

Pervasive[®] PSQL Performance

Key Performance Features of Pervasive PSQL

Pervasive PSQL White Paper
June 2008

TABLE OF CONTENTS

INTRODUCTION	3
PERFORMANCE BASICS:	
MORE MEMORY, LESS DISK AND NETWORK I/O	3
64-BIT SUPPORT	3
DATABASE MANAGEMENT SYSTEMS (DBMS)	3
MORE EFFICIENT MEMORY USE	4
BETTER READS AND WRITES	7
REDUCING NETWORK I/O	9
SUMMARY	10
APPENDIX A – PERFORMANCE CONFIGURATIONS	11
APPENDIX B – COMPARING STATIC AND DYNAMIC CACHE	12
CONTACT/TRADENAME INFORMATION	13

INTRODUCTION

Pervasive PSQL has a well-earned reputation as a database particularly suited to the requirements of small and mid-sized businesses (SMBs) - very low maintenance, self optimizing, self tuning, easy to install and embed, with flexible deployment options. And, it's always been fast. Applications running on Pervasive PSQL consistently provide a great user experience because we've spent 25 years engineering performance improvements into each new release. This whitepaper highlights some of the more important performance additions we've made to Pervasive PSQL and includes tuning suggestions to help you to optimize your database. For an in-depth look at PSQL performance tuning, please read Chapter 5 of the Pervasive PSQL Summit™ v10 Advanced Operations Guide.

Pervasive's philosophy for increasing performance is to improve overall application performance, rather than focusing on a set of arbitrary benchmarks. The purpose of this paper is to explain the performance enhancements of Pervasive PSQL and to act as a guide to developers, independent software vendors and end users, in understanding these enhancements and in predicting the benefits, both in performance testing and in deployment of applications in the field.

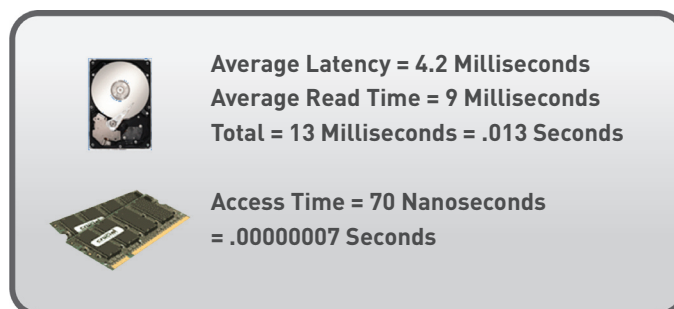
PERFORMANCE BASICS – MORE MEMORY, LESS DISK AND NETWORK I/O

There are three main database-related factors that can affect an application's performance. These are: disk I/O latencies, network I/O, and processor and memory use. Since RAM access is more than 100,000 times faster than disk access, more memory and more efficient use of memory will typically improve application performance. Pervasive PSQL includes several enhancements to enable both of these approaches.

64-BIT SUPPORT – MORE MEMORY

One very simple way to reduce an application's disk I/O is to maximize the amount of RAM available. It makes sense that maintaining data in memory, rather than retrieving it from disk, will improve application performance. After all, disk access is one of the few mechanical (as opposed to electronic) functions integral to processing, and suffers from the slowness of moving parts. On the software side, disk access also involves a "system call" that is relatively expensive in terms of performance.

Disk vs. RAM Access



DATABASE MANAGEMENT SYSTEMS (DBMS) CACHING AND FILE SYSTEM CACHING METHODS

32-bit Intel and AMD machines can theoretically access up to 4GB of RAM. In Windows-based machines, that 4GB is split between the operating system and the applications. This means the most memory that any given application can access is 2GB. So, if a 32-bit Windows machine has 4GB of RAM, adding memory won't have any effect on performance. Using Microsoft's Address Windowing Extension, (AWE), API with Windows 2000 Server and Windows Server 2003, it is possible for 32-bit applications to access more addressable memory. However, AWE usage has to be built into the application and this approach affects other parts of the operating system by reducing the amount of memory available to Windows and drivers.

Pervasive PSQL Summit v10 includes support for Windows Vista™ and 64-bit architecture. Applications with large data sets will see a big benefit from 64-bit servers and operating systems because, in many cases, they will be able to load the entire database into RAM. 64-bit Windows Vista supports from 8GB to 128GB (depending on the version), so applications can preload substantially more data into virtual memory, allowing rapid access and big improvements in application performance. The table below summarizes the address space available to 32- and 64-bit Windows applications.

64-bit vs. 32-bit architecture

ADDRESS SPACE	64-BIT WINDOWS	32-BIT WINDOWS
Virtual memory	16 terabytes	4 gigabytes
Paging file	512 terabytes	16 terabytes
System cache	1 terabyte	1 gigabyte

For developers ready to reap the benefits of a move to 64-bit platforms, Pervasive PSQL Summit v10 is the logical choice. Application users that purchase 64-bit platform operating systems will be able to leverage the additional processing and memory features even with a 32-bit application. Pervasive PSQL supports three types of configurations for optimal performance:

Good

32-bit application talking to a 32-bit PSQL database running on a 64-bit platform, allowing the operating system to utilize internal configuration and native processor power.

Better

32-bit application accessing a 64-bit PSQL database running on a 64-bit platform, extending the database memory for the application.

Best

64-bit application accessing a 64-bit PSQL database running on a 64-bit operating system. This allows all components to run in a native configuration for optimal performance.

MORE EFFICIENT MEMORY USE – XIO AND DYNAMIC CACHING

XTREME I/O

Xtreme I/O (XIO) is a Pervasive PSQL Summit v10 feature that improves performance by accelerating disk access time for Pervasive PSQL data files. XIO is implemented as a device driver for Pervasive PSQL server applications on a 32-bit Windows platform. System requirements include a minimum of 2GB of RAM installed prior to installing Pervasive PSQL. More details can be found in the Advanced Operations Guide.

XIO and the database engine work together to boost performance. The engine notifies XIO when a data file is opened. From that point on, XIO accelerates disk access for the data file. As an example, consider a Pervasive PSQL application inserting a number of records into a table. These inserts result in random I/O requests to various regions of the database file. XIO organizes and streamlines the inserts to reduce the total number of I/O requests for writes and storage. This reduces the load on the storage controller and the PCI bus, and because of the reduced processor load, frees up more bandwidth for other traffic.

XIO is able to work directly with the available RAM to reserve (or assign) its own fully manageable database cache. XIO streamlines write operations and has intelligent compression algorithms to increase the amount of data that can be stored. XIO works transparently – no intervention is required by a user or an application.

XIO Caching

Most caching subsystems support all applications on the system and attempt to optimize overall cache behavior. The cache for XIO is designed to only store application files that are accessed via the Pervasive PSQL database. It also allows the database to extend the engine's cache to whatever physical memory Windows makes available – which means past the 2GB practical limit for most 32-bit applications. XIO first uses extended memory above 4GB to be accessed using the physical address extension (PAE) if it is available. XIO retains the extended memory until the system is shut down; the cache size remains static and does not shrink or expand. If extended memory does not exist, XIO acquires its cache from standard memory (any RAM up to 4GB).

For example, Windows 2000 Advanced Server with PAE allows access up to 8GB of physical memory. With XIO, the database engine's cache is extended by whatever physical memory Windows 2000 makes available. Without XIO, the database cache size can never exceed 1.5GB (or, what's left of the 32-bit Windows 2GB limit after including executables).

With standard memory, XIO balances the memory demands of the database engine with the memory requirements for the entire operating system and other non-database applications. The XIO cache is able to dynamically shrink and expand as other systems resources acquire and release memory. This provides a high level of performance for database servers running multiple applications that are simultaneously accessed by numerous users.

XIO Compression

XIO contains a number of compression algorithms, each of which is suited to a different type of data. Instead of applying one compressor for all the incoming data, XIO applies a forward-looking algorithm on the data and selects the appropriate compression algorithm. By compressing data as part of the caching process, XIO makes better use of cache resources. For example, with a 2 to 1 compression ratio, XIO can store twice as much data in cache.

XIO Streamlined Writes

XIO uses two techniques to improve efficiency when writing data to disk: write aggregation and write ordering. By aggregating several write requests into one larger request, XIO reduces the load on the disk controller and the PCI bus. By ordering write requests, XIO reduces the amount of seeking that the hard drive head must do in order to complete the writes. Reducing the number of writes and organizing the order in which the writes occur significantly improves write times.

When to Use XIO

The following table identifies the application characteristics that are likely to lead to a performance improvement when using XIO:

FACTOR	DETAIL	USE XIO
The application uses a random data access pattern (data access tends not to be sequential).	This is a typical characteristic of an application using a database management system.	Yes
The data set is larger than will fit completely into the Windows system cache.	The more of the data set that does not fit into the Windows system cache, the more disk reads and writes occur. XIO is then able to cache the reads and writes. If the data resides entirely in the Windows system cache, XIO does not process the read or write request.	Yes
The application performs frequent reads and writes to disk.	Disk access is often a significant limiter on application performance.	Yes
You already use a third-party program to accelerate disk I/O.	The use of multiple programs to accelerate disk I/O is highly discouraged because they can interfere with one another and produce unpredictable results.	No

For more information on Xtreme I/O, please review the Performance chapter in the Pervasive PSQL Summit v10 Advanced Operations Guide.

DYNAMIC CACHING

It is a well understood principle in computing that recently requested data is likely to be requested again. Pervasive PSQL and Btrieve have always included an integrated cache feature which keeps pages of data that are also stored on disk. Pervasive.SQL V8 introduced a second level of cache, a dynamic cache designed to operate in changing memory conditions and take advantage of unused memory.

Statically allocated cache (a contiguous block of memory defined during configuration) may be acceptable on a dedicated server where memory use can be closely monitored and predicted, but it is limited in most situations because a) it can't grow to take advantage of unused memory in a system, and b) the contiguous block may be limited by memory fragmentation or by the operating system. For a system where memory availability may change frequently, such as a workstation or server with multiple applications, the ability to adapt dynamically to changing memory requirements as users start and complete other processes becomes critical.

If there are few other memory intensive processes running on a machine, there will typically be a large amount of memory that is not being used and could be made available to cache data and increase overall database performance. Early versions of PSQL Servers had a default configuration to use 20% of physical memory. For a dedicated database server, the default left a considerable amount of RAM unutilized. In addition to configuring a static amount of cache, Pervasive PSQL also uses the operating system cache (to read and write from OS cache instead of to disk). Since no knowledge is shared between the Pervasive PSQL cache and the operating system cache, and both of the caches used the same LRU algorithms (when dropping pages from cache, the least recently used memory is dropped first) to determine what to keep and what to write over, much of the cached data was redundant. In other words, if the Pervasive PSQL cache has 20% of the total available memory, the system cache can have up to 80%. But, up to a quarter (20% of the total) may go to redundantly caching pages already in the Pervasive PSQL cache.

Dynamic Caching – Efficient Memory Usage and Simplified Optimization

Another reason for implementing dynamic caching in Pervasive PSQL is to eliminate the need for the end user to optimize cache settings. The static setting required tuning by the user to achieve the most efficient use of system resources for a particular environment, and re-tuning whenever the environment changed.

Finally, the static database cache required a single contiguous block of memory from the operating system. The dynamic cache doesn't require a contiguous block and can therefore take a greater advantage of the system's available memory.

Dynamic Cache Design

Pervasive PSQL uses a two-tiered cache – L1 and L2. L1 is similar to static cache used in earlier versions of the database, and L2 is the newer fully dynamic cache. Pages are moved between L1 and L2 as needed. L1 contains all the pages that are actively being accessed or modified, and L2 contains pages that have been recently accessed but are not active. The separation of active cache L1 from inactive cache L2 means that dynamic cache management can occur in the background, independent of any specific kernel activity.

Using dynamic cache, the database no longer requires any allocation from the system cache. This improves read access performance two ways: First, the amount of memory used can be controlled based on Pervasive PSQL's data requirements. Second, there are no system calls and the L2 cache can operate under the assumption that only one process is interested in its contents. The system cache must assume that multiple processes may use its resources at once.

Performance Benefit Dynamic Caching

The performance benefits of dynamic cache vary based on factors such as available server memory, database size, and usage patterns. With those caveats, however, we can devise a general performance improvement metric based on relative memory usage.

Let's start with an example of two systems: one using static caching and the other using dynamic caching. Assume that both static cache and L1 cache (in the dynamic caching example) are configured with 20% of available memory. System cache (because of the same type of LRU algorithms) will be mostly redundant with static cache and L1 cache.

Static Caching: With a 4GB server, the amount of memory used by the database will be 20%, or 800MB. (Even though the system cache is using another 800MB because of the database, it's not really helping performance because system and static cache are storing the same data.)

Dynamic Caching: With a 4GB server, L1 cache of 20% uses 800MB, leaving 3.2GB available. The upper limit of L2 cache size is determined by another configuration setting – max memory usage – which is defined as the percentage of available memory used by L1 + L2 cache. If we set max memory to 50% or 2GB in this example, then the amount of memory available to L2 is 2GB (max memory) – 800MB (L1) or 1.2GB.

In this simple example, we've increased the memory available for database caching from 800MB to 2GB, or by a factor of 2.5x. For a detailed calculation of the effects of dynamic cache on performance, please refer to Appendix B.

It is arguable that in some cases, similar results can be achieved by careful tuning of the database server, eliminating system cache and maximizing L1. However, this requires that the system as a whole be essentially static, dedicated as a database server, and have no memory fragmentation – a highly unlikely scenario. Users who don't have dedicated database servers or a database administrator (DBA) to provide ongoing database tuning, will appreciate how Pervasive PSQL is able to perform automatic cache tuning, improving memory utilization on servers and on workstation deployments. For SMB users this is a critical benefit when trying to achieve an optimal configuration and application performance.

BETTER READS AND WRITES – RECORD AND PAGE COMPRESSION, TURBO WRITE ACCELERATOR

Compression

Pervasive PSQL provides two types of data compression: record and page. These may be used separately or together. Compression improves performance by reducing the size of the data files and providing faster reads and writes in cache.

Record Compression

Pervasive PSQL v10 includes a feature that compresses five or more of the same contiguous characters into three bytes. The result, depending on the type of records in your data, can be a significant reduction of space required for storage. Record compression will provide the best results in the following circumstances:

- The records to be compressed are structured so that the benefits of using data compression are maximized.
- The need for better disk utilization outweighs the possible increased processing and disk access times required for compressed files.
- The computer running the database has enough memory for compression buffers (to expand the record during a read).

Record compression is most effective when each record has the potential to contain a large number of repeating characters. For example, a record may contain several fields, all of which may be initialized to blanks by your task when it inserts the record into a file. Compression is more efficient if these fields are grouped together in the record, rather than being separated by fields containing other values.

Page Compression

Internally, a Pervasive PSQL data file is a series of different types of pages. Page compression controls the compression and decompression of data pages within a file.

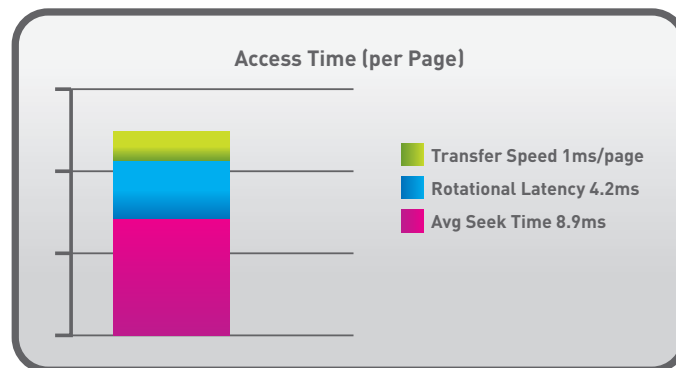
As a file is read from disk, data pages are decompressed and held in cache. Record reads and updates are performed against the uncompressed data in cache. When a write action occurs, the data page is compressed and then written to disk. Depending on cache management, the compressed page is retained in memory until accessed again. Page compression is most effective in the following conditions:

- Data is highly susceptible to being compressed using a ZIP-type compression algorithm. When the file size can be decreased by a factor of 4 to 1 or more, file performance can increase significantly.
- The application has a high percentage of reads (relative to inserts, updates or deletes).

For more information about using record and page compression, please review the Pervasive PSQL Programmer's Guide (Chapter 5) and the Advanced Operations Guide (Chapter 13).

Turbo Write Accelerator

One of the characteristics of writing to disk is that once begun, it costs much less to continue writing than to stop, reposition the head to a new physical track, and begin writing again. In other words, contiguous writes are significantly faster than non-contiguous writes. Furthermore, write calls require interaction with the OS. Eliminating multiple writes in favor of a single large write can significantly decrease the time to perform the write. Turbo Write Accelerator (TWA) pre-allocates open slots within the physical file so that multiple pages can be written as a single coalesced page. This improves I/O performance by reducing fragmentation (and therefore access time) in frequently updated files.



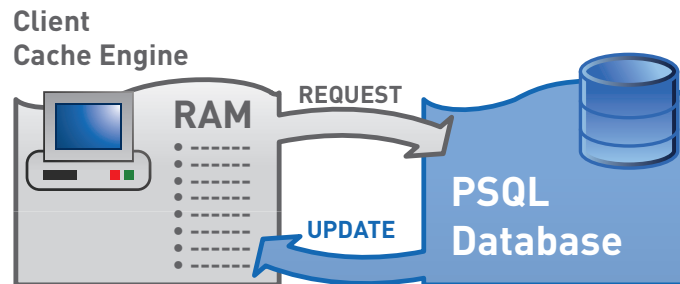
The performance benefits of Turbo Write Accelerator are dependent on the amount of writing taking place, the data page size, and the system transaction settings. Files with smaller pages will generally benefit more than files with larger page sizes because data on larger pages is already more coalesced.

The performance of disk writes using Turbo Write Accelerator can be expected to improve as the number of free pages within the file increases, due to the ability to write multiple contiguous file pages on disk. The setting for adjusting the percentage of free pages is called File Growth Factor. For more details on setting File Growth Factor, please review Chapter 4 of the Advanced Operations Guide.

REDUCING NETWORK I/O – CLIENT CACHE

Client Cache

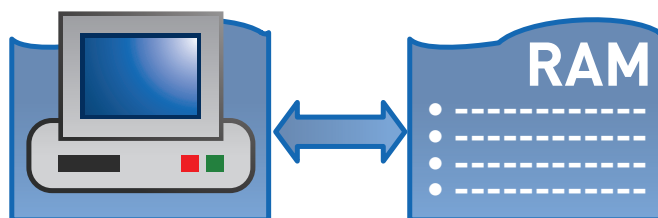
In order to reduce network I/O, Pervasive PSQL caches data at the client (in a client/server configuration). In addition to servicing record-by-record client requests, Pervasive PSQL server also includes the concept of a page server. This moves disk I/O calls from a local engine to a remote server. The Pervasive PSQL client maintains an engine that can operate on a local cache of pages. However, reads with locks, writes, and operations within a user transaction are passed to the server as record-based rather than page based. The client cache and page server manage cache concurrency, and will dynamically detect and switch between client cache and record requester reads, and client cache hit ratio and overall system traffic.



From the application's perspective, frequently used data is always close at hand, maintained in the client cache. All updates, inserts, deletes, locks and transactions are passed through to the server, so client caching does not directly affect write performance. The client cache engine also supports dynamic switching between page/server client cache and record based client/server mode if the performance cost of the client cache begins to outweigh the benefit for a given file.

When Client Caching Helps

The primary benefit of client cache is maintaining a cache of data local to the application and avoiding network traffic or disk I/O when the same data is read more than once. Expect to see the greatest benefit in heavy read operations, such as report generation. In cases where a relatively small and static set of data is referenced repeatedly (for example, a chart of accounts in an accounting table), or where there exists a strong likelihood of reading the same page multiple times, that data may already be available in client cache, and can be returned to the application very quickly without network or disk I/O.



SUMMARY

This paper has highlighted some of the more significant performance enhancements that Pervasive has made to its database products over the course of the past 25 years. Several of these features are unique to the Pervasive PSQL database (and have been patented), while others are examples of how Pervasive PSQL is able to optimize on platform-level technology advancements (like the shift to 64-bit). What has been covered here is by no means an exhaustive list. SQL optimizations, improved utilities, better developer tools and more are all part of the mix in making developers and applications perform better. Pervasive's typical SMB customer is not characterized by dealing with performance challenges in the classic enterprise IT fashion – throwing hardware at it. Because SMBs' limited budgets don't allow the ongoing acquisition of hardware, Pervasive has become very creative at finding different ways to improve database and application performance. The common thread through all of the changes is that Pervasive PSQL is constantly advancing and finding ways to get the very most performance out of the hardware and OS platforms used by SMB customers, and making sure that the same customers can easily adopt new higher-performance platforms as easily as possible.

APPENDIX A – PERFORMANCE CONFIGURATIONS

The following table offers some general configuration guidelines based on RAM available.

SETTINGS	DEFAULT	32-BIT OS <2GB RAM	32-BIT OS 2 TO 4GB RAM	64-BIT OS 2 TO 4GB RAM	64-BIT OS >4GB RAM
Cache Allocation Non-Dedicated Server	20% of RAM	20%	20%	20%	20%
Cache Allocation Dedicated Database Server	20% of RAM	40%	40% up to a maximum of around 500MB	40%	Data set size ¹
XIO ²	On	N/A	On	N/A	N/A
Max Microkernel Memory Usage	60% of RAM	60%	0% ³	0%	0% ¹
I/O Threads ⁴	32	32	32	64	64
Allocate Resource at Startup	Off	On	On	On	On
Back to Minimal State	Off	Off	Off	Off	Off
Index Balancing	Off	Off	Off	Off	Off
Limit Segment Size to 2GB	On	Off	Off	Off	Off
Log Buffer Size	1MB	1MB	1MB	1MB	1MB
Transaction Log Size	2MB	2MB	2MB	2MB	2MB
Tracing	Off	Off	Off	Off	Off
System Cache	Off	Off	Off ⁵	Off	Off

Notes:

- 1) With a 64-bit system you may have enough RAM to cache the entire data set in use. Just be sure to leave sufficient RAM for the OS. Note however for more than 16GB of cache allocation v10 Service Pack 1 or above is required. If there is not enough RAM for the entire data set in use, we recommend setting cache allocation to 20% of RAM and Max Microkernel Memory Usage to 60%.
- 2) XIO is 32-bit only and must have a minimum of 2GB of RAM. XIO default is On.
- 3) With XIO, L2 cached is automatically turned off.
- 4) Windows OS allocates 1MB of memory per thread. For best results, use 1 thread for each 8 open files.
- 5) If XIO is not installed or is disabled, turn system cache On.

Another setting to check when tuning for performance is File Growth Factor, which is based on total database size. The following are good starting points for setting File Growth Factor:

DB SIZE	FILE GROWTH FACTOR
Up to 1GB	5%
1GB to 5GB	2%
10GB+	1%

For a detailed review of Pervasive PSQL performance tuning, read Chapter 5 of the Pervasive PSQL v10 Advanced Operations Guide.

APPENDIX B – COMPARING STATIC AND DYNAMIC CACHE

Based on a set of performance timings, we determined the average cost of reading a page from L1 cache to be about 110 μ s. Cache management tasks in L2 account for another 210 μ s on average, or 320 μ s total per page. The cost of reading the same page off of disk is 1900 μ s, or about 18 times longer than reading a page from cache. These timings are of course specific to a particular hardware configuration and operating system, and should not be generalized, but they will serve to illustrate the difference between dynamic cache and the static cache used in prior versions of Pervasive PSQL.

Using static cache, assume 20% of reads would hit the Pervasive PSQL cache (L1), ideally 20% would hit system cache, and 60% would require disk I/O. In the ideal case where L1 and system cache contain no redundant data, and access to system cache incurred no penalty, 1000 reads would be the sum of the cache access times ($400 \times 110 \mu$ s) = 0.044 sec., and the disk access times ($600 \times 1900 \mu$ s) = 1.14 sec., or 1.1840 sec. In reality, L1 and system cache would probably be largely redundant, resulting in only 20% of the data in cache, and access time of $(200 \times 110 \mu$ s) + $(800 \times 1900 \mu$ s) = 1.542 sec.

With dynamic cache, we have a 20% chance that the reads will hit L1, a 50% chance that reads will hit L2, and a 30% chance that reads will require disk I/O. Thus, 1000 reads would take $(200 \times 110 \mu$ s) + $(500 \times 320 \mu$ s) + $(300 \times 1900 \mu$ s) = .022 sec. + .16 sec. + .57 sec. = .7522 sec., between 1.5 and 2 times the performance of static cache.

Contact Information

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

United States

800.287.4383
512.231.6000
Fax: 512.231.6010
info@pervasive.com

EMEA

+800.1212.3434
cic@pervasive.com

<http://www.pervasive.com>

©2008 Pervasive Software Inc. All rights reserved. All Pervasive brand and product names are trademarks or registered trademarks of Pervasive Software Inc. in the United States and other countries. All other marks are the property of their respective owners.