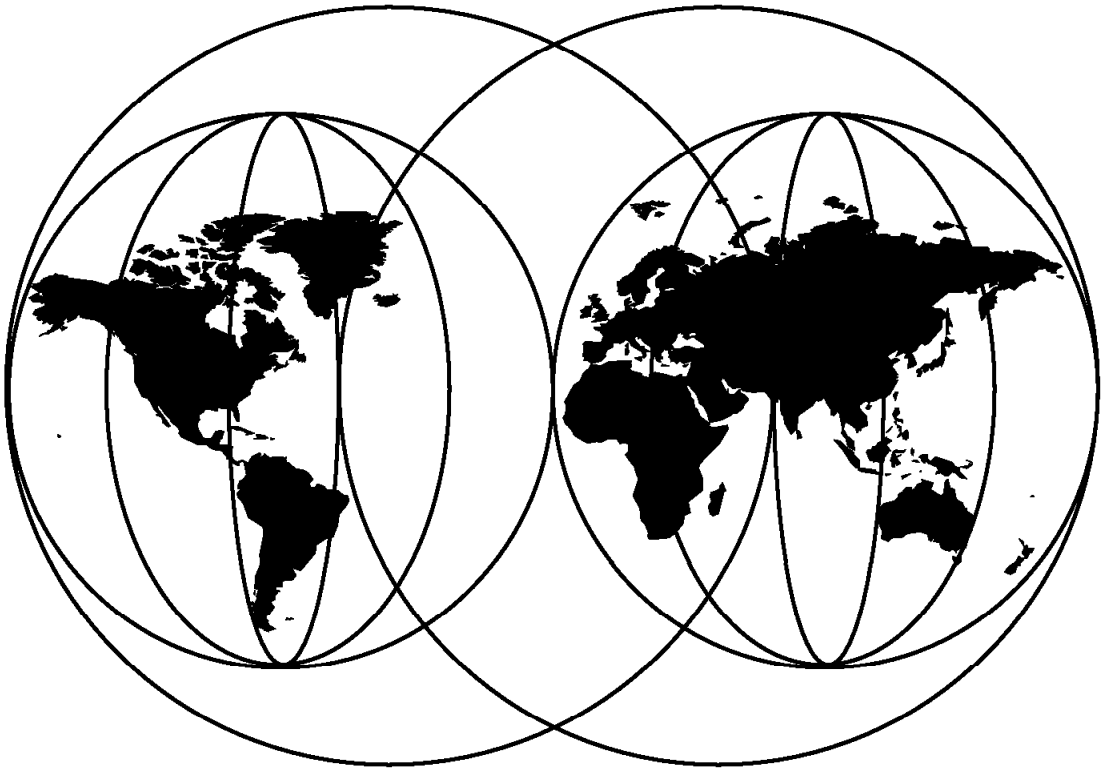
DB2 for OS/390 Terabyte Database: Design and Build Experiences

Dave Clitherow Andrea Harris Luanne McDonough Darren Swank
Dino Tonelli



International Technical Support Organization

http://www.redbooks.ibm.com

This book was printed at 240 dpi (dots per inch). The final production redbook with the RED cover will be printed at 1200 dpi and will provide superior graphics resolution. Please see "How to Get ITSO Redbooks" at the back of this book for ordering instructions.

SG24-2072-01

International Technical Support Organization

SG24-2072-01

DB2 for OS/390 Terabyte Database: Design and Build Experiences

April 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special Notices” on page 35.

Second Edition (April 1998)

This edition applies to Version 5 of IBM DATABASE 2 Server for OS/390, 5655-DB2, for use with the OS/390 or MVS/ESA SP Version 5 Operating Systems.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Preface

This redbook details the 1 terabyte (TB) database build experience of the IBM S/390 Teraplex Integration Center using IBM′s relational database product, DB2 for OS/390 Version 5 in a Parallel Sysplex environment.

The initial release of this redbook (dated May 1997) and this revision do not differ substantially in technical content. Key tables and data from the initial release are carried forward unchanged. This revision clarifies, consolidates, and in a number of instances, reorganizes information originally provided. In addition, we have attempted to more clearly identify areas where our laboratory-oriented VLDB build methodologies and techniques would most likely differ from typical production practices.

This redbook will be of value to users implementing a Very Large Database (VLDB) for business intelligence, decision support or other application purposes.

Hints and tips are provided to assist in planning, setup and build of any similar massive database. These include techniques and methodologies newly developed during this experience. In addition, JCL samples are provided for many of the techniques described.

Detailed knowledge of DB2 for MVS/ESA Version 3 or higher is assumed. Functions which are new to DB2 Version 5 were exploited.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Poughkeepsie Center.

Dave Clitherow is an Advisory International Technical Support Specialist in Large Systems based in Poughkeepsie. He writes extensively and teaches IBM classes worldwide on all areas of Parallel Sysplex implementation and exploitation. Before recently joining the ITSO, Dave worked in the UK as a Senior Services Specialist for IBM Global Services.

Andrea Harris is a Software Engineer at the S/390 Teraplex Integration Center in Poughkeepsie, NY. She has worked extensively in the Decision Support field and is knowledgeable in the areas of function and performance testing and analysis in a DB2 for OS/390 DSS environment. Andrea also has experience building and maintaining very large databases. Before working in the Decision Support field, Andrea was a JES3/MVS Performance Tester for eight years.

Luanne McDonough is a Software Engineer at the S/390 Teraplex Integration Center in Poughkeepsie, NY. She has worked extensively in a DB2 for OS/390 DSS environment. This work includes the building and maintenance of very large databases and function testing.

Darren Swank is a Software Engineer at the S/390 Teraplex Integration Center in Poughkeepsie, NY. He has worked extensively in the Decision Support field. He holds a Masters degree in Information Systems. Darren is knowledgeable in the areas of functional, performance, and stress testing and analysis in a DB2 for OS/390 DSS environment.

Dino Tonelli is a Software Engineer at the S/390 Teraplex Integration Center in Poughkeepsie, NY. He has conducted extensive evaluations of DB2 for OS/390 as a platform for Decision Support. He holds a Masters degree in Computer Science from Polytechnic University. Prior to working in the Decision Support arena, Dino worked on performance evaluations of XCF, GRS, JES2 and TSO.

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

Steve Carbaugh
IBM Poughkeepsie S/390 Teraplex Center, Project Manager

Ralph McCliment
IBM Poughkeepsie S/390 Development Laboratory, Performance Analyst

Chris Panetta
IBM Poughkeepsie S/390 Development Laboratory, Performance Design

Jim Ruddy
IBM Santa Teresa Laboratories, DB2 Utilities Development

Akira Shibamiya
IBM Santa Teresa Laboratories, DB2 Performance

Bryan Smith
IBM Santa Teresa Laboratories, DB2 Query Parallelism Development Team Leader

Michael Tebolt
IBM Poughkeepsie S/390 Development Laboratory, Systems Specialist

Kathryn Zeidenstein
IBM Santa Teresa Laboratories, DB2 Information Development

Thanks also to the entire DB2 Query Parallelism Team.

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 45 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com/`
  For IBM Intranet users `http://w3.itso.ibm.com/`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. Introduction

This chapter covers document objectives, introduces the S/390 Teraplex Integration Center, and presents an overview of our 1TB build phases.

1.1 Objectives

This redbook summarizes the 1 terabyte (TB) database build experience of the S/390 Teraplex team. At the time of writing, this database contained the largest single DB2 table (782GB) and largest nonpartitioning index (238GB) in the world.

The quick, efficient design and build of very large databases entails detailed planning and systematic execution. This redbook details the tasks and considerations involved in this effort and the topics covered include:

- Load performance results
- Database build phases
- Data partitioning
- Utility parallelism
  - LOAD
  - RECOVER INDEX
  - RUNSTATS
- Nonpartitioning index design for query parallelism

We guide you, in detail, through the layout and build of our 1TB database. Various considerations and experiences from this endeavor are discussed and reinforced via “Tips from the TPLEX Team.” After reading this redbook, you should have a basic understanding of how we approach building VLDBs on the S/390 platform.

You will note that we frequently point out that we operate from a lab environment perspective. Therefore, certain methodologies and techniques which we employed to achieve various test and performance objectives (such as manual data distribution across our I/O subsystem versus using STOGROUPS and/or SMS, manually varying initiators to control system resource utilization versus using a job scheduler, and so on) do not necessarily reflect most-common-practice in a production environment.

However, the main techniques presented in this redbook hold true for both lab and non-lab environments.

1.2 Scope

The primary focus of this document is database layout and build. The database schema consisted of eight tables of various size. See 2.2, “General Database Information” on page 8 for our 1TB database structure.

Substantive discussions begin with database build phases and end with table and nonpartitioning index loading. For a complete discussion on data warehousing in the S/390 environment refer to *Data Warehousing with DB2 for OS/390 Version 2.5*, SG24-2249.

1.3 IBM S/390 Teraplex Integration Center Mission

The IBM S/390 Teraplex Integration Center is funded to help support IBM S/390 objectives to achieve growth and leadership in the VLDB Business Intelligence arena.

A primary objective is to demonstrate S/390 hardware and software capabilities.

Our 1TB VLDB build exercise focused on scalability, functionality, performance, usability and concurrency in a Parallel Sysplex environment.

Building a 1TB database in the IBM S/390 Teraplex Integration Center provided:

- IBM S/390 hardware and software VLDB support proof-points.
- VLDB build recommendations for customers and IBM technical sales and marketing staff.
- Additional experience and insight into the effective use of parallelism in the VLDB environment
- Groundwork and a VLDB for subsequent Business Intelligence studies.

This redbook highlights the areas of opportunity where the use of parallelism and our build experience can potentially assist others involved in VLDB implementations.

1.4 1TB Database Build Overview

This section describes the key phases of our 1TB database build. Information contained within this section and the chapters which follow provide much of the rationale behind the methodologies and approach we used to construct our VLDB environment.

The key phases were:

1. System design (including hardware and software configuration determination, acquisition and installation)
2. Physical design
3. Execution of the LOAD & RUNSTATS utilities

Although our build was accomplished in a test environment, the key execution phases are, in most cases, representative of what could be expected for a production environment.

1.4.1 System Design

The first step was to design the 1TB database system which included hardware and software configuration determination, acquisition and installation. Our selected hardware configuration included IBM S/390 Enterprise Server G3 technology in a Parallel Sysplex, and IBM RAMAC direct access storage devices (DASD). Key software products included the OS/390 operating system and DB2 for OS/390, the database management system (DBMS).

Key system and performance design factors which we considered and addressed included:

- Projected hardware and database growth

- Required data type designations
- Overall partitioning scheme
- Nonpartitioning indexes determination
- I/O subsystem design
- Number of work file data sets and placement across control units
- Data and indexes placement across control units:
  - Data spread evenly across controllers
  - Data distributed to minimize workload contention
  - Overall DASD space determinations and allocations

1.4.2 Hardware and Software Configurations

At load time our hardware configuration consisted of:

- Two S/390 G3 RX4 Enterprise Servers
- One S/390 Coupling Facility Model C04
- Four Escon Directors (9032-003)
- Eleven RAMAC II Array Subsystems (2 CUs/rack)
- Six 3990 CUs backed by RAMAC II Array DASD
- Two 3990 CUs backed by RAMAC I Array DASD
- Two RAMAC I Array Subsystems (2 CUs/rack)

Prior testing had shown that for very I/O-intensive workloads, one G3 RX4 (10 CPUs) server was capable of handling the bandwidth supplied by 20 RAMAC II control units (CUs). In other words, each single CPU could handle 2 RAMAC II CUs. Most workloads, ours included, are neither totally I/O-intensive nor CPU-intensive. Therefore, we selected 34 CUs (various types/models) in combination with two G3 RX4 (20 total CPUs) servers as a reasonably balanced system configuration for our testing purposes.

For a diagram of the IBM S/390 Teraplex Integration Center hardware configuration, see Figure 1 on page 12.

Significant software configuration components included:

- OS/390 Rel 1
- DB2 for OS/390 V5 (pre-GA level code)
- DFSORT R13
- SmartBatch for OS/390

The principal software components were OS/390 and DB2 for OS/390. DB2 for OS/390 Version 5 has many VLDB related enhancements, including Sysplex Query Parallelism and support for partitioned table spaces larger than 64GB. For a complete list of these enhancements, refer to *DB2 for OS/390 Version 5 Release Guide*, SC26-8965.

Additional software components included DFSORT for sorting, and SmartBatch to improve the largest nonpartitioning index recovery.

1.4.3 Physical Design

1.4.3.1 Partitioning

The database consisted of six tables, each partitioned 250 ways, and two smaller lookup tables. We partitioned the data this many ways solely to stress new function in DB2 V5 supporting up to 254 partitions. While awaiting hardware installation and subsequent to defining our database layout, we set up the data generation and load jobs.

Note: We used a data generation program to create data in flat files. Of course, you are unlikely to "generate" little, if any, of your warehouse data in a similar manner. Most warehouse data typically originates from production systems and databases.

1.4.3.2 Physical placement

To evenly distribute the tables and indexes across the I/O subsystem, we used user-defined VSAM datasets. Lab environments can be quite dynamic. We frequently move tables and indexes at a VSAM data set level to accommodate various tests. In a production environment, a more suitable approach would be to use DB2 STOGROUPS or SMS managed datasets. In either case, for performance reasons, good practice dictates striving for a well distributed load spread over the entire I/O subsystem.

1.4.3.3 Load

Data definition language (DDL) creation is fairly straightforward. The DDL for a 1MB and a 1TB table is similar. The key difference is the LARGE parameter and partitioning key ranges required for any table space greater than 64GB.

We were able to run multiple LOAD jobs in parallel utilizing our servers and I/O subsystem bandwidth. In a future release, DB2 will manage LOAD parallelism internally. For now, however, it is managed by the DBA. The setups to achieve LOAD and RUNSTATS parallelism are very similar.

1.4.3.4 Indexes

Nonpartitioning indexes can be built during LOAD or can be deferred to later. We opted to defer to reduce LOAD process elapsed time.

The three methods of building nonpartitioning indexes are:

Method One

Build all indexes while loading the table.

Method Two

Build the partitioning index while loading the table. The RECOVER INDEX utility is then used to build the nonpartitioning index.

Method Three

Build the partitioning index while loading the table. Then build the nonpartitioning index using a process we call the *parallel recover index method*, which can reduce elapsed time by parallelizing certain steps of the RECOVER INDEX utility.

We found that the parallel recover index method gave very significant elapsed time reductions for the largest nonpartitioning index by parallelizing certain stages in the RECOVER INDEX process. The RECOVER INDEX utility was used

for the remaining indexes. For more details on this, see 4.2, “Parallel Recover Index Method” on page 18. Sample JCL is also provided in A.1, “JCL Used in the Parallel Recover Index Process” on page 25. A similar procedure to the parallel recover index method will be incorporated in DB2 UDB for OS/390 V6.

1.4.4 Execution

We optimized the LOAD by minimally driving the available processing power to 100% busy. Following the layout described in 3.4, “Data Layout” on page 13, all tables were spread evenly across the I/O subsystem. For us, the simplest and most efficient approach was to complete the load of one table before starting the next. Since LOAD and RUNSTATS were done one table at a time, the work was always distributed evenly over the I/O subsystem. In a shared environment, it is not necessary to do one table at a time, unless a fully optimized LOAD is required.

Next, we proceeded to recover the nonpartitioning indexes. We used the parallel recover index method for the largest nonpartitioning index, which was 238GB. The parallel recover index method is not completely CPU-intensive, it leaves CPU resources available for consumption by other work. We concurrently recovered the smaller nonpartitioning indexes using the RECOVER INDEX Utility and the largest nonpartitioning index using the parallel recover index method. RUNSTATS on the nonpartitioning indexes were also performed concurrently.

Chapter 2. Performance Results

The total elapsed time to build our 1TB database, had we run all jobs back-to-back (no intermediate analysis between jobs) would have been approximately 27 hours. This included loading the data, building the primary indexes, and running the RUNSTATS utility. The database was then ready for query execution.

The following sections break the performance down into greater detail.

- 2.1, “People Effort” describes the personnel resources required to build the database. The total people resources and skills required to build a DB2 database on the S/390 platform are similar regardless of database size.
- 2.2, “General Database Information” on page 8 gives the structure of the tables we used. As expected, various table-related parameters (that is row count, row size, partition count, data distribution within partitions, and so on) affected the performance results achieved.
- 2.3, “Table Load Information” on page 8, 2.4, “Table RUNSTATS Information” on page 9, and 2.5, “Nonpartitioning Index Build Information” on page 9 give more detail on the performance of LOAD, RUNSTATS, and recovering nonpartitioning indexes.

2.1 People Effort

Performance measurements for building our VLDB logically include people as well as machine resources expended. From experience, we have determined that the same skills and approximately the same total effort is required to build a 100GB, 300GB or a 1TB DB2 database on the S/390 platform. The only notable difference is that larger databases are generally spread over a larger I/O subsystem.

Not including upfront database design activities, our build consisted of database design implementation/physical placement, setting up and actually executing the load, running RUNSTATS and building six nonpartitioning indexes. These activities took one person a total of six days to complete. Using DB2 STOGROUPS would likely have reduced this time somewhat.

Database layout consisted of spreading all eight tables, partitioning indexes, nonpartitioning indexes, and work file data sets across 34 CUs. The layout minimized contention between data and indexes, according to the design specifications. This activities subset took four of the six days to complete. If DB2 STOGROUPS were used, less time would have been required.

The remaining two days were spent setting up the build and ensuring execution readiness, which included a code walk-through for correctness.

During build execution, one person occasionally monitored the jobs to check return codes.

2.2 General Database Information

Our database consisted of eight tables. The total raw data, not including partitioning or nonpartitioning indexes, work files and free space, was 1TB. The largest table was 782GB. Due to the 2.8GB volume capacity, each partition of this table necessarily spanned two volumes. There were two very small lookup tables used in joins with the larger tables. Data was evenly distributed across partitions and across the 34 control units. There were eight partitions of each table per control unit. Table 1 shows the sizes of the tables in our 1TB database.

Table 1. General Table Information

Table	Parts	Total GB	Pages/part	Rows/part	Row size
Table A	250	782.49	823,678	24,000,000	141
Table B	250	169.49	178,415	6,000,000	146
Table C	250	113.81	119,804	3,200,000	225
Table D	250	26.05	27,426	800,000	174
Table E	250	25.06	26,385	600,000	233
Table F	250	1.43	1,515	40,000	205
Table G	25	N/A	1	1	195
Table H	5	N/A	1	1	191

2.3 Table Load Information

The information in Table 2 includes the loading of the raw data and primary indexes. We loaded the raw data from flat files on DASD. From the individual table load times, we calculate that running the table LOADs back-to-back and keeping the servers 100% busy would have resulted in a total elapsed time of 17.6 hours. Each individual table LOAD was run in parallel with up to 30 LOAD jobs running concurrently at any one point in time.

In actuality, each completed table load was not followed immediately by the next. After each load, we reviewed output to look for ways to improve our process before proceeding further.

Table 2. Load Information

Table	ET/part	CP time/part	rows/sec	MB/sec	Total ET
Table A	70.66 min	56.92 min	119,178	16	839.13 min
Table B	11.40 min	8.41 min	194,932	22	128.25 min
Table C	5.39 min	3.68 min	247,463	35	53.88 min
Table D	1.79 min	1.09 min	183,251	24	18.10 min
Table E	1.23 min	.75 min	200,803	34	12.45 min
Table F	.40 min	.05 min	102,249	15	1.63 min

2.4 Table RUNSTATS Information

The information in Table 3 includes RUNSTATS results for data and primary indexes. From the individual RUNSTATS times, we calculate that running RUNSTATS back-to-back and keeping the servers 100% busy would have resulted in a total elapsed time of 9.2 hours. RUNSTATS against each individual table was run in parallel with up to 40 RUNSTATS jobs running concurrently at any one point in time.

As was the case with our table LOADS, each completed RUNSTATS was not followed immediately by the next. After each completed RUNSTATS, we reviewed output to look for ways to improve our process before proceeding further.

Table 3. RUNSTATS information

Table	ET/part	CP time/part	rows/sec	MB/sec	Total ET
Table A	43.40 min	22.27 min	245,786	32	406.88 min
Table B	10.81 min	4.38 min	272,064	31	91.89 min
Table C	4.79 min	1.75 min	436,586	62	30.54 min
Table D	1.68 min	.69 min	259,808	34	12.83 min
Table E	1.18 min	.48 min	279,330	47	8.95 min
Table F	.11 min	.04 min	213,675	31	.78 min

2.5 Nonpartitioning Index Build Information

Nonpartitioning indexes, though not required, were developed to improve performance of specific queries which we subsequently ran against our completed 1TB database. We defined six nonpartitioning indexes, totaling 370GB. The largest nonpartitioning index was 238GB.

In order to achieve parallelism when recovering a large nonpartitioning index, we followed the ″parallel recover index method″. This method significantly reduced the total elapsed time of recovering large nonpartitioning indexes from a partitioned table. See 4.2, “Parallel Recover Index Method” on page 18 for implementation information. The total elapsed time to build the 238GB nonpartitioning index on Table A using the parallel recover index method was 40 hours, versus an estimated 130 hours had we used the sequential RECOVER INDEX utility. This was a substantial savings in elapsed time.

After completing the total recover process, RUNSTATS was run for the nonpartitioning index. RUNSTATS for this large index took 36 hours. The other five nonpartitioning indexes were run concurrently with, and were totally contained within, the time to recover and RUNSTAT the largest nonpartitioning index. The total elapsed time for all six nonpartitioning indexes was 76 hours.

Note: Nonpartitioning indexes are not required to achieve query parallelism. The number and size of nonpartitioning indexes varies with different workload and application requirements.

Chapter 3. Database Layout Considerations

The platform configuration consisted of the hardware and software used during the build. Certain software announced, but not generally available at the time, was used. This afforded us the opportunity to test, debug, and refine this product prior to general availability. In certain instances, using pre-released product was a necessity since the function did not exist in earlier versions. For example, without DB2 Version 5, we could not have created our 783GB table. All products are now generally available.

3.1 Hardware

As detailed earlier (see 1.4.2, “Hardware and Software Configurations” on page 3), our hardware primarily consisted of what was then the most current S/390 Enterprise Server technology, a mix of RAMAC DASD types and models, and the requisite connectivity between servers and DASD. Figure 1 on page 12 shows the hardware configuration used.

Each G3 RX4 server had 10 engines (CPUs) and 2GB of central storage, coupled in a Parallel Sysplex. We used two servers to demonstrate building a database using parallelism across a Parallel Sysplex. Proportionately better database build performance could have been achieved had we added additional servers or faster processors to the sysplex.

Note: We used one coupling facility, which was more than adequate from a performance perspective. IBM recommends two coupling facilities for highest availability purposes.

Although the database was 1TB of physical raw data, we used 3 TB of disk space. From customer experience, we typically find total required disk space versus raw data ratios of 1.5 to 2.7. The 3TB space allocation included the data, partitioning indexes, nonpartitioning indexes, and work file data sets. Also included was enough room (1.2TB) to experiment with additional nonpartitioning indexes, work file data sets, and other partitioning schemes. We used 1.1TB of disk space for tables, 340GB for partitioning indexes, and 370GB for nonpartitioning indexes.

Remember, storage requirements will vary, especially in the areas of work file data sets, nonpartitioning indexes, and the amount of experimentation to be undertaken.

Connectivity between servers and DASD was selected and implemented with two objectives in mind: achieving optimal query performance, with the fewest possible ESCON Directors. We used 136 server channels in total for DASD connectivity. For each server, four channels were shared between every pair of control units. The four lower paths of each control unit were distributed across the ESCON Directors.

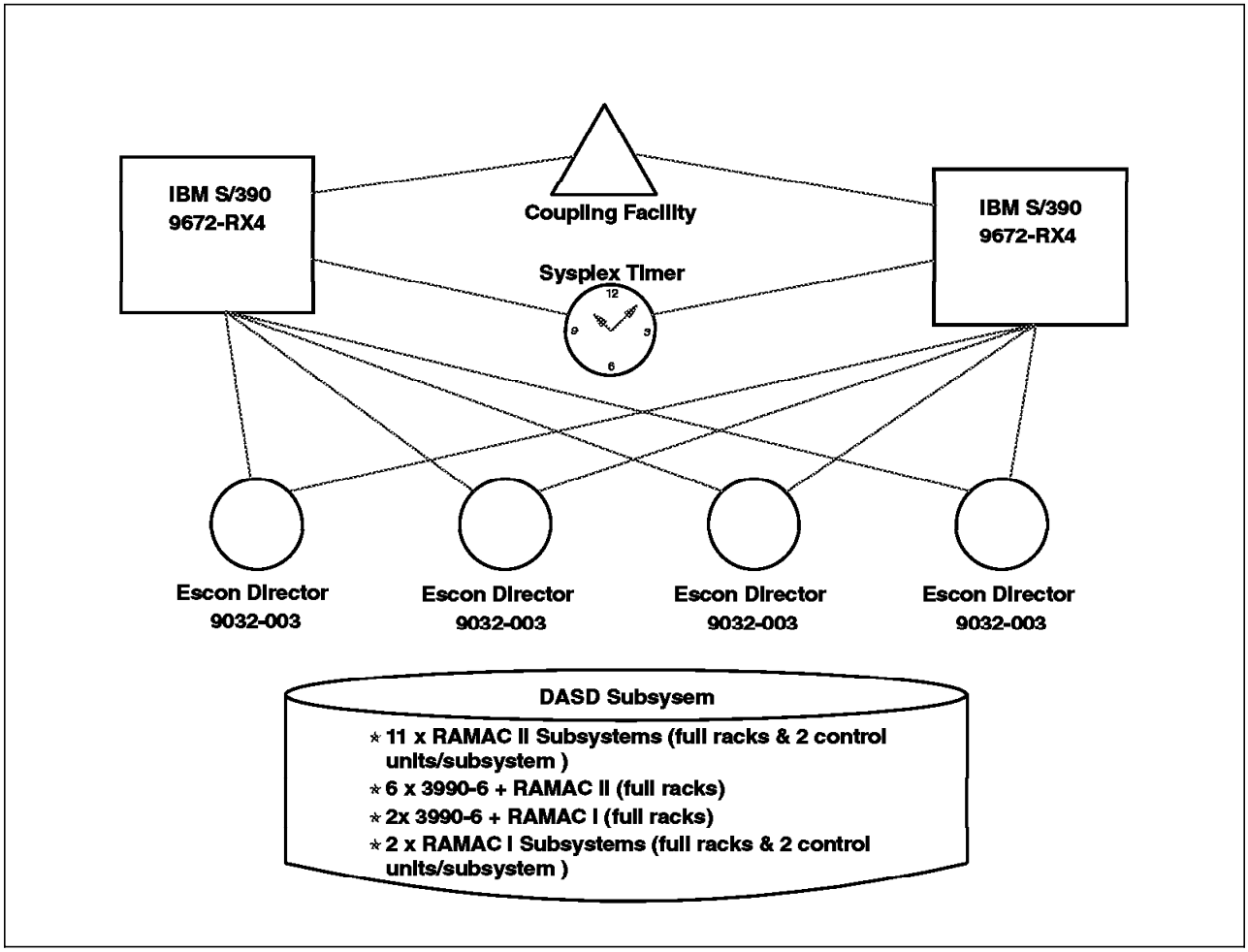


Figure 1. Schematic Showing Hardware Configuration

3.2 Software

As previously detailed (see 1.4.2, “Hardware and Software Configurations” on page 3), the principal software components were OS/390 and DB2 for OS/390.

3.3 Partitioning the Data

DB2 Version 5 now supports single DB2 tables up to 1TB and allows partitioning of any table up to 254 ways. Our 782GB table was partitioned 250 ways. Our experience shows it is best to use a partitioning scheme that considers query performance and overall system management requirements including utilities and maintenance. Initial load performance should not be a primary consideration.

If significant system growth is projected in the areas of CPU requirement, amount of data, or number of I/O control units, consider partitioning the tables to accommodate the growth:

The DB2 optimizer determines the appropriate degree of parallelism and logically partitions the data for each query. There is no performance benefit by partitioning greater than is necessary for your configuration.

For LOAD optimization:

- Consider splitting and sorting your input data according to your partitioning key
- Ensure the resulting number of input data sets is evenly divisible by the number of partitions

To keep our partitioning scheme consistent and simple, we partitioned our six large tables in the same manner. If tables are joined on the partitioning key, they should be partitioned on the same value or a subset of that value. This technique reduces the total I/Os needed to process joins.

The remaining two tables were small lookup tables used in joins with the larger tables. We fully partitioned these tables to provide maximum parallelization when joining with the larger tables. The 5 and 25 row tables were partitioned 5 and 25 ways respectively.

3.4 Data Layout

This section discusses the layout of the data across our I/O subsystem.

3.4.1 Tables and Indexes

Every partition for each of the eight tables had an associated index partition. All partitions were appropriately placed across the 34 control units to minimize contention and optimize performance.

We broke our scheme into a 4-address layout. Every four addresses in a control unit contained one partition of each of the six larger tables. We then repeated the layout over all the devices and partitions. The 4-address layout was key. It broke up the 32-device control unit into eight even pieces. Each piece contained all the data from one partition, along with the nonpartitioning index pieces and work file data sets (discussed later in this section). This method ensured no more than eight parts of one table resided on a control unit. See Figure 2 on page 14 for a diagram of the 4-address layout concept. The two smaller tables were insignificant in size and were configured in afterwards.

Although most of our control units had 32 addresses, some only had 16 and some addresses were shared with system volumes. Our objective was to spread the tables and indexes evenly across the I/O subsystem. We also staggered some of the partitions to reduce the I/O contention. For simplicity, we staggered the partitions by 10. For example, we placed Partition 1 for some tables and Partition 11 for others on the same address.

Note: Remember, in our lab environment we used physical placement. This process could be simplified by grouping multiple volume addresses by control units into STOGROUPS and letting DB2 manage placement. See 3.4.3, “Alternate Data Placement Strategy” on page 14 for more details.

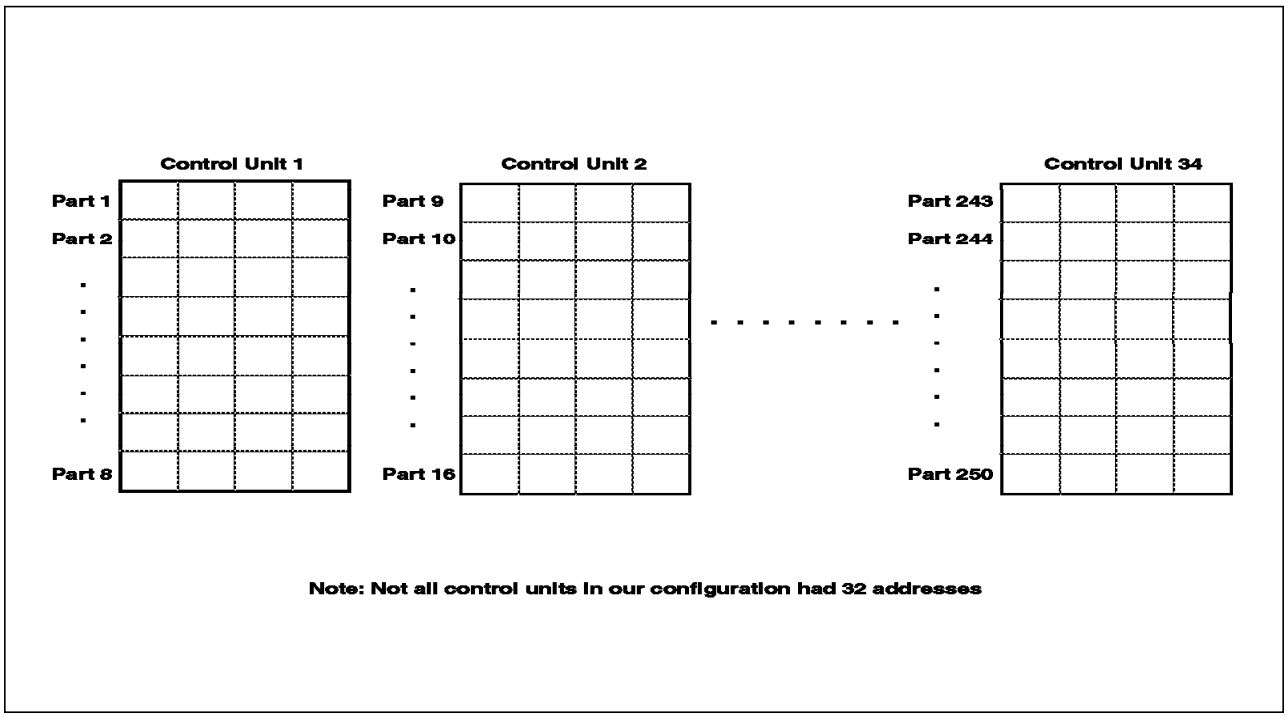


Figure 2. 4-Address Layout of DASD Volumes

3.4.2 Work File Data Sets

The number and size of work file data sets is workload-dependent. The number can be roughly estimated by taking the number of expected concurrent parallel tasks in the system at any one given time and dividing by five. For example, for two large concurrent sorts with a database partitioned 250 ways, start with 100 work file data sets:

```
250 parts * 2 sorts / 5 = 100
```

The larger the work file data sets (up to a full volume), the better. We placed one work file data set in each of the 4-address layouts to evenly distribute the sorts. This was only a starting point which we were prepared to vary as the system was tuned.

3.4.3 Alternate Data Placement Strategy

Historically, many DB2 installations only used partitioned tablespaces to overcome size constraints and to simplify Data Warehouse maintenance. The advent of parallelism presents another reason for partitioning: query performance. For a single application, the approach to implementing the partitions can follow the methods we used — determining where data should be placed, and placing it there. However, in an environment with many applications, a more generic approach simplifies administration, while impacting performance minimally, if at all.

The objectives of DASD management for performance are to:

1. Ensure related partitions are placed on different volumes to minimize physical access delays. Related partitions are those used in concert with each other, such as part 1 of a table and part 1 of its partitioning index, or part x of two tables that are frequently joined for query processing.

2. Provide contiguous data storage for data within a partition to minimize physical access delays.

In an environment where DASD volumes are a shared resource, that is, not "owned" by an application, DB2 and/or SMS storage groups may be used for managing partition placement, as illustrated in Figure 3. Each storage group may have one or more volumes associated with it. As more data is added to the environment, the size of each storage group can be grown accordingly. Tablespace definitions in this environment place different partitions in different storage groups across different control units to accommodate separation needs.

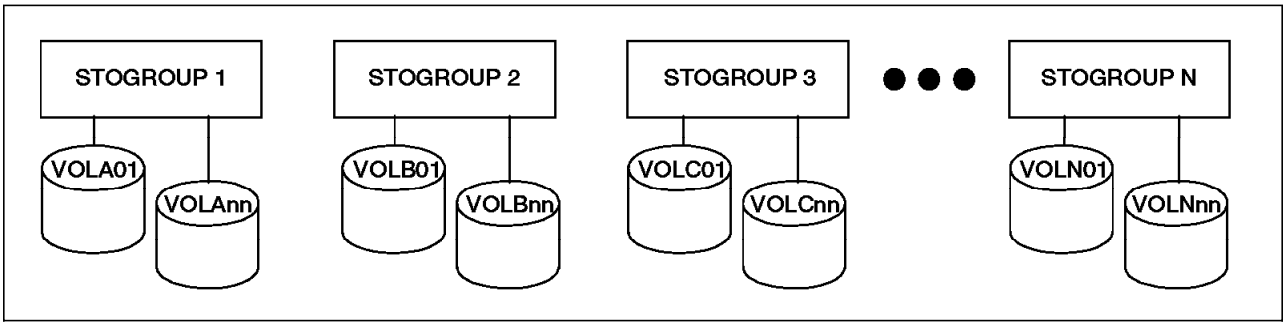


Figure 3. Storage Groups for Parallel Environments

Providing contiguous storage for data is accomplished by specifying appropriate PRIQTY and SECQTY parameters for the partition. If free space on the volumes is contiguous, (SMS Defrags are run periodically), secondary allocations do not entail a significant performance hit.

3.5 Other Considerations

Be aware of database management system boundaries and design your configuration accordingly. For example, in DB2 V5:

- The maximum single table size is 1TB.
- The maximum number of partitions is 254.
- The maximum number of pieces for a nonpartitioning index is 128.
- The size of each nonpartitioning index piece ranges from 256KB to 4GB. See *DB2 for OS/390 Version 5 SQL Reference*, SC26-8966 for a detailed explanation.

Our largest table was smaller than 1TB and was not expected to grow. As data was added to our database, other data was deleted, keeping the total size about the same.

Other areas that warrant investigation are those dictated by the data. For example, certain data type sizes may exceed the maximum allowable value when dealing with terabytes of data. In our database at 1TB of data, the values for some columns defined as INTEGER would have exceeded the highest allowed value. We changed one column in two tables from INTEGER to CHAR 10. This gave us the most flexibility with the least performance impact and least total DASD consumption.

Tips from the TPLEX Team

- Partitioning should consider query performance and overall system management requirements including utilities and maintenance.
- Tables joined on the partitioning key should have the same number of partitions breaking on the same values.
- Small lookup tables joined to larger tables should be fully partitioned.
- Break the layout into manageable pieces (that is, 4-address layout).
- Be aware of data definition boundaries as the database grows.

Chapter 4. Optimizing Database Build

In this chapter we explore the benefits of parallelism in two areas:

- Decreasing build elapsed time
- Optimizing query performance via indexes

The topics we discuss are:

1. LOAD and RUNSTATS parallelism
2. Parallel recovery of nonpartitioning indexes
3. Design of nonpartitioning indexes for query parallelism

The first two topics are directly related to database build. We show how we achieved optimal performance when loading data, when running RUNSTATS on data, and when recovering nonpartitioning indexes.

The third topic is a special case. A method of building nonpartitioning indexes for better query parallelism has been developed. We show how to design large nonpartitioning indexes so that queries that access them encounter minimized contention in a concurrent query environment.

4.1 Load and Runstats Parallelism

To achieve parallelism during raw data load, we set up a LOAD for each partition. This is not new; DB2 has used partition independence for years to let the DBA work with individual partitions while not affecting the rest of the table or database. In Chapter 3, “Database Layout Considerations” on page 11 we discussed spreading the data uniformly across the I/O subsystem. Since each table was spread in this manner, it was easy to achieve optimal parallelism.

First we created the loads. For each table, each step was basically the same; only the step name, table name, and partition number required changing.

One LOAD parameter worth mentioning is SORTKEYS. We used it to reduce total sort work space needed and to speed up LOAD. One advantage of using SORTKEYS is that the index keys are passed to sort in memory, rather than written to sort work files. This avoids the I/O to SYSUT1 and SORTOUT. There was also overlap between loading and sorting, since the sort was running as a separate task. Avoiding I/O to sort work files and the overlapping of tasks improved LOAD performance.

SORTKEYS was calculated for each partition and since the data was distributed evenly, it was the same for each partition. For example, on our largest table we specified SORTKEYS 6010000 for each partition. Refer to *DB2 for OS/390 Version 5 Utility Guide and Reference*, SC26-8967 to calculate the SORTKEYS value.

Once the build was set up, the execution needed to be managed in the system. In our dedicated environment, we released the work into the system at a rate designed to keep all the processors 100% busy while minimizing I/O contention. We submitted one table at a time to distribute the work over the I/O subsystem, and varied the number of initiators to keep the system at 100% busy.

Note: In a non-dedicated environment, a more likely approach would be to use a job scheduler to distribute the resource consumption. OS/390 V2 R4 provides dynamic control of initiators via WLM.

For each table, we spread the LOADs evenly across the I/O subsystem, using the 4-address layout, by staggering the partitions being loaded. We staggered our partitions by 10. For example, we started with 1, 11, 21, 31 and so on. This made it easy to keep track of the partitions and helped prevent loading different partitions on the same CU concurrently. This was only necessary for super tuning the LOAD to minimize I/O contention.

We kept all work classes the same and varied the step name for every partition. This facilitated increasing and decreasing the amount of work in the sysplex by changing the number of available initiators with the specific class. We also used the DB2 generic group attachment name for the DB2 subsystem name in the batch utilities. This allowed the system to direct the work across the sysplex.

For each table of the LOAD we submitted all 250 partitions simultaneously, starting with 20 active initiators across the sysplex. For our system configuration and table attributes a good rule of thumb was 1.5 to 2 LOADs per processor. Of course, these numbers would vary given different table attributes and/or different processors. In our specific case, 15 initiators per server was adequate to drive the sysplex to 100% busy.

RUNSTATS was executed in the same manner as LOAD. In some cases we needed 20 jobs per server to keep each 100% busy. The number of jobs per server varied, depending on table attributes. The only option we used was KEYCARD. For example:

```
RUNSTATS
  INDEX (creator.SXTABLEA KEYCARD)
```

4.2 Parallel Recover Index Method

LOAD and RUNSTATS are straightforward and very familiar to most. You may already have performed LOADs and RUNSTATS using a methodology similar to what we have discussed. This section focuses on a newer method for recovering nonpartitioning indexes using parallelism. Using this new method, we saw a 3X performance improvement as compared to the standard DB2 RECOVER INDEX utility.

In order to achieve parallelism when recovering a large nonpartitioning index, we followed a process we call the parallel recover index method. This process significantly reduced the total elapsed time for recovering large nonpartitioning indexes from a partitioned table.

The RECOVER INDEX utility has three main phases:

1. Unload
2. Sort
3. Build

The parallel recover index method uses the RECOVER INDEX utility, along with an additional merge phase, to achieve parallelism for the unload and sort phases of the RECOVER INDEX utility.

Figure 4 on page 19 represents the parallel recover index method for our largest nonpartitioning index. The table the index was recovered from was Table A, 782GB, partitioned 250 ways. The nonpartitioning index was 238GB.

Note: We used SmartBatch for OS/390 to pipe the data between processes. Although not required, when compared to the parallel recover index process without SmartBatch, this product provided additional elapsed time reduction, as well as intermediate DASD savings. For more information on SmartBatch, refer to *IBM SmartBatch for OS/390 Overview*.

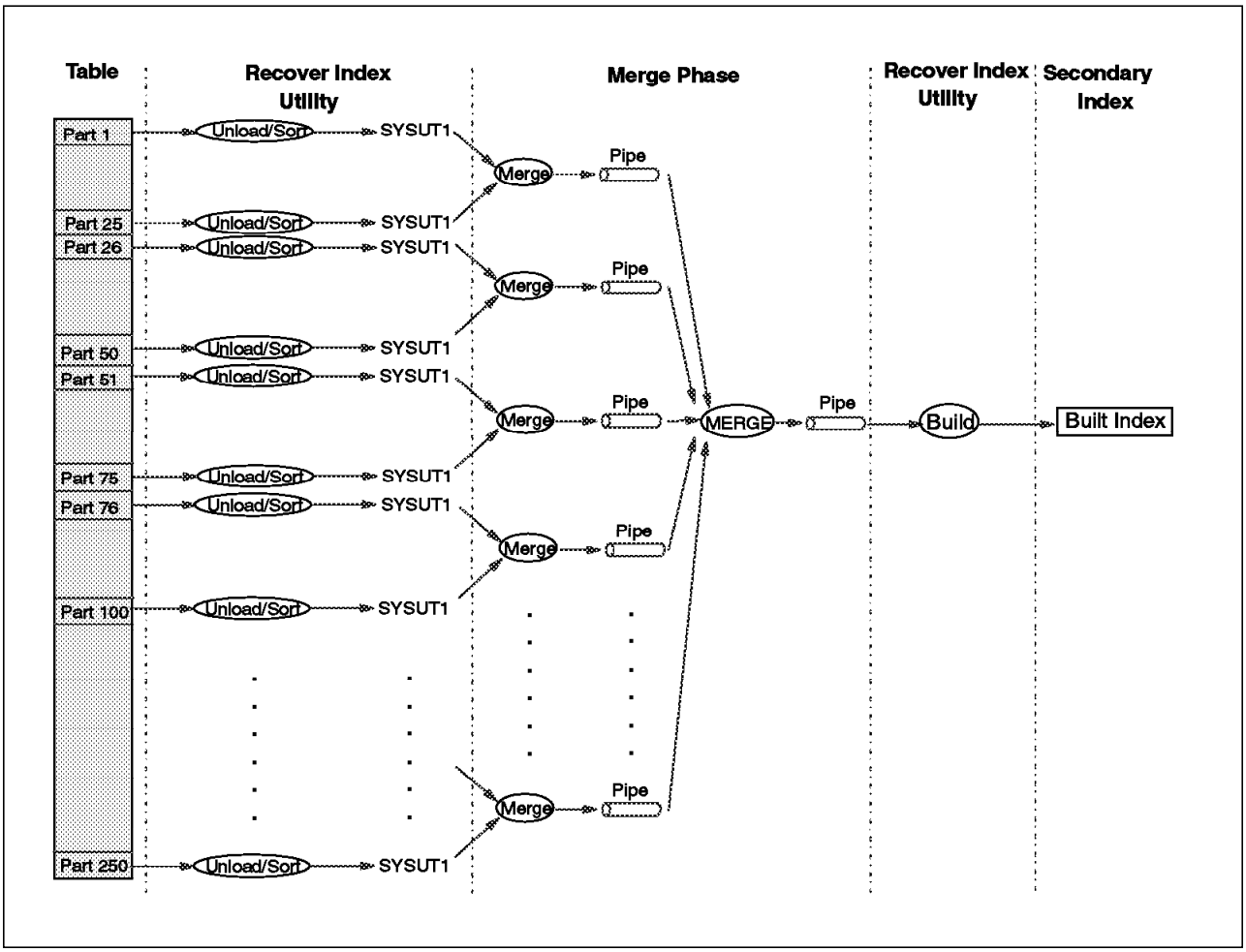


Figure 4. Parallel Recover Index Methodology

The rest of this section gives a high-level description of how we implemented the parallel recover index method.

The RECOVER INDEX utility was started for each partition. Approximately 25 partitions kept one G3 RX4 Server 100% busy. As before, we controlled the number of initiators started to control the amount of work in the system. After completing the Unload and Sort phases for each partition, the utility was stopped. After all RECOVER INDEX utilities had been stopped, there was an individually sorted SYSUT1 data set for each partition. Since very large sort elapse times tend to increase exponentially, significant savings can be achieved by sorting subsets and merging results. We accomplished this by merging data sets via Smartbatch into 10 groups of 25. One pipe from each group was then created that flowed into another merge job that piped the final merged data back into the restarted RECOVER INDEX utility. Only one RECOVER INDEX utility was

restarted at the build phase. The output of the utility was the 238GB nonpartitioning index.

After the total process was complete, RUNSTATS was run for the nonpartitioning index. Since this did not require 100% of the available CPU resources, we were able to execute other work concurrently with RUNSTATS. This consisted of five other concurrently running nonpartitioning indexes which were completed within the largest nonpartitioning index build and RUNSTAT window.

We used two classes of initiators for recovering our nonpartitioning indexes (call them class A and B, for ease of explanation). In the Work Load Manager (WLM) policy, Class A had a higher priority than Class B. We started the parallel recover index method under Class A initiators and the five nonpartitioning indexes recover jobs under class B initiators. After the unload and sort phases of the parallel recover index were finished and the build phase was started, we recovered the rest of the nonpartitioning indexes using DB2′s RECOVER INDEX utility. Using this method, the longest running work (under class A initiators) was given higher priority, but when there were unused CPU cycles, the other indexes (running under class B initiators) were recovered concurrently.

4.2.1 Details on Parallel Recover Index

Following is a more detailed list of the steps involved in the parallel recover index process. This example is for an index being recovered from a 250-partition table using two G3 RX4 Servers. These steps need to be modified slightly if a table is partitioned differently, or if a different number or type of server is used. A detailed description of each step follows the list, starting at 4.2.1.1, “Setup” on page 21.

1. Build MERGE FIELD statements (for DFSORT) for merging SYSUT1s. (SYSUT1s are generated in each parallel RECOVER INDEX job.)

   Note: The MERGE FIELD specifications should be identical to SORT FIELD specifications from the RECOVER INDEX DFSORT invocation.

2. Define VSAM CLUSTERS for “SHADOW” TABLESPACE and primary INDEXSPACE partitions.
3. Create “SHADOW” DATABASE, TABLESPACE, TABLE, and primary INDEXSPACE. Insert 1 row in table.
4. Build OBIDXLAT input to edit “SHADOW” TABLESPACE partitions.
5. DSN1COPY OBIDXLAT into “SHADOW” TABLESPACE for all partitions.
6. Define VSAM CLUSTERS and create INDEX with DEFER YES for the nonpartitioning index being parallelized.
7. RECOVER INDEX for partitions 1-25, ABEND after SORT PHASE (see 4.2.1.2, “Unload and Sort” on page 21 for details).
8. RECOVER INDEX for partitions 26-50, ABEND after SORT PHASE.
9. Repeat step 8 for remaining partitions, in groups of 25.
10. Stop original TABLESPACE partitions 1-250.
11. Rename the real TABLESPACE partitions to temporary names.
12. Rename the “SHADOW” TABLESPACE partitions to real names.
13. Start TABLESPACE partitions 1-250.
14. REPAIR LEVELID TABLESPACE partitions 1-250.

15. RECOVER INDEX on “SHADOW” TABLESPACE; ABEND after SORT PHASE.
16. MERGE SYSUT1s from parts 1-25 (from step 7) into an intermediate merged SYSOUT data set.
17. Repeat step 16 for remaining partitions, in groups of 25.
18. MERGE all 10 intermediate “merges” into one data set (to be used as the input SYSUT1 in the next step).
19. Restart RECOVER INDEX using the final merged SYSUT1 from step 18.
20. Stop TABLESPACE partitions 1-250.
21. Rename “SHADOW” TABLESPACE back to SHADOW.
22. Rename temporary TABLESPACE back to real name.
23. Start TABLESPACE partitions 1-250.
24. REPAIR LEVELID TABLESPACE partitions 1-250.

4.2.1.1 Setup

Steps 1 to 4

First, define the merge fields used to merge the SYSUT1 files. This can be done by recovering the index for one part and using the sort fields in the job’s output. The same merge fields are used for the intermediate MERGE and the final MERGE.

Next, create the VSAM clusters for the “SHADOW” database that is created in the next step. Only one row is inserted into the SHADOW database. The SHADOW database allows building of the index from the final SYSUT1 data set without unloading and re-sorting the entire table. Only the one row in the SHADOW database is unloaded and sorted.

Next the OBIDXLAT information is gathered from the “REAL” table space partitions. This information is used so the SHADOW database and the REAL database look the same to DB2. For an example of the JCL used for Steps 1-4, see A.1.1, “Setup” on page 25.

Step 5

The OBIDXLAT information is copied to every partition of the SHADOW database. Run each DSN1COPY in parallel. We made 25 groups of 10 to run in parallel. For an example of the JCL used for Step 5, see A.1.2, “Copy OBIDXLAT” on page 30.

Step 6

The last setup step is to define the VSAM CLUSTERS and create the index with DEFER YES.

4.2.1.2 Unload and Sort

Steps 7 to 9

This is the portion of the RECOVER INDEX benefitting the most from parallelism. We broke the work (250 partitions) into 10 groups of 25. We found that 25 UNLOAD, SORT and MERGE jobs kept one G3 RX4 Servers busy.

Start 25 RECOVER INDEX utilities, stopping them after the UNLOAD and SORT phases. To do this, add a DIAGNOSE statement to the SYSIN DD card:

```
//SYSIN    DD *
 DIAGNOSE
  ABEND TRACEID X'1180' NODUMP
 RECOVER INDEX
    (creator.SXL@TABLEA PART n)
```

The first 25 jobs are not the first 25 partitions. For best results, the work should be spread across all the controllers. Our first 25 jobs were partitions 1, 11, 21, and so on. For an example of the JCL used for this portion of the RECOVER INDEX utility, see A.1.3, "Recover Index" on page 31. Repeat this process until all 250 partitions have been recovered.

4.2.1.3 Rename SHADOW to REAL Tablespace

Steps 10 to 14

These steps are straightforward. Basically, the SHADOW database takes the place of the real database. The purpose of this is to "skip" the UNLOAD and SORT phase in the next section. This is done by renaming the VSAM CLUSTERS. REPAIR LEVELID fixes any information that is needed by DB2 that was changed during the rename. For an example of the JCL used for REPAIR LEVELID, see A.1.6, "Rename SHADOW to REAL Tablespace" on page 32.

4.2.1.4 Build and Merge Phases

Steps 15 and 19

In these steps, the nonpartitioning index is built. First, start the RECOVER INDEX utility, then stop it after the UNLOAD and SORT phase, as with the other 250 partitions. The difference is that this is against the entire SHADOW database. Since there is only one row in the SHADOW database, this runs very quickly.

Start 25 MERGE jobs to merge the output from the first 25 jobs into an intermediate merged pipe. The MERGE jobs merge all the output from the 25 RECOVER INDEX jobs into one final pipe. For an example of the JCL used for the MERGE, see A.1.4, "Intermediate Merge" on page 31.

After the 25 MERGE jobs complete, or the system becomes under-utilized, start the next group of 25. Repeat this process until all 250 partitions have been merged into 10 SYSUT1 pipes. Next MERGE all 10 intermediate merges into one final merged pipe. For an example of the JCL used for the final MERGE, see A.1.5, "Final Merge" on page 32. The final pipe is used as the input into the Build phase of the RECOVER INDEX utility.

There are two reasons why there is an intermediate MERGE into 10 pipes. First, each MERGE does not scan all 250 data sets. Second, the intermediate MERGEs are done in parallel. Both of these reasons contribute to reduced elapsed time.

Finally, RESTART the RECOVER INDEX utility pointing to the final pipe that was created earlier. For an example of the JCL used for these two steps, see A.1.7, "Final Build" on page 33.

When using Smartbatch, since there are no intermediate data sets, all pipes must be started to receive the output from the previous step as input.

4.2.1.5 Rename Temporary Back to REAL Tablespace

Steps 20 to 24

These steps are also straightforward. Basically the REAL database is put back to its original form renaming VSAM CLUSTERS. Again, REPAIR LEVELID fixes any information that is needed by DB2 that was changed during the rename. For an example of the JCL used for these steps, see A.1.8, “Rename Temporary Back to REAL Database” on page 34.

This concludes the procedure to recover a nonpartitioning index in parallel. A similar procedure will be incorporated in DB2 UDB for OS/390 V6. This will provide the ability to recover a nonpartitioning index using parallelism by simply executing a DB2 utility.

4.3 Design Nonpartitioned Index for Query Parallelism

Physical contention can be reduced on nonpartitioning indexes by breaking them into multiple data sets (pieces). These pieces can be spread across the I/O subsystem for maximum performance. In this manner, multiple accesses to different index pieces can take place simultaneously.

For nonpartitioning indexes, use the PIECESIZE parameter on the CREATE statement to indicate the size of each index data set. Smaller values result in more data sets defined. The default is 4GB for table spaces defined as LARGE and 2GB for all others. Up to 128 data sets (pieces) per index are allowed.

To choose a PIECESIZE value, divide the nonpartitioning index size by the number of pieces desired and round up to closest valid PIECESIZE value. (In some case it may be necessary to round down so the PIECESIZE does not have a lot of wasted space. In this case there will be more pieces of smaller size.) The more pieces spread across the I/O configuration, the better. For example, our largest nonpartitioning index was 237.5GB. To spread it across as many devices as possible, we used a PIECESIZE of 2GB and a total of 120 pieces to spread the index across the I/O subsystem.

```
2GB * 120 pieces = 240GB nonpartitioning index.
```

If the nonpartitioning index is likely to grow, a larger PIECESIZE value should be considered. Keep the PIECESIZE value in mind when choosing values for primary and secondary quantities for the VSAM data sets. Ideally, the value of the primary quantity plus the value of the secondary quantity should be evenly divisible into PIECESIZE. PIECESIZE needs to be converted to cylinders before the quantities can be compared.

Tips from the TPLEX Team

- We found it systematic to work on one table at a time for LOADS and RUNSTATS.
- Use the DB2 generic group attachment name to distribute work across the sysplex.
- Other work can be run concurrently with the Parallel Recover Index.
- Give the longest-running work the higher priority.
- Break nonpartitioning indexes into pieces and spread across the I/O subsystem to reduce contention.

Appendix A. Sample JCL

This appendix provides some examples of the JCL we used when building our 1TB database.

A.1 JCL Used in the Parallel Recover Index Process

This section provides the sample JCL for the parallel recover index process.

A.1.1 Setup

```
//RECINDEX JOB CLASS=A,TIME=NOLIMIT,REGION=5M,MSGCLASS=H,
//           NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//JOBLIB DD DSN=DSN510.SDSNLOAD,DISP=SHR
//       DD DSN=DB2CT510.RUNLIB.LOAD,DISP=SHR
//*********************************************************************
//* SORT FIELDS and RECORD TYPE from a RECOVER INDEX SORT must be used
//* to generate the MERGE FIELDS and RECORD TYPE for MERGE
//*
//* In order to obtain the SORT fields you must run a RECOVER INDEX
//* against 1 partition of the table. Then extract the SORT field
//* information from the output and ensure you DROP the Secondary
//* index. Once you have the SORT field information, just replace
//* the word SORT with MERGE.
//*
//* Example:
//*
//*  SORT FIELDS=(0012,0026,A,0001,0005,A),FORMAT=BI
//*  RECORD TYPE=F,LENGTH=(00000037,00000037,00000037)
//* gives
//*  MERGE FIELDS=(0012,0026,A,0001,0005,A),FORMAT=BI
//*  RECORD TYPE=F,LENGTH=(00000037,00000037,00000037)
//*
//MERGEFLD EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT2   DD DISP=(NEW,CATLG,DELETE),
//            SPACE=(TRK,(1)),UNIT=SYSALLDA,VOL=SER=DSSPK0,
//            DSN=creator.RISXTBLA.MERGFLD
//SYSIN    DD DUMMY
//SYSUT1   DD *
  MERGE FIELDS=(00012.0,00032.0,A,00001.0,00005.0,A),FORMAT=BI
  RECORD TYPE=F,LENGTH=(0043,0043,0043)
//*********************************************************************
//DROP     EXEC PGM=IKJEFT01,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-DIS UTIL(*)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP51)
//SYSIN    DD *
 DROP   DATABASE DBS1TBS;
//*********************************************************************
//*********************************************************************
//DEFINE   EXEC PGM=IDCAMS,TIME=NOLIMIT,COND=(EVEN)
//SYSPRINT  DD SYSOUT=*
//SYSIN  DD *
```

```
DELETE -
   DB2DBS.DSNDBC.DBS1TBS.PXTABLEA.I0001.A001 CLUSTER PURGE

etc.
etc.
etc.

DELETE -
   DB2DBS.DSNDBC.DBS1TBS.PXTABLEA.I0001.A250 CLUSTER PURGE

DELETE -
   DB2DBS.DSNDBC.DBS1TBS.TSTBLA.I0001.A001 CLUSTER PURGE

etc.
etc.
etc.

DELETE -
   DB2DBS.DSNDBC.DBS1TBS.TSTBLA.I0001.A250 CLUSTER PURGE

SET LASTCC = 0
SET MAXCC = 0

DEFINE CLUSTER -
   ( NAME(DB2DBS.DSNDBC.DBS1TBS.PXTABLEA.I0001.A001) -
   LINEAR     SPEED                        -
   SHAREOPTIONS(3 3)                       -
   TRACKS(1 1) VOLUMES(BI2450)         -
   REUSE    )                              -
 DATA                                      -
   ( NAME(DB2DBS.DSNDBD.DBS1TBS.PXTABLEA.I0001.A001))

etc.
etc.
etc.

DEFINE CLUSTER -
   ( NAME(DB2DBS.DSNDBC.DBS1TBS.PXTABLEA.I0001.A250) -
   LINEAR     SPEED                        -
   SHAREOPTIONS(3 3)                       -
   TRACKS(1 1) VOLUMES(BI2450)         -
   REUSE    )                              -
 DATA                                      -
   ( NAME(DB2DBS.DSNDBD.DBS1TBS.PXTABLEA.I0001.A250))


DEFINE CLUSTER -
   ( NAME(DB2DBS.DSNDBC.DBS1TBS.TSTBLA.I0001.A001) -
   LINEAR     SPEED                        -
   SHAREOPTIONS(3 3)                       -
   TRACKS(1 1) VOLUMES(BI2452)         -
   REUSE    )                              -
 DATA                                      -
   ( NAME(DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A001))

etc.
etc.
etc.
```

```
 DEFINE CLUSTER -
    ( NAME(DB2DBS.DSNDBC.DBS1TBS.TSTBLA.I0001.A250) -
    LINEAR     SPEED                        -
    SHAREOPTIONS(3 3)                       -
    TRACKS(1 1) VOLUMES(BI2452)         -
    REUSE    )                              -
  DATA                                      -
    ( NAME(DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A250))

//*********************************************************************
//CREATE   EXEC PGM=IKJEFT01,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-DIS UTIL(*)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP51)
//SYSIN    DD *
CREATE  DATABASE DBS1TBS ;
COMMIT;
//*********************************************************************
//STARTDB  EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-STA DB(DBS1TBS)
-DIS DB(DBS1TBS)
RUN PROGRAM(DSNTEP2) PLAN(DSNTEP51)
//SYSIN    DD *
CREATE TABLESPACE TSTBLA  IN DBS1TBS
    NUMPARTS 250
    (
    PART   1 USING VCAT DB2DBS ,

    etc.
    etc.
    etc.

    PART 250 USING VCAT DB2DBS
    )

  FREEPAGE 0
  PCTFREE 10
  BUFFERPOOL BP1
  LOCKSIZE ANY
  CLOSE NO;


    CREATE TABLE creatorS.TABLEA (
   COL_A CHAR(10)    NOT NULL,
   COL_B INT         NOT NULL,
   COL_C INT         NOT NULL,
   COL_D INT         NOT NULL,
   COL_E REAL        NOT NULL,
   COL_F REAL        NOT NULL,
   COL_G REAL        NOT NULL,
   COL_H REAL        NOT NULL,
   COL_I CHAR(1)     NOT NULL,
```

```
   COL_J CHAR(1)     NOT NULL,
   COL_K DATE        NOT NULL,
   COL_L DATE        NOT NULL,
   COL_M DATE        NOT NULL,
   COL_N CHAR(25)    NOT NULL,
   COL_O CHAR(10)    NOT NULL,
   COL_P VARCHAR(44) NOT NULL)
     IN  DBS1TBS.TSTBLA;



CREATE INDEX creatorS.SXTABLEA
  ON creatorS.TABLEA
(COL_A,COL_K,COL_C,COL_I,COL_F,COL_G)
  USING VCAT DB2DBS
  FREEPAGE 0
  PCTFREE 10
  BUFFERPOOL BP2
  CLOSE NO
  CLUSTER
  (

   PART   1 VALUES('  24000000'),

   etc.
   etc.
   etc.

   PART 250 VALUES('6000000000')
);
INSERT INTO creatorS.TABLEA VALUES(
'         1',
,1
,1
,1
,1
,1.0
,1.0
,1.0
,'1'
,'1'
,CURRENT DATE
,CURRENT DATE
,CURRENT DATE
,'1'
,'1'
,'1');
//*********************************************************************
//OBIDXLAT EXEC PGM=IKJEFT01,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
RUN PROGRAM(DSNTIAUL) PLAN(DSNTIB51) PARMS('SQL')
//SYSPUNCH DD DUMMY
//SYSRECOO DD DISP=(NEW,CATLG,DELETE),
//            SPACE=(TRK,(1)),UNIT=SYSALLDA,VOL=SER=DSSPKO,
```

```
//           DSN=creator.RISXTBLA.OBIDXLAT
//SYSIN    DD *
   SELECT
   SUBSTR(
   SUBSTR(DIGITS(A.DBID),2,4) ||
   ',' ||  SUBSTR(DIGITS(B.DBID),2,4) || ' DBID ' ||
   A.DBNAME || ' ' ||
   B.DBNAME ||
'                                                                  '
       ,1,70) , 'DBID    '
   FROM SYSIBM.SYSTABLESPACE A , SYSIBM.SYSTABLESPACE B
   WHERE A.DBNAME  = 'DBS1TBS'
    AND  B.DBNAME  = 'DBS1TB'
   UNION
   SELECT
   SUBSTR(
   SUBSTR(DIGITS(A.PSID),2,4) ||
   ',' ||  SUBSTR(DIGITS(B.PSID),2,4) || ' PSID ' ||
   A.NAME || ' ' ||
   B.NAME ||
'                                                                  '
       ,1,70)
   , 'PSID    '
   FROM SYSIBM.SYSTABLESPACE A , SYSIBM.SYSTABLESPACE B
   WHERE A.DBNAME  = 'DBS1TBS'
   AND   B.DBNAME  = 'DBS1TB'
   AND   A.NAME  = 'TSTBLA'
   AND   A.NAME  = B.NAME
   UNION
   SELECT
   SUBSTR(
   SUBSTR(DIGITS(A.OBID),2,4) ||
   ',' ||  SUBSTR(DIGITS(B.OBID),2,4) || ' OBID ' ||
   A.NAME || ' ' ||
   B.NAME ||
'                                                                  '
       ,1,70)
   , 'TBID    '
   FROM SYSIBM.SYSTABLES A , SYSIBM.SYSTABLES B
   WHERE A.DBNAME  = 'DBS1TBS'
   AND   B.DBNAME  = 'DBS1TB'
   AND   A.TSNAME  = 'TSTBLA'
   AND   A.TSNAME  = B.TSNAME
   AND   A.NAME  = 'TABLEA'
   AND   A.NAME  = B.NAME
   ORDER BY 2
   ;
//*********************************************************************
//STOP01   EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
-STO DB(DBS1TBS) SPACE(TSTBLA)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
//*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

A.1.2 Copy OBIDXLAT

```
//OBIDED10 JOB CLASS=A,TIME=NOLIMIT,REGION=5M,MSGCLASS=H,
//           NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//*********************************************************************
//OBIDEDIT PROC
//*        SIZE=4096K
//*        DSN=''
//*
//EDITTEMP EXEC PGM=DSN1COPY,REGION=4096K,PARM='RESET,OBIDXLAT'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//         DD DSN=DB2CT510.RUNLIB.LOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD DISP=SHR,DSN=&DSN
//SYSUT2   DD DISP=(NEW,PASS,DELETE),
//            SPACE=(TRK,(1)),UNIT=SYSALLDA,
//            DSN=&&OBIDEDIT
//COPYTEMP EXEC PGM=DSN1COPY,REGION=4096K,PARM='RESET'
//STEPLIB  DD DSN=DSN510.SDSNLOAD,DISP=SHR
//         DD DSN=DB2CT510.RUNLIB.LOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD DISP=(OLD,DELETE),DSN=&&OBIDEDIT
//SYSUT2   DD DISP=SHR,DSN=&DSN
//OBIDEDIT PEND
//*********************************************************************
//EDIT001  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A001
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*********************************************************************
//EDIT002  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A002
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*********************************************************************
//EDIT003  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A003
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*********************************************************************
//EDIT004  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A004
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*********************************************************************
//EDIT005  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A005
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*********************************************************************
//EDIT006  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A006
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*********************************************************************
//EDIT007  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A007
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*********************************************************************
//EDIT008  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A008
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*********************************************************************
//EDIT009  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
```

```
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A009
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*******************************************************************
//EDIT010  EXEC OBIDEDIT,TIME=NOLIMIT,COND=(EVEN),
//         DSN=DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A010
//SYSXLAT  DD DISP=OLD,DSN=creator.RISXTBLA.OBIDXLAT
//*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

A.1.3 Recover Index

```
//RCVIX1  JOB CLASS=A,TIME=NOLIMIT,REGION=5M,MSGCLASS=H,
//           NOTIFY=&SYSUID,MSGLEVEL=(1,1)
/*JOBPARM SYSAFF=QP01
//JOBLIB DD DSN=DSN510.SDSNLOAD,DISP=SHR
//       DD DSN=DB2CT510.RUNLIB.LOAD,DISP=SHR
//*******************************************************************
//RCVIX1  EXEC DSNUPROC,UID='RCVIX1',TIME=NOLIMIT,COND=(EVEN),
//         UTPROC='',SYSTEM='D511'
//DFSPARM DD *
  OPTION MAINSIZE=32M,FILSZ=U24000000
//UTPRINT  DD SYSOUT=*
//SORTDIAG DD SYSOUT=*
//SORTWK01 DD UNIT=SYSALLDA,SPACE=(CYL,(500,500)),VOL=SER=DSSPK2
//SORTWK02 DD UNIT=SYSALLDA,SPACE=(CYL,(500,500)),VOL=SER=DSSPK3
//SYSUT1   DD DISP=(NEW,CATLG,CATLG),
//         UNIT=SYSALLDA,VOL=SER=DSSPK1,SPACE=(CYL,(420,420)),
//         DSN=creator.RCVIX001
//SYSIN    DD *
 DIAGNOSE
  ABEND TRACEID X'1180' NODUMP
 RECOVER INDEX
    (creator.SXTABLEA PART 1)
//*******************************************************************
//TERMUTIL EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-TERM UTIL(RCVIX1)
//*
```

A.1.4 Intermediate Merge

```
//MERGE1   JOB CLASS=A,TIME=NOLIMIT,REGION=4M,MSGCLASS=H,
//        NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//*******************************************************************
//MERGE1   EXEC PGM=SORT,TIME=NOLIMIT,COND=(EVEN)
//DFSPARM DD *
  OPTION MAINSIZE=32M
//SYSPRINT DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//SORTIN01 DD   DISP=(OLD,KEEP,KEEP),DSN=creator.RCVIX001
//SORTIN02 DD   DISP=(OLD,KEEP,KEEP),DSN=creator.RCVIX011
//SORTIN03 DD   DISP=(OLD,KEEP,KEEP),DSN=creator.RCVIX021
   etc.
   etc.
   etc.
//SORTIN25 DD   DISP=(OLD,KEEP,KEEP),DSN=creator.RCVIX241
//SORTOUT  DD   DSN=creator.RCVIXM01,SUBSYS=BP01,
//              DCB=(LRECL=43,RECFM=FB,BUFNO=50)
```

```
//SYSIN    DD DISP=SHR,DSN=creator.RISXTBLA.MERGEFLD
//*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

A.1.5 Final Merge

```
//MERGEXX  JOB CLASS=A,TIME=NOLIMIT,REGION=6M,MSGCLASS=H,
//           NOTIFY=&SYSUID,MSGLEVEL=(1,1)
/*JOBPARM SYSAFF=QPO2
//*********************************************************************
//MERGEALL EXEC PGM=SORT,TIME=NOLIMIT,COND=(EVEN)
//DFSPARM DD *
  OPTION MAINSIZE=32M
//SYSOUT   DD   SYSOUT=*
//SORTIN01 DD   DSN=creator.RCVIXM01,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN02 DD   DSN=creator.RCVIXM02,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN03 DD   DSN=creator.RCVIXM03,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN04 DD   DSN=creator.RCVIXM04,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN05 DD   DSN=creator.RCVIXM05,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN06 DD   DSN=creator.RCVIXM06,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN07 DD   DSN=creator.RCVIXM07,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN08 DD   DSN=creator.RCVIXM08,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN09 DD   DSN=creator.RCVIXM09,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTIN10 DD   DSN=creator.RCVIXM10,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//SORTOUT  DD  DSN=creator.RCVIXALL,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//*
//SYSIN    DD DISP=SHR,DSN=creator.RISXTBLA.MERGEFLD
//*
//*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

A.1.6 Rename SHADOW to REAL Tablespace

```
//SWITCH1  JOB CLASS=A,TIME=NOLIMIT,REGION=5M,MSGCLASS=H,
//           NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//JOBLIB DD DSN=DSN510.SDSNLOAD,DISP=SHR
//       DD DSN=DB2CT510.RUNLIB.LOAD,DISP=SHR
//*********************************************************************
//STOPO2   EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
-STO DB(DBS1TBS) SPACE(TSTBLA) PART(1)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
//*********************************************************************
//SWITCH   EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
```

```
DELETE   'DB2DBS.DSNDBC.DBS1TBS.TSTBLA.ITEMP.A001'
ALTER    'DB2DBS.DSNDBC.DBS1TBS.TSTBLA.I0001.A001' +
 NEWNAME('DB2DBS.DSNDBC.DBS1TBS.TSTBLA.ITEMP.A001')
ALTER    'DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A001' +
 NEWNAME('DB2DBS.DSNDBD.DBS1TBS.TSTBLA.ITEMP.A001')
ALTER    'DB2DBS.DSNDBC.DBS1TBSS.TSTBLA.I0001.A001' +
 NEWNAME('DB2DBS.DSNDBC.DBS1TBS.TSTBLA.I0001.A001')
ALTER    'DB2DBS.DSNDBD.DBS1TBSS.TSTBLA.I0001.A001' +
 NEWNAME('DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A001')

//*********************************************************************
//START02  EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
-STA DB(DBS1TBS) SPACE(TSTBLA) PART(1)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
//*********************************************************************
//REPAIRL1 EXEC DSNUPROC,UID='REPAIR1',TIME=NOLIMIT,COND=(EVEN),
//         UTPROC='',SYSTEM='D511'
//SYSIN    DD *
 REPAIR LEVELID TABLESPACE DBS1TBS.TSTBLA PART    1
//*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

A.1.7 Final Build

```
//RCVIXMRG JOB CLASS=A,TIME=NOLIMIT,REGION=6M,MSGCLASS=H,
//           NOTIFY=&SYSUID,MSGLEVEL=(1,1)
/*JOBPARM SYSAFF=QP02
//*********************************************************************
//RCVIXXXX EXEC DSNUPROC,UID='RCVIXXXX',TIME=NOLIMIT,COND=(EVEN),
//         UTPROC='',SYSTEM='D511'
//SYSUT1   DD DISP=(NEW,DELETE,DELETE),
//         DSN=creator.RCVIXXXX,UNIT=SYSALLDA,
//         VOL=SER=BI7173,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
 DIAGNOSE
  ABEND TRACEID X'1180' NODUMP
 RECOVER INDEX
    (creator.SXTABLEA)
//*********************************************************************
//RCVIXMRG EXEC DSNUPROC,UID='RCVIXXXX',TIME=NOLIMIT,COND=(EVEN),
//         UTPROC='RESTART(PHASE)',SYSTEM='D511'
//SYSUT1   DD  DSN=creator.RCVIXALL,
//          SUBSYS=BP01,DCB=(LRECL=43,RECFM=FB,BUFNO=50)
//*
//SYSIN    DD *
 RECOVER INDEX
    (creator.SXTABLEA)
//*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

A.1.8 Rename Temporary Back to REAL Database

```
//SWATCH1  JOB CLASS=A,TIME=NOLIMIT,REGION=5M,MSGCLASS=H,
//           NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//JOBLIB DD DSN=DSN510.SDSNLOAD,DISP=SHR
//       DD DSN=DB2CT510.RUNLIB.LOAD,DISP=SHR
//*****************************************************************
//STOPO3   EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
-STO DB(DBS1TBS) SPACE(TSTBLA) PART(1)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
//*****************************************************************
//SWATCH   EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN  DD *
ALTER    'DB2DBS.DSNDBC.DBS1TBS.TSTBLA.I0001.A001' +
 NEWNAME('DB2DBS.DSNDBC.DBS1TBSS.TSTBLA.I0001.A001')
ALTER    'DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A001' +
 NEWNAME('DB2DBS.DSNDBD.DBS1TBSS.TSTBLA.I0001.A001')
ALTER    'DB2DBS.DSNDBC.DBS1TBS.TSTBLA.ITEMP.A001' +
 NEWNAME('DB2DBS.DSNDBC.DBS1TBS.TSTBLA.I0001.A001')
ALTER    'DB2DBS.DSNDBD.DBS1TBS.TSTBLA.ITEMP.A001' +
 NEWNAME('DB2DBS.DSNDBD.DBS1TBS.TSTBLA.I0001.A001')

//*****************************************************************
//START02  EXEC PGM=IKJEFT01,TIME=NOLIMIT,COND=(EVEN)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(D511)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
-STA DB(DBS1TBS) SPACE(TSTBLA) PART(1)
-DIS DB(DBS1TBS) SPACE(*) LIMIT(*)
//*****************************************************************
//REPAIRL1 EXEC DSNUPROC,UID='REPAIR1',TIME=NOLIMIT,COND=(EVEN),
//         UTPROC='',SYSTEM='D511'
//SYSIN    DD *
 REPAIR LEVELID TABLESPACE DBS1TBS.TSTBLA PART    1
//*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Appendix B. Special Notices

This publication is intended to help technical people undertaking the build of VLDB using DB2 for OS/390. The information in this publication is not intended as the specification of any programming interfaces that are provided by DB2 or OS/390. See the PUBLICATIONS section of the IBM Programming Announcement for DB2 and OS/390 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document 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 customer′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.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

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

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of

including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

DB2	DFSORT
IBM	OS/390
Parallel Sysplex	RAMAC
S/390	S/390 Parallel Enterprise Server
Smartbatch	S/390 Parallel Enterprise Server

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

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

Appendix C. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 39.

- *DB2 for MVS/ESA Version 4 Data Sharing Implementation*, SG24-4791

C.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

C.3 Other Publications

These publications are also relevant as further information sources:

- *DB2 for OS/390 Version 5 Release Guide*, SC26-8965
- *DB2 for OS/390 Version 5 Utility Guide and Reference*, SC26-8967
- *DB2 for OS/390 Version 5 SQL Reference*, SC26-8966
- *IBM SmartBatch for OS/390 Overview*, GC28-1627
- *Data Warehousing with DB2 for OS/390 Version 2.5*, SG24-2249
- *DB2 for OS/390 Capacity Planning*, SG24-2244
- *DB2 for OS/390 Application Design Guidelines for High Performance*, SG24-2233

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com/.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

  For a list of product area specialists in the ITSO: type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  http://w3.itso.ibm.com/redbooks/

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.html). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com/
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.html). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

Special Characters

Numerics

B

C

D

E

H

I

J

L

N

O

P

R

S

T

U

V

W

ITSO Redbook Evaluation

DB2 for OS/390 Terabyte Database:Design and Build Experiences
SG24-2072-01

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____

_____

_____

_____

_____

SG24-2072-01
Printed in the U.S.A.