Pervasive.SQL 2000i

What's New in Pervasive.SQL 2000i

Pervasive.SQL 2000 Service Pack 3

Pervasive Software, Inc.
12365 Riata Trace Parkway
Building II
Austin, TX 78727 USA

Telephone: +1 512 231 6000 or 800 287 4383
Fax: +1 512 231 6010
E-Mail: info@pervasive.com
Web: http://www.pervasive.com

disclaimer

PERVASIVE SOFTWARE INC. LICENSES THE SOFTWARE AND DOCUMENTATION PRODUCT TO YOU OR YOUR COMPANY SOLELY ON AN "AS IS" BASIS AND SOLELY IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF THE ACCOMPANYING LICENSE AGREEMENT. PERVASIVE SOFTWARE INC. MAKES NO OTHER WARRANTIES WHATSOEVER, EITHER EXPRESS OR IMPLIED, REGARDING THE SOFTWARE OR THE CONTENT OF THE DOCUMENTATION; PERVASIVE SOFTWARE INC. HEREBY EXPRESSLY STATES AND YOU OR YOUR COMPANY ACKNOWLEDGES THAT PERVASIVE SOFTWARE INC. DOES NOT MAKE ANY WARRANTIES, INCLUDING, FOR EXAMPLE, WITH RESPECT TO MERCHANTABILITY, TITLE, OR FITNESS FOR ANY PARTICULAR PURPOSE OR ARISING FROM COURSE OF DEALING OR USAGE OF TRADE, AMONG OTHERS.

trademarks

Btrieve, Tango, Client/Server in a Box, and the Pervasive Software logo are registered trademarks of Pervasive Software Inc.

Built on Pervasive, Built on Pervasive Software, Extranet in a Box, Pervasive.SQL, Jtrieve, Plug n' Play Databases, SmartScout, Solution Network, Ultra-light Z-DBA, Z-DBA, ZDBA, UltraLight, MicroKernel Database Engine, and MicroKernel Database Architecture are trademarks of Pervasive Software Inc.

Microsoft, MS-DOS, Windows, Windows NT, Win32, Win32s, and Visual Basic are registered trademarks of Microsoft Corporation.

Windows 95 is a trademark of Microsoft Corporation.

NetWare and Novell are registered trademarks of Novell, Inc.

NetWare Loadable Module, NLM, Novell DOS, Transaction Tracking System, and TTS are trademarks of Novell, Inc.

All other company and product names are the trademarks or registered trademarks of their respective companies.

This product includes software developed by Powerdog Industries.
© Copyright 1994 Powerdog Industries. All rights reserved.

The ODBC Driver Manager for NetWare (ODBC.NLM) included in this product is based on the GNU iODBC software © Copyright 1995 by Ke Jin <kejin@empress.com> and was modified by Simba Technologies Inc. in June 1999.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License is included in your installation of Pervasive.SQL 2000 at \pvsw\doc\lesser.htm. If you cannot find this license, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. You may contact Pervasive Software Inc. using the contact information on the back cover of this manual.

What's New in Pervasive.SQL 2000i
March 2001
100-004067-002

Contents

About This Manual vii
Who Should Read This Manual viii
Manual Organization ix
Conventions x

1 What's New in Pervasive.SQL 2000i SP3 1-1
An Overview of New Features in Service Pack 3

List of New Features and Improvements 1-2
Programming Interfaces 1-3
Full Support for OLE DB IRowset and ICommand 1-3
Full Support for JDBC 2.0 1-3
Dynamic Cursors 1-3
Updated OEM/Partner Toolkit (PTK) 1-3
Enhanced Stat Extended (65) Operation 1-4
Performance and Reliability Enhancements 1-12
Client/Server Version Checking 1-12
Row Level Locking 1-12
SRDE Improvements 1-13
Expanded Operating System Support 1-15
Terminal Server Support 1-15
NSS Volume Support 1-15
Comma as Decimal Separator 1-16
Easier Installation and Configuration 1-19
Pervasive System Analyzer 1-19
All Size Configuration Settings Now in Bytes 1-19
Dynamic Configuration Settings 1-19
Updated Default Settings 1-20
No Dependence on PERVASIVE_PATH 1-23
Improved Workgroup Gateway Behavior 1-23
Improved Networking Support 1-29
Pervasive Auto-Reconnect 1-29
Improved SQL Support 1-31
Additional SQL Syntax 1-31
New and Enhanced Documentation 1-34

2 Pervasive System Analyzer (PSA) 2-1
An Overview of the New Diagnostic Utility in Pervasive.SQL 2000i SP3

Overview of Pervasive System Analyzer 2-2
Summary of Functionality 2-2

Replaces Previously Released Utilities 2-2
Using PSA During Installation 2-3
Steps Performed During Installation 2-3
Using PSA Outside of the Installation Process 2-8
Why Use PSA? 2-8
Starting PSA 2-8
Common PSA Tasks 2-9

3 SQL Syntax Enhancements 3-1

Detailed Information on New and Improved SQL Syntax

Global Variables 3-2
@@IDENTITY 3-2
@@ROWCOUNT 3-4
USING, IN DICTIONARY, WITH REPLACE 3-6
Changed Grammar 3-6
USING 3-6
IN DICTIONARY 3-10
WITH REPLACE 3-11
SELECT in UPDATE 3-14
Changed Grammar 3-14
Remarks 3-14
Examples 3-15
Improved ALTER TABLE Support 3-17
Changed Syntax 3-17
Remarks 3-17
Examples 3-19
Additional Scalar Functions 3-20
String Functions 3-20
Numeric Functions 3-22
Date and Time Functions 3-25
Logical Functions 3-28
Utility Functions 3-29

4 Dynamic Cursors in Pervasive.SQL 2000i SP3 4-1

An Overview of the New Dynamic Cursor Functionality in Pervasive.SQL 2000i SP3

Features at a Glance 4-2
Overview of Dynamic Cursors and the ODBC API 4-3
Terminology 4-3
ODBC Cursor Library 4-3
ODBC APIs that are Affected 4-4
Temporary Tables 4-4
ODBC APIs Affected by New Functionality 4-5
Updated ODBC Functionality 4-5
New ODBC Functionality 4-10

Temporary Tables 4-12
New Limitations 4-12
Performance 4-13
Positioned UPDATE and DELETE 4-14
New Limitations 4-14
Performance 4-14

5 Improved OLE DB Provider in Pervasive.SQL 2000i 5-1

An Overview of the Updated OLE DB provider in Pervasive.SQL 2000i

Overview of New Features in ADO and OLE DB Provider 5-2
Command-Based Recordsets Supported 5-2
ADOX 5-2
Navigational Recordsets in the New Provider 5-2
Large Binary Objects 5-3
Programming Notes for Pervasive OLE DB Provider 5-4
Seek with Static Cursors 5-4
Remote Connections 5-4
Table Definitions 5-4
Default LockType 5-5
Initialization Properties 5-5
Performance Considerations with OLE DB 5-7
Best Performance is Navigational 5-7
Static vs. Dynamic Cursors 5-7
Disable Unused Services 5-7
COM+ Services Support 5-8
What is COM+ Services? 5-8
Example of COM+ Services for Visual Basic Programmers 5-8
Execute Method (ADO Command) 5-11
SELECT operations 5-11
Batch Insert, Update, or Delete 5-11
Limitations of the OLE DB Provider 5-13

6 JDBC 2 Enhancements in Pervasive.SQL 2000i 6-1

An Overview of the New JDBC 2 Functionality in Pervasive.SQL 2000i

Overview of Pervasive JDBC 2 Driver 6-2
Specifications 6-2
JDBC API Improvements 6-2
JDBC Optional Package Support 6-2
Backward Compatibility 6-2
Class Names 6-3
Unsupported APIs 6-3
Driver Limitations 6-3
JDBC Connection String Enhancements 6-4
How to Connect 6-4

Using Character Encoding 6-4
Notes on Character Encoding 6-5
JDBC 2.0 Standard Extension API 6-6
DataSource. 6-6
Connection and Concurrency Notes 6-10
Scrollable Result Set Notes 6-11
JDBC Programming Sample 6-12

About This Manual

This manual contains information about the features and enhancements that are new in this release of Pervasive.SQL. This release is referred to as Pervasive.SQL 2000i or Pervasive.SQL 2000 SP3. The internal version number is 7.9.

This manual describes the new and changed behaviors of the product relative to Pervasive.SQL 2000 SP2.

Who Should Read This Manual

This document is designed for any user who is familiar with Pervasive.SQL and wants to know what has changed in this release of the software.

This manual does not provide comprehensive usage instructions for the software. Its purpose is to explain what is new and different in this particular release of the product.

Pervasive Software, Inc. would appreciate your comments and suggestions about this manual. As a user of our documentation, you are in a unique position to provide ideas that can have a direct impact on future releases of this and other manuals. If you have comments or suggestions for the product documentation, post your request at http://www.pervasive.com/devtalk or send e-mail to docs@pervasive.com.

Manual Organization

This manual begins with an overview of the new features, then provides chapters containing additional details where appropriate. *What's New in Pervasive.SQL 2000i* is divided into the following sections:

- Chapter 1—"What's New in Pervasive.SQL 2000i SP3"

  This chapter provides an overview of the changes in this release of the software.

- Chapter 2—"Pervasive System Analyzer (PSA)"

  This chapter covers how to use PSA to troubleshoot problems.

- Chapter 3—"SQL Syntax Enhancements"

  This chapter covers the elements that have been added or changed in the SQL syntax grammar, excluding those for dynamic cursor support, which are described in Chapter 4.

- Chapter 4—"Dynamic Cursors in Pervasive.SQL 2000i SP3"

  This chapter covers the SQL syntax and programming considerations specific to the new dynamic cursors implementation.

- Chapter 5—"Improved OLE DB Provider in Pervasive.SQL 2000i"

  This chapter details improvements to the OLE DB provider that are available to programmers who use the Microsoft Data Access Components.

- Chapter 6—"JDBC 2 Enhancements in Pervasive.SQL 2000i"

  This chapter details the enhancements in this version of the JDBC driver.

This manual also contains an index.

Conventions

Unless otherwise noted, command syntax, code, and examples use the following conventions:

CASE	Commands and reserved words typically appear in uppercase letters. Unless the manual states otherwise, you can enter these items using uppercase, lowercase, or both. For example, you can type `MYPROG`, `myprog`, or `MYprog`.
Bold	Words appearing in bold include the following: menu names, dialog box names, commands, options, buttons, statements, etc.
`Monospaced font`	Monospaced font is reserved for words you enter, such as command syntax.
[ ]	Square brackets enclose optional information, as in [*log_name*]. If information is not enclosed in square brackets, it is required.
|	A vertical bar indicates a choice of information to enter, as in [*file_name* | @*file_name*].
< >	Angle brackets enclose multiple choices for a required item, as in `/D=<5 | 6 | 7>`.
variable	Words appearing in italics are variables that you must replace with appropriate values, as in *file_name*.
…	An ellipsis following information indicates you can repeat the information more than one time, as in [*parameter* …].
::=	The symbol ::= means one item is defined in terms of another. For example, a::=b means the item *a* is defined in terms of *b*.

chapter

1

What's New in Pervasive.SQL 2000i SP3

An Overview of New Features in Service Pack 3

The purpose of this chapter is to summarize and explain the major new features and differences in behavior between this product and the previous service pack. If further information is available for a given change or feature, a hot-link or cross-reference to that information is provided.

- "List of New Features and Improvements" on page 1-2
- "Programming Interfaces" on page 1-3
- "Performance and Reliability Enhancements" on page 1-12
- "Expanded Operating System Support" on page 1-15
- "Easier Installation and Configuration" on page 1-19
- "Improved Networking Support" on page 1-29
- "Improved SQL Support" on page 1-31
- "New and Enhanced Documentation" on page 1-34

List of New Features and Improvements

This release offers the following new features and improvements over the last release of the product:

- Programming Interfaces
  - Full Support for OLE DB IRowset and ICommand
  - Full Support for JDBC 2.0
  - Dynamic Cursors
  - Updated OEM/Partner Toolkit (PTK)
  - Enhanced Stat Extended (65) Operation
- Performance and Reliability Enhancements
  - Client/Server Version Checking
  - Row Level Locking
  - SRDE Improvements
- Expanded Operating System Support
  - Terminal Server Support
  - NSS Volume Support
  - Comma as Decimal Separator
- Easier Installation and Configuration
  - Pervasive System Analyzer
  - All Size Configuration Settings Now in Bytes
  - Dynamic Configuration Settings
  - Updated Default Settings
  - No Dependence on PERVASIVE_PATH
  - Improved Workgroup Gateway Behavior
- Improved Networking Support
  - Pervasive Auto-Reconnect
- Improved SQL Support
  - Additional SQL Syntax
- New and Enhanced Documentation

These features are described in the sections that follow.

Programming Interfaces

This section describes the interface improvements offered in this release.

Full Support for OLE DB IRowset and ICommand

This release offers full OLE DB IRowset and ICommand support, including support for COM+ (Microsoft Transaction Server). For detailed information, see "Improved OLE DB Provider in Pervasive.SQL 2000i" on page 5-1.

The updated OLE DB interfaces are part of the client and engine installation packages and are immediately available for use after installing the SP3 client and engine. No additional software is needed.

Full Support for JDBC 2.0

This release offers full JDBC 2.0 support, including forward and backward scrollable cursors using live data. For detailed information, see "JDBC 2 Enhancements in Pervasive.SQL 2000i" on page 6-1.

The updated JDBC interfaces are part of the client and engine installation packages and are immediately available for use after installing the SP3 client and engine. No additional software is needed.

Dynamic Cursors

An all-new implementation of dynamic cursors allows developers to build applications that scroll forward and backward through live data. Applications based on Pervasive.SQL 7 that use dynamic cursors can now be upgraded without losing this functionality. For detailed information on this topic, see Chapter 4, "Dynamic Cursors in Pervasive.SQL 2000i SP3."

Dynamic Cursors support is an integral part of the engine, and is automatically available after the SP3 engine is installed. No additional software is needed.

Updated OEM/ Partner Toolkit (PTK)

The Partner Toolkit for OEMs and ISVs has been updated to reflect Pervasive System Analyzer replacing InstallScout and SmartScout.

Enhanced Stat Extended (65) Operation

As a result of customer requests, a variety of new sub-function behaviors have been added to the Stat Extended (65) operation code. The position block is now returned by the engine.

The following new sub-functions are now available:

- you can determine the record address and key number of a record that generated a Status Code 5 (Duplicate Key) in the previous operation.
- you can gather a variety of information about an open data file, including the unique file ID used by the MicroKernel, the system time when the file was opened, the number of file handles currently open on that file, and a number of other statistics, detailed later in this section.
- you can identify the Gateway engine that is handling data access requests for the given file handle.
- you can identify the client ID and the type of lock that generated the most recent Status Code 84 or 85.

Parameters

	Op Code	Pos Block	Data Buf	Data Buf Len	Key Buffer	Key Number
Sent	✔	✔	✔	✔		✔
Returned		✔	✔	✔		

Prerequisites

The file must be open.

Procedure

1. Set the Operation Code to 65.
2. Pass the Position Block for the file.
3. Store the extended stat structure in the Data Buffer. See the section "Details" on page 1-5 for more information about the extended stat structure.
4. Specify the Data Buffer Length.

5 Set the Key Number to 0.

Result

If the operation is successful, the return structure required by the sub-function is returned and the data buffer length is set to the length of the data returned.

If the operation is unsuccessful, the MicroKernel may return one of the following non-zero status codes:

Table 1-1 Stat Extended (65) Status Codes

Status Code	Meaning
8	The current positioning is not valid.
22	The data buffer parameter is too short.
62	The descriptor is incorrect.
139	The key number is not a valid option.

Positioning

This operation does not require a current position to be established. It only requires a valid position block.

Details

The Extended Stat descriptor is a structure placed in the data buffer that defines how the Extended Stat operation will be executed. The first four bytes in the structure define a signature that helps to assure to the MicroKernel that this is a valid Extended Stat descriptor. Use the four characters "ExSt" as a four byte string with no termination. Another representation is 0x74537845 as a LoHi Integer.

The next four bytes is a LoHi integer indicating the sub-function to be executed by the MicroKernel. The valid sub-functions are:

Table 1-2 Stat Extended (65) Sub-functions

Sub-function ID	Description
1	Listing of extension file names
2	System Data information for the file
3	Duplicate conflict record and key identification

Table 1-2 Stat Extended (65) Sub-functions

Sub-function ID	Description
4	File information
5	Gateway identification
6	Lock owner identification

Because the first two sub-functions are already documented in *API Programmer's Reference* available with the SDK, this section discusses sub-functions 3 through 6.

Duplicate Conflict Information

The purpose of this function is to identify the record address and key number that caused a status 5 (Duplicate Key) on a previous failed insert or update operation.

Set the sub-function to 3 and the databuf length to 8.

Output Structure

```
typedef struct duplicate_error_info
{
   BYTE   DuplicateRecord[4];
   WORD   DuplicateKeyNum;
} DUPLICATE_ERROR_INFO;
```

File Information

This sub-function is designed to return a variety of status information about the file open with the specified handle. It returns the unique file ID used internally in the MicroKernel to identify the file and additional information that is usually retrieved through the Distributed Tuning Interface.

Set the sub-function to 4 and the databuf length to 20.

Output Structure

```
#define FLAG_EXT_STAT_FI_LOCKS      0x00000001
#define FLAG_EXT_STAT_FI_TRANS      0x00000002
#define FLAG_EXT_STAT_FI_READ_ONLY  0x00000004
#define FLAG_EXT_STAT_FI_CONT_OPS   0x00000008
#define FLAG_EXT_STAT_FI_RI         0x00000010
#define FLAG_EXT_STAT_FI_OWNER_RW   0x00000020
```

```
#define FLAG_EXT_STAT_FI_OWNER_ROK  0x00000040
#define FLAG_EXT_STAT_FI_WRONG_OWNER 0x00000080

typedef struct
{
   unsigned long  FileID;
   unsigned long  NumberOfHandles;
   unsigned long  OpenTimeStamp;
   unsigned long  FileUsageCount;
   unsigned long  Flags;
} FILE_DATA_INFO;
```

The structure elements and the permitted values for the `Flags` field are described in the tables below.

Table 1-3 File Information Structure

Element	Description
FileID	A unique number which the MicroKernel uses to identify the file.
NumberOfHandles	The current number of handles that the MicroKernel has open on this file.
OpenTimeStamp	The system time when the file was last opened by the MicroKernel.
FileUsageCount	This number gets incremented at each checkpoint or System Transaction. It is also the usage count placed in the FCR. The number returned here is the usage count of the file as it is represented in the Microkernel cache. When a checkpoint starts, this number gets incremented.
Flags	A four-byte bitmap in which various values may be set. See the table below for descriptions of the possible values. More flags may be added in the future.

Table 1-4 File Information Flags

Value	Name	Description
0x00000001	Explicit Locks	There are explicit locks currently on the file.
0x00000002	Client Transactions	There is at least one client transaction currently open on the file.
0x00000004	Read Only	The file was opened by the MicroKernel as ReadOnly. This may be a CD-ROM drive or a read-only directory.

Table 1-4 File Information Flags

Value	Name	Description
0x00000008	Continuous Operations	The file is currently in continuous operations.
0x00000010	Referential Integrity	The file has referential integrity constraints on it.
0x00000020	Owner Read/ Write	The file has a Read/Write Owner name assigned to it. The owner name is required to read from or write to the file.
0x00000040	Owner Reads OK	The file has an owner name that is required only to write to the file. Reads can be done without an owner name.
0x00000080	Opened with Wrong Owner	The file has a Reads-OK Owner name assigned to it and the handle was opened with the wrong owner name.

Gateway Identification

This sub-function returns an identification structure for the MicroKernel engine that is executing the requests associated with the specified handle. In a Gateway environment using the workgroup engines, it may be difficult to determine which engine in the workgroup is servicing the requests. Without this API, the only way to know for sure is to view the contents of the locator file in the directory where the file exists. With Pervasive.SQL 2000 SP2, the introduction of redirectable locator files made this even more confusing. This sub-function allows applications to display to the user the name of the Gateway engine.

Set the sub-function to 5 and the databuf length to at least 74.

Output Structure

```
typedef struct ext_stat_gateway_info
{
   WORD MajorVersion;
   WORD MinorVersion;
   WORD PatchLevel;
   WORD Platform;
   BYTE GatewayName[64];
} EXT_STAT_GATEWAY_INFO;
```

The `GatewayName` field is a null-terminated string. The data buffer length returned reflects the actual length of the structure including the null terminator in `GatewayName`.

Lock Owner Identification

The purpose of this sub-function is to identify what client caused the most recent Status Code 84 or 85. As soon as you get a status 84 or 85, call this sub-function and you can gather information about the client that is holding that lock. This information reflects the most recent blocking client and is stored with your client information in the MicroKernel. It is overwritten each time a Lock error occurs, so this sub-function must be called immediately upon receiving a status 84 or 85.

Set the sub-function to 6 and the databuf length to at least 104.

Output Structure

```
#define FLAG_EXT_STAT_LI_IMPLICIT     0x00000001
#define FLAG_EXT_STAT_LI_EXPLICIT     0x00000002
#define FLAG_EXT_STAT_LI_FILE         0x00000010
#define FLAG_EXT_STAT_LI_PAGE         0x00000020
#define FLAG_EXT_STAT_LI_RECORD       0x00000040
#define FLAG_EXT_STAT_LI_DATA_PAGE    0x00000100
#define FLAG_EXT_STAT_LI_KEY_PAGE     0x00000200
#define FLAG_EXT_STAT_LI_VAR_PAGE     0x00000400
#define FLAG_EXT_STAT_LI_SAME_PROCESS 0x00000800
#define FLAG_EXT_STAT_LI_WRITE_NO_WAIT 0x00001000
#define FLAG_EXT_STAT_LI_WRITE_HOLD   0x00002000
#define FLAG_EXT_STAT_LI_READ_NO_WAIT 0x00004000
#define FLAG_EXT_STAT_LI_READ_MULTIPLE 0x00008000

typedef struct ext_stat_lock_owner
{
   BYTE         ClientId[16];
   ULONG        Flags;
ULONG        TimeInTrans;
   ULONG        KeyNum;
   ULONG        TransLevel;
   CHAR         Reserved[8];
   CHAR         DisplayName[64];
} EXT_STAT_LOCK_OWNER;
```

The data buffer length returned reflects the actual length of the structure including the null terminator in `DisplayName`.

If there is no record in the MicroKernel of a previous blocking client, then the output data buffer length is set to zero.

The structure elements and the permitted values for the `Flags` field are described in the tables below.

Table 1-5 Lock Owner Structure

Element	Description
ClientID	The 16-byte client ID of the blocking client.
Flags	A four-byte bitmap containing flags indicating the type of conflict that occurred. See the table below for a description of each flag value.
TimeInTrans	Number of milliseconds in which the blocking client has been in a transaction. This can be helpful in determining whether to retry the operation.
KeyNum	If the conflict occurred on a key page, this element indicates which key is involved. Tracking this information can be useful in designing a database with fewer potential conflicts.
TransLevel	If this number is non-zero, then the blocking client is currently in a transaction. Since some page and record locks are held until the transaction completes, this information might be useful in determining if the operation should be retried.
Reserved	Reserved for future use. If there is some information about the blocking client which you think may be useful, please contact Pervasive Software.
DisplayName	This is a null-terminated string which is the same identifying name that is displayed in Monitor for each client. Use at least 64 bytes since that is the current maximum display name length.

Table 1-6 Lock Owner Flags

Value	Name	Description
0x00000001	Implicit Lock	The blocking client is using an implicit lock.
0x00000002	Explicit Lock	The blocking client is using an explicit lock.
0x00000010	File Lock	The blocking client is using a file lock.
0x00000020	Page Lock	The blocking client is using a page lock.
0x00000040	Record Lock	The blocking client is using a record lock.
0x00000100	Data Page	If the conflict was a Page Lock, this flag indicates the conflict occurred on a data page.

Table 1-6 Lock Owner Flags

Value	Name	Description
0x00000200	Key Page	If the conflict was a Page Lock, this flag indicates the conflict occurred on a key page.
0x00000400	Variable Page	If the conflict was a Page Lock, this flag indicates the conflict occurred on a variable page.
0x00000800	Same Process	If this flag is set, then the first 12 bytes of the blocking client ID are the same as the first 12 bytes of the client that got blocked, that is, the client that is issuing the Stat Extended call. In this case, it means that the two blocking clients came from the same process on the same computer. If you have a single threaded application making Btrieve calls, then retrying this operation will not help. You need to complete or abort the work that is blocking.
0x00001000	Write No Wait	Indicates that the blocking client is using the 500 bias.
0x00002000	Write Hold	Indicates that the blocking client made a change to a page that caused that client to keep the full page lock until its transaction completes. This situation can occur on implicit key page locks when a change causes key entries to move to another page.
0x00004000	Read No Wait	For explicit record locks, this flag indicates that the blocking client is using either lock bias 200 or 400.
0x00008000	Read Multiple	For explicit record locks, this flag indicates that the blocking client is using either lock bias 300 or 400.

Performance and Reliability Enhancements

This section describes the performance and reliability enhancements offered in this release.

Client/Server Version Checking

This release introduces a new feature designed to guarantee engine-to-client version compatibility. When a client requester first connects to an engine, the client requester compares its internal router version with the value returned from the engine by a Btrieve Version (26) call. If the client version is older than the engine, a message dialog box is displayed on the client system with the message "Engine components' Version is different from Clients'" along with a suggestion to run Pervasive System Analyzer (PSA). The same message is also logged in the client's PVSW.LOG file. This message is only a warning. Although the client is not prevented from connecting to the engine in this situation, keep in mind that older clients are not tested against newer engines. Pervasive only guarantees compatibility between engines and clients if the clients are the same version as, or newer than, the engines. If you choose not to run PSA when prompted by this message, you can expect the product to behave unpredictably until the client version is equal to or greater than the engine version.

After running PSA and archiving the old client components, you should upgrade to the latest client.

Row Level Locking

Row level locking improves database engine performance in multi-user environments in which many updates occur at the same time, or in which transactions remain open for an extended period of time.

Prior to this release, the database engine locked an entire page containing more than one record in order to update any record on that page. Any other client attempting to update records on that page while the page was still locked had to wait until the first operation released the page lock. This behavior occurred even though the records needed by the second operation were not affected by the first operation.

With the new row level locking architecture, a transaction locks only the records that it affects directly, not the entire page. One client can update records on a given page at the same time as another client updates different records on the same page. Waiting is necessary only

when a second operation attempts to modify the exact same records currently locked by the first operation. Thus, row level locking decreases overall wait time and improves performance in a multi-user environment.

This feature is completely transparent within the MicroKernel Database Engine. There are no changes to the Btrieve API, data file format, configuration settings, or any external component. This feature is always on and is supported across Server, Workgroup, and Workstation on all supported operating system platforms. This feature is supported for data file format v6.x and later. It is not supported for data file format v5.x or earlier.

In this release, row level locking is implemented for key pages and data pages only, not variable pages. Furthermore, a small percentage of key page changes may cause key entries to move from one page to another. An example is when a key page is split or combined. These changes retain a full page lock until the transaction is completed.

SRDE Improvements

Several improvements have been made to the SQL Relational Database Engine (SRDE).

Column Number Limits Increased

The maximum number of columns permitted in a table and the maximum number of columns that can be listed in a SELECT statement have both increased.

Table 1-7 Column Number Limits

Attribute	Value
Maximum number of columns allowed in a table	1536
Maximum number of columns allowed in a SELECT statement	1600

Optimized Queries Involving Predicates with Scalar Functions

Queries containing a WHERE clause with a scalar function are now optimized. The only scalar functions that can be optimized are RTRIM and LEFT. They cannot be optimized if they are contained in a complex expression on either side of the predicate.

For example, consider the following query:

```
SELECT * FROM T1, T2 WHERE T1.C1 = LEFT(T2.C1, 2)
```

In this case, both sides of the predicate are optimized. The *predicate* is the complete search condition following the WHERE keyword. Depending on the size of the tables involved in the join, the optimizer chooses the appropriate table to process first.

If you have similar queries in your applications, you should expect to see a significant performance gain on these queries.

Expanded Operating System Support

This section describes the expanded operating system support offered in this release.

Terminal Server Support

Pervasive.SQL has in the past supported installation of client software only on systems running a terminal server. The server engine components had to be installed on a separate system. With Service Pack 3, access from terminal sessions to server engine components located on the same system is supported.

The following environments are supported:

- Windows NT 4.0 Terminal Server Edition (Service Pack 6)
- Windows 2000
- Citrix MetaFrame

Server, Workgroup, and Workstation engines are supported on the above terminal server configurations. In each case, the engine must be started by the administrator prior to any terminal users logging onto the terminal server. Terminal users may not start up their own database engine.

Installing on Microsoft Terminal Server

Prior to installation on a computer with Microsoft terminal server installed, you must switch the terminal server to install mode using the `change user /install` command. This ensures that the software is available to all users after installation. When the installation is complete, you should set the terminal server back to runtime mode using the command `change user /execute`.

For more information about this command, see the Microsoft knowledge base article (especially the "Additional Notes" at the bottom of the article):

http://support.microsoft.com/support/kb/articles/Q186/5/04.ASP

NSS Volume Support

This is not a new feature in the product, but rather a clarification. Pervasive.SQL supports NetWare Storage Services (NSS) volumes on NetWare 5 and up, provided that you load the NSS volumes prior to starting the database engine. For example, you should issue the

BSTART or MGRSTART command only after loading the NSS volumes as shown here:

```
LOAD NSS
MOUNT ALL
SYS:ETC\INITSYS.NCF
MGRSTART or BSTART
```

Also, please note that database updates performed against data files on NSS volumes may run more slowly than with earlier versions of NetWare. As noted in Novell TID 2952147 (http://www.novell.com), "NSS is optimized for reading files." Updates "will almost always perform a little faster on the legacy file system."

Based on this information, you may wish to store frequently-updated data files on regular NetWare volumes rather than NSS volumes.

Comma as Decimal Separator

Many locales, especially in Europe, use a comma to separate whole numbers from fractional numbers within a floating point numeric field. For example, these locales would use 1,5 instead of 1.5 to represent the number one-and-one-half.

Starting with Service Pack 3, Pervasive.SQL 2000i can support both the period "." and the comma "," as decimal separators. The database engine uses the decimal separator that is defined by the regional settings in the operating system.

Note When the decimal separator is not a period, numbers appearing in SQL statements must be enclosed in quotes.

Client/Server Considerations

Support for the comma as decimal separator is based on the locale setting in the operating system. Both the client operating system and the server operating system have a locale setting. The expected behavior varies according to both settings.

- If the server and/or client locale setting uses the comma as decimal separator, then the SRDE accepts both period-separated values and quoted comma-separated values.
- If neither the server nor the client locale setting uses the comma decimal separator, then the SRDE does not accept comma-separated values.

Changing the Locale Setting

Decimal separator information can only be retrieved or changed for a Win32 machine (Windows95/98/NT/2000).

The decimal setting for NetWare and Unix is not configurable, and it is set to a period. If you have a NetWare or Unix server engine and you want to use the comma as decimal separator, you must ensure that all your client computers are set to a locale that uses the decimal separator.

➤ To view or change your locale setting on Windows

1. From the Start menu, open the **Control Panel**.
2. In the Control Panel window, double-click **Regional Settings**.
3. On the Regional Settings tab, select the desired country.
4. You must stop and restart the Pervasive.SQL engines.

Examples

Example A - Server locale uses the comma for decimal separator

Client's locale uses comma "," as decimal separator:

```
create table t1 (c1 decimal(10,3), c2 double)
insert into t1 values (10.123, 1.232)
insert into t1 values ('10,123', '1.232')
select * from t1 where c1 = 10.123
select * from t1 where c1 = '10,123'
```

The above two select statements, if executed from the client, return:

```
10,123, 1,232
10,123, 1,232
```

Client's locale uses period "." as decimal separator:

```
create table t1 (c1 decimal(10,3), c2 double)
insert into t1 values (10.123, 1.232)
insert into t1 values ('10,123', '1.232')
select * from t1 where c1 = 10.123
select * from t1 where c1 = '10,123'
```

The above two select statements, if executed from the client, return:

```
10.123, 1.232
10.123, 1.232
```

Example B - Server locale uses the period for decimal separator

Client's locale uses comma "," as decimal separator:

Same as client using comma "," in Example A.

Client's locale uses period "." as decimal separator:

```
create table t1 (c1 decimal(10,3), c2 double)
insert into t1 values (10.123, 1.232)
insert into t1 values ('10,123', '1,232')
    -- error in assignment
select * from t1 where c1 = 10.123
select * from t1 where c1 = '10,123'
    -- error in assignment
```

The first select statement above, if executed from the client, returns:

```
10.123, 1.232
```

Easier Installation and Configuration

This section describes new features that make installing and configuring Pervasive.SQL easier.

Pervasive System Analyzer

Pervasive System Analyzer (PSA) is a new graphical program that detects network problems, previous versions of Btrieve or Pervasive.SQL, and other notable environmental factors. It also allows you to archive or restore previous installations of Pervasive.SQL. If you choose, PSA can auto-fix a variety of problems for you.

PSA replaces the features that were previously offered by SmartScout and InstallScout. These two programs are no longer available in Pervasive.SQL 2000i.

By default, PSA runs automatically during installation before and after the database engine is installed. You can also run it manually to search for problems later. To run it manually, choose **Start | Programs | Pervasive | Pervasive.SQL 2000i | Utilities | Pervasive System Analyzer.**

For further details, see Chapter 2, "Pervasive System Analyzer (PSA)."

All Size Configuration Settings Now in Bytes

Previous releases of Configuration measured some size settings in bytes, and others in kilobytes. Starting with this release, all size settings are standardized in bytes to prevent confusion over different units.

Please note, if your company has documentation about how to change configuration settings, be sure to verify that this material is still accurate. For example, with the previous release, the default value for a buffer size setting may have been 32(KB). Your company's instructions may specify to change this value to 64. Now that the values are measure in bytes, not kilobytes, you may think your instructions mean 64 *bytes,* when your instructions actually intend 64 *kilobytes* (a value of 65536, not 64). Please review any documentation you have in order to address this issue.

Dynamic Configuration Settings

With this release, several configuration settings related to memory management have become obsolete. These resources are now managed dynamically by the database engine. The engine allocates

additional memory as needed during runtime operation to allow for increased resource usage.

If memory usage decreases, each chunk of memory is returned to the operating system when none of the objects within that chunk remain in use. If the engine is configured with the Server | **Memory Usage | Back to Minimal State if Inactive** setting enabled, the engine returns nearly all memory to the operating system when the engine goes to minimal state.

Obsolete Settings

The following settings are now dynamically configured on all supported platforms. These settings have been removed from Configuration in Pervasive Control Center, but some of them still appear in Monitor. These settings can be read, but not set, by DTI and DTO. All of these settings were previously located in the **Server | Access** category of Configuration:

Active Clients
Logical File Handles
Maximum Databases
Maximum Open Files

Also, please recall that two additional configuration settings in the **Server** category were made obsolete (dynamically configured by the engine) starting in Service Pack 2:

Largest Compressed Record Buffer Size
Extended Operation Buffer Size

Updated Default Settings

An assortment of default configuration settings have been changed to provide better performance and improved support for modern computing environments. These settings can be accessed through Configuration within PCC.

Server | Performance Tuning | Log Buffer Size

Previous default value: 65,536 bytes
New default value: 262,144 bytes

If **Server | Data Integrity | Transaction Durability** is set to **On**, then all changes, whether transactional or not, are logged. Many Btrieve customers do not use transactions and thus have no interest in transaction durability. In fact, it can slow these customers down by

as much as 50%. When no transactions are occurring, the log buffer is flushed whenever it gets full. Batch updates and inserts can be accomplished faster if **Transaction Durability** is set to **Off** or if the transaction **Log Buffer Size** is larger.

This performance increase occurs because each flush of the log has a certain amount of fixed overhead. Flushing the log less frequently may take slightly longer for each flush because the log data is larger, but the overall effect can be a significant increase in performance. Increasing the default buffer size to 256KB instead of 64KB helps significantly without incurring too much memory usage. Further performance advantages diminish above that level.

Server | Performance Tuning | Communications Threads

Previous default value: 3
New default value: 16

Pervasive AutoReconnect requires more resources from the communications threads than previous releases of the product. As a result, Pervasive has increased the default number of communications threads to 16.

Server | Communications Buffer Size | MKDE Communications Buffer Size

Previous default value: 16,384 bytes
New default value: 65,536 bytes; 64,512 bytes on Win32

Previously, if your application used records larger than the default value, you had to set this parameter. Pervasive has changed the default value to the maximum value so that applications always work (you will not receive "Buffer size too small" error), but more memory is used by default. The MicroKernel allocates one buffer of this size per worker thread. If you know the exact record size, you can save a small amount of memory by tuning this value down to the approximate size of your records.

Server | Communications Buffer Size | Communications Buffer Size

Previous default value: 16,384 bytes
New default value: 65,536 bytes; 64,512 bytes on Win32

This value should be kept the same as that for **MKDE Communications Buffer Size** parameter.

Server | Data Integrity | Operation Bundle Limit

Previous default value: 1000
New default value: 65,535

The operation bundle limit specifies how many operations can occur against a single file before a system transaction occurs. During batch inserts, it is very possible to insert 5,000 - 10,000 small records within a few seconds. There is no compelling reason to initiate a system transaction 5 to 10 times within such a small time interval. Increasing the number of operations between system transactions allows fewer system transactions to occur and thus can improve overall performance.

Server | Data Integrity | Wait Lock Timeout

Previous default value: 30
New default value: 15

Most timeout situations that will return a Status Code after 15 seconds are unlikely to be resolved by waiting an additional 15 seconds. By reducing the wait timeout, control is returned to the application 15 seconds sooner with most likely the same result as if the user had waited the full 30 seconds.

Server | Access | Number of Sessions

Previous default value: 15
New default value: 500 (dynamic value on Win32, see below)

The Communications module uses only a small amount of memory per session, so there is little harm in allowing many more sessions. A higher default value can avoid errors that occur when the server runs out of available sessions.

On Win32 platforms, this default value is adjusted based on the maximum number of users reported by the user count manager. The engine starts with a default value equal to five times the installed user count, then adjusts that number so that it is between 100 and 1000, inclusive. The current value as read from the registry is adjusted to 100 if it is less than the calculated number. If the calculated value is different than the value in the registry, the calculated value is written

to the registry. Note that Win32 systems that currently have the old default value of 15 will get replaced by 100. On UNIX and NetWare platforms, the default is now 500, but no adjustments are made based on the user count since these are server-only platforms.

No Dependence on PERVASIVE_PATH

Pervasive.SQL no longer depends on the environment variable PERVASIVE_PATH at installation or runtime on Windows.

In order to provide installation and run-time reliability, the environment variable PERVASIVE_PATH is no longer used during component loading of the Pervasive.SQL engines. When the engine is first started, the operating system searches the default path for w3btrv7.dll and w3scmv7.dll. On Windows 9X, these DLLs are installed into the system directory.

Once these DLLs are loaded, when looking for additional downstream components, Smart Components first explores the directory where Pervasive.SQL is installed. If the requested component is not found, Smart Components then searches on the platform's default path.

Improved Workgroup Gateway Behavior

This release introduces a new feature of Gateway engine operation that guarantees transaction atomicity for multi-directory databases and also makes it much easier to change the name of a Gateway engine across multiple data directories.

Limitations of Previous Model

The previous version of the product supported two modes of Workgroup Gateway operation:

- a *permanent* Gateway database engine always serves the data files in a given directory—if the specified database engine is not up, then the data in that directory is not available.
- with a *dynamic* Gateway, whichever database engine opens the data files first serves as the Gateway; this architecture is also referred to as a *floating* Gateway.

This model was useful but had two limitations. First, if you had data in many different directories, and you wanted to change the permanent Gateway to a different engine, you had to use the Gateway Locator Utility to update the Locator File in every data directory (or update all the Locator Files by hand).

Second, if you had a single database consisting of data files in more than one directory, it became possible for two different engines to handle data access within the same database, thus failing to ensure transaction atomicity. This situation could occur if the Gateway Locator files in the two directories pointed to different Gateway engines. For some users this may not be an issue, but transaction durability can only be guaranteed if a single database engine performs all data access operations on a given database. If transaction durability is important, then you must ensure that the same engine services all data files within the same database, regardless of their directory locations.

New Behavior

First, recall that the Pervasive.SQL client uses the following approach to access remote data files:

- First, attempt to connect to a database engine on the same computer as the data files.
- Second, if no database engine is available on the remote machine, attempt to use a local engine to take ownership of the remote directory and create a Locator File. If a Gateway Locator File already exists, the local engine is not used.
- Third, try to use the specified Gateway engine.

It is important to remember that the Gateway configuration only goes into effect when there is no database engine available on the same computer as the data files.

In this release, you can now allow a dynamic (floating) Gateway engine while at the same time preserving transaction durability for multi-directory databases on the same volume. This benefit is provided by a new type of Gateway Locator File that points to another Gateway Locator File. The new type is called a *Redirecting Locator File*. By having Redirecting Locator Files in directories A, B, and C that point to the Locator File in directory D, you can ensure that the Gateway engine specified by the Locator File in directory D services data files in the other directories as well.

Regardless of whether the Locator file in directory D specifies a permanent Gateway or is dynamically created by the first engine to open those files, this architecture ensures that all the specified directories use the same Gateway engine. Likewise, if you decide to change the permanently assigned Gateway engine for several

directories, Redirecting Locator Files allow you to do so by changing only one Locator File, rather than all of them. Thus, it is possible to specify that all data files on a given hard drive must use the same Gateway engine, with or without designating a permanent Gateway.

Redirecting Locator File Requirements

The first line of a Redirecting Locator File must start with "=>" and be followed by a path specifying another Locator File, which must be on the same drive. You can use any combination of forward slash and back slash in the path name. All slashes are converted to the type of separator used by the local operating system.

If your specified path ends with a slash, the database engine assumes the default Locator File name (~PVSW~.LOC) and appends it to the path. If the specified path does not end with a slash, the database engine assumes that the path already contains the file name.

The table below lists the ways a Redirecting Locator File path can be specified:

Table 1-8 Redirecting Locator File Path Descriptions

Path	Meaning
=>*path_name*	Specifies the path from the root of the drive where the current Locator File is stored.
=>.*path_name*	Specifies the path relative to the current directory.
=>..*path_name*	Specifies the path relative to the parent directory of the current directory.

You can assign multiple levels of redirection to these Locator Files. For example, you can have the first Locator File pointing to a second Locator File, the second Locator File pointing to a third Locator File, and so on. Each engine opens each Locator File sequentially, looking for the actual Gateway name. It stops searching once it has found the locator file that does not start with "=>". The engine then assumes this Locator File specifies the Gateway engine.

Creating Redirecting Locator Files

As with any Locator File, a Redirecting Locator File is a plain text file. You can create Redirecting Locator Files by hand or

programmatically. A Redirecting Locator File must be flagged as read-only, or it will be overwritten by the first engine to attempt to access the data files in that directory.

➤ To Create a Redirecting Locator File

1. Open Notepad or a text editor, and open a new text file.
2. Decide where you are going to save the file when you are finished. You will save the file in the same directory as the data files which you want to redirect to another locator file.

   For example, if you want to ensure that the data files in C:\data are accessed by the same Gateway engine as other data files, then you will want to keep in mind the folder C:\data.
3. Type in `=>` and the path name of the next Locator File. Continuing the example from the previous step, if you want the current data files in C:\data to be owned by the Gateway engine specified in the Locator File located in c:\moredata, then you would type the following:

   `=>..\moredata\` *(recommended) or*

   `=>\moredata\` *(not recommended)*

   In the first case, you are specifying a relative path from the current directory. In the second case, you are specifying an absolute path from the root of the current drive. In this particular example, both cases resolve to the same target directory.

Note Pervasive strongly recommends that you use relative path names (starting with ./ or ../) in your Redirecting Locator Files, and that you use the same share names on all workstations to access the same data. Following these two recommendations can prevent errors that may occur with network path name resolution over mapped drives.

4. Save the file as ~PVSW~.LOC in the directory where the data files exist that you want to specify a Gateway engine for.
5. Close Notepad or the text editor.
6. Flag the text file as read-only.

To mark the file as read-only on Windows, you can use the Properties dialog box (right-click on the file icon) in Windows Explorer, or you can use the ATTRIB command in a DOS session or in a program:

```
ATTRIB +R ~PVSW~.LOC
```

➤ To synchronize many data directories on a permanent Gateway

1 Either by hand or by using the Gateway Locator program, create a read-only (permanent) Locator File that does not redirect. It must specify a Workgroup engine to use as the Gateway.

For example, your locator file may specify the computer named "workgroup1" as the Gateway engine, and the file may be located in C:\DATA\DB1.

2 For each of the other data directories that you want to use the Gateway engine specified in the previous step, you need to create a Redirecting Locator File in that directory. Each Redirecting Locator File must point to the file you created in the previous step.

Continuing the example, each Redirecting Locator File in C:\DATA\DB2 and C:\DATA\DB3 would then contain the following text:

```
=>..\DB1\
```

This causes any engine reading this file to follow the relative path and search the specified directory C:\DATA\DB1 for another Locator File. In this case, the specified directory contains a Locator File that names "workgroup1" as the Gateway computer.

➤ To synchronize many data directories on a dynamic Gateway

1 Follow the steps above, only in step #1, ensure that the Locator File is writable, not permanently-assigned.

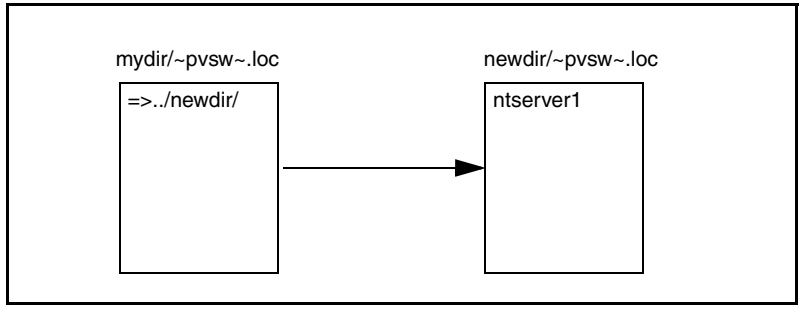
In this case, remember that if no engines are accessing any data files in the redirecting hierarchy, then there will be no Locator File in the target directory. This is normal. The dynamic Locator File is created each session by the first engine to access the data,

and the file is deleted when the last user session ends. It is permissible to have Redirecting Locator Files that point to a data directory that has no Locator File in it. In this case, the first engine to open those data files creates the Locator File.

Example

Using the example Locator Files shown in Figure 1-1, the Redirecting Locator File on the left forces the database engine to go up one directory, then look in the sub-directory `newdir` for another Locator File with the default name (~PVSW~.LOC). This Locator File, in turn, specifies that the Workgroup engine on the computer named `ntserver1` is the correct Gateway engine. As a result, the database engine on `ntserver1` is used to access the data files in the directory `mydir`.

Figure 1-1 Redirecting Locator File Example



Improved Networking Support

This section describes features in this release that improve reliability across product versions and over imperfect networks, and provide easier troubleshooting of network-related problems.

Pervasive Auto-Reconnect

Pervasive Auto-Reconnect (PARC) allows client-server or workgroup applications to endure temporary network interruptions without canceling the current database operation. When Pervasive.SQL detects a network interruption, it automatically attempts to reconnect at specific intervals for a configurable amount of time. This feature also preserves the client context so that when communications are re-established, database access continues exactly where it left off when the network interruption occurred.

This feature preserves the application context and attempts to reconnect regardless of whether the client or server was attempting to send data at the moment when the network communications were interrupted.

When a network interruption occurs, the reconnect attempts occur at specific intervals. For all connections, successive attempts are made at 0.5, 1, 2, 4, and 8 seconds, continuing every 8 seconds thereafter until the AutoReconnect Timeout value is reached. If no attempt is successful before the maximum wait time is reached, then the current operation fails and the client connection is reset. The maximum wait time is configurable between 45 seconds and 65,535 seconds.

Interface Support

Options for this feature are supported by Distributed Tuning Interface (DTI), Distributed Tuning Objects (DTO), and Pervasive Control Center (PCC) Configuration program.

Configuration

Server | Communications Protocols | Enable AutoReconnect
Type: Bool
Range: On/Off
Default: Off
This setting specifies whether you want the server to support clients

attempting to auto-reconnect during a network outage. A setting of "On" means AutoReconnect is enabled.

Server | Communications Protocols | AutoReconnect Timeout
Type: Numeric
Range: 45 - 65535 seconds
Default: 180 seconds
This setting specifies how long the client will attempt to connect to the server before giving up. When a AutoReconnect-enabled client first connects to a AutoReconnect-enabled server, the server communicates this value to the client so that both components know how long to attempt to reconnect in the event of a network interruption.

Client | Communications Protocols | Enable AutoReconnect
Type: Bool
Range: On/Off
Default: Off
This setting specifies whether you want the client to attempt to reconnect during a network outage, if it is initially connected to a server with AutoReconnect enabled. A setting of "On" means the AutoReconnect feature is enabled.

Other Considerations

This feature is supported for Btrieve, ODBC, and DTI connections. All 16-bit (when thunking to Win32) and 32-bit Windows applications are supported, but not pure DOS or Windows 3.x platforms. DOS using Btrieve DOS box support on 32-bit Windows platforms is supported. This feature is supported on all supported server platforms.

The Btrieve communication servers may write out *.PAR or *.SAR files to the Transaction Log Directory. These are temporary files that contain the context for the last item that the server tried to send to the client. When a reconnection occurs, the client may ask for data to be re-sent. The server reads these files to obtain the appropriate data. These files are normally deleted by the server after the data is read or later when the connection is finally terminated.

Improved SQL Support

This section describes new features that make it easier to work with SQL and give you more flexibility to develop powerful relational applications.

Additional SQL Syntax

An assortment of additional SQL functions and keywords have been added. These new features improve backward compatibility, improve compatibility with Microsoft SQL Server, and offer greater power and flexibility in your SQL statements.

These new features are discussed in detail in Chapter 3, "SQL Syntax Enhancements." Summaries are provided below:

@@IDENTITY, @@ROWCOUNT

These two new global variables allow you to access the most recent IDENTITY column value inserted by the SRDE for the current database connection, or to access the number of rows affected by the most recently executed SQL statement for the current database connection. For further details, see "Global Variables" on page 3-2.

USING, IN DICTIONARY, WITH REPLACE

These keywords are additions to CREATE TABLE, ALTER TABLE, and DROP TABLE allowing you to specify a file name when creating a table. This set of keywords also allows you to make certain changes to the DDFs without affecting the data files. This functionality was available in Pervasive.SQL 7, but until now was not available in Pervasive.SQL 2000. For further details, see "USING, IN DICTIONARY, WITH REPLACE" on page 3-6.

SELECT in UPDATE

In the SET clause of an UPDATE statement, you may specify a sub-query. This feature allows you to update information in a table based on information in another table or another part of the same table. This functionality was available in Pervasive.SQL 7, but until now was not available in Pervasive.SQL 2000. For further details, see "SELECT in UPDATE" on page 3-14.

Table Creation with Legacy Null Support

This version of the product allows you to set the default format for creation of tables with regard to NULL support. Normally, Pervasive.SQL creates new tables using the true NULL data record format first offered in Pervasive.SQL 2000, which adds a NULL indicator byte to the beginning of every field. By turning off this engine setting using a SQL statement, you can create new tables that use the legacy NULL data record format that was used in Pervasive.SQL 7.

The creation mode remains in effect until it is changed by issuing the statement again, or until the connection is disconnected. Because this setting is maintained on a per-connection basis, separate database connections can maintain different creation modes, even within the same application. Every connection starts with the setting in default mode, where new tables are created with true NULL support.

This feature does not affect existing tables or available column data types. All tables are created using Pervasive.SQL 2000 data types. For example, old data types such as NOTE or LVAR will not be available for use regardless of which type of NULL support is selected.

To toggle the setting and specify that new tables should be created with legacy NULL support, use this SQL statement:

```
SET TRUENULLCREATE=OFF
```

To toggle the setting and return the engine to the default, which is table creation with true NULL support, use this SQL statement:

```
SET TRUENULLCREATE=ON
```

This setting can only be toggled using SQL, it cannot be set using Pervasive Control Center.

ALTER TABLE—Change Data Type or Nullability

In Pervasive.SQL 7, you could change the nullability or the data type of an existing column in a table. In releases of Pervasive.SQL 2000 prior to SP3, this capability was not available. In this release of the product, this capability has been restored and enhanced. For further details, see "Improved ALTER TABLE Support" on page 3-17.

Additional Scalar Functions

This release supports a variety of new string, numeric, date, time, and logical functions. For further details, see "Additional Scalar Functions" on page 3-20.

Improved OEM to ANSI Support

Applications can now store or retrieve character data in the OEM character set using Pervasive.SQL, while allowing the data to be manipulated and displayed using the ANSI Windows character set. The Pervasive ODBC driver translation DLL can perform all necessary translations between the two character sets. This feature can be turned on or off for each DSN. To access the switch, click **Options...** on the Pervasive ODBC DSN Setup dialog box.

The Pervasive Control Center (PCC) and the SQL Data Manager (SQLDM) are not fully OEM-character aware if you use extended ASCII characters for column or table names. However, any character data that is passed to and from the database is correctly translated between the OEM and ANSI character sets. OEM-character-aware column and table names within PCC and SQLDM will be fully enabled in a future service pack.

If your application connects to the data source using SQLDriverConnect, you can also specify the translation DLL using the connection string option TRANSLATIONDLL=*path_and_DLL_name*. The translation DLL name for Pervasive is W32BTXLT.DLL, and it is located in the bin sub-directory of the installation directory (C:\pvsw\bin by default).

Note The OEM to ANSI translation option is available only for Client DSNs or local Engine DSNs.

New and Enhanced Documentation

This release introduces a host of new and improved material and provides an enhanced format on platforms where support for the format is available.

Support for Microsoft HTML Help

On Windows platforms, this release supports both RTF-based Windows 95 Help and HTML-based Windows 98/2000 Help, also known as Microsoft HTML Help. If you choose the Typical installation or if you use the Custom installation to install the online documentation, Pervasive.SQL automatically detects which version of Help is supported on your computer, and installs the appropriate set of online documentation.

Getting Started with Pervasive.SQL

Both the Server and Workstation/Workgroup editions of this title have been re-structured, re-written, and expanded to provide more of the information you need when you're getting started, and to make it easier for you to find relevant information quickly. This release includes a new chapter specifically dedicated to help you successfully upgrade Btrieve 6.x installations to Pervasive.SQL 2000i.

Advanced Operations Guide

This release offers a brand new title, *Advanced Operations Guide*, aimed squarely at network administrators, value-added resellers, and other technical staff who need to know a lot more than just the basics. This book goes into specific detail on backup/restore, database security, client and engine configuration, periodic maintenance, and a variety of other useful topics.

Status Codes and Messages

This resource includes updated and expanded troubleshooting and problem resolution information.

chapter

2

Pervasive System Analyzer (PSA)

An Overview of the New Diagnostic Utility in Pervasive.SQL 2000i SP3

The Pervasive System Analyzer is a new utility that replaces and extends SmartScout and InstallScout utilities.

The following are the sections found in this chapter:

- "Overview of Pervasive System Analyzer" on page 2-2
- "Using PSA During Installation" on page 2-3
- "Using PSA Outside of the Installation Process" on page 2-8

Overview of Pervasive System Analyzer

The Pervasive System Analyzer (PSA) is a utility that searches your system for previous Pervasive components and provides feedback regarding the appropriate course of action to remedy potential version conflicts.

PSA also provides you with the ability to archive previous installations.

Summary of Functionality

- Checks a machine at the start of an installation to ensure that preexisting components will not interfere
- If existing components are found, you have the option to move them to an archive directory from which they can be later restored if necessary
- If you are installing a client, PSA tests the network connectivity to the machine running the Pervasive.SQL server.
- After your new Pervasive.SQL files are installed, PSA tests the new client's ability to perform basic database operations.

Replaces Previously Released Utilities

If you have been a Pervasive customer before this release, you will notice that certain utilities are no longer included with Pervasive.SQL 2000i. PSA integrates the most useful functionality of these utilities in one convenient user interface and further extends their functionality. For reference, these utilities are:

- InstallScout
- SmartScout
- Supportability

Using PSA During Installation

PSA performs two primary functions during your Pervasive.SQL installation process:

- Before copying new files, PSA scans your system for previous versions of Pervasive products and allows you to archive them so that they do not interfere with your new installation

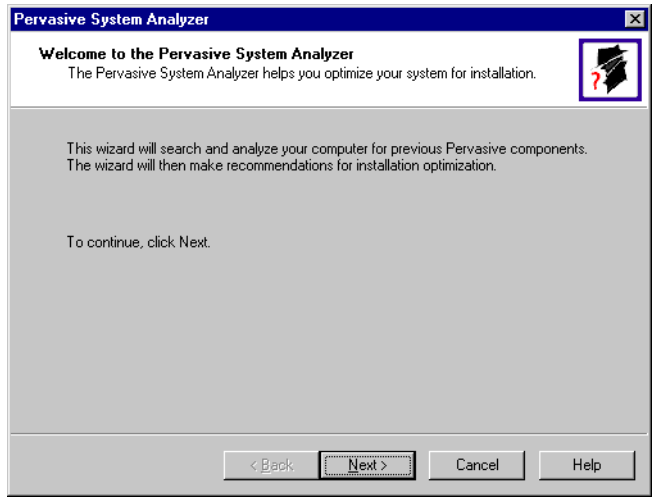
  If you are installing a client, PSA also tests your Network connectivity to the machine on which a Pervasive.SQL server engine is installed.

- After installing new files, PSA allows you to test your new installation's ability to perform basic database functions.

Steps Performed During Installation

1 Early in the installation process, PSA displays its welcome screen.

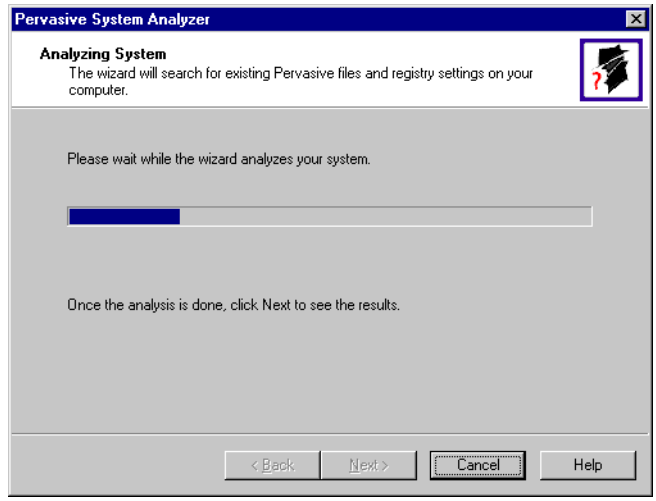
Figure 2-1 PSA Welcome Screen



Click **Next** to begin the analysis.

2 The analysis may take several minutes, depending on your system. When PSA runs during installation, only the Pervasive installation path and the system path are searched for existing components.

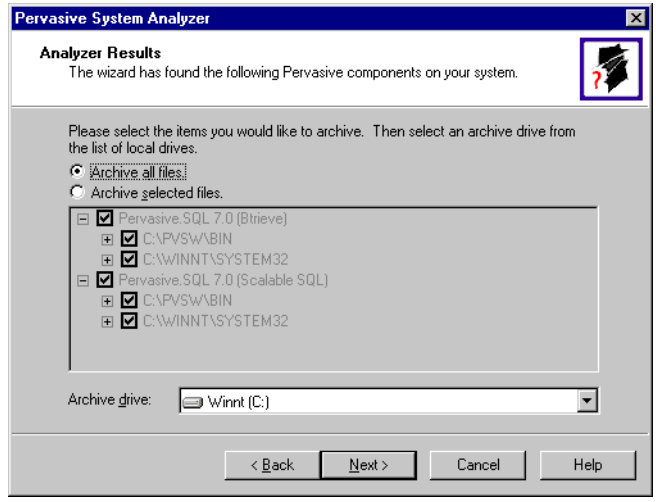
Figure 2-2 Analysis in Progress



When the analysis is complete, click **Next** to display the results.

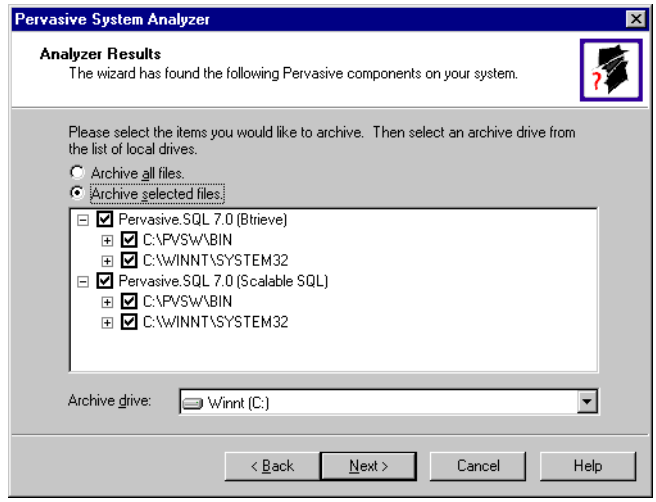
3 PSA displays the files it found from previous installations. *The default is to archive all files* as shown in the following screen.

Figure 2-3 Archive All Files



4 *If you do not wish to archive all files*, select Archive selected files and select only the groups you wish to archive as shown in the following screen.

Figure 2-4 Archive Selected Files

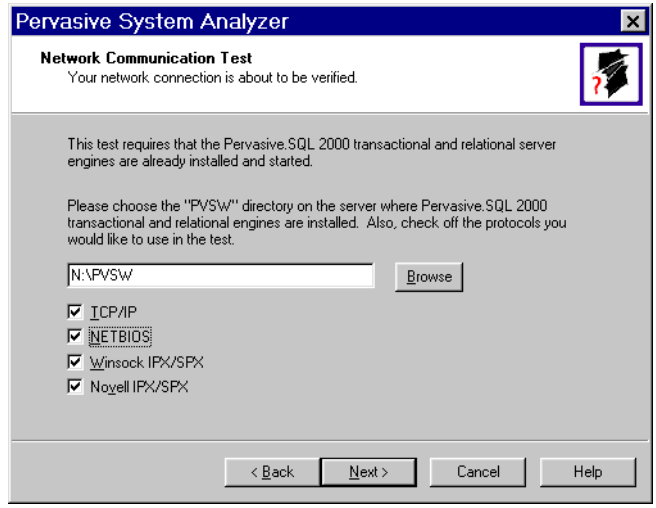


Caution Failure to archive old files may lead to runtime problems after installation.

The files are archived to the selected archive drive, in the path \pvswarch*version*, where *version* describes the product release that was discovered and is being archived.

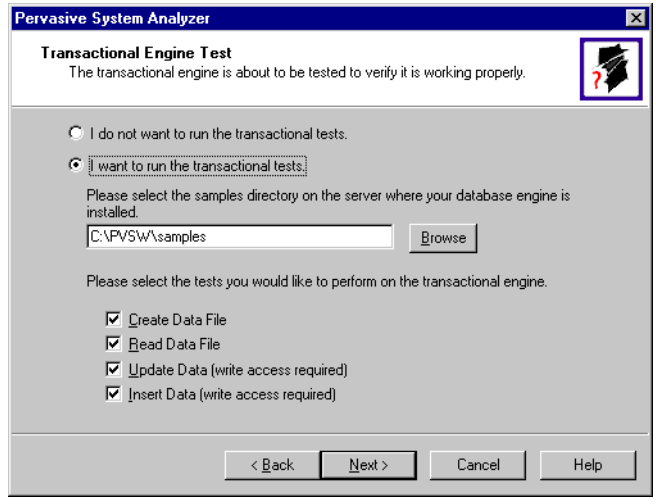
5 After PSA has archived existing files, client installs ONLY perform the Network Connectivity test. If you are installing a Pervasive.SQL server or workstation/workgroup engine locally, then this test is skipped.

Figure 2-5 Network Connectivity Test (CLIENT installs only)



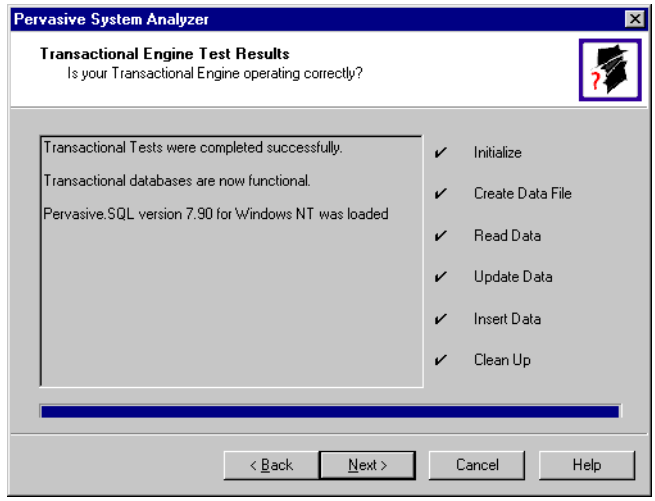
6 The current installation now copies its files to your machine. When the files are finished copying, PSA then displays the Transactional test screen as shown in the following illustration:

Figure 2-6 Start of Transactional Test



7 After completing the Transactional test, the following screen is displayed. If any errors occurred during the test, this screen shows which tests failed and offers additional information to help diagnose the cause.

Figure 2-7 Transactional Test Completed



8 Your Pervasive.SQL installation completes at this point. Please read the README file for important information.

Using PSA Outside of the Installation Process

You may want to use PSA after the installation process if another installation interferes with Pervasive operations.

Why Use PSA?

Here are some scenarios where you might use PSA after installation:

- You installed another application based on Pervasive.SQL and now one or more of your Pervasive-based applications are no longer functioning.
- You are encountering network errors and wish to test your client's connectivity to a machine running a Pervasive.SQL server.
- Your Pervasive-based application is not functioning correctly and you wish to test the functionality of the Transactional or Relational interface components to the Pervasive.SQL engine.
- You wish to restore Pervasive components archived previously by PSA.
- You wish to archive components currently in use on your system in preparation for another install. PSA runs automatically during the installation of Pervasive.SQL 2000 SP3 and higher.

Starting PSA

From the Start Menu:

1. Click **Start**, point to **Programs** | **Pervasive** | **Pervasive.SQL 2000i**
2. Select **Utilities** | **Pervasive System Analyzer**

From the Command Prompt:

1. Click **Start** | **Run**
2. Type `C:\Program Files\Common Files\Pervasive Software\psawizrd` and press Enter.

Common PSA Tasks

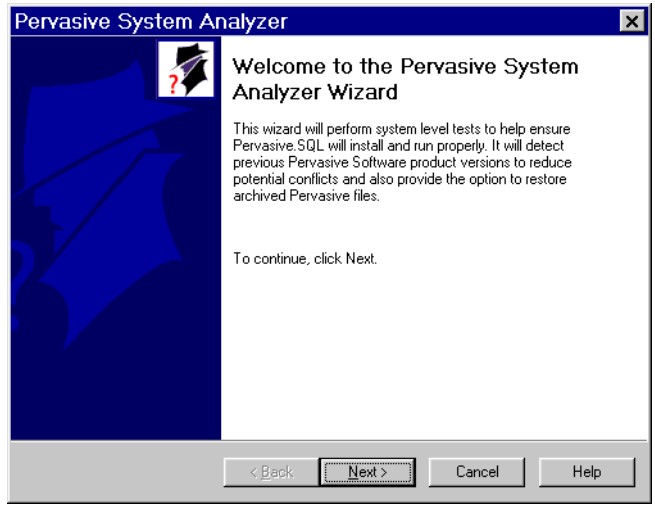
The following shows common tasks that you can perform using PSA:

- "Choosing the Functions You Wish to Perform"
- "Testing Your Network" on page 2-11
- "Testing the Transactional Interface" on page 2-12
- "Archiving Previous Pervasive Components" on page 2-13
- "Restoring Archived Pervasive Components" on page 2-13
- "Viewing PSA Log Information" on page 2-15

Choosing the Functions You Wish to Perform

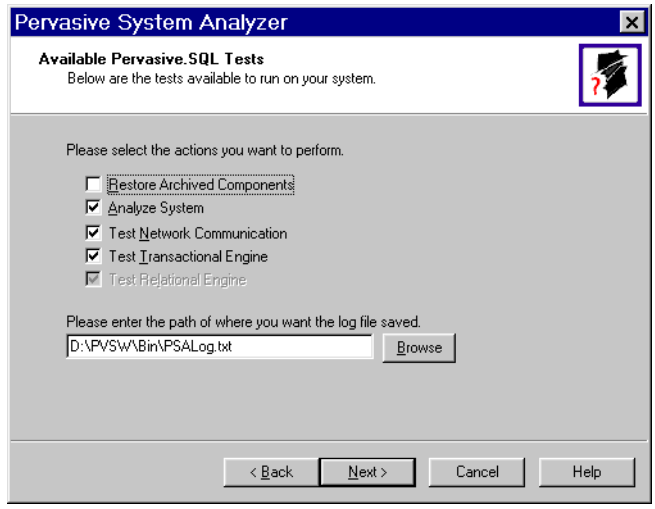
1 PSA starts by displaying its welcome screen. Click **Next** to begin the analysis.

Figure 2-8 PSA Welcome Screen



Click **Next** to select the tests that you wish to perform.

Figure 2-9 PSA Available Tests



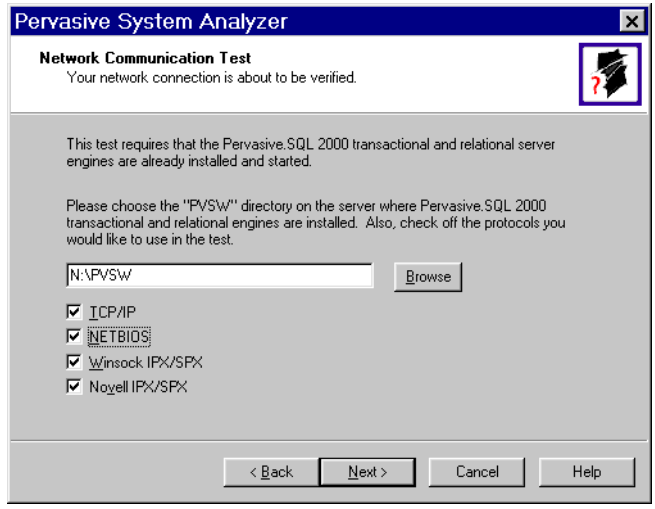
Select the tests you wish to perform. Also, choose the path to which you wish to write the PSA log file. The PSA log can be viewed later using any text editor such as Notepad. You also have the option of viewing it at the end of your PSA session.

If you specify the same file for more than one test, the results of subsequent tests are appended to the file.

Testing Your Network

This test reports on the connectivity to a given machine running Pervasive.SQL.

Figure 2-10 PSA Network Connectivity Test



Note This test only functions if you are connecting to a remote machine running a Pervasive.SQL engine via a mapped drive. This test cannot be run against a local Pervasive.SQL engine.

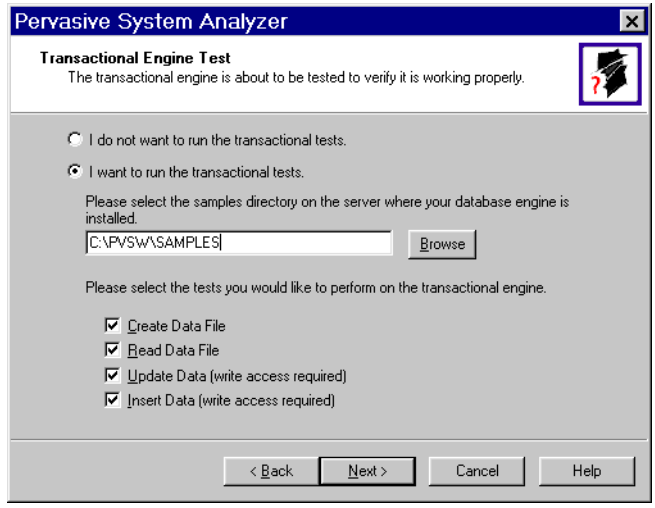
Testing the Transactional Interface

This test verifies the ability of your client interface to connect to the Pervasive.SQL transactional engine (Btrieve).

When you run this test, PSA attempts to perform basic database operations that are common to most Btrieve applications. If your machine passes this test, then the following is verified:

- Your Pervasive.SQL engine is running
- Your client interface components are installed correctly
- Btrieve applications running on your computer should function correctly

Figure 2-11 PSA Transactional Test



Archiving Previous Pervasive Components

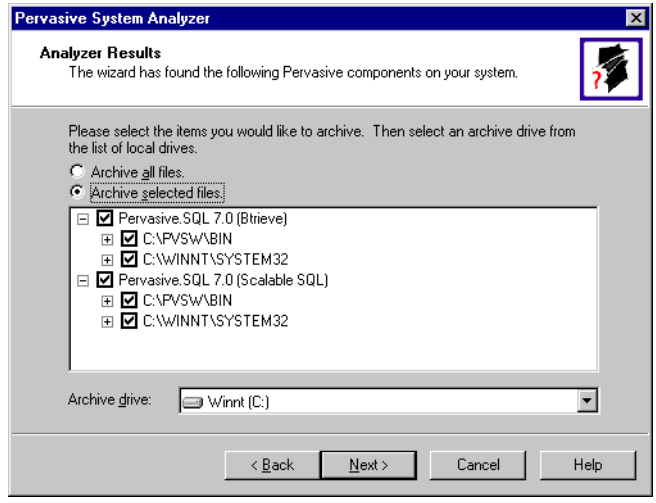
PSA moves the selected files from their current locations to the archive folder.

All archived files are written to the Program Files subdirectory of the selected archive volume.

For example, if you selected C:\ as your archive volume, then Pervasive archived files are stored at:

```
C:\PVSWARCH\
```

Figure 2-12 PSA Archiving



Restoring Archived Pervasive Components

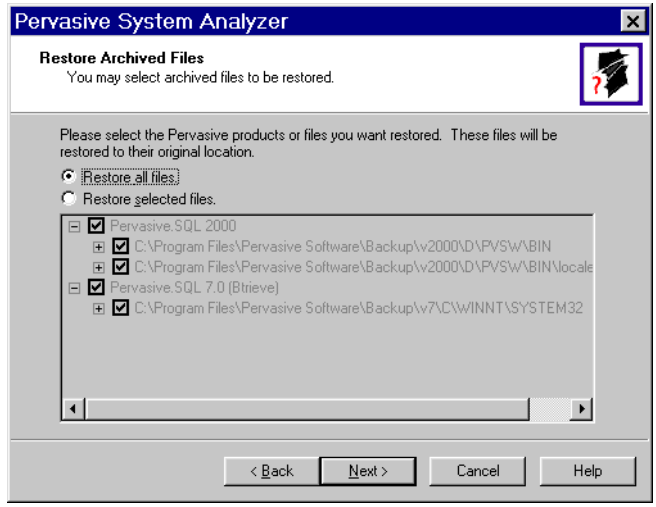
All archived files are written to the Program Files subdirectory of the drive that was selected when the archive was performed.

For example, if the archive was performed to drive D:\ then Pervasive archived files are stored at:

```
D:\PVSWARCH\
```

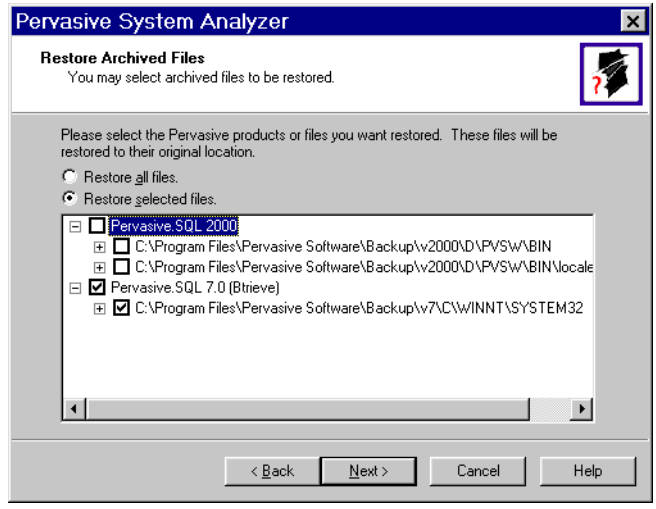
PSA scans this directory and displays the components available for restoration as shown in the following screen.

Figure 2-13 PSA Restore



If you do not wish to restore all components, then select Restore selected files and choose the components that you wish to restore to their previous location.

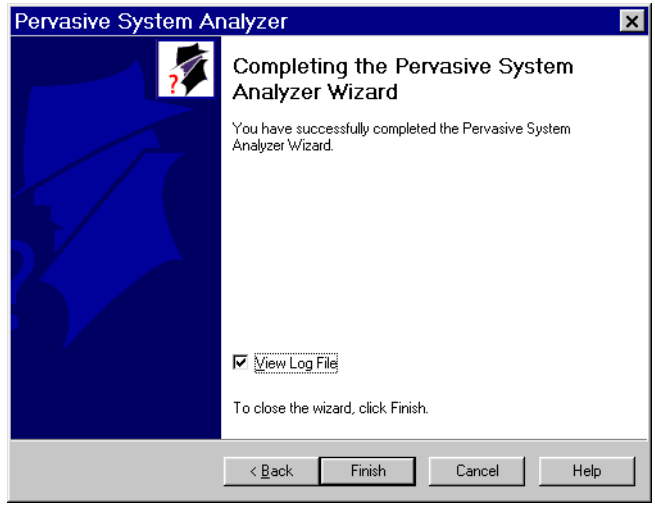
Figure 2-14 PSA Restore Selected Files



Viewing PSA Log Information

The PSA Log contains useful information. You can view this log by selecting the **View Log File** option at the end of your PSA session.

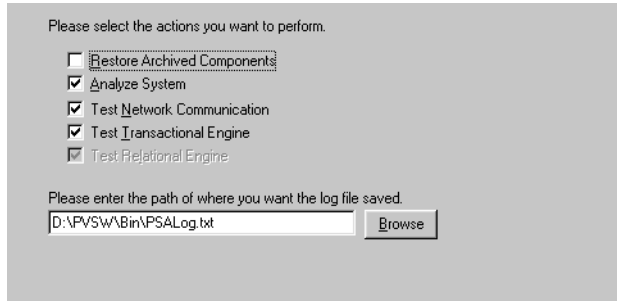
Figure 2-15 PSA Completion and Viewing the Log File



This file is written to the location specified when you started your PSA session and selected which tests you wished to perform.

It is a text file and can be viewed by Notepad or any file editor later if you choose. If the same file name is specified for more than one test run, the results of each test are appended to the file.

Figure 2-16 Specifying the PSA Log File Location



chapter

SQL Syntax Enhancements

3

Detailed Information on New and Improved SQL Syntax

The purpose of this chapter is to provide specific technical detail on the new and improved SQL syntax available in this release of the product.

New SQL syntax in support of Dynamic Cursors is covered in Chapter 4, "Dynamic Cursors in Pervasive.SQL 2000i SP3."

This chapter is divided into the following sections:

- "Global Variables" on page 3-2
- "USING, IN DICTIONARY, WITH REPLACE" on page 3-6
- "SELECT in UPDATE" on page 3-14
- "Improved ALTER TABLE Support" on page 3-17
- "Additional Scalar Functions" on page 3-20

Global Variables

A new global variable, @@IDENTITY, allows you to access the most recent IDENTITY column value inserted by the SRDE for the current database connection. Another new global variable, @@ROWCOUNT, allows you to access the number of rows affected by the most recently executed SQL statement for the current database connection.

Either variable can be prefaced with two at-signs (@@) or an at-sign and a colon (@:). For example, @@IDENTITY and @:IDENTITY are equivalent.

@@IDENTITY

This variable returns the value of the most recently inserted IDENTITY column value (IDENTITY or SMALLIDENTITY). The value is a signed integer value. The initial value is NULL.

This variable can only refer to a single column. If the target table includes more than one IDENTITY column, the value of this variable refers to the IDENTITY column that is the table's primary key. If no such column exists, then the value of this variable refers to the first IDENTITY column in the table.

If the most recent insert was to a table without an IDENTITY column, then the value of @@IDENTITY is set to NULL.

Changed Grammar

Old Grammar

```
query-specification ::=    ( query-specification )
| SELECT [ ALL | DISTINCT ] select-list
  FROM table-reference [ , table-reference ] . . .
  [ WHERE search-condition ]
  [ GROUP BY expression [ , expression ] . . .
  [ HAVING search-condition ] ]
```

New Grammar

```
query-specification ::=    ( query-specification )
| SELECT [ ALL | DISTINCT ] select-list   [ table-expression ]

table-expression ::=
  FROM table-reference [ , table-reference ] . . .
  [ WHERE search-condition ]
  [ GROUP BY expression [ , expression ] . . .
```

```
[ HAVING search-condition ] ]
```

An *expression* is now permitted to contain:

```
@:IDENTITY
| @:ROWCOUNT
| @@IDENTITY
| @@ROWCOUNT
```

Examples

```
SELECT @@IDENTITY
```

Returns NULL if no records have been inserted in the current connection, otherwise returns the IDENTITY column value of the most recently inserted row.

```
SELECT * FROM T1 WHERE @:IDENTITY = 12
```

Returns the most recently inserted row if it has an IDENTITY column value of 12. Otherwise, returns no rows.

```
INSERT INTO T1(C2) VALUES (@@IDENTITY)
```

Inserts the IDENTITY value of the last row inserted into column C2 of the new row.

```
UPDATE T1 SET T1.C1 = (SELECT @@IDENTITY) WHERE T1.C1 =
@@IDENTITY + 10
```

Updates column C1 with the IDENTITY value of the last row inserted, if the value of C1 is 10 greater than the IDENTITY column value of the last row inserted.

```
UPDATE T1 SET T1.C1 = (SELECT NULL FROM T2 WHERE T2.C1 =
@@IDENTITY)
```

Updates column C1 with the value NULL if the value of C1 equals the IDENTITY column value of the last row inserted.

The example below creates a stored procedure and calls it. The procedure sets variable V1 equal to the sum of the input value and the IDENTITY column value of the last row updated. The procedure then deletes rows from the table anywhere column C1 equals V1. The procedure then prints a message stating how many rows were deleted.

```
CREATE PROCEDURE TEST (IN :P1 INTEGER);
   BEGIN
      DECLARE :V1 INTERGER;
          SET :V1 = :P1 + @@IDENTITY;
          DELETE FROM T1 WHERE T1.C1 = :V1;
```

```
                IF (@@ROWCOUNT = 0) THEN
                    PRINT 'No row deleted';
                ELSE
                    PRINT CONVERT(@@ROWCOUNT, SQL_CHAR) +
                        ' rows deleted';
                END IF;
        END;
CALL TEST (@@IDENTITY)
```

@@ROW-COUNT

This variable returns the number of rows that were affected by the most recent operation in the current connection. The value is an unsigned integer. The initial value is zero (0).

Changed Grammar

See "Changed Grammar" for @@IDENTITY.

Examples

```
SELECT @@ROWCOUNT
```

Returns zero (0) if no records were affected by the previous operation in the current connection, otherwise returns the number of rows affected by the previous operation.

```
CREATE TABLE T1 (C1 INTEGER, C2 INTEGER)
INSERT INTO T1 (C1, C2) VALUES (100,200)
INSERT INTO T1(C2) VALUES (100, @@ROWCOUNT)
SELECT * FROM T1
SELECT @@ROWCOUNT FROM T1
```

Results:

```
C1     C2
----   ----
100    200
100    1

2
```

As shown above, the first SELECT generates 2 rows, and shows that the value of @@ROWCOUNT was 1 when it was used to insert a row. The second SELECT returns 2 as the value of @@ROWCOUNT, that is, after the first SELECT returned 2 rows.

If you cut and paste these examples into SQL Data Manager to run them, you must terminate each statement with the appropriate statement delimiter, either the pound sign (#) or the semi-colon (;).

See Also Examples for @@IDENTITY.

USING, IN DICTIONARY, WITH REPLACE

The USING clause is an addition to CREATE TABLE, ALTER TABLE, and DROP TABLE allowing you to specify a file name when manipulating a table. This feature also allows you to make certain changes to the DDFs without affecting the data files. This functionality was available in Pervasive.SQL 7, but until now was not available in Pervasive.SQL 2000.

Changed Grammar

```
CREATE TABLE table-name [IN DICTIONARY] [USING 'path_name'
[WITH REPLACE]]
( table-element [ , table-element ] ... )

ALTER TABLE table-name [IN DICTIONARY] [USING 'path_name'
    [WITH REPLACE]]
alter-option-list

DROP TABLE table-name [IN DICTIONARY]
```

USING

The USING keyword allows you to associate a new or existing table with a particular data file.

Note The USING clause is a powerful feature that can easily be misused. You can use this statement to associate a table definition with simply any file, with no errors. When you attempt to access the file later, however, then you will receive errors if the file is not a MicroKernel data file with exactly the record structure defined in the associated table definition.

When a CREATE TABLE USING or ALTER TABLE USING statement is executed, no check or verification is performed on the contents of the specified file. It is not checked to see if it is already referenced in another table definition in the database, nor is it checked to ensure that it is even readable by the database engine. If you specify a file that has the wrong structure or is not a MicroKernel file, you will not receive any errors until you attempt to access the file.

Path Name Notes

Because Pervasive.SQL requires a Named Database in order to connect, the *path_name* provided must always be a simple file name or relative path and file name. Paths are always relative to the first

Data Path specified for the Named Database to which you are connected.

The path/file name passed is partially validated when SQLPrepare is called. The following rules must be followed when specifying the path name:

- The text must be enclosed in single quotes, per the grammar definition.
- Text must be 1 to 64 characters in length, such that the entry as specified fits in Xf$Loc in X$File. The entry is stored in Xf$Loc exactly as typed (trailing spaces are truncated and ignored).
- The path must be a simple, relative path. Paths that reference a server or volume are not allowed. For NetWare, a volume-based path (such as SYS:/path/testfile.btr) is not considered a simple, relative path.
- Relative paths containing a period (‘.’ - current directory) , double-period (‘..’ - parent directory), slash ‘\’, or any combination of the three are allowed. The path must contain a file name representing the SQL table name (*path_name* cannot end in a slash ‘\’ or a directory name). All file names, including those specified with relative paths, are relative to the first Data Path as defined in the Named Database configuration.
- Root-based relative paths are also allowed. For example, assuming that the first data path is D:\PVSW\DEMODATA, the SRDE interprets the path name in the following statement as D:\TEMP\TEST123.BTR.

  ```
  CREATE TABLE t1 USING '\temp\test123.btr' (c1 int)
  ```

- Slash (‘\’) characters in relative paths may be specified either UNIX style (‘/’) or in the customary backslash notation (‘\’), depending on your preference. You may use a mixture of the two types, if desired. This is a convenience feature since you may know the directory structure scheme, but not necessarily know (or care) what type of server you are connected to. The path is stored in X$File exactly as typed. The SRDE engine converts the slash characters to the appropriate platform type when utilizing the path to open the file. Also, since data files share binary compatibility between all supported platforms, this means that as long as the directory structure is the same between platforms (and path-based file names are specified as relative paths), the

database files and DDFs can be moved from one platform to another with no modifications. This makes for a much simpler cross-platform deployment with a standardized database schema.

- If specifying a relative path, the directory structure in the USING clause must first exist. The SRDE does not create directories to satisfy the path specified in the USING clause.

CREATE TABLE

Include a USING clause to specify the physical location of the data file associated with the table. This is necessary when you are creating a table definition for an existing data file, or when you want to specify explicitly the name or physical location of a new data file.

If you do not include a USING clause, Pervasive.SQL generates a unique file name (based on the table name with the extension .MKD) and creates the data file in the first directory specified in the data file path associated with the database name.

If the USING clause points to an existing data file, the SRDE creates the table in the DDFs and returns SQL_SUCCESS_WITH_INFO. The informational message returned indicates that the dictionary entry now points to an existing data file. If you want CREATE TABLE to return only SQL_SUCCESS, specify IN DICTIONARY on the CREATE statement. If WITH REPLACE is specified (see below), then any existing data file with the same name is destroyed and overwritten with a newly created file.

Note Change in behavior: Pervasive.SQL v7 returned a Status Code 59 if you specified an existing data file. Pervasive.SQL 2000i SP3 returns a successful status code.

Examples

See “Examples” on page 3-12.

ALTER TABLE

Include a USING clause to specify the physical location and name of an existing data file to associate with an existing table. A USING clause also allows you to create a new data file at a particular location using an existing dictionary definition. (The string supplied in the

USING clause is stored in the Xf$Loc column of the dictionary file X$File.)

In the sample database, the Person table is associated with the file PERSON.MKD. If you create a new file named PERSON2.MKD, the statement in the following example changes the dictionary definition of the Person table so that the table is associated with the new file.

```
ALTER TABLE Person IN DICTIONARY USING 'person2.mkd'#
```

You must use either a simple file name or a relative path in the USING clause. If you specify a relative path, Pervasive.SQL interprets it relative to the data file path associated with the database name.

For Pervasive.SQL 2000i, the USING clause functions the same as it did in Pervasive.SQL v7, but with the following changes and improvements in behavior:

- The USING clause can be specified in addition to any other standard ALTER TABLE option. This means columns can be manipulated in the same statement that specifies the USING path.
- Previously in Pervasive.SQL v7, whenever ALTER TABLE USING was specified without IN DICTIONARY, and the specified file did not exist, the specified file was created as a new, empty file. Even though the old physical Btrieve file still existed, no data was transferred to the new file.

  Because Pervasive.SQL 2000i allows all other ALTER TABLE options at the same time as USING, the new data file created is fully populated with the existing tables data. The file structure is not simply copied, but instead the entire contents are moved over, similar to a Btrieve BUTIL -CREATE and BUTIL -COPY. This can be helpful for rebuilding a SQL table, or compressing a file that once contained a large number of records but now contains only a few. The original, physical data file (previously pointed to by the table) remains intact.

Note Change in behavior: When Pervasive.SQL v7 performed an ALTER TABLE USING, the new data file it created contained no data. Pervasive.SQL 2000i copies the contents of the existing data file into the newly specified data file, leaving the old data file intact but unlinked.

Examples

See "Examples" on page 3-13.

IN DICTIONARY

The purpose of using this keyword is to notify the SRDE that you wish to make modifications to the DDFs, while leaving the underlying physical data unchanged. IN DICTIONARY is a very powerful and advanced feature. It should only be used by system administrators or when absolutely necessary. Normally, the SRDE keeps DDFs and data files totally synchronized, but this feature allows users the flexibility to force table dictionary definitions to match an existing data file. This can be useful when you want to create a definition in the dictionary to match an existing data file, or when you want to use a USING clause to change the data file path name for a table.

You cannot use this keyword on a bound database.

The following text outlines the differences between this implementation and the previous implementation:

- IN DICTIONARY is now allowed on CREATE and DROP TABLE, in addition to ALTER TABLE. It was only available on ALTER TABLE in Pervasive.SQL v7. This change was made to allow Pervasive.SQL 2000i users greater flexibility over their table definitions via SQL statements.
- In Pervasive.SQL 2000i, IN DICTIONARY affects dictionary entries *only*, no matter what CREATE/ALTER options are specified. Previously, Pervasive.SQL 7 only allowed one option at a time for ALTER, and for primary or foreign key alterations, this option was ignored. Since Pervasive.SQL 2000i allows multiple options (any combination of ADD, DROP, ADD CONSTRAINT, and so on), IN DICTIONARY is honored under all circumstances to guarantee only the DDFs are affected by the schema changes.

Remarks

Tables that exist in the DDFs only (the data file does not exist) are called *detached* entries. These tables are inaccessible via queries or other operations that attempt to open the physical underlying file. For this reason, IN DICTIONARY was added to DROP TABLE, because it is now possible to create detached entries using CREATE TABLE. Note that errors such as "Table not found" are generated by

attempts to access these detached entries. One can verify whether a table really exists by using SQLTables or directly querying the Xf$Name column of X$File:

```
SELECT * FROM X$File WHERE Xf$Name = 'table_name'
```

It is possible for a detached table to cause confusion, so the IN DICTIONARY feature must be used with extreme care. It is crucial that it should be used to force table definitions to match physical files, not to detach them. Consider the following examples, assuming that the file test123.btr does not exist:

```
CREATE TABLE t1 USING 't1.btr' (c1 INT)
ALTER TABLE t1 IN DICTIONARY USING 'test123.btr'
```

Or, combining both statements:

```
CREATE TABLE t1 IN DICTIONARY USING 'test123.btr' (c1
INT)
```

If you then attempt to SELECT from t1, you receive an error that the table was not found. Confusion can arise, because you just created the table—how can it not be found? Likewise, if you attempt to DROP the table without specifying IN DICTIONARY, you receive the same error. These errors are generated because there is no data file associated with the table.

WITH REPLACE

Whenever WITH REPLACE is specified in conjunction with the USING keyword, Pervasive.SQL automatically overwrites any existing file content with the specified file content. The existing file is always overwritten as long as the operating system allows it.

WITH REPLACE only affects the data file, it never affects the DDFs.

The following rules apply when using WITH REPLACE:

- WITH REPLACE can only be used with USING.
- When used with IN DICTIONARY, WITH REPLACE is ignored because IN DICTIONARY specifies that only the DDFs are affected.

CREATE TABLE

If you include WITH REPLACE in your CREATE TABLE statement, Pervasive.SQL creates a new data file to replace the existing file (if the file exists at the location you specified in the USING clause). Pervasive.SQL discards any data stored in the original file with the

same name. If you do not include WITH REPLACE and a file exists at the specified location, Pervasive.SQL returns SQL_SUCCESS_WITH_INFO and does not create a new file.

WITH REPLACE affects only the data file; it does not affect the table definition in the dictionary.

Examples

Example A

```
CREATE T1 USING 'f1.mkd' (c1 int)
```

If f1.mkd does not exist, the file is created.

If f1.mkd already exists, then the table definition for table T1 is associated with the designated file. The file is not verified in any way, nor is any association with an existing table definition canceled. That is, if f1.mkd already exists and already has a table definition associated with it, associating another table definition with it will succeed so that both table definitions access the same data file. If f1.mkd is not a MicroKernel file, or does not contain the record structure defined in the table definition, errors will occur only when you attempt to access table T1.

Example B

```
CREATE T1 USING 'f1.mkd' WITH REPLACE (c1 int)
```

If f1.mkd does not exist, the file is created.

If f1.mkd already exists, then the existing file is deleted, and a new, empty file with the same name is created in its place.

ALTER TABLE

Include WITH REPLACE in a USING clause to instruct Pervasive.SQL to replace an existing file (the file must reside at the location you specified in the USING clause). If you include WITH REPLACE, Pervasive.SQL creates a new file and copies all the data from the existing file into it. If you do not include WITH REPLACE and a file exists at the specified location, Pervasive.SQL returns a status code and does not create the new file.

Note Change in behavior: With Pervasive.SQL 2000i, no data is lost or discarded when WITH REPLACE is used with ALTER TABLE. The newly created data file, even if overwriting an existing file, still contains all data from the previous file. You cannot lose data by issuing an ALTER TABLE command.

Examples

Example A

```
ALTER TABLE T1 USING 'f1.mkd'
```

If f1.mkd does not exist, the file is created and the data in the previous data file associated with table T1 is copied into f1.mkd. The previous data file is not deleted, but any future changes to table T1 are written to f1.mkd.

If f1.mkd already exists, no action is taken and the database engine returns status code 59.

Example B

```
ALTER TABLE T1 USING 'f1.mkd' WITH REPLACE
```

If f1.mkd does not exist, the file is created and the data in the previous data file associated with table T1 is copied into f1.mkd. The previous data file is not deleted, but any future changes to table T1 are written to f1.mkd.

If f1.mkd already exists, its contents are erased and the data in the previous data file associated with table T1 is copied into f1.mkd. The previous data file is not deleted, but any future changes to table T1 are written to f1.mkd.

SELECT in UPDATE

In the SET clause of an UPDATE statement, you may specify a subquery. This feature allows you to update information in a table based on information in another table or another part of the same table. This functionality was available in Pervasive.SQL v7, but until now was not available in Pervasive.SQL 2000.

The UPDATE statement can only update a single table at a time. UPDATE can relate to other tables via a subquery in the SET clause. This can be a *correlated* subquery that depends in part on the contents of the table being updated, or it can be a *non-correlated* subquery that depends only on another table.

Correlated Subquery

```
UPDATE T1 SET T1.C2 = (SELECT T2.C2 FROM T2 WHERE T2.C1
= T1.C1)
```

Non-correlated Subquery

```
UPDATE T1 SET T1.C2 = (SELECT SUM(T2.C2) FROM T2 WHERE
T2.C1 = 10)
```

Changed Grammar

The following SQL grammar addition has been implemented to support this feature:

Old Grammar

```
UPDATE table-name [ alias-name ]
SET column-name = expression [ , column-name = expression ] . . .
[ WHERE search-condition ]
```

New Grammar

```
UPDATE table-name [ alias-name ]
SET column-name = expression-or-subquery [ , column-name = expression-
    or-subquery] . . .
[ WHERE search-condition ]
```

Remarks

The same logic is used to process pure SELECT statements and subqueries, so the subquery can consist of any valid SELECT statement. There are no special rules for subqueries.

If SELECT within an UPDATE returns no rows, then the UPDATE inserts NULL. If the given column(s) is/are not nullable, then the

UPDATE fails. If select returns more than one row, then UPDATE fails.

Comparison to Pervasive.SQL 7 Behavior

The UPDATE syntax in the Pervasive.SQL v7 engine is not the same as that of Pervasive.SQL 2000i. Although the previous UPDATE syntax was not ANSI compliant, the v7 engine allowed several join tables in an UPDATE statement. For example, the following statement was permitted:

```
UPDATE T1, T2 SET T1.C2 = T2.C2 WHERE T1.C1 = T2.C1
```

This syntax is not supported by the SRDE. The above query needs to be rewritten via a correlated subquery in the SET clause as follows:

```
UPDATE T1 SET T1.C2 = (SELECT T2.C2 FROM T2 WHERE T2.C1
= T1.C1)
```

Examples

Example A

In Example A, two tables are created and rows are inserted. The first table, t5, is updated with a column value from the second table, t6, in each row where table t5 has the value 2 for column c1. Because there is more than one row in table t6 containing a value of 3 for column c2, the first UPDATE fails because more than one row is returned by the subquery. This result occurs even though the result value is the same in both cases. As shown in the second UPDATE, using the DISTINCT keyword in the subquery eliminates the duplicate results and allows the statement to succeed.

```
CREATE TABLE t5 (c1 INT, c2 INT)
CREATE TABLE t6 (c1 INT, c2 INT)
INSERT INTO t5(c1, c2) VALUES (1,3)
INSERT INTO t5(c1, c2) VALUES (2,4)

INSERT INTO t6(c1, c2) VALUES (2,3)
INSERT INTO t6(c1, c2) VALUES (1,2)
INSERT INTO t6(c1, c2) VALUES (3,3)

select * from t5
c1          c2
----------- -----
1           3
2           4

UPDATE t5 SET t5.c1=(SELECT c2 FROM t6 WHERE c2=3) WHERE
    t5.c1=2 -- query fails
```

```
UPDATE t5 SET t5.c1=(SELECT DISTINCT c2 FROM t6 WHERE
    c2=3) WHERE t5.c1=2 -- query succeeds

SELECT * FROM t5
c1          c2
----------- -----
1           3
3           4
```

Example B

In Example B, two tables are created and a variety of valid syntax examples are demonstrated. Note the cases where UPDATE fails because the subquery returns more than one row. Note that UPDATE succeeds and NULL is inserted if the subquery returns no rows (where NULL values are allowed).

```
CREATE TABLE T1 (C1 INT, C2 INT)
CREATE TABLE T2 (C1 INT, C2 INT)

INSERT INTO T1 VALUES (1, 0)
INSERT INTO T1 VALUES (2, 0)
INSERT INTO T1 VALUES (3, 0)
INSERT INTO T2 VALUES (1, 100)
INSERT INTO T2 VALUES (2, 200)

UPDATE T1 SET T1.C2 = (SELECT SUM(T2.C2) FROM T2)
UPDATE T1 SET T1.C2 = 0
UPDATE T1 SET T1.C2 = (SELECT T2.C2 FROM T2 WHERE T2.C1
    = T1.C1)

UPDATE T1 SET T1.C2 = @@IDENTITY
UPDATE T1 SET T1.C2 = @@ROWCOUNT
UPDATE T1 SET T1.C2 = (SELECT @@IDENTITY)
UPDATE T1 SET T1.C2 = (SELECT @@ROWCOUNT)

UPDATE T1 SET T1.C2 = (SELECT T2.C2 FROM T2) -- update fails
INSERT INTO T2 VALUES (1, 150)
INSERT INTO T2 VALUES (2, 250)
UPDATE T1 SET T1.C2 = (SELECT T2.C2 FROM T2 WHERE T2.C1
    = T1.C1) -- update fails
UPDATE T1 SET T1.C2 = (SELECT T2.C2 FROM T2 WHERE T2.C1
    = 5) -- update succeeds, NULL is inserted for all rows of T1.C2

UPDATE T1 SET T1.C2 = (SELECT SUM(T2.C2) FROM T2 WHERE
    T2.C1 = T1.C1)
```

Improved ALTER TABLE Support

In Pervasive.SQL v7, you could change the nullability or the data type of an existing column in a table. In more recent releases of Pervasive.SQL 2000i, the same capability was not available. In this release of the product, the capability has been restored and enhanced. You can now modify the data type or NULL property of an existing column.

Changed Syntax

The grammar that has been added to support this feature is shown in bold type below:

```
ALTER TABLE table-name alter-option-list

table-name ::=user-defined-name

alter-option-list ::=alter-option
| (alter-option [, alter-option]...)

alter-option ::= ADD [ COLUMN ] column-definition
| ADD table-constraint-definition
| DROP [ COLUMN ] column-name
| DROP CONSTRAINT constraint-name
| DROP PRIMARY KEY
| MODIFY [ COLUMN ] column-definition
| ALTER [ COLUMN ] column-definition
```

Remarks

The ability to modify the nullability or data type of a column is subject to the following restrictions:

- The target column cannot have a PRIMARY/FOREIGN KEY constraint defined on it.
- If converting the old type to the new type causes an overflow (arithmetic or size), the ALTER TABLE operation is aborted.
- If a nullable column contains NULL values, the column cannot be changed to a non-nullable column.

If you must change the data type of a key column, you can do so by dropping the key, changing the data type, and re-adding the key. Keep in mind that you must ensure that all associated key columns remain synchronized. For example, if you have a primary key in table T1 that is referenced by foreign keys in tables T2 and T3, you must first drop the foreign keys. Then you can drop the primary key. Then

you need to change all three columns to the same data type. Finally, you must re-add the primary key and then the foreign keys.

The syntax supported in Pervasive.SQL v7 is different from ANSI standard. Pervasive.SQL v7 uses the MODIFY keyword while ANSI uses the ALTER keyword. Pervasive.SQL 2000i allows both keywords in the ALTER TABLE statement as follows:

```
ALTER TABLE T1 MODIFY C1 INTEGER
ALTER TABLE T1 ALTER C1 INTEGER
ALTER TABLE T1 MODIFY COLUMN C1 INTEGER
ALTER TABLE T1 ALTER COLUMN C1 INTEGER
```

Comparison to Pervasive.SQL v7 Behavior

Version 7 does not allow altering a column to a smaller length regardless of whether the actual data can be converted. Pervasive.SQL 2000i allows altering a column to a smaller length if the actual data does not overflow the new, smaller length of the column. This behavior is similar to that of Microsoft SQL Server.

Pervasive.SQL 2000i allows adding, dropping, or modifying multiple columns on a single ALTER TABLE statement. Although it simplifies operations, this behavior is not considered ANSI-compliant. Pervasive.SQL v7 only allowed you to add, drop, or modify a single column at a time. A sample multi-column ALTER statement is shown below.

```
ALTER TABLE T1 (ALTER C2 INT, ADD D1 CHAR(20), DROP C4,
ALTER C5 LONGVARCHAR, ADD D2 LONGVARCHAR NOT NULL)
```

Conversion of Legacy Data Types

All legacy data types (Pervasive.SQL v7 or earlier) can be converted to data types that are natively supported by Pervasive.SQL 2000i. But the new data types cannot be converted backwards to legacy data types.

A LONGVARCHAR/LONGVARBINARY column is no longer allowed to be added to a legacy table that contains a NOTE/LVAR column. In order to add a LONGVARCHAR/LONGVARBINARY column to a legacy table that contains a NOTE/LVAR column, the NOTE/LVAR column has to first be converted to a LONGVARCHAR or LONGVARBINARY column. After converting the NOTE/LVAR column to LONGVARCHAR/LONGVARBINARY, more LONGVARCHAR/LONGVARBINARY columns can be added to the

table. But the legacy engine will not be able to work with this legacy table because the old engine can only work with one variable length column per table.

Examples

```
ALTER TABLE T1 (ALTER C2 INT, ADD D1 CHAR(20), DROP C4,
ALTER C5 LONGVARCHAR, ADD D2 LONGVARCHAR NOT NULL)

ALTER TABLE T2 (ALTER C1 CHAR(50), DROP CONSTRAINT
    MY_KEY, DROP PRIMARY KEY, ADD MYCOLUMN INT)
```

Additional Scalar Functions

This section describes the additional scalar functions that are available in this release. They are divided into the following categories:

- String Functions on page 3-20
- Numeric Functions on page 3-22
- Date and Time Functions on page 3-25
- Logical Functions on page 3-28

String Functions

String functions are used to process and manipulate columns that consist of text information, such as CHAR or LONGVARCHAR data types.

ASCII

`ASCII (`*string_exp*`)`

Returns the ASCII value of the leftmost character of *string_exp*.

Example

This example creates a new table with an integer and a character column. It inserts 4 rows with values for the character column only, then updates the integer column of those rows with the ASCII character code for each character.

```
CREATE TABLE numchars(num INTEGER, chr CHAR(1) CASE)

INSERT INTO numchars (chr) VALUES('a')
INSERT INTO numchars (chr) VALUES('b')
INSERT INTO numchars (chr) VALUES('A')
INSERT INTO numchars (chr) VALUES('B')

UPDATE numchars SET num=ASCII(chr)

SELECT * FROM numchars
```

Results of SELECT:

```
num         chr
----------- ---
97          a
98          b
65          A
66          B
```

```
SELECT num FROM numchars WHERE num=ASCII('a')
```

Results of SELECT:

```
num
------
97
```

BIT_LENGTH

`BIT_LENGTH (`*string_exp*`)`

Returns the length in bits of *string_exp*.

CHAR

`CHAR (`*Code*`)`

Returns the ASCII character corresponding to ASCII value *Code*. The argument must be an integer value.

CHAR_LENGTH

`CHAR_LENGTH(`*string_exp*`)`

Returns the number of characters in *string_exp*.

CHARACTER_LENGTH

`CHARACTER_LENGTH(`*string_exp*`)`

Same as CHAR_LENGTH.

OCTET_LENGTH

`OCTET_LENGTH (`*string_exp*`)`

Returns the length in bytes of *String_Exp*.

POSITION

`POSITION(`*string_exp1*`, `*string_exp2*`)`

Returns the position of *string_exp1* in *string_exp2*. If *string_exp1* does not exist in *string_exp2*, a zero is returned.

REPLICATE

`REPLICATE(`*string_exp*`, `*count*`)`

Returns a character string composed of *string_exp* repeated *count* times. The value of *count* is an integer.

REPLACE

`REPLACE(`*string_exp1*`, `*string_exp2*`, `*string_exp3*`)`

Searches *string_exp1* for occurrences of *string_exp2* and replaces each with *string_exp3*. Returns the result. If no occurrences are found, *string_exp1* is returned.

If *string_exp3* is empty, *string_exp1* is returned. If *string_exp2* is longer than *string_exp1*, *string_exp1* is returned.

SPACE

`SPACE(`*count*`)`

Returns a character string consisting of *count* spaces.

STUFF

`STUFF(`*string_exp1*`, `*start*`, `*length*`, `*string_exp2*`)`

Returns a character string where *length* characters in *string_exp1* beginning at position *start* have been replaced by *string_exp2*. The values of *start* and *length* are integers. If *length* is greater than the size of *string_exp1* or *string_exp2*, the replacement occurs to the end of *string_exp1* or *string_exp2*, whichever is greater. If *start* is beyond the length of *string_exp1*, nothing is returned.

Numeric Functions

Numeric functions are used to process and manipulate columns that consist of strictly numeric information, such as decimal and integer values.

ABS

`ABS(`*numeric_exp*`)`

Returns the absolute value of *numeric_exp*.

ACOS

`ACOS(`*float_exp*`)`

Returns the arccosine of *float_exp* as an angle, expressed in radians.

ASIN

`ASIN` (*float_exp*)

Returns the arcsine of *float_exp* as an angle, expressed in radians.

ATAN

`ATAN` (*float_exp*)

Returns the arctangent of *float_exp* as an angle, expressed in radians.

ATAN2

`ATAN2` (*float_exp1*, *float_exp2*)

Returns the arctangent of the x and y coordinates, specified by *float_exp1* and *float_exp2*, respectively, as an angle, expressed in radians.

CEILING

`CEILING` (*numeric_exp*)

Returns the smallest integer greater than or equal to *numeric_exp*.

COS

`COS` (*float_exp*)

Returns the cosine of *float_exp*, where *float_exp* is an angle expressed in radians.

COT

`COT` (*float_exp*)

Returns the cotangent of *float_exp*, where *float_exp* is an angle expressed in radians.

DEGREES

`DEGREES` (*numeric_exp*)

Converts *numeric_exp* radians to degrees and returns the result.

EXP

`EXP` (*float_exp*)

Returns the value of *e* raised to the power of *float_exp*.

FLOOR

`FLOOR(`*numeric_exp*`)`

Returns the largest integer less than or equal to *numeric_exp*.

LOG

`LOG(`*float_exp*`)`

Returns the natural logarithm (base *e*) of *float_exp*.

LOG10

`LOG10(`*float_exp*`)`

Returns the base 10 logarithm of *float_exp*.

PI

`PI()`

Returns the constant value Pi as a floating point value.

POWER

`POWER(`*numeric_exp*`, `*integer_exp*`)`

Returns the value of *numeric_exp* to the power of *integer_exp*.

RADIANS

`RADIANS(`*numeric_exp*`)`

Converts *numeric_exp* degrees to radians and returns the result.

RAND

`RAND([`*integer_exp*`])`

Returns a random floating-point value using *integer_exp* as the optional seed value.

ROUND

`ROUND(`*numeric_exp*`, `*integer_exp*`)`

Returns *numeric_exp* rounded to *integer_exp* places right of the decimal point. If *integer_exp* is negative, *numeric_exp* is rounded to |*integer_exp*| (absolute value of *integer_exp*) places to the left of the decimal point.

SIGN

`SIGN(`*numeric_exp*`)`

Returns an indicator of the sign of *numeric_exp*. If *numeric_exp* is less than zero, -1 is returned. If *numeric_exp* equals zero, 0 is returned. If *numeric_exp* is greater than zero, 1 is returned.

SIN

`SIN(`*float_exp*`)`

Returns the sine of *float_exp*, where *float_exp* is an angle expressed in radians.

SQRT

`SQRT(`*float_exp*`)`

Returns the square root of *float_exp*.

TAN

`TAN(`*float_exp*`)`

Returns the tangent of *float_exp*, where *float_exp* is an angle expressed in radians.

TRUNCATE

`TRUNCATE(`*numeric_exp*`, `*integer_exp*`)`

Returns *numeric_exp* truncated to *integer_exp* places right of the decimal point. If *integer_exp* is negative, *numeric_exp* is truncated to |*integer_exp*| (absolute value) places to the left of the decimal point.

Date and Time Functions

Date and time functions can be used to generate, process, and manipulate data that consists of date or time data types, such as DATE and TIME.

CURRENT_DATE

`CURRENT_DATE()`

Returns the current date in format yyyy-mm-dd. In INSERT statements, you should use the CURDATE variable in the values clause to insert the current date into a table.

CURRENT_TIME

```
CURRENT_TIME()
```

Returns the current time in format hh:mm:ss. In INSERT statements, you should use the CURTIME variable in the values clause to insert the current time into a table.

DAYNAME

```
DAYNAME(date_exp)
```

Returns a character string containing the data source-specific name of the day (for example, Sunday through Saturday or Sun. through Sat. for a data source that uses English, or Sonntag through Samstag for a data source that uses German) for the day portion of *date_exp*.

DAYOFYEAR

```
DAYOFYEAR(date_exp)
```

Returns the day of the year based on the year field in *date_exp* as an integer value in the range of 1-366.

EXTRACT

```
EXTRACT(extract_field, extract_source)
```

Returns the *extract_field* portion of the *extract_source*. The *extract_source* argument is a date, time or interval expression.

The permitted values of *extract_field* are as follows:

Table 3-1 EXTRACT Field Codes

Code	Description
YEAR	Specifies the function should return the year field from the target expression.
MONTH	Specifies the function should return the month field from the target expression.
DAY	Specifies the function should return the day-of-month field from the target expression.
HOUR	Specifies the function should return the hours field from the target expression.

Table 3-1 EXTRACT Field Codes

Code	Description
MINUTE	Specifies the function should return the minutes field from the target expression.
SECOND	Specifies the function should return the seconds field from the target expression.

Example

To extract the year from the current date, you can use the following statement:

```
SELECT EXTRACT(YEAR, CURRENT_DATE()) FROM "room"
```

MONTHNAME

`MONTHNAME(`*date_exp*`)`

Returns a character string containing the data source-specific name of the month (for example, September through December or Sept. through Dec. for a data source that uses English, or Settembre through Dicembre for a data source that uses Italian) for the month portion of *date_exp*.

QUARTER

`QUARTER()`

Returns the quarter in *date_exp* as an integer value in the range of 1-4, where 1 represents January 1 through March 31.

TIMESTAMPADD

`TIMESTAMPADD(`*interval*`,` *integer_exp*`,` *timestamp_exp*`)`

Returns the timestamp calculated by adding *integer_exp* intervals of type *interval* to *timestamp_exp*. The allowed values for *interval* are specified here:

Table 3-2 TIMESTAMP Interval Types

Type Code	Description
SQL_TSI_YEAR	Specifies the interval as a year.
SQL_TSI_QUARTER	Specifies the interval as a calendar quarter.

Table 3-2 TIMESTAMP Interval Types

Type Code	Description
SQL_TSI_MONTH	Specifies the interval as a month.
SQL_TSI_WEEK	Specifies the interval as a week.
SQL_TSI_DAY	Specifies the interval as a day.
SQL_TSI_HOUR	Specifies the interval as a hour.
SQL_TSI_MINUTE	Specifies the interval as a minute.
SQL_TSI_SECOND	Specifies the interval as a second.

TIMESTAMPDIFF

`TIMESTAMPDIFF(`*interval*`, `*timestamp_exp1*`, `*timestamp_exp2*`)`

Returns the integer number of intervals of type *interval* by which *timestamp_exp2* is greater than *timestamp_exp1*. If *timestamp_exp2* is earlier than *timestamp_exp1*, the returned value is negative. For a description of permitted values of interval, see Table 3-2, "TIMESTAMP Interval Types."

WEEK

`WEEK(`*date_exp*`)`

Returns the week of the year based on the week field in *date_exp* as an integer value in the range of 1-53.

Logical Functions

Logical functions are used to manipulate data based on certain conditions.

IFNULL

`IFNULL(`*exp*`, `*value*`)`

If *exp* is null, *value* is returned. If *exp* is not null, *exp* is returned. The possible data type or types of *value* must be compatible with the data type of *exp*.

NULLIF

`NULLIF(`*exp1*`, `*exp2*`)`

NULLIF returns *exp1* if the two expressions are not equivalent. If the expressions are equivalent, NULLIF returns a null value.

Utility Functions

USER

```
USER()
```

This function returns the login name of the current user.

chapter

4

Dynamic Cursors in Pervasive.SQL 2000i SP3

An Overview of the New Dynamic Cursor Functionality in Pervasive.SQL 2000i SP3

This chapter explains the effects and ramifications of the dynamic cursors support that has been added in this service pack. This chapter is divided into the following sections:

- "Features at a Glance" on page 4-2
- "Overview of Dynamic Cursors and the ODBC API" on page 4-3
- "ODBC APIs Affected by New Functionality" on page 4-5
- "Temporary Tables" on page 4-12
- "Positioned UPDATE and DELETE" on page 4-14

Features at a Glance

Dynamic cursors are now supported in Pervasive.SQL 2000. The table below summarizes the changes that have been made.

Table 4-1 Summary of New Functionality with Pervasive dynamic cursors

Feature Element	Notes
Cursor and concurrency types	Not previously supported by driver. Now they are.
SQLExtendedFetch	Now fully supported by Pervasive driver.
SQLSetPos	Not supported by ODBC cursor library, but is now fully supported by Pervasive ODBC driver.
positioned UPDATE and DELETE	Now supported by Pervasive driver.
OLE DB developers	OLE DB 2.5 Command objects now supported.
JDBC developers	JDBC 2.0 API now supported with bi-directional cursors.

Note Information relevant to developing Pervasive.SQL applications discussed in this chapter supersedes that of the Pervasive.SQL 2000 SDK documentation found in Service Pack 2. The SDK documentation will be updated after the release of Pervasive.SQL 2000i Service Pack 3.

Overview of Dynamic Cursors and the ODBC API

This section provides an overview of the changes made to the Pervasive ODBC driver in the Pervasive.SQL 2000i (SP3) release.

Terminology

Table 4-2 Dynamic Cursors Terminology

Term	Definition
SP3 driver	The ODBC driver of Pervasive.SQL 2000i Service Pack 3.
Pre-SP3 driver	The ODBC driver of Pervasive.SQL 2000 prior to Service Pack 3.
Result set	A result set is the set of all rows that exist in the database that match the specifications of a query. A result set is the same as a record set in ADO, or a rowset in OLE DB.
Cursor set	A cursor set is a subset of a result set. A cursor set is a window on the result set that moves as the application makes fetch requests. A cursor set is the same as active rows in ADO or OLE DB. A cursor set is also the same as ODBC's definition of rowset. Specifically, the cursor set is the set of rows retrieved by the most recent SQLExtendedFetch call.
ODBC cursor library	A dynamic-link library (DLL) that resides between the Driver Manager and the driver. When an application calls a function, the Driver Manager calls the function in the cursor library, which either executes the function or calls it in the specified driver. For a given connection, an application specifies whether the cursor library is always used, is used if the driver does not support scrollable cursors, or is never used.

ODBC Cursor Library

Previous versions of our Pervasive.SQL 2000 ODBC driver relied on the ODBC cursor library to provide scrollable cursors via the SQLExtendedFetch API. The primary limitation of the ODBC cursor library is that it does not support dynamic cursors, cursors that display the most current image of the data when scrolling through a result set rather than a static image. The static image used by the ODBC cursor library may not reflect the actual state of the database, in terms of actual data values in a given row in a given table, at any given time.

The ODBC driver of the Pervasive.SQL 2000i (Service Pack 3) release provides all the functionality provided by the ODBC cursor library, while adding support for additional cursor types, concurrency control types and full support for the SQLSetPos API.

ODBC APIs that are Affected

The APIs affected by this new feature are listed here. The differences are described in more detail in the section that follows.

- "SQLSetConnectOption" on page 4-5
- "SQLGetFunctions" on page 4-6
- "SQLGetInfo" on page 4-7
- "SQLSetStmntOption" on page 4-8
- "SQLExtendedFetch" on page 4-10
- "SQLSetPos" on page 4-10

Temporary Tables

Temporary tables are used internally to process queries when an index is required but one is not available. For example, the query

```
SELECT c1, c2, c3 FROM t1 ORDER BY c1
```

will require the use of a temporary table unless column c1 is specified as the leading segment in an index.

The internal processing of temporary tables has changed. These changes may affect both the performance and/or the functionality of some queries. These changes are described in detail in a subsequent section in this document.

ODBC APIs Affected by New Functionality

This section lists the APIs with behavioral differences from the previous version.

- "Updated ODBC Functionality" on page 4-5
- "New ODBC Functionality" on page 4-10

Updated ODBC Functionality

This section describes ODBC functionality that was previously supported but now has changed behavior.

SQLSetConnectOption

The option of interest is SQL_ODBC_CURSORS.

For both the pre-SP3 driver and the SP3 driver, SQL_CUR_USE_ODBC causes the ODBC cursor library to be used while SQL_CUR_USE_DRIVER bypasses the ODBC cursor library and forces all calls to go directly to the ODBC driver.

The difference involves SQL_CUR_USE_IF_NEEDED. Because the pre-SP3 driver did not support the SQL_FETCH_PRIOR option in SQLExtendedFetch, (it did not support SQLExtendedFetch at all) SQL_CUR_USE_IF_NEEDED caused the ODBC cursor library to be used. Since the SP3 driver supports the SQL_FETCH_PRIOR option in SQLExtendedFetch, SQL_CUR_USE_IF_NEEDED now causes the ODBC cursor library to be bypassed.

Table 4-3 SQL_ODBC_CURSORS Options

Using	SQL_CUR_USE_IF_NEEDED Option
Pre-SP3 driver	ODBC Cursor Library used
SP3 driver	Pervasive driver used

SQLSetConnectOption also supports the following options, just as they are supported in SQLSetStmtOption:

Table 4-4 Other Supported SQLSetConnectOption Options

SQL_CURSOR_TYPE	SQL_CONCURRENCY

Table 4-4 Other Supported SQLSetConnectOption Options

SQL_RETRIEVE_DATA	SQL_BIND_TYPE
SQL_ROWSET_SIZE	SQL_USE_BOOKMARKS

If these options are set at the connection level, then all statement handles allocated through the given connection automatically have these options set as specified in SQLSetConnectOption.

In terms of functionality, the pre-SP3 driver without use of the ODBC cursor library provides the least desirable scenario. The pre-SP3 driver and the SP3 driver provide identical functionality when the ODBC cursor library is used. The SP3 driver without use of the ODBC cursor library (bypassing the ODBC cursor library) provides the most functionality including all functionality provided by both the pre-SP3 driver and the ODBC cursor library with the exceptions involving temporary tables as described in "Temporary Tables" on page 4-12.

SQLGetFunctions

The pre-SP3 driver, when bypassing the ODBC cursor library, indicates that SQLExtendedFetch, SQLSetPos and SQLSetScrollOptions are not supported.

The ODBC cursor library (used with either the pre-SP3 or SP3 driver) and the SP3 driver (bypassing the ODBC cursor library) both indicate that SQLExtendedFetch and SQLSetPos are supported. Because the driver is ODBC 2.0 compliant, it supports SQLSetStmtOptions as opposed to SQLSetScrollOptions.

Table 4-5 SQLGetFunctions Results

Using	SQL Functions	SQLGetFunctions Reports Supported
Pre-SP3 driver	SQLExtendedFetch SQLSetPos	
ODBC cursor library	SQLExtendedFetch SQLSetPos	✔
SP3 driver	SQLExtendedFetch SQLSetPos	✔

SQLGetInfo

The options that have changes for Service Pack 3:

- SQL_BOOKMARK_PERSISTENCE
- SQL_FETCH_DIRECTION
- SQL_LOCK_TYPES
- SQL_POS_OPERATIONS
- SQL_SCROLL_CONCURRENCY
- SQL_SCROLL_OPTIONS
- SQL_STATIC_SENSITIVITY.

The following tables indicate the returned values for each of these options for the pre-SP3 driver, the ODBC cursor library and the SP3 driver.

Table 4-6 Bookmark Persistence

Using	SQL_BOOKMARK_ PERSISTENCE	SQL_FETCH_DIRECTION	SQL_LOCK_TYPES
Pre-SP3 driver	Not available	SQL_FD_FETCH_NEXT	Not available
ODBC cursor library	SQL_BP_DELETE | SQL_BP_UPDATE | SQL_BP_SCROLL	SQL_FD_FETCH_NEXT | SQL_FD_FETCH_FIRST | SQL_FD_FETCH_LAST | SQL_FD_FETCH_PRIOR | SQL_FD_FETCH_ABSOLUTE | SQL_FD_FETCH_RELATIVE | SQL_FD_FETCH_BOOKMARK	SQL_LCK_NO_CHANGE
SP3 driver	SQL_BP_UPDATE | SQL_BP_SCROLL	SQL_FD_FETCH_NEXT | SQL_FD_FETCH_FIRST | SQL_FD_FETCH_LAST | SQL_FD_FETCH_PRIOR | SQL_FD_FETCH_ABSOLUTE | SQL_FD_FETCH_RELATIVE | SQL_FD_FETCH_BOOKMARK	SQL_LCK_NO_CHANGE

Table 4-7 Scrolling

Using	SQL_POS_OPERATIONS	SQL_SCROLL_CONCURRENCY	SQL_SCROLL_OPTIONS
Pre-SP3 driver	Not available	SQL_SCCO_READ_ONLY	SQL_SO_FORWARD_ONLY | SQL_SO_STATIC
ODBC cursor library	SQL_POS_POSITION	SQL_SCCO_READ_ONLY | SQL_SCCO_OPT_VALUES	SQL_SO_FORWARD_ONLY | SQL_SO_STATIC
SP3 driver	SQL_POS_POSITION | SQL_POS_REFRESH | SQL_POS_UPDATE | SQL_POS_DELETE | SQL_POS_ADD	SQL_SCCO_READ_ONLY | SQL_SCCO_LOCK | SQL_SCCO_OPT_ROWVER	SQL_SO_FORWARD_ONLY | SQL_SO_STATIC | SQL_SO_DYNAMIC

Table 4-8 Static Sensitivity

Using	SQL_STATIC_SENSITIVITY
Pre-SP3 driver	Not available
ODBC cursor library	SQL_SS_UPDATES
SP3 driver	SQL_SS_ADDITIONS | SQL_SS_DELETIONS | SQL_SS_UPDATES

SQLSetStmntOption

The options of interest:

- SQL_BIND_TYPE
- SQL_CONCURRENCY
- SQL_CURSOR_TYPE
- SQL_RETRIEVE_DATA
- SQL_ROWSET_SIZE
- SQL_USE_BOOKMARKS

The following tables indicate valid set values for each option.

Table 4-9 Binding, Concurrency, and Cursor Types

Using	SQL_BIND_TYPE	SQL_CONCURRENCY	SQL_CURSOR_TYPE
Pre-SP3 driver	Returns SQL_ERROR and sets SQLSTATE to S1C00	Returns SQL_ERROR and sets SQLSTATE to S1C00	SQL_CURSOR_FORWARD_ONLY (the driver returns SQL_ERROR and sets SQLSTATE to S1C00 for all other values)
ODBC cursor library	SQL_BIND_BY_COLUMN or a length to indicate row-wise binding	SQL_CONCUR_READ_ONLY or SQL_CONCUR_ VALUES (for SQL_CONCUR_ROWVER the library substitutes SQL_CONCUR_VALUES, returns SQL_SUCCESS_WITH_INFO and sets SQLSTATE to 01S02) (for SQL_CONCUR_LOCK the library returns SQL_ERROR returned with SQLSTATE of S1C00)	SQL_CURSOR_FORWARD_ONLY or SQL_CURSOR_STATIC (for SQL_CURSOR_KEYSET_DRIVEN and SQL_CURSOR_DYNAMIC the library substitutes SQL_CURSOR_STATIC, returns SQL_SUCCESS_WITH_INFO and sets SQLSTATE to 01S02)
SP3 driver	SQL_BIND_BY_COLUMN or a length to indicate row-wise binding	SQL_CONCUR_READ_ONLY or SQL_CONCUR_LOCK or SQL_CONCUR_ROWVER (for SQL_CONCUR_VALUES the driver substitutes SQL_CONCUR_ROWVER, returns SQL_SUCCESS_WITH_INFO and sets SQLSTATE to 01S02)	SQL_CURSOR_FORWARD_ONLY or SQL_CURSOR_STATIC or SQL_CURSOR_DYNAMIC (for SQL_CURSOR_KEYSET_DRIVEN the driver substitutes SQL_CURSOR_STATIC, returns SQL_SUCCESS_WITH_INFO and sets SQLSTATE to 01S02)

Table 4-10 Rowset Size and Bookmarks

	SQL_RETRIEVE_DATA	SQL_ROWSET_SIZE	SQL_USE_BOOKMARKS
Pre-SP3 driver	Returns SQL_ERROR and sets SQLSTATE to S1C00	Returns SQL_ERROR and sets SQLSTATE to S1C00	Returns SQL_ERROR and sets SQLSTATE to S1C00
ODBC cursor library	SQL_RD_ON (for SQL_RD_OFF the library returns SQL_ERROR returned with SQLSTATE of S1C00)	Any value indicating number of rows in the rowset as long as it does not exceed maximum rowset size.	SQL_UB_ON or SQL_UB_OFF
SP3 driver	SQL_RD_ON or SQL_RD_OFF	Any value indicating number of rows in the rowset as long as it does not exceed maximum rowset size.	SQL_UB_ON or SQL_UB_OFF

New ODBC Functionality

This section describes ODBC functionality supported previously in Pervasive.SQL with the ODBC cursor library but that is now supported by the Pervasive driver.

SQLExtendedFetch

The pre-SP3 driver relied entirely on the ODBC cursor library for SQLExtendedFetch support.

The ODBC cursor library and the SP3 driver both support all fetch types except SQL_FETCH_RESUME. Furthermore, both the library and the driver support row-wise and column-wise binding and binding of column zero, the bookmark column.

SQLSetPos

The pre-SP3 driver relied entirely on the ODBC cursor library for SQLSetPos support.

The ODBC cursor library only supports the SQL_POSITION option and only the SQL_LOCK_NO_CHANGE lock type.

The SP3 driver supports all the options but only the SQL_LOCK_NO_CHANGE lock type. Furthermore, the SP3 driver supports data-at-execution columns for the SQL_ADD and SQL_UPDATE options.

Of course the SQL_ADD, SQL_DELETE and SQL_UPDATE options are only supported on updateable result sets. If the result set is not updateable, then SQLSetPos returns SQL_ERROR and sets SQLSTATE to 42000. A result set is not updateable if:

- it is defined by a query that contains more than one table in the FROM clause
- contains an aggregate function in the selection list (for example, COUNT, SUM and AVG)
- contains DISTINCT in the selection list
- or contains a UNION.

Even if the result set is updateable, particular columns in the result set may not be updateable. If an attempt is made to use result set columns that are not updateable to specify values for the SQL_ADD or SQL_UPDATE option of a SQLSetPos call, the driver returns SQL_ERROR and sets SQLSTATE to 22005.

A result set column is not updateable if any of the following is true:

- It is defined by any expression other than a single column name (e.g.: <column name> + 1 and RTRIM(<column name>) are legal selection-list items that are not updateable).

Temporary Tables

Special consideration needs to be taken for the case where a temporary table is required to process a query when the cursor type (SQL_CURSOR_TYPE) has been defined as dynamic (SQL_CURSOR_DYNAMIC). This consideration is necessary because a result set generated through the use of a temporary table is in fact a static cursor rather than a dynamic cursor. Therefore, a SQLPrepare, SQLExecute or SQLExecDirect that generates a result set requiring a temporary table with the cursor type defined as dynamic, returns SQL_SUCCESS_WITH_INFO and sets SQLSTATE to 01000.

New Limitations

The new implementation of temporary tables has introduced a limitation that only a single LONGVARCHAR or CLOB column may be referenced in a manner that would make it part of the index in the temporary table. Specifically, only one column of type LONGVARCHAR or CLOB may appear

- in a query's selection-list if DISTINCT is specified
- in the ORDER BY clause or in the GROUP BY clause
- in any selection-list in a UNION query.

This limitation can be largely overcome though through the use of the LEFT, RIGHT, and SUBSTRING scalar functions. Multiple columns of type LONGVARCHAR and CLOB may be referenced in a manner that would make them part of the index in the temporary table, such as those cases listed above, if all or all but one of the columns are referenced inside of a LEFT, RIGHT or SUBSTRING scalar function.

For performance reasons, a column of type LONGVARCHAR or CLOB should always be specified within a LEFT, RIGHT, or SUBSTRING scalar function, preferably resulting in a value less than 251 bytes in length (for example, LEFT(co1, 250)).

Similar to the notes above, only a single LONGVARBINARY or BLOB may be referenced in a manner that would make it part of the index in the temporary table. Specifically, only one column of type LONGVARBINARY or BLOB may appear in a query's selection-list if DISTINCT is specified or in any selection-list in a UNION query. Columns of type LONGVARBINARY and BLOB are not allowed in

an ORDER BY or GROUP BY clause in either the pre-SP3 or SP3 drivers.

Performance

The SP3 driver executes some queries requiring temporary tables more slowly than the pre-SP3 driver while executing some queries much more quickly. In general, expect queries requiring temporary tables that contain 256KB or less of data to execute more slowly with the SP3 driver, but expect queries requiring temporary tables that contain greater than 256KB of data to execute more quickly.

Positioned UPDATE and DELETE

Positioned updates and deletes as supported by the ODBC cursor library are supported in the SP3 driver. Positioned updates and deletes are the UPDATE and DELETE statements that contain the 'WHERE CURRENT OF' phrase as defined in the Pervasive.SQL 2000i *Pervasive.SQL Programmer's Reference* manual.

In previous versions of this manual, these statements are supported by the Pervasive.SQL 2000 ODBC driver only within a stored procedure or trigger and rely on the ODBC cursor library to support these statements at the session level. The SP3 driver supports these statements within stored procedures and triggers as well as at the session level.

New Limitations

The limitations involving whether or not a result set is updateable and whether or not a particular column within an updateable result set is updateable that apply to the SQL_ADD, SQL_DELETE and SQL_UPDATE options of the SQLSetPos API also apply to the positioned update statement.

Performance

The ODBC cursor library transforms a positioned update or delete into a regular searched update or delete statement. The SP3 driver takes advantage of SQLSetPos implementation thus not requiring a search through the result set in order to find the 'current' row and in general it executes much more quickly than the ODBC cursor library.

chapter

Improved OLE DB Provider in Pervasive.SQL 2000i

5

An Overview of the Updated OLE DB provider in Pervasive.SQL 2000i

The following are the overview sections found in this chapter:

- "Overview of New Features in ADO and OLE DB Provider" on page 5-2
- "Programming Notes for Pervasive OLE DB Provider" on page 5-4
- "Performance Considerations with OLE DB" on page 5-7
- "COM+ Services Support" on page 5-8
- "Execute Method (ADO Command)" on page 5-11
- "Limitations of the OLE DB Provider" on page 5-13

Overview of New Features in ADO and OLE DB Provider

This section provides an overview of the changes made to the Pervasive OLE DB driver in the Pervasive.SQL 2000i SP3 release.

Command-Based Recordsets Supported

The OLE DB provider that shipped with Pervasive.SQL 2000 prior to SP3 did not include support for SQL statements. This meant that Command objects were not supported and that result sets required table names to open successfully. This version includes support for SQL commands and conforms to 2.5 specifications. Unlike the SQL Server provider or the ODBC bridge, the provider can open result sets that are either command based or purely navigational. Additionally, server-side cursors on either can be forward-only, static, or dynamic. Both command-based result sets and navigational (table-based) result sets can be open and operated on at the same time.

Command-based recordsets provide the power and flexibility of the SQL engine, but server-side navigational result sets provide direct access to indexes; this feature is not available with command-based result sets (indirect access is provided by the query optimizer). With the index capability comes the ability to perform Seek operations. A routine that uses Seek can significantly outperform a similar routine that performs the same functions via a SQL statement. When used properly, server-side navigational recordsets can improve application performance by allowing rapid positioning on records that contain specific values.

ADOX

We now support ADO Extensions for Data Definition Language and Security (ADOX). ADOX is used to create tables, modify schema definitions, and view the contents of database tables. Currently, catalog, table, column, and index objects are supported. The creation of tables and indexes is supported; database creation is not supported at this time.

Navigational Recordsets in the New Provider

In order to open a navigational result set, **adCmdTableDirect** must be used in the options of the Open method. In the previous version, **adCmdTable** would successfully open a navigational result set. However, ADO will turn this into a "`SELECT * FROM`" SQL statement. This will make the result set command-based, and indexes will be unavailable.

Note Since ADO treated this differently when command support was unavailable, existing applications that take advantage of index capabilities will no longer function unless the Open method calls use **adCmdTableDirect**.

Large Binary Objects

ISequentialStream support has been added to the OLE DB provider. In ADO, this translates into AppendChunk/GetChunk functionality of the recordset object. It also allows complex data binding to transfer BLOB data to and from visual controls.

Programming Notes for Pervasive OLE DB Provider

The following section lists notes about the Pervasive OLEDB driver:

Seek with Static Cursors

To use seek on a static cursor, the index must be set before opening the result set. For example:

```
Dim rs AsNew ADODB.Recordset
    rs.Index = "segment"
    rs.Open "Simple", "Provider=PervasiveOLEDB;Data
   Source=MyData", adOpenStatic, adLockOptimistic,
   adCmdTableDirect
    rs.Seek Array(2, 9)
    rs.Close
```

Remote Connections

Pervasive's OLE DB provider cannot create a remote connection. This means you cannot set a remote server within the connection string. However, there are alternatives for our provider to perform the same functionality.

- Use RDS, which has been tested with the Pervasive provider
- Develop a business object using COM+ services, and create the object using CreateObject. For example:

```
Dim m_busObj As projDLL.busObj
rs As New ADODB.Recordset
Set m_busObj =
   CreateObject("sampProj2.TwoPhaseSampleProduct",
   "RemoteServer")
Set rs = m_busObj.GetData()
```

Table Definitions

ITableDefinition does not support creation of the following columns because they are not supported by the underlying Pervasive.SQL engine.

- BSTR
- WCHAR
- VarWChar
- LongVarWChar
- UserDefined Types
- GUID

Default LockType

If no *LockType* is specified, the cursor is set to immediate update mode. This has a couple of repercussions:

- All changes are immediately transferred to the database (without the need to call update)
- **Update** and **UpdateBatch** really have no meaning (as the database is already updated). However, the **Supports** method will still return true of update, but will return false for **UpdateBatch**.
- **GetOriginalValue** is not usable.

To programmatically determine when **GetOriginalValue** is available, you must use the "supports" method with **adUpdateBatch** as the argument. For example:

```
if rs.Supports(adUpdateBatch) then
   someValue = rs.fields(iCol).OriginalValue
end if
```

Initialization Properties

The following table lists the properties Pervasive supports for initialization in OLE DB, and the corresponding connection string identifiers

Table 5-1 Connection String Identifiers

Connection String Identifier	Property
Cache Authentication	DBPROP_AUTH_CACHE_AUTHINFO
Connect Timeout	DBPROP_INIT_TIMEOUT
Data Source	DBPROP_INIT_DATASOURCE
Encrypt Password	DBPROP_AUTH_ENCRYPT_PASSWORD
Locale Identifier	DBPROP_INIT_LCID
Location	DBPROP_INIT_LOCATION
Password	DBPROP_AUTH_PASSWORD
Persist Encrypted	DBPROP_AUTH_PERSIST_ENCRYPTED
Persist Security Info	DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO
User ID	DBPROP_AUTH_USERID

You can also set the following properties:

- DBPROP_INIT_HWND,
- DBPROP_INIT_PROMPT,
- DBPROP_INIT_ASYNCH,
- DBPROP_INIT_OLEDBSERVICES

Performance Considerations with OLE DB

The performance of the provider has been significantly improved from the previous release.

Best Performance is Navigational

As mentioned previously, server-side navigational recordsets will have a significant performance advantage over command-based recordsets for tasks that require frequent positioning on records that contain specific values.

Static vs. Dynamic Cursors

Static cursors will create a temporary table behind the scenes whenever one would not have been created by the relational engine (as described previously in "Temporary Tables" on page 4-12). This will be the case for both command-based and navigational tables. When bandwidth is not a significant consideration, dynamic cursors can provide higher performance, since they do not always involve temporary tables. However, in low bandwidth scenarios, round-trips can be too expensive to justify dynamic cursors; in this case RDS is often a good solution. The drawback to RDS is that Microsoft has implemented it as a command-based only solution, which means that index functionality (using Seek) is unavailable. Performance can be maintained regardless of deployment by implementing an abstraction layer that works identically on RDS-based and local recordsets. The nature of this abstraction would depend on the needs of the application and would likely take the form of a runtime business object.

Disable Unused Services

When developing OLEDB applications, a way to improve performance is to turn off any OLE DB Services that are not being used. See the documentation for DBPROP_INIT_OLEDBSERVICES for more information.

Turning off the Automatic Transaction Enlistment will not instantiate the **ITransactionJoin** interface on the session and will also keep the provider from looking for MTS objects.

COM+ Services Support

This section describes how Pervasive's OLE DB provider interacts with Microsoft's COM+ Services.

What is COM+ Services?

COM+ services (Component Object Model) is a Microsoft specific technology used to create business objects in a multi-threaded context. COM+ is primarily designed for, but not limited to, Visual Basic programmers. COM+ allows for rapid creation of multi-tier applications and includes many benefits that would normally have to be implemented by the developer. The benefits of COM+ include

- contexts
- concurrency
- added security
- object pooling
- just-in-time activation
- queued components
- events

For further information about COM+, see Microsoft's documentation.

COM+ services are an extension to the benefits provided by MTS (Microsoft Transaction Server). MTS is Microsoft's previous business object server implementation. In general, references (in Microsoft and Pervasive documentation) to MTS can be substituted with COM+ services.

Pervasive's OLE DB provider is supported in COM+ services. ADO calls made within COM+ services behave like any ADO client call.

Example of COM+ Services for Visual Basic Programmers

As a Visual Basic programmer, you must be aware of the **MTSTransactionMode** property. Setting this to anything other than *NoTransactions* will invoke Microsoft transactions. Please refer to COM+ services documentation for the complete reference of this feature.

The following example demonstrates use of Microsoft Transactions. In order for the calls to **GetObjectContext** to succeed, you must set the **MTSTransactionMode** property to something other than

NoTransactions. Using Microsoft Transactions will allow Microsoft's Transaction Coordinator to do a two-phase commit.

```
Public Function UpdatePayroll(employeeID As
Integer, salary As Currency)
    On Error GoTo ErrHandler
    Dim rs As New ADODB.Recordset
    rs.Index = "employeeID"
    rs.Open "PayrollTable",
"Provider=PervasiveOLEDB;Data Source=CompanyDB",
adOpenDynamic, adLockOptimistic, adCmdTableDirect
    rs.Seek employeeID, adSeekFirstEQ

    If rs.EOF = True Then
        GetObjectContext.SetAbort
    Else
        rs!salary = salary
    End If
    rs.Update
    rs.Close
    rs = Nothing
    GetObjectContext.SetComplete
Exit Function
ErrHandler:
    GetObjectContext.SetAbort
    If Not IsNull(rs) Then
        If rs.State = adStateOpen Then
            rs.Close
        End If
    End If
    rs = Nothing
End Function
```

However, you could rewrite this business object using ADO transactions (with a connection object). This would allow you to set the **MTSTransactionMode** property to *NoTransactions*. Without Microsoft Transactions, you no longer have the overhead of the two-phase commit. Also, objects that do not support transactions are allowed to stay resident in memory, whereas those that do are constructed and destroyed on each reference.

```
Public Function UpdatePayroll(employeeID As
Integer, salary As Currency)
    On Error GoTo ErrHandler
    Dim cn As New Connection
    Dim rs As New ADODB.Recordset
```

```
    cn.Open "Provider=PervasiveOLEDB;Data
Source=CompanyDB"
    cn.BeginTrans
    rs.Index = "employeeID"
    rs.Open "PayrollTable", cn, adOpenDynamic,
adLockOptimistic, adCmdTableDirect
    rs.Seek employeeID, adSeekFirstEQ

    If rs.EOF = True Then
        cn.RollbackTrans
    Else
        rs!salary = salary
    End If
    rs.Update
    rs.Close
    cn.CommitTrans
Exit Function
ErrHandler:
    cn.RollbackTrans
    If Not IsNull(rs) Then
        If rs.State = adStateOpen Then
            rs.Close
        End If
    End If
End Function
```

Execute Method (ADO Command)

When using the Pervasive OLE DB Provider, the *RecordsAffected* parameter of the Execute Method on a command will return different results based on the type of operation.

SELECT operations

When performing a SELECT statement, *RecordsAffected* will return a –1 (negative one), which means that this option is not supported. For example:

```
cn.Open "Provider=PervasiveOLEDB;Data Source=TestData;"
SQLst = "Select * From MyData"
cmd.ActiveConnection = cn
cmd.CommandText = SQLst
Set rs = cmd.Execute(RecordsAffected)
```

In this scenario *RecordsAffected* equals –1.

If you want to obtain the number of records returned by a SELECT query, use the **RecordCount** property, as shown in the following example:

```
recordcount = rs.RecordCount
' number of records on MyData
```

Batch Insert, Update, or Delete

RecordsAffected will return the correct number of records that the operation affected when performing a batch insert, update or delete.

Example - Batch Insert

```
cn.Open "Provider=PervasiveOLEDB;Data Source=TestData;"
SQLst = "Insert into MyData(utinyint_,
   usmallint_,uinteger_, ubigint_, char_, character_,
   bit_) Values (1, 12, 13, 100, 'testdata', 'chardata',
   1)"
cmd.ActiveConnection = cn
cmd.CommandText = SQLst
cmd.Execute RecordsAffected
```

In this scenario *RecordsAffected* equals 1.

Example - Batch Update

```
SQLst = "Update MyData set char_ = 'SampleTest'  where
uinteger_ = 13"
```

```
cmd.ActiveConnection = cn
cmd.CommandText = SQLst
cmd.Execute RecordsAffected
```

In this scenario *RecordsAffected* equals *x,* which is all the records that have the value 13.

Limitations of the OLE DB Provider

- The OLE DB Provider is a thick client. It communicates with the Pervasive.SQL engine using the navigational interface only; this means that some SQL queries will be disproportionately affected by client-server operation. Navigational recordsets should be relatively unaffected by client-server deployment. Remote access to the provider can be accomplished by using either RDS or DCOM. See the Microsoft documentation for in-depth understanding of these two Microsoft technologies.
- Asynchronous operations are not supported.
- The Record and Stream objects are not supported.
- The Index property cannot be set on a static navigational cursor once the recordset has been used. Set the index appropriately before performing the open operation on the recordset. After opening the recordset, the index cannot be changed.

chapter

6

JDBC 2 Enhancements in Pervasive.SQL 2000i

An Overview of the New JDBC 2 Functionality in Pervasive.SQL 2000i

JDBC 2 API is now supported in Pervasive.SQL 2000i.

Table 6-1 Summary of New Functionality with Pervasive JDBC driver

Feature Element	Notes
Connection Strings	Extra parameter added that allows you to specify which code page to use with the connection.
DataSource Interface Support	Register Pervasive.SQL databases in JNDI and your applications can be shielded from Pervasive-specific driver features.

Note Information relevant to developing Pervasive.SQL applications discussed in this chapter supersedes that of the Pervasive.SQL 2000 SDK documentation found in Service Pack 2. The SDK documentation will be updated after the release of Pervasive.SQL 2000i Service Pack 3.

The following are the sections found in this chapter:

- "Overview of Pervasive JDBC 2 Driver"
- "JDBC Connection String Enhancements" on page 6-4
- "JDBC 2.0 Standard Extension API" on page 6-6
- "Connection and Concurrency Notes" on page 6-10
- "Scrollable Result Set Notes" on page 6-11
- "JDBC Programming Sample" on page 6-12

Overview of Pervasive JDBC 2 Driver

Specifications

- 100% Java certified
- JDBC 2 compliant driver
- Type 4 JDBC driver

JDBC API Improvements

- Scrollable – ability to perform both relative and absolute positioning.
- Updateable – ability to insert, update, and delete without the need to execute more SQL.
- Dynamic & Static server-side cursors – ability to choose whether you want to see changes made by you or other users.
- Batch Updates – ability to queue up many operations and have them execute at once.
- Character Encoding support - allows you to change encoding per connection.

JDBC Optional Package Support

- DataSource interface - allows data source objects to be registered in JNDI.
- Connection Pooling support – complete implementation of the connection pooling interfaces.

Backward Compatibility

Pervasive JDBC version 2 is backward compatible. Applications compiled with previous releases of Pervasive JDBC drivers will work with the new driver without the need to recompile. Applets will need to change the jar file name from pervasiveJDBC.jar to pvjdbc2.jar in the HTML file.

The new driver is accessed through a different package name:

- com.pervasive.jdbc.v2.Driver will load a JDBC 2 driver
- pervasive.jdbc.PervasiveDriver will load the JDBC 1 driver

Class Names

With the Pervasive JDBC version 2 driver, class names have changed to comply with Sun recommended standard. All Pervasive classes now are prefaced by com.pervasive.jdbc.v2. In older versions, classes were prefaced by pervasive.jdbc.

Unsupported APIs

Pervasive's JDBC driver does not support the following JDBC interfaces:

- Array
- Blob
- Clob
- Ref
- Struct
- SQLData
- SQLInput
- SQLOutput

These are not supported due to the fact the Pervasive.SQL engine does not currently support the underlying SQL 3 data types.

Driver Limitations

- You cannot use long data in "out" parameters
- The smallest actual fetch size is two rows.
- You cannot have an updateable result set with a join.
- You cannot have an updateable result set with a "group by".

JDBC Connection String Enhancements

The Pervasive JDBC driver now supports character encoding, which allows you to filter data you read through a specified code page so that it is formatted and sorted correctly.

How to Connect

The following shows how to connect to a Pervasive.SQL database using JDBC:

Driver Classpath

```
com.pervasive.jdbc.v2
```

Instantiate Driver

```
Class.forName("com.pervasive.jdbc.v2.Driver");
```

URL

```
jdbc:pervasive://server:port/DSN;encoding
```

argument	description
server	Enter the server name using an ID or a URL.
port	Pervasive's relational engine default port is 1583. If no port is specified, the default is used.
DSN	Set up a DSN on the server using regular ODBC methods.
encoding	See "Using Character Encoding"

Using Character Encoding

Character encoding is specified using a parameter in the connection string passed to the driver manager.

Encoding parameter allows user to specify the code page to be used for translating the string data stored in the database. The default system encoding is used if no encoding is explicitly specified.

Example of Using Character Encoding

```
public static void main(String[] args)
{
    //specify latin 2 encoding
```

```
   String url = "jdbc:pervasive://MYSERVR:1583/
SWEDISH_DB;encoding=cp850";
   try{

Class.forName("com.pervasive.jdbc.v2.Driver");
       Connection conn =
DriverManager.getConnection(url);
       Statement stmt = conn.createStatement();
       ResultSet rs = stmt.executeQuery("select *
from SwedishTable");
       rs.close();
       stmt.close();
       conn.close();
   }
   catch(Exception e)
   {
       e.printStackTrace();
   }
}
```

Notes on Character Encoding

- If one database contains tables that were populated using two different encodings, two distinct connections need to be established.
- Pervasive JDBC driver uses Java native support for code pages. The list of supported code pages can be obtained from http://java.sun.com/products/jdk/1.2/docs/guide/internat/encoding.doc.html

JDBC 2.0 Standard Extension API

Because connection strings are vendor-specific, Sun specified a DataSource interface. It takes advantage of JNDI, which functions as a Java registry. The DataSource interface allows JDBC developers to create named databases. As a developer, you register the database in JNDI along with the vendor-specific driver information. Then, your JDBC applications can be completely database agnostic and be "pure JDBC".

Pervasive JDBC driver now supports JDBC 2.0 Standard Extension API. Currently Pervasive JDBC supports the following interfaces

- javax.sql.ConnectionEvent
- javax.sql.ConnectionEventListener
- javax.sql.ConnectionPoolDataSource
- javax.sql.DataSource
- javax.sql.PooledConnection

Note These interfaces are packaged separately in pvjdbc2x.jar in order to keep the core JDBC API 100% pure java.

Although at this time Pervasive does not provide implementation of RowSet interfaces, Pervasive JDBC driver has been tested with Sun's implementation of RowSet interface.

DataSource

Sun has provided a way for application developers to write applications that are driver independent. By using DataSource interface and JNDI applications can access data using standard methods completely illuminating things like connection strings and other driver specifics aspects. In order to use DataSource interface, a database has to be registered with a JNDI service provider. An application can then access it by name.

The following is an example of using the DataSource interface:

```
// this code will have to be executed by the
// administrator in order to register the
// DataSource.
// This sample uses Sun's reference JNDI
// implementation
```

```
public void registerDataSources()
    {
// this example uses the JNDI file system
// object as its registry

    Context ctx;
    jndiDir = "c:\\jndi";

        try
        {
          Hashtable env = new Hashtable (5);
         env.put (Context.INITIAL_CONTEXT_FACTORY,

"com.sun.jndi.fscontext.RefFSContextFactory");

      env.put(Context.PROVIDER_URL, jndiDir);
          ctx = new InitialContext(env);
        }
        catch (Exception e)
        {
         System.out.println(e.toString());
        }

        //register demodata as regular data source
        com.pervasive.jdbc.v2.DataSource ds = new
com.pervasive.jdbc.v2.DataSource();
        String dsName = "";

        try
        {
           // Set the user name, password, driver
type and network protocol
            ds.setUser("administrator");
            ds.setPassword("admin");
            ds.setPortNumber("1583");
            ds.setDatabaseName("DEMODATA");
            ds.setServerName("127.0.0.1");

ds.setDataSourceName("DEMODATA_DATA_SOURCE");
       ds.setEncoding("cp850");
            dsName = "jdbc/demodata";

       // Bind it
        try
        {
            ctx.bind(dsName,ds);
             System.out.println("Bound data source
[" + dsName + "]");
        }
        catch (NameAlreadyBoundException ne)
```

```
        {
            System.out.println("Data source [" +
dsName + "] already bound");
        }
        catch (Throwable e)
         {
          System.out.println("Error in JNDI
binding occurred:");
          throw new Exception(e.toString());
        }
     }
    }
}

//in order to use this DataSource in application the
following code needs to be executed

public DataSource lookupDataSource(String ln)
throws SQLException
    {
      Object ods = null;
   Context ctx;

        try
        {
          Hashtable env = new Hashtable (5);
         env.put (Context.INITIAL_CONTEXT_FACTORY,

"com.sun.jndi.fscontext.RefFSContextFactory");

         //this will create the jndi directory and
return its name
          //if the directory does not already exist
          String jndiDir = "c:\\jndi";

      env.put(Context.PROVIDER_URL, jndiDir);
          ctx = new InitialContext(env);
        }
        catch (Exception e)
        {
         System.out.println(e.toString());
        }
        try
        {
            ods = ctx.lookup(ln);
            if (ods != null)
             System.out.println("Found data source
[" + ln + "]");
            else
```

```
                System.out.println("Could not find
data source [" + ln + "]");
        }
        catch (Exception e)
        {
            throw new SQLException(e.toString());
        }

        return (DataSource)ods;
    }

//note that ConnectionPoolDataSource is handled
similarly.
```

Connection and Concurrency Notes

A single Pervasive JDBC connection can easily serve multiple threads. However, while the Connection may be thread-safe, the objects created by the Connection are not. For example, a user can create four threads. Each of these threads could be given their own **Statement** object (all created by the same **Connection** object). All four threads could be sending or requesting data over the same connection at the same time. This works because all four **Statement** objects have a reference to the same **Connection** object and their reading and writing is synchronized on this object. However, thread #1 cannot access the Statement object in thread #2 without this access being synchronized. The above is true for all other objects in the JDBC API.

Scrollable Result Set Notes

Scrollable result sets allow you to move forward and backward through a result set. This type of movement is classified as either relative or absolute. You can position absolutely on any scrollable result set by calling the methods `first()`, `last()`, `beforeFirst()`, `afterLast()`, and `absolute()`. Relative positioning is done with the methods `next()`, `previous()`, and `relative()`.

A scrollable result set can also either be updateable or read-only. This refers to whether or not you are able to make changes to the underlying database. Another term, sensitivity, refers to whether these changes are reflected in your current result set.

A sensitive result set will reflect any insert, updates, or deletes made to it. In Pervasive.SQL's case, an insensitive result set does not reflect any changes made to it (it is a static snapshot of the data). In other words, you do not see your updates or those made by anyone else.

Sensitive and insensitive result sets correspond to dynamic and static in ODBC, respectively. A sensitive result set reflects your own changes and can reflect others changes if the transaction isolation level is set to `READ_COMMITTED`. Transaction isolation is set using the **Connection** object. The result set type is set upon statement creation.

If your result set is insensitive, then it is possible to make calls to the method `getRow()` in order to determine your current row number. On an insensitive result set, you can also make calls to `isLast()`, `isFirst()`, `isBeforeFirst()`, and `isAfterLast()`. On a sensitive result set, you can only make calls to `isBeforeFirst()` and `isAfterLast()`. Also, on an insensitive result set, the driver will honor the fetch direction suggested by the user. The driver ignores the suggested fetch direction on sensitive result sets.

JDBC Programming Sample

The following example creates a connection to the database named "DB" on server "MYSERVER". It then creates a statement object on that connection that is sensitive and updateable. Using the statement object a "SELECT" query is performed. Once the result set object is obtained a call to "absolute" is made in order to move to the fifth row. Once on the fifth row the second column is updated with an integer value of 101. Then a call to "updateRow" is made to actually make the update.

```
Class.forName("com.pervasive.jdbc.v2.Driver");
Connection conn=
DriverManager.getConnection("jdbc:pervasive://
MYSERVER:1583/DB");

Statement stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_SENSITI
VE, ResultSet.CONCUR_UPDATABLE);

ResultSet rs =
m_stmt.executeQuery("SELECT * FROM mytable");


rs.absolute(5);
rs.updateInt(2, 101);
rs.updateRow();

rs.close();
stmt.close();
conn.close();
```

Index

Symbols

@@IDENTITY 3-2
@@ROWCOUNT 3-4
~PVSW~.LOG file 1-25

A

ABS() 3-22
Absolute value
    how to find 3-22
ACOS() 3-22
Active Clients (obsolete) 1-20
Adding
    timestamp values 3-27
ADO 5-1
Advanced Operations Guide 1-34
ALTER COLUMN 3-17
ALTER TABLE 1-32, 3-17
    Pervasive.SQL 7 compared 3-18
Analyzer results 2-15
Analyzing your system 2-1
Arccosine value
    how to find 3-22
Archiving previous components 2-13
Arcsine value
    how to find 3-23
Arctangent of x,y coords
    how to find 3-23
Arctangent value
    how to find 3-23
ASCII character for ASCII value
    how to find 3-21
ASCII value of character
    how to find 3-20
ASCII() 3-20
ASIN() 3-23
ATAN() 3-23
ATAN2() 3-23
AutoReconnect Timeout 1-30
Auto-reconnect, see Pervasive auto-reconnect

B

Back to Minimal State if Inactive 1-20
Base-10 log
    how to find 3-24
BIT_LENGTH() 3-21
Bits in string, number of
    how to find 3-21
BLOB
    and temporary tables 4-12
    prohibited in ORDER BY or GROUP BY 4-12
    UNION and 4-12
Bound databases
    IN DICTIONARY not permitted 3-10
BSTART, loading NSS volumes first 1-15
Btrieve, testing 2-1, 2-12
Bytes
    Configuration settings measured in 1-19
Bytes in string, number of
    how to find 3-21

C

Cache manager 1-20
CEILING() 3-23
Changing
    column datatype
        Changing
            column NULL property 3-17
Changing a table
    specifying file name 3-8
CHAR() 3-21
CHAR_LENGTH() 3-21
Character encoding 6-4
Character translation
    OEM to ANSI 1-33
Characters in string, number of
    how to find 3-21
Checking
    client and server versions at load time 1-12
    for NULL value 3-28
Citrix MetaFrame 1-15
Client version

server and, checking compatibility 1-12
CLOB
and temporary tables 4-12
LEFT and 4-12
RIGHT and 4-12
SUBSTRING and 4-12
UNION and 4-12
Columns
maximum number in SELECT 1-13
maximum number in table 1-13
COM+ services 5-8
Comma
as decimal separator 1-16
Communications
auto-reconnect feature 1-29
Communications Buffer Size 1-21
Communications Threads 1-21
Components
detecting 2-13
restoring previous 2-13
Concurrency in JDBC 6-10
Configuration settings
all size settings now in bytes 1-19
Communications Buffer Size 1-21
Communications Threads 1-21
dynamic 1-19
Log Buffer Size 1-20
MKDE Communications Buffer Size 1-21
newly obsolete 1-19
Number of Sessions 1-22
obsolete 1-20
Operation Bundle Limit 1-22
Wait Lock Timeout 1-22
Connection pools 6-6
Connection strings
in JDBC 6-4
TRANSLATIONDLL 1-33
Converting
ASCII character to ASCII code 3-20
ASCII code to ASCII character 3-21
data types of columns 3-17
date expr to month name 3-27
date expr to quarter (ordinal) 3-27
date expr to week of year 3-28
date to any date sub-field 3-26
date to day of year (ordinal) 3-26
date to name of day 3-26
degrees to radians 3-24
legacy data types to newer types 3-18
radians to degrees 3-23
Correlated subquery 3-14
COS() 3-23
Cosine value
how to find 3-23
COT() 3-23
Cotangent value
how to find 3-23
CREATE TABLE USING 3-6
Creating
redirecting Gateway Locator File 1-26
tables with legacy null support 1-32
tables,
specifying file name 3-6
CURDATE 3-25
CURRENT_DATE() 3-25
CURRENT_TIME() 3-26
Cursors
dynamic 2-1, 4-1, 4-2
CURTIME 3-26

D

Data files
binary compatible cross-platform 3-7
replacing existing 3-11
Data Manipulation
statements 4-12
Data type
changing for a column 3-17
Data types
converting legacy 3-18
DataSource interface 6-6
Date
converting to month of year (name) 3-27
converting to quarter of year (ordinal) 3-27
converting to week of year (ordinal) 3-28
Date functions
see Functions, date and time
Date value, how to extract sub-fields 3-26
Date, today's, how to find 3-25
Day
of year (ordinal), determining from date 3-26
DAYNAME() 3-26

DAYOFYEAR() 3-26
DDFs
changing data definitions without affecting data files 3-10
Decimal separator
comma as 1-16
Degrees
converting from radians 3-23
converting to radians 3-24
DEGREES() 3-23
Detecting previous components 2-13
DISTINCT
BLOB and 4-12
CLOB and 4-12
in subquery 3-15
LONGVARBINARY and 4-12
LONGVARCHAR and 4-12
Documentation
new and enhanced 1-34
DTI, and Pervasive auto-reconnect 1-29
DTO, and Pervasive auto-reconnect 1-29
Duplicating a string 3-21
Dynamic configuration 1-19
Dynamic cursors 2-1, 4-1, 4-2

E

Enable Auto Reconnect
client 1-30
server 1-29
Encoding, character 6-4
EXP() 3-23
Exponential value
how to find 3-23
Extended Operation Buffer Size (obsolete) 1-20
EXTRACT() 3-26

F

File name
changing for a table 3-8
specifying for table 3-6
FLOOR() 3-24
FOREIGN KEY
cannot ALTER column 3-17
Functions
date and time 3-25
CURRENT_DATE() 3-25
CURRENT_TIME() 3-26
DAYNAME() 3-26
DAYOFYEAR() 3-26
EXTRACT() 3-26
MONTHNAME() 3-27
QUARTER() 3-27
TIMESTAMPADD() 3-27
TIMESTAMPDIFF() 3-28
WEEK() 3-28
logical 3-28
IFNULL() 3-28
NULLIF() 3-28
numeric 3-22
ABS() 3-22
ACOS() 3-22
ASIN() 3-23
ATAN() 3-23
ATAN2() 3-23
CEILING() 3-23
COS() 3-23
COT() 3-23
DEGREES() 3-23
EXP() 3-23
FLOOR() 3-24
LOG() 3-24
LOG10() 3-24
PI() 3-24
POWER() 3-24
RADIANS() 3-24
RAND() 3-24
ROUND() 3-24
SIGN() 3-25
SIN() 3-25
SQRT() 3-25
TAN() 3-25
TRUNCATE() 3-25
scalar 3-20
optimized predicates with 1-13
string 3-20
ASCII() 3-20
BIT_LENGTH() 3-21
CHAR() 3-21
CHAR_LENGTH() 3-21
OCTET_LENGTH() 3-21
POSITION() 3-21
REPLACE() 3-22

REPLICATE() 3-21
SPACE() 3-22
STUFF() 3-22
utility 3-29
USER() 3-29
Functions, scalar 1-33

G

Gateway engine, see Workgroup engine
Gateway Locator File 1-23
redirecting 1-24
creating a 1-26
Getting Started Guide 1-34
Global variables, see Variables, global
Greater than zero, determining if 3-25
GROUP BY
BLOB columns prohibited 4-12
CLOB and 4-12
LONGVARBINARY columns prohibited 4-12
LONGVARCHAR and 4-12

H

HTML Help 1-34

I

IFNULL() 3-28
IN DICTIONARY 3-10
not permitted on bound databases 3-10
Installation Toolkit 1-3
Installation, tests performed during 2-3
Installing
on a terminal server 1-15
InstallScout
Installation Toolkit and 1-3
OEM Toolkit and 1-3
replaced by Pervasive System Analyzer 1-19, 2-1
Integer
largest less than or equal to
how to find 3-24
smallest greater than or equal to
how to find 3-23
Internal version number vii
Intervals
adding to timestamp values 3-27
subtracting from timestamp values 3-28
Invalid row-count in subquery
returned if SELECT within UPDATE returns
multiple rows 3-15

J

javax extensions 6-6
JDBC 2 API 6-1
JDBC standard extension API 6-6

K

Kilobytes
Configuration settings changed to bytes 1-19

L

Largest Compressed Record Buffer Size (obsolete) 1-20
Legacy data types, converting 3-18
Length
of path name in USING 3-7
Length of string in bits
how to find 3-21
Length of string in bytes
how to find 3-21
Length of string in characters
how to find 3-21
Less than zero, determining if 3-25
Loading
NSS volumes before BSTART/MGRSTART 1-15
Locale-specific behavior
comma as decimal separator 1-16
Locator Files, see Gateway Locator Files
Locking
row level 1-12
Locks, see Locking
Log Buffer Size 1-20
Log file, viewing 2-15
LOG() 3-24
LOG10() 3-24
Logarithm
base-10, how to find 3-24
natural, how to find 3-24
Logical File Handles (obsolete) 1-20
Logical functions
see Functions, logical
Login name

how to determine current 3-29
LONGVARBINARY
and temporary tables 4-12
prohibited in ORDER BY or GROUP BY 4-12
UNION and 4-12
LONGVARCHAR
and temporary tables 4-12
LEFT and 4-12
RIGHT and 4-12
SUBSTRING and 4-12
UNION and 4-12
LVAR column
converting to newer type 3-18

M

Management
memory 1-20
Maximum
number of columns allowed in SELECT 1-13
number of columns allowed in table 1-13
Maximum Databases (obsolete) 1-20
Maximum Open Files (obsolete) 1-20
Memory
now measured in bytes in Configuration 1-19
Memory management 1-20
MGRSTART, loading NSS volumes first 1-15
Microsoft Terminal Server 1-15
Microsoft Transaction Server 5-8
MKDE Communications Buffer Size 1-21
MODIFY COLUMN 3-17
Month
name, determining from date expr 3-27
MONTHNAME() 3-27
Moving a table
specifying file name 3-8
MTS 5-8

N

Name
of this product vii
Name of day, determining 3-26
Named Database
and file names 3-6
Natural log
how to find 3-24
NetWare NSS volumes 1-15
slower on updates 1-16
Network
auto-reconnect feature 1-29
Network connectivity, testing 2-1
Network testing 2-11
New utilities 2-1
Non-correlated subquery 3-14
NOTE column
converting to newer type 3-18
NSS volume support 1-15
NULL
cannot make column nullable 3-17
inserted by UPDATE if subquery returns no rows 3-14
modifying column property 3-17
Null support
creating tables with legacy 1-32
NULL value
how to check for 3-28
how to return if 2 expr are equal 3-28
NULLIF() 3-28
Number of bits in string
how to find 3-21
Number of bytes in string
how to find 3-21
Number of characters in string
how to find 3-21
Number of Sessions 1-22
Numeric functions, see Functions, numeric

O

Obsolete configuration settings 1-20
OCTET_LENGTH() 3-21
OEM to ANSI
character translation 1-33
connection string 1-33
OEM Toolkit 1-3
OLE DB provider 5-1
Online help 1-34
Operation Bundle Limit 1-22
Optimization
predicates with scalar functions 1-13
Options
obsolete configuration 1-20
ORDER BY
BLOB columns prohibited 4-12

CLOB and 4-12
LONGVARBINARY columns prohibited 4-12
LONGVARCHAR and 4-12
Ordinal day of year, determining 3-26

P

Page level concurrency, see Row level locking
PAR file 1-30
PARC, see Pervasive auto-reconnect
Path name
length in USING 3-7
Performance
NSS volumes slower on updates 1-16
Pervasive auto-reconnect
defined 1-29
Pervasive Control Center
OEM characters and 1-33
Pervasive System Analyzer utility 2-1
Pervasive.SQL 7
ALTER TABLE comparison 3-18
status code 59 3-8
UPDATE comparison 3-15
PI() 3-24
Pi, value of
how to find 3-24
Position of sub-string in string
how to find 3-21
POSITION() 3-21
Positional replace in a string 3-22
POWER() 3-24
Predicates
optimized with scalar functions 1-13
PRIMARY KEY
cannot ALTER column 3-17
PSA, see Pervasive System Analyzer
PVSW.LOG
client and server compatibility 1-12

Q

Quarter (ordinal), determining from date expr 3-27
QUARTER() 3-27

R

Radians
converting from degrees 3-24
converting to degrees 3-23
RADIANS() 3-24
Raising x to the power of y
how to find the value 3-24
RAND() 3-24
Random numbers
how to generate 3-24
Redirecting Locator File, see Gateway Locator File
Regional settings
comma as decimal separator 1-16
Release version vii
Repeating a string 3-21
REPLACE() 3-22
Replacing data files 3-11
REPLICATE() 3-21
Restoring previous components 2-13
Retrieving
current user name 3-29
ROUND() 3-24
Rounding
down
use FLOOR() 3-24
to x decimal places
use ROUND() 3-24
up
use CEILING() 3-23
Row level locking 1-12

S

SAR file 1-30
Scalar functions, see Functions, scalar
Search and replace 3-22
SELECT
in UPDATE 3-14
maximum number of columns permitted 1-13
Server version
client and, checking compatibility 1-12
Settings
all sizes now measured in bytes 1-19
dynamic, new 1-19
obsolete 1-20
Sign of a value, how to find 3-25
SIGN() 3-25
SIN() 3-25
Sine value
how to find 3-25

Sizes
now measured in bytes in Configuration 1-19
SmartScout
Installation Toolkit and 1-3
OEM Toolkit and 1-3
replaced by Pervasive System Analyzer 1-19, 2-1
SPACE() 3-22
Space-padding
how to create 3-22
SQL Data Manager
OEM characters and 1-33
SQL Statements
data manipulation 4-12
SQL syntax
changed 3-1
global variables 3-2
new 3-1
SELECT subquery in UPDATE 3-14
TRUENULLCREATE 1-32
USING 3-6
SQRT() 3-25
Square root
how to find 3-25
SRDE
optimizations for scalar functions 1-13
Statements
data manipulation 4-12
Status Code 59 3-8
Status Codes and Messages manual 1-34
String
consisting of spaces, how to create 3-22
finding number of bits 3-21
finding number of bytes 3-21
finding number of characters 3-21
how to repeat 3-21
how to replace characters by position 3-22
how to search and replace a sub-string 3-22
String functions, see Functions, string
Structured Query Language. *See* SQL
STUFF() 3-22
Subquery
correlated 3-14
eliminating duplicate rows with DISTINCT 3-15
in UPDATE 3-14
non-correlated 3-14
Sub-string
how to search and replace in a string 3-22
in a string, how to find position 3-21
inserting into a string by position 3-22
Subtracting
timestamp values 3-28

T

Table
maximum number of columns allowed 1-13
Tables
creating with legacy null support 1-32
TAN() 3-25
Tangent value
how to find 3-25
Tasks
archiving existing components 2-13
choosing PSA functions 2-9
restoring previous components 2-13
testing transactional interface 2-12
testing your network 2-11
viewing PSA log 2-15
Temporary tables
BLOB columns and 4-12
CLOB columns and 4-12
LONGVARBINARY columns and 4-12
LONGVARCHAR columns and 4-12
Terminal Server 1-15
Testing
Btrieve 2-12
for NULL value 3-28
network connectivity 2-11
your system 2-1
Time
how to find current 3-26
Time functions
see Functions, date and time
TIMESTAMPADD() 3-27
TIMESTAMPDIFF() 3-28
Today's date, how to find 3-25
TRANSLATIONDLL connection string 1-33
Troubleshooting information
added to Status Codes and Messages 1-34
TRUENULLCREATE 1-32
TRUNCATE() 3-25

U

UNC pathnames
and Gateway Locator Files 1-25
UNION
BLOB and 4-12
CLOB and 4-12
LONGVARBINARY and 4-12
LONGVARCHAR and 4-12
UPDATE
fails if subquery returns multiple rows 3-15
Pervasive.SQL 7 compared 3-15
with SELECT 3-14
Updates
slower on NetWare NSS volumes 1-16
User name
how to determine current 3-29
USER() 3-29
USING 3-6
length of path name 3-7
Utilities
new 2-1
obsolete 2-2
replaced 2-2
Utility functions
see Functions, utility

V

Variables
CURDATE 3-25
CURTIME 3-26
Variables, global 3-2
@@IDENTITY 3-2
@@ROWCOUNT 3-4
indicated by @@ or @ 3-2
Version
client and server compatibility 1-12
Version checking, automatic 1-12
Version number
internal vii
Visual Basic programming 5-1

W

W32BTXLT 1-33
Wait Lock Timeout 1-22
Week of year (ordinal), determining from date expr 3-28
WEEK() 3-28
Windows 2000 Terminal Server 1-15
Windows 98 Help 1-34
WITH REPLACE 3-11
Workgroup engine
and transaction durability 1-24
floating Gateway 1-23
new Gateway behavior 1-23
syncronizing multiple data directories under one Gateway 1-27