Improving Performance

This document presents information on how to improve the performance of your system. In general, techniques for improving performance are designed to reduce unnecessary processing, network traffic, and blocking. The techniques minimize:

- The number of unneeded records stored in tables
- The number of records retrieved in queries
- The number of locks on records in queries
- The duration of locks on records
- The size and duration of transactions
- The fragmentation of tables and indexes.

For information on customization and performance, see the document *Modifying Infor ERP SL 8*.

Hardware

You must have appropriate hardware to meet the demand put on your system. Refer to the *Guide to Technology* here (http://www.infor365.com) for minimum requirements.

- **Server usage** - Our recommended server usage is detailed in the Introduction chapter of the *Infor ERP SL Installation Guide*.

- **Transaction log drives vs. data drives** - Use separate physical drives for the data and log files. Because transaction logs are written sequentially, they require fewer dedicated drives than do data files. The number of physical drives, capacity, and performance are more important for the data drives than for the transaction log drives.

SQL Server Settings

SQL requirements are listed at the beginning of each chapter in the *Infor ERP SyteLine System Administration Guide* where appropriate. This section includes some settings that can be used for a performance improvement. Refer to SQL documentation for more information.

- **Auto Shrink** - On the application database machine, always have *Auto shrink* disabled for all databases. If it is disabled, your system will not show significant performance loss related to shrinking the database. If it is enabled, SQL Server checks every 30 minutes to see if it needs to shrink the database; this can cause a huge performance hit. You can use the DBCC SHRINKDATABASE or DBCC SHRINKFILE commands when you need to shrink databases, or you can use the SQL Server Agent to schedule regular file-shrinking instead of enabling *Auto shrink*.

- **Auto update statistics** - We recommend that you enable *Auto update statistics* for all databases. This feature is enabled by default. With this feature enabled, SQL Server updates the statistics of an index based on the following criteria:

  - If the number of rows in a table is greater than 6, but 500 or less, statistics are updated when there have been 500 modifications made, OR
• If the number of rows in the table is greater than 500, updates are made when 500 plus 20% of the number of rows in the table have been modified.

When a SQL Server database is under very heavy load, this feature can update the statistics during busy times, causing a performance issue. If you find that enabling the feature causes more problems than it solves, you can turn it off, and then manually update the statistics when the database is under a less heavy load.

We recommend that you both enable Auto update statistics and update statistics manually. See “Update Statistics” on page 5.

• Tempdb - Set the original size of the tempdb database files to a reasonable size (about the size of the ledger table) to prevent the files from automatically expanding as more space is needed. If the tempdb database expands too frequently, performance can be affected. Set the file-growth increment percentage to a reasonable size (10% is a good choice) to avoid the tempdb database files from growing by too small a value. If the file growth is too small compared to the amount of data being written to the tempdb database, then tempdb may need to constantly expand, thereby affecting performance. Place the tempdb database on a fast I/O subsystem to ensure good performance. Stripe the tempdb database across multiple disks for better performance. Use filegroups to place the tempdb database on disks different from those used by user databases.

• Minimum server memory and Maximum server memory - Set these values based on the size and activity of your instance of SQL Server.

• MAXDOP - Set the max degree of parallelism option to 8 or less by using sp_configure.

Unneeded Data

Unneeded data in tables with a large number of records can increase query time and slow certain processes. Infor ERP SL provides utilities for reducing unneeded data. SQL Server system stored procedures aid in understanding table size.

Purge or Compress Unneeded Data

The following forms allow you to purge or compress data to improve performance. Determining when to use these forms is primarily a business decision you need to make. We’ve made recommendations for some listed below. Refer to the online Help for information on how to use the forms.

It is important to formulate a data retention plan for each area. You should decide how long to retain data and who will purge or compress records that are older than the planned retention period.

• Compress General Ledger Transactions
• Delete Material Transactions - Do this as part of year-end procedures.
• Delete Job Transactions
• Delete A/P Posted Transactions
• Activate/Deactivate Posted Transactions - A/P
• Delete A/R Posted Transactions
• Activate/Deactivate Posted Transactions - A/R
• Delete Audit Logs
• **Audit Log Types** - Look at the types you have on this form and verify that you need all the ones you have created. Types 1 and 2 are standard default types, and you cannot delete them. All other types (10,000 and above) are custom types created by you. You can delete these types.

**Examine Table Size**

The SQL Server system stored procedure `sp_spaceused` reports information about a table that can be useful in forming and implementing a data retention plan. The stored procedure shows:

- Number of rows in a table
- Space reserved for a table
- Space used by data in a table
- Space used by the index in a table
- Unused space in a table

In SQL Server Management Studio, with the Infor ERP SL application database selected as the current database, use the following syntax to generate a report on a table:

```sql
EXEC sp_spaceused table_name
```

Example:

```sql
EXEC sp_spaceused ledger
```

To report on tables that are likely to need attention in a data retention plan, you can use the following script:

```sql
-- Ledger
EXEC sp_spaceused ledger
EXEC sp_spaceused ledger_all

-- Material Transaction
EXEC sp_spaceused matltran
EXEC sp_spaceused matltran_all
EXEC sp_spaceused matltran_amt
EXEC sp_spaceused matltran_amt_all

-- Job Transactions
EXEC sp_spaceused jobtran

-- AR Transactions
EXEC sp_spaceused artran
EXEC sp_spaceused artran_all

-- AP Transactions
EXEC sp_spaceused aptrx
EXEC sp_spaceused aptrx_all

-- Audit Logs
EXEC sp_spaceused AuditLog
```

To report on all tables in the database, you can use this script:

```sql
DECLARE @table_name sysname
DECLARE Tables_Cursor CURSOR FOR
SELECT name
```
FROM sysobjects
WHERE type = 'U' ORDER BY 1

OPEN Tables_Cursor

FETCH NEXT FROM Tables_Cursor
INTO @table_name
WHILE @@FETCH_STATUS = 0
BEGIN
    EXEC sp_spaceused @table_name
    FETCH NEXT FROM Tables_Cursor
    INTO @table_name
END

CLOSE Tables_Cursor
DEALLOCATE Tables_Cursor

To select tables with similar names, modify the WHERE clause in the script. For example, to report on only tables with the _all suffix, replace

WHERE type = 'U' ORDER BY 1

with

WHERE type = 'U' AND name LIKE '%[_]all' ORDER BY 1

See the Help for SQL Server for more information about sp_spaceused.

**Filter Inactive Records in Data Integration**

Data integration between this system and other products may require the transfer of a large number of records with each update. Some records that are maintained in this system are not required in integration with these programs. You can improve performance by excluding them.

You can modify a product to filter out specified customer, vendor, and item records. To specify a record to be filtered, clear the field *Active for Data Integration* for the record in the Customers, Vendors, or Items form. By default, the field is selected for each record, and the record is replicated and synchronized with the other products.

**SQL Server Maintenance**

SQL Server statistics that are out of date and tables and indexes that are significantly fragmented adversely affect system performance. You can monitor their condition and take steps to enhance their performance.

**Statistical Information**

SQL Server uses statistical information about the distribution of values in a column to determine the optimal strategy for evaluating a query. Distribution statistics help the system estimate how efficient an index would be in retrieving data associated with a key value or range specified in the query.

As the data in a column changes, index and column statistics can become out-of-date, affecting query performance. The statistics should be refreshed anytime significant numbers of changes to keys occur in the index.

We recommend that you update statistics nightly or weekly for best performance (see “Update Statistics” on page 5).
You can use the `dbcc show_statistics` statement to generate a report on the distribution statistics for an index. The statements in this section use the following syntax:

```
dbcc show_statistics (table_name, index_name)
```

In SQL Query Analyzer, with the application database selected as the current database, the following statements show the current statistics and the last time statistics were updated for primary keys in major tables:

```
dbcc show_statistics (item, pk_item)
dbcc show_statistics (customer, pk_customer)
dbcc show_statistics (ledger, pk_ledger)
dbcc show_statistics (matltran, pk_matltran)
dbcc show_statistics (matltran_amt, pk_matltran_amt)
dbcc show_statistics (journal, pk_journal)
dbcc show_statistics (ledger_all, pk_ledger_all)
```

The results indicate the selectivity of an index (the lower the density returned, the higher the selectivity) and provide the basis for determining whether an index is useful in optimizing queries.

See SQL Server Help for `dbcc show_statistics` and other DBCC (Database Console Commands) statements.

**Update Statistics**

Use the Transact-SQL statement `UPDATE STATISTICS` if

- A process suddenly takes much longer than usual to run
- There is a significant change in the key values in an index
- A large amount of data in an indexed column has been added, changed, or removed, or the table has been truncated using the TRUNCATE TABLE statement and then repopulated.

We recommend that you update statistics nightly or weekly.

This example updates the statistics for all indexes on the customer table.

```
UPDATE STATISTICS customer
```

To update statistics for all tables in the in the current database, you can run the SQL Server stored procedure `sp_updatestats`, which uses `UPDATE STATISTICS`:

```
EXEC sp_updatestats
```

For more information, see SQL Server Help for `UPDATE STATISTICS` and `sp_updatestats`.

**Fragmentation Information**

Fragmentation occurs through data modifications (INSERT, UPDATE, and DELETE). For queries that scan part or all of a table, this fragmentation can cause additional pages to be read, adversely affecting performance.

You can use the Transact-SQL `DBCC SHOWCONTIG` statement to display fragmentation information for the data and indexes of a specified table.

To determine whether a table is heavily fragmented, use the following syntax in SQL Query Analyzer, with the application database selected as the current database:

```
DBCC SHOWCONTIG (table_name)
```
In the result set, the value of **Logical Scan Fragmentation** gives an indication of the table's fragmentation level. The value should be close to zero, although a value from 0% through 10% may be acceptable.

To show in a grid an abbreviated result set for every index on every table, use:

```
DBCC SHOWCONTIG WITH TABLERESULTS, FAST
```

To show the full result set for every index on every table, use:

```
DBCC SHOWCONTIG WITH TABLERESULTS, ALL_INDEXES
```

For more information, see SQL Server Help for **DBCC SHOWCONTIG**.

**Defragment Indexes**

We recommend that you rebuild your table indexes on a weekly basis if possible.

The **Transact-SQL DBCC INDEXDEFRAG** statement defragments indexes of a specified table, improving index-scanning performance.

```
DBCC INDEXDEFRAG (database_name, table_name, index_name)
```

The script below uses **DBCC INDEXDEFRAG** and **DBCC SHOWCONTIG** to defragment all indexes in a database fragmented above a declared threshold of 30 percent. The script is from Microsoft’s **Transact-SQL Reference**, copyright © 2004 Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 U.S.A.; all rights reserved.

Note that you must specify a database before you run the script.

```sql
/*Perform a 'USE <database name>' to select the database in which to run the script.*/
-- Declare variables
SET NOCOUNT ON
DECLARE @tablename VARCHAR (128)
DECLARE @execstr   VARCHAR (255)
DECLARE @objectid  INT
DECLARE @indexid   INT
DECLARE @frag      DECIMAL
DECLARE @maxfrag   DECIMAL

-- Decide on the maximum fragmentation to allow
SELECT @maxfrag = 30.0

-- Declare cursor
DECLARE tables CURSOR FOR
    SELECT TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_TYPE = 'BASE TABLE'

-- Create the table
CREATE TABLE #fraglist (ObjectName CHAR (255), ObjectId INT, IndexName CHAR (255), IndexId INT, Lvl INT, CountPages INT, CountRows INT, MinRecSize INT,
MaxRecSize INT,
AvgRecSize INT,
ForRecCount INT,
Extents INT,
ExtentSwitches INT,
AvgFreeBytes INT,
AvgPageDensity INT,
ScanDensity DECIMAL,
BestCount INT,
ActualCount INT,
LogicalFrag DECIMAL,
ExtentFrag DECIMAL)

-- Open the cursor
OPEN tables

-- Loop through all the tables in the database
FETCH NEXT
FROM tables
INTO @tablename

WHILE @@FETCH_STATUS = 0
BEGIN
-- Do the showcontig of all indexes of the table
INSERT INTO #fraglist
EXEC ('DBCC SHOWCONTIG (''' + @tablename + ''')
        WITH FAST, TABLERESULTS, ALL_INDEXES, NO_INFOMSGS')

FETCH NEXT
FROM tables
INTO @tablename
END

-- Close and deallocate the cursor
CLOSE tables
DEALLOCATE tables

-- Declare cursor for list of indexes to be defragged
DECLARE indexes CURSOR FOR
    SELECT ObjectName, ObjectId, IndexId, LogicalFrag
    FROM #fraglist
    WHERE LogicalFrag >= @maxfrag
        AND INDEXPROPERTY (ObjectId, IndexName, 'IndexDepth') > 0

-- Open the cursor
OPEN indexes

-- loop through the indexes
FETCH NEXT
FROM indexes
INTO @tablename, @objectid, @indexid, @frag

WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Executing DBCC INDEXDEFRAG (0, ' + RTRIM(@tablename) + ',

    EXEC (''DBCC INDEXDEFRAG (0, '' + RTRIM(@tablename) + '))
END
' + RTRIM(@indexid) + ') - fragmentation currently ' + RTRIM(CONVERT(varchar(15),@frag)) + '%'
SELECT @execstr = 'DBCC INDEXDEFRAG (0, ' + RTRIM(@objectid) + ',
    ' + RTRIM(@indexid) + ')
EXEC (@execstr)
FETCH NEXT
    FROM indexes
    INTO @tablename, @objectid, @indexid, @frag
END

-- Close and deallocate the cursor
CLOSE indexes
DEALLOCATE indexes

-- Delete the temporary table
DROP TABLE #fraglist
GO

Customizations

Customizations to Infor ERP SL should be evaluated for performance along with standard product components. You should ensure that indexes for new tables are designed correctly and maintained adequately. If a custom feature performs slower than when it was first implemented, determine whether unneeded records are causing the performance reduction.

Custom reports and processes should be evaluated to see if they are reading the least number of records. BI queries should be similarly evaluated for efficiency.

User Actions

End users’ practices can slow Infor ERP SL performance. Actions such as querying an unlimited number of records into a form, specifying overly broad query criteria in reports, and running unneeded reports increase network traffic and can tax database resources.

Reduce the Number of Rows Returned in Queries

In WinStudio, users can choose to retrieve all rows or any specified maximum number of rows in queries. This option overrides a default limit set on queries that return data records and items in drop-down lists. However, unlimited queries can degrade system performance or exceed the resources of the utility server and the client machine.

The process default WinStudio Max Record Cap allows you to set a systemwide limit on the number of records or drop-down list items that users can query into forms. The limit overrides any setting made by the user in WinStudio. See the Help for the Process Defaults form.

Reduce the Scope of Reports

Report users should be sure to set criteria in a such way that the system returns only the information needed for the purposes of the report. Users should limit the range of time frames and other criteria to prevent needless processing.

Users should avoid running unnecessary reports.
Replication

**Configure Multi-site Replication with a Master Site and Shared Tables**

Set up an intranet with a master site and share certain _all tables. This allows other sites to use views into shared _all tables on the master site, reducing replication traffic between the sites. Advantages and requirements are described in the *Multi-Site Planning Guide*. The process is described in the *Multi-Site Implementation Guide*.

Locking and Blocking

*Locking* prevents users from reading data being changed by other users, and prevents multiple users from changing the same data at the same time. If locking is not used, data within the database may become logically incorrect, and queries executed against that data may produce unexpected results. SQL Server enforces locking automatically. Locking can occur at record, page, or table level.

*Blocking* occurs when one user holds a lock and a second user requires a conflicting lock type. This forces the second user to wait, blocked by the first. Typically, the second user sees an hourglass while trying to process or save records. Most blocking problems happen because a single process holds locks for an extended period of time, causing a chain of blocked processes. A design goal is to minimize the amount of time a record is locked to reduce the potential blocking of another user.

A *deadlock* arises when two processes have data locked, and each process cannot release its lock until the other process has released its lock. SQL Server rolls back one of the transactions and then allows the other transaction to continue.

**Monitor Blocking**

The utility *SyteLine SQL Performance Log* allows you to monitor blocking and to log the results. Download the utility from the support site http://www.infor365.com/. Navigate to the Infor ERP SL Knowledgebase and select the Solution By Solution ID option. The Solution ID is 2003SyteLine33733.

**Save Each Modified Row in a Separate Transaction**

You can specify that WinStudio save each modified row in a separate transaction. By default, without this setting, all modified rows are sent to the mid-tier to be processed within a single transaction.

Saving one row per transaction can alleviate blocking problems in some forms. Whether the setting enhances performance depends on the complexity of a form’s save operation and the speed of the network connection. Forms that require highly complex saves may benefit from the setting, especially if connection speed is adequate. Forms that require simple save operations are less likely to benefit, especially if the additional network traffic required in saving one row at a time slows the system.

In default WinStudio behavior, if one row fails, the entire transaction is rolled back. When you save each row in a separate transaction and a row fails, all previous rows remain committed.

If your form design requires that all modified rows be committed as a unit, with processing on the unit before and/or after a save operation, saving rows in separate transactions may not be appropriate. If an error occurs on a row, some rows may be committed and others not committed.

The following forms save rows in separate transactions:
• Job Orders
• Customer
• Vendor
• Item
• Job Operations
• Job Materials
• Current Operations
• Current Materials
• Purchase Orders
• Purchase Order Lines
• Customer Orders
• Customer Order Lines

For instructions on setting this feature, see "Save One Row Per Transaction" in the Help for WinStudio edit mode.

**Set the Collection Read Mode (Transaction Isolation Level)**

You can specify whether form queries read committed or uncommitted data by setting the **Collection Read Mode**. The setting applies to queries that load primary collections, secondary collections, and lists, and to in-collection validations. It also applies to background-task queries that generate reports and to background-task stored procedures. The setting does not affect SQL SELECT operations or other processing coded in stored procedure (method) calls. The default **Collection Read Mode**, UNCOMMITTED, corresponds to the Transact-SQL statement SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED.

With the UNCOMMITTED setting, which allows the reading of uncommitted data, users do not have to wait for other long-running transactions that access the same dataset to complete before their queries can complete.

With the COMMITTED setting, a query reads committed data and returns only data for which the query can get a shared lock.

The base, systemwide transaction isolation level is set on the **Process Defaults** form. Note that if this setting is unsuitable for all forms and tasks, you can override it for selected forms and tasks. You can set the isolation level for individual reports and stored procedures on the **Background Task Definitions** form. You can also override the system setting at the form level in WinStudio edit mode. See "Set the Read Mode for a Collection" in the Help for Infor ERP SL.

**Prevent Locking of the Journal Table During Mass Journal Posting**

Posting a large number of transactions with the **Mass Journal Posting** form can set an exclusive lock on the journal table. This prevents users from inserting data into the table until the posting is complete.

The process default **Journal Posting - No Table Locks** allows you to override this behavior. The value 1 prevents an exclusive table lock from being taken during mass journal posting. The value 0 does not prevent a lock. The setting applies to both forms and the background task that runs journal posting without middleware. See the Help for the **Process Defaults** form for more information.
Prevent Blocking of Other Processes When Rolling Current Costs to Standard Costs

Normally, the Roll Current Cost to Standard Cost utility processes all current costing data at one time. When processing large amounts of data, this can block other processes, such as adding jobs, adding CO lines, using the Purchase Order Receiving form, or opening the Customer Order Lines form.

The process default Roll Current to Standard - No Table Locks allows you to override this behavior. Changing this setting from 0 (the default) to 1 can eliminate this blocking. You should understand, however, that selecting this option can also slow down the Roll Current Cost to Standard Cost utility processing.

Prevent Deadlocks on the Item Table During Certain Operations

Deadlocks on the item table can occur during certain operations involving bills of materials that contain many items. The Lock Job Items process default determines whether job items are locked during these operations. With the value 1, operations such as releasing a job, which copies the bill of materials, and posting a job will lock all item records in blocks according to the operation number. The default value, 0, does not lock item records. See the Help for the Process Defaults form.

Avoid Long Delays from Deadlocks

A high value for the process default Number of Deadlock Retry Attempts can cause users to experience excessive delays from deadlocks. The value 3 is a recommended starting point. See the Help for the Process Defaults form.

Windows Tools

- Memory - Use Perfmon to determine memory usage and to determine if expansion is needed. If additional memory is used as AWE memory, verify in the SQL Server error log that the statement "Address Windowing Extensions enabled" exists.

SQL Server Stored Procedures and Commands

The items in this section can be used to return information related to performance.

- sp_who and sp_who2 - The stored procedure sp_who shows what SPID is blocked; sp_who2 shows who is blocking.
- sp_helpindex (table_name) - Gives index information on a table.
- DBCC OPENTRAN - Determines whether an open transaction exists within the log.
- DBCC INPUTBUFFER (SPID) - Displays the last statement sent from a client to SQL Server.
- DBCC Trace On - Enables specified trace flags.