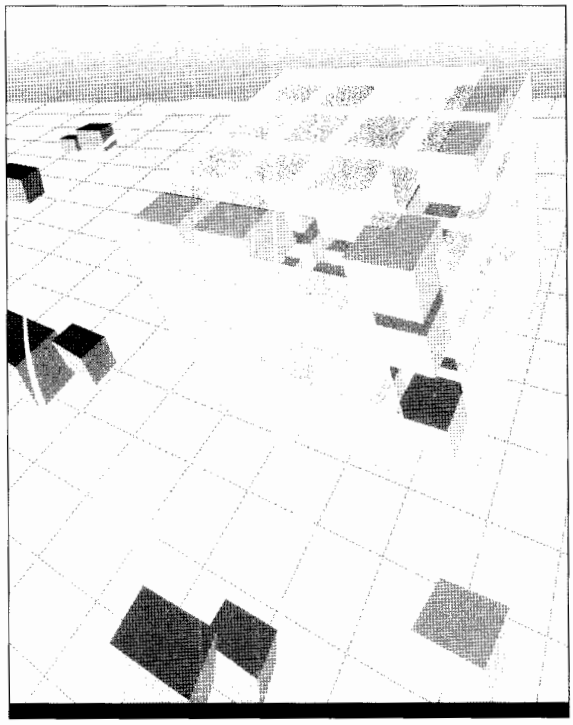
T A N D E M

SYSTEMS REVIEW



Implementing Decision Support Systems

DSS Database Design

Late Binding in NonStop SQL/MP

Technical Information and Education

Product Update

T A N D E M

SYSTEMS REVIEW

V O L U M E 1 0 , N U M B E R 4

O C T O B E R 1 9 9 4

The Tandem Systems Review *publishes technical information about Tandem software releases and products. Its purpose is to help programmers, analysts, and other IS professionals to plan for, install, use, and tune Tandem systems.*

TANDEM

Editor's Note

The first two articles in this issue of the *Tandem Systems Review* focus on decision support systems (DSS). "Implementing Decision Support Systems" discusses issues MIS planners and analysts will need to consider when they design, create, and maintain a DSS. It explores each stage of DSS development, from designing a data warehouse to selecting workstation-based software tools for querying the data. The article draws on the experiences of users who have implemented DSS on Tandem systems.

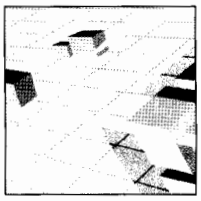
"Issues in DSS Database Design" describes important differences between OLTP and DSS databases. It focuses on three topics in DSS database design: designing dimensional DSS databases, defining views to simplify query generation for end users, and partitioning strategies for handling large amounts of historical data.

The third article, "Late Binding and High Availability Compilation in NonStop SQL/MP," describes new features in NonStop SQL/MP that make it easier to install and manage application programs. By reducing the need for SQL-compilations and auto-compilations, the new features improve the availability of applications and the productivity of programmers and database administrators. The user scenarios in the article suggest how and when these features may be useful.

We regret to announce that this will be the last issue of the *Tandem Systems Review*. The magazine started in 1985 (replacing the *Tandem Journal*, which started in 1983), and we hope that you have found it useful over these many years. Tandem continues to be committed to providing technical information that addresses its users' business concerns and to delivering that information in a variety of forms.

–The TSR staff

Tandem Systems Review is published quarterly by Tandem Computers Incorporated. All correspondence should be addressed to *Tandem Systems Review*, 10400 Ridgeview Court, Loc 208-65, Cupertino, CA 95014.



DECISION SUPPORT

NONSTOP SQL/MP

DEPARTMENTS

Product Update

NonStop-UX Based Software

NonStop-UX Operating System, Release B30

July 1994

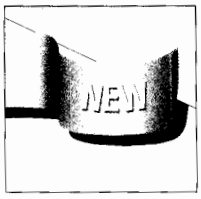
Tandem has announced a new release, B30, of the NonStop-UX operating system for Integrity FT systems. The B30 release of the operating system includes the following new availability features and performance improvements:

- New online software upgrade. This software upgrade reduces system downtime for future system upgrades to under two minutes (the previous version required several hours).
- New remote software installation. This feature increases flexibility in installing operating environment options for networks of Integrity FT systems.
- New scheduled CPU reintegration. This feature allows selective scheduling of CPU reintegration, facilitating the reintegration of applications at off-peak times.
- New tunable CPU reintegration. This feature sets an extended time frame for CPU reintegration in order to limit the amount of time that an application is unavailable during CPU reintegration.
- Faster CPU reintegration time. This improvement speeds up CPU reintegration by 30 to 65 percent.
- Improved availability monitoring tools. These tools improve system outage analysis by logging additional availability events and system statistics.
- New PIXIE application profiling tool. This tool allows a developer to profile an application during development in order to improve application performance.
- New multiprotocol support on synchronous controller. This feature enables both X.25 and SNA protocols to coexist on a single four-port synchronous controller.

Sybase System 10 for Integrity NR Servers

June 1994

Sybase System 10 software is now available from Sybase for the Tandem Integrity NR family of servers. Sybase System 10 is a modular family of integrated software products designed for enterprise-wide client/server computing. The System 10 software consists of a distributed relational database management system (RDBMS), application development tools, system administration tools, and interoperability tools. The Sybase distributed RDBMS runs on the server to support online mission-critical applications as well as decision support applications.



The Product Update department provides brief descriptions of new products announced by Tandem. For more information on any of these products, please consult your local Tandem representative.

Integrity Systems

New Integrity FT Systems Products

July 1994

Tandem is offering a number of new products for Integrity FT systems that complement and take advantage of new features in the B30 release of the NonStop-UX operating system. The new products include the following:

- A 256-megabyte local memory option for Integrity FT 1475 systems that takes advantage of the new 384-megabyte total memory maximum supported by the NonStop-UX operating system.
- An SCSI-2 controller that doubles the maximum SCSI bus capacity compared to the older-technology SCSI controller; the two-channel SCSI-2 controller supports twice as many SCSI devices.
- An Async II communications controller that improves performance by as much as four times compared to the asynchronous controller it replaces; it also improves modem connectivity and security as well as line utilization.
- A fiber distributed data interface (FDDI) communications option that doubles the potential throughput compared to TCP/Ethernet in a LAN environment; the FDDI controller supports 100-megabyte/second communication over fiber-optic networks.
- Mass storage cabinets that are no longer bundled with disk drives, allowing for maximum ordering flexibility.
- Integrity CO System cabinet doors with air filters to meet NEBS regulations.

Client/Server Computing Products

Pathway Open Environment Toolkit, Release 2

June 1994

Pathway Open Environment Toolkit (POET) is a middleware product that makes it possible for programmers to use popular Windows and OS/2 based client tools in developing clients for Tandem NonStop OLTP applications. Release 2 of POET includes the following major enhancements:

- Custom server interface generation and data conversion for VisualBasic, PowerBuilder, and SQLWindows.
- Automatic data conversion by the Communications Manager. A C compiler is no longer required to use POET data conversion.
- Pathmaker is no longer required when using POET. DDL dictionaries and Pathmaker projects can be accessed directly.
- Support for Remote Server Call (RSC) UMS for unsolicited message processing.
- Data conversion support for servers written in C and TAL.

Communications and Networking Products

SX25 Software for Integrity NR Servers and UNIX Workstations

July 1994

Based on the Spider Systems X.25 software, Tandem's SX25 software package supports an integrated PAD, Network Layer Interface (NLI) programmatic interfaces, switched virtual circuits (SVC), and permanent virtual circuit (PVC) operation. The software supports the CCITT 1980, 1984, and 1988 standards as well as the X.3, X.28, and X.29 standards. Currently, SX25 can support as many as 255 virtual circuits per system. The software also provides built-in tracing and monitoring tools to help users manage their X.25 configurations.

The SX25 software product runs on the Integrity NR servers as well as on the Indy™ and Indigo^{2}™ workstations available from Tandem. The software includes the system-specific drivers for each of the supported servers and workstations.

Token Ring Software for Integrity NR SMP Servers

July 1994

The new token ring software for the Integrity NR SMP servers provides token ring system drivers and various utilities. This software is required, along with the new token ring controller, to run Transmission Control Protocol (TCP) over token ring. This software and the new token ring controller plus the SNA server software and the appropriate SNA client software is required in order to run SNA over token ring.

SNA Support for Integrity NR Servers and UNIX Workstations

July 1994

Tandem now offers three new software products that support standard SNA communications. These products operate in a client/server configuration and run on the Integrity NR servers as well as on the Indy and Indigo^2 workstations.

The new SNA Server software provides the path from an NR server or a workstation to another SNA host or peer system. The physical path can be either SDLC (via a synchronous controller) or token ring. The SMP servers (NR4436 and NR4412) support four SDLC lines; the uniprocessor servers and the workstations support two lines. The SNA Server software supports 254 LU6.2 sessions and 32 3270 LU sessions.

The new SNA 3270 software provides emulation of IBM 3278 and 3279 terminals (models 2 through 5). The software includes HLLAPI, support for extended attributes and write structured field program interface support, IBM IND$FILE file transfer, cursor support for nongraphics terminals, and IBM 3270 international keyboard support.

The new SNA LU6.2 software supports advanced program-to-program communications (APPC) with other LU6.2 nodes in an SNA network. The software supports 254 parallel sessions per logical unit (LU), mapped and basic conversations, multiple modes of service, and a programmable operator interface.

OSI Transport Software for Integrity NR Servers and UNIX Workstations

July 1994

The new OSI Transport software provides support for third-party X.400 applications and other applications that use the OSI transport protocol. The OSI Transport software provides Level 4 OSI transport protocol; TP0, TP2, and TP4 service support; CLTP support; and support for the UNIX System V standard Transport Library Interface (TLI) for application usage. The software is compatible with UK GOSIP 3.1 and USA GOSIP 2.0 and supports connectionless-mode network service (CLNS) and connection-mode network service (CONS) as well as end system to intermediate system (ES-IS) routing.

The OSI Transport software supports OSI communications on the Integrity NR servers and Indy and Indigo^2 workstations over Ethernet, fiber distributed data interface (FDDI), or X.25. The server or workstation must include the network controller offered by Tandem appropriate for the chosen protocol.

Multiport Communication Controllers for Integrity NR Servers and UNIX Workstations

July 1994

Tandem now offers three new synchronous multiport controllers that support X.25 and SNA/SDLC communications on Integrity NR servers and Indy and Indigo^2 workstations. These multiport controllers can support both SNA and X.25 on the same controller, freeing a system bus slot and reducing the system's communications cost when using both protocols. Each controller port can be configured for either an RS-232D, V.35, or X.21 electrical interface.

The new four-port synchronous, VME interface controller for the Integrity NR SMP servers provides four ports of synchronous connectivity at line speeds up to 64 kilobits/second each. The NR4412 server supports one synchronous controller; the NR4436 server supports as many as four controllers.

The new two-port synchronous, ISA interface controller for the Integrity NR uniprocessor servers and the Indigo^2 workstations provides two ports of synchronous connectivity at line speeds up to 64 kilobits/second each. The uniprocessor servers support four synchronous controllers; the Indigo^2 workstations support two synchronous controllers.

The new two-port synchronous, GIO interface controller for Indy workstations provides two ports of synchronous connectivity at line speeds up to 64 kilobits/second each. The Indy workstations support one synchronous controller.

Token Ring Controllers for Integrity NR SMP Servers and Indy Workstations

July 1994

Two new token ring controllers are now available to provide IBM host access and/or TCP/IP or SNA access to other Integrity NR SMP servers and Indy workstations. These new controllers comply with IEEE 802.2 and 802.5 standards.

The new token ring, VME interface controller for the Integrity NR SMP servers operates at standard 4-megabit or 16-megabit line speeds to support TCP/IP or SNA network access. The SMP servers can support one token ring controller per system.

The new token ring, GIO interface controller for Indy workstations operates at standard 4-megabit or 16-megabit line speeds to support TCP/IP or SNA network access. Indy workstations can support one token ring controller per system.

Ethernet Controller for Indy Workstations

July 1994

A new Ethernet controller for the Indy workstations is now available. Adding this controller increases the total number of Ethernet ports from one to two on the Indy workstations. This new AIU Ethernet controller supports TCP/IP and OSI networks in the same manner as the built-in Ethernet port. The Ethernet drivers and utilities are resident on the Indy workstations.

Storage Products

Modular Storage System

August 1994

Tandem's Modular Storage System is an innovative packaging scheme that lets users mix disk and tape devices for NonStop servers in the space normally occupied by a single-purpose cabinet. In the Modular Storage System, disk and tape drives stack vertically, thus reducing the number of required cabinets, the floor space requirements, and the overall system size. For example, a fully configured stack using the 4560 disk subsystem provides 160 gigabytes of formatted storage capacity in a footprint of less than 67 by 89 centimeters (26.2 by 35 inches).

Users can begin a Modular Storage System with one module, then add more as their needs increase. Each stack can hold a mix of disk and tape modules, and the system does not have to be placed in a special computer room.

5411 and 5420 Optical Storage Subsystems

August 1994

Tandem's 5411 and 5420 optical storage subsystems provide online access to large archives of data. These devices use write once, read many (WORM) optical technology to provide online access to archived data with response times ranging from 0.5 to 8.5 seconds. The 5411 and 5420 provide cost-effective alternatives to conventional media for archiving application data and for storing and retrieving large objects such as images.

The 5411 subsystem is a fully automated device for medium-sized applications. This low-cost, entry-level subsystem includes an automatic disk changer, one optical disk drive, and a control unit. A fully configured 5411 contains 12 disk cartridges, providing nearly 79 gigabytes of online storage.

The 5420 subsystem is designed for very large storage requirements. It includes an automatic disk changer, one to four optical disk drives, and a control unit. When fully configured, the 5420 handles as many as 77 disk cartridges, providing more than 504 gigabytes of online storage. With the 5420, users can customize their applications by choosing the most cost-effective combination of performance (one to four optical disk drives) and storage capacity (308 to 504 gigabytes).

4560 Disk Subsystem

August 1994

The Tandem 4560 disk subsystem provides high-capacity, cost-effective storage for high-performance online transaction processing (OLTP) and large-database applications. The 4560 uses high-capacity, low-cost-per-megabyte disk drives that each store up to 2 gigabytes of formatted data.

As part of Tandem's Modular Storage System, 4560 modules provide high storage density and can be stacked with various tape devices to keep pace with growing storage needs while saving floor space. The 4560 is designed to operate in a normal office environment and can be serviced online.

Using advanced fiber-optic cabling, the Tandem 3129 disk controller connects 4560 disk subsystems to the NonStop servers. The 3129 supports cable lengths of up to 2,000 meters (6,560 feet), which allows users to locate disk subsystems in a different room or on a different floor from the NonStop server.

Multifunction Controller for 5190 Cartridge Tape Subsystems

June 1994

Multifunction Controller (MFC) support is now available for the 5190 family of cartridge tape subsystems. With the 5190 attached to the MFC, the entire range of tape utilities, including "TAPEBOOT," is fully supported on NonStop systems such as the CLX800, K100, and K1000. MFC support provides the following user benefits:

- An additional tape drive is no longer needed to support utility functions.
- The cost of a dedicated tape controller can be saved on small systems where the sustained throughput of the MFC is adequate to perform all tape and backup functions.
- The overall cost of a Tandem NonStop system can be reduced, since users can now match tape subsystems to performance requirements as needed.

Workstation and Terminal Products

Outside View with Tandem Terminal Emulation v3.2

June 1994

Outside View with Tandem Terminal Emulation (TTE) v3.2 is a new, more powerful version of TTE for Windows. The new release contains many new features including the following:

- Added Communications. TCP/IP: FTP software; TCP/IP: Novell LAN Workplace; TCP/IP: Windows Sockets; interrupt 14h.
- New User Functionality. Programmable toolbar; active status line; preference dialog; extensive online help.
- New Macro Language. VisualBasic language compatible; built-in compiler and editor; over 160 BASIC commands.
- Additional New Features. IBM TN3270 emulation; expanded documentation; Windows installation program.

Implementing Decision Support Systems

In a decision support system (DSS), users retrieve information from a large database, called a *data warehouse*, that stores historical as well as recent corporate data gathered from multiple sources. DSS applications allow users to query the data warehouse and perform various types of information analysis. For example, corporate managers can identify trends in geographic data, demographics, product sales, or any other information that facilitates decision-making.

Rapidly decreasing technology costs make it feasible to build a stand-alone DSS environment that contains a data warehouse separate from existing operational databases. By using separate technology for DSS, one can ensure that the DSS environment does not affect the performance of day-to-day operational systems. However, given the large amount of data to be queried and the variety of queries to be executed, a decision support solution can succeed only with proper planning and design.

Since the data warehouse stores historical data for long periods of time, the appropriate architecture must be in place before one loads data into the warehouse. Once loaded, the data is difficult, if not impossible, to redesign without reloading the entire data warehouse.

The applications that access the data warehouse vary both in purpose and architecture. DSS applications can be host-based or workstation-based. The trend is to use the workstation (client/server) approach, because with it one can quickly apply new hardware and software solutions for accessing the data warehouse.

This article discusses some of the issues one needs to address when implementing a decision support system. It explores the following topics: designing the data warehouse, loading data from various sources, keeping the data warehouse up-to-date, performing capacity planning, using client/server technology and workstation-based software tools, and, finally, developing appropriate DSS applications with those tools.

The article is based in part on the experiences of users who recently implemented a decision support system on a Tandem™ database server. The case study cited in the article shows how, with careful planning and design, users successfully built a Tandem DSS environment.

Decision Support Environment

A decision support environment usually consists of one or more legacy systems such as batch or online production systems, a data warehouse system, and many client systems that access both the legacy systems and the data warehouse. The decision support environment discussed in this article contains all three components: a legacy system, a data warehouse, and client systems.

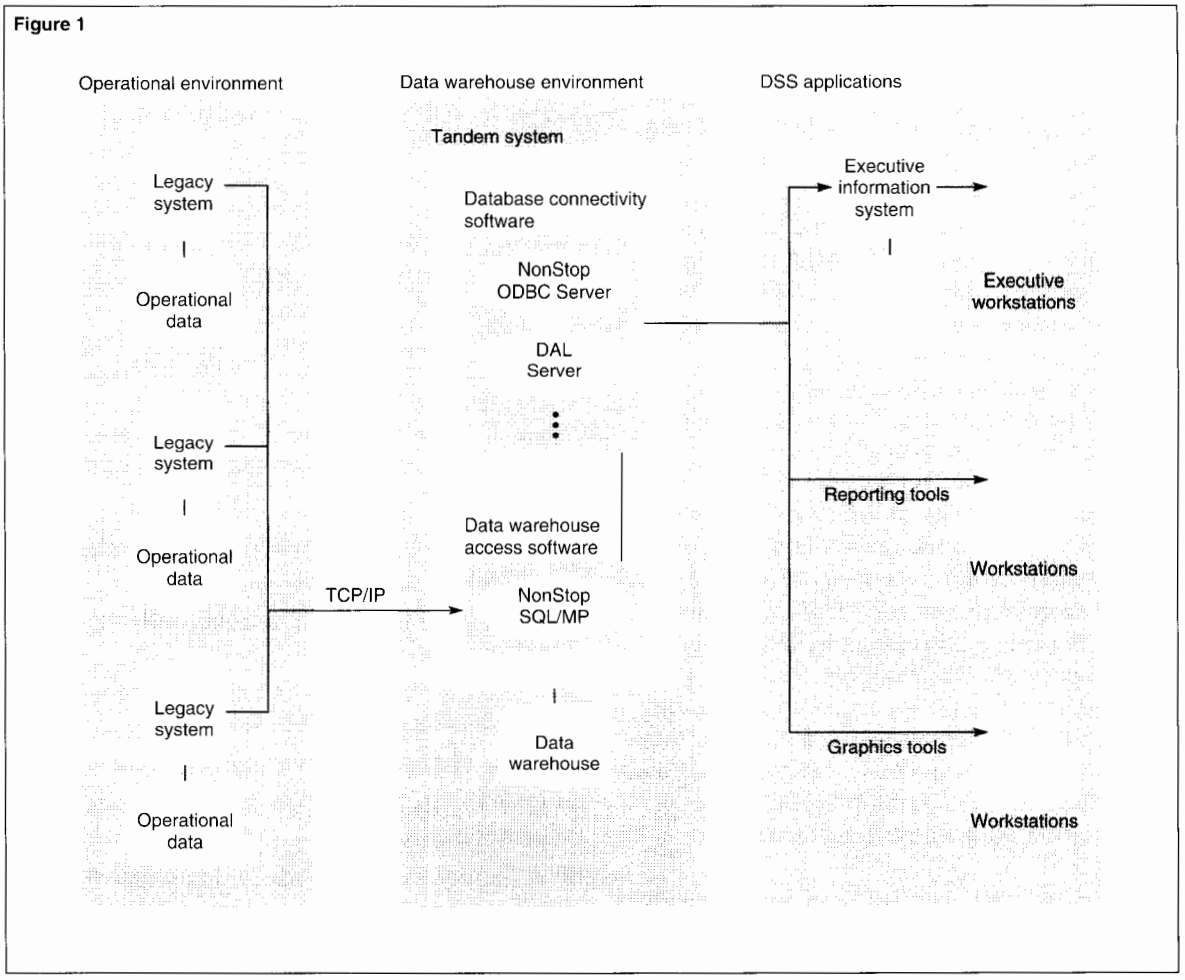
Before installing DSS components, one must be sure one can integrate the new components into the existing operational environment. One must consider issues such as the network infrastructure, connectivity to the legacy system, and support for client workstations and software. In the case-study system, the users met the requirement to seamlessly integrate DSS components into the existing environment by using several Tandem products. For example, TCP/IP software provided access to the legacy system, and the NonStop™ ODBC Server allowed client systems to access the data warehouse. Mahbod and Slutz (1994) describe the NonStop ODBC Server in the July 1994 issue of the *Tandem Systems Review*.

The users in the case study selected a Tandem database server to support the data warehouse because of Tandem's NonStop SQL/MP relational database management system, which allows parallel execution of queries and online management of the database. By using the parallel capabilities of NonStop SQL/MP, the users could populate the data warehouse in a relatively short time. (It took only five days to load a 140-gigabyte database that had 80 partitions.) Several articles in the July 1994 issue of the *Tandem Systems Review* describe the new features in NonStop SQL/MP (Ho et al., 1994; Troisi, 1994; and Zeller, 1994).

Decision support encompasses the way data warehouse information is used and the applications that access that information. The data warehouse maintains, in fixed formats, excerpts (snapshots) of integrated data (summarized and aggregated data from one or more legacy systems) that at one time was operational. It can support various types of information-access processing required by decision support, executive information, reporting, and other systems. The data warehouse is updated periodically from operational and other types of databases and is designed to separate the operational environment from the historical environment. The basic characteristics of the data warehouse are as follows:

- It contains a large amount (often hundreds of gigabytes) of physical data.
- The data is historical, possibly reflecting years of information.
- The data architecture is integrated to reflect the business needs of the organization.
- It is a nonvolatile environment.
- Queries applied against the data may be scheduled or ad hoc.
- A facility exists to perform large data transfers between the legacy system and the data warehouse.
- Tools provide useful information to the end users.

Figure 1.
A typical decision support environment.



To build the proper data structures into the data warehouse, one must understand that it will be the querying environment for end-user decision support applications. These applictions create ad hoc queries that fluctuate greatly in duration and complexity. Moreover, because the data warehouse is so large, it can take weeks to load the database for a data warehouse. Figure 1 illustrates a decision support environment similar to the one discussed in this article.

Designing the Data Warehouse

Before designing the data warehouse, one must create a corporate data model. (The structure of the data warehouse will be a subset of the corporate data model.) Data modeling consists of gathering requirements for the data that users will access, designing a logical model based on the requirements, and then imposing a physical structure (including data keys, data fields, and data attributes) on the logical model. Glasstone (1994) discusses design features of the decision support database elsewhere in this issue of the *Tandem Systems Review*.

In the case study, the users created a data model from the ground up. This allowed them to design the model to reflect the needs of the DSS users, not those of the legacy system users.

After creating the corporate data model, one must design the process of loading data into the data warehouse. Thus, before the data is actually loaded, one must include the following activities in the plan to populate the data warehouse:

- Determine which operational data is needed for the decision-making process.
- Add a time element to the key structure of the data warehouse (if one is not already present).
- Create appropriate processes to aggregate data.
- Transform data relationships into data artifacts (data objects with a known meaning).
- Accommodate the different levels of granularity (for example, the need to summarize data at multiple levels) found in the data warehouse.
- Identify like data from different operational tables to avoid redundancy.

This article does not discuss the details of each of the preceding activities. As these activities indicate, however, it takes careful planning to build and populate a data warehouse. The users in the case study took about four months to design their 140-gigabyte data warehouse and create a plan to load it.

Loading the Data Warehouse

Once the environment, the data structures, and the connectivity to the data warehouse have been designed and planned for, users must develop a plan to load the data warehouse. Three types of data loading must be addressed (Inmon, 1992 and 1993b):

- Loading data already archived from the legacy system. This process is called the *initial load*.
- Loading current operational data generated by existing applications.
- Trapping ongoing changes in the operational environment (those that have occurred since the last time data was loaded into the data warehouse).

The initial load moves archived data from bulk storage into the data warehouse. The second type of loading accounts for changes that occur in the legacy (operational) system during the time lag between the latest archiving of data from the operational system and the end of the initial load. A process must load these changes into the data warehouse, synchronizing the data warehouse with the operational database. The third type of loading, trapping data, is performed periodically (daily or weekly, for example) throughout the life of the system to keep the data up to date. There are many ways to trap data. For example, a process can capture after-images of transactions, or one can run batch jobs that scan the operational database for changes based on some criterion, usually a date range.

The users in the case study planned to load the data warehouse with archival and operational data from bulk storage and then perform an update with recent operational data. To facilitate this strategy, they created two plans. The first addressed the loading of the archival and operational data; the second addressed trapping ongoing changes. To minimize the loading time and periodic synchronization of data, the users decided to make use of the ability of Tandem systems to perform parallel loads. By using multiple tape drives as input devices and allowing many NonStop SQL processes to run in parallel on separate processors, they could load the data warehouse in a few days. If the load had been performed sequentially, it would have taken eight times longer (40 days instead of 5).

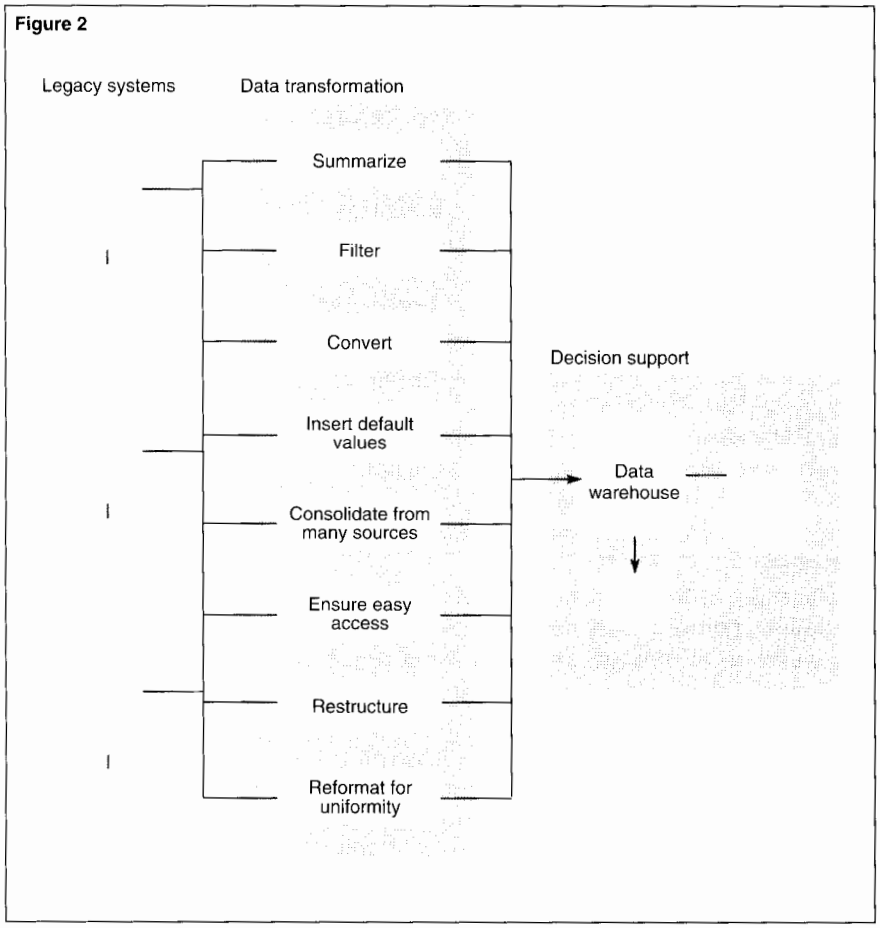


Figure 2.

Components of data transformation that must take place before data is loaded into the data warehouse.

The archival and operational data from the legacy system was sorted by date and loaded onto input tapes. The users did not perform data transformation on the legacy system; they intended to minimize the overhead associated with gathering the data. Instead, data transformation was performed on the Tandem database server. The users employed the Tandem Databuild product to convert the data from the legacy-system format into a Tandem (ASCII) format. The transformation program addressed issues such as binary data representation and date representation. Once a record was transformed and all edits passed, the record was added to the user's data warehouse.

Figure 2 illustrates components of data transformation that must be accomplished before the data can be loaded into the data warehouse.

Loading Archival Data

Loading existing archival data into the empty data warehouse was the largest and most time-consuming of the three data loading operations undertaken by the users in the case study. However, this was also the simplest load to plan and implement. The users established a recovery point where partitions of the database could be reinitialized, if necessary; the load could resume from the recovery point. Recovery during a load is not so easily accomplished after the data warehouse goes online.

Archival data is usually stored in some form of sequential bulk-storage device (such as a StorageTek Silo or another type of tape library). The data is read from bulk storage and transformed into the format needed by the data warehouse. The same data transformation occurs for all three types of loads (Inmon, 1993a).

A load from archival data is performed only once. The users in the case study were not too concerned about the resources consumed because the Tandem database server performed both the loading and the data transformation; the operational systems were not affected. The users loaded the data warehouse using as much processor utilization as possible for as long as it took to complete the load.

Loading Data Contained in Existing Systems

Unlike the archival data, which is loaded when the data warehouse is empty, the data contained in existing operational systems must be loaded into a growing data warehouse. The execution of this process must be carefully timed because the data in the warehouse begins to age as soon as it is placed there. In addition, one must consider that loading the data entails moving it to a different type of system. (The data warehouse technology is rarely the same as the operational one.)

Instead of directly reading the existing system's database, one can download it to a sequential medium such as tape and then transform and load the data into the data warehouse using the flat files. This approach has many desirable aspects. First, often one can download the data to a sequential medium by executing a utility that operates efficiently. Once the data is downloaded, another processor can perform the transformation. Thus, the download places a minimal burden on the online operational system.

Second, by moving the data from the legacy system to a sequential medium and then transforming the sequential data in the data warehouse, one minimizes the complexity of manipulating the data in the existing operational environment. It is easier to convert the data after it resides in the data warehouse because the data warehouse can be designed to accommodate the processing demands of data transformation. As a result, resource (processor and disk) consumption in the legacy system may also be minimized.

The users in the case study followed this approach. They downloaded the data contained in their existing systems to an intermediate sequential file. The Tandem database server read the sequential file, transformed the existing operational data into the appropriate format, and loaded the data warehouse. The Tandem server could accomplish this operation efficiently because it could perform parallel queries and parallel data loading, as shown in Figure 3.

Trapping Changes in Existing Systems Since the Last Load

When the initial load is complete, many months (or years) of data will be available to end users. To keep the data in the data warehouse current, the implementation plan must include processes that move new and changed data from the operational environment to the DSS environment.

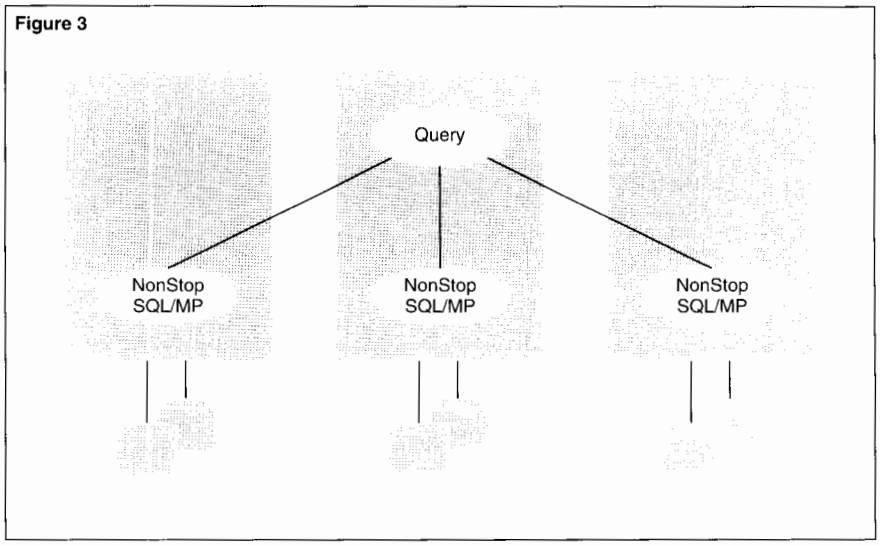


Figure 3.
Tandem servers can break up individual queries and execute them in parallel across many processors and disk volumes.

The ongoing maintenance of the data warehouse has two objectives. First, the data warehouse must reflect the current state of the business; the data must remain up-to-date. Second, when data becomes too old to be useful, it should be *rolled over* (removed) from the data warehouse.

The third load operation loads changes made in the operational environment since the last time the data warehouse was refreshed. In the case study system, the data is refreshed weekly. The data from the new week rolls over data from the oldest week. The rollover period will vary for each decision support system. Most installations store data for one to five years. For some businesses, ten years is not impractical.

Figure 4

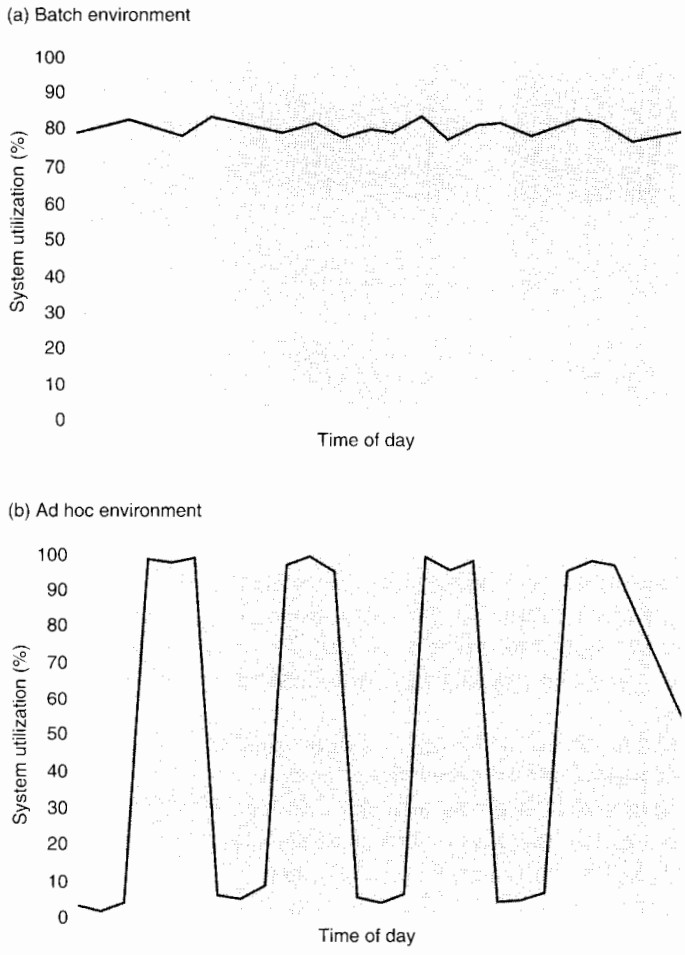


Figure 4.

(a) In an operational (batch) environment, system utilization is fairly steady. (b) In a decision support (ad hoc) environment, system utilization fluctuates greatly.

In the case-study system, data from the operational environment refreshes the data warehouse each weekend. During the week, the operational environment sustains updates; end users insert, delete, and change individual records. On Friday, it is time to refresh the data warehouse again. Only the changes made during the week need to be entered into the data warehouse. The challenge is how to identify the changed records.

The operational data usually cannot be recreated, scanned, and rescanned before it refreshes the data warehouse. The resources and time required to do this are beyond the means and desires of most companies. In a good interface design between the operational system and the data warehouse, each unit of data is scanned and transformed only once in its life.

Companies can use many techniques to identify and segregate periodic changes in a file. (Most companies have more than one approach to this task.) The techniques include the following (Inmon, 1993a):

- Replacing an entire table.
- Using dates embedded in the operational file to select only the appropriate records.
- Using a *delta* or audit file created by the operational application.
- Altering application code to create a delta or audit file.
- Trapping changes at the database management system (DBMS) level.
- Comparing before-and-after images of the file.

The users in the case study use dates to identify records that have been added or changed since the last refreshment cycle. The changes are written to a large sequential file. The users then download the file from the legacy system to the Tandem database server through a high-speed connection. A program in the Tandem server transforms the data and loads the new or changed records, refreshing the data warehouse.

Maintaining the Data Warehouse

Given the volume of data and the types of queries that execute on that data, a DSS application can consume large amounts of resources. Organizations can prepare for their hardware resource utilization and anticipate response times by developing maintenance schedules and doing capacity planning before they build the data warehouse.

However, capacity planning for the data warehouse is challenging. First, unlike operational systems, which have fairly regular workloads and utilization patterns, DSS application workloads can fluctuate greatly. Figure 4 compares the system utilization in a DSS environment with that of a batch environment on a given day.

Second, the data warehouse usually contains much more data than does the operational environment. The size of the data warehouse is directly related to its design (which, on a Tandem system, includes issues such as partitioning and index location) and the length of time the data is stored in the warehouse. Longer duration and finer granularity of data result in more data in the warehouse and require more disk space to support. The volume of data affects not only disk storage, but also the resources (such as CPU and memory) required to query the data. Thus, in a data warehouse, the performance and capacity of a system are closely linked to its design.

The users in the case study plan to enlarge their DSS environment in a modular fashion, adding disk and processing capacity as end-user activity increases. The scalable Tandem database server's ability to perform near-linear speedup and scaleup satisfies these requirements.

Third, the dynamics of the operational environment, and thus the planning model for that environment, cannot be applied to the data warehouse. One must understand the differences between an OLTP (operational) environment in which response time is critical and the DSS environment in which response time may not be critical. Further, in the DSS environment, system resources may be lightly utilized one moment and totally consumed the next. Tandem's mixed workload environment addresses the dynamics of the data warehouse environment by dynamically adjusting the priorities of long-running queries so that queries of shorter duration can execute quickly. This feature helps to balance resource utilization, making the DSS environment more predictable in spite of the unpredictable requests users make of the DSS resources.

The operational environment uses hardware in a static fashion. There are peaks and valleys, but they change only as the business changes; they are predictable and fairly constant. In contrast, the users of the data warehouse use resources constantly or not at all, and at different times. Thus, the pattern of hardware utilization is unpredictable, making it more difficult to address capacity planning for a data warehouse. The different utilization patterns of the two environments apply to all components of the system, from disk and processor utilization to the amount of memory available.

The following paragraphs describe a few more characteristics of the data warehouse environment as they apply to capacity planning (Inmon, 1993a). Since the data warehouse is usually built on database server technology, capacity planning tends to center around issues such as disk and processing resources.

Processor capacity is a function of the workload passing through the environment. The workload consists of background processing (such as loads, database reorganizations, and deletes), predictable DSS processing, and unpredictable DSS processing. The capacity planner pieces together the profiles of these transaction and query types to produce the anticipated model of the resources needed.

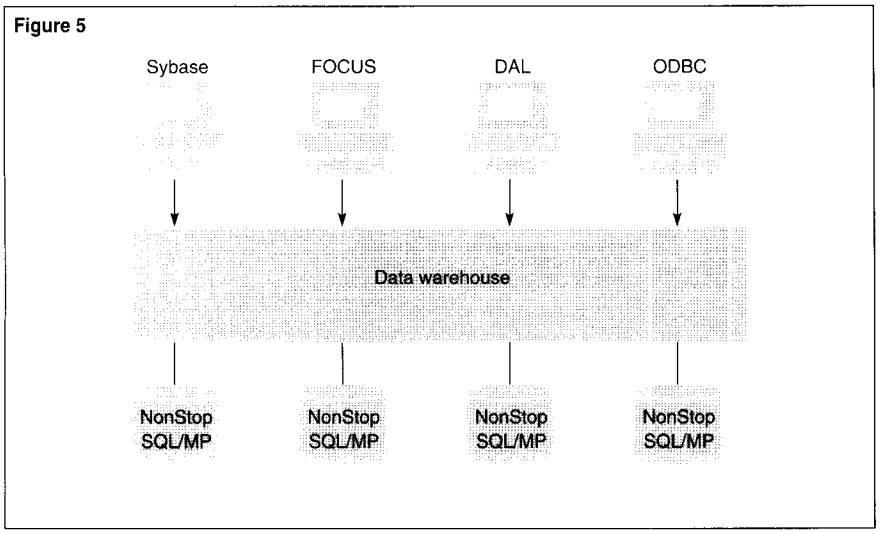


Figure 5.
An example of a data warehouse being accessed by various applications.

In practice, capacity planning requires ongoing attention. After the current planning effort is completed, another planning effort is undertaken to refine and enhance the previously implemented changes. This cycle continues for the life of the system.

Client/Server Environment

In a client/server environment, workstations and servers are interconnected in a network. Both client and server software can reside on workstations or on other systems. Typically, client applications reside on workstations, use graphical user interfaces (GUIs), and query servers to retrieve data. A server manages data for the network of clients that access it. Because of the large databases typically used in a decision support system, employing a client/server solution can accelerate application development and simplify the management of the database server.

In the case-study system, the users employed a client/server architecture to isolate database activity (queries) from application activity. The data warehouse resides on multiple platforms; the Tandem database server is the central server. The client/server environment houses a wide variety of applications that access the data warehouse. Figure 5 shows an example of a data warehouse being accessed by various client applications.

By combining the data warehouse architecture and the client/server environment, one can access large databases and develop applications in a shorter time than was ever thought possible. The rapid advances made recently in client/server application development actually make it feasible to employ a dedicated decision support system. As with any information system, one must carefully consider the design of the overall decision support system (both clients and servers).

Decision Support Tools

The word tools has a many-faceted meaning in a DSS environment. Some tools, such as those supplied by Prism Technologies, Inc., manage the data warehouse and the transformation of data into the warehouse. Other tools facilitate access to the data. These tools can be either host-based or workstation-based (in a client/server architecture). One can use host-based tools such as FOCUS when ease of implementation is a requirement. One should use a client tool if one intends to distribute the processing, and potentially the data, to a workstation environment. Many effective client tools can access a Tandem database server using Tandem open connectivity products such as the NonStop ODBC Server, the Dynamic Data Exchange (DDE) Gateway, or the Data Access Language (DAL) Server. (One can easily implement a client/server-based DSS environment by using client tools.)

Selecting user tools can be complicated when one expects many end-user communities to have access to the data warehouse. The users in the case study found that no single tool could satisfy the diverse needs of the many end-user groups in their organization. After gathering information from the various groups, the DSS designers derived a common set of similar requirements. They could then select three or four tools that would likely satisfy all of the important user requirements.

To bring the data to the end users, the DSS designers had to gather end-user requirements, select the suite of tools that would satisfy those requirements, and create pilot applications with the selected tools.

Gathering Requirements

The first step in gathering requirements is to document them. The DSS designers in the case study used a standard form to document requirements, which simplified the process. They held many meetings with end-user groups. The discussions revealed that the end users knew what they needed from the data warehouse but not what they wanted in an application. (The initial list included more than 50 tools.) After compiling application-specific and data warehouse requirements into an overall document, the DSS designers published the results, which were reviewed by the end-user communities.

Based on the requirements, the DSS designers created an evaluation matrix and graded all the tools on the list. Many tools were dropped from the list because of low scores, which indicated missing features or lack of support for the intended environment. The remaining tools underwent a thorough hands-on evaluation; they were also used to generate prototypes of the DSS application.

Prototyping

The prototyping of a test application for each tool lasted about a month. The prototypes were then evaluated and shown to the end-user community. The end users had a lab where they tested the tools and provided feedback.

The DSS designers used the end-user feedback to evaluate and rate the tools and, finally, to decide on a final suite of tools. The next task was to distribute and support the tools. Fortunately, the DSS designers had begun preparing a plan for supporting the end users before the tools had been selected.

Support for Client/Server Applications

Many tools can access a Tandem database server as soon as they are installed, using, for example, the NonStop ODBC Server. The positive impact of simply installing a suite of tools cannot be underestimated. In the case study, end users could access the data warehouse as soon as the system was made available to them.

However, the quick access and ease of use of these tools, which give end users the power to work with the data warehouse almost immediately, can affect the performance of DSS applications. The client/server architecture leads to unpredictable performance because it is hard to control the load on the server at any given time. When testing a client/server application, one needs to consider issues such as the GUI behavior, the speed of execution and data access, the application-specific logic, and the accuracy of the returned data.

Testing client/server applications is difficult because they are event-driven, run across a LAN, may access one or many database servers, use common objects that may change frequently, and are completely under end-user control. Interprocess and interapplication communications, especially between multiple vendors' products, make the performance of the applications even more unpredictable.

Ultimately, client/server testing requires that one find out if the server returns the results expected by the client. Tandem provides many debugging tools with its client/server products. For example, the NonStop ODBC Server comes with the ODBC Administrator, Tandem Debug, ODBC Test, and ODBC Tool to support ODBC development. The Tandem DDE Gateway allows the tester to trap unsupported messages and provides a Browser and a Bridge utility to support client/server development.

By providing debugging and development-support tools, Tandem allows DSS designers to focus on the specifics of their applications; they can concentrate on evaluating how well the selected tools match up against the requirements provided by the end users.

It is important to start the tools selection and analysis process early. Extensive evaluation and testing of a tool, regardless of the environment in which it is running, is imperative because of the impact it can have both on the end-user community and the data warehouse environment. To develop a productive suite of tools, one must understand end-user requirements and scrutinize the tools before they are implemented.

Conclusion

Decision support systems are becoming an integral part of many companies' strategies for using historical data as a competitive tool. Creating a dedicated environment for decision support makes it easier to address the implementation and support issues discussed in this article. Tandem's massively parallel and open connectivity products make it feasible to build a dedicated data warehouse environment. However, one must still carefully plan the implementation of a decision support system.

One must design a plan for loading, replacing, and maintaining the data in the data warehouse. It is equally important to select user tools that will access the data warehouse. Although tools selection appears to be the simplest of these tasks, it can require as many people and take as long as designing and loading the data warehouse.

Client/server tools are constantly changing and adding new features. To continue to use the data warehouse effectively, one should evaluate these tools frequently. By having appropriate requirements in place, one can greatly simplify the selection of new client tools.

References

Glasstone, R. 1994. Issues in DSS Database Design. *Tandem Systems Review*. Vol. 10, No. 4. Tandem Computers Incorporated. Part no. 104402.

Ho, F., Jain, R., and Troisi, J. 1994. An Overview of NonStop SQL/MP. *Tandem Systems Review*. Vol. 10, No. 3. Tandem Computers Incorporated. Part no. 104400.

Inmon, W. 1992. Snapshots in the Data Warehouse. *Tech Topic*. Vol. 1, No. 4. Prism Solutions, Inc.

Inmon, W. 1993a. Capacity Planning for the Data Warehouse. *Tech Topic*. Vol. 1, No. 10. Prism Solutions, Inc.

Inmon, W. 1993b. Loading Data into the Warehouse. *Tech Topic*. Vol. 1, No. 11. Prism Solutions, Inc.

Mahbod, H. and Slutz, D. 1994. NonStop ODBC Server. *Tandem Systems Review*. Vol. 10, No. 3. Tandem Computers Incorporated. Part no. 104400.

Troisi, J. 1994. NonStop Availability and Database Configuration Operations. *Tandem Systems Review*. Vol. 10, No. 3. Tandem Computers Incorporated. Part no. 104400.

Zeller, H. 1994. A New Hash-Based Join Algorithm in NonStop SQL/MP. *Tandem Systems Review*. Vol. 10, No. 3. Tandem Computers Incorporated. Part no. 104400.

Wayne Pearson is a Professional Services project manager who has supported DSS implementation and other major projects for Tandem users. He joined Tandem in 1990.

Issues in DSS Database Design

Many, if not most, data-center managers would prefer to have their decision support systems (DSS) use the same database and run on the same hardware platforms as their online transaction processing (OLTP) workloads. In support of this, recent years have seen dramatic improvements in performance when OLTP workloads run concurrently with batch and DSS workloads. However, for high-volume DSS workloads, most experts still find that a separate database specifically designed for DSS offers better performance and is easier to use than a combined OLTP/DSS database.

This article describes important differences between OLTP and DSS databases and focuses on three topics in DSS database design: designing dimensional DSS databases, creating views to simplify query generation for end users, and partitioning tables to reduce contention and optimize parallel processing. Dimensional databases provide high performance on DSS queries, are simple in design, and are easy for nontechnical end users to understand and use.

Differences Between OLTP and DSS

OLTP databases manage information used for the day-to-day running of a business. In contrast, DSS databases include data from completed transactions and are typically used to analyze trends and patterns that may affect the business over an extended period of time. These functional differences are reflected in a number of further differences, each of which has implications for database design.

DSS is Information Oriented; OLTP is Transaction Oriented

In OLTP, user transactions constantly change the contents of the database. Most of these transactions only involve reading, updating, inserting, or deleting a small number of rows. As a result, it is important to optimize an OLTP database for frequent updates and rapid retrieval of random rows. This is primarily achieved by following the rules of database normalization and usually produces a database with a large number of interrelated tables and many alternate keys.

DSS is very different. DSS end users rarely update the database and they frequently generate queries that require a large number of rows to be processed. A DSS analyst is typically interested in group data summarizing a large number of instances, rather than information about individual cases. For example, a DSS analyst might be interested in

- Weekly sales of a given product.
- Comparative sales of similar products from different manufacturers.
- All customers in a given geographical area.
- All customers with a specific spending profile.
- The effectiveness of an advertising campaign.

To provide acceptable response times, a DSS database needs to be optimized for reading large amounts of data. This requirement can best be met by designing databases with a few very large tables for storing data from business operations and by supporting each data table with several smaller *dimension* tables. The dimension tables are used to provide descriptions and map data for the larger tables, as described later, in the section "Designing DSS Databases Through Dimensional Database Modeling."

Because of the need for grouped data in DSS, it is often desirable for a DSS database to contain both detail tables, consisting of data on individual transactions, and separate summary tables containing grouped data derived from the detail tables. Summary tables are primarily for the benefit of specific user groups and individuals who regularly generate queries for the same types of summarized data.

If a database has detail and summary tables for the same data, there will be considerable redundancy between tables. This is usually not acceptable in OLTP, because it makes updates and maintaining database integrity more complicated. In a DSS database, where existing data is rarely modified, redundancy is not as great a problem, and the performance benefits of a summary table are likely to outweigh its disadvantages. For example, if analysts regularly look at weekly sales data, defining a summary table for weekly sales may eliminate the need to scan a tremendous number of rows containing data for daily sales.

Users Generate DSS Queries; Programmers Code OLTP Transactions

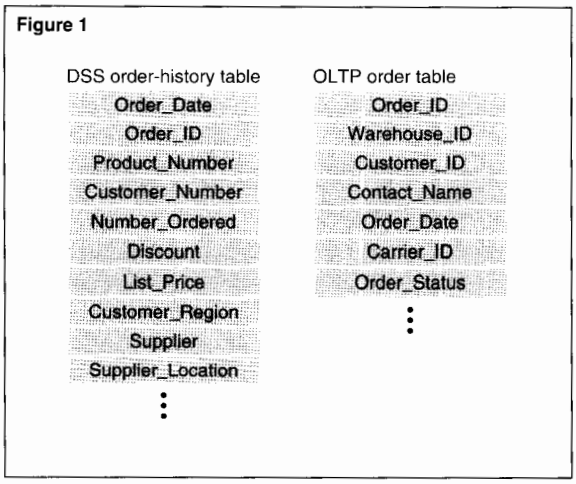
The structure of an OLTP database is often highly complex, but the complexity is dealt with by the programmers of an application and is invisible to the end user. Once implemented, the transaction types defined in an OLTP application are used repeatedly and seldom changed. When new transactions are added or existing transactions modified, the changes are implemented by programmers with technical knowledge of the database and the application.

In DSS, queries are generated by the end user, not hard coded in an application. The end user may enter complete queries online or generate them through the use of a query generation tool. DSS queries are often ad hoc and may be very complex. The precise form of a query is likely to change from case to case, and some queries are rarely, if ever, repeated.[1]

Although DSS queries may be very complex, the users who formulate them rarely have the opportunity that OLTP application programmers do to optimize performance or test the accuracy of results. This makes it highly desirable to keep the design of a DSS database as simple as possible.

[1]This is not to say that there are not significant patterns of usage. Overall, users will want to access tables and have data grouped or summarized in some ways more than others. Advance knowledge of such patterns typically plays an important part in the process of database design.

Figure 1.
Comparison of columns in a DSS order-history table with columns in an OLTP order table.



DSS Databases Contain Historical Data

An OLTP database only needs to store the information necessary for completing current transactions. Much of this information, such as the price of a product or a customer's address, is subject to change.

DSS databases have a temporal dimension and may include transactions that were completed months or even years ago. Historical data, such as the price of a product on October 5, 1993, is generally not subject to change. One consequence of this is that a DSS history table often has columns that would not be found in the corresponding OLTP table. For example, Figure 1 shows columns in a DSS order-history table compared with columns in an OLTP order table.

In Figure 1, the DSS order-history table contains columns for List_Price, Customer_Region, Supplier, and Supplier_Location that are not included in the OLTP order table. In an OLTP database, each of these columns would be in a reference table, from which it could be accessed to provide a current value for use in transactions. The columns would be redundant in the OLTP order table and, because they are subject to updates, could eventually lead to inconsistencies between the order table and one of the reference tables. The columns can safely be included in the DSS order-history table, because they contain historical values associated with a specific time period and are not subject to change.

A second consequence of storing historical data is that copying operational data, such as the bin a part is stored in or special delivery instructions for a particular order, from an OLTP database to a DSS database would be a waste of disk space. In Figure 1, for example, Carrier_ID and Order_Status are operational information that is not likely to be of interest to a DSS analyst.

DSS Databases are Batch Updated

In OLTP, a large number of transactions may be executed in the course of a single day, and the database is updated in real time as transactions are completed.

In DSS, data is usually added to and deleted from tables in batches at fixed time intervals, such as nightly or weekly. During the day, data that is already in the database is unlikely to be modified. This means that there is little risk of reading inconsistent data during the day and that queries can usually be run with browse access.

The columns used to partition a table and the way partitions are defined can have a significant effect on the addition and deletion of batch data. If rapid addition and deletion of data is a high priority for a DSS history table, it may influence the choice of a first key column and partitioning scheme for a table.

DSS Data Often Comes From Multiple Sources

Within a single organization, many OLTP and batch databases may be distributed over different hardware and software platforms. This usually has little effect on an OLTP application or its users, since each database is likely to be designed for a different application or for a specific department within the company. OLTP applications rarely combine data from two or more databases. If an application does use combined data, it is the responsibility of the application to handle any complexities that arise and make the functionality transparent to the end user.

A DSS query often requires data from multiple databases, and this can be a source of difficulties for an end user or DSS application. For example, suppose an organization has data in two separate databases on computer systems in the United States and the United Kingdom. The databases are on completely different hardware platforms, the U.K. database on a UNIX system and the U.S. database on a Tandem™ NonStop™ Kernel system. The databases have similar, but not identical, data structures, and each has its own product table. The left side of Figure 2 shows the two product tables. As illustrated, most, but not all, columns of the two tables contain the same type of information. Columns with the same type of information may have different names and contents (for example, dollars versus pounds), and different data-type specifications.

If a DSS user wanted to carry out an analysis combining data from both tables, for example, total U.S. and U.K. sales, by product, either the user or the DSS application would have to deal with the differences between data types in the two tables and the conversion of currencies to a common base. If this is something of a problem with unrelated databases in two countries, consider the implications of multiple unrelated databases belonging to a large multinational corporation. A far simpler approach is to define a single DSS product table, with its own column names and specification of data types, as shown on the right side of Figure 2, and load it with data from the U.S. and U.K. product tables. In this case, the application software used to load the DSS database carries out the conversions necessary in going from the two OLTP product tables to the DSS product table, and DSS users only need to generate queries against the composite DSS table.

Figure 2

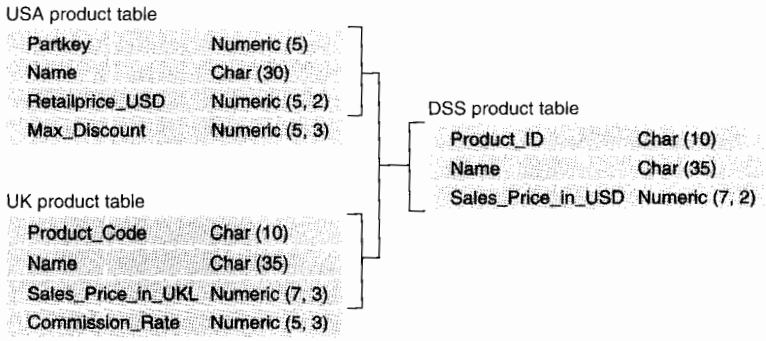


Figure 2.
Comparison of columns in DSS and OLTP product tables.

DSS and OLTP Have Different Performance Requirements

OLTP applications need to handle tens, or possibly hundreds, of discrete transactions per second. Typically each transaction performs from 5 to 20 random reads and writes. OLTP response time requirements are usually on the order of seconds or less per transaction.

In DSS, users execute queries, rather than transactions, and a single DSS query is likely to read thousands, and even millions, of records from disk. DSS throughput is measured in queries per hour or per day; acceptable response times may be measured in minutes or hours.

Figure 3

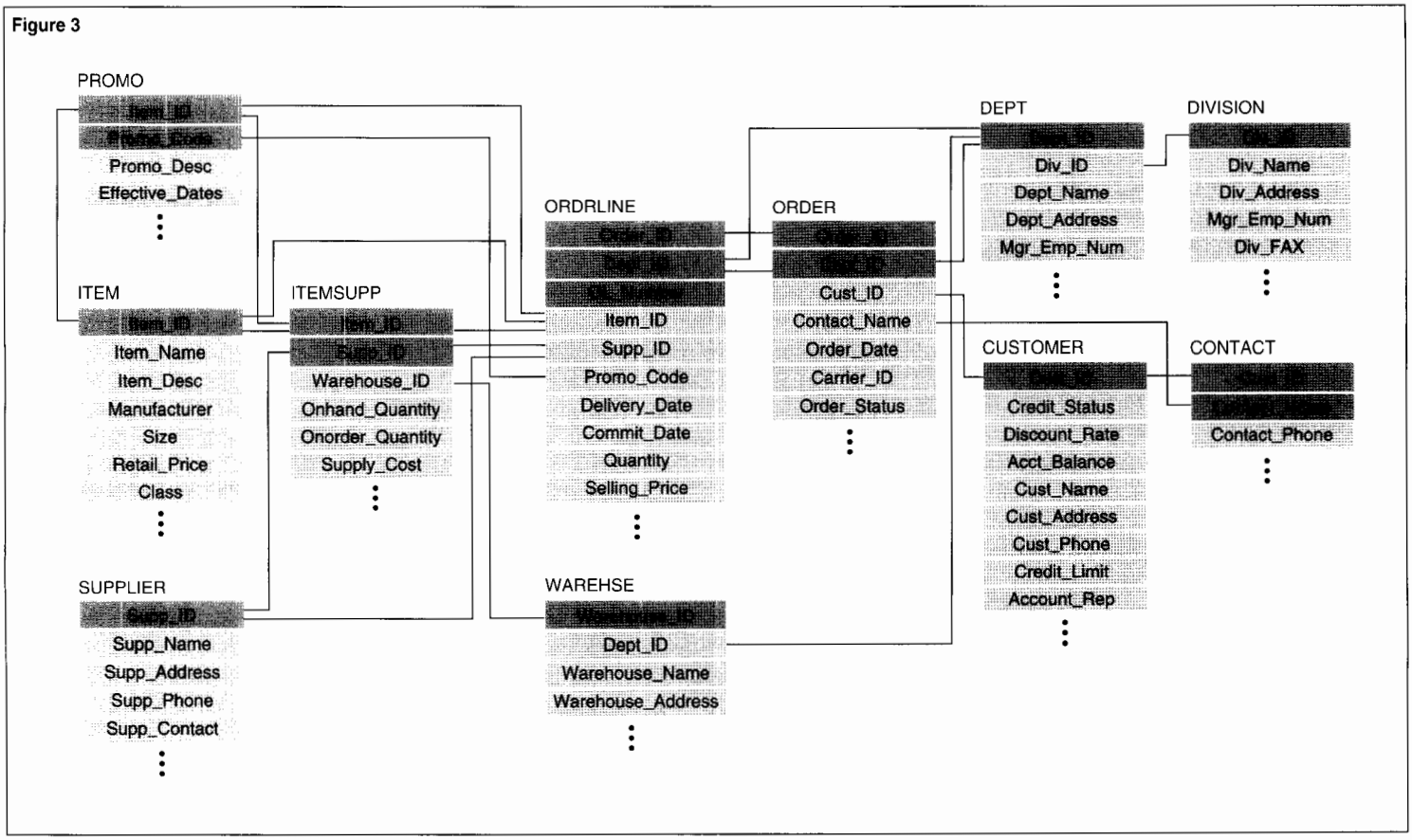


Figure 3.
Tables and columns in a simplified OLTP order-entry database.

In OLTP, random I/O is very efficient for accessing data, since only a small number of rows are processed within a single transaction. In DSS, random I/O on queries that require large numbers of rows to be processed would result in unacceptably slow response times. For effective processing on DSS queries, a database must be designed to maximize sequential I/O. The following section describes an approach that optimizes performance when large amounts of data must be read and provides the end user with an easily accessible database structure.

Designing DSS Databases Through Dimensional Database Modeling

Dimensional database modeling is an approach to database design that is especially suitable for DSS. A dimensional database schema, often called a star schema, has few tables, and the tables stand in simple relationships to one another. For the end user, the database is easy to understand and easy to use. In addition, a dimensional database lends itself to sequential I/O, which is necessary for achieving optimal performance on DSS queries. As a further benefit, several new performance features of NonStop SQL/MP, including cross products, hash joins, and hash groupings, are particularly effective with a dimensional database.[2]

[2]For an overview of new features in NonStop SQL/MP, see Ho, et al., 1994.

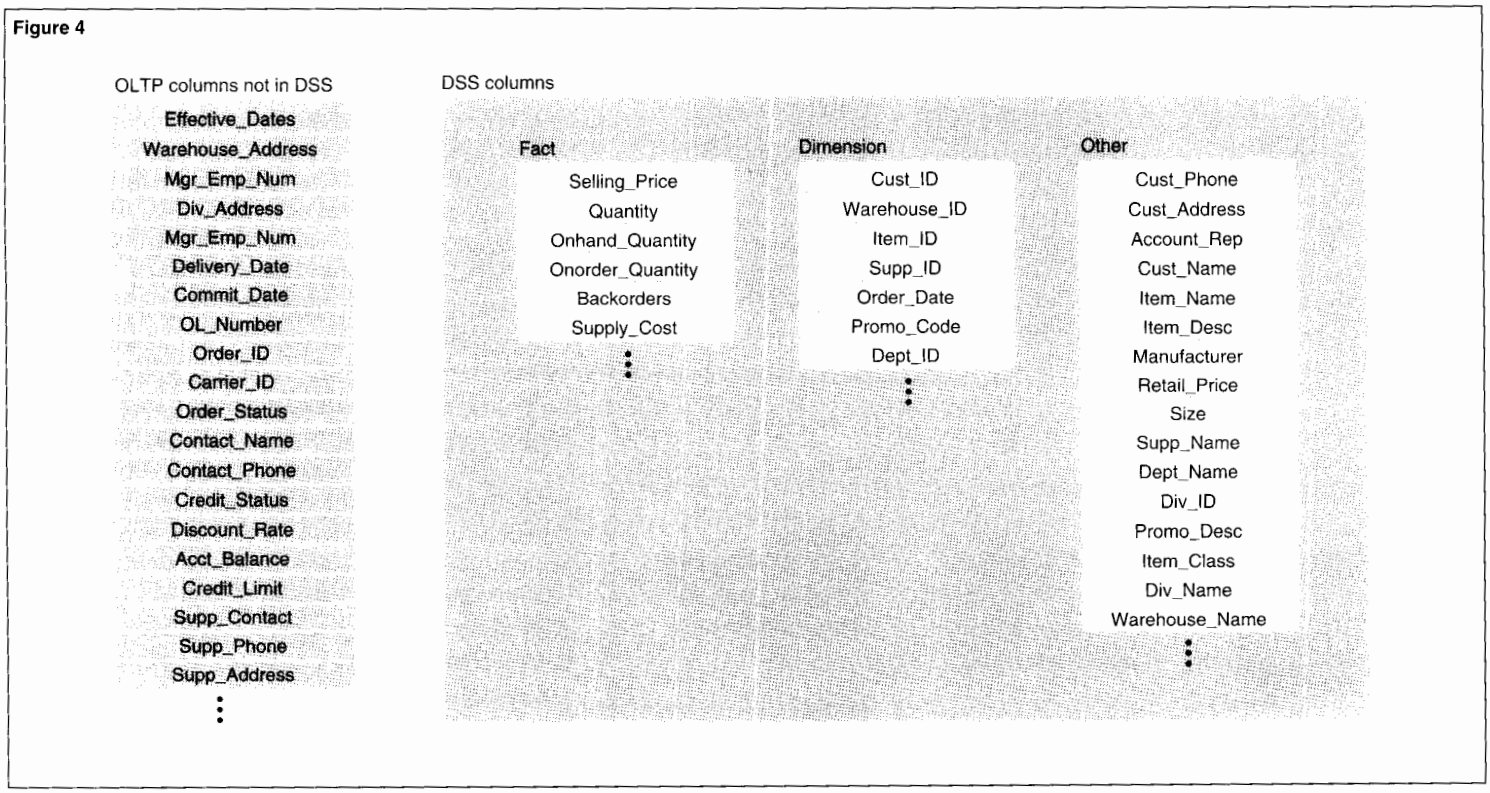


Figure 4.

OLTP columns from the database of Figure 3 that are likely to be used in DSS.

In dimensional database modeling, there are two primary types of tables, fact tables and dimension tables. Fact tables contain the numerical or transaction data that is to be analyzed. Dimension tables correspond to primary-key columns in fact tables and typically represent basic elements in the functioning of a business, such as products, stores, and customers.[3]

The data in a dimensional DSS database comes from one or more OLTP databases, but not all data necessary for OLTP is useful for DSS. For example, as discussed earlier, strictly operational data in an OLTP database should not be carried over to a DSS database. Figures 3 and 4 illustrate the relationship between the tables and columns in a DSS database and an OLTP database. Figure 3 shows tables and columns in a simplified order-entry OLTP database and the relationships between tables. In Figure 4, columns in the OLTP database of Figure 3 are divided into columns that would be carried over into DSS and those that would not. Columns carried over to DSS are further divided into fact columns, dimension columns, and reference columns. Fact columns contain numerical or transaction data. This type of data is usually aggregated in DSS queries. Dimension columns are primary-key columns in fact tables and dimension tables. They usually correspond to the GROUP BY columns in a query. Reference columns are columns in dimension tables and typically provide descriptive data.

[3]Although all dimension tables correspond to primary-key columns, there can be columns in the primary key that do not have corresponding dimension tables.

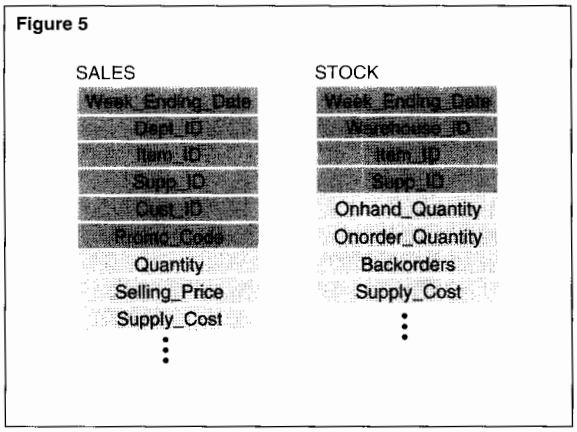


Figure 5.
Fact tables based on columns from the OLTP database of Figure 3. The fact tables contain summary data by week, rather than individual transactions. Week_Ending_Date is derived from Order_Date in the OLTP ORDER table.

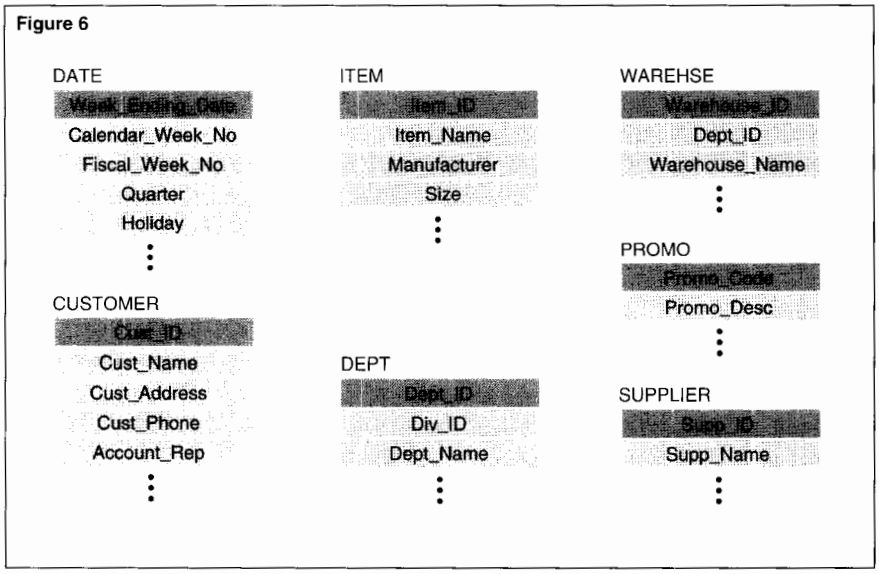


Figure 6.
Dimension tables corresponding to primary-key columns in the fact tables of Figure 5.

Fact Tables

Fact tables store an organization's business data. They can be very large, often tens or even hundreds of gigabytes in size. A DSS fact table is usually different from an OLTP table in two regards: it stores historical rather than current data and it condenses, and often summarizes, OLTP data. The data may be condensed by omitting operational or individual data, such as customer names and numbers, from a DSS table. Summarized data is presented in summary tables containing, for example, daily sales or sales by department, rather than individual sales transactions.

Fact tables have multicolumn primary keys. The leading columns, and sometimes all the columns, in the primary key of a fact table correspond to dimension tables. A dimensional DSS database is likely to contain several fact tables representing different aspects of a business, such as sales, shipping, and inventory. The database may also have detail and summary fact tables for the same data, as in the case of a sales detail table, and separate summary tables for daily and monthly sales. Detail tables and their summary tables will share many of the same primary-key columns and dimension tables; they are also likely to share key columns and dimension tables with other fact tables. Large fact tables rarely have alternate indexes, because sequential access via an index results in exessive random reads of the base table. Figure 5 shows the columns and primary keys of two fact tables derived from the OLTP database in Figure 3.

Dimension Tables

Dimension tables are relatively small tables, ranging in size from tens of rows up to a few hundred-thousand rows. Figure 6 shows dimension tables for the primary key columns in the fact tables of Figure 5. Dimension tables are primarily used to provide descriptive or mapping data in joins with fact tables. For example, the ITEM table in Figure 6 contains an Item_ID column and several reference columns with descriptive data. If the ITEM table is joined with the SALES table of Figure 5, the reference columns can be used to provide descriptive information about Item_IDs retrieved from the SALES table: Item1535 is an infant car seat from manufacturer X. In this case, the SALES table is the outer table in the join and the ITEM table the inner table. As the outer table in a join with the SALES table, the ITEM table can map descriptive information to Item_IDs and determine what is to be retrieved from the SALES table: data on infant car seats is to be found under item numbers 1293, 1452, 1535, and 1986.

Schematic Structure of a Dimensional Database

In an OLTP database, including the simplified database of Figure 3, one table may have a complex pattern of relationships with a number of other tables. In a dimensional DSS database, a fact table and its dimension tables are connected in a simple pattern that is referred to as a star schema. Figure 7 illustrates the star schema for the SALES table of Figure 5 and its dimension tables.

Defining Views To Simplify the Database for End Users

Although the schematic design of a dimensional DSS database appears far simpler than the typical OLTP design, this does not always make it easy for the DSS end user to generate efficient queries. In many cases, satisfactory performance on DSS queries requires joins that an end user would be unlikely to specify without detailed knowledge of the database. For example, the first and second key columns of the SALES table in Figure 5 are Week_Ending_Date and Dept_ID. Suppose a DSS analyst wants sales data for departments 23, 24, 36, and 42 during the weeks between May 1, 1994 and May 31, 1994. The simplest query for obtaining this data would be

```
SELECT <column names>
FROM SALES
WHERE
   Week_Ending_Date BETWEEN
      "1994-05-01" AND "1994-05-31" AND
      Dept_ID in (23,24,36,42)
GROUP BY ...
ORDER BY ...
```

However, the NonStop SQL/MP optimizer treats the date range in the query as a set of continuous values and generates an execution plan that reads all rows belonging to the date range from disk. Each of these rows is then read for the specified departments. A far more efficient query joins the SALES table with the DATE and DEPT tables and allows the new cross-products feature of NonStop SQL/MP to be used:

```
SELECT <column names>
FROM SALES
WHERE
   SALES.Week_Ending_Date =
      DATE.Week_Ending_Date AND
      DATE.Week_Ending_Date BETWEEN
      "1994-05-01" AND "1994-05-31" AND
      SALES.Dept_ID =
      DEPT.Dept_ID AND
      DEPT.Dept_ID IN (23, 24, 36, 42)
GROUP BY ...
ORDER BY ...
```

Figure 7

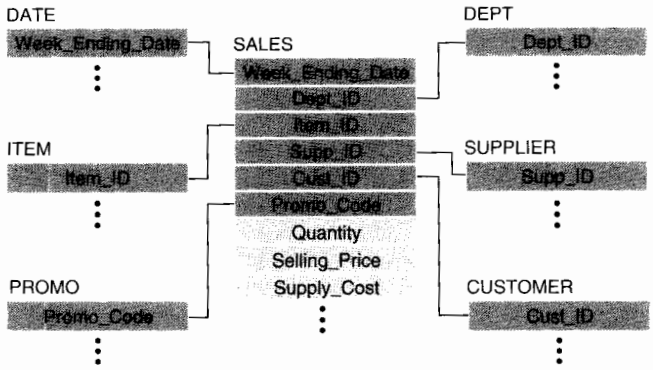


Figure 7.
Fact and dimension tables forming a star schema.

Table 1.
Join on qualifying Week-Ending-Dates from the DATE table and qualifying Dept_IDs from the DEPT table.

Week-Ending-Date	Dept_ID
1994-05-07	23
1994-05-07	24
1994-05-07	36
1994-05-07	42
1994-05-14	23
1994-05-14	24
1994-05-14	36
1994-05-14	42
1994-05-21	23
1994-05-21	24
1994-05-21	36
1994-05-21	42
1994-05-28	23
1994-05-28	24
1994-05-28	36
1994-05-28	42

Given this query, the NonStop SQL/MP cross-product feature first joins qualifying rows from the DATE and DEPT tables. There are four qualifying Week_Ending_Dates in the DATE table and four qualifying Dept_IDs in the DEPT table. Table 1 shows the join result.

The join result consists of 16 Week_Ending_Date/Dept_ID pairs. NonStop SQL/MP uses these pairs as key values for probing into the SALES table, so that it only has to read rows for sales made in the specified departments during the given date range and can skip rows for all other dates and departments. On a query involving large quantities of data, this can save many minutes, if not hours, of processing time.

Although the preceding query is optimal for NonStop SQL/MP, a DSS end user without specific knowledge of the database would be unlikely to request joins between the SALES table and the DATE and DEPT dimension tables. A solution to the problem, in this case, would be to define a view that joins the SALES table with the DATE and DEPT tables in advance, and then to present only the view, rather than the SALES table, to the end user.[4] Figure 8 illustrates such a view for the SALES table of Figure 5.

Now, if the DSS analyst of the example is presented with SALESVW instead of the SALES table, it is only necessary to generate the straightforward query

```
SELECT <column names>
FROM SALESVW
WHERE
   SALESVW.Week_Ending_Date BETWEEN
   "1994-05-01" AND "1994-05-29" AND
   SALESVW.Dept_ID IN (23, 24, 36, 42)
GROUP BY ...
ORDER BY ...
```

[4]Although the view is defined in advance, it is not materialized by NonStop SQL/MP until the query is executed.

The underlying form of this query is identical to the previous query with explicit joins between the SALES table and the DATE and DEPT tables.

In general, if the selection restrictions on a query involve multiple column values from both the first and second key columns of a fact table, joins between the fact table and the corresponding dimension tables will greatly improve performance. In all such cases, if the end user sees a view that joins the fact table with its dimension tables in advance, as in Figure 8, it will be much easier to generate an efficient query.

Figure 8

```
CREATE VIEW SALESVW
  AS SELECT  DATE.Week_Ending_Date,
             DEPT.Dept_ID,
             <Other column names from DATE>
             <Other column names from DEPT>
             <Other column names from SALES>
  FROM SALES, DATE, DEPT
    WHERE
        SALES.Week_Ending_Date=DATE.Week_Ending_Date AND
        SALES.Dept_ID=DEPT.Dept_ID;
```

Figure 8.
A view combining the SALES table of Figure 5 with the dimension tables corresponding to its first two key columns.

Partitioning Strategies for Tables in a Dimensional DSS Database

The way a table is partitioned will have important consequences for parallel processing within a query, if large amounts of data need to be scanned, and for parallel processing between queries, when multiple queries contend for the same data. It also determines how efficiently a table can be updated, if data is added and deleted in batches at fixed time intervals.

The following subsections describe four partitioning strategies. The first three strategies are for partitioning a DSS fact table. In each of these cases, it is assumed that the primary key of the fact table contains a Date column and a Department column, that most queries select data on the basis of date, and that new data is added and old data deleted on a weekly basis. In the first strategy, the Date column is the first key column and determines all partitioning; data is added and deleted by adding and deleting partitions. The second strategy makes Department the first key column and uses it to determine partitioning; new data is added to the end of each department in a partition and deleted from the beginning of each department in a partition. In the third strategy, a special Partition column, defined only for purposes of partitioning, is made the first key column. Dates and departments are striped across many partitions; the addition and deletion of data depends on the implementation.

Figure 9

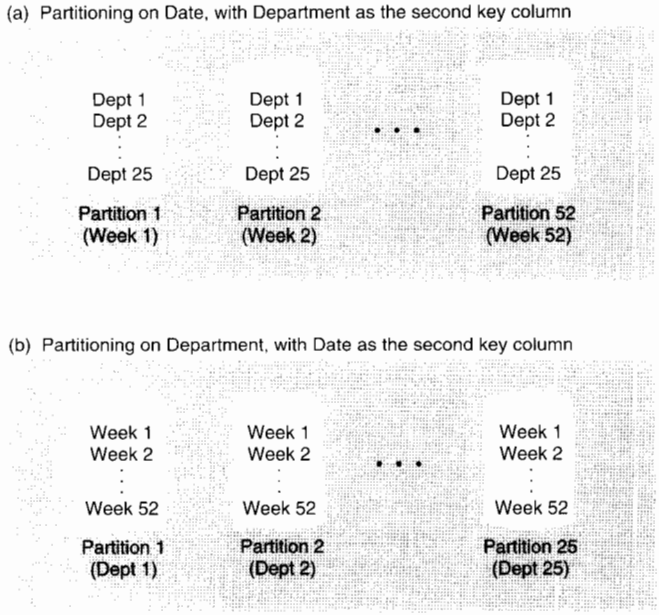


Figure 9.

Comparison of partitioning on Date with partitioning on Department for storing weekly data.

The fourth partitioning strategy is for partitioning small dimension tables. In this strategy, a special Partition column is used to partition a dimension table across CPUs in order to optimize the use of resources when the dimension table is joined with a large fact table.

It should be noted that the discussion of partitioning strategies in this section is meant to be suggestive, rather than comprehensive. Each of the four strategies described can be varied and extended in many ways.

Partitioning on the Date Column

Making the Date column the first key column is likely to be the simplest and most effective strategy when the majority of queries select data according to date and contention for the same date is likely to be low. Partitioning on Date is especially efficient when data is added and deleted at fixed time intervals. For example, if data is added and deleted weekly, a common approach is to define partitions in terms of date ranges for a week. Each week a new partition is added at the end of the table and an old partition deleted from the beginning of the table. Data is added using the NonStop SQL/MP LOAD utility. This is the fastest method of adding new data to a table, but it can only be used to load data into an empty table or partition. Data is deleted from old partitions using the NonStop SQL/MP DROP PARTITION statement. Dropping an entire partition takes only seconds. In contrast, removing the rows of a partition through delete operations can take many hours.

If a large amount of data is added at weekly intervals or there is likely to be a high level of contention among queries for access to newly added data, rather than defining only one new partition based on date, it may be desirable to add several new partitions. In this case, the LOAD utility can be used to add data to all of the new partitions in parallel.

Partitioning on the Department Column

If partitioning on Date is likely to result in excessive contention for the same partitions, an alternative strategy is to promote what would have been the second key column, Department, to first key column, and make the Date column the second key column. This has the effect of distributing weekly data across department partitions. Figure 9 compares partitioning on Date with partitioning on Department.

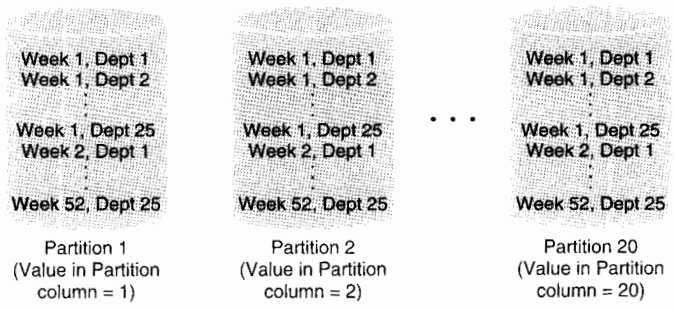
In Figure 9a, the Date column is used to define weekly partitions. All queries requiring data from the same week contend for access to the same partition. In Figure 9b, weekly data is distributed across 25 partitions. In this case, if multiple users need data from the same week, the likelihood of contention is considerably reduced.

If weekly data is added to partitions based on Department, the LOAD utility cannot be used, since it requires empty partitions and, presumably, all Department partitions will contain some data after the first week or so. Although it is not likely to be as fast as the LOAD utility, to quickly insert a large amount of weekly data into Department partitions, a site can develop software that receives the data, divides it into separate streams on the basis of partition boundaries, and launches the streams as parallel processes. Each stream must then sort its data by the primary key and sequentially insert it into the corresponding partition. A disadvantage of partitioning on Department is that weekly data cannot be deleted with the DROP PARTITION statement, and must be removed through far slower row deletions in each Department.

Partitioning on a Partition Column

If partitioning on the Date column is likely to cause contention and partitioning on the Department column is also likely to cause contention, a third strategy is available. Under this strategy, a special Partition column is defined and used only for purposes of partitioning. In addition, a separate partition table is created and application software developed for distributing data across partitions. Figure 10 illustrates partitioning based on a Partition column. This strategy maximizes parallel processing, since the distribution of data forces all queries to retrieve data from all partitions.

Figure 10

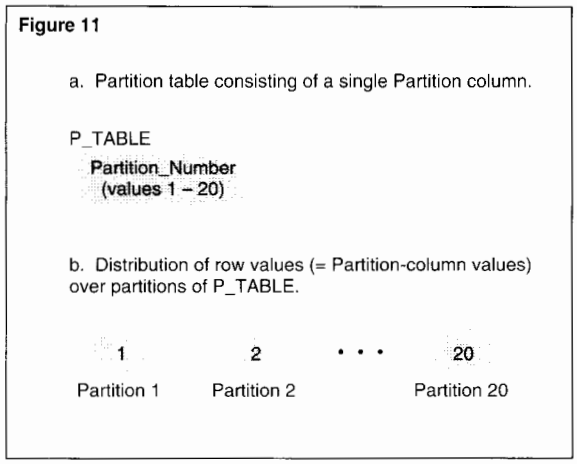


In Figure 10, a Partition column has been used to define 20 partitions. User-developed software has assigned rows to each partition in such a way that for each week, rows for each department are distributed across all 20 partitions. Distributing rows across partitions by both Date and Department should keep contention on these columns to a minimum.

Defining a Partition Column. To implement the third partitioning strategy, when a DSS fact table is created, a Partition column is defined and made the first key column. It is used to define as many partitions as necessary for the amount of data expected in the table. Partitions are defined on sequential values in the Partition column, as shown in Figure 10.

Figure 10.
Partitioning on a Partition column, with Partition the first key column, Date the second, and Department the third. Rows are assigned to partitions by user-developed software.

Figure 11.
Partition table (P_TABLE) for use with the partitioned fact table in Figure 10.



Developing Software to Assign Partition Values to Rows. To distribute weekly data across partitions of the fact table, user-developed software must read the data, assign each row a Partition-column value, and insert the data into the table. There are many possible ways of making row assignments. To achieve the distribution of rows in Figure 10, a user-developed application could obtain a count of the number of rows in each department, divide the count by 20, and distribute the rows for each department equally across all 20 partitions. For example, with 1,000 rows for Department 1 and 2,000 rows for Department 2, the application would need to assign 50 rows from Department 1 to each partition and 100 rows from Department 2 to each partition. To assign rows from Department 1 to partitions, the application could simply give the first 50 rows of Department 1 a Partition-column value of 1, the second 50 rows a Partition-column value of 2, and so on. In order to insert rows into partitions sequentially, the application would need to sort the data for each partition by primary key before inserting it.

When a Partition column is used to partition a fact table, the LOAD utility cannot be used to add data to partitions, and the same considerations apply to the insertion and deletion of data as in the case of partitioning on Department.

Defining a Partition Table. As discussed earlier, the leading columns in the primary key of a fact table correspond to dimension tables. In many queries, joins between a fact table and two or more of its dimension tables are necessary for efficient performance. If the first key column of a fact table is a Partition column, there needs to be a corresponding partition table that can play the role of a dimension table and be used in joins with the fact table. In keeping with this, the partition table also counts as a dimension table when a view such as the one in Figure 8 is defined to join a fact table with its main dimension tables.

The partition table consists of a single Partition column, which is used to define partitioning for the table. In some cases, the partition table is assigned exactly the same partition breaks as its fact table. For the example illustrated in Figure 10, this would mean creating a partition table with 20 partitions based on Partition-column values 1 through 20. In addition, the partition table would have to be populated with rows having Partition-column values of 1 through 20. Figure 11 shows a partition table, P_TABLE, that corresponds to the Partition column of Figure 10.

When two tables are joined, NonStop SQL/MP starts an executor server process (ESP) for each partition of the outer table. Each ESP joins its partition with the inner table. A partition table is always the outer table in a join with its fact table. If a fact table is divided into a large number of partitions per CPU, and its partition table is partitioned in the same way, a query that requires a join of the two tables will result in many ESPs per CPU. For example, if partitions are evenly divided over 10 CPUs and a fact table with 200 partitions is joined with a partition table containing 200 partitions, 20 ESPs will have to start and execute in each CPU.

To avoid generating a large number of ESPs on a single query, when a fact table is divided into many partitions per CPU, it is often useful to divide a partition table into a limited number of partitions, sometimes as little as one per CPU. For example, if 200 partitions of the fact table are divided over 10 CPUs, the partition table can be divided into 10 partitions, one in each CPU, as illustrated in Figure 12.

In Figure 12, each partition is defined to contain a range of 20 Partition-column values. These values must be inserted into the table before it is used in a join. When the partition table is joined with its fact table, an ESP in CPU 0 will join Partition 1 of the partition table with partitions 1 through 20 of the fact table. In parallel, an ESP for each of the other partition table partitions will join its partition with 20 consecutive partitions of the fact table. For optimal processing, the partitions of the partition table and the fact table should be aligned so that the ESP for each partition table partition only has to deal with fact table partitions in its own CPU. In the example, this would mean that fact table partitions 1 through 20 were placed in CPU 0, partitions 21-40 in CPU 1, and so on.

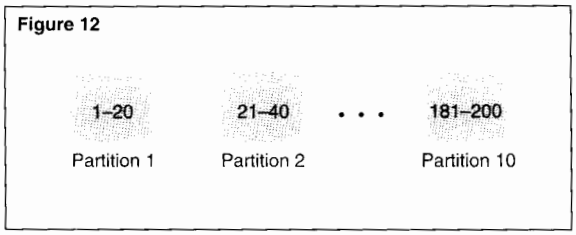


Figure 12.
Partition table with 10 partitions for use with a fact table of 200 partitions.

Using a Partition Column to Partition a Dimension Table Across CPUs

The preceding section showed how to optimize the use of system resources by limiting the number of ESPs used to join a partition table with a fact table containing a large number of partitions. A similar strategy can be used when the first key column of a fact table is represented by a small dimension table.[5] In this approach, a Partition column is made the primary key of the dimension table and user-developed software assigns a Partition-column value to rows of the dimension table.

[5]A partitioned table is considered small if none of its partitions contain more rows than will fit into a single 56-kilobyte read. Larger partitions may also be acceptable, depending on the number of I/Os necessary for reading them. In many cases, spending a few extra I/Os to read a partition may not be significant relative to the total amount of time needed for executing a large join.

Figure 13.
100 partitions of a fact table distributed across 10 CPUs.

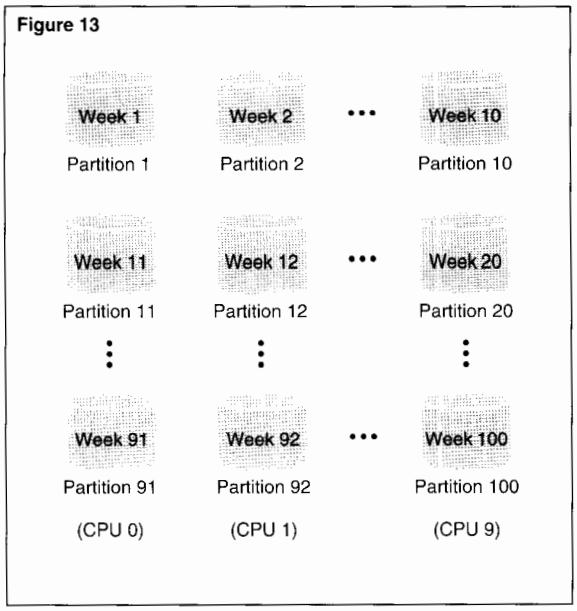
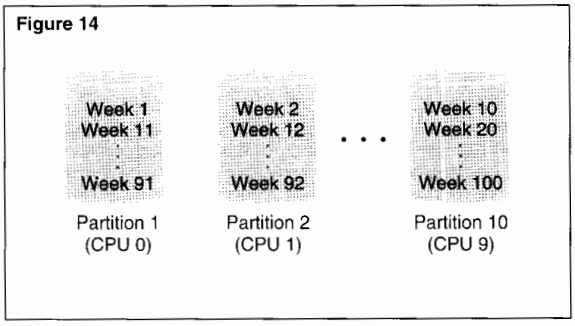


Figure 14.
Partitioning the DATE table across CPUs so that it is aligned with the partitions of the fact table in Figure 13.



The use of a Partition column to partition a dimension table can be illustrated with a large fact table partitioned on Date and a DATE dimension table. The fact table is divided into 100 partitions distributed across 10 CPUs, as illustrated in Figure 13.

In Figure 13, the fact table is partitioned so that consecutive dates are in different partitions and in separate CPUs. Since DSS queries frequently specify date ranges, separating consecutive dates in this way is likely to reduce contention for the same partition and balance processing across CPUs.

If the DATE table were partitioned in the same way as the fact table, joining the two tables would require 100 ESPs, with 10 ESPs in each CPU. This would result in a considerable expenditure of memory and system resources. To save resources, one can reduce the number of ESPs used for the join. Suppose the goal is to partition the DATE table so that there is one ESP per CPU, with each ESP only joining the DATE table partition in its CPU with the fact table partitions in its CPU. This requires dividing the DATE table into 10 partitions, one per CPU, and defining partitions so that the dates in each DATE table partition match the dates of the 10 fact table partitions in the same CPU. The necessary partitioning is illustrated in Figure 14.

Partitions can only be defined on consecutive values in a given key column. Thus, the Date column of the DATE table cannot be used to achieve the partitioning in Figure 14, since the individual partitions do not contain consecutive dates. To achieve the required partitioning, a Partition column needs to be defined as the primary key of the DATE table. In addition, user-developed software must read data intended for the DATE table and assign each row a Partition-column value based on the date. Weeks 1, 11, 21, and so on, up to 91, would be assigned a Partition-column value of 1. Weeks 2, 12, and so on, up to 92, would be given a Partition-column value of 2. Partition-column values for each of the remaining partitions would be assigned in the same way.

The Partition column is the only column in the primary key of the DATE table. Although an ESP must read an entire DATE table partition in order to execute a join on Date with the fact table, this takes very little time, since each DATE partition can be retrieved from disk with a single 56-kilobyte read operation and the disk process filters out dates not required for the join. Partitioning a DSS dimension table on a Partition column can reduce the number of ESPs necessary for joins and makes it possible to conserve memory resources, improve query startup times, and reduce the number of messages sent between ESPs for initiating queries and other purposes.

Conclusion

OLTP and DSS databases have different design requirements. For optimal performance, each type of processing should have its own database. Dimensional database modeling is an approach particularly suited to designing DSS databases that provide high performance and are easily accessible to an end user. Adding views to a dimensional DSS database can further simplify the appearance of the database to an end user and can make it easier to generate efficient queries.

The way a table is partitioned can have an important effect on query processing and the addition and deletion of data. In some cases, partitioning a DSS fact table on a specially defined Partition column can reduce contention for individual partitions and improve parallel processing performance. In a DSS dimension table, a Partition column can be used to reduce the number of ESPs that process a query. This may result in significant savings of memory and system resources.

References

Ho, F., Jain, R., and Troisi, J. 1994. An Overview of NonStop SQL/MP. *Tandem Systems Review*. Vol. 10, No. 3. Tandem Computers Incorporated. Part no. 104400.

Acknowledgments

Thanks to all of the reviewers of the article, who helped to keep me honest, and to the NonStop SQL/MP development team, which implemented the features that make DSS databases work. Thanks also to the Tandem users and prospective users whose projects over the past 18 months have helped me to understand the special requirements of DSS databases.

Ray Glasstone is an SQL performance specialist in the Tandem OLTP performance group. He joined Tandem in 1984 as a staff analyst in the United Kingdom, moving to the Cupertino Benchmark Center in 1988 and to his current position in 1993.

Late Binding and High Availability Compilation in NonStop SQL/MP

The Tandem™ NonStop™ SQL/MP relational database management system contains new *late binding* features that significantly reduce the number of SQL compilations needed when users develop and install programs. These features also reduce auto-recompilations when SQL statements are executed at run time.

Program compilations affect application performance and can cause temporary user outages. Reducing the frequency of compilations increases the availability of user applications. It also limits the time-consuming activities associated with compilation. Thus, the new features make it easier to develop and manage application programs, improving the productivity of programmers and database administrators (DBAs).

In particular, the late binding features include the following:

- New execution-time name resolution rules for SQL statements that make it easier to develop certain types of programs.
- Mechanisms that enable a program to tolerate Data Definition Language (DDL) operations without recompilation.
- Facilities to install programs without SQL-compiling them (thus retaining the SQL statements' existing execution plans) and without registering them in a catalog.

Another feature introduced in NonStop SQL/MP, referred to in this article as *high availability compilation*, complements the late binding features by making it possible to compile programs faster than in previous NonStop SQL releases. New compiler options shorten SQL compilation time and allow SQL statements to use their previously-compiled execution plans.

This article discusses the late binding and high availability compilation features. It briefly explains the way a typical application is managed in previous releases of NonStop SQL, then describes the new NonStop SQL/MP features in detail. The last section, User Scenarios, provides examples that show when and how the features may be useful.

The article assumes that readers are familiar with NonStop SQL. Those not interested in the detailed descriptions of the new features might want to read the first pages of the article (through the Summary of New Features section) and then skip to the User Scenarios section.

Managing Application Programs in Previous Releases of NonStop SQL

Typically, a programmer develops an application program on a development system. The SQL statements in the program use DEFINE names to refer to SQL objects. The program is SQL-compiled on the development system against a test database (often in a *staging* subvolume), and the execution plans examined to ensure that they have the desired performance characteristics. The DBA then moves the program to a production system and alters the DEFINE names to point to the SQL objects on the production system. Next, the DBA installs the program by invoking the SQL compiler. Figure 1 (on the following page) shows how application programs are managed in previous releases of NonStop SQL.

The DBA must SQL-compile the program on the production system even though it was previously compiled on the development system. Recompilation can cause the new execution plans to have significantly different performance characteristics than those of the previous compilation. Therefore, the DBA must take time to reexamine the execution plans. The new features in NonStop SQL/MP allow users to install a program without recompilation.

Name Resolution in Previous Releases

In previous releases of NonStop SQL, SQL object names referenced in static SQL statements are resolved when the program is compiled. The DEFINE names are resolved again when the program is started (when the first SQL statement is executed). This feature, called compile-time name resolution, allows the DBA to rebind the static SQL statements to different physical objects than the original objects specified when the program was compiled. (The SQL executor autorecompiles the statements to establish the new bindings.)

However, after the first SQL statement is executed, further changes in the DEFINE names do not cause static SQL statements to be rebound to the new physical objects. If the users want to resolve an object name during execution (for example, allowing a transaction to execute against one of several tables), they must use dynamic SQL. Dynamic SQL, however, may affect application performance and has a more complex programming interface than static SQL.[1]

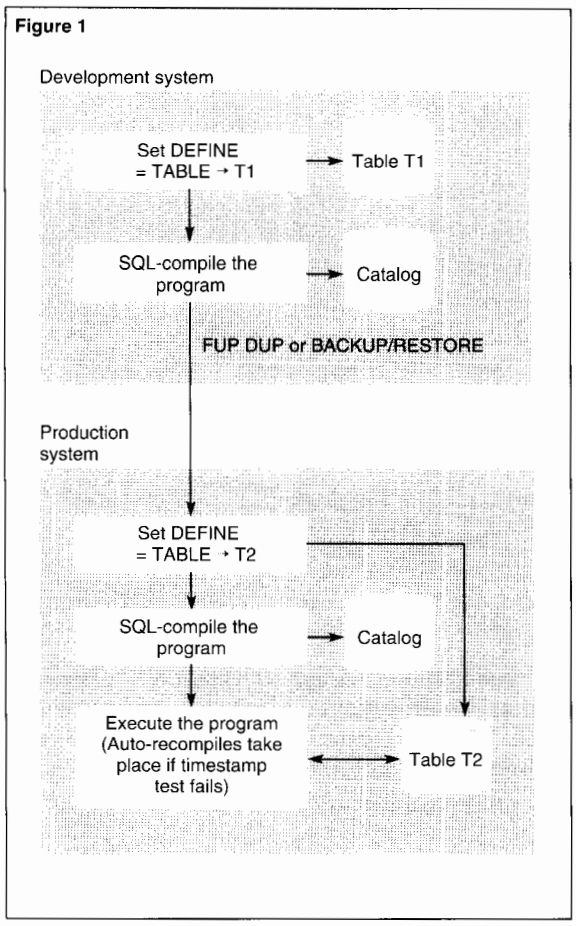
List of Topics in This Article

This article discusses the following topics:

- Managing application programs in previous releases of NonStop SQL.
- Summary of new features.
- Execution-time name resolution.
- The similarity check.
- Interaction between the features.
- The CHECK compiler option.
- Enabling the similarity check on database objects.
- The COMPILE option.
- The REGISTERONLY option.
- The NOREGISTER option.
- User scenarios.

[1]Object names in dynamic SQL statements are resolved when the statement is compiled with the PREPARE statement. After the PREPARE is issued, changes in the DEFINE names do not change the physical objects accessed by the dynamic statement when it is executed. (EXECUTE IMMEDIATE dynamic statements are compiled and executed at the same time; thus, this issue does not arise for these statements.) To access another table, one would have to alter the DEFINE name to point to the table the program wants to access, recompile the dynamic statement using PREPARE, and then execute the statement.

Figure 1.
Managing applications in previous releases of NonStop SQL.



The new execution-time name resolution rules in NonStop SQL/MP address this issue, allowing the DBA to rebind static SQL statements to new physical objects after the DEFINE names are changed without requiring recompilation.

DDL Operations, Program Invalidation, and Auto-Recompilation

After a program is installed on the production system, the DBA may change the logical schema of one or more SQL objects accessed by the program by, for example, adding a column to a table. The DBA may change the physical schema of an SQL object by, for example, moving a partition of a table to a different volume.

A schema change (DDL operation) can affect the status of a program because NonStop SQL saves the execution plans for static SQL statements in the program file. (The SQL executor can reuse the stored plans when an SQL statement is executed multiple times.) A stored execution plan may be rendered inoperable (that is, produce incorrect results) if a DDL operation is performed on an object accessed by the plan. NonStop SQL uses *program invalidation* to notify the SQL executor that a program may contain outdated plans.

The SQL executor determines whether to auto-recompile an SQL statement by comparing the timestamp stored in the statement's execution plan with the redefinition timestamp of the associated object. The process works as follows.

In previous releases of NonStop SQL, when a DDL operation is performed on an object, the SQL catalog manager invalidates all programs that access the object by marking the programs' file labels. It also alters the redefinition timestamp of the object. (The timestamp is stored in the object's file label and in the SQL catalog where the object is registered.)

Altering a redefinition timestamp causes the disk process to delete all active programs' existing OPENs against the object. When the SQL executor reopens (or opens) the object on behalf of an active (or newly active) program, it uses the outdated timestamp stored in the execution plan of the SQL statement being executed. It therefore gets a redefinition-timestamp mismatch. The SQL executor must then auto-recompile the SQL statement. The new execution plan generated by auto-recompilation is not written to the program file on disk and is lost when the program is stopped. Therefore, when the program is restarted, it will undergo auto-recompilations again.

To summarize, the SQL executor auto-recompiles the SQL statements that refer to the modified object in order to adapt to schema changes. Alternately, if the program is not executing, the DBA can manually SQL-compile it to avoid auto-recompilations at run time. (The DBA may also choose to stop an active program and then SQL-compile it.)

Both alternatives have consequences. Auto-recompilation increases the execution time of an SQL statement, which can affect the response times associated with the program. SQL-compilation increases the downtime of the program and forces the DBA to revalidate the execution plans. The new features in NonStop SQL/MP provide ways to avoid auto-recompilation and SQL-compilation.

Summary of New Features

With the new execution-time name resolution rules in NonStop SQL/MP, users can develop programs that execute SQL statements against different tables than those for which the programs were originally compiled. (The tables can reside in the same database or different databases.) Execution-time name resolution is implemented by an option in the CONTROL QUERY statement.

New features in NonStop SQL/MP reduce the extent of program invalidation caused by DDL operations and enable programs to tolerate DDL operations without recompilation. An SQL compiler option, CHECK, implements the *similarity check*, which allows the SQL executor to execute previously invalidated SQL statements without auto-recompilation.

The NonStop SQL/MP compiler option, REGISTERONLY, allows users to install a program without recompilation. With the REGISTERONLY option, users can register a program in the catalog and retain its existing execution plans. Another compiler option, NOREGISTER, enables some programs to be installed without recompilation and without being registered in a catalog. This option makes it easier to install programs that do not need to be registered in a catalog.

Definitions

Automatic recompilation. Performed by the SQL executor at run time if an SQL statement is invalid. The SQL executor compiles the SQL source statement into a new execution plan. Automatic recompilation may or may not cause name resolution of objects referenced in the SQL statement.

Execution characteristics. Characteristics of an execution plan that have no effect on its semantics. Examples are the performance of a plan, its resource consumption, and the objects it accesses.

Inoperable plan. A plan that is semantically incorrect; executing such a plan will lead to incorrect results.

Invalid plan. A plan that is semantically incorrect (inoperable) or a plan whose execution characteristics (such as performance) are sufficiently different from what they would be if the statement were recompiled. The latter plan is operable but not optimal. A plan is invalid if an object it references has been changed by a DDL operation. For a statement that uses execution-time name resolution, a plan is invalid if a DEFINE name has changed and points to a different object than when the plan was generated.

Invalid program. An SQL program whose file label has been marked invalid by a DDL operation, or a program that is started with DEFINE names pointing to different objects than the compile-time objects.

Invalid statement. An SQL statement whose current execution plan is invalid.

Name resolution. The process of resolving a name by expanding DEFINE names and by fully qualifying NonStop Kernel names using the current defaults.

Optimal plan. An operable plan that is the most efficient plan to process the statement against a given set of database objects.

Redefinition timestamp. An SQL object has a redefinition timestamp that is changed each time an invalidating DDL operation is performed on the object. The SQL executor uses the timestamp to identify execution plans that are invalidated by the DDL operation.

SQL compilation. The manual act of compiling an SQL program by invoking the SQL compiler, SQLCOMP, to generate new execution plans for SQL statements in the program. (Also called static SQL compilation.)

Figure 2

(a) New SQLCOMP *options*

```
COMPILE { PROGRAM [STORE SIMILARITY INFO] }
        { INVALID PLANS                   }
        { INOPERABLE PLANS                }

REGISTERONLY { ON  }
             { OFF }

NOREGISTER { ON  }
           { OFF }

CHECK { INVALID PROGRAM }
      { INVALID PLANS   }
      { INOPERABLE PLANS }
```

(b) { CREATE } TABLE *table-name* | *view-name*
{ ALTER }

[SIMILARITY CHECK { ENABLE }]
{ DISABLE }

(c) CONTROL QUERY BIND NAMES { AT STARTUP }
{ AT EXECUTION }

Figure 2.
(a) New compiler options. (b) New CREATE TABLE and ALTER TABLE options. (c) New CONTROL statement.

Finally, high availability compilation allows users to SQL-compile a program containing many SQL statements in much less time than was required in previous releases. With the NonStop SQL/MP compiler option, COMPILE, users can SQL-compile only those statements that require compilation. Statements that are not compiled retain their existing execution plans, thus preserving their performance and other execution characteristics. This feature reduces the total time needed to compile the program and validate new execution plans generated by the SQL compiler.

Figure 2 shows the syntax for the options that implement late binding and high availability compilation. The new features are described in the manual *NonStop SQL/MP Features for Developing and Managing Application Programs* (1994).

Execution-Time Name Resolution

New optional rules in NonStop SQL/MP implement execution-time name resolution for both static and dynamic SQL statements. These rules allow an SQL statement to access different objects each time it is executed. The syntax is as follows:

```
CONTROL QUERY BIND NAMES
{ AT EXECUTION | AT STARTUP }
```

This directive determines whether names of SQL objects referenced in SQL statements are resolved according to the previous name resolution rules in NonStop SQL (AT STARTUP) or the new rules (AT EXECUTION). If the directive is not specified, the default is AT STARTUP. One can enable execution-time name resolution for both DDL and Data Manipulation Language (DML) statements.

Figure 3 shows an example of execution-time name resolution. In the example, a static SQL statement embedded in a TAL™ program accesses two different tables. It modifies DEFINE names programmatically by invoking the relevant NonStop Kernel procedures.

To compile and execute the program shown in Figure 3, one would take the following steps:

1. Before executing the program, establish a DEFINE =T using the following TACL™ command:

   ```
   ADD DEFINE =T,CLASS MAP,FILE T1;
   ```

2. Compile the program with the TAL compiler and then with the SQL compiler. The cursor C is now bound to table T1.
3. Execute the program.

Figure 3

```
EXEC SQL CONTROL QUERY          --Enable execution-time name resolution
  BIND NAMES AT EXECUTION;

EXEC SQL DECLARE C CURSOR       --The cursor C is bound to table T1 at compile time because DEFINE =T
  FOR SELECT * FROM =T;         --maps to table T1 at compile time (see steps 1 and 2 on the previous page).

                                --Alter DEFINE =T to table T2 using the following invocations of NonStop
                                --Kernel procedures;
CALL  DEFINESETLIKE(...);       --Copy the attributes of the DEFINE =T into the working set.
CALL  DEFINESETATTR(...);       --Alter the value of attribute "file" in the working set to T2.
CALL DEFINEDELETE(...);         --Delete the existing DEFINE =T.
CALL DEFINEADD(...);            --Add DEFINE =T using the attributes in the working set.

EXEC SQL OPEN C;                --Read rows from T2.
EXEC SQL FETCH C INTO           --With execution-time name resolution, the cursor C is bound to the current
  host-variables;               --table that DEFINE =T maps to, which is table T2.
EXEC SQL CLOSE C;

                                --Alter DEFINE =T to table T3 using NonStop Kernel procedures shown
                                --above.

EXEC SQL OPEN C;                --Read rows from T3.
EXEC SQL FETCH C INTO           --With execution-time name resolution, the cursor C is bound to the current
  host-variables;               --table that DEFINE =T maps to, which is table T3.
EXEC SQL CLOSE C;
```

Figure 3.
Execution-time name resolution. A static SQL statement embedded in a TAL program accesses two different tables.

Without execution-time name resolution, both cursor scans shown in Figure 3 would have returned data from table T1, because the DEFINE =T mapped to table T1 when the program was compiled. With execution-time name resolution, the accessed table is determined when the cursor is opened. Thus, in Figure 3, the first cursor scan returns data from table T2 and the second returns data from table T3.

Execution-time name resolution requires that each time a statement is executed against a different object, it must be bound to that object. The next section describes a new mechanism, the similarity check, that allows one to bind a statement to a different object without auto-recompilation, thereby significantly decreasing the performance penalty of execution-time name resolution.

The Similarity Check

The similarity check can determine if two SQL objects are similar enough to allow the same execution plan to access either object. For example, when the object a statement refers to at run time is similar to the object the statement was compiled against, the similarity check lets the statement execute without auto-recompilation. One can use the similarity check together with execution-time name resolution and to recover from DDL operations; in both cases it can avoid the penalty of auto-recompilation.

To enable the similarity check at run time, one invokes the new CHECK INOPERABLE PLANS compiler option when SQL-compiling the program. To enable the similarity check at compile time, one invokes the COMPILE INOPERABLE PLANS compiler option. Before using the similarity check, one must also enable it on the object (for tables and protection views) by invoking a new option in the CREATE TABLE | VIEW or ALTER TABLE | VIEW command. The SQL catalog entry for programs, tables, and protection views indicates whether the similarity check is enabled or disabled.

Instead of auto-recompiling an invalid statement, the SQL executor performs the similarity check on the statement's plan (if the check is enabled). If the plan passes the check, it is deemed *operable* (that is, the plan will produce the correct results if executed) and will run without recompilation. In addition, the executor updates the redefinition timestamps in the plan (in memory). If the similarity check fails (the plan is *inoperable*) or is disabled, the statement is auto-recompiled.

The similarity check compares the schemas of the compile-time and run-time objects. Information about the schema of a compile-time object is saved in the SQL statements' execution plans. The SQL executor retrieves information about the schema of a run-time object from the disk process.

The similarity check reduces the need for recompilation and thus can improve the response time of a program. However, an operable plan may not be optimal for a specific set of database objects. The user must weigh the cost of recompiling a statement against the cost of reusing an execution plan that might not be optimal.

Each invalid statement undergoes one or more similarity checks for each object it references. The cost of each similarity check is about the same as an OPEN operation (one message to the disk process). The disk process may have to perform several physical disk I/Os to retrieve the object's schema information from disk labels. However, the cost of performing similarity checks for any SQL statement is far lower than the cost of compiling the statement. (The similarity check currently does not support parallel plans.)

Interaction Between the Features

The interactions between the features described in this article are complex. To make the features easier to understand and use, they have been designed to be *orthogonal*. That is, the use of one feature does not influence the behavior of other features.

Assume, for example, that an SQL statement in a program uses execution-time name resolution. Because of the orthogonality principle, this feature will work whether or not another feature such as the similarity check is enabled.

Table 1.
Behavior of the new compiler (SQLCOMP) options in NonStop SQL/MP.

Option	During compilation	At run time
CHECK INVALID PROGRAM	No impact on compilation.	The SQL executor auto-recompiles all statements if the program is invalid or if run-time DEFINE names differ from compile-time DEFINE names.
CHECK INVALID PLANS	No impact on compilation.	The SQL executor auto-recompiles invalid statements. Valid statements are not auto-recompiled; their existing plans are reused.
CHECK INOPERABLE PLANS	No impact on compilation.	The SQL executor performs the similarity check for invalid statements. Only plans that fail the check (inoperable plans) are auto-recompiled. Plans that pass the check and plans for valid statements are reused.
COMPILE PROGRAM	All statements are compiled.	No impact on run-time behavior.
COMPILE INVALID PLANS	Invalid statements are compiled. Valid statements are not compiled; their existing plans are reused.	No impact on run-time behavior.
COMPILE INOPERABLE PLANS	The similarity check is performed for invalid statements. Only plans that fail the check (inoperable plans) are compiled. Plans that pass the check and plans for valid statements are reused.	No impact on run-time behavior.
REGISTERONLY ON	No statements are compiled. All execution plans are retained. The program is registered in a specified catalog.	No impact on run-time behavior.
NOREGISTER ON	Statements are compiled according to the COMPILE option one uses. The program can be installed on a new system without registering it in a catalog (without processing by SQLCOMP on the new system).	No impact on run-time behavior.

For example, if the similarity check is disabled, the SQL statement will always undergo auto-recompilation if an object reference in the statement changes to a different physical object. (Auto-recompilation binds the statement to a different physical object.) If the similarity check is enabled and it passes, the statement is bound to a different object without auto-recompilation. Thus, execution-time name resolution takes effect in both cases.

The similarity check plays a special role among the features described in this article. When a program is executed, one can use the similarity check to avoid auto-recompilations that might result because one has used another feature. Thus, the similarity check can improve the performance of the other features. However, in keeping with the principle of orthogonality, the other features will function as they are supposed to whether or not one uses the similarity check. Table 1 describes the behavior of the new compiler options during compilation and at run time.

In practice, of course, the different combinations of features produce different results, and users will want to choose combinations that best suit their application environment. The user scenarios at the end of the article give a few examples.

The CHECK Compiler Option

When used with the RECOMPILEALL option, the CHECK compiler option determines the extent to which auto-recompilations take place when an invalid program is executed. (RECOMPILEALL, an existing compiler option, tells the SQL executor to perform all auto-recompilations at program-startup time, when the first SQL statement is executed.)

Although one specifies the CHECK option when SQL-compiling a program, it influences the behavior of the SQL executor at run time. The CHECK option has no impact on compilation.

The syntax for the CHECK option is as follows:

```
CHECK { INVALID PROGRAM | INVALID
        PLANS | INOPERABLE PLANS }
```

CHECK INVALID PROGRAM, the default option, provides the behavior found in previous releases of NonStop SQL. All statements in an invalid program are auto-recompiled.

CHECK INVALID PLANS causes only invalid statements to be auto-recompiled. The SQL executor reuses plans for valid statements.

CHECK INOPERABLE PLANS enables the similarity check for the program. The SQL executor only auto-recompiles inoperable statements (that is, invalid statements whose plans fail the similarity check).

The CHECK option also determines whether the SQL executor auto-recompiles a statement that is invalidated during program execution or reuses the existing execution plan. This also applies when one uses the RECOMPILEONDEMAND option; with this option, auto-recompilation of an invalid statement is deferred until the statement is executed.

If one uses CHECK INVALID PROGRAM or CHECK INVALID PLANS, the invalid statement is auto-recompiled. If one uses CHECK INOPERABLE PLANS, the invalid statement undergoes similarity checks. If all the similarity checks pass, the existing execution plan is reused. Otherwise, the statement is auto-recompiled.

Using the CHECK Option With Execution-Time Name Resolution

To illustrate how the similarity check allows one to avoid auto-recompilations when using execution-time name resolution, assume first that the program shown in Figure 3 is compiled without enabling the similarity check.

When the program is statically compiled, the DEFINE =T referenced by cursor C points to table T1. Therefore, the statement is bound to table T1 when the statement is compiled. When the DEFINE =T is altered to point to table T2 and the cursor C is opened, the cursor must be rebound to table T2 (because the cursor uses execution-time name resolution). To change the binding of the statement, it must be auto-recompiled.

Now assume one used the CHECK INOPERABLE PLANS option to compile the program. If the similarity check determines that table T1 is similar to table T2, the statement can be bound to table T2 at execution time without auto-recompilation. This significantly speeds up the process of reestablishing a new binding for the statement.

Reducing the Impact of Program Invalidation

In NonStop SQL/MP, many DDL operations on an SQL object do not invalidate programs that reference the object if a program is compiled with the CHECK INOPERABLE PLANS option and the object has the similarity check enabled. Not invalidating a program has the beneficial side effect of retaining the usages for the program in the SQL catalog. These DDL operations will update the redefinition timestamp of the object.

Enabling the Similarity Check on Database Objects

The CHECK INOPERABLE PLANS option enables the similarity check for an SQL program. To succeed, the similarity check must also be enabled on the objects being checked. The user must explicitly enable the similarity check for tables and protection views; it is implicitly enabled for all other objects (except shorthand views, which are not supported).

In the example shown in Figure 3, when the DEFINE =T is altered to point to table T2 and the cursor C is opened, two components of the similarity check must be enabled. First, the program must have been compiled with CHECK INOPERABLE PLANS. Second, the similarity check must be enabled on table T2. The similarity check must ensure that, among other attributes, the column names are identical in the two objects.

To accomplish this, one uses the new SIMILARITY CHECK ENABLE option in the CREATE TABLE | VIEW and ALTER TABLE | VIEW commands, which adds column names to the disk labels for tables and protection views.[2] Figure 4 shows the syntax of the SIMILARITY CHECK options for these commands.

The similarity check also needs to be enabled on objects if the COMPILE INOPERABLE PLANS option is used (described in the next section).

Figure 4

```
CREATE TABLE table-name [ SIMILARITY CHECK { ENABLE | DISABLE } ] ;

CREATE VIEW view-name [ SIMILARITY CHECK { ENABLE | DISABLE } ] FOR PROTECTION ;

ALTER TABLE table-name [ SIMILARITY CHECK { ENABLE | DISABLE } ] ;

ALTER VIEW view-name [ SIMILARITY CHECK { ENABLE | DISABLE } ] ;
```

Figure 4.
Syntax of the SIMILARITY CHECK option in the CREATE TABLE | VIEW and ALTER TABLE | VIEW commands.

[2]Adding column names to the disk labels may cause the NonStop SQL version of the object to be incremented; the version will be at least 310. This will make the object inaccessible from remote systems running NonStop SQL software that is a lower version than the version of the object. Because of the potential for reduced network accessibility, NonStop SQL/MP gives the DBA explicit control over enabling the similarity check for table and protection views by providing the new CREATE and ALTER options. (The similarity check is implicitly enabled for other objects because they do not have columns.)

The COMPILE Option

A new NonStop SQL/MP feature, high availability compilation, allows one to selectively compile only those static SQL statements that need compilation. By not having to compile all SQL statements, this feature speeds up compilation and retains existing execution plans whenever possible.

The new feature, enabled with the COMPILE option, differs from the CHECK option. The COMPILE option influences the behavior of the SQL compiler, whereas the CHECK option influences the behavior of the program when it is executed.

The COMPILE option has the following syntax:

```
COMPILE
   { PROGRAM [ STORE SIMILARITY INFO ] }
   { INVALID PLANS                     }
   { INOPERABLE PLANS                  }
```

COMPILE PROGRAM, the default option, specifies the behavior in previous releases of NonStop SQL. (All statements are compiled.) If STORE SIMILARITY INFO is specified, the SQL compiler stores similarity information in the program file so that the user may later compile the program with the COMPILE INVALID PLANS or COMPILE INOPERABLE PLANS option.

The COMPILE INVALID PLANS option specifies to the SQL compiler that it should compile only invalid statements. Valid statements are not compiled and their existing plans are reused.

The COMPILE INOPERABLE PLANS option further reduces the number of statements that are compiled. With this option, the SQL compiler performs the similarity check for invalid statements. Only invalid statements whose plans fail the similarity check (that is, are inoperable) are compiled. For all other statements (valid or invalid), the existing plans are reused.

The execution plan for an invalid statement may not be optimal even if it passes the similarity check (that is, is operable). Such a plan may not have the best performance characteristics. If this is a concern, one should use the COMPILE INVALID PLANS or COMPILE PROGRAM option.

The REGISTERONLY Option

The new NonStop SQL/MP compiler option, REGISTERONLY ON, allows one to install a previously SQL-compiled program without recompiling it. The program is registered in a catalog and the existing execution plans retained. The syntax for this option is as follows:

```
REGISTERONLY { ON | OFF }
```

REGISTERONLY OFF, the default option, specifies the previous behavior. The program is SQL-compiled and registered in the specified catalog.

The REGISTERONLY ON option does not SQL-compile the program; instead, it retains the existing execution plans. The program is registered in the specified catalog.

With REGISTERONLY ON, one uses the SQL compiler as a simple utility; the SQL compiler does not modify the SQL objects (execution plans) in the program.

A program installed using REGISTERONLY ON is marked valid, but SQL statements in the program may be invalid, because the program (when executed) may access different objects than those with which the program was last fully SQL-compiled (using REGISTERONLY OFF). In addition, REGISTERONLY ON does not create usage entries in the catalog for the program. Therefore, when a DDL operation is performed on an object referenced by the program, the SQL catalog manager cannot explicitly invalidate it.

In these cases, the user will not know that some statements in the program are invalid and may have to be auto-recompiled before they are executed. To avoid auto-recompilation, one should enable the similarity check, using the CHECK INOPERABLE PLANS option, during the last full SQL compilation.

In some cases, the statistics in the test-database catalog do not represent the production database. In others, parallel execution is important for the application, and the table partitioning in the test database differs from the partitioning in the production database. In these cases, plans generated using the production catalog may be more efficient than plans that used the test catalog. Although REGISTERONLY ON is attractive for change-control purposes, it is not recommended in these cases. Instead, users should SQL-compile the program on the production system.

The NOREGISTER Option

A new NonStop SQL/MP feature makes it possible to install an SQL program without registering it in a catalog (without invoking SQLCOMP). Thus, one can move the program to a new location (using the RESTORE or FUP DUP operation) and execute it without any processing by SQLCOMP. To accomplish this, the last SQL-compile of the program must have been done with the new compiler option, NOREGISTER ON. The syntax for this option is as follows:

```
NOREGISTER {ON | OFF}
```

NOREGISTER OFF, the default option, specifies the previous behavior. That is, to install a program in a new location, one must register the program in an SQL catalog at the new location.

The NOREGISTER ON option differs from REGISTERONLY ON in two ways. First, it allows one to install a program without registering the program in a catalog. Second, one invokes it during compilation when the program is still on the old (original) system. In contrast, REGISTERONLY ON is invoked after the program is moved to the new system. One cannot use NOREGISTER ON in conjunction with REGISTERONLY ON.

For a program to be compiled successfully with the NOREGISTER ON option, all static DML statements in the program should use execution-time name resolution. When a program uses dynamic SQL, or static SQL statements together with execution-time name resolution, it does not have strong bindings to any SQL objects. Therefore, there is no need to register the program in a catalog in order to record dependencies on any objects. For more information on the affected DML statements, refer to the new manual *NonStop SQL/MP Features for Developing and Managing Application Programs* (1994).

When one installs a program compiled with the NOREGISTER ON option, it may undergo auto-recompilation if it accesses different objects at execution time than those with which it was SQL-compiled. In addition, the SQL catalog manager will not invalidate the program after a DDL operation because the program is not registered in a catalog, and there is no record of dependencies it has on SQL objects.

As with REGISTERONLY ON, one can avoid auto-recompilations by enabling the similarity check for the program. To do this, one SQL-compiles the program using CHECK INOPERABLE PLANS together with NOREGISTER ON.

User Scenarios

The new options offer many choices that allow the DBA to control the extent and timing of SQL compilation and auto-recompilation. For example, REGISTERONLY ON allows the DBA to install a program on a new system without SQL-compiling it. However, REGISTERONLY ON does not record dependencies, and statements may undergo auto-recompilation at run time. Alternatively, the DBA can install the program using COMPILE INVALID PLANS or COMPILE INOPERABLE PLANS. With these options, compilation is minimized, but the SQL compiler records dependencies and compiles statements that might otherwise undergo auto-recompilation. The following user scenarios give examples of how to use the new options.

Scenario 1: Moving a Program From Development to Production Without SQL-Compilation

In this scenario, users develop application programs on a development system and move them to execute on one or more production systems. After the programs are moved to a new system, they must refer to a new set of SQL objects (such as tables). However, the objects on the two systems have identical logical schemas. Moreover, the users have made sure that compiling the programs against the test database on the development system produces execution plans with appropriate performance characteristics for the production system. The users would like to retain these execution plans when the programs are installed on the production system.

With NonStop SQL/MP, the users can move a program to the production system without having to recompile it. In addition, the program retains its existing execution plans. Thus, NonStop SQL/MP eliminates recompilation, and the users save time because they do not have to verify new execution plans.

In this scenario, the users should take the following steps:

1. Compile the program on a development system using the CHECK INOPERABLE PLANS compiler option. Use DEFINE names in SQL statements to refer to objects.
2. Move the program to a production system.
3. Register the program in the catalog on the production system by using the REGISTERONLY ON option. This step installs the program quickly and does not change the execution plans.
4. Enable the similarity check for tables and protection views (referenced by the program) on the production system by using the ALTER ...SIMILARITY CHECK ENABLE statement (if this has not been done previously).
5. Execute the program with DEFINE names that point the program to the objects on the production system.

Figure 5 illustrates Scenario 1. The CHECK INOPERABLE PLANS option enables the similarity check and stores similarity information in the execution plans when the program is SQL-compiled. The REGISTERONLY ON option stores the program information in the PROGRAMS table of the production catalog. The program retains all of its attributes. Thus, it is significantly faster than the full compile required by previous releases of NonStop SQL. Since static compilation is not performed on the production system, the execution plans are unchanged.

At run time, the SQL executor performs a similarity check on the production objects associated with the statement being executed. The executor determines if the attributes of the objects are the same as those of the development-system objects with which the program was compiled. If they are the same, the execution plan is reused. Otherwise, auto-recompilation is attempted.

Users should be aware of two conditions in this scenario. First, if the production objects have different attributes than the development objects (such as different default values for columns), the similarity checks will fail, and auto-recompilation will take place at run time. Second, the REGISTERONLY ON option does not store usages in the catalog on the production system. If users want usages, they should SQL-compile the program on the production system without using REGISTERONLY ON.

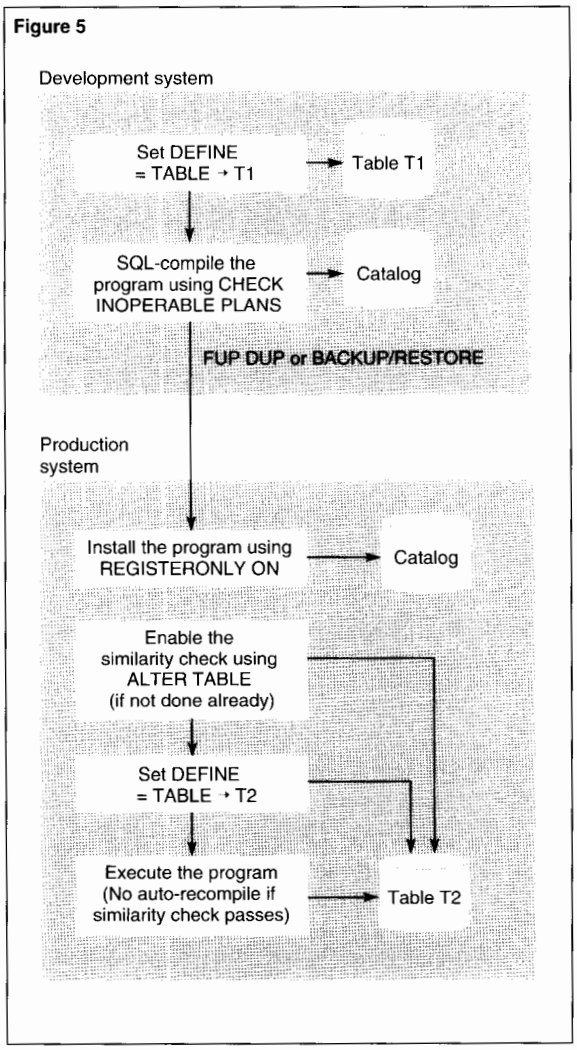
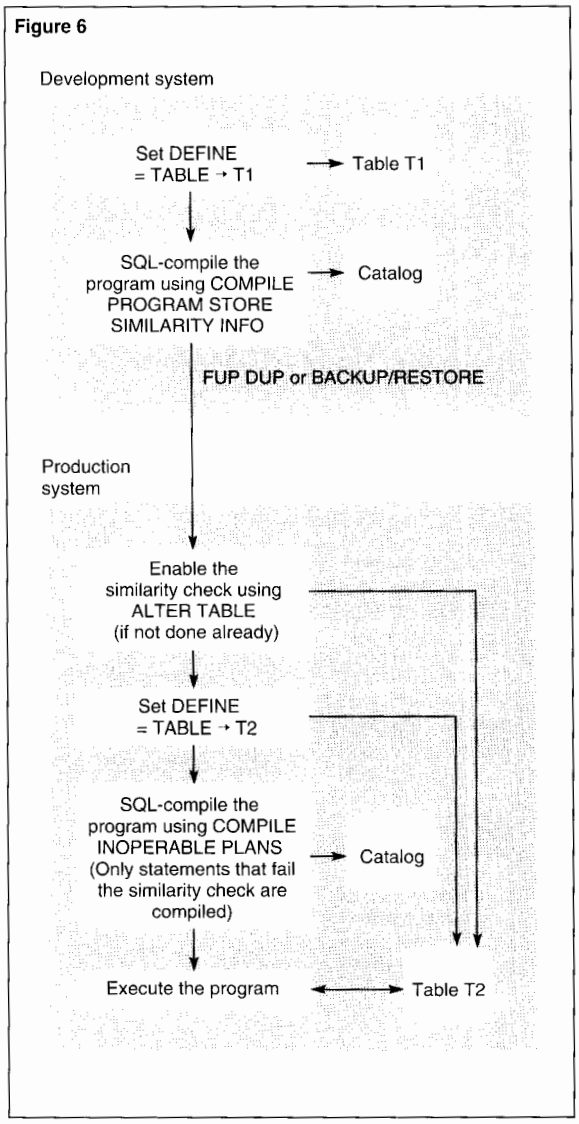


Figure 5.
Scenario 1: Moving a program from development to production without SQL-compilation.

Figure 6.
Scenario 2: Moving a program from development to production with minimal SQL-compilation.



Scenario 2: Moving a Program From Development to Production With Minimal SQL-Compilation

This scenario is similar to Scenario 1 except that the users have changed a few SQL objects on the production system. The execution plans for the SQL statements that refer to the changed objects will be inoperable when the program is moved from the development to the production system. However, only a few objects and statements are affected. The majority of production objects are logically identical to those on the development system, and most of the existing execution plans (compiled on the development system) are both semantically correct and have appropriate performance characteristics for the production system.

With NonStop SQL/MP, the users can selectively recompile only the inoperable plans after moving the program to the production system. They do not have to recompile the entire program. Thus, NonStop SQL/MP greatly reduces recompilation time, and the users save time because they only have to examine a few new execution plans.

In this scenario, the users should take the following steps:

1. Compile the program on the development system using the COMPILE PROGRAM STORE SIMILARITY INFO compiler option. Use DEFINE names in SQL statements to refer to objects.
2. Move the program to the production system.
3. Enable the similarity check for tables and protection views (referenced by the program) by using the ALTER ...SIMILARITY CHECK ENABLE statement (if this has not been done previously).
4. Alter the DEFINE names to point the program to the objects on the production system.
5. Compile the program using the COMPILE INOPERABLE PLANS option.
6. Execute the program.

Figure 6 illustrates Scenario 2. Instead of using REGISTERONLY ON (as in scenario 1), the users SQL-compile the program to install it on the production system. They use the COMPILE INOPERABLE PLANS option, which ensures that inoperable plans are recompiled now so that they do not undergo auto-recompilation at run time. The program thus avoids the response-time penalty caused by auto-recompilation.

With this COMPILE option, the SQL compiler compiles only the inoperable plans. It retains the existing plans for statements that are valid or have operable plans, thus retaining their performance and other execution characteristics. The SQL compiler also records the dependencies for the program in the catalog.

Scenario 3: Avoiding Unnecessary Auto-Recompilations After a DDL Operation

In this scenario, the users' application is working fine. However, because the system is growing or physically changing, the users perform DDL operations (such as Split Partition) on SQL objects.

The similarity check, together with new DDL operation behavior, allows the users to avoid explicit SQL-recompilation and run-time auto-recompilation after DDL operations occur. This improves application availability and permits the users to retain existing execution plans.

In this scenario, the users should take the following steps:

1. Compile the program using the CHECK INOPERABLE PLANS option when developing the program.
2. Install the program on the production system, as in Scenario 1.
3. On the production system, enable the similarity check for tables and protection views (referenced by the program) by using the ALTER ...SIMILARITY CHECK ENABLE statement (if this has not been done previously).
4. Execute the program.
5. Perform the DDL operation.

Figure 7 illustrates Scenario 3. The users specify the CHECK INOPERABLE PLANS option during the SQL compilation step when they are developing the program. This enables the similarity check for program execution. The users also enable the similarity check for the SQL objects (in Step 3). If the DDL operation will be performed on objects other than tables and protection views, Step 3 is not needed.

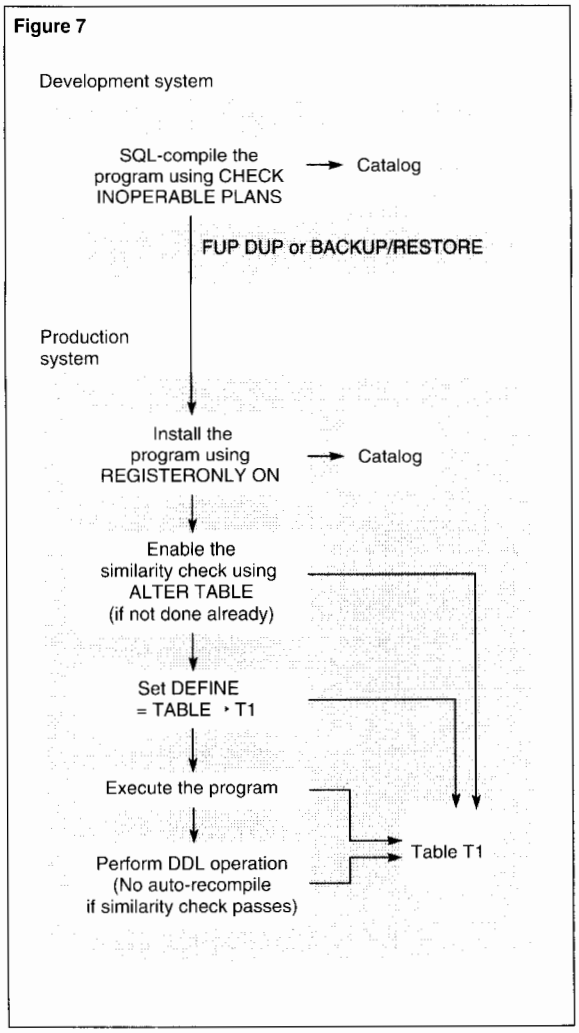
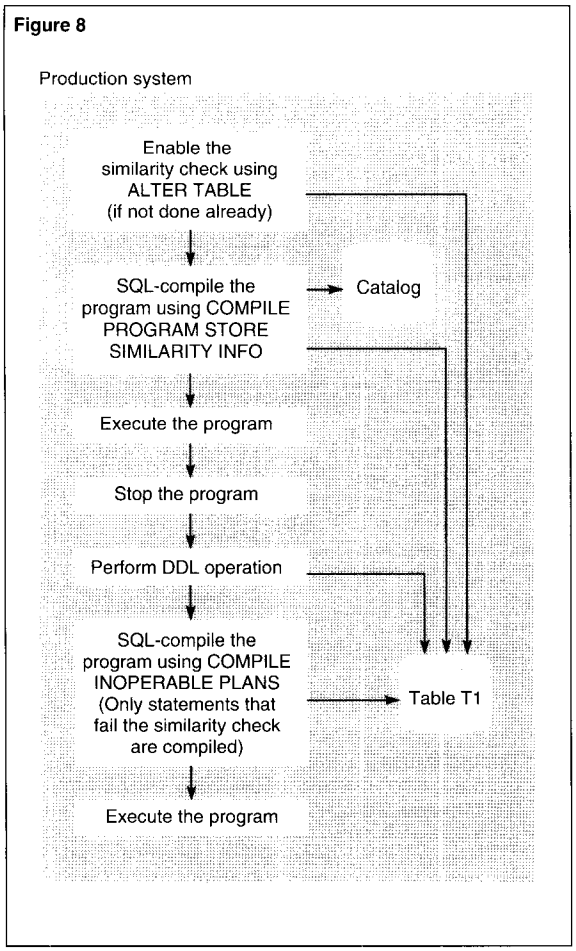


Figure 7.
Scenario 3: Avoiding unnecessary auto-recompilations after a DDL operation.

After the DDL operation occurs, the SQL executor performs the similarity check on statements that refer to the object. If the DDL operation did not change any attributes relevant to the correct execution of the plan, the plan is executed without auto-recompilation. In addition, the DDL operation will not invalidate the program. Thus, for example, the users can increase the number of partitions in a table or create a new index on a table without invalidating the program.

Figure 8.
Scenario 4: Recovering from a DDL operation with minimal SQL-compilation (for situations in which a time window is available for system management).



Most, but not all, DDL operations support the behavior described in this scenario. For a list of DDL operations that do support this behavior, refer to the new manual *NonStop SQL/MP Features for Developing and Managing Application Programs* (1994).

Scenario 4: Recovering from a DDL Operation with Minimal SQL-Compilation

In this scenario, an object has been changed so that the similarity check will fail when the statements that refer to the object are executed. For example, the users drop an index that is used by a plan. If the similarity check fails, recompilation takes place. (In Scenario 3, these statements would be auto-recompiled at run time.)

In this scenario, the users prevent auto-recompilations by SQL-compiling the program after the DDL operation is performed on the object. They use an option that performs the similarity check during explicit SQL compilation (rather than at run time), and that only compiles the statements that fail the similarity check. (This scenario assumes that a time window is available for system management.)

Thus, this scenario allows the users to recover from DDL operations with minimal recompilation. It also enables the SQL executor to run the program without having to auto-recompile any execution plans, thus retaining the application's response time.

In this scenario, the users should take the following steps:

1. Compile the program with the COMPILE PROGRAM STORE SIMILARITY INFO option to ensure that the execution plans contain similarity information and that all the SQL statements are compiled.
2. Execute the program.
3. Stop the program.
4. Perform the DDL operation on the object referenced by the program.
5. Recompile the program using the COMPILE INOPERABLE PLANS option.
6. Execute the program.

Figure 8 illustrates Scenario 4. The COMPILE INOPERABLE PLANS option causes the SQL compiler to perform a similarity check and compile the inoperable plans in the program. It does not compile statements that are valid or have operable plans; these statements retain their existing execution plans.

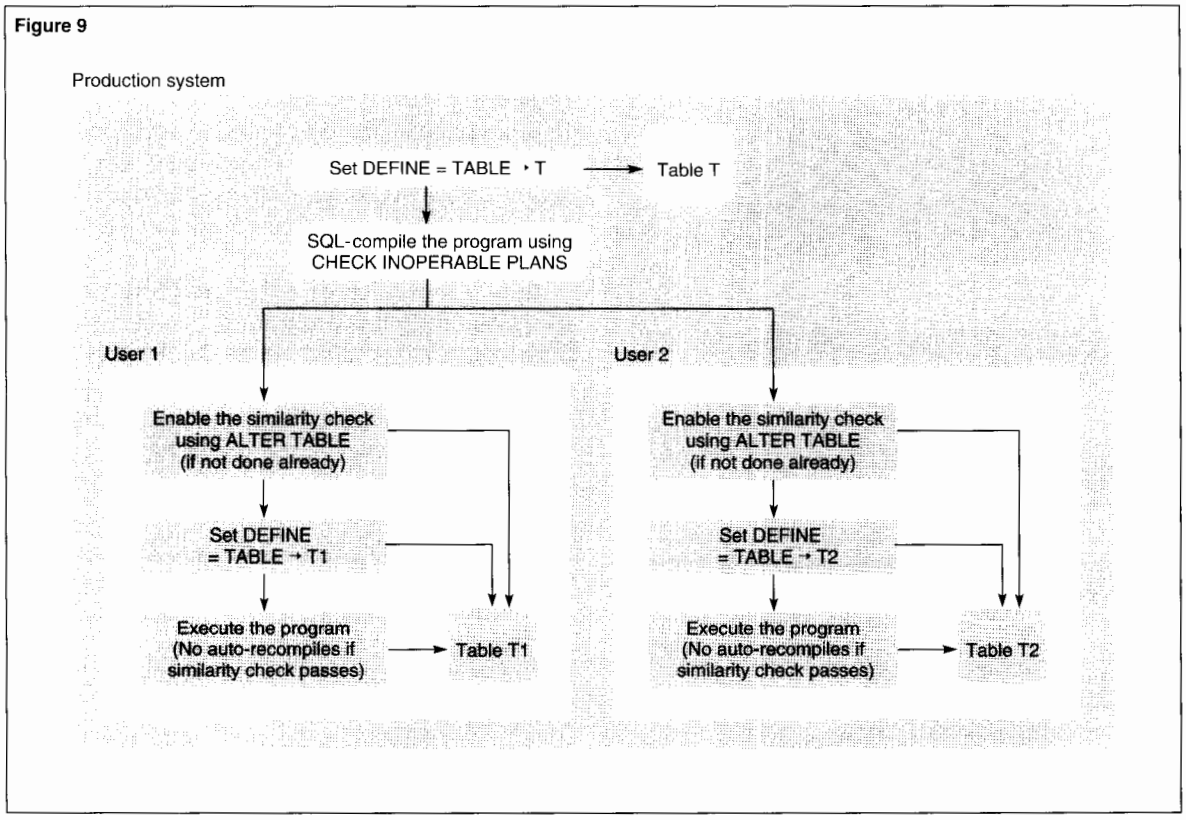


Figure 9.
Scenario 5: Allowing different users to run a program against their own databases without auto-recompilation.

Scenario 5: Allowing Different Users to Run a Program Against Their Own Databases Without Auto-Recompilation

In this scenario, each user of a program wishes to run a copy of the program against the user's own database. The database is supplied at program-startup time by using DEFINE names.

In previous releases of NonStop SQL, supplying a new set of DEFINE names at program-startup time causes auto-recompilation. With NonStop SQL/MP, the users can employ the similarity check to avoid auto-recompilation. The similarity check eliminates auto-recompilation when the program is run against objects different from those the program was compiled against, as long as the two sets of objects have similar attributes.

In this scenario, the users should take the following steps:

1. Compile the program using the CHECK INOPERABLE PLANS option and use DEFINE names for object names.
2. If the objects the program will access are tables or protection views, enable the similarity check for them by using the ALTER ...SIMILARITY CHECK ENABLE operation (if this has not been done previously).
3. Execute the program with DEFINE names pointing to a different set of objects.

Auto-recompilations will not take place. Figure 9 illustrates Scenario 5.

Figure 10

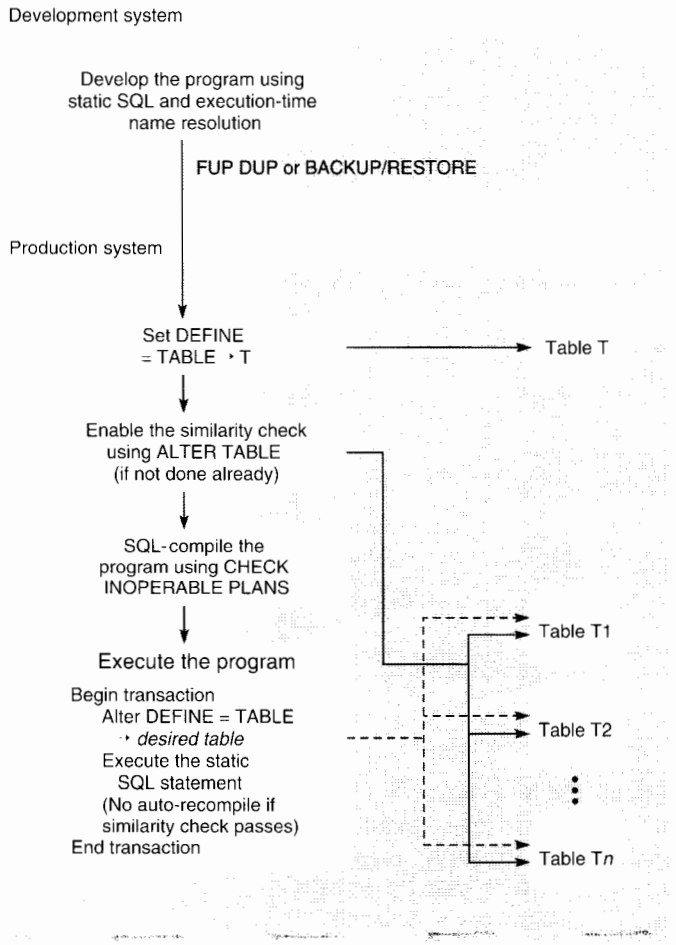


Figure 10.
Scenario 6: Dynamically changing the database a program will access (without requiring dynamic SQL).

This solution has one potential disadvantage. Whenever NonStop SQL reuses an execution plan against a different set of objects than those specified when the plan was generated, the plan may not be optimal for the new objects. Therefore, the SQL statement may take longer to execute than it would if the statement had been auto-recompiled.

One effect of using this scenario is that dependencies for this program are only available for the objects the program was compiled against.

Scenario 6: Dynamically Changing the Database a Program Will Access, Without Requiring Dynamic SQL

In this scenario, the users have several databases that are all similar and contain identical sets of tables and other objects. A program decides on a per-transaction basis which database to use.

In previous releases of NonStop SQL, the users can write this program by using dynamic SQL to specify at run time the database to be used for the query. Dynamic SQL, however, may affect application performance and has a more complex programming interface than static SQL. Another option is to combine all of the databases into one, but managing such a large database would be a complex task. NonStop SQL/MP solves this dilemma by providing execution-time name resolution for SQL statements.

In this scenario, the users should take the following steps:

1. Add the CONTROL QUERY BIND NAMES AT EXECUTION directive to specify execution-time name resolution. Use DEFINE names in static SQL statements to refer to a dynamic database; write code to modify the DEFINE values to specify the desired database. (Figure 3 shows an example of code that modifies DEFINE names.)
2. If the objects the program will access are tables or protection views, enable the similarity check for them by using the ALTER ...SIMILARITY CHECK ENABLE operation (if this has not been done previously).
3. Compile the program with the CHECK INOPERABLE PLANS option.
4. Execute the program.

Figure 10 illustrates Scenario 6. In Step 1, the users add the CONTROL QUERY BIND NAMES AT EXECUTION directive to the program's SQL code, together with program logic to manipulate the DEFINE values used in the SQL statements. When this directive is specified, NonStop SQL rebinds the DEFINE names used in the SQL query at statement execution time.

Thus, at run time, the program determines the database to be used. The program logic modifies the DEFINE names to point to the appropriate database objects before it executes an SQL statement. The similarity check then enables the statement to execute against the new database without undergoing auto-recompilation.

This scenario differs from Scenario 5 in the following manner. In Scenario 5, DEFINE names are set externally to the program. In this scenario, DEFINE names are set by the program. In addition, in this scenario, the static SQL statements in the program use execution-time name resolution to cause the DEFINE changes to take effect when the statement is executed. This scenario applies to several user situations, some of which are described below.

Log File Rollover. Some applications create a new log file and insert data into it starting at midnight. Such applications would use the CREATE TABLE operation to create the new log file; they would then modify the DEFINE name to point to the new log file. The next insertion into the log file would automatically insert data into the newly-created file.

A Utility That Works on a Single Subvolume at a Time. Some programs switch to a different subvolume whenever the user switches context. An example is a source-control program that switches to a different database when the user wants to manage a different set of source-code programs. (The databases have identical schemas but reside on different subvolumes.) Such a program would modify all of its DEFINE names to point to a different database whenever the program wanted to switch context.

Choosing a Database on a Per-Transaction Basis. In this situation, the program would change its DEFINE names to point to a new set of tables based on the transaction it would execute next.

Scenario 7: Switching Between Databases and Always Using the Optimal Plan

This scenario is a variation of Scenario 6. In this scenario, the users want to dynamically switch between databases, or between objects in the same database, and always use the optimal plan. The users can accomplish this by using the CONTROL QUERY BIND NAMES statement and changing DEFINE names dynamically to switch to a different database object, as in Scenario 6. The users can use either static or dynamic SQL.

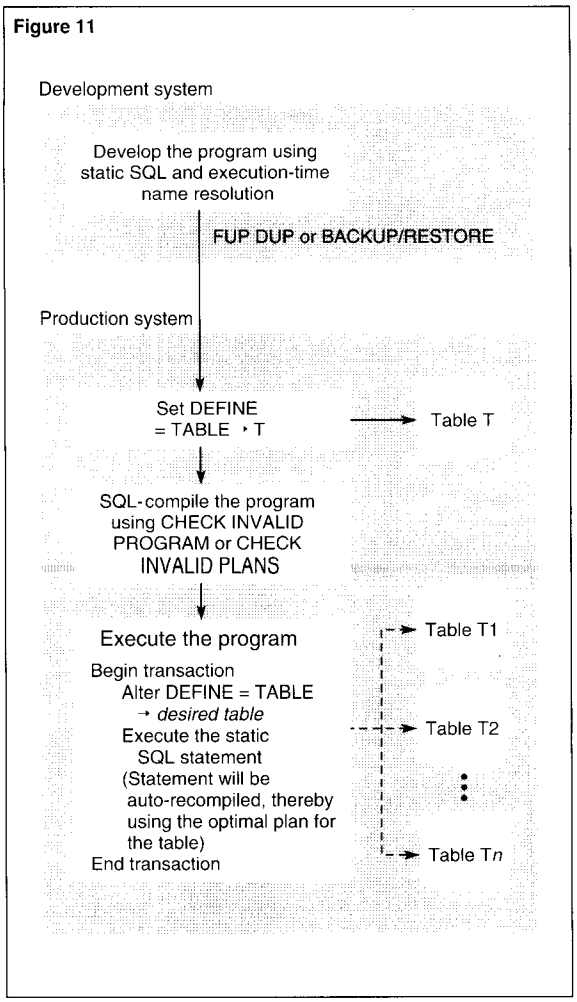
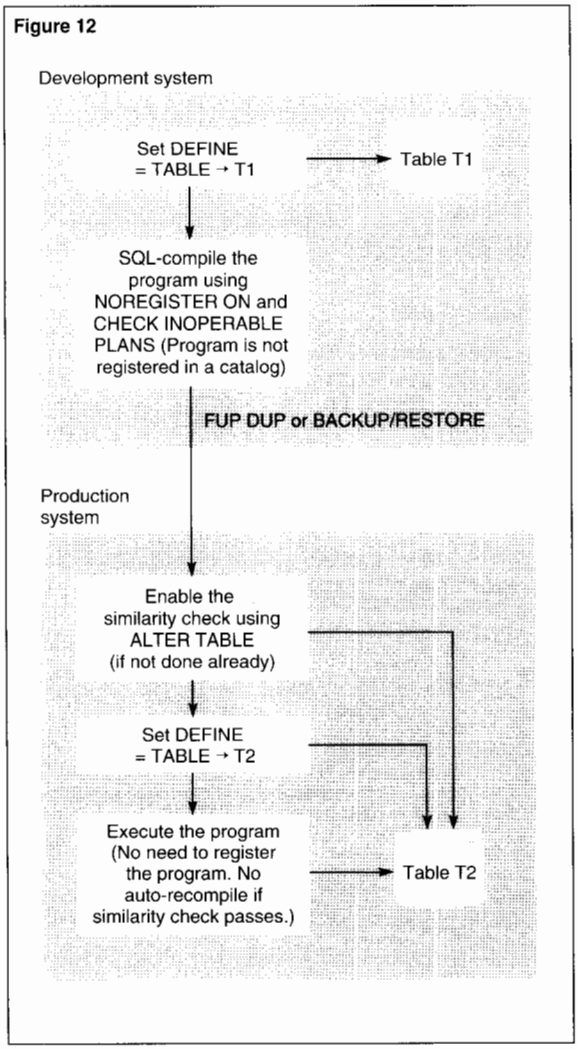


Figure 11.
Scenario 7: Switching between databases and always using the optimal plan.

Figure 12.
Scenario 8: Installing a program without registering it in a catalog.



The users compile the program using the CHECK INVALID PROGRAM or CHECK INVALID PLANS option. At run time, these options will cause the SQL executor to auto-recompile the SQL statement whenever DEFINE names change, thus ensuring that the execution plan for the SQL statement will be optimal for the database being accessed. Figure 11 illustrates this scenario.

Scenario 8: Installing a Program Without Registering It in a Catalog

In this scenario, the users have an application that only contains statements that do not have any usages. These can be dynamic SQL statements, static SQL statements that use execution-time name resolution, and DDL and Data Control Language (DCL) statements. The users want to distribute the program to other users without requiring them to recompile or register the program on their systems. This simplifies the process of installing the program.

In this scenario, the users should take the following steps:

1. Develop the program as in previous releases of NonStop SQL. Compile it using the NOREGISTER ON and CHECK INOPERABLE PLANS options.
2. Move the program to any system, using Enscribe or NonStop SQL utilities.
3. Execute the program. No SQLCOMP step is required to install the program.

Figure 12 illustrates Scenario 8. The NOREGISTER ON compiler option allows the SQL executor to run the program even though it has not been compiled on the target system or registered in a catalog on the target system.

The disadvantage of this scenario is that the program installed using this method is not invalidated by DDL operations. (Invalidation does not occur because the program's dependencies on SQL objects are not registered in a catalog.)

Conclusion

NonStop SQL/MP contains several new features that help users develop and manage application programs. Execution-time name resolution enables a static SQL statement to access different SQL tables (with identical schemas). This feature allows users to develop programs that access multiple databases or tables using static SQL.

One can use the REGISTERONLY compiler option to install a program without compiling the SQL statements in the program. This speeds up installation and preserves the execution plans in the program. The NOREGISTER compiler option allows users to install a program without registering it in a catalog, thus making it easy to install programs. The COMPILE option preserves existing execution plans in a program and speeds up the process of SQL-compiling a program.

Another new mechanism, the similarity check, allows SQL statements in a program to tolerate DDL operations (on objects referenced by the statements) without auto-recompilation. The similarity check makes execution-time name resolution efficient by avoiding auto-recompilation whenever possible. The similarity check also enables programs installed with REGISTERONLY ON or compiled with NOREGISTER ON to avoid auto-recompilation, thus making these features more efficient.

These late binding and high availability compilation features reduce explicit and implicit recompilations of programs, thereby decreasing application downtime and increasing the availability of applications running on Tandem NonStop systems.

References

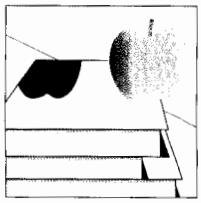
NonStop SQL/MP Features for Developing and Managing Application Programs. 1994. Tandem Computers Incorporated. Part no. 103386.

Acknowledgments

Many individuals in the NonStop SQL Development Group contributed to the specification of the features described in this article, including Pedro Celis, Liz Chambers, Gary Gilbert, Louise Madrid, Gary Ngai, Franco Putzolu, Jim Troisi, and Hans Zeller.

The following people implemented the new features: Pei-Jyun Leu, Haleh Mahbod, Richard Meier, Tom O'Shea, Mike Stewart, Jim Troisi, and Hans Zeller. Theresa Ledet provided quality assurance, and John Spencer documented the new features. Jim Troisi developed the scenarios in the article. The reviewers greatly improved the clarity of the article.

Sunil Sharma helped design the late binding features and implemented the changes to the SQL compiler and SQL executor. Since joining Tandem in 1983, Sunil has developed software for the NonStop Kernel operating system and the Disk Process (DP2) as well as NonStop SQL. He has a B.Tech. in Electrical Engineering from the Indian Institute of Technology, New Delhi, and an M.S. in Computer Engineering from Rensselaer Polytechnic Institute, New York.



Tandem Education

The following paragraphs provide highlights of the latest education courses offered by Tandem. To sign up for a class or to order an independent study program (ISP), users should call 1-800-621-9198. Full descriptions of all available courses and ISPs appear in the *Tandem Education Course Catalog* and on InfoWay.

Integrity FT and NonStop-UX Installation

In this five-day lecture-and-lab course, students learn everything they need to know about installing the Tandem Integrity FT System and the NonStop-UX operating system. This course covers the details of pre-installation concerns for space, flooring, operating environment, electrical power, and security. Extensive lab exercises reinforce the skills needed to install the hardware and begin operation of the NonStop-UX operating system. After completing this course, students are able to perform NonStop-UX operating system software installations, for both new installations and upgrades.

(Note that this course does not cover specific administration of the Veritas storage subsystem. For general administration of the Veritas subsystem, users should complete the NonStop-UX Veritas Administration course before enrolling in this course.)

Integrity Systems Support, SVR3 to SVR4 Upgrade

This nine-day lecture-and-lab course is an update of the Integrity Systems Support SVR3 version of the Systems Maintenance course. This course covers the maintenance and installation of the Integrity S300 and S1300 models, with a strong emphasis on their differences. Students review UNIX System V, Release 4, administrative functions and practice the troubleshooting of hardware and administrative problems. After completing the course, users are familiar with the Tandem enhancements that differentiate the Integrity systems from other available UNIX systems.

NonStop IPX/SPX Product Overview

This independent study program (ISP) consists of a short videotape on the NonStop IPX/SPX product, Tandem's connectivity vehicle to Novell LANs. After completing this ISP, users are familiar with the principles of NonStop IPX/SPX, its architecture, and related planning and management issues.

The Technical Information and Education department is an annotated list of new Tandem education courses and consulting and information services, as well as other technical information of interest to Tandem users.

SNAX/XF Token Ring System Management

This independent study program (ISP) teaches students how to plan, configure, and manage the Tandem SNAX/XF Token Ring product in a token ring LAN environment. Students learn about Tandem's strategic commitment to the token ring market, and about the availability of SNAX/XF Token Ring LAN background and planning information for Tandem's SNAX users. Students also engage in a technical overview of the IEEE 802.2 and 802.5 specifications as they apply to the implementation in Tandem's SNAX/XF Token Ring LAN product. After completing this ISP, users have an understanding of token ring LAN management methodologies and are able to conduct detailed SNAX/XF Token Ring PTrace protocol analysis.

Token Ring Architecture and Products

With this independent study program (ISP), students learn the fundamentals of the IEEE 802.5 token ring architecture and related products. Students become familiar with the concepts and implementations of the Physical and MAC layers associated with the token ring network. After completing this ISP, users understand the operation of the token ring protocol and internetworking operation.

Tandem Education on the Internet

Tandem Education made its Internet debut on the World Wide Web in August 1994. The United States customer catalog and schedule are now electronically available to all Tandem users. Clicking on highlighted words, phrases, or pictures will take users to up-to-the-moment information 24 hours a day, seven days a week.

To access the Tandem Education information, users need to have access to the Internet. Users should check with their Internet provider to find out how to access the World Wide Web.

TandemSystemsReview*Index*

The *Tandem Journal* became the *Tandem Systems Review* in February 1985. Four issues of the *Tandem Journal* were published:

- Volume 1, No. 1 Fall 1983
- Volume 2, No. 1 Winter 1984
- Volume 2, No. 2 Spring 1984
- Volume 2, No. 3 Summer 1984

As of this issue, 27 issues of the *Tandem Systems Review* have been published:

- Volume 1, No. 1 Feb. 1985
- Volume 1, No. 2 June 1985
- Volume 2, No. 1 Feb. 1986
- Volume 2, No. 2 June 1986
- Volume 2, No. 3 Dec. 1986
- Volume 3, No. 1 March 1987
- Volume 3, No. 2 Aug. 1987
- Volume 4, No. 1 Feb. 1988
- Volume 4, No. 2 July 1988
- Volume 4, No. 3 Oct. 1988
- Volume 5, No. 1 April 1989
- Volume 5, No. 2 Sept. 1989
- Volume 6, No. 1 March 1990
- Volume 6, No. 2 Oct. 1990
- Volume 7, No. 1 April 1991
- Volume 7, No. 2 Oct. 1991
- Volume 8, No. 1 Spring 1992
- Volume 8, No. 2 Summer 1992
- Volume 8, No. 3 Fall 1992
- Volume 9, No. 1 Winter 1993
- Volume 9, No. 2 Spring 1993
- Volume 9, No. 3 Summer 1993
- Volume 9, No. 4 Fall 1993
- Volume 10, No. 1 Jan. 1994
- Volume 10, No. 2 April 1994
- Volume 10, No. 3 July 1994
- Volume 10, No. 4 Oct. 1994

The articles published in all 31 issues are arranged by subject below. (*Tandem Journal* is abbreviated as TJ and *Tandem Systems Review* as TSR.) A second index, arranged by product, is also provided.

Index by Subject

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
APPLICATION DEVELOPMENT AND LANGUAGES					
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
An Overview of Client/Server Computing on Tandem Systems	H. Cooperstein	TSR	8,3	Fall 1992	89803
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
Application Code Conversion for D-Series Systems	K. Liu	TSR	9,2	Spring 1993	89805
Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility	D. Broyles	TSR	9,3	Summer 1993	89806
Automating Call Centers With CAM	W. Choi	TSR	10,2	April 1994	104398
Basic Uses and New Features of Extended GDS	A. Hotea	TSR	10,1	Jan. 1994	104396
Debugging TACL Code	L. Palmer	TSR	4,2	July 1988	13693
Designing and Implementing a Graphical User Interface	S. Wolfe	TSR	9,3	Summer 1993	89806
Designing Client/Server Applications for OLTP on Guardian 90 Systems	W. Culman	TSR	8,3	Fall 1992	89803
Extending the Client/Server Model With Object-Oriented Technology	T. Rohner	TSR	10,1	Jan. 1994	104396
Implementing Client/Server Using RSC	M. Iem, T. Kocher	TSR	8,3	Fall 1992	89803
Implementing Decision Support Systems	W. Pearson	TSR	10,4	Oct. 1994	104402
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
New TAL Features	C. Lu, J. Murayama	TSR	2,2	June 1986	83837
PATHFINDER–An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
APPLICATION DEVELOPMENT AND LANGUAGES *(cont.)*					
PATHWAY IDS: A Message-level Interface to Devices and Processes	M. Anderton, M. Noonan	TSR	2,2	June 1986	83937
The RESPOND OLTP Business Management System for Manufacturing	H. Bolling, W. Bronson	TSR	9,1	Winter 1993	89804
State-of-the-Art C Compiler	E. Kit	TSR	2,2	June 1986	83937
TACL, Tandem's New Extensible Command Language	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
Tandem's New COBOL85	D. Nelson	TSR	2,1	Feb. 1986	83936
The DAL Server: Client/Server Access to Tandem Databases	W. Schlansky, J. Schrengohst	TSR	9,1	Winter 1993	89804
The ENABLE Program Generator for Multifile Applications	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
TMF and the Multi-Threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
Writing a Command Interpreter	D. Wong	TSR	1,2	June 1985	83935
CLIENT/SERVER					
An Overview of Client/Server Computing on Tandem Systems	H. Cooperstein	TSR	8,3	Fall 1992	89803
Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility	D. Broyles	TSR	9,3	Summer 1993	89806
Client/Server Availability	A. Wood	TSR	10,2	April 1994	104398
Designing and Implementing a Graphical User Interface	S. Wolfe	TSR	9,3	Summer 1993	89806
Designing Client/Server Applications for OLTP on Guardian 90 Systems	W. Culman	TSR	8,3	Fall 1992	89803
Extending the Client/Server Model With Object-Oriented Technology	T. Rohner	TSR	10,1	Jan. 1994	104396
Gateways to NonStop SQL	D. Slutz	TSR	6,2	Oct. 1990	46987
Implementing Client/Server Using RSC	M. Iem, T. Kocher	TSR	8,3	Fall 1992	89803
NonStop ODBC Server	H. Mahbod, D. Slutz	TSR	10,3	July 1994	104400
The DAL Server: Client/Server Access to Tandem Databases	W. Schlansky, J. Schrengohst	TSR	9,1	Winter 1993	89804
DATA COMMUNICATIONS					
An Overview of SNAX/CDF	M. Turner	TSR	5,2	Sept. 1989	28152
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
Basic Uses and New Features of Extended GDS	A. Hotea	TSR	10,1	Jan. 1994	104396
Changes in FOX	N. Donde	TSR	1,2	June 1985	83935
Connecting Terminals and Workstations to Guardian 90 Systems	E. Siegel	TSR	8,2	Summer 1992	69848
Expand High-Performance Solutions	D. Smith	TSR	9,3	Summer 1993	89806
Introduction to MULTILAN	A. Coyle	TSR	4,1	Feb. 1988	11078
Overview of the MULTILAN Server	A. Rowe	TSR	4,1	Feb. 1988	11078
SNAX/APC: Tandem's New SNA Software for Distributed Processing	B. Grantham	TSR	3,1	March 1987	83939
SNAX/HLS: An Overview	S. Saltwick	TSR	1,2	June 1985	83935
TLAM: A Connectivity Option for Expand	K. MacKenzie	TSR	7,1	April 1991	46988
Using the MULTILAN Application Interfaces	M. Berg, A. Rowe	TSR	4,1	Feb. 1988	11078

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
DATA MANAGEMENT					
A Comparison of the B00 DP1 and DP2 Disc Processes	T. Schachter	TSR	1,2	June 1985	83935
A New Hash-Based Join Algorithm for NonStop SQL/MP	H. Zeller	TSR	10,3	July 1994	104400
An Overview of NonStop SQL/MP	F. Ho, R. Jain, J. Troisi	TSR	10,3	July 1994	104400
An Overview of NonStop SQL Release 2	M. Pong	TSR	6,2	Oct. 1990	46987
Batch Processing in Online Enterprise Computing	T. Keefauver	TSR	6,2	Oct. 1990	46987
Concurrency Control Aspects of Transaction Design	W. Senf	TSR	6,1	March 1990	32968
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
DP1-DP2 File Conversion: An Overview	J. Tate	TSR	2,1	Feb. 1986	83936
Determining FCP Conversion Time	J. Tate	TSR	2,1	Feb. 1986	83936
DP2's Efficient Use of Cache	T. Schachter	TSR	1,2	June 1985	83935
DP2 Highlights	K. Carlyle, L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files	T. Schachter	TSR	1,2	June 1985	83935
Enhancing Availability, Manageability, and Performance With NonStopTM/MP	M. Chandra, D. Eicher	TSR	10,3	July 1994	104400
Gateways to NonStop SQL	D. Slutz	TSR	6,2	Oct. 1990	46987
High-Performance SQL Through Low-Level System Integration	A. Borr	TSR	4,2	July 1988	13693
Improvements in TMF	T. Lemberger	TSR	1,2	June 1985	83935
Issues in DSS Database Design	R. Glasstone	TSR	10,4	Oct. 1994	104402
Late Binding and High Availability Compilation in NonStop SQL/MP	S. Sharma	TSR	10,4	Oct. 1994	104402
NetBatch: Managing Batch Processing on Tandem Systems	D. Wakashige	TSR	5,1	April 1989	18662
NetBatch-Plus: Structuring the Batch Environment	G. Earle, D. Wakashige	TSR	6,1	March 1990	32986
NonStop Availability and Database Configuration Operations	J. Troisi	TSR	10,3	July 1994	104400
NonStop ODBC Server	H. Mahbod, D. Slutz	TSR	10,3	July 1994	104400
NonStop SQL: The Single Database Solution	J. Cassidy, T. Kocher	TSR	5,2	Sept. 1989	28152
NonStop SQL Data Dictionary	R. Holbrook, D. Tsou	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Basic Concepts	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Query Optimization and User Influence	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Reliability	C. Fenner	TSR	4,2	July 1988	13693
Online Information Processing	J. Viescas	TSR	9,1	Winter 1993	89804
Online Reorganization of Key-Sequenced Tables and Files	G. Smith	TSR	6,2	Oct. 1990	46987
Optimizing Batch Performance	T. Keefauver	TSR	5,2	Sept. 1989	28152
Overview of NonStop SQL	H. Cohen	TSR	4,2	July 1988	13693
Parallelism in NonStop SQL Release 2	M. Moore, A. Sodhi	TSR	6,2	Oct. 1990	46987
The NonStop SQL Release 2 Benchmark	S. Englert, J. Gray, T. Kocher, P. Shah	TSR	6,2	Oct. 1990	46987
The Outer Join in NonStop SQL	J. Vaishnav	TSR	6,2	Oct. 1990	46987
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
Tandem's NonStop SQL Benchmark	Tandem Performance Group	TSR	4,1	Feb. 1988	11078
The TRANSFER Delivery System for Distributed Applications	S. Van Pelt	TJ	2,2	Spring 1984	83932
TMF Autorollback: A New Recovery Feature	M. Pong	TSR	1,1	Feb. 1985	83934
DECISION SUPPORT SYSTEMS					
An Overview of NonStop SQL/MP	F. Ho, R. Jain, J. Troisi	TSR	10,3	July 1994	104400
Implementing Decision Support Systems	W. Pearson	TSR	10,4	Oct. 1994	104402
Issues in DSS Database Design	R. Glasstone	TSR	10,4	Oct. 1994	104402
NonStop ODBC Server	H. Mahbod, D. Slutz	TSR	10,3	July 1994	104400
Online Information Processing	J. Viescas	TSR	9,1	Winter 1993	89804
The DAL Server: Client/Server Access to Tandem Databases	W. Schlansky, J. Schrengohst	TSR	9,1	Winter 1993	89804
The RESPOND OLTP Business Management System for Manufacturing	H. Bolling, W. Bronson	TSR	9,1	Winter 1993	89804

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
OBJECT-ORIENTED TECHNOLOGY					
Extending the Client/Server Model With Object-Oriented Technology	T. Rohner	TSR	10,1	Jan. 1994	104396
OPERATING SYSTEMS					
Application Code Conversion for D-Series Systems	K. Liu	TSR	9,2	Spring 1993	89805
Highlights of the B00 Software Release	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
Increased Code Space	A. Jordan	TSR	1,2	June 1985	83935
Managing System Time Under GUARDIAN 90	E. Nellen	TSR	2,1	Feb. 1986	83936
Migration Planning for D-Series Systems	S. Kuukka	TSR	9,2	Spring 1993	89805
New GUARDIAN 90 Time-keeping Facilities	E. Nellen	TSR	1,2	June 1985	83935
New Process-timing Features	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Overview of the C00 Release	L. Marks	TSR	4,1	Feb. 1988	11078
Overview of the D-Series Guardian 90 Operating System	W. Bartlett	TSR	9,2	Spring 1993	89805
Overview of the NonStop-UX Operating System for the Integrity S2	P. Norwood	TSR	7,1	April 1991	46988
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Approach	A. Borr	TSR	1,2	June 1985	83935
The GUARDIAN Message System and How to Design for It	M. Chandra	TSR	1,1	Feb. 1985	83934
The NonStop Himalaya K10000 Interprocessor Bus	R. Jardine, S. Hamilton, K. Krishnakumar	TSR	10,2	April 1994	104398
The Tandem Global Update Protocol	R. Carr	TSR	1,2	June 1985	83935
PERFORMANCE AND CAPACITY PLANNING					
A Performance Retrospective	P. Oleinick, P. Shah	TSR	2,3	Dec. 1986	83938
Buffering for Better Application Performance	R. Mattran	TSR	2,1	Feb. 1986	83936
Capacity Planning Concepts	R. Evans	TSR	2,3	Dec. 1986	83938
Capacity Planning With TCM	W. Highleyman	TSR	7,2	Oct. 1991	65248
C00 TMDS Performance	J. Mead	TSR	4,1	Feb. 1988	11078
Credit-authorization Benchmark for High Performance and Linear Growth	T. Chmiel, T. Houy	TSR	2,1	Feb. 1986	83936
Debugging Accelerated Programs on TNS/R Systems	D. Cressler	TSR	8,1	Spring 1992	65250
DP2 Performance	J. Enright	TSR	1,2	June 1985	83935
Estimating Host Response Time in a Tandem System	H. Horwitz	TSR	4,3	Oct. 1988	15748
Expand High-Performance Solutions	D. Smith	TSR	9,3	Summer 1993	89806
FASTSORT: An External Sort Using Parallel Processing	J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan	TSR	2,3	Dec. 1986	83938
Getting Optimum Performance from Tandem Tape Systems	A. Khatri	TSR	2,3	Dec. 1986	83938
How to Set Up a Performance Data Base with MEASURE and ENFORM	M. King	TSR	2,3	Dec. 1986	83938
Implementing a Systems Management Improvement Program	J. Dagenais	TSR	9,4	Fall 1993	89807
Improved Performance for BACKUP2 and RESTORE2	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Improving Performance on TNS/R Systems With the Accelerator	M. Blanchet	TSR	8,1	Spring 1992	65250
MEASURE: Tandem's New Performance Measurement Tool	D. Dennison	TSR	2,3	Dec. 1986	83938
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Message System Performance Enhancements	D. Kinkade	TSR	2,3	Dec. 1986	83938
Message System Performance Tests	S. Uren	TSR	2,3	Dec. 1986	83938
Network Design Considerations	J. Evjen	TSR	5,2	Sept. 1989	28152
NonStop NET/MASTER: Configuration and Performance Guidelines	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop VLX Performance	J. Enright	TSR	2,3	Dec. 1986	83938
Optimizing Sequential Processing on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
Pathway TCP Enhancements for Application Run-Time Support	R. Vannucci	TSR	7,1	April 1991	46988

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
PERFORMANCE AND CAPACITY PLANNING *(cont.)*					
Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2	S. Englert, J. Gray	TSR	6,2	Oct. 1990	46987
Performance Considerations for Application Processes	R. Glasstone	TSR	2,3	Dec. 1986	83938
Performance Measurements of an ATM Network Application	N. Cabell, D. Mackie	TSR	2,3	Dec. 1986	83938
Predicting Response Time in On-line Transaction Processing Systems	A. Khatri	TSR	2,2	June 1986	83937
RDF Enhancements for High Availability and Performance	M. Mosher	TSR	10,3	July 1994	104400
Sizing Cache for Applications that Use B-series DP1 and TMF	P. Shah	TSR	2,2	June 1986	83937
Sizing the Spooler Collector Data File	H. Norman	TSR	4,1	Feb. 1988	11978
Tandem's 5200 Optical Storage Facility: Performance and Optimization Considerations	S. Coleman	TSR	5,1	April 1989	18662
Tandem's Approach to Fault Tolerance	B. Ball, W. Bartlett, S. Thompson	TSR	4,1	Feb. 1988	11078
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
The PATHWAY TCP: Performance and Tuning	J. Vatz	TSR	1,1	Feb. 1985	83934
The Performance Characteristics of Tandem NonStop Systems	J. Day	TJ	1,1	Fall 1983	83930
Understanding PATHWAY Statistics	R. Wong	TJ	2,2	Spring 1984	83932
PERIPHERALS					
5120 Tape Subsystem Recording Technology	W. Phillips	TSR	3,2	Aug. 1987	83940
An Introduction to DYNAMITE Workstation Host Integration	S. Kosinski	TSR	1,2	June 1985	83935
Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility	D. Broyles	TSR	9,3	Summer 1993	89806
Data-Encoding Technology Used in the XL8 Storage Facility	D. S. Ng	TSR	2,2	June 1986	83937
Data-Window Phase-Margin Analysis	A. Painter, H. Pham, H. Thomas	TSR	2,2	June 1986	83937
Introducing the 3207 Tape Controller	S. Chandran	TSR	1,2	June 1985	83935
Peripheral Device Interfaces	J. Blakkan	TSR	3,2	Aug. 1987	83940
Plated Media Technology Used in the XL8 Storage Facility	D.S. Ng	TSR	2,2	June 1986	83937
Streaming Tape Drives	J. Blakkan	TSR	3,2	Aug. 1987	83940
Terminal Selection	E. Siegel	TSR	8,2	Summer 1992	69848
The 5200 Optical Storage Facility: A Hardware Perspective	A. Patel	TSR	5,1	April 1989	18662
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
The DYNAMITE Workstation: An Overview	G. Smith	TSR	1,2	June 1985	83935
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
The Role of Optical Storage in Information Processing	L. Sabaroff	TSR	3,2	Aug. 1987	83940
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage	M. Whiteman	TSR	1,2	June 1985	83935

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
PROCESSORS					
Fault Tolerance in the NonStop Cyclone System	S. Chan, R. Jardine	TSR	7,1	April 1991	46988
A Hardware Overview of the NonStop Himalaya K10000 Server	C. Kong	TSR	10,1	Jan. 1994	104396
NonStop CLX: Optimized for Distributed On-Line Transaction Processing	D. Lenoski	TSR	5,1	April 1989	18662
NonStop VLX Hardware Design	M. Brown	TSR	2,3	Dec. 1986	83938
Overview of Tandem NonStop Series/RISC Systems	L. Faby, R. Mateosian	TSR	8,1	Spring 1992	65250
The High-Performance NonStop TXP Processor Transaction Processing	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop Himalaya K10000 Interprocessor Bus	R. Jardine, S. Hamilton, K. Krishnakumar	TSR	10,2	April 1994	104398
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
The VLX: A Design for Serviceability	J. Allen, R. Boyle	TSR	3,1	March 1987	83939
SECURITY					
Dial-In Security Considerations	P. Grainger	TSR	7,2	Oct. 1991	65248
Distributed Protection with SAFEGUARD	T. Chou	TSR	2,2	June 1986	83937
Enhancing System Security With Safeguard	C. Gaydos	TSR	7,1	April 1991	46988
SYSTEM CONNECTIVITY					
Basic Uses and New Features of Extended GDS	A. Hotea	TSR	10,1	Jan. 1994	104396
Building Open Systems Interconnection with OSI/AS and OSI/TS	R. Smith	TSR	6,1	March 1990	32986
Connecting Terminals and Workstations to Guardian 90 Systems	E. Siegel	TSR	8,2	Summer 1992	69848
Implementing Client/Server Using RSC	M. Iem, T. Kocher	TSR	8,3	Fall 1992	89803
Network Design Considerations	J. Evjen	TSR	5,2	Sept. 1989	28152
Terminal Connection Alternatives for Tandem Systems	J. Simonds	TSR	5,1	April 1989	18662
Terminal Selection	E. Siegel	TSR	8,2	Summer 1992	69848
The OSI Model: Overview, Status, and Current Issues	A. Dunn	TSR	5,1	April 1989	18662
SYSTEM MANAGEMENT					
Configuring Tandem Disk Subsystems	S. Sitler	TSR	2,3	Dec. 1986	83938
Data Replication in Tandem's Distributed Name Service	T. Eastep	TSR	4,3	Oct. 1988	15748
Enhancements to TMDS	L. White	TSR	3,2	Aug. 1987	83940
Event Management Service Design and Implementation	H. Jordan, R. McKee, R. Schuet	TSR	4,3	Oct. 1988	15748
Implementing a Systems Management Improvement Program	J. Dagenais	TSR	9,4	Fall 1993	89807
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
Introducing TMDS, Tandem's New On-line Diagnostic System	J. Troisi	TSR	1,2	June 1985	83935
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Network Statistics System	M. Miller	TSR	4,3	Oct. 1988	15748
NonStop NET/MASTER: Configuration and Performance Guidelines	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop NET/MASTER: Event Management Architecture	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop NET/MASTER: Event Processing Costs and Sizing Calculations	M. Stockton	TSR	9,4	Fall 1993	89807
Overview of DSM	P. Homan, B. Malizia, E. Reisner	TSR	4,3	Oct. 1988	15748
SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface	T. Lawson	TSR	4,3	Oct. 1988	15748
RDF: An Overview	J. Guerrero	TSR	7,2	Oct. 1991	65248
RDF Enhancements for High Availability and Performance	M. Mosher	TSR	10,3	July 1994	104400
RDF Synchronization	F. Jongma, W. Senf	TSR	8,2	Summer 1992	69848

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
SYSTEM MANAGEMENT *(cont.)*					
Tandem's Subsystem Programmatic Interface	G. Tom	TSR	4,3	Oct. 1988	15748
Using FOX to Move a Fault-tolerant Application	C. Breighner	TSR	1,1	Feb. 1985	83934
Using the Subsystem Programmatic Interface and Event Management Services	K. Stobie	TSR	4,3	Oct. 1988	15748
VIEWPOINT Operations Console Facility	R. Hansen, G. Stewart	TSR	4,3	Oct. 1988	15748
VIEWSYS: An On-line System-resource Monitor	D. Montgomery	TSR	1,2	June 1985	83935
Writing Rules for Automated Operations	J. Collins	TSR	7,2	Oct. 1991	65248
UTILITIES					
Enhancements to PS MAIL	R. Funk	TSR	3,1	March 1987	83939

Index by Product

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
3207 TAPE CONTROLLER					
Introducing the 3207 Tape Controller	S. Chandran	TSR	1,2	June 1985	83935
5120 TAPE SUBSYSTEM					
5120 Tape Subsystem Recording Technology	W. Phillips	TSR	3,2	Aug. 1987	83940
5200 OPTICAL STORAGE					
Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility	D. Broyles	TSR	9,3	Summer 1993	89806
Tandem's 5200 Optical Storage Facility: Performance and Optimization Considerations	S. Coleman	TSR	5,1	April 1989	18662
The 5200 Optical Storage Facility: A Hardware Perspective	A. Patel	TSR	5,1	April 1989	18662
The Role of Optical Storage in Information Processing	L. Sabaroff	TSR	4,1	Feb. 1988	11078
6100 COMMUNICATIONS SUBSYSTEM					
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
6530 TERMINAL					
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
6600 AND TCC6820 COMMUNICATIONS CONTROLLERS					
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
BASIC					
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
C					
State-of-the-art C Compiler	E. Kit	TSR	2,2	June 1986	83937
CAM					
Automating Call Centers With CAM	W. Choi	TSR	10,2	April 1994	104398
CIS					
Customer Information Service	J. Massucco	TSR	3,1	March 1987	83939
CLX					
NonStop CLX: Optimized for Distributed On-Line Transaction Processing	D. Lenoski	TSR	5,1	April 1989	18662
COBOL85					
Tandem's New COBOL85	D. Nelson	TSR	2,1	Feb. 1986	83936
COMINT (CI)					
Writing a Command Interpreter	D. Wong	TSR	1,2	June 1985	83935
CYCLONE					
Fault Tolerance in the NonStop Cyclone System	S. Chan, R. Jardine	TSR	7,1	April 1991	46988
DAL SERVER					
The DAL Server: Client/Server Access to Tandem Databases	W. Schlansky, J. Schrengohst	TSR	9,1	Winter 1993	89804

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
DP1 AND DP2					
A Comparison of the B00 DP1 and DP2 Disc Processes	T. Schachter	TSR	1,2	June 1985	83935
Determining FCP Conversion Time	J. Tate	TSR	2,1	Feb. 1986	83936
DP1-DP2 File Conversion: An Overview	J. Tate	TSR	2,1	Feb. 1986	83936
DP2 Highlights	K. Carlyle, L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files	T. Schachter	TSR	1,2	June 1985	83935
DP2 Performance	J. Enright	TSR	1,2	June 1985	83935
DP2's Efficient Use of Cache	T. Schachter	TSR	1,2	June 1985	83935
Sizing Cache for Applications that Use B-series DP1 and TMF	P. Shah	TSR	2,2	June 1986	83937
DSM					
Data Replication in Tandem's Distributed Name Service	T. Eastep	TSR	4,3	Oct. 1988	15748
Event Management Service Design and Implementation	H. Jordan, R. McKee, R. Schuet	TSR	4,3	Oct. 1988	15748
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Network Statistics System	M. Miller	TSR	4,3	Oct. 1988	15748
Overview of DSM	P. Homan, B. Malizia, E. Reisner	TSR	4,3	Oct. 1988	15748
SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface	T. Lawson	TSR	4,3	Oct. 1988	15748
Tandem's Subsystem Programmatic Interface	G. Tom	TSR	4,3	Oct. 1988	15748
Using the Subsystem Programmatic Interface and Event Management Services	K. Stobie	TSR	4,3	Oct. 1988	15748
VIEWPOINT Operations Console Facility	R. Hansen, G. Stewart	TSR	4,3	Oct. 1988	15748
Writing Rules for Automated Operations	J. Collins	TSR	7,2	Oct. 1991	65248
DYNAMITE					
An Introduction to DYNAMITE Workstation Host Integration	S. Kosinski	TSR	1,2	June 1985	83935
The DYNAMITE Workstation: An Overview	G. Smith	TSR	1,2	June 1985	83935
ENABLE					
The ENABLE Program Generator for Multifile Applications	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
ENCOMPASS					
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
ENCORE					
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
ENSCRIBE					
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
EXPAND					
Expand High-Performance Solutions	D. Smith	TSR	9,3	Summer 1993	89806
FASTSORT					
FASTSORT: An External Sort Using Parallel Processing	J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan	TSR	2,3	Dec. 1986	83938

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
FOX					
Changes in FOX	N. Donde	TSR	1,2	June 1985	83935
Using FOX to Move a Fault-tolerant Application	C. Breighner	TSR	1,1	Feb. 1985	83934
FUP					
Online Reorganization of Key-Sequenced Tables and Files	G. Smith	TSR	6,2	Oct. 1990	46987
GDS					
Basic Uses and New Features of Extended GDS	A. Hotea	TSR	10,1	Jan. 1994	104396
GUARDIAN 90					
Application Code Conversion for D-Series Systems	K. Liu	TSR	9,2	Spring 1993	89805
B00 Software Manuals	S. Olds	TSR	1,2	June 1985	83935
C00 Software Manuals	E. Levi	TSR	4,1	Feb. 1988	11078
Highlights of the B00 Software Release	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
Improved Performance for BACKUP2 and RESTORE2	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Increased Code Space	A. Jordan	TSR	1,2	June 1985	83935
Managing System Time Under GUARDIAN 90	E. Nellen	TSR	2,1	Feb. 1986	83936
Message System Performance Enhancements	D. Kinkade	TSR	2,3	Dec. 1986	83938
Message System Performance Tests	S. Uren	TSR	2,3	Dec. 1986	83938
Migration Planning for D-Series Systems	S. Kuukka	TSR	9,2	Spring 1993	89805
New GUARDIAN 90 Time-keeping Facilities	E. Nellen	TSR	1,2	June 1985	83935
New Process-timing Features	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Overview of the C00 Release	L. Marks	TSR	4,1	Feb. 1988	11078
Overview of the D-Series Guardian 90 Operating System	W. Bartlett	TSR	9,2	Spring 1993	89805
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Multiprocessor Approach	A. Borr	TSR	1,2	June 1985	83935
Tandem's Approach to Fault Tolerance	B. Ball, W. Bartlett, S. Thompson	TSR	4,1	Feb. 1988	11078
The GUARDIAN Message System and How to Design for It	M. Chandra	TSR	1,1	Feb. 1985	83934
The Tandem Global Update Protocol	R. Carr	TSR	1,2	June 1985	83935
HIMALAYA					
A Hardware Overview of the NonStop Himalaya K10000 Server	C. Kong	TSR	10,1	Jan. 1994	104396
The NonStop Himalaya K10000 Interprocessor Bus	R. Jardine, S. Hamilton, K. Krishnakumar	TSR	10,2	April 1994	104398
INTEGRITY S2					
Overview of the NonStop-UX Operating System for the Integrity S2	P. Norwood	TSR	7,1	April 1991	46988
MEASURE					
How to Set Up a Performance Data Base with MEASURE and ENFORM	M. King	TSR	2,3	Dec. 1986	83938
MEASURE: Tandem's New Performance Measurement Tool	D. Dennison	TSR	2,3	Dec. 1986	83938
MULTILAN					
Introduction to MULTILAN	A. Coyle	TSR	4,1	Feb. 1988	11078
Overview of the MULTILAN Server	A. Rowe	TSR	4,1	Feb. 1988	11078
Using the MULTILAN Application Interfaces	M. Berg, A. Rowe	TSR	4,1	Feb. 1988	11078

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
NETBATCH-PLUS					
NetBatch: Managing Batch Processing on Tandem Systems	D. Wakashige	TSR	5,1	April 1989	18662
NetBatch-Plus: Structuring the Batch Environment	G. Earle, D. Wakashige	TSR	6,1	March 1990	32986
NONSTOP NET/MASTER					
NonStop NET/MASTER: Configuration and Performance Guidelines	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop NET/MASTER: Event Management Architecture	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop NET/MASTER: Event Processing Costs and Sizing Calculations	M. Stockton	TSR	9,4	Fall 1993	89807
NONSTOP ODBC SERVER					
NonStop ODBC Server	H. Mahbod, D. Slutz	TSR	10,3	July 1994	104400
NONSTOP SQL/MP					
A New Hash-Based Join Algorithm for NonStop SQL/MP	H. Zeller	TSR	10,3	July 1994	104400
An Overview of NonStop SQL/MP	F. Ho, R. Jain, J. Troisi	TSR	10,3	July 1994	104400
An Overview of NonStop SQL Release 2	M. Pong	TSR	6,2	Oct. 1990	46987
Concurrency Control Aspects of Transaction Design	W. Senf	TSR	6,1	March 1990	32986
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
Gateways to NonStop SQL	D. Slutz	TSR	6,2	Oct. 1990	46987
High-Performance SQL Through Low-Level System Integration	A. Borr	TSR	4,2	July 1988	13693
Late Binding and High Availability Compilation in NonStop SQL/MP	S. Sharma	TSR	10,4	Oct. 1994	104402
NonStop Availability and Database Configuration Operations	J. Troisi	TSR	10,3	July 1994	104400
NonStop SQL Data Dictionary	R. Holbrook, D. Tsou	TSR	4,2	July 1988	13693
NonStop SQL: The Single Database Solution	J. Cassidy, T. Kocher	TSR	5,2	Sept. 1989	28152
NonStop SQL Optimizer: Basic Concepts	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Query Optimization and User Influence	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Reliability	C. Fenner	TSR	4,2	July 1988	13693
Overview of NonStop SQL	H. Cohen	TSR	4,2	July 1988	13693
Parallelism in NonStop SQL Release 2	M. Moore, A. Sodhi	TSR	6,2	Oct. 1990	46987
Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2	S. Englert, J. Gray	TSR	6,2	Oct. 1990	46987
Tandem's NonStop SQL Benchmark	Tandem Performance Group	TSR	4,1	Feb. 1988	11078
The NonStop SQL Release 2 Benchmark	S. Englert, J. Gray, T. Kocher, P. Shah	TSR	6,2	Oct. 1990	46987
The Outer Join in NonStop SQL	J. Vaishnav	TSR	6,2	Oct. 1990	46987
NONSTOP TM/MP					
Improvements in TMF	T. Lemberger	TSR	1,2	June 1985	83935
Enhancing Availability, Manageability, and Performance With NonStop TM/MP	M. Chandra, D. Eicher	TSR	10,3	July 1994	104400
TMF and the Multi-Threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
TMF Autorollback: A New Recovery Feature	M. Pong	TSR	1,1	Feb. 1985	83934
OSI					
Building Open Systems Interconnection with OSI/AS and OSI/TS	R. Smith	TSR	6,1	March 1990	32986
The OSI Model: Overview, Status, and Current Issues	A. Dunn	TSR	5,1	April 1989	18662
PATHFINDER					
PATHFINDER–An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
PATHWAY					
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
PATHWAY IDS: A Message-level Interface to Devices and Processes	M. Anderton, M. Noonan	TSR	2,2	June 1986	83937
Pathway TCP Enhancements for Application Run-Time Support	R. Vannucci	TSR	7,1	April 1991	46988
The PATHWAY TCP: Performance and Tuning	J. Vatz	TSR	1,1	Feb. 1985	83934
Understanding PATHWAY Statistics	R. Wong	TJ	2,2	Spring 1984	83932
POET					
Designing Client/Server Applications for OLTP on Guardian 90 Systems	W. Culman	TSR	8,3	Fall 1992	89803
PS MAIL					
Enhancements to PS MAIL	R. Funk	TSR	3,1	March 1987	83939
RDF					
RDF: An Overview	J. Guerrero	TSR	7,2	Oct. 1991	65248
RDF Enhancements for High Availability and Performance	M. Mosher	TSR	10,3	July 1994	104400
RDF Synchronization	F. Jongma, W. Senf	TSR	8,2	Summer 1992	69848
RESPOND					
The RESPOND OLTP Business Management System for Manufacturing	H. Bolling, W. Bronson	TSR	9,1	Winter 1993	89804
RSC					
Implementing Client/Server Using RSC	M. Iem, T. Kocher	TSR	8,3	Fall 1992	89803
SAFEGUARD					
Dial-In Security Considerations	P. Grainger	TSR	7,2	Oct. 1991	65248
Distributed Protection with SAFEGUARD	T. Chou	TSR	2,2	June 1986	83937
Enhancing System Security With Safeguard	C. Gaydos	TSR	7,1	April 1991	46988
SNAX					
An Overview of SNAX/CDF	M. Turner	TSR	5,2	Sept. 1989	28152
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
SNAX/APC: Tandem's New SNA Software for Distributed Processing	B. Grantham	TSR	3,1	March 1987	83939
SNAX/HLS: An Overview	S. Saltwick	TSR	1,2	June 1985	83935
SPOOLER					
Sizing the Spooler Collector Data File	H. Norman	TSR	4,1	Feb. 1988	11078
TACL					
Debugging TACL Code	L. Palmer	TSR	4,2	July 1988	13693
TACL, Tandem's New Extensible Command Language	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
TAL					
New TAL Features	C. Lu, J. Murayama	TSR	2,2	June 1986	83837
TCM					
Capacity Planning With TCM	W. Highleyman	TSR	7,2	Oct. 1991	65248
TLAM					
TLAM: A Connectivity Option for Expand	K. MacKenzie	TSR	7,1	April 1991	46988
TMDS					
C00 TMDS Performance	J. Mead	TSR	4,1	Feb. 1988	11078
Enhancements to TMDS	L. White	TSR	3,2	Aug. 1987	83940
Introducing TMDS, Tandem's New On-line Diagnostic System	J. Troisi	TSR	1,2	June 1985	83935
TNS/R					
Debugging Accelerated Programs on TNS/R Systems	D. Cressler	TSR	8,1	Spring 1992	65250
Improving Performance on TNS/R Systems With the Accelerator	M. Blanchet	TSR	8,1	Spring 1992	65250
Overview of Tandem NonStop Series/RISC Systems	L. Faby, R. Mateosian	TSR	8,1	Spring 1992	65250

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
TRANSFER					
The TRANSFER Delivery System for Distributed Applications	S. Van Pelt	TJ	2,2	Spring 1984	83932
TXP					
The High-Performance NonStop TXP Processor	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
V8					
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage	M. Whiteman	TSR	1,2	June 1985	83935
VIEWSYS					
VIEWSYS: An On-line System-resource Monitor	D. Montgomery	TSR	1,2	June 1985	83935
VLX					
NonStop VLX Hardware Design	M. Brown	TSR	2,3	Dec. 1986	83938
NonStop VLX Performance	J. Enright	TSR	2,3	Dec. 1986	83938
The VLX: A Design for Serviceability	J. Allen, R. Boyle	TSR	3,1	March 1987	83939
XL8					
Data-encoding Technology Used in the XL8 Storage Facility	D. S. Ng	TSR	2,2	June 1986	83937
Plated Media Technology Used in the XL8 Storage Facility	D. S. Ng	TSR	2,2	June 1986	83937

TandemSystemsReview*ReaderSurvey*

The purpose of this questionnaire is to help the *Tandem Systems Review* staff select topics for publication. Postage is prepaid when mailed in the United States. Readers outside the U.S. should send their replies to their nearest Tandem sales office.

1. How useful is each article in this issue?

Product Update

01 ☐ Indispensable 02 ☐ Very 03 ☐ Somewhat 04 ☐ Not at all

Implementing Decision Support Systems

05 ☐ Indispensable 06 ☐ Very 07 ☐ Somewhat 08 ☐ Not at all

Issues in DSS Database Design

09 ☐ Indispensable 10 ☐ Very 11 ☐ Somewhat 12 ☐ Not at all

Late Binding and High Availability Compilation in NonStop SQL/MP

13 ☐ Indispensable 14 ☐ Very 15 ☐ Somewhat 16 ☐ Not at all

Technical Information and Education

17 ☐ Indispensable 18 ☐ Very 19 ☐ Somewhat 20 ☐ Not at all

2. I specifically would like to see more articles on (select one):

21 ☐ Overview discussions of new products and enhancements
22 ☐ Performance and tuning information
23 ☐ High-level overviews on Tandem's approach to solutions
24 ☐ Application design and customer profiles
25 ☐ Technical discussions of product internals
26 ☐ Strategic information and statements of direction
27 ☐ Other _____

3. Your title or position:

28 ☐ President, VP, Director 29 ☐ Systems analyst 30 ☐ System operator
31 ☐ MIS manager 32 ☐ Software developer 33 ☐ End user
34 ☐ Other _____

4. Your association with Tandem:

35 ☐ Tandem customer 36 ☐ Tandem employee 37 ☐ Third-party vendor 38 ☐ Consultant
39 ☐ Other _____

5. Comments

_____

_____

_____

_____

NAME

_____

COMPANY NAME

_____

ADDRESS

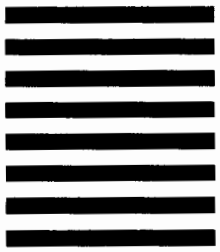
_____

10/94

▲ FOLD

▲ FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 482 CUPERTINO, CA U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

TANDEM SYSTEMS REVIEW
LOC 208-65
TANDEM COMPUTERS INCORPORATED
19333 VALLCO PARKWAY
CUPERTINO, CA 95014–9862

▼ FOLD

▼ FOLD

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-2599

Part No. 104402

10/94 Printed in USA