Configuration and Coding Techniques

Performance and Migration
Progress DataServer for Microsoft® SQL Server™

PROGRESS
SOFTWARE

Introduction

This white paper is intended for designers and programmers who plan to develop Progress 4GL applications that run with Microsoft® SQL Server™ databases. This white paper focuses on techniques that help designers and programmers maximize the benefits of the Progress DataServer for Microsoft SQL Server. Topics covered in this white paper include:

- Hardware configuration
- Network configuration
- Software configuration
- Programming techniques
- Migration
- Performance optimization
- Error messages
- Suggested reading

Information contained in this white paper covers specific development or deployment environments and scenarios. Since these specific environments might not match your environment exactly, you will need to make configuration decisions based on your goals and constraints. Use this information as a guide when you set up and implement your specific environment. Before you decide to use any of this information for deployment, test these techniques in your specific environment and select the ones that benefit the optimization of your specific deployment.

The DataServer for Microsoft SQL Server operates in a complex environment of interconnections between software, hardware, and network components. All of these areas are evolving and changing. In order to keep up with these changes this document will be periodically updated. As additional information becomes available, Progress Software Corporation will release updates to this white paper.

NOTE: This document does not replace or exclude the Progress DataServer for Microsoft SQL Server documentation that ships with the Progress DataServer for Microsoft SQL Server product. The Progress DataServer for Microsoft SQL Server documentation is the primary source of information for this product. To make full use of this white paper, Progress Software Corporation recommends that you first read the documentation that ships with the Progress DataServer for Microsoft SQL Server.

Throughout this document all references to Microsoft® SQL Server™ and SQL Server™ refer to Version 7 of the Microsoft® SQL Server™ product, unless otherwise stated.

System Configuration

The following sections discuss hardware, network, and software configuration considerations for SQL Server.

Hardware Configuration

The foundation of any system is the hardware. Using SQL Server follows the general rule that applies to almost all computer environments: more and faster hardware is better. Unfortunately resources are limited. Due to this limitation the competing factors of cost versus performance must be balanced.

Generally, tuning hardware requires working with three different areas. One of the three areas is always at the root of hardware bottleneck. This is not to say that a hardware change is always the solution to any given problem, though. Once a hardware bottleneck is identified, the other parts of the system, such as your network, should be reviewed to see if they should be modified instead of, or in addition to, the hardware changes under consideration.

The three hardware areas include the:

- CPU
- System memory (RAM)
- Storage system (hard drives and access methods)

If your environment is constrained by the CPU it is an indication that your system is running at or near capacity. The only solution to bottlenecks in the CPU is to add more CPUs or to switch to faster CPUs.

Bottlenecks in system memory are resolved by adding more memory. In evaluating whether or not to add system memory, make sure that the computer system, the operating system, and your software will be able to take advantage of the increased memory.

The most common hardware bottleneck is the access speed of reading and writing to the storage system. Problems in the area of the storage system can be resolved by:

- Adding more hard drives
- Increasing the speed of the access methods
- Optimizing where objects are stored on the hard disks
- Using a combination of these options

The optimal configuration for a DataServer environment depends on the specific demands of the application. However, you can use the following general guidelines as a starting point when configuring a new system or when optimizing an existing system.

When designing a new system or adding to an existing system it is almost always more beneficial to have more small-capacity hard drives than a few large-capacity hard drives. Each additional hard drive adds flexibility in how the system can be configured.

The following logical software components are areas that can be spread across hard drives to maximize access speed:

- Operating system
- Temporary files
- Application source code
- Database
- Database log file

In an ideal environment, each one of these logical software components would be placed on separate hard drives. For the database, it is especially beneficial to use multiple hard drives. In most environments, though, there are not enough hard drives to place each of these components on a separate one.

For example, if you have two hard drives, try placing the database log file on its own disk and all other components on the second disk. The advantage of this configuration is due to the high sequential write activity to the log file. With the exception of the log file all of the other components are often accessed randomly from one read to the next. With the log file, however, the access is usually a series of sequential writes. As the database is processing transactions, it must write to the log file to complete a unit of work. The entire system often waits during these log file writes. If the log file is on its own disk, the write head is almost always in the correct position for the next write. This allows the system to complete transactions faster. If even one other component is placed on the hard disk with the log file, you lose this performance advantage.

For systems that have three hard drives, the database should be placed on its own disk drive. Or as an alternative, and a better configuration for a larger database, a three-disk configuration should be set up to split the database across two disks with the third disk containing the database log file. Place the other components on the two disks that hold the database. If you have additional disks, you can expand this idea further by either moving additional components, such as temporary files, to their own disk or by continuing to spread the database across more disk drives.

Network Configuration

Typically the DataServer runs over a TCP/IP network. Progress Software Corporation provides no recommendations beyond the standard configuration for optimizing the network configuration.

As a general rule, you can best utilize the network by minimizing the amount of network traffic. For remote connections you might want to consider locating the schema holder, application source code, and the Progress client code on the client machine instead of on the network or on a local server. Any benefit in performance that may be gained from locating these objects locally needs to be weighed against the increased maintenance that is required to update these objects in multiple locations, as opposed to a single location on a network.

One network consideration is the choice of using the Enterprise DataServer versus the Personal DataServer. The Enterprise version of the product uses Progress Networking. The Personal DataServer uses an ODBC driver to send data over the network. Implementations vary so it is recommended that you do tests in your specific environment to see which of these options provides the best performance. In addition to performance, consider the implementation and maintenance costs in your implementation of the DataServer.

Software Configuration

You must use several software components in any configuration that uses the Progress DataServer for Microsoft SQL Server. These components include, but are not limited to, the server operating system, the client operating system, SQL Server, the Progress Client, Progress server side software, the Progress DataServer for Microsoft SQL Server, and the ODBC driver configuration.

Server Operating System

The standard configuration of the server operating system works for the DataServer environment under most circumstances. As a general rule, follow the performance-tuning guidelines for your operating system.

If you have other applications running on your operating system in addition to the DataServer environment components, you will need to tune the software resources to balance the needs of multiple applications on a single system. Where possible, Progress Software Corporation recommends that the server be dedicated to the DataServer environment.

Client Operating System

The standard configuration of the client operating system works for the DataServer environment under most circumstances. As a general rule, follow the performance-tuning guidelines for your operating system.

Microsoft SQL Server

The standard configuration works for the DataServer environment under most circumstances. You'll need to consider certain options when installing, using tools, and optimizing the environment.

Installation

For simplicity, select the default option of case-insensitive sorting during the installation of SQL Server. This default uses the same type of sorting as Progress. If you choose the installation default of case-insensitive sorting and you are using the Progress-to-Microsoft-SQL-Server (ProToMss) migration tool, do not select the option to create the shadow fields during the migration. You do not need the shadow fields for case-insensitive sorting. You can improve performance if you avoid the unnecessary overhead of shadow fields.

Tools

The two primary tools for working with a SQL Server database are the Enterprise Manager and the Query Analyzer. Anyone working with SQL Server should familiarize themselves with these tools. The Enterprise Manager provides numerous functions including a display of the database layout. The Query Analyzer provides an interface for running Transact SQL scripts.

Additional tools to use include the Service Manager and the Profiler. You need the Service Manager to control the starting and stopping of SQL Server and other associated services. The Profiler provides a number of features. In the DataServer environment you can use the Profiler as another way to see the actual SQL code as it runs against the SQL Server database. The Profiler also provides many options that allow you to evaluate performance.

For more information on the tools that are available with SQL Server see the Microsoft SQL Server Books Online at the Microsoft Web site.

Optimizing the SQL Server Environment

Microsoft SQL Server 7 has been changed in many ways relative to its predecessor Version 6.5. A major change is that for Version 7 most of the database server environment parameters are self-tuning. However, there are some settings that can be evaluated on an environment-by-environment basis when fine-tuning the performance of SQL Server 7.

As a caution, you should only make changes to the SQL Server parameters if you understand the potential impact on SQL Server. Changes in this area are made for the entire server and will impact all SQL Server databases that run on that server. Below are a few specific areas to consider. It is recommended that prior to making any changes to these parameters you read the documentation in the SQL Server Books On Line.

Recovery Interval

The recovery interval is the desired amount of time in minutes that it would take to recover the database from a crash. It is one of the factors that determine when a checkpoint occurs. If the checkpoint occurs too frequently, the performance of the application will suffer.

The default value is 0 for SQL Server 7. This equates to approximately a one-minute recovery time. Setting the recovery interval to 10 sets the estimated recovery time to ten minutes. The recovery interval is set per server and applies to all of the SQL Server databases that are on that server.

To determine whether changes to the recovery interval can improve the performance of an environment, try a series of tests by setting the recovery interval to 0, 10, and 30. If the performance for your environment is improved by modifying the recovery interval, you can consider whether you would like to use that interval either part of the time or permanently.

To see what the current recovery interval is set to, use the following command:

```
sp_configure "recovery interval"
```

The run_value column in the output from the command shows the current recovery interval.

Changing the recovery interval is an advanced option and requires that the ability to change advance options be turned on for the server. To set the recovery interval, you can use the following command:

```
sp_configure "recovery interval",30
go
reconfigure
go
```

Note that the reconfigure command needs to be executed for the setting to take effect. Changing the recovery interval does not require SQL Server to be restarted for the command to take effect.

max worker threads

If you are running your application on an otherwise lightly used server system, you can experiment with setting the max worker threads to less than the default value of 255. This value can be set in server properties through the Enterprise Manager (on the Processor tab). The performance of the server might then improve since the system has fewer processes to track.

fibers

SQL Server allows fibers to be used in addition to threads. Windows NT fibers are off by default. This value can be set in server properties through the Enterprise Manager on the Processor tab. Using fibers can reduce the number of context switches that require calls to the operating system. Most application environments will benefit by toggling this option on.

Progress Client

The DataServer requires two database connections, one for the Progress schema holder database and another for the connection to the SQL Server database.

If you need to create tables in a Progress database in addition to the ones used by the schema holder, Progress Software Corporation recommends that you create a separate Progress database instead of adding the tables to the schema holder database. In other words, to keep the environment simple, do not use the schema holder database for any purpose except to hold definitions for DataServer. Following this rule makes it less complex to resolve and recover from any problems that might arise.

Deployment

You can set some DataServer parameters that will improve performance for the deployed application.

See *Progress DataServer for Microsoft SQL Server Guide* for information on DataServer specific parameters. In addition to setting the parameters that are required, consider using the DataServer (–Dsrv) parameter with the following options:

- Use the parameter –Dsrv SKIP_SCHEMA_CHECK to skip the check of the schema and thus save some processing time when you first access a table in your 4GL application. Because this option bypasses the client and server schema handshake when a table is opened, you must ensure that no schema mismatches exist. To prevent exposure to mismatches that can potentially corrupt data, you should perform a final schema pull just prior to deployment before setting this option. You can confirm if this option is implemented in your environment by reviewing the dataserv.lg file.

- Use the parameter –Dsrv QT_CACHE_SIZE,x to set the cache size by default for data returned from queries. When you use this parameter, x represents the number of bytes. The following is an example of how the parameter is used: –Dsrv QT_CACHE_SIZE,30000. Generally speaking, if your tables have large record sizes and/or your queries generally contain a large result set, it might make sense to increment the cache size. However, use caution in setting this value since setting it too high can also cause degradation in performance while caching takes place. To determine the optimal value, test the settings in the specific application environment where you plan to set and use the parameter.

Connection Parameters

Consider setting the following connection parameters when configuring your DataServer environment. They are not unique to the DataServer environment, but they do play a roll in the performance. The following list of connection parameters does not provide a complete definition of these parameters, you can find a detailed definitions in the *Progress Startup Command and Parameter Reference*:

- **Message Buffer Size (–Mm) —** For most environments it is best to set the message buffer size as large as possible. Setting this parameter to a larger value enables more records to be sent across the network in one packet. For example, try a setting of 2048.

- **Maximum Memory (–mmax) —** Set this value to allocate memory on the client. For example, try a setting of 4096.

- **Quick Request (–q) —** Add this parameter to avoid searching the PROPATH after the first time Progress finds a program. This improves performance, but it should only be used in a production environment where the code is not being modified.

- **Read Only (–RO) —** Use this parameter to set Progress database access to read only. In most application environments, you can access the Progress schema holder as read only. Since setting this parameter stops the logging of most of the before-image activity, setting the access to read only can provide a slight improvement in performance. To avoid a startup warning message when using the Read Only (–RO) parameter, the before-image file must be truncated.

To truncate a before-image file, enter the following command from a DOS prompt:

proutil <database name> –C truncate bi

For more information on using the proutil command, see the *Progress Database Administration Guide and Reference*.

When first using the proutil command, a common problem occurs stating that the command is not found in the DOS environment. You can fix this by adding the Progress bin directory to the system path. For example, if you installed Progress in the default directory on the C drive, the following would be added to the path: 'C:\program files\progress\bin'.

- **Temporary Directory (–T) —** Use this parameter to specify where temporary files are stored. Set the temporary directory to use a local drive for temporary files. Otherwise, if you store the temporary files on the network, there might be major performance degradation for the individual user and for the networked system as a whole.

- **Speed Sort (–TB) and Merge Number (–TM) —** Maximize these values to provide additional memory for merging and sorting. For example, set these values to: –TB 31 –TM 32.

- **Four Digit Year Default (–yr4def) —** Add this parameter so that all years are always dumped as 4 digits. This avoids ambiguity when reloading data.

Progress Server

You use the Progress Explorer to manage Progress servers. When using the Progress Explorer to allow multiple connections to the schema holder, it is common to configure a database broker instance for the schema holder database when you are configuring your environment. However, if you connect the database read only, you might not need to configure a database broker instance for the schema holder. For example, you might choose to use the Read Only (–RO) connection parameter to connect the schema holder database for improved performance. See the "Progress Client" section for more information on the (–RO) parameter.

The Progress Explorer is also used to configure DataServer connections. This configuration is needed when you are using Progress networking to connect to the SQL Server database via the DataServer. If you are connecting all of your clients self-service to SQL Server using the DataServer, you do not need to configure a DataServer connection in the Progress Explorer.

A quick way to tell if your connection is self-service versus a brokered connection is to look at the Progress connection options for the SQL Server connection. If the connection options contain the Service Name (–S) parameter, you are using a brokered connection.

See the Progress Software Corporation documentation set for information on configuring the server environment.

Progress Microsoft SQL Server DataServer

After initially setting up the DataServer environment, review the Edit Connection Information dialog box. Progress saves this information, once, after the initial connection to the DataServer. Any information that you do not want to use for the default connection should be modified or removed; for example, the user and password information. Leaving this information in the default connection string could cause a security problem or a database connection that you do not intend.

More specifically, most sessions explicitly specify a connection to the DataServer database when they first start up an application. After the session is initially connected, it is possible for the connection to be lost, and for the session to try to auto-connect to the DataServer database. The auto-connection uses the default

parameters saved in the Edit Connection Information, not the connection parameters specified on the initial connection.

See the *Progress DataServer for Microsoft SQL Server Guide* for information on how to edit the connection information.

ODBC Configuration

When configuring the Data Source Name (DSN) for the ODBC Driver, use the System DSN tab folder instead of the User DSN tab folder. The System DSN configuration makes the ODBC DSN available to all users on the system.

When configuring the DSN, the SQL Server database must already exist. When configuring the DSN, you specify the database name and a test is run to confirm that the ODBC connection to the database works. If the SQL Server database does not exist, you can create an empty SQL Server database with the correct name to allow you to configure the ODBC connection.

Programming Techniques

Programming issues fall primarily in the area of the Progress 4GL. Depending on your environment, you might also be using other languages. For example, you might use Transact SQL in areas such as stored procedures or when submitting a query directly to SQL Server from your Progress client.

Database

This section covers key information regarding programming techniques to maximize your database and SQL Server performance.

Covering Indexes

Adding or modifying indexes that act as covering indexes can improve performance. Covering indexes are SQL Server indexes that include all of the fields needed by a particular query. You gain a performance advantage by using a covering index since only the index needs to be read to return a record, not the actual record. SQL Server takes advantage of covering indexes automatically.

From a Progress 4GL perspective, you can use covering indexes on a NO-LOCK query by using the FIELD-LIST clause to specify the fields in the covering index. The covering index must include all of the fields in the field list as well as any fields that are part of the indexing. In many cases, the DataServer makes use of the RECID field. For this reason, Progress Software Corporation recommends that you include the RECID field as the last field in the covering index, if it is not already part of the index.

Every index has a cost to the performance of the system since the SQL Server must maintain the index as it performs writes and deletes on the table. Progress Software Corporation recommends that you do performance testing with and without the covering index to determine if the covering index results in an improvement in your specific environment. Gains in performance with a covering index occur only when there is a relatively large number of reads relative to the number of writes; additionally, there must be significantly fewer fields in the covering index than in the base record.

To learn more about covering indexes and how to define them, see the SQL Server Books Online that ship with the SQL Server. If you make indexing changes, be sure to update the schema holder with the modified SQL Server schema information. See the *Progress DataServer for Microsoft SQL Server Guide* for information on how to update the schema holder.

Database Format Size

One of the more common problems found in migrating a database from Progress to SQL Server is the format size relative to the actual data that is stored in the database. SQL Server has a maximum size for each field that must be specified at the time a table is created. When you use the Progress tools to migrate a database, Progress uses the format definition to specify how large the fields should be in the SQL Server database. An error will occur if the format is exceeded when data is stored into the field. This type of error occurs with two Progress data types: CHARACTER and DECIMAL.

Additionally, an option exists on the ProToMss migration tool that allows you to use the Width field instead of the Format field when specifying the physical storage size. Progress provides the Width field so that a physical size of a database field can be specified that is different than the display format. The initial value of the Width field depends on the data type. The CHARACTER data type will have a value two times the format. The DECIMAL data type will be 15 digits plus the number of decimals declared. The Width field can be modified in the Progress Data Dictionary by using the SQL Properties option. If you choose to use the Width field instead of the Format field, you can control the size of the SQL Server database field without changing anything the Progress client will use (i.e. the Format field). Since this method does not change the Progress database version of the application, Progress Software Corporation recommends this method for resolving field size problems.

Field size problems can cause errors when loading data. In the case of a CHARACTER data type, the error Progress generates is "You tried to compare or to update a character field with a value longer than the maximum length (6142)." There are two ways to resolve this error: you can either increase the field size or you can change the data type to TEXT. In almost all cases, you should increase the field size. See the "Large Formats" section for more information on TEXT data types.

If you are migrating a database using the ProToMss tool, you can go back to the Progress Data Dictionary and update the Width field. After you update the Width field, migrate the database again.

In the case where the value of a decimal exceeds the size defined for the DECIMAL data type, the Progress session will crash. Generally these types of errors are less common and harder to track down than problems with the size of CHARACTER formats.

One way to identify all of these types of problems is by using a tool, developed by Progress Software Consulting, which will read through an existing Progress database prior to migration. The tool compares the actual data in the database against the width specified in the _ Field meta-schema table. If you use this technique, you should run the tool against each set of data that is going to be migrated to the SQL Server database. If you would like more information on this tool, contact Progress Software Consulting.

Descending Indexes

SQL Server 7 does not support descending indexes. If you are migrating a Progress database that has descending indexes, you should choose the ProToMss option to add the descending indexes to the database. When the Progress descending index is selected, Progress will use the ORDER BY phrase with the DESCENDING keyword when generating the SQL statements.

Selecting the option to add the descending indexes provides an alternative to changing all of the queries that specify a descending index with USE-INDEX or that select a descending index indirectly. This option is selected in the Data Administration tool. From the DataServer menu, select the "Progress DB to MS SQL Server" menu item and activate the "Create Desc Index" toggle on the ProToMss dialog box. This option retains descending index definitions in the schema holder while creating ascending indexes in SQL Server.

If you do not used the ProToMss tool to build your MS SQL Server database with the create descending indexes selected, review the following sections for changes that might need to be done to your code.

Descending Indexes on a FIND statement

To account for the fact that SQL Server 7 does not support descending indexes, when writing or converting Progress 4GL code to work with the DataServer you must modify FIND statements that use indexes with descending components. In a Progress database environment, a FIND statement might select an index with descending components either explicitly or implicitly. The explicit case occurs where the FIND statement uses the USE-INDEX clause. The implicit case occurs when a FIND statement selects an index that has descending components using the Progress index selection algorithm at compile time. See the "Descending Indexes" section for more information.

The DataServer supports descending sorting through the DESCENDING option in the 4GL ORDER BY clause. The FIND statement does not support the BY clause. When using the DataServer, you cannot use the FIND statement for queries that need to be accessed by a sort order that contains descending components. In these cases, you must change the FIND statement to a FOR FIRST or an OPEN QUERY statement, since both of those statements support the ORDER BY clause. In the case of an OPEN QUERY statement, Progress uses the GET statement to access the record.

Descending Indexes and USE-INDEX

You must modify all queries that use USE-INDEX to specify an index with descending components. The ORDER BY clause must be used along with the DESCENDING keyword for each field that needs to be sorted in descending order instead of USE-INDEX.

FOR FIRST in Place of FIND FIRST

Using FOR FIRST in place of FIND FIRST can improve performance when retrieving a single record that is being accessed NO-LOCK.

For example, if your application uses the following FIND FIRST code:

```
FIND FIRST <table-name> WHERE <where clause> NO-LOCK.
```

The code can be replaced with:

```
FOR FIRST <table-name> WHERE <where clause> NO-LOCK:
END.
```

In the case of FOR FIRST, Progress makes the record available beyond the end of the FOR FIRST loop.

Large Formats

The ProToMss migration tool provides an option to create character fields that are greater than a specified size as SQL Server TEXT data types. The default data type for character fields is VARCHAR. Where possible you should use the VARCHAR data type, instead of a TEXT data type, since VARCHAR provides the best performance for character fields. However, the size of a field may require that you create it as a TEXT data type due to factors such as the maximum field size or the maximum record size for the SQL Server database.

For example, the DataServer and SQL Server can handle a VARCHAR field that is 2000 characters. However, if a single record contained 10 fields of 2000 characters, the total size of all the fields would exceed the maximum record size for SQL Server. The only way to create that record with those fields is to specify the fields as TEXT data types instead of VARCHAR. For more information on TEXT fields see the "TEXT Data Type" section.

TEXT Data Type

The amount of data that can be accessed in a field defined to be TEXT by a Progress application is limited by the maximum size of a record that can be passed to the client. The maximum size of a record is 32K.

There are also performance considerations when using a TEXT data type for a field in a record. Progress uses binding to optimize read performance in some cases. This optimization cannot be used if the record being returned contains a TEXT field. When possible either avoid defining fields in a record with the TEXT data type or use a field list that excludes the TEXT fields from the returned record set. Using either of these techniques allows Progress to take advantage of the fastest method available to read records.

Progress 4GL

This section documents Action Segment Overflow, Locking, Index Reposition, Meta-Schema, Record Creation, Transaction Boundaries, USE-INDEX, and Word Indexing for SQL Server performance and migration.

Action Segment Overflow

Progress 4GL code compiled against a DataServer uses more space in the action segment than code that is compiled against a Progress database. There might be action segment errors when compiling existing programs against a DataServer that compile successfully against a Progress database if the programs are near the allowed maximum action segment size.

To resolve this problem, the code must be modified to reduce the size of the action segment. One way to solve this is to move code from the main block into internal procedures. See the Progress Knowledge Base (http://www.progress.com/support/kb.htm) for more information on resolving action segment issues.

Locking

For simplicity, Progress Software Corporation recommends that you avoid using SHARE-LOCK unless absolutely required. If you use SHARE-LOCK in your code, you should remove it if you have the option.

If a lock type is not specified, Progress often defaults to a SHARE-LOCK. For this reason, Progress Software Corporation recommends that each query should always explicitly specify a lock. For example specify either an EXCLUSIVE-LOCK or a NO-LOCK on each query statement. Specifying a lock removes ambiguity so that it is clear to anyone who works with the code which lock is being selected.

Review your code to see if it relies on the automatic downgrading of EXCLUSIVE-LOCK to SHARE-LOCK. If you find any sections that rely on this functionality, you might need to remove the SHARE-LOCK dependency.

You can achieve Progress-compatible results when you use the DataServer startup parameter TXN_ISOLATION,1. This causes SQL Server to run with an isolation level of Read Uncommitted. Read Uncommitted is the closest match to the NO-LOCK/EXCLUSIVE-LOCK lock types that the Progress database uses. However, SHARE-LOCK is treated like NO-LOCK at this isolation level. If you depend on SHARE-LOCK in some parts of your code, you must program your application to achieve a similar type of locking in the DataServer environment. To achieve this, you can set your transaction isolation level in accordance with your desired share-lock behavior as defined by ODBC and its implementation on SQL Server. The unfortunate side effect of changing the isolation level from read-uncommitted is that a Progress NO-LOCK will now behave in conformance with the share lock conformance of your selected isolation level. Changing isolation level from the recommended read-uncommitted isolation level may significantly affect non-exclusive query performance. See the DataServer and SQL Server documentation for more information on transaction isolation levels.

Index Reposition

The INDEXED-REPOSITION keyword is supported by the DataServer. This can significantly improve on-line random access. Use this option in 4GL code and in browsers to reposition randomly within the result set of an open query. However, be aware that using this option causes performance degradation, since repositioning will otherwise be performed sequentially through the query result set.

Meta-Schema References

Meta-schema references in existing code that are qualified with a logical database name, such as <database>._field, must be modified. These are easily identified in a code base since compile-time errors are produced for each program that uses this type of database reference.

In the Progress database environment, the Progress meta-schema tables reside in the same database as the data. In the environment, database name qualifications to table and field references all use the same logical database name.

When using a DataServer, however, the Progress environment has two connected databases. One database is a SQL Server database. The other database is the Progress schema holder. In this environment the data resides in the SQL Server database and the Progress meta-schema tables reside in the schema holder database. In this case there are two logical database names.

You can resolve compile errors from meta-schema references that are qualified with a database name in one of two ways depending on the number of databases in your environment. The most common case occurs when your environment uses only one Progress database. This means there is only one set of Progress meta-schema tables. If this is the case, you can delete the logical database reference from the code. Progress Software Corporation recommends this method since the code change works for each type of data source. However, if you have multiple Progress databases in the environment, the database qualification should be changed to the logical database name of the Progress schema holder. In this case, the code might differ depending on the data source. See the “Migration Issues” section for information on handling differences in code that is specific to a data source.

Record Creation

Progress creates a record at a different point in the sequence of events in the Progress database than it does in the SQL Server database. Specifically, Progress creates records after all of the key fields are assigned, or when a RECID/ROWID is assigned to a variable or at the end of the record scope. The DataServer creates records in SQL Server database at the end of the record scope.

The following techniques can be used when creating database records to avoid unexpected results:

- Assign Unique Index Fields on Create —When possible, assign all index key fields right after the record is created, more specifically, right after the Create statement. Check the definitions in the Progress Data Dictionary to make sure that all of the fields in each unique index are being assigned.
- ROWID/RECID—Using the RECID or the ROWID function causes Progress to write the record to the database. Do not call the RECID/ROWID function until all of the unique key fields are set.
- Use RELEASE and VALIDATE statements—Combining RELEASE and VALIDATE statements causes Progress to write the record to the database. Do not use the RELEASE and VALIDATE statements until your code has set all of the unique key fields. You can use the RELEASE and VALIDATE statements to force Progress to write a record to the database so that it is available if the record needs to be reread prior to the end of a transaction. Remember, if you use RELEASE, the record is no longer available in the record buffer.

Transaction Boundaries

The Progress database has a unique feature in its ability to hold a record lock beyond a transaction boundary. This is in contrast to standard SQL-based data sources, like SQL Server, which release all current locks on a given connection at the end of a transaction. This difference in behavior has repercussions with the use of record buffers in your 4GL code as illustrated with the following example. The three transactions below are executed in sequence. The first and third are marked (internal) meaning they are executed from this session. The second is marked (external) meaning a separate session issues this transaction sequentially after transaction 1:

```
DEFINE BUFFER b1_state FOR state.
DEFINE BUFFER b2_state FOR state.

/* Transaction 1 (internal) */
DO TRANSACTION:
        FIND b1_state WHERE b1_state.State = "CA" EXCLUSIVE-LOCK.
        DISPLAY b1_state.State-Name.
END.

/* Transaction 2 (external) */
PAUSE MESSAGE "The state name is California – Now somebody in another session
                updates the state name associated with "CA" to "Carolina" and
                commits the new name change."

/* Transaction 3 (internal) */
DO TRANSACTION:
        FIND b2_state WHERE b2_state.State = "CA" EXCLUSIVE-LOCK.
        DISPLAY b1_state.State_Name b2_state.State_Name.
END.
```

The value displayed for State_Name in transaction 3 is "California" in both buffers even though the name was externally changed to "Carolina" in transaction 2. This is because Progress assumed the lock on the state record was held between transaction 1 and transaction 3 and therefore the buffer information in b1_state was shareable with buffer b2_state. Since SQL-based transaction scoping does not allow for lock retention beyond the transaction boundary, the second buffer, b2_state, is actually stale at the time of transaction 3. To prevent the use of stale buffers, they should be explicitly released at the end of a transaction using the Progress RELEASE statement. After transaction 1 was completed, the following syntax should have followed:

```
RELEASE b1_state.
```

USE-INDEX

As noted in the *Progress DataServer for Microsoft SQL Server Guide*, you must specify USE-INDEX or use the ORDER BY clause to return records in a predictable order.

Against a Progress database, USE-INDEX controls which index is used. However, for the SQL Server database, you cannot guarantee which index is selected. The index selected in the 4GL will dictate the ORDER BY clause generated for SQL execution. However, ultimately the cost-based query optimizer in the SQL Server database engine makes the actual index determination.

The DataServer has not implemented index hints directly through the 4GL. However, you can pass SQL through the send-sql-statement stored procedure that contains hints since the syntax is just passed through directly to the SQL Server engine. You could also call a stored procedure from your 4GL code that contains your SQL hints. When index hints are specified, it does not ensure that SQL Server will use the specified index. Keep in mind that SQL Server optimizes index selection based on the ORDER BY clause, WHERE

bracketing, table statistics, etc. and in most cases will determine through cost-based analysis the best index selection.

Using the BY clause in your 4GL statement translates into an ORDER BY in the DataServer. SQL Server uses the ORDER BY to assist in selecting the correct index. Using the ORDER BY clause leaves the SQL Server flexibility to use the provided information in selecting the best indexing method.

Progress Software Corporation recommends you limit USE-INDEX when possible since it overrides the Progress compile-time index algorithm selection. The primary reason to avoid USE-INDEX where possible is to keep flexibility for future changes. However, there are exceptions to the recommendation to avoiding the use of USE-INDEX. One advantage to USE-INDEX is that it ensures that the ORDER BY clause generated for server execution will correspond exactly to an existing index definition, so if your index definitions change, you do not have an ORDER BY clause in your code without a supporting index.

For example, based on performance, indexes may be added or deleted. Code that uses the BY clause requires only a re-compile to work with the new indexes. However, if USE-INDEX is explicitly specified, you must modify the code to take advantage of new indexing. If an index is added that can provide better performance for the query, the BY clause is flexible since it allows the Progress index algorithm selection to select the new index over the old one. However, code with USE-INDEX would need to be modified in both the case of a dropped index and in the case of a new index that offered improved performance.

Word Indexing

Word indexing is a Progress database-specific feature. At the very least, any code that uses the CONTAINS clause in a query must be removed or eliminated from the code at compile time using preprocessor statements. See the "Migration Issues" section for more information on using preprocessor statements.

SQL Server has a feature that is similar to word indexing called Full Text Search. Though these features are similar, they are not equivalent. Full Text Search indexes are not updated automatically. Administrative functions must be implemented to update Full Text indexes. Due to the performance of the update, there will always be a period of time between when the database change is made and when it is reflected in the Full Text index. Using Full Text Search in place of word indexing requires a considerable amount of design and coding. Support for Full Text Search is not currently part of the DataServer product. If you would like help in designing and/or implementing this type of feature, contact Progress Software Consulting.

Migration Issues

Migrating existing Progress-based applications to use the SQL Server database requires some changes. The amount of changes that you need to make depends on the implementation of the code. If you are familiar with the code of the application you are migrating, you can make a rough estimate of the number of changes by reviewing the specific coding issues in Chapter 2, "Programming Considerations" of the *Progress DataServer for Microsoft SQL Server Guide* and by reviewing the "Programming Techniques" section in this white paper.

Progress 4GL Code Designed to Use a Progress DataServer Product

Most of the changes made to Progress 4GL code that allow it to work against other non-Progress data sources, such as Oracle and DB2/400, apply to this DataServer environment. Applications already designed to access a DataServer are likely to need very few changes to allow them to access the SQL Server database. Keep in mind that if the application references DBTYPE anywhere within the code, you must modify it to work with this database type.

As with all DataServer products, there are two phases to the migration process:

- Phase One—Get the application to work against the new data source.
- Phase Two—Evaluate and modify the application as needed to obtain the desired performance. Changes made to optimize the application for a particular data source might not work, or might not work well, for a different data source. For performance reasons, it is likely that you will need to make specific modifications to an application when adding support for a new data source. The number and scope of the modifications depend on the target performance criteria.

Designing Progress Applications to Access More Than One Data Source

It is common for developers to design their applications so that they can substitute different data sources. This is also known as making the code database-independent. This goal allows the code base to be the same wherever possible for connections to a Progress database, a SQL Server database, or any other data source. For example, one set of source code would be able to access either a Progress database or a SQL Server database. To allow this type of data source substitution requires some planning in the design phase so that you need to maintain only a single code source.

The vast majority of the code in your Progress 4GL application does not require any changes, regardless of the data source. Though, there are a few areas that do require a change. Changes can be categorized into two types: The first type is code that, although it needs to be modified to work with the DataServer, can be changed in a way that allows the same source to run against any data source. The second type of change requires unique source code based on the connected data source. For example only the Progress database supports word indexing, so the SQL Server code will not compile if 4GL code that is specific for word indexing is referenced.

Almost all of the changes will be of the first type that allow for code that will run against any data source. There are coding techniques that are valid for a Progress database that are not valid or optimal for the DataServer environment. However, almost all of the coding techniques that are valid for the DataServer environment are also valid for the Progress database environment. Therefore, when making coding changes you should use options that are valid for both environments whenever possible. All changes should be tested for both environments to confirm compatibility.

In the few cases where a particular code segment cannot be made the same for both environments, you can use preprocessors to designate the data source. Using a preprocessor allows the code for the correct data source to be selected at compile time.

You must set the definition for the preprocessors prior to compile time. One way to simplify setting these types of preprocessors is by putting them into an include file that is included globally in all source code; or, if there are only a few programs that need the preprocessor definitions, you might choose to place the include file only in those programs.

For example a preprocessor might be defined like:

```
&GLOBAL-DEFINE DB-TYPE MSSQLS
```

The preprocessors could then be referenced as:

```
&IF DEFINED ({&MSSQLS}) &THEN
```

Or :

```
&IF NOT DEFINED ({&MSSQLS}) &THEN
```

A benefit to using this method is that it allows you identify all of the DataServer-specific areas in your code quickly.

NOTE: Although you can design and maintain one set of source code for multiple data sources, the compiled code, r-code, is unique to each data source. This means that the source code must be compiled against each data source.

Once you have modified your source code so that it is data source independent, Progress Software Corporation recommends that you implement a testing strategy for all of your supported databases on a regular basis regardless of the type of coding changes you have made.

Performance Optimization

Unfortunately, there is no single solution for performance optimization. In actuality, optimization is the sum of all of the other topics discussed in this white paper. However, there are some areas that have the potential for large gains in performance. For most environments, the areas affected by programming techniques yield the greatest performance benefits. That is not to say that the other areas are not important. You should make sure that your hardware and software configurations are set up properly. Once this is done, the majority of your time for performance optimization should be spent on programming issues.

The search for performance improvement begins by understanding all of the software and hardware components involved in the DataServer environment. For the DataServer for Microsoft SQL Server, this means learning about the DataServer, the Progress 4GL, and SQL Server. See the "Suggested Reading" section for pointers to more information on these topics. Then, review all of the sections in this document. The following section lists a few suggestions on potential performance improvements not covered elsewhere in this white paper.

Reduce Network Traffic & Read and Write Access to the Database

Almost all performance issues can be traced to reading and writing records to the database. The bottleneck in reading and writing these records is the network, the DataServer, or the database engine itself. It may be obvious to many, but it is worth documenting this important point. Any time you reduce the number of reads and writes to the database, you will increase the performance of the application.

There are two ways to reduce reads and writes to the database. First, find unneeded queries and updates of database records. Second, improve how the reads and writes take place.

- **Unneeded queries and updates of database records —** To solve this problem, you must review the types of queries and the available indexes. Some types of queries read many records and then discard all but a few. Modifying the query or adding an index can reduce the number of records that need to be found and read by the database and the number of records that must be sent over the network. Temporary tables provide another method to reduce the network traffic and database reads.
- **Improve how the reads and writes take place —** Use 4GL field lists to return only the needed fields. Using field lists reduces the size of the record that Progress returns to the server. The reduced record size allows more records to be sent over the network in a single packet.

Stored Procedures

For critical sections of code where an increase in performance is necessary, consider customizing the code to take advantage of the native features of the SQL Server database. For example stored procedures are available in SQL Server. The use of stored procedures may be especially important when running programs in batch as the 4GL is designed to enhance applications by providing presentation logic and on-line transaction control, which are absent in most batch routines. Stored procedures are strongly recommended for any area of your software that suffers from database I/O bottlenecks. You can use stored procedures to increase the performance of an application since the code is running on the same machine as the database engine. The DataServer provides an interface that allows you to use stored procedures. Specifically, it provides a way to call stored procedures and to return values from stored procedures. See the *Progress DataServer for Microsoft SQL Server Guide* for more information on implementing stored procedures in a DataServer environment.

Transact SQL

In a few cases you might find a way to write a Transact SQL statement that performs better than a 4GL statement. If this is the case, you can take advantage of the DataServer feature that allows you to submit SQL statements directly to SQL Server. The DataServer supports 4GL syntax for using a send-sql-statement procedure, which sends Transact SQL directly to SQL Server.

Progress AppServer

Another way to increase performance in a distributed-computing environment is to use the Progress AppServer. AppServer goes a step beyond what a stored procedure provides since it provides the flexibility to run business logic on the same system as the database, or on another system. This flexibility allows you to design an application that can take best advantage of the networking and computing environments available. By designing or modifying an application to take advantage of OpenApp Server, you can make the decision of where to run the business logic at deployment time.

When implementing a DataServer solution, the AppServer allows for moving the processing of business logic off of the client machine and onto a high performance computer. This type of design places the logic in an environment that reduces the network traffic to the client and thus boosts performance.

Error Messages

This section lists some error messages that you might encounter when working with the DataServer along with suggested resolutions. Due to the number of potential error messages in each environment and to the wide range of causes that can be at the root of some errors, it is impossible to cover every case. However, by including some of the more common cases in this document you can resolve these problems quickly.

Several different components can generate error messages in your application environment. For example, SQL Server, the DataServer, the Progress Client, the Progress database server, ODBC, or the operating system might generate error messages.

Due to environmental differences, such as specific ODBC drivers, the text of an error message below might not match the exact message in your environment.

Message

A numeric data type overflow. (6174)
22003: [Microsoft][ODBC SQL Server Driver] Numeric value out of range.

Suggested Resolution

A decimal value is too large for a SQL Server field. Either increase the size of the format or reduce the size of the decimal value. If this occurred while loading a data (.d) file see the error (.e) file for more information.

Message

Non-PROGRESS database <database name> is unknown. (1008)

Suggested Resolution

The logical name of the database is not set correctly, or the logical (–ld) database name is incorrect. Correct the logical name specified for the database.

Message

IM002:[Microsoft][ODBC Driver Manager] Data source name not found and no default driver specified.

Suggested Resolution

The data source name is not defined on the computer. Verify that the Service Name (–S) parameter is set correctly on both the client and server computers. Also, verify that the data source is configured properly using the ODBC Data Source Administrator. If test tools are available, you should always test a direct connection to your data source before connecting through the DataServer.

Suggested Reading

You might decide that you need additional or more in-depth information on one or more of the topics covered in this white paper. This section includes other recommended resources for learning about these topics. However, remember that the first place to start when looking into issues regarding the Progress DataServer for Microsoft SQL Server is the *Progress DataServer for Microsoft SQL Server Guide*.

The *Progress DataServer for Microsoft SQL Server Guide* is on the Progress Electronic Documentation CD and is also available at http://www.progress.com/v9/documentation/index.htm. Consult the list of additional Progress manuals in the *Progress DataServer for Microsoft SQL Server Guide*.

Progress Technical Support Knowledge Base

This Progress reference site is available at http://www.progress.com/support/kb.htm. This site is updated regularly with new information.

Microsoft SQL Server Home Page

This Microsoft reference site is available at http://www.microsoft.com/sql. This site is updated regularly with new information.

MSDN Online SQL Server Developer Center

This Microsoft reference site is available at http://msdn.microsoft.com/sqlserver/default.asp. This site is updated regularly with new knowledge.

Microsoft SQL Server 7.0 Performance Tuning Technical Reference

Written by Steve Adrien DeLuca, et al. (Microsoft Press, 1999), this guide contains detailed information from the basics to advanced topics of performance tuning that is specific to Microsoft SQL Server 7.0.

Inside Microsoft SQL Server 7.0

Written by Ron Soukup and Kalen Delaney (Microsoft Press, 1999), this guide is a appropriate for someone new to Microsoft SQL Server. It provides information on understanding what makes SQL Server tick.

Microsoft SQL Server 7.0 Performance Tuning Guide

Written by Henry Lau (Microsoft Web Site http://msdn.microsoft.com/library/techart/msdn_sql7perftune.htm , 1998), this guide provides information on tuning Microsoft SQL Server.

Microsoft SQL Server 7.0 Books Online (BOL)

This online reference ships with Microsoft SQL Server as the primary source of information on Microsoft SQL Server.

Corporate Headquarters
Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA Tel: 781 280 4000 Fax: 781 280 4095

Europe/Middle East/Africa Headquarters
Progress Software Europe B.V. Schorpioenstraat 67 3067 GG Rotterdam, The Netherlands Tel: 31 10 286 5700 Fax: 31 10 286 5777

Latin American Headquarters
Progress Software Corporation, 2255 Glades Road, One Boca Place, Suite 300 E, Boca Raton, FL 33431 USA Tel: 561 998 2244 Fax: 561 998 1573

Asia/Pacific Headquarters
Progress Software Pty. Ltd., 1911 Malvern Road, Malvern East, 3145, Australia Tel: 61 39 885 0544 Fax: 61 39 885 9473

PROGRESS SOFTWARE

www.progress.com