1001
20	Select/Sort, 4-table Join	9999
21	Select/sort, IN Pred, 3-table join (no index)	10000
22	Sort Tablea on field4	10000
23	Select All From Tablea	10000
24	Select with BETWEEN	1000
25	Select with LIKE	0
26	Select with DATE	0
27	Select with DECIMAL * FLOAT operation	4
28	Select with INTEGER * INTEGER operation	4
29	Update without index	20
30	Update with index	20
31	Update 999 records	999
32	Insert from Tablej (field12)	10
33	Insert from Tablej (field3)	10
34	Insert 990 without index	990
35	Insert 990 with index	990
36	Delete from TableJ (field12)	10
37	Delete from Tablej (field3)	10
38	Delete 1990 records	1990
39	Reorganize Tablea	10000
40	RUNSTATS on Tablea	10000
41	Export from Tablea	10000
42	Import into Tablea	10000

Figure 6. Summary of SPEC Benchmark Queries

[1] Despite the name, SPEC is not affiliated with the SPEC (System Performance Evaluation Cooperative) consortium.

Figure 7. LOCAL SPEC Measurements for AIX and OS/2 (elapsed times in seconds)

Platform BUFFPAGE Isolation Blocking	AIX, DB2 V1.2 24MB RR UNAMBIG	AIX, DB2 V2.1 24MB RR UNAMBIG	AIX, DB2 V2.1 24MB RS UNAMBIG	AIX, DB2 V2.1 24MB CS UNAMBIG	AIX, DB2 V2.1 24MB UR UNAMBIG	OS/2, DB2 V1.2 20MB CS UNAMBIG	OS/2, DB2 V2.1 20MB CS UNAMBIG
1	6.08	7.88	9.05	8.99	7.83	24.68	17.40
2	0.19	0.23	0.23	0.24	0.24	0.53	0.47
3	9.99	7.15	12.21	12.13	7.16	24.29	21.91
4	0.65	0.51	0.50	0.51	0.50	2.19	0.97
5	0.65	6.79	10.48	10.45	6.83	23.88	19.43
6	0.50	0.52	0.52	0.52	0.54	1.10	0.94
7	54.93	39.79	40.58	42.00	38.66	97.00	68.56
8	77.24	39.55	40.61	41.76	38.59	80.29	68.03
9	6.92	7.85	9.63	9.53	7.88	23.84	17.44
10	0.48	0.55	0.53	0.53	0.53	2.06	0.94
11	7.04	6.12	10.08	10.01	6.18	23.78	18.50
12	0.44	0.22	0.22	0.22	0.22	1.10	0.41
13	1.52	0.82	0.82	0.82	0.86	3.12	1.75
14	0.22	0.30	0.30	0.31	0.30	0.75	0.50
15	0.33	0.30	0.30	0.31	0.30	0.97	0.57
16	37.47	5.12	5.59	5.86	4.73	46.84	15.63
17	41.01	4.81	5.69	5.93	4.71	50.94	59.50
18	31.01	20.50	22.60	23.85	20.53	59.50	33.78
19	191.13	5.44	6.44	6.72	5.66	182.46	10.35
20	34.27	25.99	28.59	29.89	26.20	59.84	36.97
21	85.29	32.62	35.13	36.50	32.37	102.97	54.03
22	14.76	9.26	9.75	9.84	9.26	17.41	15.81
23	12.65	9.63	10.06	10.16	9.61	9.00	14.25
24	2.11	1.61	1.59	1.63	1.57	3.50	2.66
25	0.88	1.00	1.09	1.08	1.00	4.35	2.06
26	0.78	1.01	1.32	1.30	1.02	3.78	2.22
27	1.46	1.18	1.67	1.66	1.12	3.65	2.78
28	1.22	1.03	1.24	1.23	1.02	4.54	2.50
29	7.39	7.81	9.54	9.42	9.54	23.97	19.93
30	0.47	0.52	0.51	0.52	0.52	1.32	0.88
31	2.63	2.38	2.36	2.36	2.36	6.03	3.75
32	6.75	7.99	9.25	9.15	7.90	23.97	17.35
33	0.49	0.47	0.46	0.47	0.46	1.19	0.72
34	3.28	1.36	1.39	1.41	1.35	9.62	5.50
35	7.22	5.27	5.14	5.32	5.16	14.87	10.85
36	6.48	8.06	9.45	9.31	9.35	24.13	17.81
37	0.49	0.60	0.60	0.61	0.60	1.75	1.03
38	6.50	6.62	6.65	6.61	6.66	10.44	9.53
39	21.38	8.86	8.90	8.79	8.79	43.69	22.85
40	3.52	3.74	3.71	3.71	3.73	16.54	6.35
41	16.60	12.76	12.84	13.08	12.74	28.65	22.71
42	79.07	63.46	63.23	63.53	62.92	129.31	111.15
Arith. Mean	18.65	8.75	9.54	9.72	8.75	28.43	16.58
A.Mean Impr. in V2.1		53%					42%
Geo. Mean	4.24	3.203	3.514	3.558	3.210	10.54	6.43
G.Mean Impr. in V2.1		24%					39%

Figure 8. CLIENT/SERVER SPEC Measurements for AIX (elapsed times in seconds)				
Platform **BUFFPAGE** **Isolation** **Blocking**	**AIX, DB2 V1.2** **24MB** **RR** **UNAMBIG**	**AIX, DB2 V2.1** **24MB** **RR** **UNAMBIG**	**AIX, DB2 V2.1** **24MB** **RR** **ALL**	**AIX, DB2 V2.1** **24MB** **RR** **NO**
1	6.06	7.98	7.99	7.90
2	0.19	0.25	0.26	0.24
3	9.84	7.36	7.87	7.34
4	0.53	0.52	0.53	0.59
5	7.18	6.49	6.41	6.27
6	0.48	0.53	0.52	0.61
7	74.11	87.97	88.40	152.55
8	95.77	88.09	89.27	148.79
9	6.59	8.10	7.96	7.66
10	0.48	0.54	0.56	0.62
11	6.69	6.24	6.58	6.37
12	0.44	0.22	0.21	0.23
13	1.39	1.25	0.96	1.24
14	0.26	0.30	0.31	0.33
15	0.35	0.32	0.28	0.32
16	36.87	5.27	5.66	8.90
17	40.09	5.46	5.43	8.97
18	30.25	23.26	23.23	58.70
19	197.23	6.18	6.17	9.45
20	32.04	28.20	28.85	61.31
21	89.06	39.38	39.21	70.98
22	16.23	15.89	15.58	50.62
23	14.26	13.62	13.74	49.35
24	2.20	2.13	2.01	5.52
25	0.79	1.04	1.03	1.03
26	0.76	1.04	1.04	1.02
27	1.40	1.21	1.25	1.18
28	1.10	1.06	1.04	1.05
29	6.51	7.82	7.77	7.76
30	0.46	0.55	0.54	0.55
31	2.63	2.65	2.40	2.40
32	6.02	7.91	7.92	8.08
33	0.48	0.49	0.48	0.48
34	3.32	1.41	1.40	1.35
35	7.05	5.85	5.27	5.51
36	6.18	8.15	8.10	8.04
37	0.51	0.66	0.62	0.62
38	6.51	6.70	6.93	6.68
39	22.73	9.24	9.15	9.08
40	3.66	3.79	3.86	3.78
41	19.50	24.71	24.58	24.28
42	85.37	96.89	99.79	96.27
Arith. Mean	20.09	12.783	12.888	20.099
A.Mean Impr. in V2.1		36%		
Geo. Mean	4.52	3.676	3.633	4.462
G.Mean Impr. in V2.1		19%		

SPEC results have greatly improved since V1.2. Among the various AIX and OS/2, and local and client/server tests, the V2.1 arithmetic and geometric means of elapsed times ranged from 31% to 55% lower (1.45 to 2.20 times faster). See the bottom of Figure 7 on page 19 and Figure 8 on page 21 for all the improvement percentages. The benchmark was run on the same machines and in exactly the same way on both versions of DB2.

The most significant differences in performance arise when DB2 V2.1.0 picks a different plan from DB2 V1.2. The explain outputs for these queries are located in A.1.6, "SPEC Explain Output" on page 101, and you may want to compare these to the DB2 V1.2 plans in *DB2/6000 V1.2 Performance Report*.

There are many different improvements to the old plans. The following is a discussion of plan changes for queries #8,12,19.

Figure 9. SPEC query performance V1.2 and V2.1

SPEC query #	Version 1.2	Version 2.1	V1.2/V2.1
8	77.24 sec	38.46 sec	2.01
12	0.44 sec	0.22 sec	2.00
19	191.13 sec	5.72 sec	33.41 times faster!

```
Section = 8


SQL Statement:

  SELECT *
  FROM TABLEB, TABLEC
  WHERE TABLEB.FIELD2 = TABLEC.FIELD3


Access Table Name = SPEC.TABLEC   ID = 18   #Columns = 12
  Scan Direction   = Forward
  Relation Scan
  Lock Share
Nested Loop Join
  Access Table Name = SPEC.TABLEB   ID = 17   #Columns = 12
    Scan Direction   = Forward
    Index Scan:   Name = SPEC.INDXB2   ID = 1   #Key Columns = 1
    Lock Intent Share
```

Figure 10. V1.2 Explain for SPEC query 8.

```
SQL Statement:

  SELECT *
  FROM TABLEB, TABLEC
  WHERE TABLEB.field2 = TABLEC.field3


Access Table Name = SPEC.TABLEB  ID = 3
|  #Columns = 12
|  Relation Scan
|  |  Prefetch Eligible
|  Lock Intents
|  |  Table: Share
|  |  Row   : None
|  Create/Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 12
|  |  #Sort Key Columns = 1
|  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 12
|  Relation Scan
|  |  Prefetch Eligible
Merge Join
|  Data Stream 1: Evaluate at Open
|  |  Not Piped
|  |  Access Table Name = SPEC.TABLEC  ID = 4
|  |  |  #Columns = 12
|  |  |  Relation Scan
|  |  |  |  Prefetch Eligible
|  |  |  Lock Intents
|  |  |  |  Table: Share
|  |  |  |  Row   : None
|  |  |  Create/Insert Into Sorted Temp Table  ID = t2
|  |  |  |  #Columns = 12
|  |  |  |  #Sort Key Columns = 1
|  |  |  |  Not Piped
|  |  Sorted Temp Table Completion  ID = t2
|  End of Data Stream 1
|  Access Temp Table  ID = t2
|  |  #Columns = 12
|  |  Relation Scan
|  |  |  Prefetch Eligible
End of Section
```

Figure 11. V2.1 Explain for SPEC query 8.

For query #8, the plans differ in three major ways::

a) V1.2 does a Nested-Loop Join to join the two tables, and V2.1 does a Merge Join instead.

b) There is a different Join order.

c) There is a different access method.

And most of the performance improvement comes from the decision by the V2.1 optimizer to drop the index scan on TABLEB: it chose instead to just load both tables into memory and to sort them. This is the faster method, because the TABLEB index for FIELD2 is totally unclustered, and an index scan would cause pages to be read from disk in a random order, causing a lot of disk head movement. By loading all the data directly into memory, the disk head just moves in one direction and the process is much faster.

```
Section = 12


SQL Statement:

  SELECT COUNT(*) INTO :numrows
  FROM TABLEJ
  WHERE FIELD3 = 770 OR FIELD3 = 990


Access Table Name = SPEC.TABLEJ  ID = 23  #Columns = 1
  Scan Direction  = Forward
  Index Scan:  Name = SPEC.INDXJ3  ID = 3  #Key Columns = 1
    Index-only Access
  Lock Intent Share
  Sargable Predicate(s)
    #Predicates = 1
    Create/Insert Into Sorted Temp Table  ID = t8
      Sort  #Columns = 1
      Piped
      Duplicate Reduction
Sorted Temp Table Completion  ID = t8
Access Table Name = SPEC.TABLEJ  ID = 23  #Columns = 1
  Scan Direction  = Forward
  Index Scan:  Name = SPEC.INDXJ3  ID = 3  #Key Columns = 1
    Index-only Access
  Lock Intent Share
  Sargable Predicate(s)
    #Predicates = 1
    Insert Into Sorted Temp Table  ID = t8
Sorted Temp Table Completion  ID = t8
Index OR Filter
Access Table Name = SPEC.TABLEJ  ID = 23  #Columns = 1
  Scan Direction  = Forward
  Fetch Direct
  Lock Share
  Residual Predicate(s)
    #Predicates = 1
    Predicate Aggregation
      Column Function(s)
Aggregation Completion
  Column Function(s)
```

Figure 12. V1.2 Explain for SPEC query 12.

```
SQL Statement:

  SELECT COUNT(*) INTO :numrows
  FROM TABLEJ
  WHERE field3 = 770 or field3 = 990


Table Constructor
|  2-Row(s)
Nested Loop Join
|  Access Table Name = SPEC.TABLEJ  ID = 9
|  |  #Columns = 1
|  |  Index Scan:  Name = SPEC.INDXJ3  ID = 3
|  |  |  #Key Columns = 1
|  |  |  Index-only Access
|  |  |  |  Predicate Aggregation
|  |  |  |  |  Column Function(s)
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row  : Share
Aggregation Completion
|  Column Function(s)
End of Section
```

Figure 13. V2.1 Explain for SPEC query 12.

For query #12, the plans are different, but the work is similar. In V1.2, the optimizer decided to run two separate index scans, produce two sets of RIDs, and join them together (Index OR). In V2.1, the optimizer created a two-row table containing the values 990,770, and with a Nested-Loop Join found all the matches in TABLEJ. Because of the nature of Nested-Loop Joins, this amounts to the same work (i.e. two index scans) but avoids much of the overhead. The savings come from not having to to put the RIDs into a temporary table and sort them.

```
Section = 19


 SQL Statement:

   SELECT DISTINCT FIELD1, FIELD2, FIELD3, FIELD4, FIELD5, FIELD6
   FROM TABLEA
   WHERE FIELD1 IN (
      SELECT TABLEA.FIELD3
      FROM TABLEA, TABLEG, TABLEH
      WHERE TABLEA.FIELD3 = TABLEG.FIELD3 AND TABLEA.FIELD3 =
              TABLEH.FIELD3 )
   GROUP BY FIELD1, FIELD2, FIELD3, FIELD4, FIELD5, FIELD6
   ORDER BY TABLEA.FIELD5


 Access Table Name = SPEC.TABLEA  ID = 16  #Columns = 1
   Scan Direction  = Forward
   Index Scan:  Name = SPEC.INDXA3  ID = 3  #Key Columns = 0
     Index-only Access
     Sargable Index Predicate(s)
       #Predicates = 1
       Create/Insert Into Temp Table  ID = t20
   Lock Share
 Temp Table Completion  ID = t20
 Access Table Name = SPEC.TABLEG  ID = 20  #Columns = 1
   Scan Direction  = Forward
   Index Scan:  Name = SPEC.INDXG3  ID = 3  #Key Columns = 0
     Index-only Access
   Lock Share
 Merge Join
   Access Temp Table  ID = t20  #Columns = 1
     Scan Direction  = Forward
     Relation Scan
     Residual Predicate(s)
       #Predicates = 1
       Create/Insert Into Sorted Temp Table  ID = t21
         Sort  #Columns = 1
         Not Piped
 Sorted Temp Table Completion  ID = t21
 Access Table Name = SPEC.TABLEH  ID = 21  #Columns = 1
   Scan Direction  = Forward
   Index Scan:  Name = SPEC.INDXH3  ID = 3  #Key Columns = 0
     Index-only Access
   Lock Share

 Merge Join
   Access Temp Table  ID = t21  #Columns = 1
     Scan Direction  = Forward
     Relation Scan
 Distinct Filter  #Columns = 1
 Create/Insert Into Temp Table  ID = t22
 Access Table Name = SPEC.TABLEA  ID = 16  #Columns = 6
   Scan Direction  = Forward
   Index Scan:  Name = SPEC.INDXA1  ID = 1  #Key Columns = 1
   Lock Intent Share
   Residual Predicate(s)
     #Predicates = 2
     ANY Ordered
       Access Temp Table  ID = t22  #Columns = 1
         Scan Direction  = Forward
         Relation Scan
     Create/Insert Into Sorted Temp Table  ID = t23
       Sort  #Columns = 6
       Piped
       Duplicate Reduction
 Sorted Temp Table Completion  ID = t23
 Access Temp Table  ID = t23  #Columns = 6
   Scan Direction  = Forward
   Relation Scan
   Residual Predicate(s)
     #Predicates = 1
     Predicate Aggregation
       Group By
 Aggregation Completion
   Group By
```

Figure 14. V1.2 Explain for SPEC query 19.

```
SQL Statement:

  Select distinct field1, field2, field3, field4, field5, field6
  from tablea
  where field1 in (
     select tablea.field3
     from tablea, tableg, tableh
     where tablea.field3 = tableg.field3 and tablea.field3 =
             tableh.field3 )
  group by field1, field2, field3, field4, field5, field6
  order by tablea.field5


Access Table Name = SPEC.TABLEA  ID = 2
|  #Columns = 6
|  Relation Scan
|  |  Prefetch Eligible
|  Lock Intents
|  |  Table: Share
|  |  Row   : None
|  Create/Insert Into Sorted Temp Table  ID = t1
|  |  #Columns = 7
|  |  #Sort Key Columns = 1
|  |  Piped
Sorted Temp Table Completion  ID = t1
Access Temp Table  ID = t1
|  #Columns = 7
|  Relation Scan
|  |  Prefetch Eligible
Nested Loop Join
|  Access Table Name = SPEC.TABLEG  ID = 6
|  |  #Columns = 1
|  |  Single Record
|  |  Index Scan:  Name = SPEC.INDXG3  ID = 3
|  |  |  #Key Columns = 1
|  |  |  Index-only Access
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row   : Share
Nested Loop Join
|  Access Table Name = SPEC.TABLEH  ID = 7
|  |  #Columns = 1
|  |  Single Record
|  |  Index Scan:  Name = SPEC.INDXH3  ID = 3
|  |  |  #Key Columns = 1
|  |  |  Index-only Access
|  |  |  Sargable Index Predicate(s)
|  |  |  |  #Predicates = 1
|  |  Lock Intents
|  |  |  Table: Intent Share
|  |  |  Row   : Share
Merge Join
|  Data Stream 1: Evaluate at Open
|  |  Not Piped
|  |  Access Table Name = SPEC.TABLEA  ID = 2
|  |  |  #Columns = 1
|  |  |  Index Scan:  Name = SPEC.INDXA3  ID = 3
|  |  |  |  #Key Columns = 0
|  |  |  |  Index-only Access
|  |  |  |  |  Create/Insert Into Temp Table  ID = t2
|  |  |  |  |  |  #Columns = 1
|  |  |  Lock Intents
|  |  |  |  Table: Share
|  |  |  |  Row   : None
|  |  Temp Table Completion  ID = t2
|  End of Data Stream 1
|  Access Temp Table  ID = t2
|  |  #Columns = 1
|  |  Single Record
|  |  Relation Scan
|  |  |  Prefetch Eligible
|  |  Residual Predicate(s)
|  |  |  #Predicates = 1
Create/Insert Into Sorted Temp Table  ID = t3
|  #Columns = 6
|  #Sort Key Columns = 6
|  Piped
Access Temp Table  ID = t3
|  #Columns = 6
|  Relation Scan
|  |  Prefetch Eligible
Aggregation
|  Group By
End of Section
```

Figure 15. V2.1 Explain for SPEC query 19.

In query #19, there are several major differences between the V1.2 and V2.1 plans. All these differences combine to produce a huge performance boost.

a) When joining tables TABLEA and TABLEG, the V2.1 optimizer notices that the index on TABLEA.FIELD3 is totally unclustered, so it decides to sort the data instead of index-scanning to save on disk-head movement (see discussion on query 8).

b) In V2.1, the optimizer joins TABLEA and TABLEG with a Nested-Loop Join, so it doesn't have to create a temporary table. This performance improvement is visible once again when the result table is joined with TABLEH. Again, a Nested-Loop Join is preferred.

c) The V1.2 optimizer, after joining the two tables TABLEA and TABLEG, sorts the resulting set to prepare for the Merge Join. This is unnecessary, because the result set of the Merge Join was already ordered. Maybe you didn't see it, and the V1.2 optimizer didn't see it, but the V2.1 optimizer sure did!

d) Finally, at the highest level Join, the old optimizer ran a subquery Join, just like the SQL statement says. The new optimizer created a temporary table with the six needed columns, and Merge-Joined this table to the previous set.

2.3 RQW Benchmark

RQW is an IBM-defined set of 30 moderately complex queries used primarily for release-to-release query performance comparisons. On average, this benchmark has improved by 46% over DB2 V1.2; i.e., the average query runs in just over half the time it does in V1.2. The syntax of the queries is in A.2.1, "RQW Query Descriptions" on page 101, while the database is described in A.2.2, "Catalog Statistics for RTW Database" on page 102.

The queries were run one at a time via static SQL in a C program, with no other users or activity on the system. After each query a CONNECT RESET and CONNECT were done to cause the buffer pool to be deallocated and allocated; i.e., to minimize performance (I/O) impacts between queries.

The other pertinent configuration information is:

- Default blocking (UNAMBIG).
- Default isolation level (CS).
- Default optimization class (5).
- Default DBM and DB configuration parameters, except for:

  STAT_HEAP_SZ=1096 MAXLOCKS=100 MAXAPPLS=1

The following elapsed time measurements were taken locally on the machine described in B.7, "Configuration for CB2 benchmark machine (AIX)" on page 104.

Query	DB2/6000 V1.2	DB2/6000 V2.1	Ratio V2/V1
1	2.492	2.216	0.889
2	0.253	0.492	1.946
3	0.246	0.344	1.397
4	8.879	7.186	0.809
5	1.148	0.907	0.791
6	4.202	2.922	0.695
7	0.196	0.307	1.567
8	13.735	5.624	0.409
9	2.616	2.070	0.791
10	1.474	1.102	0.748
11	5.664	3.806	0.672
12	5.558	4.331	0.779
13	0.822	0.728	0.886
14	0.211	0.331	1.571
15	12.484	6.537	0.524
16	38.478	16.669	0.433
17	6.038	1.527	0.253
18	0.262	0.434	1.656
19	3.678	2.236	0.608
20	11.978	14.669	1.225
21	1.066	0.901	0.845
22	0.202	0.367	1.820
23	86.559	33.087	0.382
24	12.896	8.384	0.650
25	9.235	5.033	0.545
26	5.430	0.416	0.077
27	2.631	1.299	0.494
28	15.773	12.438	0.789
29	5.019	1.965	0.391
30	1.103	1.433	1.299
Total	260.328	139.761	
Arith.Mean	8.68	4.66	<--- 46% reduction in V2.1
Geo.Mean	2.84	2.06	<--- 27% reduction in V2.1

Figure 16. RQW Results

In the cases where V2.1 is slower than V1.2, much of the extra overhead is in loading additional table and index information into memory (the database heap; viz. the dbheap configuration parameter). This loading only takes place upon the first reference to each table, and because these runs were done with the database started up for each query, the overhead would not normally be seen after the first few queries are executed against a production database.

2.4 Customer Benchmarks

2.4.1 Customer Benchmark 1 (CB1)

CB1 is a customer-defined set of 30 moderately complex queries used primarily to test interactive query performance. The database is 2.2GB in size. The benchmark was run on the machine described in B.6, "Configuration for CB1 benchmark machine (AIX)" on page 104.

The queries were run one at a time via static SQL in a C program, with no other users or activity on the system. After each query a CONNECT RESET and CONNECT were done to cause the DB2 V2.1.0 buffer pool to be deallocated and allocated; i.e., to minimize performance (I/O) impacts between queries.

The other pertinent configuration information is:

- Default blocking (UNAMBIG).
- Isolation level RR.
- Default optimization class (5).
- Default DBM and DB configuration parameters, except for:

```
SHEAPTHRES=12000    DBHEAP=512      BUFFPAGE=11250    LOCKLIST=2500
SORTHEAP=2250       STMTHEAP=512    APPLHEAPSZ=512    PCKCACHESZ=20
STAT_HEAP_SZ=1096   MAXLOCKS=100    MAXAPPLS=1
```

As shown in Figure 18 on page 33, the queries in CB1 are long running, with an average elapsed time of 55 minutes in V1.2. With V2.1, the average elapsed time dropped to 19 minutes, a reduction of 65%; i.e., the queries now run almost 3 times faster. Much of the improvement can be attributed to V2.1′s new SQL compiler, which has its greatest impact on complex queries.

2.4.1.1 CB1 Query Analysis

The following list describes the headings used in Figure 17.

Query No.	Query Number
Tables In Main Query Block	# of tables in the main query block
Join Predicates (Explicit)	# of join predicates (explicit)
Local Predicates	# of local predicates
Group By Columns	# of group by columns
Having Clause Predicates	# of having clause predicates
Order By Columns	# of order by columns
Cols Fns (MIN, MAX, etc)	# of column functions (MIN,MAX etc.)
Cols Fns w/Distinct	# of column functions with distinct (SUM(DISTINCT C1))
Distinct Occurrences	# of DISTINCT occurrences
Unions	# of UNIONS
Uncorr Scalar Subqueries	# of Uncorrelated Scalar Subqueries
Scalar (1-row) Subqueries	# of Scalar (1-row) Subqueries
Uncorr Subqueries	# of Uncorrelated Subqueries
Corr Subqueries	# of Correlated Subqueries

2.4.1.2 Miscellaneous Definitions

Correlated Subquery A subquery (part of a WHERE or HAVING clause) which must be evaluated for each row of the outer query block since it references columns not defined in the subquery.

Scalar Subquery A subquery (part of a WHERE or HAVING clause) which returns at most a single value.

Figure 17 (Page 1 of 2). CB1 Query Analysis

Query No.	Tables In Main Query Block	Join Predi- cates (Explicit)	Local Predi- cates	Group By Columns	Having Clause Predi- cates	Order By Columns	Col Fns (MIN, MAX, etc)	Col Fns w/Distinct	Distinct Occur- rences	Unions	Uncorr Scalar Sub- queries	Scalar (1-row) Sub- queries	Uncorr Sub- queries	Corr Sub- queries
Original Queries														
1	1						1							
2	1							1						
3	3	2	5				1	1						
4	7	10	8			2								
5	7	7	4			3								
6	8	7	6	6		6	1							
7	6	7	3	6		6	1							
8	1		4	1			2			1			1	
9	1			2	1	2	1							
10	4	3	4	5		5								
11	2	1	1											
12	1		1											
13	1		1										1	
14	N/A													
15	N/A													
16	3	2				1								
17	2	1	1			2								
18	2	1	1	3		3	1							
19	4	4	1			1								
20	3	5		2		2	1							
21	5	5	4			3								
22	1	1	1	2			3			2				
23	1		1				1	1						
24	Deleted													
25	6	8	2			4								
26	3	3	3	7										
27	6	10	7			2								

Figure 17 (Page 2 of 2). CB1 Query Analysis

Query No.	Tables In Main Query Block	Join Predi-cates (Explicit)	Local Predi-cates	Group By Columns	Having Clause Predi-cates	Order By Columns	Col Fns (MIN, MAX, etc)	Col Fns w/Distinct	Distinct Occur-rences	Unions	Uncorr Scalar Sub-queries	Scalar (1-row) Sub-queries	Uncorr Sub-queries	Corr Sub-queries
Original Queries														
28	Deleted													
29	7	8	7			4								
30	8	9	3			3								
31	2	1	1						1					
32	1		1						1				1	
33	1	1	1											1
34	1		1										1	

2.4.1.3 CB1 Results

Query	DB2 V1.2 Total Time	DB2 V2.1 Total Time	Ratio V2/V1
1	49.556	64.657	1.305
2	97.361	106.559	1.094
3	919.287	195.200	0.212
4	130.540	9.601	0.074
5	627.534	197.354	0.314
6	3187.798	1051.922	0.330
7	49011.172	16152.632	0.330
8	1683.943	52.242	0.031
9	798.800	362.787	0.454
10	157.074	83.480	0.531
11	1286.748	887.039	0.689
12	494.229	2.620	0.005
13	1014.518	607.378	0.599
16	2878.132	1979.783	0.688
17	2033.858	92.970	0.046
18	371.067	159.692	0.430
19	1609.139	1751.127	1.088
20	5419.433	4359.441	0.804
21	15720.195	8180.867	0.520
22	14.113	10.199	0.723
23	912.428	576.975	0.632
25	867.234	138.475	0.160
26	984.165	466.633	0.474
27	3284.076	1275.344	0.388
29	193.399	5.334	0.028
30	2125.755	33.263	0.016
31	36.145	19.571	0.541
32	213.402	1.418	0.007
33	1816.273	1.387	0.001
34	201.266	1.363	0.007
Total	98138.640	38827.313	
Arith.Mean	3271.290	1294.240	<---- 65% reduction in V2.1
Geo.Mean	734.750	124.970	<---- 83% reduction in V2.1

Figure 18. CB1 Results for DB2 V1.2 vs. V2.1

2.4.2 Customer Benchmark 2 (CB2)

CB2 is a customer-defined set of 203 simple, short-running queries used primarily to test interactive query performance. The benchmark was run on the CB2 machine, as described in B.7, “Configuration for CB2 benchmark machine (AIX)” on page 104.

The queries were run one at a time via dynamic SQL, with no other users or activity on the system. After each query a CONNECT RESET and CONNECT were done to cause the DB2 V2.1.0 buffer pool to be deallocated and allocated; i.e., to minimize performance (I/O) impacts between queries.

The other pertinent configuration information is:

- Default blocking (UNAMBIG).
- Default isolation level (CS).
- Default optimization class (5).
- Default DBM and DB configuration parameters, except for:

  APPLHEAPSZ=96 STAT_HEAP_SZ=8030 MAXLOCKS=100 MAXAPPLS=1

- Query and table characterstics are given in A.7, “Customer Benchmark 2” on page 102.

The arithmetic mean of elapsed times was reduced by 44% in V2.1, while the geometric mean increased by 4%. The total “execution” time (the total elapsed time of the queries, not counting the Prepare times) decreased by 49% in arithmetic mean and 17% in geometric mean.

Comparing the execution times and the elapsed times (see the totals at the end of Figure 19 on page 35), we see that the total Prepare time increased from about 18 secs. in V1.2 to about 61 secs. in V2.1. This reflects the increased complexity of the V2 compiler, (as well as larger overhead to load table information, as explained in 2.3, “RQW Benchmark” on page 28).

Lowering the optimization class is a way to reduce Prepare times. In one experiment with this particular benchmark (not in this report), there were a few queries that had much better access plans with the default optimization class than a lower one, and thus the average execution time suffered when the class was lowered. In general, however, very simple queries will have shorter Prepare and elapsed times with a lower optimization class. See 3.5, “Optimization Classes” on page 54 for more information on optimization classes.

Query	Total Time DB2 V1.2 Total Time	Total Time DB2 V2.1 Total Time	Ratio V2/V1	Execution Time DB2 V1.2 Exec Time	Execution Time DB2 V2.1 Exec Time	Ratio V2/V1
1	0.4326	0.4596	1.0623	0.3215	0.3290	1.0232
2	7.7932	6.2683	0.8043	7.7082	5.9720	0.7748
3	24.8306	17.1555	0.6909	24.7469	16.8535	0.6810
4	2.2016	2.0346	0.9241	2.1191	1.7393	0.8207
5	0.8601	0.7492	0.8711	0.7265	0.4645	0.6393
6	2.6278	5.6356	2.1446	2.5476	5.3606	2.1042
7	0.4046	0.4986	1.2324	0.3245	0.2231	0.6876
8	15.6122	9.7882	0.6270	15.5244	9.4797	0.6106
9	0.4758	0.7220	1.5176	0.3899	0.4456	1.1428
10	0.4579	0.6745	1.4728	0.3748	0.3923	1.0468
11	8.0340	6.3490	0.7903	7.9533	6.0755	0.7639
12	2.6153	3.6079	1.3795	2.5340	3.3281	1.3134
13	1.0268	0.6998	0.6815	0.9472	0.4249	0.4486
14	0.5577	0.8376	1.5017	0.4746	0.5474	1.1535
15	2.6798	5.9338	2.2143	2.5975	5.6538	2.1766
16	1.9490	2.3345	1.1978	1.8688	2.0576	1.1010
17	7.6368	8.1258	1.0640	7.4607	7.1507	0.9585
18	1.0228	2.0423	1.9967	0.9436	1.6968	1.7983
19	2.2440	2.6528	1.1822	2.1625	2.3685	1.0952
20	8.1041	6.1578	0.7598	8.0133	5.8835	0.7342
21	0.3810	0.4812	1.2631	0.2951	0.2034	0.6891
22	0.8589	0.7200	0.8383	0.7781	0.4394	0.5647
23	0.7316	0.8507	1.1628	0.6485	0.5619	0.8666
24	1.1273	5.5400	4.9142	1.0478	5.2620	5.0221
25	0.5450	0.6116	1.1223	0.4612	0.3253	0.7053
26	0.1702	0.3518	2.0672	0.0911	0.0821	0.9005
27	0.3704	0.4858	1.3116	0.2898	0.2093	0.7222
28	0.3982	0.6014	1.5103	0.3131	0.2812	0.8979
29	3.1314	1.5549	0.4966	3.0512	1.2837	0.4207
30	1.3437	1.6701	1.2429	1.2595	1.3878	1.1019
31	1.1824	5.1492	4.3547	1.1028	4.8205	4.3710
32	0.4037	0.4325	1.0713	0.2809	0.1618	0.5759
33	4.1255	4.0458	0.9807	4.0226	3.6787	0.9145
34	0.7587	0.9189	1.2112	0.6743	0.6428	0.9534
35	1.3627	1.8423	1.3520	1.2879	1.5831	1.2292
36	0.5053	0.6122	1.2115	0.4268	0.3435	0.8048
37	0.4368	0.4712	1.0788	0.3149	0.1967	0.6247
38	6.0132	4.1333	0.6874	5.9326	3.8526	0.6494
39	0.5817	0.9202	1.5820	0.5001	0.6419	1.2836
40	5.7465	3.4183	0.5949	5.6670	3.1400	0.5541
41	0.2453	0.4197	1.7108	0.1641	0.1472	0.8970
42	5.7288	5.8479	1.0208	5.6142	5.3489	0.9527
43	4.6801	1.9844	0.4240	4.5964	1.7144	0.3730
44	0.1581	0.4753	3.0057	0.0712	0.1096	1.5398
45	9.3082	6.0638	0.6514	9.2277	5.7881	0.6273
46	0.4610	0.8338	1.8085	0.3786	0.5135	1.3563
47	1.0954	1.1829	1.0799	0.9984	0.8890	0.8904
48	0.4583	0.5927	1.2932	0.3727	0.3081	0.8268
49	5.7129	4.4411	0.7774	5.6144	4.1418	0.7377
50	1.8374	2.2622	1.2312	1.7559	1.9478	1.1093
51	8.5635	8.8852	1.0376	8.4682	8.5072	1.0046
52	7.9724	2.3555	0.2955	7.8914	2.0855	0.2643
53	0.1782	0.4833	2.7114	0.0947	0.2066	2.1820
54	0.5247	0.6367	1.2135	0.4447	0.3581	0.8052
55	1.7742	4.3746	2.4657	1.6861	4.0913	2.4264
56	0.9067	1.0054	1.1088	0.8259	0.7207	0.8727
57	0.4982	0.6376	1.2799	0.4005	0.3409	0.8511
58	5.7401	4.3615	0.7598	5.6605	4.0777	0.7204
59	0.3685	0.4786	1.2986	0.2901	0.2047	0.7055
60	25.5270	20.1328	0.7887	25.4178	19.7423	0.7767
61	8.0164	2.3329	0.2910	7.9365	2.0613	0.2597
62	0.1657	0.3732	2.2523	0.0813	0.0923	1.1349
63	8.8982	6.4536	0.7253	8.8173	6.1726	0.7001
64	10.0002	6.3588	0.6359	9.9105	6.0754	0.6130
65	0.9863	1.0510	1.0656	0.9061	0.7601	0.8389
66	0.6266	0.6866	1.0956	0.5252	0.3891	0.7408
67	0.3448	0.4522	1.3115	0.2671	0.1805	0.6758
68	0.3709	0.4933	1.3301	0.2922	0.2187	0.7485
69	0.8308	0.9475	1.1404	0.7020	0.5331	0.7594
70	1.2043	1.9169	1.5918	1.1207	1.6377	1.4613

Figure 19 (Part 1 of 3). CB2 Results for DB2 V1.2 vs. V2.1

Query	Total Time DB2 V1.2 Total Time	Total Time DB2 V2.1 Total Time	Ratio V2/V1	Execution Time DB2 V1.2 Exec Time	Execution Time DB2 V2.1 Exec Time	Ratio V2/V1
71	0.2437	0.4803	1.9706	0.1368	0.1621	1.1855
72	0.9216	0.7699	0.8354	0.8423	0.4971	0.5902
73	0.3560	0.5939	1.6682	0.2715	0.3156	1.1625
74	1.0955	1.2629	1.1528	1.0004	0.9086	0.9082
75	48.4441	0.5910	0.0122	48.3624	0.2962	0.0061
76	3.2159	1.6093	0.5004	3.1356	1.3360	0.4261
77	1.9177	2.5608	1.3353	1.8370	2.2828	1.2427
78	29.1285	7.6809	0.2637	29.0405	7.2858	0.2509
79	0.5720	0.6324	1.1055	0.4900	0.3559	0.7264
80	0.3684	0.4781	1.2976	0.2870	0.1984	0.6913
81	1.0847	5.4783	5.0505	1.0032	5.2060	5.1892
82	0.4360	0.6851	1.5713	0.3501	0.4023	1.1491
83	1.0933	1.1251	1.0291	1.0094	0.8379	0.8301
84	49.0044	16.1349	0.3293	48.9224	15.8400	0.3238
85	3.1226	1.5827	0.5069	3.0424	1.3130	0.4316
86	0.4148	0.6518	1.5714	0.3322	0.3668	1.1041
87	1.5777	2.1122	1.3388	1.4816	1.7178	1.1594
88	0.4069	0.5017	1.2330	0.3226	0.2183	0.6768
89	0.3185	0.3826	1.2011	0.2066	0.1033	0.4998
90	5.4892	3.5713	0.6506	5.4058	3.2935	0.6093
91	0.4318	0.6634	1.5362	0.3489	0.3825	1.0963
92	1.7909	1.4986	0.8368	1.6963	1.2039	0.7097
93	0.6166	0.8589	1.3929	0.5210	0.5692	1.0924
94	4.4312	6.0063	1.3555	4.3527	5.7369	1.3180
95	0.2503	0.4386	1.7527	0.1719	0.1542	0.8972
96	0.9339	1.1983	1.2832	0.8392	0.8523	1.0156
97	1.8036	1.4397	0.7982	1.7157	1.1565	0.6741
98	0.2610	0.4668	1.7884	0.1794	0.1845	1.0281
99	34.0687	6.2612	0.1838	33.9883	5.9877	0.1762
100	36.7855	16.4157	0.4463	36.7004	16.1296	0.4395
101	1.6680	1.3858	0.8308	1.5753	1.0966	0.6961
102	1.4873	1.3248	0.8907	1.3883	1.0313	0.7429
103	4.6058	6.3354	1.3755	4.5270	6.0657	1.3399
104	1.9342	2.4044	1.2431	1.8490	2.1175	1.1452
105	1.3668	2.1432	1.5681	1.2728	1.8027	1.4163
106	0.5094	0.6289	1.2347	0.4296	0.3594	0.8366
107	0.5756	0.7810	1.3568	0.4925	0.4994	1.0141
108	35.4344	6.6332	0.1872	35.3540	6.3596	0.1799
109	2.6049	3.4076	1.3082	2.5233	3.1245	1.2383
110	1.1208	5.3243	4.7503	1.0400	5.0384	4.8448
111	37.6252	2.4989	0.0664	37.5419	2.1423	0.0571
112	2.6607	5.5203	2.0748	2.5786	5.2414	2.0327
113	0.5231	0.9435	1.8036	0.4439	0.6682	1.5053
114	1.6027	1.4358	0.8958	1.5061	1.0837	0.7195
115	1.2481	1.7291	1.3853	1.1690	1.4518	1.2419
116	2.3037	2.6942	1.1695	2.2196	2.4088	1.0852
117	33.6278	6.4128	0.1907	33.5476	6.0752	0.1811
118	1.2705	1.2341	0.9713	1.1323	0.9526	0.8414
119	1.1275	5.4831	4.8631	1.0480	5.2072	4.9687
120	40.9253	2.3014	0.0562	40.8391	2.0111	0.0492
121	1.1697	4.6816	4.0025	1.0884	4.4075	4.0496
122	0.4075	0.5070	1.2441	0.3237	0.2263	0.6991
123	0.1245	0.3481	2.7965	0.0475	0.0822	1.7298
124	0.3742	0.4807	1.2848	0.2933	0.2081	0.7094
125	0.1534	0.4895	3.1915	0.0690	0.1491	2.1598
126	3.0725	1.4358	0.4673	2.9823	1.1642	0.3904
127	0.2960	0.5131	1.7333	0.2114	0.2318	1.0963
128	1.2035	5.5408	4.6039	1.1211	5.2596	4.6913
129	40.4933	14.7396	0.3640	40.4113	14.4484	0.3575
130	1.1022	4.5821	4.1572	1.0141	4.3074	4.2473
131	2.0745	2.2174	1.0689	1.9917	1.9353	0.9717
132	0.1699	0.3993	2.3500	0.0950	0.1382	1.4549
133	0.3718	0.4727	1.2712	0.2937	0.2042	0.6951
134	0.2649	0.4321	1.6313	0.1823	0.1511	0.8285
135	4.0902	2.1995	0.5377	4.0125	1.9284	0.4806
136	0.4043	0.5072	1.2546	0.3225	0.2290	0.7101
137	7.7678	5.8394	0.7518	7.6881	5.5613	0.7234
138	0.4243	0.8403	1.9804	0.3449	0.5096	1.4776
139	3.8008	3.7529	0.9874	3.7066	3.4190	0.9224
140	0.9810	1.5709	1.6013	0.9021	1.3054	1.4471

Figure 19 (Part 2 of 3). CB2 Results for DB2 V1.2 vs. V2.1

Query	Total Time DB2 V1.2 Total Time	DB2 V2.1 Total Time	Ratio V2/V1	Execution Time DB2 V1.2 Exec Time	DB2 V2.1 Exec Time	Ratio V2/V1
141	1.3607	1.8634	1.3695	1.2856	1.6010	1.2454
142	2.0535	4.9131	2.3925	1.9748	4.6426	2.3509
143	0.5242	0.6415	1.2238	0.4409	0.3581	0.8123
144	1.0817	1.6198	1.4975	0.9881	1.3276	1.3435
145	2.4621	3.0381	1.2339	2.3786	2.7549	1.1582
146	35.2847	6.3971	0.1813	35.1764	6.0090	0.1708
147	0.4201	0.6463	1.5386	0.3386	0.3697	1.0918
148	7.6323	8.0664	1.0569	7.4686	7.0958	0.9501
149	1.0034	1.7157	1.7098	0.9249	1.4481	1.5658
150	0.2337	0.4598	1.9675	0.1396	0.1379	0.9873
151	0.3754	0.4960	1.3211	0.2956	0.2192	0.7417
152	1.1621	1.2316	1.0598	1.0784	0.9504	0.8813
153	0.9474	1.0563	1.1150	0.8666	0.7695	0.8880
154	2.1787	2.0459	0.9391	2.0931	1.7525	0.8373
155	33.4378	6.6186	0.1979	33.3434	6.3216	0.1896
156	0.3641	0.4789	1.3154	0.2844	0.2048	0.7202
157	21.2428	13.0506	0.6144	21.1455	12.7014	0.6007
158	26.2177	2.4812	0.0946	26.1377	2.2082	0.0845
159	0.1521	0.3561	2.3420	0.0687	0.0798	1.1615
160	1.7553	5.2316	2.9804	1.6745	4.9519	2.9573
161	1.6564	3.5770	2.1595	1.5732	3.2947	2.0943
162	0.9127	0.9815	1.0754	0.8318	0.6967	0.8376
163	2.5384	2.1802	0.8589	2.4406	1.8831	0.7716
164	6.5826	2.9081	0.4418	6.4971	2.6196	0.4032
165	0.3612	0.4745	1.3137	0.2817	0.2013	0.7146
166	5.6533	6.3294	1.1196	5.5089	5.7844	1.0500
167	28.0960	3.5403	0.1260	28.0130	3.2685	0.1167
168	0.3726	0.4794	1.2866	0.2896	0.1999	0.6902
169	9.0038	5.5485	0.6162	8.9228	5.2587	0.5894
170	14.8685	14.8551	0.9991	14.7863	14.5737	0.9856
171	0.9071	1.0171	1.1212	0.8262	0.7316	0.8855
172	2.5704	2.0951	0.8151	2.4681	1.7979	0.7285
173	3.2203	1.6473	0.5115	3.1426	1.3766	0.4381
174	0.6269	0.7334	1.1699	0.5009	0.4587	0.9157
175	1.0684	1.1771	1.1017	0.9728	0.8322	0.8554
176	1.1691	1.1464	0.9806	1.0878	0.8646	0.7948
177	0.1658	0.4133	2.4923	0.0841	0.0757	0.9005
178	3.9012	2.3976	0.6146	3.8210	2.1253	0.5562
179	0.5973	0.7442	1.2460	0.5113	0.4602	0.9000
180	1.0270	1.0299	1.0028	0.9471	0.7487	0.7905
181	48.2831	3.0120	0.0624	48.2012	2.7151	0.0563
182	0.3522	0.4698	1.3341	0.2722	0.1953	0.7175
183	0.2956	0.4280	1.4481	0.2168	0.1489	0.6867
184	0.8130	1.2602	1.5501	0.7162	0.9036	1.2617
185	2.1608	1.8785	0.8694	2.0789	1.6004	0.7698
186	0.2608	0.4364	1.6735	0.1793	0.1601	0.8926
187	2.4324	5.4182	2.2275	2.3529	5.1509	2.1891
188	0.5656	0.7613	1.3461	0.4817	0.4803	0.9971
189	0.9851	1.0546	1.0706	0.9040	0.7680	0.8496
190	0.4185	0.5117	1.2228	0.3203	0.2216	0.6920
191	3.0924	1.5397	0.4979	3.0130	1.2667	0.4204
192	1.8833	2.3419	1.2435	1.7943	2.0586	1.1473
193	17.2015	8.8368	0.5137	17.1023	8.4847	0.4961
194	0.6074	0.6488	1.0682	0.5259	0.3638	0.6918
195	0.3882	0.5014	1.2915	0.3067	0.2228	0.7266
196	8.3346	6.0807	0.7296	8.2494	5.8033	0.7035
197	8.7750	16.2958	1.8571	8.6721	15.9914	1.8440
198	1.7656	1.4454	0.8187	1.6851	1.1502	0.6826
199	2.6373	2.1282	0.8070	2.5417	1.8361	0.7224
200	4.5863	5.9880	1.3056	4.5078	5.7183	1.2685
201	0.4755	0.6796	1.4291	0.3694	0.3940	1.0667
202	4.2133	4.9710	1.1798	4.1272	4.6613	1.1294
203	0.3887	0.4852	1.2482	0.3063	0.2106	0.6875

1044.58	585.22	<<<--- Total Elapsed Time
5.15	2.88	<<<--- V2.1 Arithmetic Mean 44% lower
1.57	1.64	<<<--- V2.1 Geometric Mean 4% higher

Total Execution Time --->>>	1026.83	524.47
V2.1 Arithmetic Mean 49% lower --->>>	5.06	2.58
V2.1 Geometric Mean 17% lower --->>>	1.37	1.13

Figure 19 (Part 3 of 3). CB2 Results for DB2 V1.2 vs. V2.1

3.0 Performance of Queries

3.1 Table Scans

3.1.1 General Overview

For some database queries, the database manager may choose to access each row in a table sequentially. This can be a very time consuming process, primarily from the amount of I/O involved in reading every row from the database.

3.1.2 Improvements

One of the key enhancements in this version which will make a significant improvement in table scan time is multiple tablespaces and multiple containers. Since the entire table needs to be scanned, there will be many reads from the database, so cheaper disk I/O will have a major impact. The more time the CPU can spend processing instead of waiting, the better. The introduction of I/O prefetching also plays a major role. Since the table access is sequential for a table scan, prefetchers can go ahead and retrieve several data pages in advance and put them in the buffer pool before the agent requests them. This enhancement will again cut down on I/O wait during the reads.

3.1.3 Considerations

There are several factors which affect the total time for a table scan:

- Whether the database resides on a DMS or SMS tablespace
- How many disks the database spans
- Whether raw devices are used
- Tablespace prefetch quantity
- Tablespace extent size

The different combinations of these key factors will affect the amount of time the database agent waits for disk I/O while scanning the database.

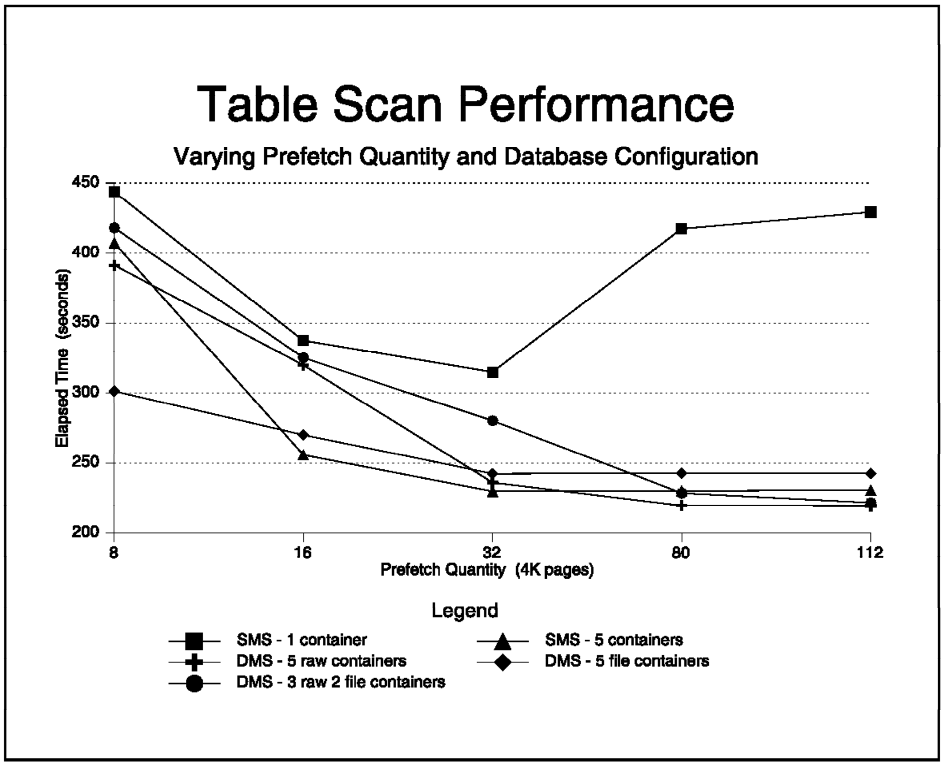


Figure 20. Table Scan Measurements

Figure 21. Table Scan Measurements for Varying Database Configurations and Prefetch Quantities. Elapsed times in seconds; Prefetch quantity in 4K pages; Extent sizes 16 pages.						
Prefetch Quantity	**SMS - 1 container**	**SMS - 5 containers**	**DMS - 5 raw cont.**	**DMS - 5 file cont.**	**DMS - 3 raw 2 file**	**V1.2**
8	443.64	412.33	298.50	301.38	418.09	N/A
16	337.36	255.71	319.77	270.02	325.11	N/A
32	314.88	258.81	265.04	242.31	280.14	N/A
80	417.42	231.17	219.70	242.70	228.58	N/A
112	429.20	240.72	224.50	242.60	221.76	N/A
N/A	N/A	N/A	N/A	N/A	N/A	370.00

The different database configurations can be described as follows:

- SMS - 1 container

  The database is created using one SMS container which resides on its own filesystem and spans four independent disks.

- SMS - 5 containers

  The USERSPACE1 of the database resides on 5 SMS containers with each container on its own filesystem. Each filesystem is on its own independent disk.

- DMS - 5 raw cont.

  The USERSPACE1 tablespace is created to span 5 DMS raw disk device containers. Each container is on its own independent disk.

- DMS - 5 file cont.

  The USERSPACE1 tablespace is created on 5 DMS file containers, with each container on its own file system which is on its own independent disk.

- DMS - 3 raw 2 file

  The USERSPACE1 tablespace is created on 3 DMS raw device containers. A separate tablespace is then created on 2 DMS file containers. The indexes for the table are placed in this tablespace with the actual data residing on the 3 raw devices. Each container is on its own independent disk.

- V1.2

  The database is created using one filesystem spanning four independent disks.

The database used for this experiment was the ID2 database (9 million rows and 1.2GB of raw data; see A.4, “Internal Database 2 (ID2)” on page 102 for details). The query consisted of a ″SELECT COUNT(*)″ which would scan the entire table but return no rows. The default database and database manager configurations were used except for <BUFFPAGE> which was set to 4096 4K pages. The machine used was SERVER 1 (refer to B.1, “Configuration for machine SERVER 1 (AIX)” on page 104).

The best case for V2.1 (DMS - 5 raw containers and prefetch quantity of 80, which equals the number of containers times the extent size) represented 19.7 GB/hr. and a 41% reduction in elapsed time vs. the V1.2 case. In general, I/O wait percentages for these measurements were 0 when more than one disk was being read from concurrently. In V1.2, I/O wait was 3%. So the elapsed time reduction in V2.1 was not merely from reducing I/O wait time, but from large reductions in CPU consumption through the efficiencies of large-scale blocking.

3.1.4 Performance Tips

Choosing the layout of the database and the method of data storage can make a significant difference in table-scan performance. Due to the large amount of I/O involved, it is important to design the database in such a way to minimize I/O wait. Regardless of whether system managed storage (SMS) or database managed storage (DMS) is used, it is important to spread the database across multiple disks. Putting a 2 GB database on 4 disks of 500 MB each will be much more efficient than one 2 GB volume on a single disk. This way 4 disk arms and 4 prefetchers will be able to access data in parallel, hence reducing I/O wait.

From the above graph, DMS has the edge over SMS for overall performance. DMS with raw devices or files generally comes out ahead, particularly with the raw devices. SMS on multiple independent containers performs particularly well, but DMS raw devices pulls out ahead at higher prefetch quantities. Whether to use DMS raw devices or files depends on the limits of system resources. Raw devices provide better performance if properly tuned. If resource constraints require you to use a smaller buffer pool (i.e. less than 2000 pages), you may not want it to be filled with prefetched data pages at the expense of other potential uses. In that case letting the AIX file system handle some basic prefetching, as is done with DMS files, could be sufficient.

The graph also indicates the impact of prefetching data pages. One key point that must be stressed is that for buffer pools smaller than 1100 pages, the prefetch quantity will default to 16 pages regardless of any other values specified through the CREATE DATABASE or ALTER TABLESPACE commands. This is for the reason described above, regarding using up buffer pool space.

Other tips:

- Spread the database across at least two independent containers - each on its own disk. This is especially true in the case of SMS. The graph shows that there is a substantial difference in performance between using one SMS container on 4 disks, or individual containers on separate disks.
- Exploit parallelism by using a prefetch quantity that is a multiple of the extent size. For example, if your database is spread across 4 containers with an extent size of 16 pages, use a prefetch quantity of 32 pages to read from two containers asynchronously or even 64 pages to read from all containers in parallel. This dramatically reduced I/O wait.
- If only a small buffer pool is available, it's not necessary to prefetch huge quantities to greatly reduce I/O wait. Two extents worth of prefetched pages makes a large difference.
- Try using a smaller extent size to allow for more parallelism. With a database spread on 4 disks and extent size of 32 pages, you would need a prefetch quantity of 64 pages for a parallelism degree of 2. With an extent size of 16, those same 64 prefetched pages would achieve 4th-degree parallelism and further reduce I/O times.
- Use multiples of extent sizes rather than factors for prefetching. The graph illustrates the higher elapsed times for prefetching 8 pages as opposed to 16 pages.
- There is little benefit in specifying a huge buffer pool for table scans. As long as <BUFFPAGE> is set above 1100, prefetch quantities can be varied above 16 pages. In all cases (SMS, DMS files, DMS raw) an excessive buffer pool actually hinders performance. If the buffer pool is too big, the memory occupied by it could be more profitably used for such things as a larger sort heap, for example.

3.2 Index Scans

3.2.1 General Overview

The use of indexes can greatly improve query performance and can save having to scan an entire table to find certain data. An index scan will help to provide a range of qualifying rows to search through instead of scanning the whole table, as well as produce ordered output.

3.2.2 Improvements

The index manager has been greatly improved in this version and allows for more efficient access of keys.

As in the case with table scans, the ability to have databases span multiple containers and either be managed by the operating system or the database manager has made a significant impact on index scans. The resulting reduction in I/O wait time is easily apparent.

3.2.3 Considerations

There are several factors which affect the total time for a index scan:

- Database configuration
  - Whether the database resides on a DMS or SMS tablespace
  - How many disks the database spans.
  - Whether raw devices are used.

  The different combinations of these key factors will affect the amount of time you are left waiting for disk I/O while the scanning the index and the database.
- Nature of query

  The way the index is used for an index scan depends greatly on the query, and this in turn influences the overall performance. For example, a query requiring index-only access (e.g., a SELECT COUNT(*)) is going to perform differently than a query requiring access to data pages of the actual table after scanning the index.
- Cluster factor of indexes

  The cluster factor of the index which is scanned is a huge factor in the time to access data pages, because sequential prefetch can be used if clustering is high.

Using the ID2 database on SERVER 1 (refer to B.1, "Configuration for machine SERVER 1 (AIX)" on page 104), two types of index scans were studied. The first was a selective scan, via SELECT COUNT(*) with one BETWEEN predicate that limited the scan to 1.25M of 9M entries of a perfectly clustered index. The second was a non-selective scan, via the same SELECT statement but with the clustered index replaced by an unclustered one. All 9M index entries were examined.

Each of these index scans was executed in two variations: with data page access (having a predicate referencing a column not in the index, but with the predicate not disqualifying any rows), and index-only access. The queries only returned a count of the qualifying rows.

Below are the experimental results. Note that the scenario with an SMS database with a single container on 4 disks was not studied, since it appears to be the worst performer of the different database configurations from the table scan measurements (per 3.1, "Table Scans" on page 37).

Figure 22. Index Scan Measurements for Varying Database Configurations and Prefetch Quantities. Elapsed times in seconds; Prefetch quantity in 4K pages; Extent sizes 16 pages.

Type of Index Scan	Prefetch Quantity	SMS - 5 containers	DMS - 5 raw cont.	DMS - 5 file cont.	DMS - 3 raw 2 file
Selective Index-only	16	42.38	69.20	42.47	43.07
Selective Index-only	80	42.31	69.18	42.74	44.85
Selective w/Data access	16	135.59	171.60	148.51	136.33
Selective w/Data access	80	134.87	184.87	142.82	134.09
Non-sel. Index-only	16	248.98	349.02	250.62	252.15
Non-sel. w/Data access	16	1717.37	2083.45	2046.00	2158.00
Non-sel. w/Data access	80	1108.28	1573.23	1212.93	1750.27

The same database configurations as those described in the table scan section (see 3.1, “Table Scans” on page 37) were used in this experiment. The database configuration parameters used were the defaults except for <BUFFPAGE> which was set at 4096 pages.

Note

Version 1 results were not obtained for AIX. For a V1-V2 comparison on OS/2, see 3.3.2, “DB2/2 Index Scans” on page 45.

3.2.4 Performance Tips

With the absence of index-page prefetching in this release, DMS raw does not perform the best for index-only scans. For such cases, DMS files or SMS containers provide the best performance due to prefetching done by AIX. For index scans accessing data pages, prefetching can make a major difference in reducing I/O wait. Index prefetching will be available in the next release of DB2.

As in the case of table scans (per 3.1, “Table Scans” on page 37), the database layout and method of data storage impact performance significantly. Spreading the data out on multiple disks and specifying multiple containers on independent disks is extremely beneficial, even in the SMS case. The SMS - 5 scenario performs very well. In the case of DMS, if index-scans are the main tuning criteria, DMS files performs better than raw. The importance of prefetching comes out clearly in the case of index scans which require access to data pages. If the index is clustered, sequential detection will kick in and a higher prefetch quantity will make a significant difference.

For the case of a workload involving both index scans and table scans, one approach for V2.1.0 is to use the mixed DMS scenario (e.g. 3 raw containers for the data and 2 file containers for the indexes). This will provide the best performance for both table scans, which benefit from raw devices, and index scans, which are helped by file containers. The difference in performance between this mixed configuration and the optimal numbers for each respective type of scanning (i.e. raw for table scans and files for index scans) is relatively small. Note, however, that V2.1.1 index scans on raw devices will be much faster than in V2.1.0, so the benefit of a mixed DMS configuration will be much reduced, if not eliminated.

Other tips:

- Spread the database across at least two independent containers - each on its own disk.
- Exploit parallelism by using a prefetch quantity that is a multiple of the extent size. For example, if your database is spread across 4 containers with an extent size of 16 pages, use a prefetch quantity of 32 pages to read from two containers asynchronously or even 64 pages to read from all containers in parallel. This dramatically reduced I/O wait.
- The cluster factor of the index makes a huge difference. For maximimum index scan performance, it may be worth it to do a Reorg on the specific index in question, which would then result in the data being sequential. With sequential detection enabled, prefetching will be able to reduce I/O waits dramatically, particularly for non-selective index scans.
- The effects of a large buffer pool are minimal, as long as a minimum of 1100 pages is specified. Below this threshold DB2 will restrict the prefetch quantity to 16 4K pages.

3.3 Performance of Queries on OS/2

3.3.1 DB2/2 Table Scans

The database used for this experiment was the ID2 database (6 million rows, 0.8GB; see A.4, "Internal Database 2 (ID2)" on page 102) and the query consisted of a ″SELECT COUNT(*)″ which would scan the entire table but return no rows. The default database and database manager configurations were used except for <BUFFPAGE> which was set to 2000 4K pages. The machine used was SERVER 5, described in B.5, "Configuration for machine SERVER 5 (OS/2)" on page 104.

Figure 23. OS/2 Table Scan Measurements for Varying Database Configurations and Prefetch Quantities. Elapsed times in seconds; Prefetch quantity in 4K pages; Extent sizes 32 pages.

Prefetch Quantity	SMS - 1 container	SMS - 3 containers	DMS - 3 containers	V1.2
16	N/A	604.83	599.20	N/A
48	N/A	544.20	550.40	N/A
64	669.68	597.01	601.63	N/A
80	N/A	610.80	615.00	N/A
96	642.92	593.80	610.74	N/A
N/A	N/A	N/A	N/A	891.00

The different database configurations can be described as follows:

- SMS - 1 Container

  The database was created using an SMS tablespace with 1 container.

- SMS - 3 Containers

  The database was created using an SMS tablespace with 3 containers, each on its own disk.

- DMS - 3 Containers

  The database was created using a DMS tablespace with 3 containers, each on its own disk.

- V1.2

  As prefetching and media spanning are not available in V1.2 the database was created on a single disk drive.

In the best case observed (SMS with 3 containers, prefetch=48), V2.1 was 39% faster than the V1.2 result. Even in its worst case (SMS with 1 container, prefetch=64), V2.1 was 25% faster than V1.2.

Having multiple containers for SMS reduced elapsed times by about 10%. The difference between SMS and DMS with 3 containers was not significant.

Increasing prefetch quantity provided much less benefit on OS/2 than on AIX (see 3.1, "Table Scans" on page 37) and even prefetching two full extents of 32 pages started to degrade performance. Separate experiments not using DB2 indicate that this is a function of the hardware and operating systems in use and is not reflective of DB2 limitations.

3.3.2 DB2/2 Index Scans

The database used for this experiment was the ID2 database (6 million rows, 0.8GB; see A.4, "Internal Database 2 (ID2)" on page 102) and the same queries were run as for AIX (see 3.2, "Index Scans" on page 41). The default database and database manager configurations were used except for <BUFFPAGE>, which was set to 2000 4K pages. The machine used was SERVER 5, described in B.5, "Configuration for machine SERVER 5 (OS/2)" on page 104.

Figure 24. OS/2 Index Scan Measurements for Varying Database Configurations and Prefetch Quantities. Elapsed times in seconds; Prefetch quantity in 4K pages; Extent sizes 32 pages.

Type of Index Scan	**Prefetch Quantity**	**SMS - 1 container**	**SMS - 3 containers**	**DMS - 3 containers**	**V1.2**
Selective Index-only	16	N/A	59.40	58.60	N/A
Selective Index-only	48	N/A	58.60	59.00	N/A
Selective Index-only	64	62.25	69.82	64.48	N/A
Selective Index-only	96	63.75	62.00	61.40	N/A
Selective Index-only	N/A	N/A	N/A	N/A	181.17
Selective w/Data access	16	N/A	267.30	295.72	N/A
Selective w/Data access	48	N/A	278.46	267.80	N/A
Selective w/Data access	64	270.70	270.78	265.71	N/A
Selective w/Data access	96	270.18	267.73	270.78	N/A
Selective w/Data access	N/A	N/A	N/A	N/A	315.75

The different database configurations can be described as follows:

- SMS - 1 Container

  The database is created using an SMS tablespace with 1 container.

- SMS - 3 Containers

  The database is created using an SMS tablespace with 3 containers, each on its own disk.

- DMS - 3 Containers

  The database is created using a DMS tablespace with 3 containers, each on its own disk.

- V1.2

  As prefetching and media spanning are not available in V1.2 the database was created on a single disk drive.

For the selective index-only tests, V2.1 elapsed time was up to 68% faster than the V1.2 result. With data access, the V2.1 advantage was up to 16%.

There was no significant difference between SMS and DMS in these measurements.

Here are some utilization figures for the above measurements as supplied by SPM/2. The disk utilization is average % time busy for the database disks.

```
                                          DMS - 3 Containers
                                          ------------------
Selective Index-only, Prefetch = 64       98% CPU,  8% disk
Selective w/ Data access, Prefetch 64     90% CPU, 18% disk
```

3.4 SQL Methods and Implementations

3.4.1 Dynamic SQL Caching

This is a new feature in V2.1. What ***dynamic SQL caching*** does is quite simple: When DB2′s SQL compiler is passed a dynamic DML statement, it will produce a plan for executing that statement. The section for that statement is stored in the cache for the agent. If that agent is later called upon to execute the identical statement, and the section is still in the cache, the statement will be executed from the cache without re-compilation.

In a transaction or query environment where prepare time is a performance issue, this technique can have a major effect on performance: as the following measurements show, performance can be improved up to 91% !

Be aware that there are limitations on the life of a cached plan. For example, DDL executed by any user could render cached plans invalid, so the agent caches are flushed to be sure no problems arise.

The following measurements were run on the IB1 database (see A.3, "Internal Benchmark/Database 1 (IB1)" on page 102) on SERVER 2 (see B.2, "Configuration for machine SERVER 2 (AIX)" on page 104) with query#1. The whole test consists of 60 different versions of query#1. The sections are described below.

Statement #	Elapsed Time (s)	Comment
	section 1	
1	0.301	query 1
2	0.012	query 1
3	0.011	query 1
4	0.011	query 1
5	0.011	query 1
6	0.011	query 1
7	0.012	query 1
8	0.011	query 1
	section 2	
9	0.038	update record
10	0.011	query 1
11	0.011	query 1
12	0.011	query 1
13	0.011	query 1
14	0.012	query 1
15	0.011	query 1
16	0.011	query 1
17	0.012	query 1
	section 3	
18	0.114	query 1.2
19	0.011	query 1.2
20	0.011	query 1.2
21	0.011	query 1.2
22	0.012	query 1.2
23	0.011	query 1.2
24	0.011	query 1.2
25	0.011	query 1.2
	section 4	
26	0.121	query 1.3
27	0.040	query 1.3
28	0.039	query 1
29	0.038	query 1.2
30	0.039	query 1.3

Figure 25 (Part 1 of 2). Caching measurements.

section 5		
31	0.115	query 1.4
32	0.102	query 1.5
33	0.103	query 1.6
34	0.108	query 1.7
35	0.114	query 1.8
36	0.107	query 1.9
37	0.107	query 1.10
38	0.107	query 1.11
39	0.094	query 1.12
40	0.090	query 1.13
41	0.084	query 1.14
42	0.123	query 1.15
43	0.084	query 1.16
44	0.090	query 1.17
45	0.087	query 1.18
section 6		
46	0.038	query 1.4
47	0.038	query 1.5
48	0.038	query 1.6
49	0.038	query 1.7
50	0.039	query 1.8
51	0.038	query 1.9
52	0.039	query 1.10
53	0.041	query 1.11
54	0.039	query 1.12
55	0.038	query 1.13
56	0.037	query 1.14
57	0.038	query 1.15
58	0.038	query 1.16
59	0.038	query 1.17
60	0.037	query 1.18

Figure 25 (Part 2 of 2). Caching measurements.

These measurements contain a treasure of information about transaction processing. The numbers that you see prove several concepts and performance issues. We will go through these numbers section by section. The following performance issues will be discussed and demonstrated.

- Dynamic SQL caching.
- First-time table access overhead.
- I/O wait.
- Ramp-up.

We will demonstrate these concepts as we go through the measurements, one section at a time.

- Section 1. Please take a look at statement #1. Notice that it has the longest execution time of all. This statement executed in 300 milliseconds, whereas all the other statements executed in 10 to 120 milliseconds. This is what we call ***First-time table access overhead.*** As we look further down section 1, the execution time becomes a constant (11ms). The subsequent executions are faster because there is no I/O wait, no prepare time (thanks to SQL caching), and no first-time table access overhead.
- Section 2. This section runs query#1 again 8 times. But first, it updates that row which query#1 returns. Apparently, this did not affect performance. The old plans were still good.

- Section 3. Here we run query 1.2. This query is exactly like query 1 but it has different constants. The section starts with statement #18, which runs in 114ms. This execution time is affected by I/O wait and by the prepare time. However it is not affected by first-time table access which we noticed in statement 1. Subsequent executions of this query show the same peak performance (11ms) we noticed above.
- Section 4. A new query is introduced here (query#1.3) with 121ms execution time. Then we run queries 1, 1.2 and 1.3 again. Notice that the execution time is much slower now because these queries were run interactively through db2batch, which doesn't always benefit from SQL Caching. In db2batch, you can take advantage of SQL caching by running the same statement consecutively.

  Another way to benefit of SQL caching is to write your own embedded dynamic application. In V2.1.1, db2batch will have its own statement memory and will get an even bigger performance boost from SQL caching.
- Section 5. In this section we ran 15 similar queries, with randomly selected rows accessed. Please look at the execution times from query 1.4 to query 1.18. You will notice a gradual improvement. This is what we call the ***ramp-up***. As the database runs more of these queries, portions of the tables and indexes are loaded into the buffer pool. As the buffer pool fills up, more queries can access their data directly from memory. I/O wait is gradually reduced. Eventually, the execution time reaches the peak performance threshold. This is discussed in the next section.
- Section 6. Here we run the same 15 queries from section 5 again. Exactly the same queries. As you can see, the execution time is at its peak performance threshold of 38ms. All the data is in the buffer pool, and I/O time is eliminated completely. This is the ideal scenario, where all the tables and all the indexes fit into the buffer pool. So we recommend that you use a large buffer pool, because when you increase the buffer pool size, you approach the peak performance scenario.

3.4.2 Compound SQL

Compound SQL is probably one of the most under-used DB2 performance techniques. Its use shows maximum benefit over client/server network configurations, including DDCS environments. In this type of setup, the commmunication time is greatly reduced. However, Compound SQL is also advantageous on a local configuration; for example, refer to Figure 36 on page 59.

To measure the performance of Compound SQL vs. regular Static SQL, we ran 80 update statements against the the SPEC database, on SERVER 3 (see B.3, "Configuration for machine SERVER 3 (AIX)" on page 104), either individually or within one Compound SQL statement. In the Client/Server measurements, we used SERVER 2 (see B.2, "Configuration for machine SERVER 2 (AIX)" on page 104) as the client.

In the client/server environment, Atomic Compound SQL saved 64% of the execution time vs. individual SQL statements, by avoiding 80 network crossings. Even in the local configuration, Atomic Compound saved 37%.

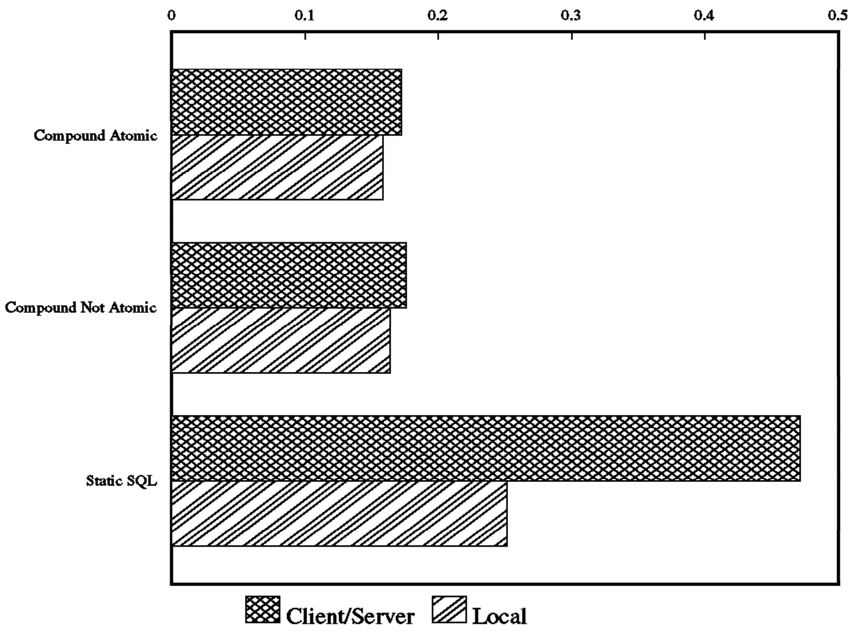


Figure 26. Compound SQL Performance

Figure 27. Compound SQL Performance		
SQL method	**Client/Server**	**Local**
Atomic Compound SQL	0.1726	0.1584
Not Atomic Compound SQL	0.1758	0.1638
Non-Compound SQL	0.4715	0.2518

3.4.3 Static vs. Dynamic vs. CLI

This portion compares the performance of embedded dynamic SQL, embedded static SQL, and CLI. The IB1 workload was tested in each case. For more information on the IB1 workload please refer to A.3, "Internal Benchmark/Database 1 (IB1)" on page 102

Figure 28. Elapsed times for different Application Coding methods

Query #	Embedded Static	Embedded Dynamic	CLI	Dynamic with (?)	CLI with (?)
1	0.07	0.10	0.16	0.16	0.19
2	4.88	4.89	4.89	4.92	5.29
3	75.44	75.48	76.06	75.90	76.17
4	77.84	78.67	77.96	78.26	77.19
5	16.69	18.18	18.19	18.15	20.77
6	10.87	11.05	11.08	10.93	11.11
7	518.30	314.51	311.30	566.12	549.12
8	4.73	6.54	6.54	9.33	9.19
9	134.31	135.77	137.19	136.14	136.57
10	99.47	99.82	100.18	98.77	99.09
11	541.84	551.61	551.61	549.10	542.55
12	162.20	164.82	165.84	165.47	164.03
13	999.28	1004.50	1004.59	1005.53	1007.55
14	197.38	203.71	203.09	202.85	200.43
15	50.19	50.67	50.98	50.81	50.33
16	0.01	0.03	0.08	0.04	0.05
17	0.07	0.09	0.14	0.10	0.09
18	0.05	0.08	0.10	0.08	0.10
19	0.23	0.26	0.30	0.25	0.21

- All number are in seconds.
- (?) = parameter markers.

These results prove 4 major points:

a) Generally, Embedded Dynamic and CLI have the same performance. The numbers are almost identical, except for the transaction queries, where dynamic has a clear advantage. This is caused by CLI overhead.

b) _For complex queries_ (2 to 15), Static SQL performs worse than Dynamic SQL, because it is hindered by host variables. When the optimizer is choosing a plan for a static application, some predicates will be defined with host variables instead of constants, so it won't have all the information it needs to find the best plan. Though in most cases, the static optimizer did find the best plans, in which cases it performed slightly faster because it doesn't have to prepare the statements.

   For example, in query #7, the difference in performance between dynamic and static can be explained by host variables. The static explain output shows an index scan where the dynamic explain output has a table scan. The optimizer working in dynamic could find the better plan because it was working with constants. Even though in the dynamic case we are also counting the prepare time, this becomes a negligible factor (1-2 seconds).

For simple OLTP queries (1 and 15 to 19), static is the better choice because the dynamic prepare time is no longer negligible.

c) Embedded Dynamic SQL, when loaded with parameter markers (where the Static had host variables) will have the same plans as Embedded Static SQL. The parameter markers hurt the performance in both ways: In complex queries, it doesn't have enough information to find a good plan, and in simple queries, it is hindered by the prepare time.

   Therefore we do not recommend the use of parameter markers - unless you have proof that the static plan is better than the dynamic plan. There are two ways to avoid parameter markers: we suggest you prepare your statements dynamically, or else you can use host variables. If you use host variables then the optimizer will produce the same plan as it would with parameter markers, but at least the performance won't be hampered by the prepare time.

d) CLI with parameter markers: As expected, these numbers are extremely close to Dynamic SQL with parameter markers.

3.4.4 db2batch

The db2batch program (found in the “misc” subdirectory of sqllib) is very useful for performance evaluations. This program is used for measuring benchmarks and timing queries. It is an excellent alternative to CLP, from a performance point of view, when evaluating a stream of queries, each of which is executed once.

CLP′s design objective was rich functionality and access to all APIs. Db2batch will execute your queries in an embedded dynamic environment, thus bypassing a lot of overhead associated with CLP. Therefore, db2batch will run any query faster than CLP, and the advantage is most obvious when you have a large answer set.

The first query reported in Figure 29 returns a single row using an index. The second query returns 1200 rows using a table scan. The third query is just ″select * from table″. The database used for theese queries is the ID2 database (for a description of the database, see A.4, “Internal Database 2 (ID2)” on page 102). The two variations of db2batch tests show the cost of fetching no rows vs. all rows in the answer set (this is a db2batch option).

Figure 29. db2batch vs. CLP

Query #	rows fetched	db2batch (no-rows)	db2batch (all-rows)	CLP (all-rows)
1	1	0.1	0.1	2.0
2	1200	255.6	256.5	264.7
3	600000	567.1	965.6	2349.1

Db2batch is also an excellent tool for tuning your queries, and it gives you an idea of what performance you can expect from embedded dynamic SQL. If you try running your workload with db2batch, use PERF_DETAIL=4, and you will obtain information which will help you tune your database and your workload.

More information on db2batch is provided in its help text (displayed via “db2batch -h”) and in *Administration Guide for DB2 Common Servers*.

3.5 Optimization Classes

The default optimization class is 5. It typically gives the best performance on a broad range of queries. Choosing a lower optimization class is appropriate when the prepare time is hurting your performance and the plans are all pretty simple.

Choosing a high optimization class is a good idea when prepare time is a small fraction of the total execution time. We recommend that you use the default optimization class for all your work, or else you should be prepared to do some experimenting.

Here is a 3D graphic which shows how prepare time decreases when you use a lower optimization class. These are the prepare times for the IB1 set of queries. See A.3, "Internal Benchmark/Database 1 (IB1)" on page 102 for a description of these queries. The highest optimization class (5) is at the background, while the lowest optimization class (0) is at the foreground.

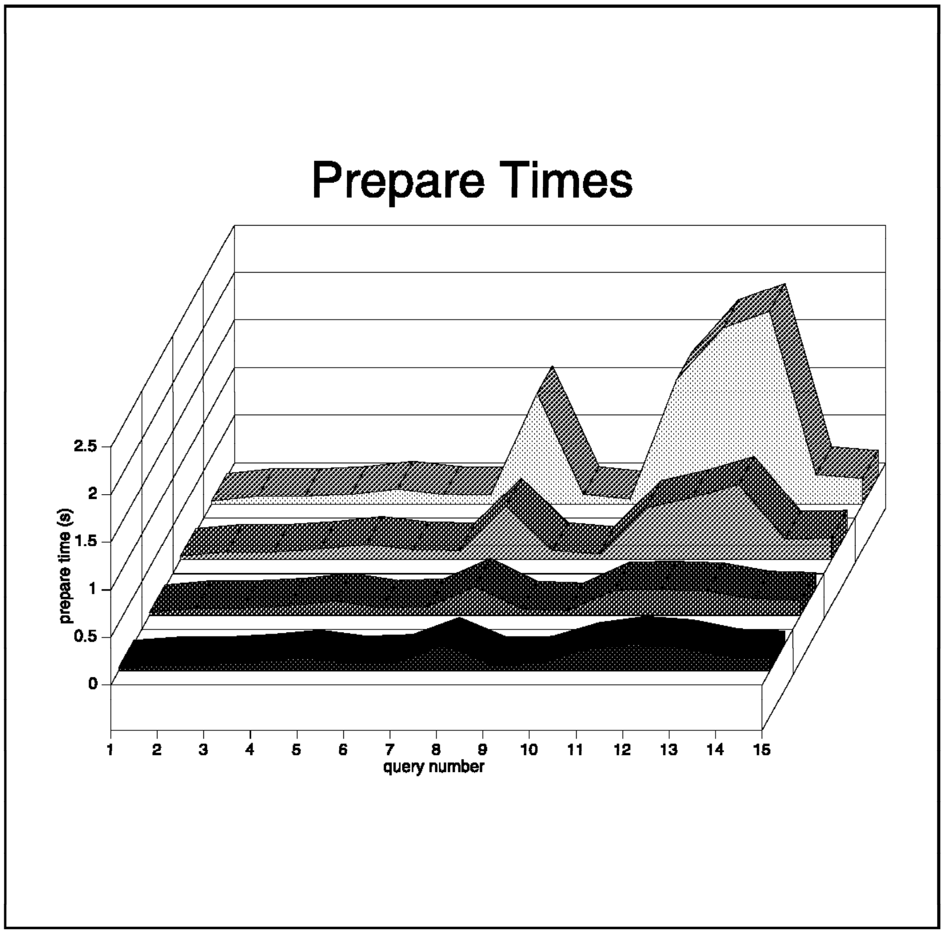


Figure 30. Prepare Time for Optimization classes 5,3,1,0

In the following graph, the same queries are measured for their run time. The run time of the query is reduced when you use a higher optimization class. This graph is different in from the previous graph because now the highest optimization class (5) is at the *foreground* and the lowest optimization class (0) is at the *background*.

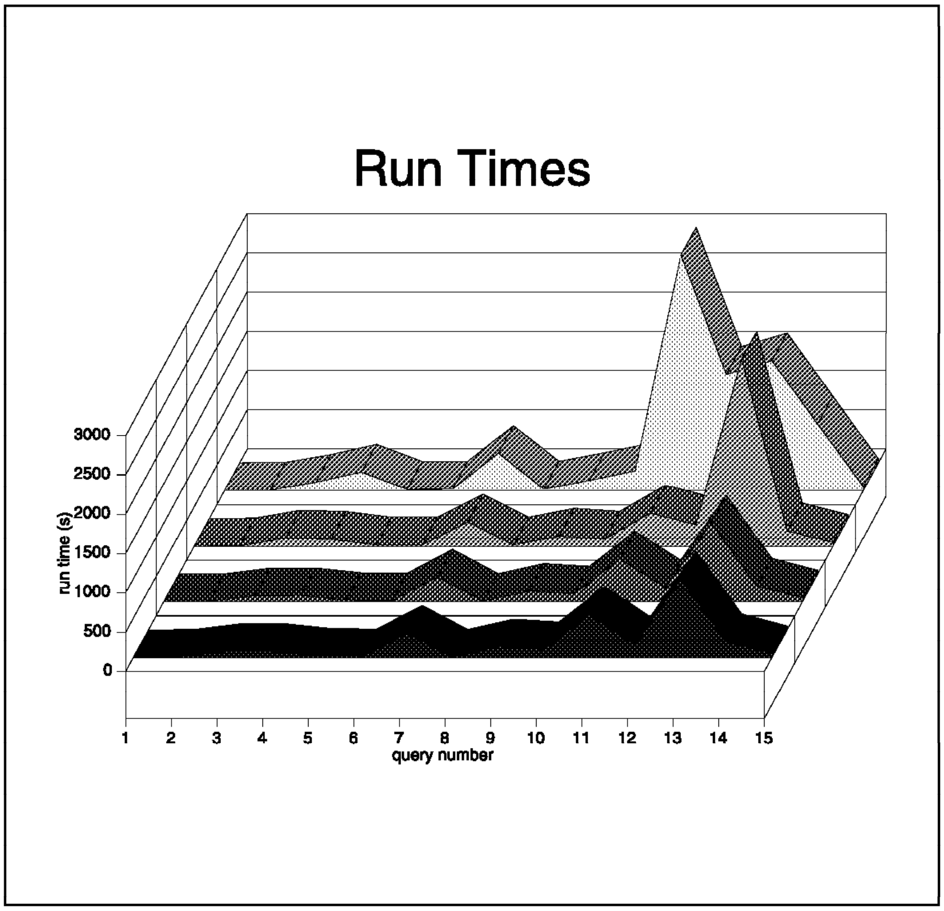


Figure 31. Run Time for Optimization classes 0,1,3,5

And now for the raw data:

Figure 32. Prepare Times				
Query Number	**Optimization Class 5**	**Optimization Class 3**	**Optimization Class 1**	**Optimization Class 0**
1	0.0318	0.0371	0.0302	0.0320
2	0.0843	0.0801	0.0747	0.0671
3	0.0821	0.0796	0.0749	0.0670
4	0.1061	0.1097	0.0990	0.0938
5	0.1611	0.1623	0.1513	0.1384
6	0.1047	0.1049	0.0771	0.0757
7	0.0992	0.0953	0.0929	0.0927
8	1.1657	0.5671	0.3055	0.2708
9	0.1033	0.0980	0.0684	0.0667
10	0.0527	0.0563	0.0496	0.0733
11	1.3100	0.5423	0.2683	0.2161
12	1.8556	0.6536	0.2746	0.2815
13	2.0292	0.7940	0.2580	0.2472
14	0.3134	0.2213	0.1724	0.1501
15	0.2730	0.2267	0.1581	0.1292

Figure 33. Run Times				
query number	**Optimization Class 5**	**Optimization Class 3**	**Optimization Class 1**	**Optimization Class 0**
1	0.0814	0.0776	0.1319	0.1004
2	4.8261	4.8733	5.9873	6.1096
3	75.7283	76.4583	105.4755	105.5529
4	77.7549	78.1163	89.8271	235.4317
5	17.8335	17.7952	17.5354	17.8000
6	10.7487	10.9004	19.2371	19.4316
7	307.1994	315.4243	310.0255	476.3797
8	5.2844	6.1671	17.7569	18.8902
9	135.3387	135.1032	129.8200	129.4204
10	99.9273	99.6539	92.1233	250.6045
11	548.9655	548.0452	422.6164	22771.6640
12	163.5137	179.2566	273.8195	1479.7744
13	1000.1810	1001.9269	2381.6468	1652.2660
14	201.9086	201.2844	196.1280	850.2619
15	50.1490	50.4793	50.7096	50.7398

The data shows that for all the queries with a run time of 5 seconds or more, the prepare time is not a factor. So in these cases the default optimization class is perfect. For the queries with a sub-second run time, the prepare time is no longer negligible, and you may want to reduce the optimization class. However, - as the data shows on query #1 - the prepare time might not drop all that much for very simple queries.

If the prepare time is still too costly, you may want to switch to static. For a discussion of Static vs. Dynamic, please refer to 3.4.3, “Static vs. Dynamic vs. CLI” on page 51. The topic of optimization class is well covered in the Administration Guide.

3.6 Sorting

Sorting has improved by 79% since V1, according to one simple test:

Query 1: SELECT FIELD0 FROM T1
Query 2: SELECT FIELD0 FROM T1 ORDER BY FIELD0

The sorting time is approximately equal to the run time of query 2 minus the run time of query 1. With V1.2, the difference in run time was 309 seconds. With V2.1 the difference in run time was 64 seconds. This shows an improvement of 79%, and a V2.1 sort rate of 15,600 rows per second (with sorting on one INTEGER column).

Figure 34. Sorting costs

Version	Query 2	Query 1	Sorting cost
V1.2	4020	3711	309
V2.1	3255	3191	64

The queries were applied to the ID3 database (see A.5, “Internal Database 3 (ID3)” on page 102 for details) with 1,000,000 rows on SERVER 2, and index ″I1″ was dropped to make sure that the data is sorted, and not retrieved through an index scan. The queries were executed through the DB2 command line processor (CLP).

The amount of improvement in V2.1. will depend on many factors, such as datatypes, sort column width, SORTHEAP and SHEAPTHRES values, amount of data that overflows the sort heap to temporary tables, and temporary tablespace configuration.

4.0 Performance of Utilities

4.1 Import

4.1.1 General Overview

While the recommended method of inserting data in version 2.1 is the LOAD utility (see 4.2, “Load” on page 61), the IMPORT utility remains available, with some modifications from Version 1. There are cases when it is better to use IMPORT as opposed to LOAD (see 4.4, “Load vs. Import” on page 67).

Up to a 77% reduction in Import time (4 times speedup) was observed in V2 (see scenario 1 vs. 4 in Figure 36 on page 59).

4.1.2 Improvements

The principal improvement in IMPORT for version 2.1 is support for Non-Atomic Compound SQL. IMPORT actually executes SQL INSERTs, and this can be quite costly, especially when inserting to host databases connected by a DDCS gateway, for example. (This is because there would be one network crossing per row.) With the compound SQL improvement, groups of insert statements are blocked together and sent over as one unit. This reduces the total number of crossings and hence elapsed time.

4.1.3 Considerations

There are several factors which can influence the total import time:

- Number of indexes specified

  As in the case of the LOAD utility, IMPORT can build indexes that are previously defined while the data is being inserted. The number of indexes and columns the index is on can affect the total time.

- Constraint checking

  The IMPORT utility checks for table constraints and referential integrity as the data is loaded; unlike LOAD which will only leave the database in ″check pending″ state and forcing use of SET CONSTRAINTS. The number and complexity of the constraints set on the table will contribute to the overall time.

- Logging

  Since IMPORT is doing actual SQL insert statements, DB2 logs all of these new rows. Once the log files are filled, a COMMIT is done. There is an overhead cost involved in this step, and the frequency of the COMMITS depends entirely on the different logging-related parameters.

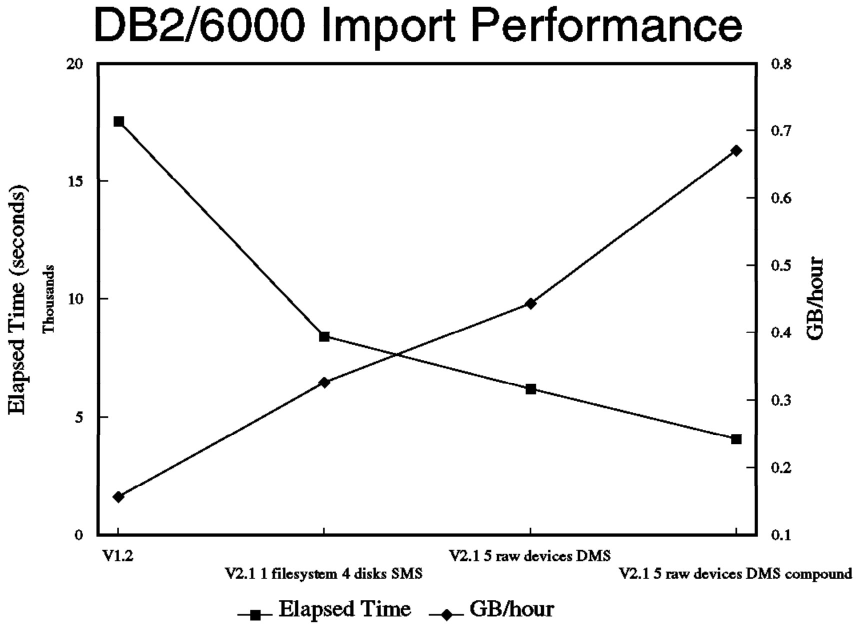


Figure 35. Import Performance

Figure 36. Comparison of Import Performance with Varying Configurations. Elapsed times in seconds.			
Scenario	**Elapsed Time**	**GB / hr**	**Average I/O wait %**
1 - V1.2	17541.75	0.156	65
2 - V2.1 SMS	8405.35	0.326	36
3 - V2.1 DMS	6179.72	0.443	15
4 - V2.1 DMS Compound SQL	4088.69	0.670	20

- Scenario 1 - Version 1.2 database created on 1 filesystem spanning 4 disks.
- Scenario 2 - Version 2.1 database created on 1 filesystem spanning 4 disks (SMS).
- Scenario 3 - Version 2.1 database with user tablespace created on 5 DMS raw devices.
- Scenario 4 - Version 2.1 database with user tablespace created on 5 DMS raw devices and COMPOUND=100 parameter specified to enable use of compound SQL.

The table definition from the ID2 database was used for this experiment (see A.4, "Internal Database 2 (ID2)" on page 102), except only 6 million rows were loaded (0.788 GB). The machine used was SERVER 1.

All configuration parameters were left at the defaults, except:

BUFFPAGE = 20000 SORTHEAP = 10000 SHEAPTHRES = 50000
NUM_IOSERVERS=6 (V2 only)

4.1.4 Performance Tips

The IMPORT utility seems to perform best when using database managed storage (DMS) as opposed to system managed storage (SMS). With SMS, the database files that are created which store the actual data pages must be extended each time a new page is filled. The resulting file system overhead is substantial. With DMS the space required for the tablespace is allocated at the start, so the operating system does not need to search for free space to put newly inserted data pages. There is a dramatic difference in the total number of system calls when using DMS, and this results in a faster IMPORT.

The use of compound SQL has a huge impact as well. The limit to the number of rows that can be blocked is 100; however, that alone can give a 30% improvement on a local IMPORT. This improvement would be amplified in the case of an IMPORT through a DDCS gateway.

Other tips:

- For small volumes of data, IMPORT is a better choice than LOAD. For LOAD, the tablespace has to be quiesced and the buffer pool is flushed out prior to starting the LOAD. Only for loads of a few thousand or more rows will Load be obviously faster.
- Use LOAD for LOBs instead of IMPORT. There are periods when the CPU actually becomes idle during IMPORT of LOBs due to inactivity. In addition, LOAD provides data buffers to help reduce I/O wait time. In one case where a 2 MB character LOB was IMPORTed and LOADed, LOAD was faster by approximately 35%.

4.2 Load

4.2.1 General Overview

The V2.1 LOAD utility is a quick and efficient method for inserting or appending data into tables. The LOAD utility is capable of handling all supported data types including large object and user-defined types. It is several times faster than the version 1.1.0 IMPORT utility and is targeted for loading large volumes of data.

4.2.2 Improvements

The LOAD utility is new for V2.1 and provides several performance-related features.

Unlike IMPORT (see 4.1, “Import” on page 58), which is actually executing multiple INSERT statments, LOAD writes data pages directly into the database. As a result, LOAD is almost entirely CPU bound. There is little time wasted in waiting for disk I/O (with the exception of LOBs). LOAD also does not do the formal row-by-row logging of IMPORT.

The LOAD utility allows for building indexes and gathering statistics for a table as part of the load. This is more efficient than executing the three phases separately. Keys are sorted and statistics are collected during the load phase, which removes some of the overhead involved in issuing separate CREATE INDEX and RUNSTATS statements.

For the loading of data records (particularly longs and LOBs), and in the index creation phase, the LOAD utility uses its own buffers allocated from the utility heap <UTIL_HEAP_SZ> database configuration parameter.

A reduction of 77% in elapsed time (4.3 times speedup) was observed between V1.2 and the combined scenario (#3) in V2.1. Data loading alone had up to a 92% reduction (13 times speedup).

Create Index was 15% faster in V2.1 in this experiment, but 40% reductions have been seen.

The slowdown in RUNSTATS in V2.1 was caused by the use of DMS raw containers and the current lack of index prefetching in V2.1 in this environment. As shown in 4.3, “DB2/2 Load, Create Index, RUNSTATS” on page 65, RUNSTATS in V2.1 is 30% faster than in V1.2 on OS/2 for both SMS and DMS file configurations, and this is similar to AIX results in these configurations.

4.2.3 Considerations

There are several factors which can influence the total load time:

- Number of indexes specified and statistics collection

  For larger tables the number of indexes and whether statistics are collected become key factors. If unique indexes are specified, any duplicate rows will be sent to an exception table resulting in significant overhead.

- Constraint checking

  The LOAD utility does not check table or referential integrity constraints. If such constraints exist, the table will be placed in ″check pending″ state and will be unusable until the SET CONSTRAINTS utility is run. The added overhead involved will depend on the number and type of constraints.

- Recoverability issues

Since no logging is done with the LOAD utility, a backup copy of the loaded data can be created for recovery purposes using the COPY YES option. This is the recommended approach, because LOAD performance will not be significantly affected. With the COPY YES option the same LOAD would take only a 3% hit.

Specifying COPY NO will save very little. However, if the <LOGRETAIN> is set to ON, then the table will be put in ″backup pending″ state and cannot be updated. It will remain in this read-only state until the database is backed up and cannot be connected to until then.

- Use of named pipe

  The LOAD utility can accept input from a named pipe. If a data generation program running on the server is used to direct the input data to the pipe, the program will consume CPU cycles otherwise usable by the LOAD utility and lengthen the total execution time.

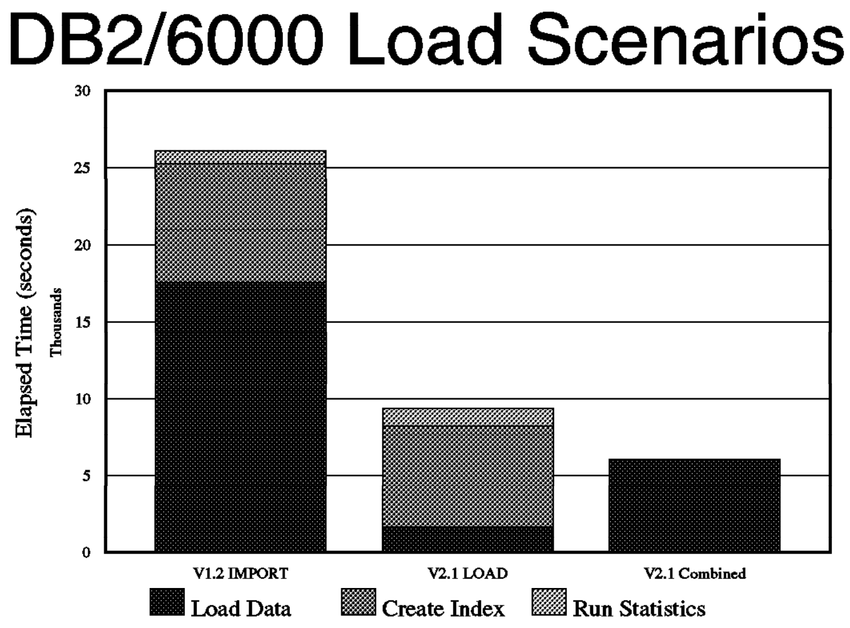


Figure 37. DB2/6000 Load Scenarios

Figure 38. Comparison of Different Data Loading Scenarios. Elapsed times in seconds			
Loading Stage	**Scenario 1 IMPORT, V1.2**	**Scenario 2 LOAD, V2.1**	**Scenario 3 LOAD/INDEX/RUNSTATS**
Data Loading	17541.75	1680.14	n/a
Create Index	7746.62	6540.12	n/a
RUNSTATS	818.47	1140.20	n/a
Total Time	26106.84	9360.46	6055.57

- Scenario 1

  Version 1.2 database created on 1 filesystem spanning 4 disks. Using 6 million row ID2 database. Data loaded through IMPORT utility. 5 indexes created after the IMPORT on integer and date columns.

All configuration parameters set to defaults except for <BUFFPAGE> 20000 and <SORTHEAP> 10000.

- Scenario 2

  Version 2.1 DMS database created on 5 raw containers each on its own disk. Using 6 million row ID2 database. Data loading through LOAD utility. 5 indexes created after the LOAD on integer and date columns. Detailed statistics gathered after the LOAD and index creation. All configuration parameters set to defaults except for <BUFFPAGE> 20000, <SORTHEAP> 10000, <UTIL_HEAP_SZ> 20000.

- Scenario 3

  Version 2.1 DMS database created on 5 raw containers each on its own disk. Using 6 million row ID2 database. 5 indexes created before the LOAD. Data loading, index creation, and detailed statistics gathering done through LOAD utility. 2 sort directories used for indexes. All configuration parameters set to defaults except for <BUFFPAGE> 20000, <SORTHEAP> 10000, <UTIL_HEAP_SZ> 20000.

Figure 39 demonstrates the linearity of load time with database size and the similarity of load rates for DMS and SMS.

Figure 39. LOAD Performance over Varying Database Sizes. LOAD is executed without building indexes and statistics collection, for a DMS database on 5 raw containers.

Database Size (GB)	**Elapsed Time (sec)**	**GB / hour**
0.0095	22.3	1.52
0.079	163.7	1.74
0.788	1684.4	1.68
1.18 (9M rows)	2534.0	1.68
1.18 (SMS with 5 disks)	2615.0	1.62
1.75 (13.3M rows)	3767.0	1.67

Figure 40. LOAD - build index vs. # of Sort Directories. Elapsed times in seconds

# of Sort Dirs	**LOAD time**
1	3444.89
2	3205.93
4	3135.26

The table definition from the ID2 database was used for this experiment, except only 6 million rows were loaded (0.788 GB); see A.4, "Internal Database 2 (ID2)" on page 102. There were 5 indexes created on a combination of integer and date keys. The sort directories were on 4 separate file systems and the sort buffer was set to 4096 4K pages. The machine used was SERVER 1.

4.2.4 Performance Tips

An important factor is the amount of memory allocated for <UTIL_HEAP_SZ>. Since the LOAD utility does not use any of the engine buffers such as the buffer pool <BUFFPAGE>, sort heap <SORTHEAP>, or database heap <DBHEAP>, these configuration parameters can be lowered and the utility heap can be increased to maximize use of available memory.

Using database managed storage (DMS) as opposed to system managed storage (SMS) had very little impact on the 1.18 GB test table. There was a slight improvement, but any benefits would only be apparent for much bigger databases. Similarly, changing the extent size from 16 to 32 had little impact during the load.

Other tips, particularly on LOAD command parameters:

- The LOAD utility works best on large quantities of data. For small volumes, IMPORT is a better choice.
- LOAD performs best when REPLACE mode is used rather than INSERT.
- Use multiple sort directories for temporary space in ″build index″ phase of the load. To minimize disk contention, these directories should be on separate file systems (one per disk). The more sort directories used, the less time spent in I/O wait during sort.
- If creating indexes with the LOAD utility, use the largest possible sort buffer. There is a significant improvement over using the default.
- Increasing the data buffer can help reduce I/O wait when loading long fields or LOBs. If indexes are built on the table as well, tuning is required to determine how much of the utility heap should go to the data buffer and how much to the sort buffer.
- If the COPY YES parameter is specified, use multiple target directories to reduce I/O time for backup image. In a case where 4 directories were specified, the additional overhead was less than 5%.

4.3 DB2/2 Load, Create Index, RUNSTATS

Measurements were performed to observe the improvements in V2.1 for Load (vs. V1.2 Import), Create Index, and RUNSTATS. These were measured both individually and as a combined operation in V2.1.

A reduction of 87% in elapsed time (7.4 times speedup) was observed between V1.2 and the combined scenario (#5) in V2.1. Data loading alone was seen to be up to 13 times faster than V1.2 Import; the reasons for this are given in 4.2, "Load" on page 61.

Create Index is over twice as fast in V2.1, thanks largely to the sorting of key values that is new to DB2/2 but was previously available in DB2/6000.

The machine used for these measurements was SERVER 5, described in B.5, "Configuration for machine SERVER 5 (OS/2)" on page 104.

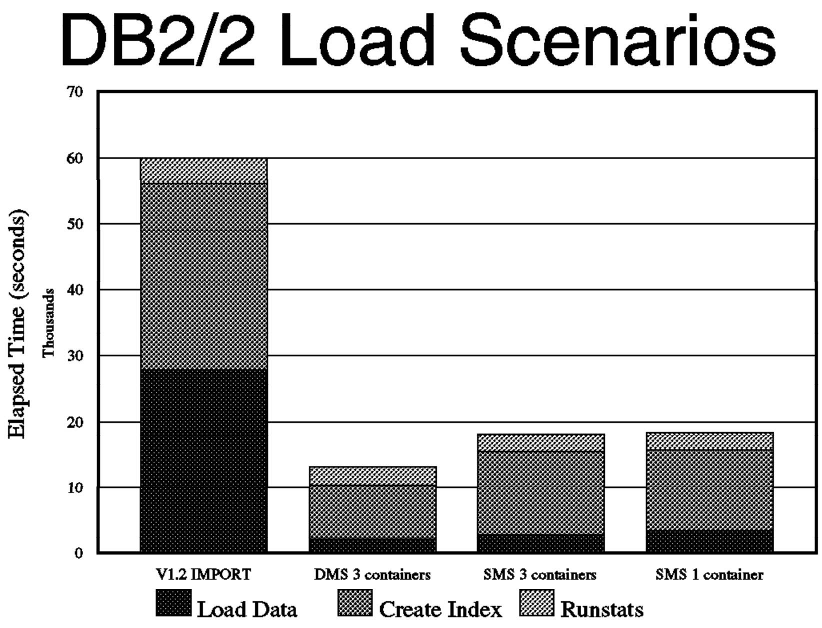


Figure 41. DB2/2 Load Scenarios

Figure 42. DB2/2 Comparison of Different Data Loading Scenarios. Elapsed times in seconds

Loading Stage	Scenario 1 - V1.2	Scenario 2 - DMS3	Scenario 3 - SMS3	Scenario 4 - SMS1	Scenario 5 - 1 Step
Data Loading	27821.40	2125.20	2899.20	3392.40	N/A
Create Index	28248.60	8280.00	12532.80	12285.00	N/A
RUNSTATS	3925.80	2794.20	2734.80	2698.20	N/A
Total Time	59995.80	13199.40	18166.80	18375.60	8053.85

- Scenario 1

  Version 1.2 database created on a single disk drive. Using 6 million row ID2 database (0.8GB; see A.4, "Internal Database 2 (ID2)" on page 102). Data loaded through IMPORT utility. 5 indexes created after the IMPORT on integer and date columns. All configuration parameters set to defaults except for <BUFFPAGE> 13000 and <SORTHEAP> 16000 (in 4K pages; actual value was 1000 in V1.2 units of 64K segments).

- Scenario 2

  Version 2.1 DMS database created on 3 containers, each on its own disk. Using 6 million row ID2 database. Data loading through LOAD utility. 5 indexes created after the LOAD on integer and date columns. All configuration parameters set to defaults except for <BUFFPAGE> 20000 and <SORTHEAP> 10000.

  ```
  LOAD FROM id2.tbl OF DEL MODIFIED BY COLDEL? MESSAGES id2.msg
  REPLACE INTO util.id2 USING j:\temp SORT BUFFER 4096;
  ```

- Scenario 3

  Same as Scenario 2, except SMS instead of DMS.

- Scenario 4

  Same as Scenario 2, except a single SMS container on one disk.

- Scenario 5

  Same as Scenario 2, except that the same 5 indexes were defined BEFORE the Load and built during it. In addition, statistics were collected during the Load. The Load statement used was:

  ```
  LOAD FROM id2.tbl OF DEL MODIFIED BY COLDEL? MESSAGES id2.msg
  REPLACE INTO util.id2 STATISTICS YES AND DETAILED INDEXES ALL
  USING j:\temp SORT BUFFER 4096 ;
  ```

Here are some utilization figures for the above measurements as supplied by SPM/2. The disk utilization is average % time busy for the database disk(s).

	Scenario 1	Scenario 2
Data Loading	30% CPU, 70% disk	100% CPU, 25% disk
Create Index	65% CPU, 70% disk	65% CPU, 30% disk
RUNSTATS	100% CPU, 5% disk	100% CPU, 5% disk

4.4 Load vs. Import

4.4.1 General Overview

LOAD vs IMPORT: which one is better? There is no clear winner. Both methods are important and both methods have their advantages. In most cases, LOAD is much faster than IMPORT, but there are times where IMPORT is a better choice (for example, if you need frequent consistency points). IMPORT is also faster when loading small amounts of data (up to 2000 rows) to an empty table, or when appending small amounts of data (up to 10,000 rows) to an existing table.

4.4.2 Improvements

LOAD is new for V2.1. To compare V2.1 with V1.2 is to compare LOAD with IMPORT. LOAD can be, in some cases, up to ten times faster than import. LOAD has also been designed to incorporate index building and statistics collecting. Thus LOAD will be faster than IMPORT both directly (inserting data) and indirectly (creating indexes, gathering statistics).

The following tests were done with an empty database, loading uniform, randomly generated data. The create table command was:

db2 ″CREATE TABLE mytable (FIELD0 INT NOT NULL, FIELD1 INT NOT NULL, FIELD2 DECIMAL (16,8) NOT NULL, FIELD3 DECIMAL (16,8) NOT NULL, FIELD4 CHAR(10) NOT NULL, FIELD5 CHAR(10) NOT NULL, FIELD6 VARCHAR(100) NOT NULL, FIELD7 VARCHAR(100) NOT NULL)″

This is from the ID3 database (see A.5, “Internal Database 3 (ID3)” on page 102 for details.) The machine used was SERVER 2.

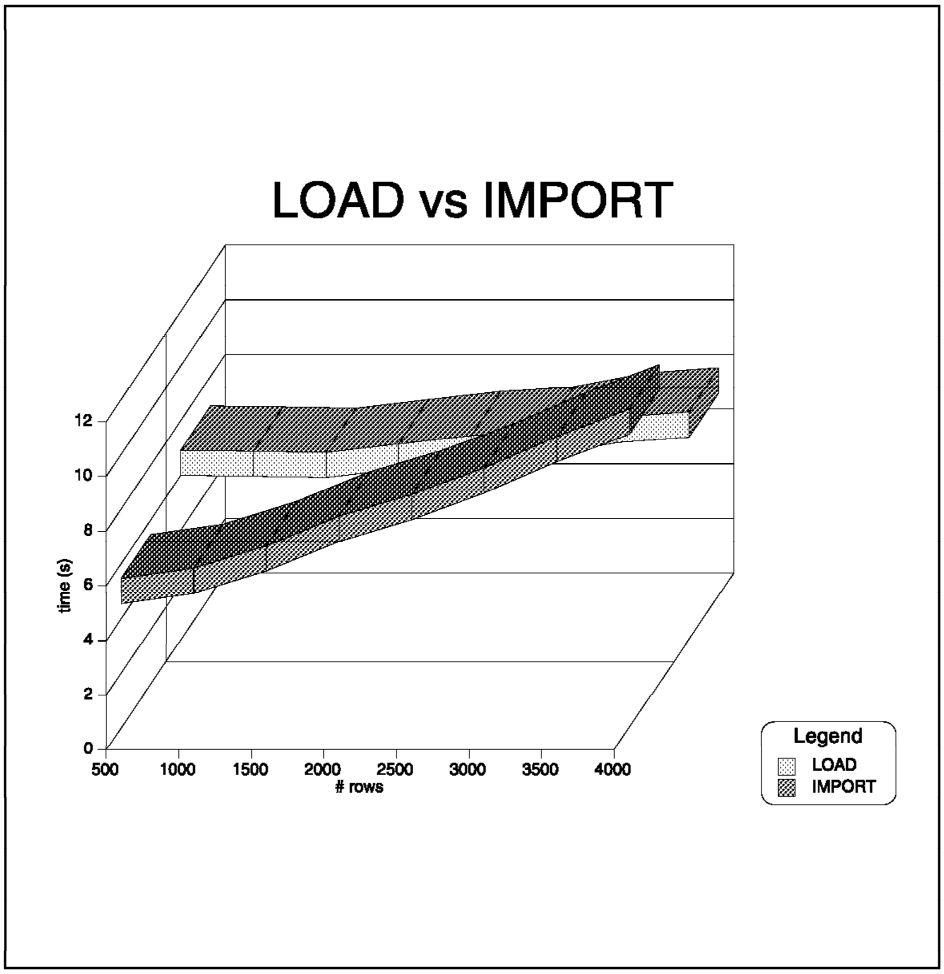


Figure 43. LOAD vs IMPORT, no indexes.

As you can see in Figure 43, both methods of inserting data progress linearly. In both cases, the amount of time it takes to load the data is directly related to the amount of data to be inserted. However LOAD has a much better slope. When loading 2000 rows or more, LOAD is faster than IMPORT. The same pattern is valid for any number of indexes. Though it may not be obvious from this graph, the cross-over point is at 1500 rows.

The numbers follow:

4.4.3 Considerations

Figure 44. LOAD vs IMPORT: no indexes. Elapsed times in seconds

Rows added:	**500 rows**	**1000 rows**	**1500 rows**	**2000 rows**	**2500 rows**	**3000 rows**	**3500 rows**	**4000 rows**
LOAD	6.462	6.427	6.379	6.692	6.992	7.157	7.664	7.858
IMPORT	5.000	5.378	6.195	7.276	8.069	9.043	10.191	11.200

Figure 45. LOAD vs IMPORT: 1 index. Elapsed times in seconds								
Rows added:	**500 rows**	**1000 rows**	**1500 rows**	**2000 rows**	**2500 rows**	**3000 rows**	**3500 rows**	**4000 rows**
LOAD	7.392	7.170	7.366	7.796	8.076	8.234	8.645	8.908
IMPORT	5.547	6.494	7.968	9.063	10.232	12.156	13.475	14.852

Figure 46. LOAD vs IMPORT: 2 indexes. Elapsed times in seconds								
Rows added:	**500 rows**	**1000 rows**	**1500 rows**	**2000 rows**	**2500 rows**	**3000 rows**	**3500 rows**	**4000 rows**
LOAD	9.000	8.541	8.750	8.980	9.045	10.001	10.049	10.591
IMPORT	7.376	7.999	9.748	11.242	12.842	15.280	16.575	18.474

Another common scenario for loading data is when you are adding data to existing data. In these cases, LOAD has to read the data and rebuild the indexes. But for Import there is much less work because it just expands the existing index one row at a time. If you are inserting very few rows (0 to 5000) then IMPORT is definitely much faster. But beyond 10000 rows, LOAD quickly becomes the better alternative. Please refer to the following table.

Figure 47. Appending data using Load and Import					
Size of Data	**Size of increase**	**Number of indexes**	**% increase**	**LOAD**	**IMPORT**
4,000,000 rows	400,000 rows	no indexes	10%	4 min	82 min
4,000,000 rows	400,000 rows	1 index	10%	21 min	638 min
4,000,000 rows	40,000 rows	1 index	1%	19 min	71 min
4,000,000 rows	4,000 rows	1 index	0.1%	13 min	9 min

As Figure 47 clearly shows, LOAD is not good at inserting 5000 rows or less to existing data. The cross-over point here is at 5000 rows, but as you increase the number of indexes, the cross-over point goes down.

4.4.4 Performance Tips

- LOAD is generally much faster than IMPORT. In 95% of cases, LOAD is the method of choice.
- IMPORT will be faster when the amount of data to be inserted is small: 0 to a few thousand rows.
- IMPORT will be faster when the amount of data to be inserted represents less than 1% of the data already in an indexed table. (The reason is that LOAD cannot simply add keys to an existing index; instead, it gathers key values from both old and new rows and rebuilds the indexes in their entirety.)
- For very large volumes of data, increasing the data by 10% or more of an indexed table will be slow for both IMPORT and LOAD. You may consider dropping the indexes, LOADing the data, and re-creating the indexes.

4.5 Create Index

4.5.1 General Overview

Adding indexes can greatly improve table access times by setting up an efficient system of retrieving rows. Creating indexes requires scanning the entire table and sorting the key value(s) extracted from each row. This can prove to be quite time-consuming for large tables, so database design and choice of configuration parameters play a key role in reducing the time required to create indexes.

4.5.2 Improvements

Several changes in this version have improved performance of CREATE INDEX. The index manager has been revamped so inserting new records into an index is more efficient. The introduction of the media spanning line item has made disk I/O cheaper, resulting in lower I/O wait periods during the scanning, sorting and building phases. Sorting itself has been made much more efficient.

For the CREATE INDEX process, there is an option to use SORT to order the keys. This can be controlled with the <INDEXSORT> configuration parameter. In V1, DB2 for AIX would use SORT to order the keys, and DB2 for OS/2 would not use SORT. With V2, you have the choice. By setting <INDEXSORT> = no, each key is inserted into the index right after its row is read. To do this, the index manager needs to traverse the index tree to determine where the key to the row would reside, possibly reading index pages in the process. Now we can avoid this lengthy process with DB2 for OS/2.

On DB2 for AIX, CREATE INDEX elapsed times are as much as 40% shorter for V2.1 than for V1.2. An example of a 16% reduction is in Figure 38 on page 62.

4.5.3 Considerations

There are several factors which can influence the total CREATE INDEX time:

- Number of rows
- Nature of data

  The data type of the column on which the index is created will affect the performance. Creating an index on a native data-type (such as FLOAT, INT) column would generally be faster then creating an index on a fixed-length data-type (such as CHAR, DECIMAL) which in turn be faster than creating an index on a variable-length data-type (such as VARCHAR).

- Cluster factor of data

  If the column on which the index is to be created is already in sorted order, this fact will be discovered and shortcuts taken during the sort phase. Advance sorting is even more desirable when <INDEXSORT=NO>, but NO is recommended only if necessary.

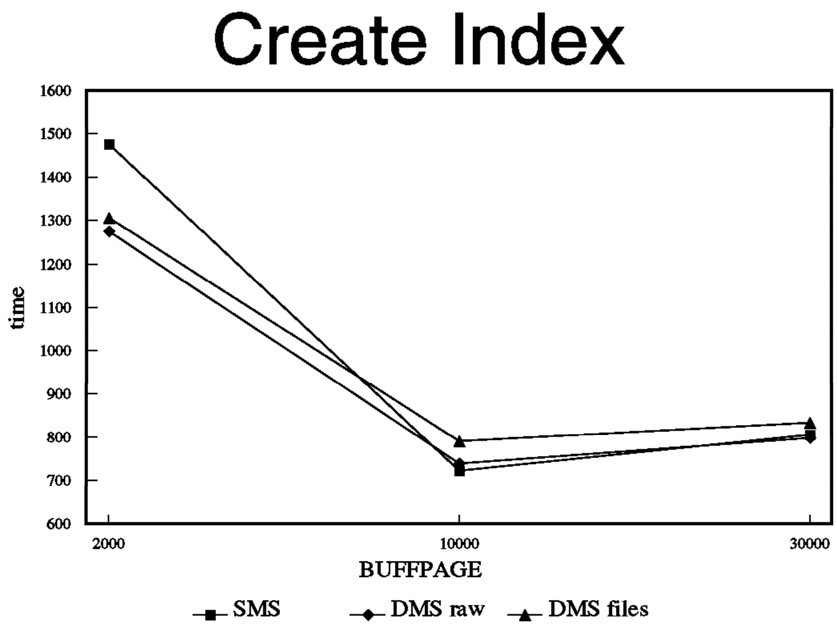


Figure 48. Create Index per buffpage

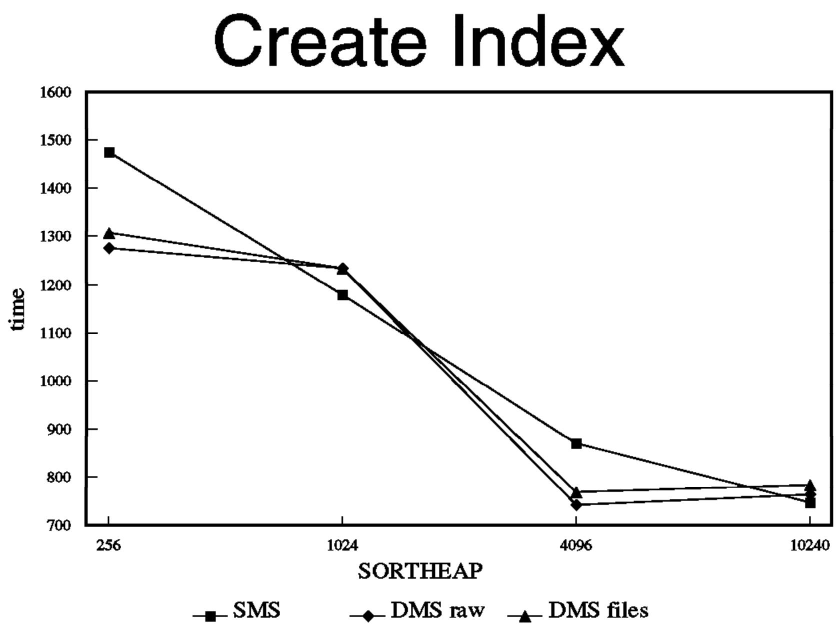


Figure 49. Create Index per sortheap

Figure 50. Impact of SORTHEAP and BUFFPAGE on CREATE INDEX. Elapsed times in seconds, config parms in 4K pages							
Database Configuration	Sortheap (Buffpage = 2000 pages)				Buffpage (Sortheap = 256 pages)		
	256	1024	4096	10240	2000	10000	30000
SMS	1476.6	1179.7	870.9	746.9	1476.6	721.8	804.8
DMS raw	1275.0	1234.1	742.8	763.3	1275.0	738.0	797.6
DMS files	1307.6	1234.6	768.1	783.3	1307.6	790.8	832.0

The database used for this experiment was the ID2 database (9 million rows, 1.2GB; see A.4, "Internal Database 2 (ID2)" on page 102). The index was created on an integer field, which was unclustered. The machine used was SERVER 1.

4.5.4 Performance Tips

An important tuning variable that influences the time for CREATE INDEX is the amount of memory resources allocated for the task. The <SORTHEAP> and <BUFFPAGE> parameters are the two memory-related parameters with the greatest influence. Both contribute significantly in reducing I/O wait during the sort and build phases. The latter generally takes up the most time while creating an index, due to high disk I/O. Reducing wait time in this phase makes the largest impact in reducing the overall time.

Another factor is whether the index is in a database managed storage (DMS) or a system managed storage (SMS) tablespace. The reduction in I/O wait with DMS is apparent but not major on its own. Coupled with changes in memory resources, the improvement with DMS is significant. Tests were conducted using both DMS files and raw devices and the results were similar, with DMS raw coming out slightly ahead.

Other tips:

- Do not use the default values for <BUFFPAGE> and <SORTHEAP>.
- Raising the values for <BUFFPAGE> and/or <SORTHEAP> improve performance; however, for lower values <SORTHEAP> is slightly more effective. E.g. <SORTHEAP> set to 1024 pages will have more of an impact than <BUFFPAGE> set at 1024.
- Increasing <BUFFPAGE> and/or <SORTHEAP> beyond a certain threshold value will actually result in a degradation of performance. The ideal ratio of the two parameters requires tuning. Generally, a higher <BUFFPAGE> parameter is more beneficial, since it is more generally used by other utilities and applications.
- Try to have the temporary tablespace used in the sort phase spread out over several disks. When the sort heap gets full and has to overflow into the temp space, there are periods of high disk activity as the data is written out to the temp space. Having this space distributed over multiple disks will help to reduce I/O wait.
- Prefetch quantity has little impact on overall elapsed time. In one experiment where I/O waits were measured with the default prefetch size of 16 pages and then with 32 pages, I/O waits were significantly reduced in the latter; however, without any tuning for <SORTHEAP> or <BUFFPAGE>, there is little additional gain.
- Keep <INDEXSORT>=yes.

4.6 RUNSTATS

4.6.1 General Overview

RUNSTATS is used to generate statistics on tables and their indexes. These statistics are then used by the optimizer to produce the best plans for accessing the data. For example, RUNSTATS will find out if any given index is clustered or not. If the index is unclustered, then data is more likely to be accessed by table scan, but if the index is clustered, then an index scan might be chosen instead.

There are many such statistics which influence the optimizer′s decisions. Therefore execution of RUNSTATS is mandatory to achieve proper optimization of queries. After the initial loading of data, RUNSTATS need only be run when the data has changed significantly.

RUNSTATS should always be run after a REORG, CREATE INDEX, LOAD, and large IMPORT. In some cases, RUNSTATS should be run weekly. It is a quick service that is very helpful and can be run when there is low table activity. So even though the performance of RUNSTATS itself may not always be the issue, RUNSTATS is closely tied in with the overall performance of the database. The performance of a RUNSTATS is closely tied to the nature of the database and the type of data it contains. For these measurements, we used different sizes of the ID3 database (see A.5, "Internal Database 3 (ID3)" on page 102 for details).

4.6.2 Improvements

In V2.1, RUNSTATS collects many new statistics. It is also more versatile, allowing you to choose which statistics to collect.

Here is a short list of the new statistics:

- most frequent values per column
- quantile statistics for columns
- page fetch pairs and clusterfactor for indexes; these improve upon clusterratio in previous releases

4.6.3 Considerations

In its most common usage, RUNSTATS is run against a table and all of its indexes. However, if you have just created an index, you may choose to run statistics just for the indexes.

A new option in V2.1 is ″WITH DISTRIBUTION″, which is very useful for many real-world scenarios. In databases with non-uniform data distribution, this option can help achieve much better performance.

Another new option in V2.1 is ″DETAILED INDEXES.″ This is a relatively inexpensive way to collect more information about index characteristics and can allow the optimizer to produce improved access plans.

Figure 51 on page 74 shows the major factor which affects RUNSTATS run time in most cases: database size. Although not reflected in this figure, requesting WITH DISTRIBUTION can be very time-consuming: this is because, in the worst case, RUNSTATS might need to record information about every column in every row as it determines what the most frequent values are, for example. The larger the values of NUM_FREQVALUES and NUM_QUANTILES, the longer RUNSTATS WITH DISTRIBUTION will take.

The elapsed time of RUNSTATS ... AND DETAILED INDEXES ALL is affected by the number of rows, the number of indexes, and how well the indexes are clustered. (When clustering is poor, the same data pages can be read several times during the index scan executed by RUNSTATS, if the buffer pool is small.)

Figure 51. RUNSTATS elapsed times in various scenarios						
indexes	**detailed**	**distribution**	**200k rows**	**400k rows**	**600k rows**	**800k rows**
0	-	yes	11m16	21m59	33m04	44m28
1	no	no	11m16	21m59	33m13	44m24
1	yes	no	11m16	21m57	33m12	44m26
1	yes	no	11m13	21m55	32m59	44m26
2	yes	no	11m16	22m03	33m10	44m40

4.6.4 Performance Tips

- RUNSTATS is absolutely necessary after a REORG, LOAD, CREATE INDEX and such.
- RUNSTATS should be run when 5%-10% or more of the data in a table has changed.
- Specifying as large a buffer pool as possible can reduce RUNSTATS time significantly for poorly clustered indexes.

4.7 Backup/Restore

4.7.1 General Overview

Periodic backing up of databases is essential in reducing the impact of system failures and the resulting recovery process. The V2.1 BACKUP and RESTORE utilities have been changed greatly since V1.2 to enhance both performance and functionality.

With disk as the backup medium, for BACKUP we see performance of 14GB/hour, and for RESTORE 9GB/hour. However, the performance numbers are highly dependent on the speed and quantity of the disks or tapes used as the backup media, and the number and characteristics of the controllers used to attach them.

4.7.2 Improvements

In V2.1, performance of BACKUP and RESTORE have been improved by introducing multiple buffers and multiple I/O devices. Previously, reading from the database and writing to disk (for BACKUP; vice-versa for RESTORE) was a serial process. As a result, the backup process would often be bottlenecked on the write side with the read side waiting for the write to be completed. With support for multiple I/O devices, data read from the database can be written out across multiple devices, hence reducing I/O wait on a single device. With multiple buffers available, data can be continually read into several buffers without having to wait for an individual buffer to be returned after a write.

4.7.3 Considerations

There are several factors which can influence the total backup/restore time:

- Offline / Online

  In an offline backup the utility has exclusive access to the database until the backup is complete. In the case of an online backup, logging for rollforward recovery can take up some disk and CPU activity, depending on the application.

- Other I/O activity on backup devices

  In the case of disk devices used for the backup, any other activity on these disks (e.g. database or application related) will hamper media writer or reader processes from full access. The resulting uneven distribution in media process and device activity will degrade performance and produce disproportionate backup image files.

- I/O device controller bandwidth

  The quantity of data that is either read or written out is limited by the total bandwidth of the controller that the devices (either database or backup) are connected to. Hooking up more devices to reduce I/O wait may not make a significant improvement if the I/O controller is saturated to its bandwidth limit.

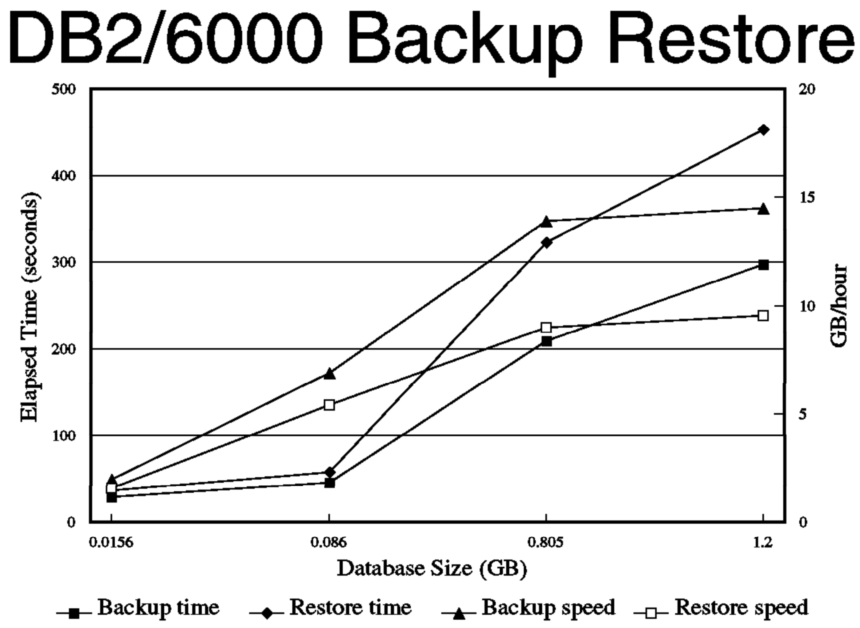


Figure 52. Backup/Restore

Figure 53. Backup and Restore Performance over Varying Database Sizes. Elapsed times in seconds.

Database Size (GB)	Backup		Restore	
	Elapsed Time	GB / hour	Elapsed Time	GB / hour
0.0156	28.8	1.95	35.9	1.56
0.0860	45.1	6.86	57.2	5.41
0.805	208.7	13.9	322.4	9.00
1.20	297.8	14.5	453.4	9.53

The database used for this experiment was the ID2 database; see A.4, "Internal Database 2 (ID2)" on page 102. The data was spread on 5 raw DMS containers and the backup was made to 2 independent disk devices (1 filesystem on each disk).

Figure 54. Comparison of Different Backup Scenarios. Elapsed times in seconds. CPU usage in %

Scenario	Elapsed Time	CPU Usage		
		User	System	I/O Wait
1 - V1.2	498.3	2	19	79
2 - V2.1, 1 filesystem	439.8	2	28	70
3 - V2.1, 4 filesystems	426.9	2	29	69

Scenario 1 -

Version 1.2 database created on 1 filesystem spanning 4 disks. The database was backed up to 1 filesystem spanning 4 disks as well. The number of buffers was the default and the buffer size was set to 1024 4K pages. All other configuration parameters set to defaults.

Scenario 2 -
Version 2.1 database created on 1 filesystem spanning 4 disks. The database was backed up to 1 filesystem spanning 4 disks as well. The number of buffers was the default and the buffer size was set to 1024 4K pages. All other configuration parameters set to defaults.

Scenario 3 -
Version 2.1 database created on 1 filesystem spanning 4 disks. The database was backed up to 4 filesystems each on its own disk. The number of buffers used was 16 and the buffer size was set to 1024 4K pages. All other configuration parameters set to defaults.

The database used for this experiment was the ID2 database; see A.4, "Internal Database 2 (ID2)" on page 102. The actual size of the database was approximately 1.5 GB. For the V2.1 scenarios, the default database configuration was used (i.e. SMS - 1 default container); previous measurements with 5 DMS raw containers for the database showed a significant improvement over the default SMS setup (approximately 25%).

4.7.4 Performance Tips

The use of buffers has a significant performance impact and the memory allocated for their use comes from the <UTIL_HEAP_SZ>. This parameter also provides memory for buffers used in the LOAD utility, so a proper balance is needed so that the utility heap is used to its full potential. As in the case of the LOAD utility, none of the engine buffers such as the buffer pool are used, so the amount of memory allocated to these parameters can be reduced in order to make more available for the utility heap.

Using database managed storage (DMS) as opposed to system managed storage (SMS) had a significant impact when performing a backup on the 1.7 GB test database. With DMS the AIX file system is bypassed for reading and writing to and from the database, so much of the overhead involved in a backup (reading) or restore (writing) is reduced.

Other tips:

- Use multiple devices to reduce I/O wait for backup and restore. There is a substantial difference between doing a backup to one file system spanning four disks and four file systems on four disks.
- If adding multiple devices does not improve performance after a certain point, examine the hardware configuration regarding which devices are on which controllers. Try to have database and backup devices on separate controllers. For further improvement try to space out database disk devices on different controllers, as well.
- Always use more than one buffer per backup device.
- Use more buffers of smaller size rather than few buffers of large size. Since the writer processes take full buffers off a queue, having multiple buffers available will reduce I/O wait more than writers waiting for one large buffer to be filled.
- More memory allocated for buffers does not necessarily improve performance. At one point, there will be a threshold where using more memory for buffers actually degrades performance.
- If a DMS database is used, try to spread it out on as many disks as possible to reduce I/O wait.

4.8 Reorg

4.8.1 General Overview

Most databases that are not completely read-only will benefit from Reorg every now and then. It will reorganize your tables by compacting the fragmented data and reconstructing the indexes, and will generally speed up your queries. It is especially useful for databases with frequent updates and deletions. Reorg should always be followed by a RUNSTATS execution.

4.8.2 Improvements

For Version 2.1, Reorg has been redesigned and rewritten to use less CPU time. As well, it has benefitted from improvements in index rebuilding, as discussed in 4.5, "Create Index" on page 70.

4.8.3 Considerations

A Reorg operation consists of two basic phases:

a) Compaction of pages. This phase includes reading all the table′s rows and writing them into new pages without fragmentation. Usually the rows are read in sequence according to some index, resulting in a 100% clustered table. The lower the cluster factor of the resequencing index, and the larger the table relative to the buffer pool size, the slower Reorg will be, because of the potential for multiple random I/O′s against each data page.

   The Reorg can also be performed with or without a temporary tablespace. Adding a temporary tablespace can increase performance if they are created on separate disks, thus allowing for some extra parallel I/O. If you do not specify a temporary tablespace, the database manager will use the current tablespace for the Reorg.

b) Rebuilding of indexes. Reorg recreates the indexes, so that having extra indexes will slow down Reorg by the time needed to recreate them. (Note, however, that there is no advantage to dropping such extra indexes before the Reorg and re-creating them afterwards.)

Here are some elapsed times of Reorg performance. These measurements were performed on SERVER 2 with 512MB of memory. The table which was reorganized was the ID3 database with 1 million rows (134 MB of data plus 38 MB of indexes) built on a single system-managed disk. For the definition of the ID3 table and its indexes, see A.5, "Internal Database 3 (ID3)" on page 102.

The table was reorganized according to:

- the I1 index, which had a cluster ratio of 3%; or
- the I2 index, which had a cluster ratio of 100%; or
- no index at all.

A 160MB buffer pool was used (BUFFPAGE = 40000).

Figure 55. Performance results for Reorg (elapsed times in seconds)

Test #	DB2 Version	Index used to resequence	Temporary tablespace	Number of indexes	Elapsed Time (secs.)
1	1.2	I1	none	2	2337
2	2.1.0	I1	none	2	1063
3	2.1.0	I2	none	2	650
4	2.1.0	none	none	2	637
5	2.1.0	none	none	0	124
6	2.1.0	I1	SMS	2	873
7	2.1.0	I1	DMS	2	945

Test 2 vs. Test 1 shows an improvement of 55% since DB2 V1.2: Reorg is over twice as fast. The other tests show some behaviors of Reorg in V2.1:

- Test 3 shows that resequencing on a clustered index is much faster than on an unclustered one (Test 2).
- Test 4 shows that not resequencing the data is only slightly faster than resequencing on a perfectly clustered index (Test 3).
- Test 5 is not particularly practical, but it shows the basic speed of compacting data with no sequencing or index rebuilding costs included. The compaction rate here was about 4GB per hour.
- Tests 6 and 7 show an SMS temporary tablespace to be preferred over a DMS one, and either to be faster than not specifying a temp tablespace at all (Test 2).

Additional measurements of Reorg were conducted as part of the SPEC benchmark. Improvements of 47% were found on OS/2, 58% on AIX; refer to query 39 in Figure 7 on page 19 and Figure 8 on page 21.

4.8.4 Performance Tips

The simplest way to determine whether Reorg is necessary is by running REORGCHK. Note that REORGCHK has two forms: one simply reports on the current statistics in the catalog (which may be out of date if RUNSTATS has not been run recently); the other form invokes RUNSTATS before doing the reporting.

After every RUNSTATS, watch closely the cluster ratio: if you run Reorg based on an index with a low cluster ratio, expect it to be slow. Don′t let the cluster ratio fall below 80. If you have just loaded data in non-sequential order, or the cluster ratio is very low, and you want to reorganize the table based on an index, then Reorg can use all the memory and buffer pool it can get.

Other tips:

- In informal measurements SMS tables proved to be faster than DMS for Reorg. This is caused by the system prefetching services which are not available in DB2 V2.1.0 for indexes in DMS.
- SMS temporary tablespaces are currently faster than DMS temporary tablespace, because of pathlength differences.
- Avoid unnecessary indexes - they will slow down Reorg because it has to recreate all the indexes each time. Check Explain outputs to see if all your indexes are being used.
- Since Reorg requires exclusive access to the table, try to run Reorg only when there is no table activity.
- Reorg using an index is greatly affected by the database <BUFFPAGE> configuration parameter. Keep <BUFFPAGE> as large as possible; the default value will be too low.

- A large <SORTHEAP> will benefit the index recreation phase of Reorg. Ensure that <SHEAPTHRES> is correspondingly high (at least twice <SORTHEAP>).
- When choosing an index on which to reorganize (order) a table, you may be inclined to choose the unique index on the most frequently used search column; say, Order_ID. However, clustering the table on such a column will not reduce I/O if the only predicates against that column are equals predicates; in such cases only one row and one data page would be accessed for the table, regardless of how the rows are ordered.

  A better approach is usually to reorganize the table on one or more columns that frequently have predicates for which multiple rows qualify. If you resequence the table by Order_date, for example, finding all orders for the last month would cause the minimal number of data pages to be accessed.

5.0 Miscellaneous Observations

5.1 Crash Recovery

5.1.1 General Overview

Crash recovery is DB2′s process of restoring a database to a consistent state after, for example, a power failure. During recovery, database updates previously recorded in the log are re-done or undone.

In our tests, a crash was simulated by destroying all the db2 processes after a long chain of transactions/updates.

5.1.2 Improvements

There is significant improvement from Version 1. Running simple tests have shown a 62% improvement. The tests were performed on SERVER 2, and the crash occured after 50000 updates.

Here are the recovery times. In this test, the configuration parameters were kept the same (default values), and the 50000 updates were done through one user connection, with a Commit after each update.

Figure 56. Version 1.2 - Version 2.1 comparison of recovery and update times

Action	Version 1.2	Version 2.1	% Improvement
Update (sec.)	2241	444	80
Recovery (sec.)	3223	1238	62

If your database scenario includes a similar workload, then you can put a ″ceiling″ on the recovery time by issuing a CONNECT RESET. The CONNECT RESET of the final (or only) user connection forces the contents of the buffer pool to be written to disk, and therefore reduces the amount of necessary recovery to zero. This buffer pool flush, however, can take a long time if the pool is large and contains many pages of updated data.

5.1.3 Considerations

The two main considerations for recovery time are:

a) amount of work that has to be recovered;
b) efficiency with which the recovery processing can be done.

The amount of work to be recovered is usually the amount of log content that has been written since the last soft checkpoint. (These soft checkpoints occur every time another <SOFTMAX> percent of a log file (<LOGFILSIZ>) is filled.) However, if some very frequently updated pages are not written out from the buffer pool to disk, a soft checkpoint may not be able to advance the position at which to start recovery. See 5.1.4, “Performance Tips” on page 82 for more information on the configuration parameters affecting recovery.

The efficiency of recovery can be maximized by having a sufficiently large value of <NUM_IOCLEANERS>; this also helps reduce the amount of work to recover. Increasing <BUFFPAGE> from its production value may reduce the amount of repeated writes of the same pages if multiple transactions against the same pages are being re-done.

The results in Figure 57 on page 82 show that recovery time is linearly related to the size of the work unit. For the example given, if you disconnect after every 60000 updates, then you limit the recovery time to a maximum of about one hour.

Figure 57. Recovery as affected by workload

Number of updates	**Updates time (sec.)**	**Recovery time (sec.)**
20000	838	1181
40000	1718	2430
60000	2588	3702

The other major factor affecting recovery is the number of I/O cleaners:

Figure 58. Recovery as affected by I/O cleaners

<NUM_IOCLEANERS>	**Recovery time (sec.)**
0	2409
1	1852
2	1673

For reasonable recovery time you need at least one I/O cleaner. For the benefit of transaction performance, you may want to consider using more.

Note: all of these measurements were taken on the IB1 database (see A.3, "Internal Benchmark/Database 1 (IB1)" on page 102), running on the SERVER 2 machine.

5.1.4 Performance Tips

Configuration parameters such as <LOGFILSIZ> and <SOFTMAX> are important, but the best way to minimize the cost of recovery is frequent commits. To effectively reset the recovery time to zero, you can deactivate the database by having all users disconnect from it.

Recovery time can also be reduced by decreasing the size of the primary log files <LOGFILSZ> and increasing the amount of page cleaning. To promote page cleaning, you can increase <NUM_IOCLEANERS> and decrease <CHNGPGS_THRESH>, but you may see a slight performance deterioration in transaction processing if <CHNGPGS_THRESH> is made too low.

The basic rule of thumb for choosing the number of I/O cleaners is: one I/O cleaner for every physical device, plus one more. However, when a new physical device is added to the system, you might forget to increase the number of I/O cleaners; if you believe this may occur, allocate extra I/O cleaners, since they don't significantly hamper performance.

5.2 CONNECT

5.2.1 General Overview

The elapsed time of CONNECT is a key factor in the performance of some applications. First, we will look at the general performance characteristics of the different types of CONNECT, and then we will suggest ways to improve the connect time. All these measurements were done based on the SPEC database. For a description of the SPEC database, please refer to A.1, “SPEC” on page 101.

Here are some general rules for CONNECT performance:

a) There is a large overhead for the first application that connects to a database. On our (slow) SERVER 3 machine, for a local connection, this overhead is composed of 1 second base + 1 second for each 10000 pages of <BUFFPAGE>.

   The connect time can be expressed with the following function:

   Connect_Time = 1 + (0.0001 * <BUFFPAGE>) in seconds (on RS/6000 550)

   Subsequent connects are much faster. Please refer to Figure 59 on page 84. A common method to work around this 1st connect overhead is to keep a permanent ″dummy″ connection to your database. This allows applications to connect and disconnect efficiently.

b) A local connect is much faster than a client/server connect, even with <AUTHORIZATION> = CLIENT.

c) The first time you connect to a database, the database directory is cached. If you then disconnect all applications, the next connect will be somewhat faster. Of course you will still pay the first-connect overhead. Having a ″dummy″ connect will improve on this aspect as well.

5.2.2 Improvements

A new concept in V2.1 leaves an idle agent in existence after an application disconnects. This allows for a significant performance improvement, because when a new application attempts a connection, an agent can be re-used. This improvement applies to both local and client/server configurations for OS/2, but only to local configurations for AIX.

Another improvement for V2.1 is the database directory caching. After a db2start, the first connect will load the database directory into the cache. When all applications disconnect, the first application to re-connect will be spared having to re-load the database directory into the cache.

The following tests were done on SERVER 3 for the local server. In the Client/Server tests, the server is SERVER 3 and the client is SERVER 2.

5.2.3 Considerations

Figure 59. Connect Times on a Local Server				
<BUFFPAGE>	**Connect #1**	**Connect #2**	**Connect #3**	**Connect #4**
20000	2.94	0.14	0.13	0.13
40000	4.96	0.15	0.13	0.15
60000	6.87	0.15	0.13	0.14
80000	9.10	0.16	0.15	0.15

As the data shows, the 1st CONNECT time grows linearly with the <BUFFPAGE> configuration parameter.

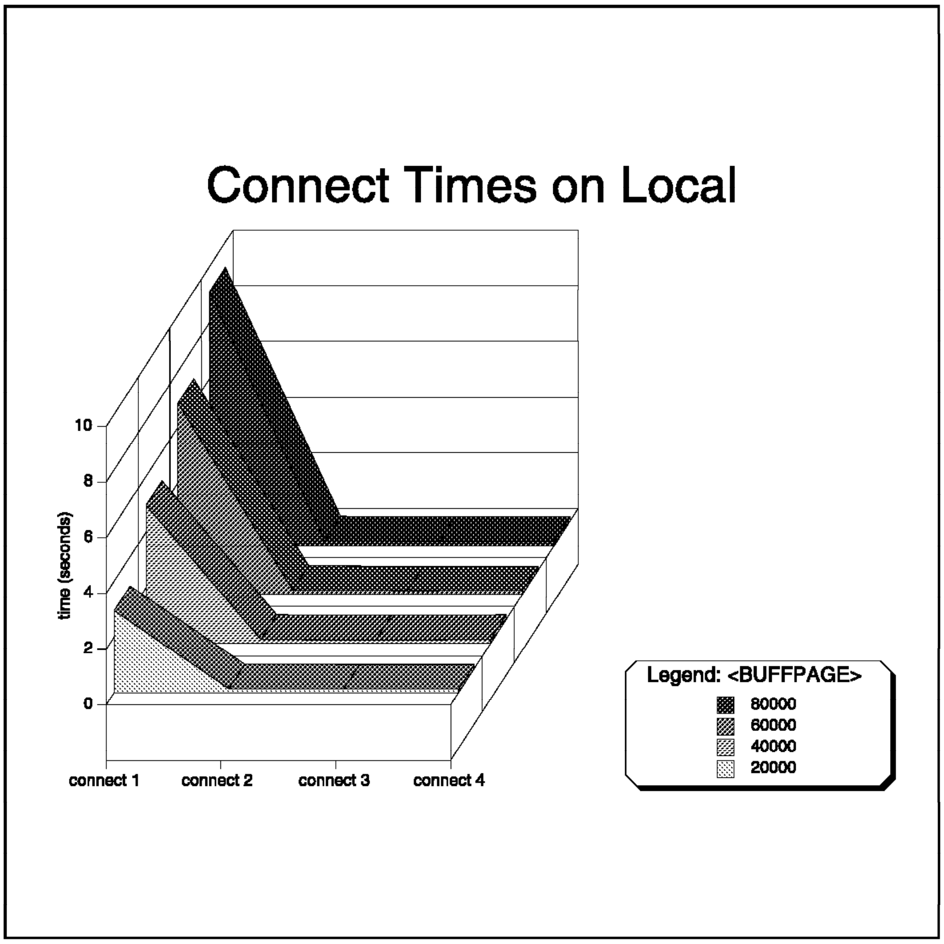


Figure 60. Connect Times on a Local Server

Figure 61. Connect Times on a Client/Server, <AUTHENTICATION> = CLIENT				
<BUFFPAGE>	**Connect #1**	**Connect #2**	**Connect #3**	**Connect #4**
20000	4.61	2.37	1.85	1.95
40000	7.29	2.07	2.19	2.40
60000	8.92	1.99	1.91	1.80
80000	10.91	2.02	2.01	2.07

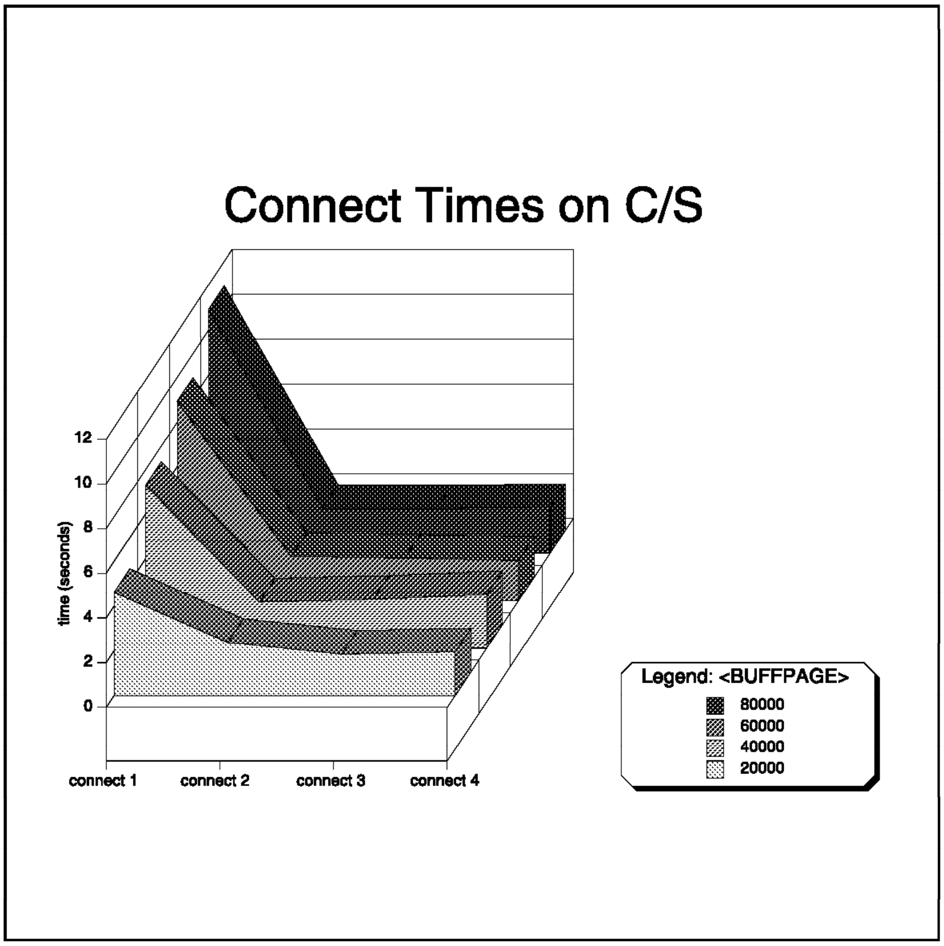


Figure 62. Connect Times on a Client/Server, <AUTHENTICATION> = CLIENT

Figure 63. Connect Times on a Client/Server, <AUTHENTICATION> = SERVER				
<BUFFPAGE>	**Connect #1**	**Connect #2**	**Connect #3**	**Connect #4**
20000	6.65	4.91	4.70	4.24
40000	9.65	4.83	4.38	4.98
60000	10.68	4.40	4.43	4.96
80000	13.23	4.10	4.02	5.97

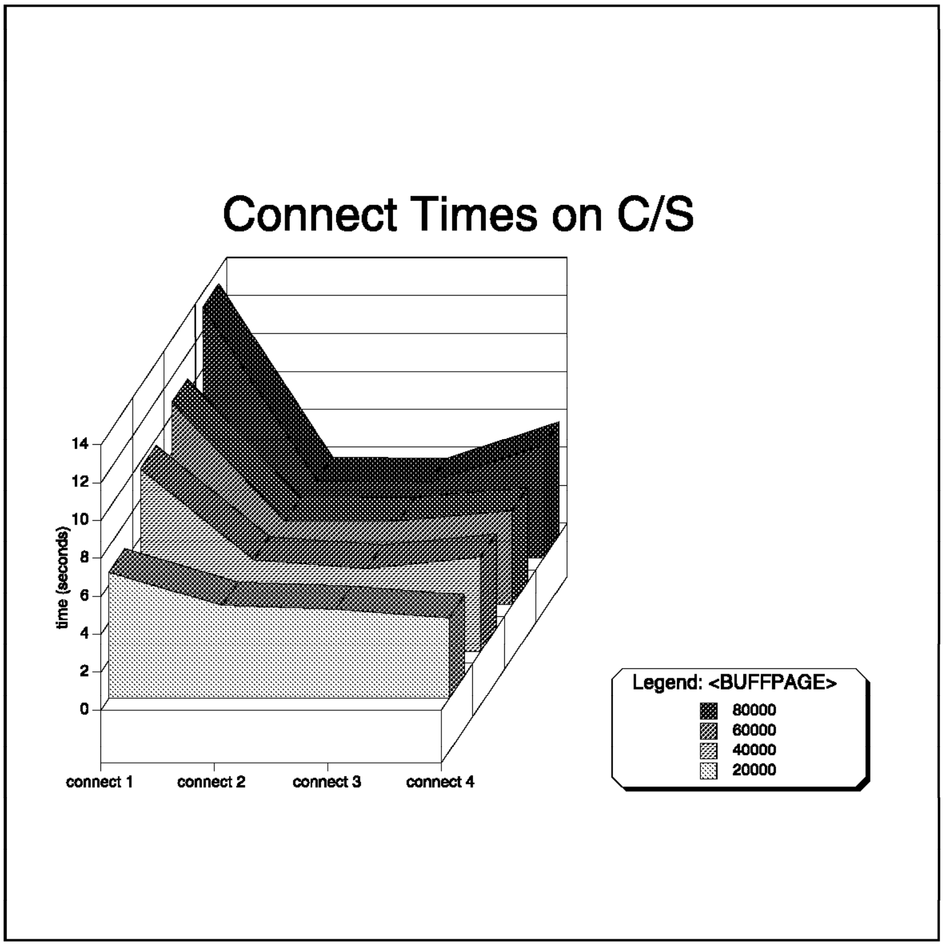


Figure 64. Connect Times on a Client/Server, <AUTHENTICATION> = SERVER

In these results for Connect time on Client/Server with <AUTHENTICATION> = SERVER, the numbers are quite inconsistent. The numbers we collected varied by over 1 second, primarily due to network activity (for these measurements only, the network was non-dedicated).

To compare with DB2 V1.2 we get these results:

Figure 65. Connect Times on a Local Server for Version 1.2				
<BUFFPAGE>	**Connect #1**	**Connect #2**	**Connect #3**	**Connect #4**
10000	1.52	0.15	0.15	0.18
20000	2.58	0.15	0.14	0.15
30000	3.62	0.15	0.15	0.18
40000	4.69	0.15	0.17	0.15

Notice that <BUFFPAGE> values are different in this test. Roughly, the first-time CONNECT was somewhat faster in V1.2 (in part because prefetcher and page cleaner processes were not created, as they are in V2.1), but subsequent CONNECTs tend to be faster on V2. However, first-time CONNECT is not significant because it can be bypassed with a dummy connection, or with the ACTIVATE DATABASE command which will be available in V2.1.1.

5.2.4 Performance Tips

- The most important thing to remember about CONNECT performance is: you will need a dummy connect. If the number of applications is small and they connect and disconnect on an intermittent basis, then all the applications may be disconnected at the same time. Therefore you need a permanent dummy connection to the database to avoid the "first connect" delays.
- For a Client/Server network configuration with the authentication at the server, the CONNECT time is highly dependent on network traffic. When the authentication is done at the client, the performance is improved.
- Some methods to improve connect time include:
  - Reduce the number of processes which are spawned each time an application connects. (Reduce the number of I/O Cleaners or I/O servers, for example. This may cause transaction or query performance to suffer, however!)
  - Reduce the buffers which are allocated each time a new agent is created. For example, reduce the <AGENT_STACK_SIZE> if all your queries are simple.
  - Use the <MAX_IDLEAGENTS> parameter to ensure that idle agents exist, which can be re-used when connections occur.

5.3 Data Management

The option of using databased managed storage (DMS) as opposed to just system managed storage (SMS) is a main feature of DB2 V2.1.0. In addition, the ability to span data over multiple media devices through tablespaces allows for better organization of data.

Some key advantages in using DMS over SMS include:

- Reduction of expensive operating system calls

  With the database manager handling things such as the allocation of data pages, the operating system need not be invoked to do it. The resulting reduction in instruction pathlength means faster execution times and lower I/O wait periods. This is particularly beneficial for OLTP-type workloads.

- Pre-allocation of pages

  This eliminates the need to extend the tablespace one page at a time, as in the case of SMS. The operating system does not need to search for free pages throughout the file system, with the one block allocated at the start. The importance of this was seen in IMPORT (see 4.1, "Import" on page 58).

- Extensibility of tablespaces

  With DMS tablespaces more containers can be added should the size of the database increase. In the SMS case, you can extend the tablespace size by increasing the filesystem size.

- Separation of data and indexes

  When creating a table in a DMS tablespace, the indexes for the table can be assigned to a separate tablespace, such as one located on faster disks. This support is not available for SMS tablespaces.

From these points DMS seems to be the most appropriate for larger databases which have the potential to grow. If the database is small and the largest size for a particular tablespace is known beforehand, SMS may prove easier to use. The reader can assume that DMS is the more strategic approach from a DB2 futures point of view, and if in doubt as to which approach to use, DMS should be preferred.

5.3.1 Performance Tips

Regardless of whether DMS or SMS is used for tablespaces, it is imperative that DB2 has as much information as possible regarding how the data is to be laid out on the available devices. For example, from DB2′s perspective one SMS filesystem / container that spans four disks is seen as only one device. There will be no parallelism in reading from the database and the striping of the data on the four disks will be left to the operating system. This may result in a less-than-optimal configuration for performance. It is much more beneficial to create filesystems on each of the four devices and define these as available containers. DB2 now sees 4 devices and will stripe the data across the containers in extent sizes defined by the user.

It is important to realize that many things that are left to the operating system to deal with, when using Version 1 or SMS, are handled by the database manager when using DMS; e.g., memory management, data allocation. The operating system uses its own set of buffers, caches, and other performance enhancing mechanisms. Since DB2 may not use all of these features in the DMS case, extra tuning may be required to make sure that the database manager has sufficient resources to deal with those things usually handled by the operating system. This is particularly true of <BUFFPAGE>.

I/O wait can be a problem for several database operations. For read operations, DMS seems to perform best, such as in the case of BACKUP (see 4.7, "Backup/Restore" on page 75). The cheaper I/O (i.e., fewer system calls, kernel instructions) and the availability of I/O prefetching makes a significant improvement in read performance for DMS.

Temporary tablespaces are heavily used in operations like sorts. Experiments indicate that it is beneficial to keep temp spaces as SMS, because devices can only be opened in synchronous mode (there is no file system to cache requests). As a result, there is little benefit of using DMS devices for temp space. Writes and reads can actually be more expensive than SMS.

5.4 Configuration Parameters

A large number of configuration parameters have been added in DB2 V2.1.0. To help guide users regarding the parameters on which to focus performance tuning efforts, the *Administration Guide for DB2 Common Servers* classifies each parameter by its importance to performance. This section builds on that classification to discuss the typically most important parameters to be considered for several common database scenarios. Note that within each of these scenarios, specific customer situations will affect which parameters are key.

It should be noted that relatively few configuration parameters can affect performance by more than a few percent. If performance is far short of the required level, effort is best spent in areas such as ensuring index definitions are optimal, making sure RUNSTATS was performed, and adjusting parameters specific to the function (such as SORT BUFFER for LOAD).

For more details on the configuration parameters, see *Administration Guide for DB2 Common Servers*.

5.4.1 OLTP

BUFFPAGE is by far the most significant parameter in an OLTP environment, where there is typically repetitive access to index pages in particular, and often to data pages. This means that a large buffer pool can reduce I/O, improving response times and throughput. A rule of thumb is to strive to have a buffer pool large enough for at least all index pages to remain in memory. A larger buffer pool than that will help, but diminishingly so.

Other key OLTP parameters are MINCOMMIT, NUM_IOCLEANERS, PCKCACHESZ, and AGENTPRI.

5.4.2 Complex Queries

In a complex query environment the key goals with respect to configuration parameters are to minimize the cost of I/O and sorting. To reduce I/O cost, the following parameters are significant:

- BUFFPAGE. The same considerations apply as described for OLTP; however, in a complex query environment it is more likely that significant amounts of I/O are unavoidable and that therefore the buffer pool size is less significant.
- NUM_IOSERVERS. To maximize prefetching, ensure that there is at least one I/O server per database disk. If some disks have multiple containers, there may be a small benefit in using the number of containers in the database.
- SEQDETECT. Leave this set at the default to ensure that prefetching will be done when deemed appropriate.
- NUM_FREQVALUES, NUM_QUANTILES. The more non-uniformity in the user data, the higher the probability that larger values of these parameters (and the use of RUNSTATS ... WITH DISTRIBUTION) will allow the optimizer to find the best possible access plan.

To reduce sort costs, try to have SORTHEAP and SHEAPTHRES as large as possible, taking into account the memory available on the system and the number of concurrent users.

5.4.3 Simple Queries

The key parameter in a simple, read-only query environment is BUFFPAGE.

One of the few other parameters worth mentioning is ASLHEAPSZ: When queries are run locally, the ASLHEAP will be filled (per the ASLHEAPSZ parameter) before the first row is returned to the user application. If answer sets are relatively large, the first response can be unexpectedly slow; one solution is to lower ASLHEAPSZ, another is to use OPTIMIZE FOR n ROWS on the query, where n is some small number.

5.4.4 Large Numbers of Users

When many users will be sharing a DB2 instance and/or database, ensure that MAXAGENTS, MAXCAGENTS, MAXAPPLS, SHEAPTHRES, and LOCKLIST are not too small and impose an unintended limit on DB2′s ability to exploit the system. On the other hand, if the system is being shared with non-DB2 activity (not the preferred approach) these parameters can be useful in limiting DB2′s share of the system resources.

To provide the optimizer with an accurate view of the degree of sharing of the buffer pool, set AVG_APPLS to the average number of concurrent applications (users) before binding static applications.

5.4.5 Heavy Loading of Data

Configuration parameters have different roles, depending on how the data is being loaded:

- Loading via Insert or Import. Both of these methods use the buffer pool.
  - If indexes are not defined, only NUM_IOCLEANERS and CHNGPGS_THRESH are very significant.
  - If indexes are defined: As rows are inserted, index pages are updated to match, by reading them into the buffer pool and writing them to disk later. This reading and writing is greatly affected by BUFFPAGE, NUM_IOCLEANERS, and CHNGPGS_THRESH. A large BUFFPAGE is especially important when keys are being inserted in random order through the entire range of key values, because each insertion could require the write and read of a page in the worst case, which will be very slow.
- Loading via LOAD command. This method does not use the buffer pool.
  - With no indexes pre-defined, no configuration parameters are significant, except for UTIL_HEAP_SZ if long fields or LOBs are being loaded. (A large value of DATA BUFFER on the LOAD command is required to exploit the utilities heap, however.)
  - With indexes pre-defined, UTIL_HEAP_SZ is very significant, in that it determines how large a SORT BUFFER can be requested on the LOAD command. It is also important if long fields or LOBs are being loaded, as mentioned above with respect to DATA BUFFER.

5.4.6 REORG

The key parameters here are BUFFPAGE, SORTHEAP, and NUM_IOCLEANERS.

6.0 Performance of DDCS

6.1.1 General Overview

DDCS is a product that allows communication between workstation database clients and host database management systems through the Distributed Relational Database Architecture (DRDA).

There are two major parts of the DDCS performance measurements documented in this report. The first part shows the cases where the application was run on the DDCS workstation and the second part is the cases where the application was run on a remote client and connected to a DDCS "gateway" workstation through 3 different protocols: APPC, NETBIOS and TCP/IP.

Four different static queries, S1, S2, S3 and S4 were executed through a sequence of CONNECT, Query Sn, DISCONNECT, up to 4 times. Only the average response time of each query was evaluated. The performance of CONNECT/DISCONNECT is outside the scope of our performance studies of DDCS. The response time, in this measurement, is the interval between the time the client sends the query request and the time it receives the answer set and a reply to its COMMIT.

Although DB2/MVS V4 demonstrated performance improvement over DB2/MVS V3 in our studies, the performance data documented in this chapter will focus only on the comparison of DDCS V2.3 with DB2 V2.1 clients, vs. the previous release of DDCS with DB2 V1.2 clients. Both sets of measurements were taken against a workload residing on a server running DB2/MVS V4.1. The host server is a 4-way, 9121-742. Host communications access is via a 3745-210 using TIC2 adapters. A 16 Mbps token ring LAN is being used throughout.

6.1.2 Improvements

In DDCS V2.3, the performance of query fetch block processing has been improved by overlapping the application's processing of a fetched block with an asynchronous request to retrieve the next block. During the processing of an application's request to open a cursor, the first block returned is quickly scanned to determine if it is the last block of the query. If it isn't, a request is issued for another block of data. Since this is an asynchronous call, the request returns immediately and the current block is returned to the user. A receive is then performed when the data in the first block is exhausted. At this time, the second block should have been returned, or shortly thereafter. The scanning and asynchronous request is then repeated for another block. By overlapping the application processing with the asynchronous request of the next query block, the response time on queries that return more than 1 block of data is greatly improved.

There are also several other new features which can improve the performance of a client application. The key features include stored procedures support, Compound SQL, directory caching, etc. This chapter demonstrates the query block fetching response time improvements of DDCS V2R3.

6.1.3 Considerations

The various parameters which can affect the performance of DDCS are related to DDCS itself, the application program, the workload types, the number of users, hardware platform, and the network settings. In our performance studies of DDCS V2.3 the focus was on the impact of RQRIOBLK and Request Unit (RU) size on a query's performance in a single user environment. The application response times were evaluated for DDCS V1.2 and V2.3 with various combinations of RU size and RQRIOBLK.

6.1.3.1 Nature of Queries

Four different queries were measured to compare the impact of the row length requested and column processing:

- S1 - Select 1 columns, total 8 bytes/row
- S2 - Select 21 columns, total 157 bytes/row.
- S3 - Select 42 columns, total 314 bytes/row
- S4 - Select 7 columns, total 157 bytes/row

16384 rows were returned for each of the queries.

6.1.3.2 Communication Subsystem RU Size

This is the maximum size for an RU that can be exchanged in a communication SEND or RECEIVE operation. In general, smaller RU sizes increase the CPU overhead and add to the overall network transmission delay time. This occurs because the communication subsystem must break the data messages into smaller messages, and it must send and receive more network packets.

A larger RU size usually requires more storage on the communication subsystem and the network controllers on the network path. The storage parameters in an existing network need to be adjusted to accommodate an application which utilizes large RUs. Certain host communication access processors such as 3745, 3172 and 3174, may be sensitive to RU size. A 3745-210 was utilized in these measurements which may tend to favor smaller RUs over large RUs.

6.1.3.3 RQRIOBLK

This parameter is the maximum size of the query fetch block. For DRDA, it is set by the Application Requester (DDCS) and determines the query fetch block size between the DDCS workstation and the DB2/MVS server. For DB2 V2 client/server processing, it also is used by the client to set the query fetch block size between the client and the DDCS gateway workstations. A larger block size may improve the performance of large requests. The block size does not usually affect the response time for small requests. A larger block size usually requires more memory on the DDCS workstation. This increases the size of the working set and may cause paging on a memory-constrained workstation.

In our studies, the RQRIOBLK between client and DDCS, and between DDCS and the DB2 server, were set to the same value to minimize the variations of the testing scenarios. RQRIOBLK 4K, 8K, 16K and 32K with 4K, 8K, and 16K RU sizes were compared.

6.1.4 Performance Measurement Results

The performance of DDCS V2.3 and DDCS/6000 V1.2 or DDCS/2 V2.2 was compared in our performance studies. The following list provides an explanation of the data presented in the various tables:

- average response time is in seconds
- S2/V1 means query S2 via DDCS pre-V2.3
- S2/V2 means query S2 via DDCS V2.3
- RU means the RUSIZE used for a particular measurement
- RQR means the RQRIOBLK used for a particular measurement

The measurements results are provided in three sections: Local, OS/2 client, and WINDOW client. In each case the level of DB2 client code corresponded to the DDCS level. Both DDCS/2 and DDCS/6000 gateways were evaluated under each section. For the hardware configurations, please see Appendix C, "DDCS Measurement Configuration" on page 105.

6.1.4.1 Client with DDCS --> DB2/MVS Measurements

These measurements were made with the application running on the same workstation on which the DDCS code was installed.

- The local client code was DB2/6000 V1.2 and DDCS/6000 V1.2 for DDCS/6000 V1 measurements.
- The local client code was DB2/6000 V2.1 and DDCS/6000 V2.3 for DDCS/6000 V2.3 measurements.
- The local client code was DB2/2 V1.2 and DDCS/2 V2.2 for DDCS/2 V1 measurements.
- The local client code was DB2/2 V2.1 and DDCS/2 V2.3 for DDCS/2 V2.3 measurements.

Figure 67 shows the comparison of the average response time for DDCS/6000 V2.3 and DDCS/6000 V1 when measurements were done on the different query types with various combinations of RU size and RQRIOBLK.

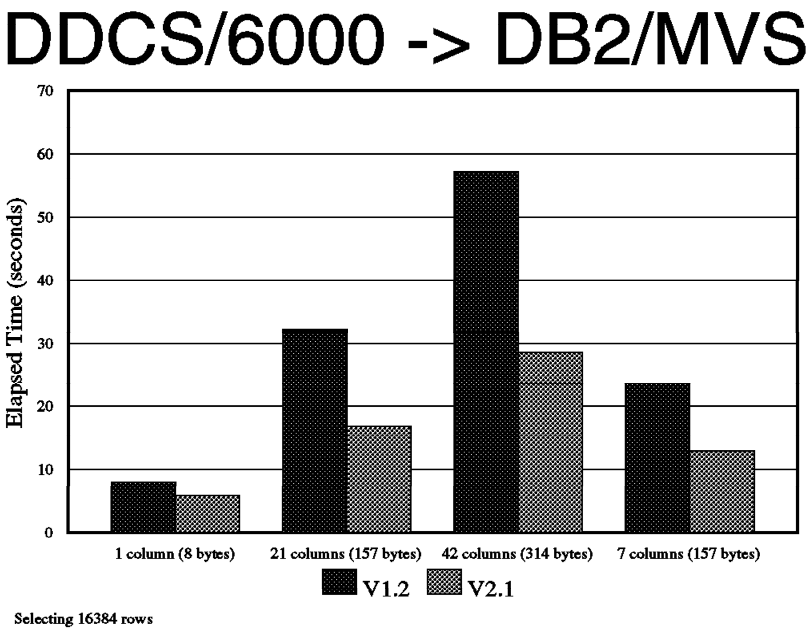


Figure 66. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (Local)

Figure 67. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (Local)

RU	RQR	S1/V1	S1/V2	S2/V1	S2/V2	S3/V1	S3/V2	S4/V1	S4/V2
4k	32k	7.216	5.346	25.462	13.191	44.586	23.084	20.480	12.303
8k	32k	7.197	5.341	25.663	13.075	44.837	23.457	20.863	12.429
16k	32k	7.323	5.298	26.545	13.650	47.142	25.109	21.881	13.401
4k	16k	7.417	5.144	27.233	13.540	48.338	25.508	22.623	13.527
8k	16k	7.426	5.147	28.256	14.399	49.877	27.083	23.143	14.228
16k	16k	7.489	5.118	30.048	16.348	53.903	31.655	25.596	16.419
4k	8k	7.714	4.937	31.491	16.337	55.912	31.644	26.334	16.529
8k	8k	7.850	4.907	32.768	18.098	59.983	34.993	28.115	18.236
4k	4k	8.267	4.930	38.370	22.069	70.913	43.160	34.593	22.431

In the measurement results contained in Figure 67, the following was observed:

- a 26% to 49% improvement in average response time when running with DDCS/6000 V2.3
- the cost of column processing was reduced, thus the response times for the S2 and S4 queries were similar regardless of the number of columns/row (the length of a row returned was the key factor of affecting the response time)
- 8K/8K was the best RU/RQRIOBLK combination for the S1 query type since it returned the smallest amount of data
- 4K/32K was the best choice of RU/RQRIOBLK combination for the other query types

Figure 68 shows the comparison of the average response time for DDCS/2 V2.3 and DDCS/2 V1 when measurements were done on the different query types with various combinations of RU size and RQRIOBLK.

Figure 68. DDCS/2 V1.2 vs. DDCS/2 V2.3 (Local)

RU	RQR	S1/V1	S1/V2	S2/V1	S2/V2	S3/V1	S3/V2	S4/V1	S4/V2
4k	32k	8.055	5.789	32.195	16.875	57.195	28.485	23.625	13.0
8k	32k	8.11	5.914	32.422	17.414	57.75	28.867	23.821	13.133
16k	32k	8.086	5.633	33.07	28.391	59.289	28.391	24.438	13.82
4k	16k	8.25	5.664	34.148	17.789	61.016	30.844	25.695	14.641
8k	16k	8.586	6.133	35.57	17.891	63.531	30.664	26.766	15.586
16k	16k	8.516	5.953	37.063	18.5	67.219	33.438	28.282	17.609
4k	8k	8.547	5.242	37.688	18.797	68.547	35.797	29.469	18.281
8k	8k	9.250	5.453	40.680	20.766	73.711	39.766	32.445	21.414
4k	4k	9.227	5.453	45.305	25.945	84.945	50.414	36.914	26.414

In the measurement results contained in Figure 68, the following was observed:

- a 28% to 48% improvement in average response time when running with DDCS/2 V2.3
- the cost of column processing was reduced, thus the response times for the S2 and S4 queries were similar regardless of the number of columns/row (the length of a row returned was the key factor of affecting the response time)
- 4K/8K was the best RU/RQRIOBLK combination for the S1 query type since it returned the smallest amount of data
- 4K/32K was the best choice of RU/RQRIOBLK combination for the other query types

6.1.4.2 OS/2 Client --> DDCS --> DB2/MVS Measurements

The following measurements were made with the client running on a remote workstation and SQL requests were sent through a DDCS gateway workstation to the DB2/MVS server.

- The OS/2 client code was DB2/2 V1.2 and DDCS/6000 V1.2 for DDCS/6000 V1 measurements.
- The OS/2 client code was DB2/2 V2.1 and DDCS/6000 V2.3 for DDCS/6000 V2.3 measurements.
- The OS/2 client code was DB2/2 V1.2 and DDCS/2 V2.2 for DDCS/2 V1 measurements.
- The OS/2 client code was DB2/2 V2.1 and DDCS/2 V2.3 for DDCS/2 V2.3 measurements.
- Three network protocols, APPC, NETBIOS, and TCP/IP, were used between the OS/2 client and the DDCS gateway.

Unlike the local measurements, both returned data size and the number of columns/row affected the average response time when the application resides on a remote workstation from the DDCS workstation.

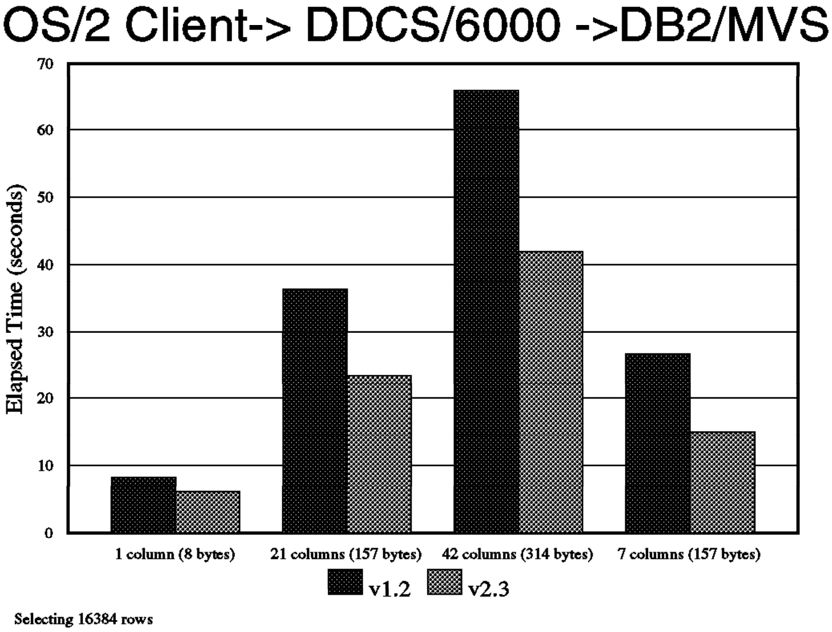


Figure 69. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (OS/2 APPC Client)

Figure 70. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (OS/2 APPC Client)

RU	RQR	S1/V1	S1/V2	S2/V1	S2/V2	S3/V1	S3/V2	S4/V1	S4/V2
4k	32k	8.211	6.041	36.227	23.375	65.844	41.953	26.586	15.016
8k	32k	8.227	6.055	36.531	23.563	66.453	42.032	26.844	14.453
16k	32k	8.281	6.062	37.485	23.610	65.711	42.157	27.883	14.289
4k	16k	8.266	5.851	37.438	24.977	68.50	45.078	27.969	15.508
8k	16k	8.313	5.836	38.188	24.969	70.031	45.156	28.687	15.938
16k	16k	8.469	5.864	40.188	24.844	74.149	44.75	30.914	17.11
4k	8k	8.422	5.898	40.141	27.915	74.07	51.735	30.665	18.407
8k	8k	8.51	5.875	41.938	27.613	77.743	50.946	32.555	19.336
4k	4k	8.761	6.198	45.719	33.555	85.797	64.391	36.664	23.687

In the measurement results contained in Figure 70, the following was observed:

- a 29% to 43% improvement in average response time when running with DDCS/6000 V2.3
- 8K/16K was the best RU/RQRIOBLK combination for the S1 query type
- 4K/32K was the best RU/RQRIOBLK combination for the other query types

Figure 71. DDCS/2 V1 vs. DDCS/2 V2.3 (OS/2 NetBIOS Client)									
RU	**RQR**	**S1/V1**	**S1/V2**	**S2/V1**	**S2/V2**	**S3/V1**	**S3/V2**	**S4/V1**	**S4/V2**
4k	32k	9.149	6.75	43.219	26.563	80.0	46.820	31.273	15.781
8k	32k	9.102	6.742	43.485	26.485	80.813	46.656	31.883	15.711
16k	32k	9.219	6.687	44.586	26.305	82.196	46.781	32.625	15.805
4k	16k	9.398	6.430	44.149	27.422	82.406	49.50	32.312	16.234
8k	16k	9.273	6.414	45.039	27.414	83.734	49.445	33.313	16.523
16k	16k	9.396	6.406	46.937	27.360	87.836	49.539	35.469	17.695
4k	8k	9.367	6.328	46.883	29.344	87.430	54.312	35.336	18.094
8k	8k	9.50	6.323	49.102	29.031	92.477	53.758	37.531	19.461
4k	4k	9.761	6.541	52.649	33.906	99.609	64.508	41.016	23.133

In the measurement results contained in Figure 71 on page 96, the following was observed:

- a 29% to 48% improvement in average response time when running with DDCS/2 V2.3
- 8K/8K was the best RU/RQRIOBLK combination for the S1 query type
- 4K/32K was the best RU/RQRIOBLK combination for the other query types

The data for DDCS/6000 V2.3 is shown for reference only, since there was no TCP/IP client support for DDCS/6000 V1.2.

Figure 72. DDCS/6000 V2.3 (OS/2 TCP/IP Client)					
RU	**RQR**	**S1/V2**	**S2/V2**	**S3/V2**	**S4/V2**
4k	32k	6.0	24.367	45.086	14.742
8k	32k	6.0	23.891	43.407	15.188
16k	32k	6.117	23.859	43.172	15.477
4k	16k	5.992	24.922	45.547	17.016
8k	16k	5.953	26.383	47.430	17.579
16k	16k	5.885	28.539	54.115	18.422
4k	8k	5.937	32.655	69.711	20.570
8k	8k	5.844	31.985	62.641	20.445
4k	4k	5.792	29.086	55.579	22.211

In the measurement results contained in Figure 72, the following was observed:

- 4K/4K was the best RU/RQRIOBLK combination for the S1 query type
- 16K/32K was the best RU/RQRIOBLK combination for the other query types

The data for DDCS/2 V2.3 is shown for reference only, since there was no measurement was done for DDCS/2 V2.2.

Figure 73. DDCS/2 V2.3 (OS/2 TCP/IP Client)					
RU	**RQR**	**S1/V2**	**S2/V2**	**S3/V2**	**S4/V2**
4k	32k	6.609	27.352	48.756	16.055
8k	32k	6.578	27.141	48.469	16.109
16k	32k	6.563	27.164	48.766	16.047
4k	16k	6.344	27.688	50.204	16.352
8k	16k	6.359	27.680	50.242	16.649
16k	16k	6.385	27.719	50.367	17.789
4k	8k	6.430	29.524	54.696	18.289
8k	8k	6.406	29.391	54.438	19.516
4k	4k	6.446	33.188	63.461	22.875

In the measurement results contained in Figure 73 on page 97, the following was observed:

- 4K/16K was the best RU/RQRIOBLK combination for the S1 query type
- 8K/32K was the best RU/RQRIOBLK combination for the S2 and S3 query type

6.1.4.3 Windows** Client --> DDCS --> DB2/MVS Measurements

These measurements were made with the application running on a workstation and the SQL requests were sent through a DDCS gateway to the DB2/MVS server.

- The Windows client code was CAE/DOS V1.2 and DDCS/6000 V1.2 for DDCS/6000 V1 measurements.
- The Windows client code was DB2/WIN V2.1 and DDCS/6000 V2.3 for DDCS/6000 V2.3 measurements.
- The Windows client code was CAE/DOS V1.2 and DDCS/2 V2.2 for DDCS/2 V1 measurements.
- The Windows client code was DB2/WIN V2.1 and DDCS/2 V2.3 for DDCS/2 V2.3 measurements.
- Two network protocols, NETBIOS and TCP/IP, were used between the Windows** client and the DDCS gateway.

Unlike the local measurements, both returned data size and the number of columns per row affected the average response time when the application resides on a remote workstation from the DDCS workstation.

Figure 74. DDCS/2 V1.2 vs. DDCS/2 V2.3 (Windows** NetBIOS Client)									
RU	**RQR**	**S1/V1**	**S1/V2**	**S2/V1**	**S2/V2**	**S3/V1**	**S3/V2**	**S4/V1**	**S4/V2**
4k	32k	16.793	14.528	57.754	59.151	100.87	111.79	38.338	35.550
8k	32k	17.095	14.528	58.385	59.017	101.96	112.71	38.489	35.701
16k	32k	16.844	14.574	58.948	59.017	103.64	111.21	39.642	36.210
4k	16k	15.736	13.773	56.971	54.183	99.373	99.950	39.079	34.232
8k	16k	16.011	13.80	57.685	53.772	101.31	99.387	40.191	32.653
16k	16k	15.910	13.805	59.744	53.291	105.06	98.481	42.100	32.392
4k	8k	15.805	13.168	59.635	56.120	105.84	103.75	42.004	32.543
8k	8k	16.294	13.109	61.955	52.220	110.59	97.135	44.792	31.554
4k	4k	16.551	13.255	68.766	53.410	126.14	100.49	51.213	33.175

In the measurement results contained in Figure 74, the following was observed:

- a 2% to 35% improvement in response time when running with DDCS/2 V2.3
- 8K/8K was the best RU/RQRIOBLK combination for all query types

Figure 75. DDCS/6000 V1.2 vs. DDCS/6000 V2.3 (Windows** TCP/IP Client)									
RU	**RQR**	**S1/V1**	**S1/V2**	**S2/V1**	**S2/V2**	**S3/V1**	**S3/V2**	**S4/V1**	**S4/V2**
4k	32k	14.775	12.138	53.154	65.965	94.512	134.07	36.882	24.222
8k	32k	14.789	12.097	53.374	66.185	94.677	133.84	37.047	24.153
16k	32k	14.793	12.083	53.978	65.947	95.940	134.10	38.077	24.033
4k	16k	14.624	12.303	52.522	66.006	93.236	137.12	37.693	25.636
8k	16k	14.679	12.386	53.085	66.239	94.512	137.29	38.255	25.897
16k	16k	14.775	12.285	55.186	66.514	98.453	136.97	40.397	25.925
4k	8k	15.104	12.152	57.603	132.51	102.37	275.14	42.073	28.012
8k	8k	15.251	12.047	59.141	132.45	105.99	273.86	43.734	28.410
4k	4k	15.525		63.603		116.50		48.980	

In the measurement results contained in Figure 75 on page 98, the following was observed:

- V2 is significantly slower than V1 for large numbers of columns (cases S2 and S3). This is being investigated.
- V2 had the expected advantage for cases S1 and S4.

The data for DDCS/2 V2.3 is shown for reference only, since there was no measurement done for DDCS/2 V2.2.

Figure 76. DDCS/2 V2.3 (Windows** TCP/IP Client)					
RU	**RQR**	**S1/V2**	**S2/V2**	**S3/V2**	**S4/V2**
4k	32k	13.031	72.281	146.62	26.818
8k	32k	12.976	72.419	146.91	26.680
16k	32k	13.127	72.172	147.02	26.611
4k	16k	12.537	72.185	149.79	26.928
8k	16k	12.798	72.186	149.75	27.106
16k	16k	12.651	72.213	149.79	27.243
4k	8k	13.004	143.82	299.08	29.069
8k	8k	13.328	143.82	299.11	29.605
4k	4k				

6.1.5 Performance Tips

In a distributed environment where DDCS acts as the intermediary between the client workstation and the DRDA server, the parameters that affect performance are mainly related to the data transmission block size.

The choice of query fetch block size should match the typical data size that is requested by the application. If large amounts of data are returned, a larger I/O block size can be used because it minimizes fragmentation and network overhead. While large I/O blocks tend to provide better performance, it should not be too large as to sacrifice response time especially in a multi-user environment.

When processing queries, always make sure that fetch blocking is used. There are several ways to do this. Bind the application packages with BLOCKING ALL. If dynamic, add the ″FOR FETCH ONLY″ specification on your SELECT.

Other tips:

- For an OS/2 client application, APPC protocol performs better than NETBIOS and TCP/IP protocol.

- In general, on a busy DB2/MVS server, a smaller RU size (e.g. 4K) in conjunction with a bigger RQRIOBLK (e.g. 32K) defined on DDCS and OS2 client workstation appear to provide better response time.
- A 4K/4K RU/RQRIOBLK combination gave the worst response time performance in all of our test scenarios when APPC or NETBIOS protocol is used.

Appendix A. Benchmark and Database Specifications

Note

As mentioned earlier, see *DB2 Version 2.1.0 Performance Report - Appendices* for details on the skeleton material in the appendices of this document.

A.1 SPEC

A.1.1 SPEC Queries

The syntax of the SPEC queries is shown here and in A.1.6, “SPEC Explain Output.”

A.1.2 SPEC Database: Data Characteristics

The SPEC database consists of seven tables and two views. All the tables have the same record layout and range in size from 10,000 to 100,000 rows.

A.1.3 SPEC Statistics on Database Tables and Indexes

The characteristics of the tables and indexes are shown here as extracted from the catalog tables. The total database size is approximately 30MB.

A.1.4 SPEC DBM and Database Configurations on AIX

The output of “db2 get database manager configuration” and “db2 get database configuration for <DB>” are provided.

A.1.5 SPEC DBM and Database Configurations on OS/2

The output of “db2 get database manager configuration” and “db2 get database configuration for <DB>” are provided.

A.1.6 SPEC Explain Output

The access plans for SPEC were identical on both OS/2 and AIX.

A.2 RQW

A.2.1 RQW Query Descriptions

The syntax of the RQW queries is shown here.

A.2.2 Catalog Statistics for RTW Database

The characteristics of the tables and indexes are shown here as extracted from the catalog tables. The total database size is approximately 130MB (data plus indexes) and consists of 19 tables.

A.3 Internal Benchmark/Database 1 (IB1)

IB1 is a set of 19 queries and the database which is associated with them. The database is 170MB in size, including the data and the indexes. There are 8 tables ranging in size from 5 to 600K rows.

A.4 Internal Database 2 (ID2)

The ID2 database consists of 1 table containing 9 million rows and 16 columns. The total size of the table with no indices created on it is approximately 1.21 GB. In the cases where only 6 million rows were loaded, the size of the database is approximately 800 MB.

A.5 Internal Database 3 (ID3)

The ID3 database consists of 1 table of 8 columns, usually with one million rows. The total size of the table plus indices is approximately 140 MB.

A.6 Customer Benchmark 1

A.6.1 CB1 Descriptions

The query syntax and number of rows returned are provided for each query.

A.6.2 CB1 Table Descriptions

The CB1 database has 19 tables encompassing 1.8GB of data and 150MB of indexes.

A.6.3 CB1 DBM and Database Configurations on AIX

The output of "db2 get database manager configuration" and "db2 get database configuration for <DB>" are provided.

A.7 Customer Benchmark 2

A.7.1 CB2 Query Descriptions

The query syntax and number of rows returned are provided for each query.

A.7.2 CB2 Table Descriptions

Total Data Size = 68.0 MB, Total Indexes Size = 10.1 MB.

A.7.3 CB2 Statistics on Database Tables and Indexes

A.7.4 CB2 DBM and Database Configurations

The output of “db2 get database manager configuration” and “db2 get database configuration for <DB>” are provided.

Appendix B. Hardware Specifications

B.1 Configuration for machine SERVER 1 (AIX)

SERVER 1 is an RS/6000 59H with 1GB of memory, running AIX 3.2.5.

B.2 Configuration for machine SERVER 2 (AIX)

SERVER 2 is an RS/6000 550 with 512MB of memory, running AIX 3.2.5.

B.3 Configuration for machine SERVER 3 (AIX)

SERVER 3 is an RS/6000 550 with 512MB of memory, running AIX 3.2.5.

B.4 Configuration for machine SERVER 4 (OS/2)

SERVER 4 is a PS/2 Model 95 XP (50 MHz 486) with 64MB of memory, running OS/2 V2.11.

B.5 Configuration for machine SERVER 5 (OS/2)

SERVER 5 is a PS/2 Model 95 566 (90 MHz Pentium) with 64MB of memory, running OS/2 V2.11.

B.6 Configuration for CB1 benchmark machine (AIX)

CB1 is an RS/6000 530H with 64MB of memory, running AIX 3.2.5.

B.7 Configuration for CB2 benchmark machine (AIX)

CB2 is an RS/6000 530 with 384MB of memory, running AIX 3.2.5.

Appendix C. DDCS Measurement Configuration

C.1.1.1 OS/2 or Windows Client

- 9595-OPT (60 MHz Pentium PS/2)
- 32MBytes+ of memory (not a factor for single user)
- IBM 16/4 Token Ring Adapter/A (Early Token Release enabled)
- OS/2 WARP V3 or Microsoft Windows V3.1
- OS/2 TCP/IP V2.1 or TCP/IP for DOS V2.1.1
- NetBIOS at MPTS 1.0 level (from LAPS 2.60.2)
- Communications Manager/2 V1.11

C.1.1.2 OS/2 DDCS Gateway (or Local)

- 9595-OPT (60 MHz Pentium PS/2)
- 32MBytes+ of memory (not a factor for single user)
- IBM LANStreamer MC32 Adapter
- OS/2 WARP V3
- OS/2 TCP/IP V2.1
- NetBIOS at MPTS 1.0 level (from LAPS 2.60.2)
- Communications Manager/2 V1.11

C.1.1.3 AIX DDCS Gateway (or Local)

- 7009-C10 (80 MHz PowerPC RISC System/6000)
- 256MBytes of memory
- IBM High Performance Token Ring Adapter
- AIX V3.2.5
- SNA Server/6000 V2.1.1

C.1.1.4 MVS Host

- 9121-742 (air-cooled 4-way CEC)
- 2GBytes of storage (1GBytes central, 1GBytes expanded)
- Channel-attached 3745-210 with TIC2s running NCP V5R4
- MVS V5R2
- MVS/VTAM V4R2
- DB2 for MVS/ESA V4

C.1.1.5 LAN

- Single-segment, 16Mbit Token Ring