Microsoft Business Solutions–Navision SQL Server Option Resource Kit

White Paper

Published: April 2005

Contents

Introduction

This document describes several considerations that Navision consultants must bear in mind when they are implementing the Microsoft SQL Server Option for Navision.

This resource kit is designed for use with Navision 3.70 and SQL Server 2000, even though some of the articles are based on previous versions of the products.

The document is divided into several sections that correspond to the different phases of implementing the SQL Server Option for Navision:

- Planning
- Design
- Implementation
- Maintenance
- Troubleshooting

Planning

One of the first things that a consultant must consider is the system requirements that the customer's installation must conform with. For a complete list of the system requirements for Microsoft Business Solutions–Navision, see http://www.microsoft.com/sql/evaluation/overview/default.asp

Maximum Capacity Specifications for SQL Server

The following table specifies the maximum sizes and numbers of various objects defined in Microsoft SQL Server databases or referenced in Transact-SQL statements.

Object	SQL Server 2000
Batch size	65,536 * Network Packet Size[1]
Bytes per short string column	8,000
Bytes per **text**, **ntext**, or **image** column	2 GB-2
Bytes per GROUP BY, ORDER BY	
Bytes per index	900[2]
Bytes per foreign key	900
Bytes per primary key	900
Bytes per row	8,060
Bytes in source text of a stored procedure	Lesser of batch size or 250 MB
Clustered indexes per table	1
Columns in GROUP BY, ORDER BY	
Columns or expressions in a GROUP BY WITH CUBE or WITH ROLLUP statement	
Columns per index	16
Columns per foreign key	16
Columns per primary key	16
Columns per base table	1,024
Columns per SELECT statement	4,096
Columns per INSERT statement	1,024
Connections per client	Maximum value of configured connections
Database size	1,048,516 TB[3]
Databases per instance of SQL Server	32,767
Filegroups per database	256
Files per database	32,767
File size (data)	32 TB
File size (log)	32 TB
Foreign key table references per table	253
Identifier length (in characters)	128
Instances per computer	16
Length of a string containing SQL statements (batch size)	65,536 * Network packet size1
Locks per connection	Max. locks per server

Locks per instance of SQL Server	2,147,483,647 (static) 40% of SQL Server memory (dynamic)
Nested stored procedure levels	32
Nested sub-queries	32
Nested trigger levels	32
Non-clustered indexes per table	249
Objects concurrently open in an instance of SQL Server[4]	2,147,483,647 (or available memory)
Objects in a database	2,147,483,647[4]
Parameters per stored procedure	1,024
REFERENCES per table	253
Rows per table	Limited by available storage
Tables per database	Limited by number of objects in a database[4]
Tables per SELECT statement	256
Triggers per table	Limited by number of objects in a database[4]
UNIQUE indexes or constraints per table	249 non-clustered and 1 clustered

[1] Network Packet Size is the size of the tabular data scheme (TDS) packets used to communicate between applications and the relational database engine. The default packet size is 4 KB and is controlled by the network packet size configuration option.

2The maximum number of bytes in any key cannot exceed 900 in SQL Server 2000. You can define a key using variable-length columns whose maximum sizes add up to more than 900, provided no row is ever inserted with more than 900 bytes of data in those columns. For more information, see Maximum Size of Index Keys.

3 The size of a database cannot exceed 2 GB when using the SQL Server 2000 Desktop Engine or the Microsoft Data Engine (MSDE) 1.0.

4 Database objects include all tables, views, stored procedures, extended stored procedures, triggers, rules, defaults, and constraints. The sum of the number of all these objects in a database cannot exceed 2,147,483,647.

Maximum Number of Processors Supported by SQL

The following table lists the number of processors that the database engine in each SQL Server 2000 edition can support on symmetric multiprocessing (SMP) computers:

Operating System	Enterprise Edition	Standard Edition	Personal Edition	Developer Edition	Desktop Engine	SQL Server CE	Enterprise Evaluation Edition
Windows 2003 Server	32	4	2	32	2	1	32
Microsoft Windows 2000 DataCenter	32	4	2	32	2	N/A	32
Windows 2000 Advanced Server	8	4	2	8	2	N/A	8
Windows 2000 Server	4	4	2	4	2	N/A	4
Windows 2000 Professional	N/A	N/A	2	2	2	N/A	2

Maximum Physical Memory Supported by the Editions of SQL Server 2000

This table shows the maximum physical memory, or RAM, that the database engine in each SQL Server 2000 edition can support:

Operating System	Enterprise Edition	Standard Edition	Personal Edition	Developer Edition	Desktop Engine	SQL Server CE	Enterprise Evaluation Edition
Windows 2003	64 GB	2 GB	2 GB	64 GB	2 GB	N/A	64 GB
Windows 2000 DataCenter	64 GB	2 GB	2 GB	64 GB	2 GB	N/A	64 GB
Windows 2000 Advanced Server	8 GB	2 GB	2 GB	8 GB	2 GB	N/A	8 GB
Windows 2000 Server	4 GB	2 GB	2 GB	4 GB	2 GB	N/A	4 GB
Windows 2000 Professional	N/A	N/A	2 GB	2 GB	2 GB	N/A	2 GB

Design

The next phase in rolling out a customer's installation is to design the application so that it meets that customer's particular needs.

There are many resources that the Navision consultant can access to learn how to design a Navision application for the SQL Server Option. These include:

- *The Application Designer's Guide* available on the product CD. This manual contains all the information you need to create all of the objects that Navision supports. For the SQL Server Option the topics covered include: data types, SIFT, numbers & sorting, locking, record level security, performance and the NDBCS Driver.
- *Installation & System Management: Microsoft SQL Server Option* available on the product CD. This manual tells you how to install and implement the SQL Server Option. Topics covered include: Installation, database creation, backups and restore, database testing.
- *C/SIDE Reference Guide Online Help*. An online Help project installed with the client that explains all of the functions supported by the C/AL Programming language.

Using Find Statements

The Navision application has traditionally used two different Find statements to locate records in tables. These are `FIND('-')` and `FIND('+').`

These functions are used to:

- Find either the first or last record in a table or set.
- Check whether or not there are any records in the table.
- Loop through the records in the table or set.

These functions work very well on Navision Database Server because it uses the ISAM (**I**ndexed **S**equential **A**ccess **M**ethod) disk access method and therefore reads records individually.

These `FIND` statements can also be used to return a set of records and to modify a key in the index that the loop is based on. While Navision Database Server has no difficulty with these 'ambiguous' statements, it is exactly this ambiguity that makes it very difficult for Navision to issue precise SQL statements. This means that SQL Server does not always interpret these functions correctly and tends to use set based queries to return small sets of records rather than just returning the single record that was requested. This is generally the case when the `WHILE FIND('-')` method is used instead of the `REPEAT UNTIL NEXT` method.

This is an extremely inefficient way to use SQL Server. SQL Server prefers more precise statements that enable it to reuse cursors more frequently.

Navision now contains more unambiguous functions that should be used to perform these tasks:

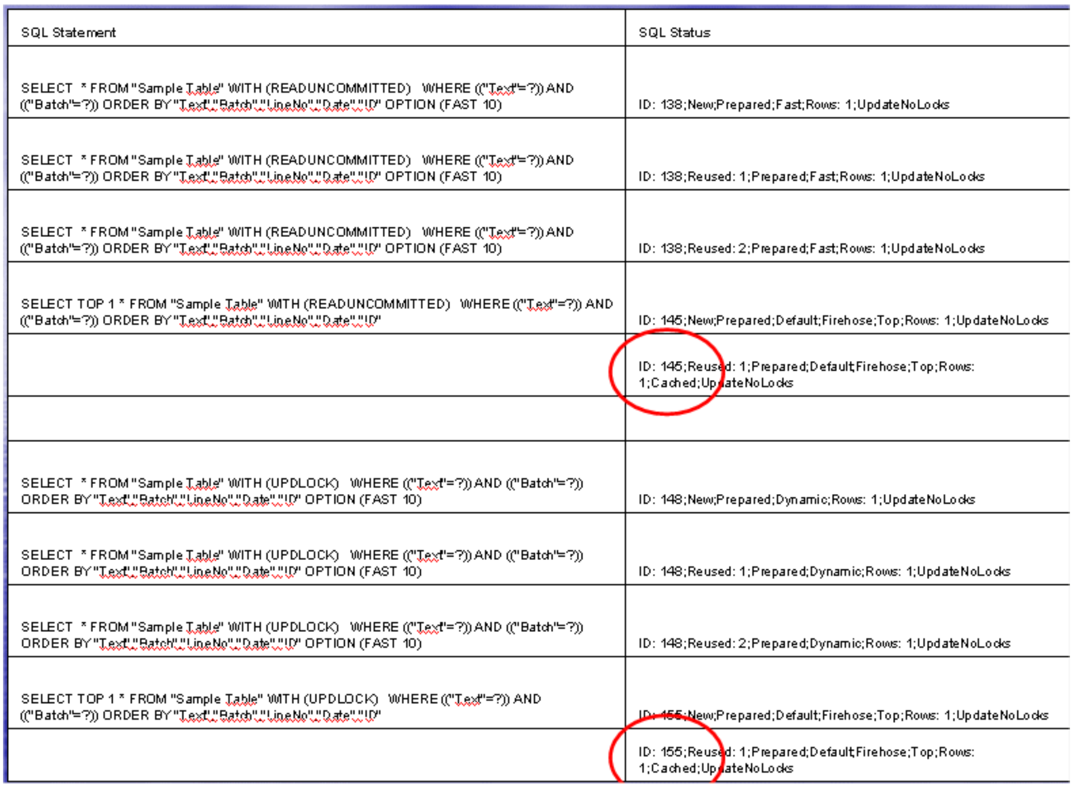
- `FINDFIRST`
- `FINDLAST`

- `FINDSET`

These functions perform the same tasks in a much more efficient manner and don’t overload the server with unnecessary cursors. These functions result in fewer server calls being made, a simpler execution plan and the more efficient reuse of cursors.

`FINDFIRST` and `FINDLAST`

The following excerpts from the Navision Client Monitor illustrate how server calls have been interpreted by SQL Server in earlier versions of Navision:



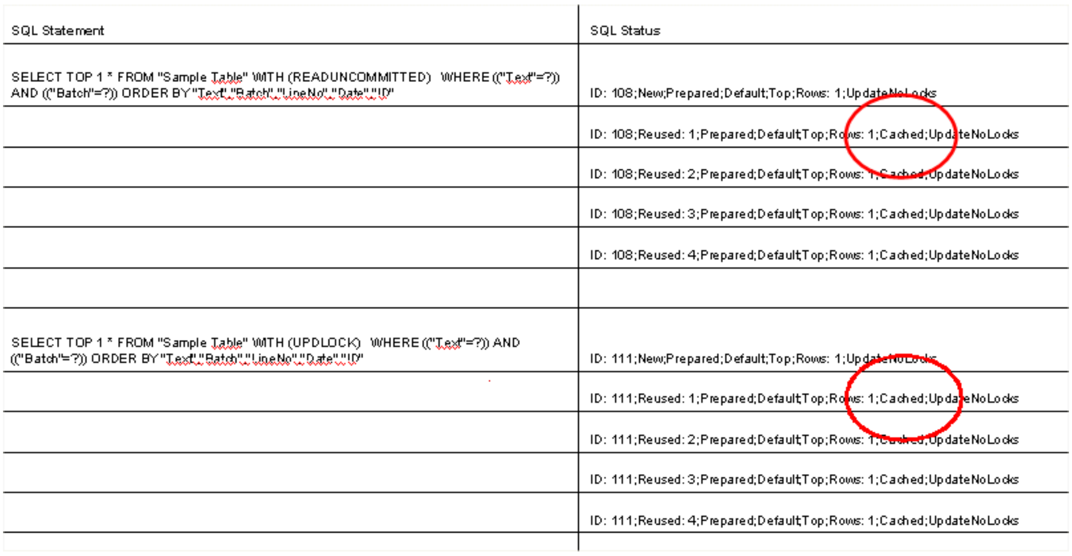
SQL Statement	SQL Status
SELECT * FROM "Sample Table" WITH (READUNCOMMITTED) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID" OPTION (FAST 10)	ID: 138;New;Prepared;Fast;Rows: 1;UpdateNoLocks
SELECT * FROM "Sample Table" WITH (READUNCOMMITTED) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID" OPTION (FAST 10)	ID: 138;Reused: 1;Prepared;Fast;Rows: 1;UpdateNoLocks
SELECT * FROM "Sample Table" WITH (READUNCOMMITTED) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID" OPTION (FAST 10)	ID: 138;Reused: 2;Prepared;Fast;Rows: 1;UpdateNoLocks
SELECT TOP 1 * FROM "Sample Table" WITH (READUNCOMMITTED) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID"	ID: 145;New;Prepared;Default;Firehose;Top;Rows: 1;UpdateNoLocks
	ID: 145;Reused: 1;Prepared;Default;Firehose;Top;Rows: 1;Cached;UpdateNoLocks
SELECT * FROM "Sample Table" WITH (UPDLOCK) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID" OPTION (FAST 10)	ID: 148;New;Prepared;Dynamic;Rows: 1;UpdateNoLocks
SELECT * FROM "Sample Table" WITH (UPDLOCK) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID" OPTION (FAST 10)	ID: 148;Reused: 1;Prepared;Dynamic;Rows: 1;UpdateNoLocks
SELECT * FROM "Sample Table" WITH (UPDLOCK) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID" OPTION (FAST 10)	ID: 148;Reused: 2;Prepared;Dynamic;Rows: 1;UpdateNoLocks
SELECT TOP 1 * FROM "Sample Table" WITH (UPDLOCK) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID"	ID: 155;New;Prepared;Default;Firehose;Top;Rows: 1;UpdateNoLocks
	ID: 155;Reused: 1;Prepared;Default;Firehose;Top;Rows: 1;Cached;UpdateNoLocks

Using FIND(‘-’)

In the first section, `FIND(‘-’)` is used without locking (`READUNCOMMITTED`) for 5 iterations. As you can see, it took four iterations before the Navision SQL driver `NDBCS.dll` realized that what we really wanted it to do was find the first record. After it generates the `SELECT TOP 1` statement, the record is cached and no more server calls are made.

In the second section, `FIND(‘-’)` is used with locking (`UPDLOCK`) and the story is the same. It took four iterations before the Navision SQL driver realized that what we really wanted it to do was find the first record.

The following excerpts illustrate how the new `FINDFIRST` function is interpreted more efficiently by SQL Server:



SQL Statement	SQL Status
SELECT TOP 1 * FROM "Sample Table" WITH (READUNCOMMITTED) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID"	ID: 108;New;Prepared;Default;Top;Rows: 1;UpdateNoLocks
	ID: 108;Reused: 1;Prepared;Default;Top;Rows: 1;Cached;UpdateNoLocks
	ID: 108;Reused: 2;Prepared;Default;Top;Rows: 1;Cached;UpdateNoLocks
	ID: 108;Reused: 3;Prepared;Default;Top;Rows: 1;Cached;UpdateNoLocks
	ID: 108;Reused: 4;Prepared;Default;Top;Rows: 1;Cached;UpdateNoLocks
SELECT TOP 1 * FROM "Sample Table" WITH (UPDLOCK) WHERE (("Text"=?)) AND (("Batch"=?)) ORDER BY "Text","Batch","LineNo","Date","ID"	ID: 111;New;Prepared;Default;Top;Rows: 1;UpdateNoLocks
	ID: 111;Reused: 1;Prepared;Default;Top;Rows: 1;Cached;UpdateNoLocks
	ID: 111;Reused: 2;Prepared;Default;Top;Rows: 1;Cached;UpdateNoLocks
	ID: 111;Reused: 3;Prepared;Default;Top;Rows: 1;Cached;UpdateNoLocks
	ID: 111;Reused: 4;Prepared;Default;Top;Rows: 1;Cached;UpdateNoLocks

Using FINDFIRST

In the first section, `FINDFIRST` is used without locking for 5 iterations (`READUNCOMMITTED`). As you can see, the Navision SQL driver was able to issue the correct call immediately. After the record is cached, no more SQL statements were generated and no more server calls were made.

Similarly, in the second section when `FINDFIRST` is used with locking (`UPDLOCK`) the correct statement is issued.

The effect is the same when you use `FINDLAST`; a more precise SQL statement is sent to the server, fewer server calls are made, a simpler 'SQL plan' is generated and cursors are avoided.

Looping with `FINDSET`

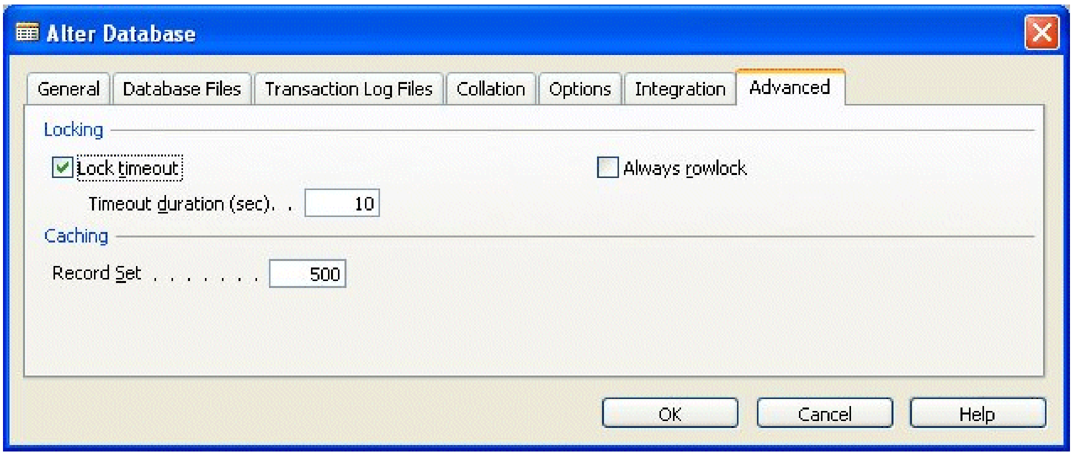
The `FINDSET` function is designed to make searching through sets of records more efficient. This function retrieves a set of records from the table and loops through them.

The `FINDSET` function optimizes loops by:

- Generating more explicit code on SQL Server. This means that the code used in the Navision SQL Server Option must use this function explicitly to ensure that the driver generates optimal statements.
- Reusing cursors more efficiently or avoiding them totally.

This function works alongside the database property *Record Set*.

To set this property, open Navision and click File, Database, Alter and click the ***Advanced*** tab:



☑ Lock timeout

Timeout duration (sec). . 10

☐ Always rowlock

Record Set 500

Alter Database

This property allows you to specify how many records are cached when Navision performs a `FINDSET` operation for a read-only loop (with the `ForUpdate` parameter set to `FALSE`). The default setting is 500 records.

If a `FINDSET` statement reads more than the number of records specified here, additional SQL statements will be sent to the server after the records have been read with `NEXT` and this will degrade performance. Increasing this value also increases the amount of memory that the client uses to support each `FINDSET` statement.

The syntax of the `FINDSET` function is:

```
[Ok :=] FINDSET([ForUpdate][, UpdateKey])
```

And the general rules for using it are:

- `FINDSET(FALSE,FALSE)`- read-only. This uses no server cursors are used and the record set is read with a single server call.
- `FINDSET(TRUE,FALSE)`- is used to update non-key fields. This uses a cursor with a fetch buffer similar to `FIND('-')`.
- `FINDSET(TRUE,TRUE)`- is used to update key fields.

Understanding Navision Indexes and the SQL Server Option

Understanding how to design indexes so that they work optimally in the SQL Server Option is one of the keys to building a successful application.

In Navision, indexes are created for several purposes the most important of which are:

- Data retrieval:

  To quickly retrieve a result set based on a filter. For example, you want to view all the customer ledger entries for a specific customer. There is a table in Navision called ***Cust. Ledger Entry*** (table 18) and the primary index of this table is the **Entry No.** field (*Integer*). There is also a secondary index consisting of the **Customer No.**, **Posting Date** and the **Currency Code** fields that you can use to quickly retrieve a result set that is filtered by **Customer No.**

- Sorting:

  To display a result set in a specific order. For example, you want to see all the different document types from the ***Cust. Ledger Entry*** table for the same customer. In this case you could filter on the index that consists of the **Document Type**, **Customer No.**, **Posting Date** and the **Currency Code** fields.

- SIFT (Sum Index FlowField Technology):

  SIFT is used to maintain pre-calculated sums for various columns. For example, if you want to have daily/monthly/yearly summaries of the Sales and Profit columns in the ***Cust. Ledger Entry*** table, you must have SIFT turned on for an index that contains the following SumIndexFields **Sales (LCY)**,**Profit (LCY)**.

How Indexes Work in Navision

- All the indexes in Navision are unique. Note that in the earlier example, the index is *Customer No., Posting Date, Currency Code, Entry No***.** to enforce uniqueness.
- A primary index in Navision translates to a unique clustered index on SQL Server and a secondary index in Navision translates to unique non-clustered index in SQL Server.
- Navision Database Server supports SIFT effortlessly. However, in the SQL Server Option, when a SIFT field is defined on any index an extra table is created on SQL Server. This table is maintained by triggers that have been placed on the source data table. Based on the earlier example, Navision creates a new SIFT table on SQL Server for the source data table. The source data table is called ***CRONUS International Ltd_$Cust_ Ledger Entry*** and the SIFT table is called ***CRONUS International Ltd_$21$0*** and looks like this:

CRONUS International Ltd_$21$0

Bucket	Contents
[bucket] [int]	total/day/month/and so on
[f3] [varchar]	Field3 in Navision = "**Customer No.**"
[f4] [datetime]	Field4 in Navision = "**Posting Date**"
[f11] [varchar]	Field11 in Navision = "**Currency Code**"
[s18] [decimal]	Sum of Field18 in Navision = "**Sales(LCY)**"
[s19] [decimal]	Sum of Field19 in Navision = "**Profit(LCY)**"
[s20] [decimal]	Sum of Field20 in Navision = "**Inv.Discount (LCY)**"

The table name is constructed as: `[Company Name$Table Number$SIFTindexNo]`

- If this table contains another index that maintains SIFT, Navision creates yet another SIFT table.
- Navision Database Server can only sort a result set in descending/ascending order. To have the results sorted by for example **Customer No.**, **Document Type**, **Posting Date**, **Currency Code**, you must create an index in Navision that matches this order.
- Tables that contain transaction details use an *Integer* as the primary index. In the ***Cust. Ledger Entry*** table, the primary index is **Entry No.** – a field of data type *Integer*.
- Boolean (Yes/No) fields in Navision are *tinyint* on SQL Server.
- Option fields in Navision are *Integer* on SQL Server. An example of an option field in Navision is the **Document Type** field in the ***Cust. Ledger Entry*** table
- The Navision client reads record by record (ISAM). Navision Database Server is designed to work this way but SQL Server is not. The SQL Server Option for Navision has been designed to detect when it is reading in a loop or reading single records.

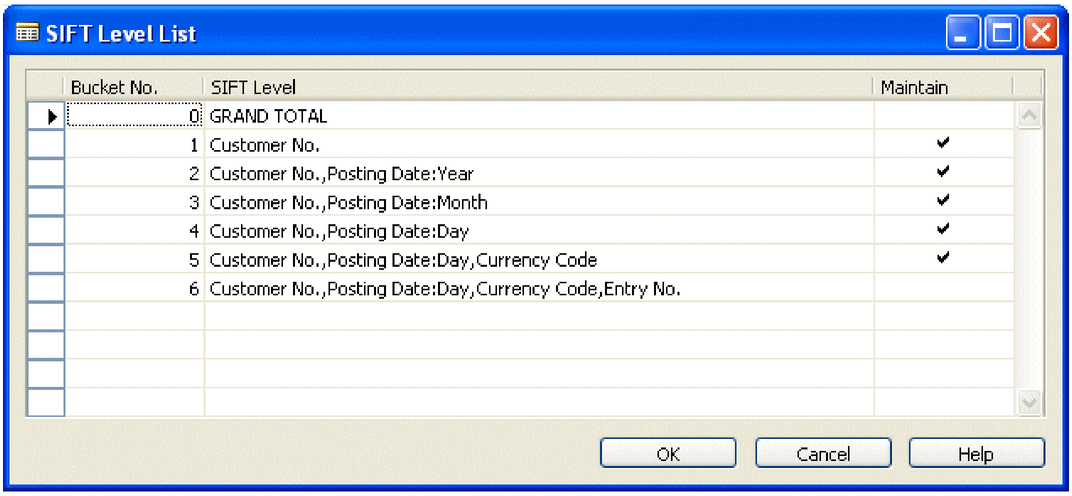
When it detects that it is reading single records only, it switches to singleton queries such as SELECT TOP 1 instead of set-based queries with subsequent fetches.

This mechanism is disturbed if the application modifies a key in the index that the loop is based on or if the `WHILE FIND('-')` method is used instead of the `REPEAT UNTIL NEXT` method. Furthermore, the SQL Server query optimizer is often disturbed by this and frequently switches to another non-clustered index scan or clustered index scan execution plan. Navision Database Server is robust in this scenario.

- Navision uses SIFT technology to display FlowFields. These are calculated fields and are not stored permanently.
- An example is the **Sales (LCY)** field in the ***Customer*** table. This field is defined as: Sum("Cust. Ledger Entry"."Sales (LCY)" WHERE (Customer No.=FIELD(No.),Global Dimension 1 Code=FIELD(Global Dimension 1 Filter),Global Dimension 2 Code=FIELD(Global Dimension 2 Filter),Posting Date=FIELD(Date Filter),Currency Code=FIELD(Currency Filter)))
- There are several different types of FlowField – SUM, AVERAGE, EXIST, COUNT, MIN, MAX, LOOKUP. However, the standard Navision application does not use the AVERAGE, MIN and MAX methods.
- In Navision, you can define which indexes should be maintained on SQL Server by setting the *MaintainSQLIndex* property for each index. If the application code refers to a specific index, it needs to be defined in the Navision table; however it doesn't need to be defined on SQL Server. Therefore, if you need two 'similar' yet different sort orders in the Navision application:
  - 1) Field1, Field2, Field3, Field4
  - 2) Field1, Field2, Field4, Field3

  You must define both indexes. However, you can disable the maintenance of one of the indexes on SQL Server. SQL Server will still be able to retrieve the result set based on the index that is maintained and sort the result set in the requested sort order.
- In Navision, you can specify which SIFT indexes should be maintained on SQL Server by setting the *MaintainSIFTIndex* property for each index. When you decide to maintain a SIFT index, you can also specify which SIFT levels should be maintained by using the *SIFTLevelsToMaintain* property.
- In the ***Cust. Ledger Entry*** table, select the secondary index that consists of the **Customer No.**, **Posting Date** and the **Currency Code** fields and open the ***Properties*** window for this key.

- In the **Value** field of the *SIFTLevelsToMaintain* property, click the AssistButton to open the ***SIFT Levels List*** window:



Bucket No.	SIFT Level	Maintain
0	GRAND TOTAL	
1	Customer No.	✓
2	Customer No.,Posting Date:Year	✓
3	Customer No.,Posting Date:Month	✓
4	Customer No.,Posting Date:Day	✓
5	Customer No.,Posting Date:Day,Currency Code	✓
6	Customer No.,Posting Date:Day,Currency Code,Entry No.	

SIFT Levels List

In this window, you specify which SIFT buckets you want to maintain.

If Navision cannot return the requested sum by using the SIFT table, it performs the SUM operation on the source table. This can take a long time if there are many records in the source table's result set. However, if the result set is small, the response time is similar. Furthermore, maintaining SIFT indexes is costly because every update of a source table causes at least one and probably multiple updates of the SIFT table(s).

- The Navision indexes have not been redesigned for SQL Server. The scope of the product development when developing SQL Server Option was defined as "one application on both server platforms".
- Navision Database Server is optimized for low selectivity keys. For example, if there is an index that consists of the **Document Type** and **Customer No.** fields and the application filters on the **Customer No.** field only, Navision Database Server will search through the index branches and retrieve the result set quickly. However, SQL Server is not optimized to do that, it scans from the beginning to the end of a range and in many cases this results in a non-clustered index scan.
- Both server options are inefficient when filtering on datetime fields, therefore the Navision application defines indexes with datetime keys towards the end of an index, for example: **Document Type**,**Customer No.**,**Currency Code**,**Posting Date**.

Minimizing the Impact on SQL Server

To minimize the impact of the way the Navision application is designed when running the SQL Server Option:

- Eliminate the maintenance of indexes that are only designed for sorting purposes. SQL Server will sort the result set without these indexes.
- Redesign the indexes so that their selectivity becomes higher by putting Boolean, Option and Date fields towards the end of the index.
- Do not maintain SIFT indexes on small tables or temporary tables. Examples of temporary tables are: ***Sales Line***, ***Purchase Line***, ***Warehouse Activity Line***, and similar tables.
- If you loop through a table, set a separate looping variable.

- Never use `WHILE FIND(‘-’)` or `WHILE FIND(‘+’)` structures.

  `WHILE FIND(‘-’)` or `WHILE FIND(‘+’)` logic is always used to look for the first or last record in a set and therefore automatically disables the read ahead mechanism. It is therefore recommended that you do not use this logic unless you have some very good reasons for doing so.

  Example of bad code:

  ```
  Customer.SETCURRENTKEY(“Currency Code”);
  Customer.SETRANGE(“Currency Code”, ‘GBP’);
  WHILE Customer.FIND(‘-‘) DO BEGIN
    Customer.“Currency Code” := ‘EUR’;
    Customer.MODIFY;
  END;
  ```

Useful SQL Server Commands

sp_helpindex

This command displays the indexes for a specified table.

Example 1

```
sp_helpindex [CRONUS International Ltd_$Cust_ Ledger Entry]
```

Result 1

Index_name	Index_description	Index_keys
$1	nonclustered, unique located on PRIMARY	Customer No_, Posting Date, Currency Code, Entry No_
$10	nonclustered, unique located on PRIMARY	Customer No_, Applies-to ID, Open, Positive, Due Date, Entry No_
$2	nonclustered, unique located on PRIMARY	Document No_, Document Type, Customer No_, Entry No_
$3	nonclustered, unique located on PRIMARY	Document Type, External Document No_, Customer No_, Entry No_
$4	nonclustered, unique located on PRIMARY	Customer No_, Open, Positive, Due Date, Currency Code, Entry No_
$5	nonclustered, unique located on PRIMARY	Open, Due Date, Entry No_
$6	nonclustered, unique located on PRIMARY	Document Type, Customer No_, Posting Date, Currency Code, Entry No_
$7	nonclustered, unique located on PRIMARY	Salesperson Code, Posting Date, Entry No_
$8	nonclustered, unique located on PRIMARY	Closed by Entry No_, Entry No_
$9	nonclustered, unique located on PRIMARY	Transaction No_, Entry No_
CRONUS International Ltd_$Cust_ Ledger Entry$0	clustered, unique, primary key located on PRIMARY	Entry No_

Some indexes are never or at most rarely used by the application (index $7 is used only for one report in Navision – ***Salesperson Commission***). You should consider dropping these and thereby no longer maintaining them on SQL Server.

Example 2

```
sp_helpindex [CRONUS International Ltd_$Warehouse Activity Line]
```

Result 2

Index_name	Index_description	Index_keys
$1	nonclustered, unique located on PRIMARY	Source Type, Source Subtype, Source No_, Source Line No_, Source Subline No_, Unit of Measure Code, Action Type, Breakbulk No_, Original Breakbulk, Activity Type, No_, Line No_
$10	nonclustered, unique located on PRIMARY	Activity Type, No_, Action Type, Bin Code, Line No_
$11	nonclustered, unique located on PRIMARY	Activity Type, No_, Item No_, Variant Code, Action Type, Bin Code, Line No_
$12	nonclustered, unique located on PRIMARY	Whse_ Document No_, Whse_ Document Type, Activity Type, Whse_ Document Line No_, Action Type, Unit of Measure Code, Original Breakbulk, Breakbulk No_, Lot No_, Serial No_, No_, Line No_
$13	nonclustered, unique located on PRIMARY	Item No_, Bin Code, Location Code, Action Type, Variant Code, Unit of Measure Code, Breakbulk No_, Activity Type, Lot No_, Serial No_, No_, Line No_
$14	nonclustered, unique located on PRIMARY	Item No_, Location Code, Activity Type, Bin Type Code, Unit of Measure Code, Variant Code, Breakbulk No_, Action Type, Lot No_, Serial No_, No_, Line No_
$15	nonclustered, unique located on PRIMARY	Bin Code, Location Code, Action Type, Breakbulk No_, Activity Type, No_, Line No_
$16	nonclustered, unique located on PRIMARY	Location Code, Activity Type, No_, Line No_
$17	nonclustered, unique located on PRIMARY	Source No_, Source Line No_, Source Subline No_, Serial No_, Lot No_, Activity Type, No_, Line No_
$2	nonclustered, unique located on PRIMARY	Activity Type, No_, Sorting Sequence No_, Line No_
$3	nonclustered, unique located on PRIMARY	Activity Type, No_, Shelf No_, Line No_
$4	nonclustered, unique located on PRIMARY	Activity Type, No_, Location Code, Source Document, Source No_, Action Type, Zone Code, Line No_
$5	nonclustered, unique located on PRIMARY	Activity Type, No_, Due Date, Action Type, Bin Code, Line No_
$6	nonclustered, unique located on PRIMARY	Activity Type, No_, Bin Code, Breakbulk No_, Action Type, Line No_
$7	nonclustered, unique located on PRIMARY	Activity Type, No_, Bin Ranking, Breakbulk No_, Action Type, Line No_
$8	nonclustered, unique located on PRIMARY	Activity Type, No_, Destination Type, Destination No_, Action Type, Bin Code, Line No_
$9	nonclustered, unique located on PRIMARY	Activity Type, No_, Whse_ Document Type, Whse_ Document No_, Whse_ Document Line No_, Line No_
CRONUS International Ltd_$Cust_ Ledger Entry$0	clustered, unique, primary key located on PRIMARY	Activity Type, No_, Line No_

This table is heavily over indexed with long composite indexes. This table could be very ‘hot’ in a busy warehousing environment. You should consider dropping the maintenance on SQL Server of indexes with the same beginning or composition; hopefully the resulting subset will be small enough and will be sorted quickly enough.

In this example, indexes $10, $11, $2, $3, $4, $5, $6, $7, $8 $9 start with the same fields **[Activity Type]**,**[No_]** which are the same as the beginning of the clustered index. If you always filter on these two keys, there is a high probability that the SQL Server query optimizer will estimate that the cost of using the non-clustered index in this scenario is too high (because using the clustered index will mean fetching the data as well), and may decide to do a *clustered index seek*, or use any of the similar indexes.

Additionally, if you reduce the number of indexes, SQL Server will not have to keep these in memory, thereby increasing page life expectancy and improving performance. Maintaining fewer indexes can mean better performance.

dbcc show_statistics

Example 1

```
dbcc show_statistics ([CRONUS International Ltd_$Cust_ Ledger
Entry],[$3])
```

Result 1

All density	Average Length	Columns
0.33333334	4.0	Document Type
0.33333334	4.0	Document Type, External Document No_
4.3478262E-2	10.086206	Document Type, External Document No_, Customer No_
1.7241379E-2	14.086206	Document Type, External Document No_, Customer No_, Entry No_

The **All Density** column shows the selectivity of the key (how distinct the values are). In this case, the first two keys have very poor selectivity, if you filter a result set on these two keys only, SQL Server will have to fetch 1/3 of the table. In such cases, and sometimes even if you filter further on this composite index, for example on the **[Customer No_]** key, the SQL Server query optimizer might switch to a *clustered index scan* and not use the ‘best’ index, or switch to another index and do a *nonclustered index scan* or seek (remember that seek is good, scan is bad).

Consider using the SQLIndex key property to drop (no longer maintain) this index, and/or create a new index in Navision for, example, **Customer No.**,**Document Type**,… and maintain this one while disabling the maintenance of the original index. This way, your Navision code will work and also give the optimum performance.

The SQLIndex property allows you to define the actual fields that are used in the corresponding index on SQL Server.

Example 2

```
dbcc show_statistics ([CRONUS International Ltd_$Cust_ Ledger
Entry],[$3])
```

Result 2 (only the ‘interesting’ part is shown)

All density	Average Length	Columns
0.02	5.7974682	Customer No_
0.01	13.797468	Customer No_, Posting Date
0.01	19.063292	Customer No_, Posting Date, Currency Code
1.2658228E-2	23.063292	Customer No_, Posting Date, Currency Code, Entry No_

The **All Density** column shows the selectivity of the key (how distinct values are). In this case, the first key has good selectivity, if you filter a result set on that key, SQL Server will have to fetch only 2 percent of the table. However, placing a field of date data type (**Posting Date**) towards the beginning of the index (second key) is not very wise because that field has too high selectivity (it’s too granular). This means that it doesn’t matter what key you define next in the index and try to filter on, SQL Server will never use the index for that part and seek or scan the index because almost the entire subset contains possible data.

Consider putting the date field at the end of the index. To avoid runtime errors in Navision, keep the index in the table design but turn off the SIFT maintenance, and design a new index which might give better performance.

Optimizing Navision Indexes and SIFT Tables

This section suggests a methodology and some tools that you can use when optimizing Navision indexes on SQL Server.

Minimize the Number of Indexes

Don’t maintain indexes that are only used for sorting purposes. SQL Server will sort the result set. The main focus should be on quickly retrieving the result set. If you, for example, have several indexes that start with the same combination of keys (index fields), you should maintain only one of them on SQL Server. Hopefully you can identify the one that is most used in the most situations. In Navision, you should then use the *MaintainSQLIndex* property of the other indexes to specify that they should no longer be maintained on SQL Server.

Indexes on ‘Hot’ Tables

Avoid having too many indexes on ‘hot’ tables because each record update means an index update producing more disk I/Os. For example, if the ***Item Ledger Entry*** table is growing by 1000 records per day and has 20 indexes, it can easily produce in excess of 20000 disk I/Os per day. However, if you reduce the number of indexes to for example 5, it greatly reduces the number of disk I/Os. Experience has shown that it is always possible to reduce the number of indexes to between 5 and 7 and even less on ‘hot’ tables.

Redesign Indexes for Better Selectivity

Redesign indexes so that their selectivity becomes higher. Remember to not place Boolean and option fields at the beginning of an index and always put date fields towards the end of the index.

Furthermore, indexes like **Posting Date**,**Customer No.**,… must be avoided because the index is like an entire book and the chances that the query optimizer would pick this index are quite small.

Indexes like **Document Type**,**Customer No.**,… have very low selectivity on the first key. You could create a new index **Customer No.**,**Document Type**, … and maintain it on SQL Server while you turn off the maintenance of the original index on SQL Server.

SQLIndex Key Property

Navision contains a key property, called SQLIndex that allows you to create indexes on SQL Server that are different from the keys defined in Navision.

This property is designed to address selectivity issues by allowing you to optimize the way that Navision uses SQL Server indexes without having to rewrite your application or redesign your keys in Navision.

This generally means rearranging the columns in the SQL Server index but you can also define a completely new index if you wish.

If you want to create a unique index, you must include all the columns from the primary key in the index you define on SQL Server. However non-unique indexes are also allowed

This property is designed to help you minimize the impact of the way the Navision application is designed when running the SQL Server Option.

You can use it to:

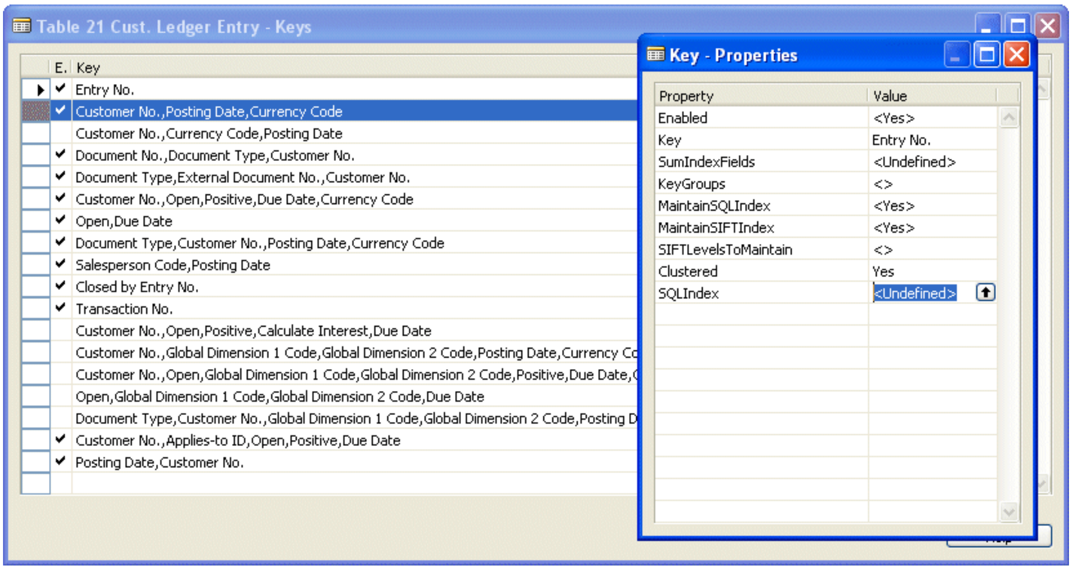
- Eliminate the maintenance of indexes that are only designed for sorting purposes. After you have removed or changed these indexes, SQL Server will still sort the result.
- Redesign the indexes so that their selectivity becomes higher by putting Boolean, Option and Date fields towards the end of the index.

Clustered Key Property

In Navision the clustered index is normally the primary key. The Clustered property allows you to specify that another key should be the clustered index. You can also have tables that do not contain any clustered key.

This is a way to help the SQL Server Query Optimizer determine the optimal execution plan for each query.

To set the SQLIndex and Clustered properties for a table, open the table you want to redesign, open the ***Keys*** window, select the key you want to modify and open the ***Properties*** window for that key:



E.	Key
✓	Entry No.
✓	Customer No.,Posting Date,Currency Code
	Customer No.,Currency Code,Posting Date
✓	Document No.,Document Type,Customer No.
✓	Document Type,External Document No.,Customer No.
✓	Customer No.,Open,Positive,Due Date,Currency Code
✓	Open,Due Date
✓	Document Type,Customer No.,Posting Date,Currency Code
✓	Salesperson Code,Posting Date
✓	Closed by Entry No.
✓	Transaction No.
	Customer No.,Open,Positive,Calculate Interest,Due Date
	Customer No.,Global Dimension 1 Code,Global Dimension 2 Code,Posting Date,Currency C
	Customer No.,Open,Global Dimension 1 Code,Global Dimension 2 Code,Positive,Due Date,
	Open,Global Dimension 1 Code,Global Dimension 2 Code,Due Date
	Document Type,Customer No.,Global Dimension 1 Code,Global Dimension 2 Code,Posting D
✓	Customer No.,Applies-to ID,Open,Positive,Due Date
✓	Posting Date,Customer No.

Property	Value
Enabled	<Yes>
Key	Entry No.
SumIndexFields	<Undefined>
KeyGroups	<>
MaintainSQLIndex	<Yes>
MaintainSIFTIndex	<Yes>
SIFTLevelsToMaintain	<>
Clustered	Yes
SQLIndex	<Undefined>

Key Properties window

Needless to say, the SQLIndex and Clustrered properties are not a cure-all for any problems that keys might be causing in your application. They should only be used after you have given lengthy consideration to the needs of your Navision application.

Important

Before you redesign a table and the indexes that it contains, you must ensure that the *Maintain relationships* database property is not checked. To see whether or not the property is used in your database, click File, Database, Alter to open the ***Alter Database*** window and click the ***Integration*** tab. Remember to check the property again after the table has been redesigned.

Small/Temporary Tables SIFT Maintenance

Do not maintain SIFT indexes on small tables or on temporary tables. Examples of temporary tables are ***Sales Line***, ***Purchase Line***, ***Warehouse Activity Line***, and similar tables. SQL Server can retrieve sums directly from the ‘source’ table when the SIFT indexes are not maintained. If the data set within the filter is small, the sum is calculated quickly and at the same time all the updates of the ‘source’ table are performed more quickly because Navision does not have to update all the associated SIFT tables.

Minimize Number of SIFT Buckets

Minimize the number of buckets that are maintained for each SIFT index. There is, for example, no reason to maintain the daily bucket for the ***Cust. Ledger Entry*** table if you only post several invoices and payments a month for the same customer. Furthermore, if you have several SIFT indexes defined on a table that you design, you should investigate whether or not some of the buckets are already maintained by another index. For example if you have two indexes **Customer No.**,**Currency Code**,… and **Customer No.**,**Open**,… which both maintain Amount (LCY) sums, you could disable the bucket that maintains the totals per Customer No. in one of the indexes.

Do not assume that the following example is 100% correct – there might be different record sizes, different fill factors, different block and sector/stripe sizes, and so on. It is merely provided as an illustration.

If a SIFT table contains a lot of buckets, every single update of the 'source' table produces a large number of bucket updates. Remember that each SIFT table has two indexes. For example, imagine that you maintain 40 buckets on the ***G/L Entry*** table and 8 indexes – every time you insert a record into the ***G/L Entry*** table, the database engine must update 40 buckets (80 index entry updates), one 'source' record and it's 8 indexes (all/some keys). This single insert could produce in excess of 100 I/Os on the disk subsystem.

Obviously the smaller the records in the table the smaller the problems associated with the indexes and the SIFT indexes become.

Tools

This section describes some useful tools that you can use with the SQL Server Option for Navision.

The topics covered include:

- Index size and structure
- Database size and growth
- Selecting and fetching on SQL Server

Indexes

This section describes how to analyze the indexes that have been created in your Navision database.

Indexes per Table

The following query will list all the indexes in a Navision database by table:

```
SELECT
  OBJECT_NAME(id) AS [Object Name],
  name AS [Index Name]
FROM sysindexes
WHERE (name NOT LIKE '_WA_%') AND (id > 255)
ORDER BY [Object Name]
```

Number of Indexes per Table

The following query will show the number of indexes per table in a Navision database sorted by the number of indexes:

```
SELECT
  OBJECT_NAME(id) AS [Object Name],
  COUNT (id) AS [No. of Indexes]
FROM sysindexes
WHERE (name NOT LIKE '_WA_%') AND (id > 255)
GROUP BY id
ORDER BY [No. of Indexes] DESC, [Object Name]
```

Number of Index Updates

If your database has been running for several months and contains data that is typical for your application, the following query will show you the number of index updates that were required when records have been inserted into the database. The results are sorted by the number of updates.

```
SELECT
  OBJECT_NAME(id) AS [Object Name],
  COUNT (id) AS [No. of Indexes],
  rowcnt AS [No. Of Rows],
  (rowcnt * COUNT(id)) AS [No. Of Updates]
FROM sysindexes
WHERE (name NOT LIKE '_WA_%') AND (id > 255)
GROUP BY id, rowcnt
ORDER BY [No. Of Updates] DESC
```

Index Structures

The following query shows the structure of the indexes in your database and sorts the results by table:

```
SET NOCOUNT ON

--DROP TABLE ##Table
--DROP TABLE ##IndexHelp

CREATE TABLE ##Table (
  [table_name] VARCHAR(255)

)

CREATE TABLE ##IndexHelp (
  [table_name] VARCHAR(255) DEFAULT '',
  [index_name] VARCHAR(255),
  [index_description] VARCHAR(255),
  [index_keys] VARCHAR(512)
)

INSERT INTO ##Table
  SELECT [name] AS [table_name]
  FROM sysobjects
  WHERE [xtype] = 'U' and [id] > 255
  ORDER BY [name]


DECLARE @Statement CHAR (255)
DECLARE @name sysname
DECLARE Get_Curs CURSOR FOR
  SELECT table_name FROM ##Table


OPEN Get_Curs

FETCH NEXT FROM Get_Curs INTO @name

WHILE @@FETCH_STATUS = 0 BEGIN

  INSERT INTO ##IndexHelp (index_name, index_description,
index_keys)
    EXEC('sp_helpindex [' + @name + ']')

  UPDATE ##IndexHelp SET table_name = @name WHERE table_name = ''

  FETCH NEXT FROM Get_Curs INTO @name
END

CLOSE Get_Curs
DEALLOCATE Get_Curs

SELECT * FROM ##IndexHelp
ORDER BY table_name
```

Number of SIFT Indexes

The following query shows the number of SIFT indexes per 'source table':

```
SELECT

  SUBSTRING(name,1,CHARINDEX('$',name,CHARINDEX('$',name)+1)) AS
[Source Table],
COUNT(SUBSTRING(name,1,CHARINDEX('$',name,CHARINDEX('$',name)+1)))
AS [No. Of SIFT Indexes]
FROM sysobjects
WHERE OBJECTPROPERTY(id, N'IsUserTable') = 1 AND name LIKE
'%'+'$'+'[0-9]'+'%'
GROUP BY
(SUBSTRING(name,1,CHARINDEX('$',name,CHARINDEX('$',name)+1)))
ORDER BY [No. Of SIFT Indexes] DESC, [Source Table]
```

Number of SIFT Buckets

The following query shows the number of buckets in the SIFT tables.

However, if your database does not contain any data and/or some SIFT ‘source’ tables have not been populated, the query will not show any or just a few buckets.

```
SET NOCOUNT ON

--DROP TABLE ##SIFTtables

CREATE TABLE ##SIFTtables (
  [table_name] VARCHAR(255) DEFAULT '',
  [bucks] INT DEFAULT 0
)


DECLARE @Statement CHAR (255)
DECLARE @tname sysname

DECLARE Get_Curs CURSOR FOR
  SELECT name
  FROM sysobjects
  WHERE OBJECTPROPERTY(id, N'IsUserTable') = 1 AND name LIKE
'%'+'$'+'[0-9]'+'%'
  ORDER BY name

OPEN Get_Curs

FETCH NEXT FROM Get_Curs INTO @tname

WHILE @@FETCH_STATUS = 0 BEGIN

  INSERT INTO ##SIFTtables (bucks)
    EXEC('SELECT COUNT(DISTINCT bucket) AS bucks FROM
['+@tname+']')

  UPDATE ##SIFTtables SET table_name = @tname WHERE table_name =
''

  FETCH NEXT FROM Get_Curs INTO @tname
END

CLOSE Get_Curs
DEALLOCATE Get_Curs

SELECT table_name AS [Table Name], bucks AS [No. Of Buckets] FROM
##SIFTtables
```

To ensure that this query gives you the right results:

- Create an empty database solely for this purpose and restore your objects into it.

- Import and compile the `FillSIFTBuckets.txt` object into Navision (it is supplied as a text file so that you can renumber the object to an unused object number).
- Run the ***Fill SIFT Buckets*** codeunit – this code inserts and deletes one record into every Navision table thereby ensuring that all the SIFT buckets are populated with zero values (Navision doesn't delete 'empty' buckets.

Key Information Tool

This tool could be used as alternative to the tools that we have just described.

This tool gives you an overview of the indexes that have been defined for each table including the number of keys that been enabled, the SumIndexFields that are supported and the SIFT levels that are enabled.

Note
The Key Information tool is a read-only tool and does not update the tables in the database. The information that the tool displays is not updated dynamically.

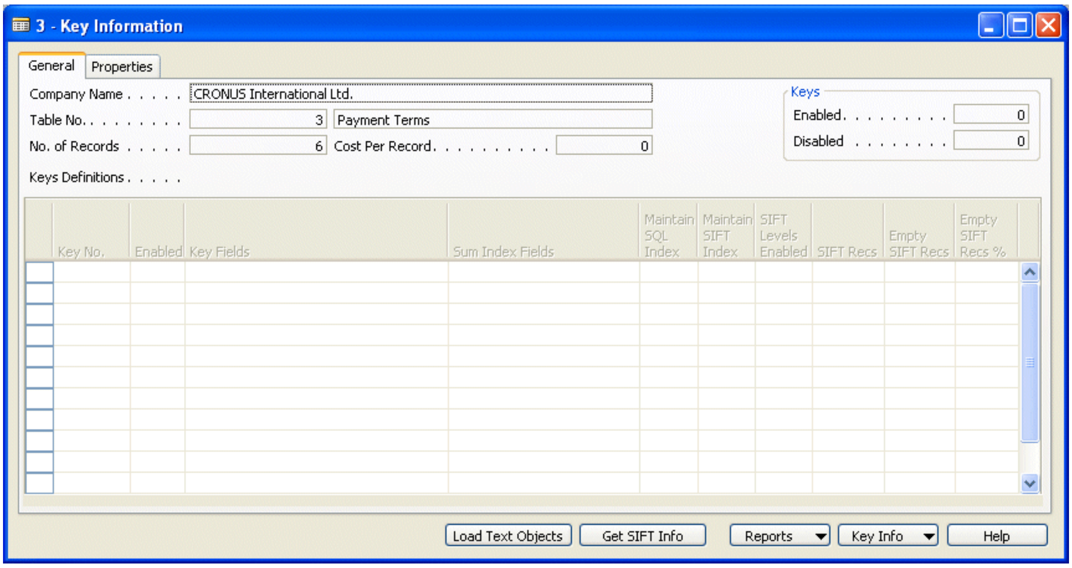
Before you can use the Key Information tool you must import the `SQL key information.fob` file that contains all the objects that make up the tool.

Setting Up the Key Information Tool

After you have imported the `SQL key information.fob` file, you must set up the tool before you can use it.

To set up the Key Information Tool:

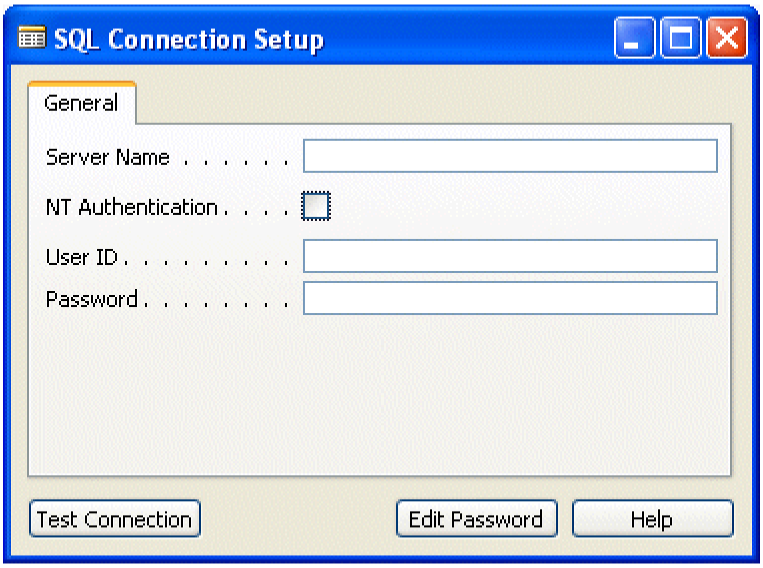
1. Select form 50070, ***Key Information*** and run it. The first time you run the form it loads all the information from the ***Object*** and the ***Table Information*** virtual tables. It also displays a form containing a more concise version of the following instructions.



Company Name: CRONUS International Ltd.
Table No.: 3 Payment Terms
No. of Records: 6
Cost Per Record: 0
Keys
Enabled: 0
Disabled: 0
Keys Definitions

Key Information

2. In the ***Key Information*** window, click Key Info, Setup SQL Connection to open the ***SQL Connection Setup*** window:



SQL Connection Setup

3. In the ***SQL Connection Setup*** window, enter the following information:

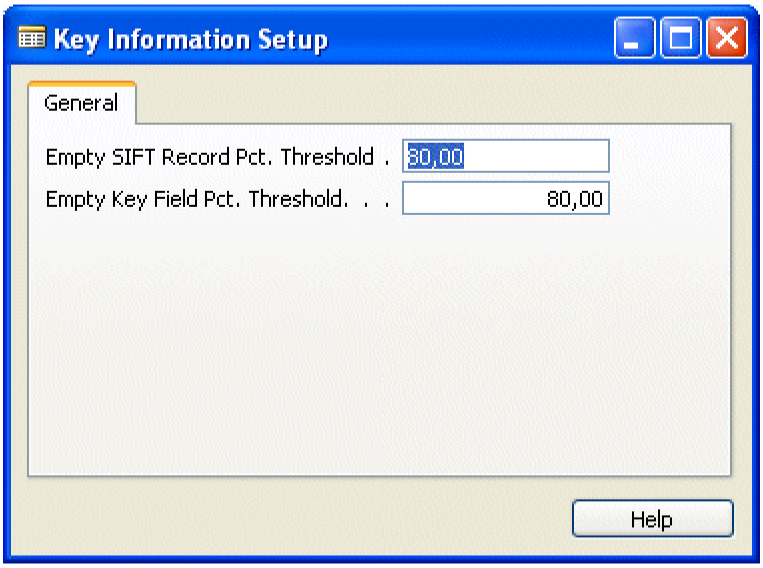
Field	**Input**
Server Name	The name of the SQL Server. You can find the name in the ***Database Information*** window.
NT Authentication	A check mark here tells the system to use the current NT Authentication ID. Otherwise you must enter a Used ID and a password.
User ID	The ID of the database login that you are using to connect to the server. This user should be either a System Administrator or DBO. This field is only used if you are not using NT Authentication.
Password	The password of the database login. This field is only used if you are not using NT Authentication.

The ***SQL Connection Setup*** window also contains two buttons:

- Test Connection –tests whether or not the information you entered allows you to connect with the server.
- Edit Password – this button opens a window that allows you to change the password of the database login.

4. After you have entered this information, close the ***SQL Connection Setup*** window.

5. In the ***Key Information*** window, click Key Info, Setup and the ***Key Information Setup*** window opens:



Empty SIFT Record Pct. Threshold . . 80,00

Empty Key Field Pct. Threshold. . . 80,00

Key Information Setup

This window contains two fields:

- **Empty SIFT Record Pct. Threshold** – the percentage of empty SIFT records that are allowed within a SIFT level. The SIFT levels that match or exceed the value entered in this field are colored in the ***Key Field List*** form. This information is also used to filter out unwanted lines in the ***Key Information*** report.
- **Empty Key Field Pct. Threshold** – the percentage of empty key fields that are allowed within the SIFT records of a particular key. The fields that match or exceed the value entered in this field are colored in the ***Key Field List*** form. This information is also used to filter out unwanted lines in the ***Key Information*** report.

When you open this window both fields contain the default value – 80%.

6. Close the ***Key Information Setup*** window.

The Key Information tool is now ready for use.

Using the Key Information Tool

You can now start to use the tool to analyze some or all of the tables in your database.

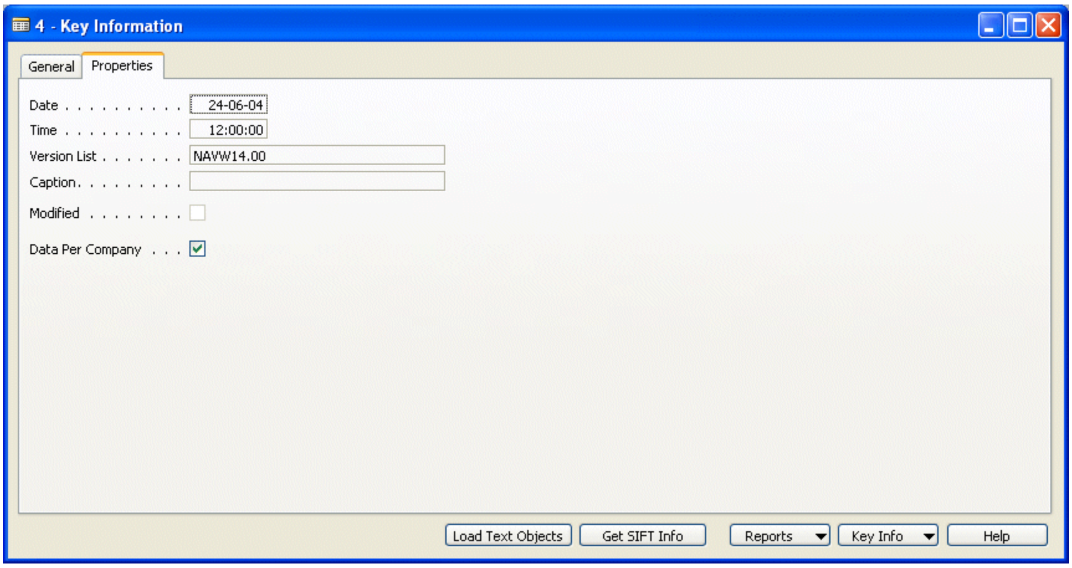
To import the information into the tool:

1. In the Object Designer, export the table definitions of the tables that you want to analyze.
2. In the ***Key Information*** window, click Load Text Objects. Browse to the text file that you just exported from the Object Designer and click Open to import the table definitions.
3. In the ***Key Information*** window, click Get SIFT Info to extract the SIFT information from the database and update the SIFT columns in the tool.

The ***General*** tab of the ***Key Information*** window contains the following fields:

Field	Meaning
Company Name	The name of the company that you are analyzing. If the database contains more than one company, you must run the tool once for each company in the database.
Table No.	The number and name of the table that is currently displayed.
No. of Records	The number of records in the table. This value is retrieved from the table Information table and is the number of records that were in the table when you imported the data into the tool. This value is not updated dynamically.
Cost Per Record	The number of updates (write transactions) that must be performed every time you insert or modify a record in this table. This value is calculated by adding the number of keys that are enabled in this table to the number of SIFT levels that are enabled for each key. This value can help you see which tables in the database have the highest performance overhead. You can then consider reducing the cost per record.
Keys – Enabled	The number of keys that are enabled in this table.
Keys – Disabled	The number of keys that are disabled in this table.
Key No.	The number of the key.
Enabled	Whether the key is enabled or not.
Key Fields	A list of the fields that make up the key.
Sum Index Fields	A list of the SumIndexFields in the key.
Maintain SQL Index	Whether or not a SQL index is maintained for this key.
Maintain SIFT Index	Whether or not a SIFT index is maintained for this key.
SIFT Levels Enabled	The number of SIFT levels that are maintained for this key.
SIFT Recs	The total number of SIFT records created for all the SIFT levels that are maintained.
Empty SIFT Recs	The total number of empty records created for all the SIFT levels that are maintained
Empty SIFT Recs %	The percentage of the total number of SIFT records that are empty.

In the ***Key Information*** window, the ***Properties*** tab displays the properties of the selected table:



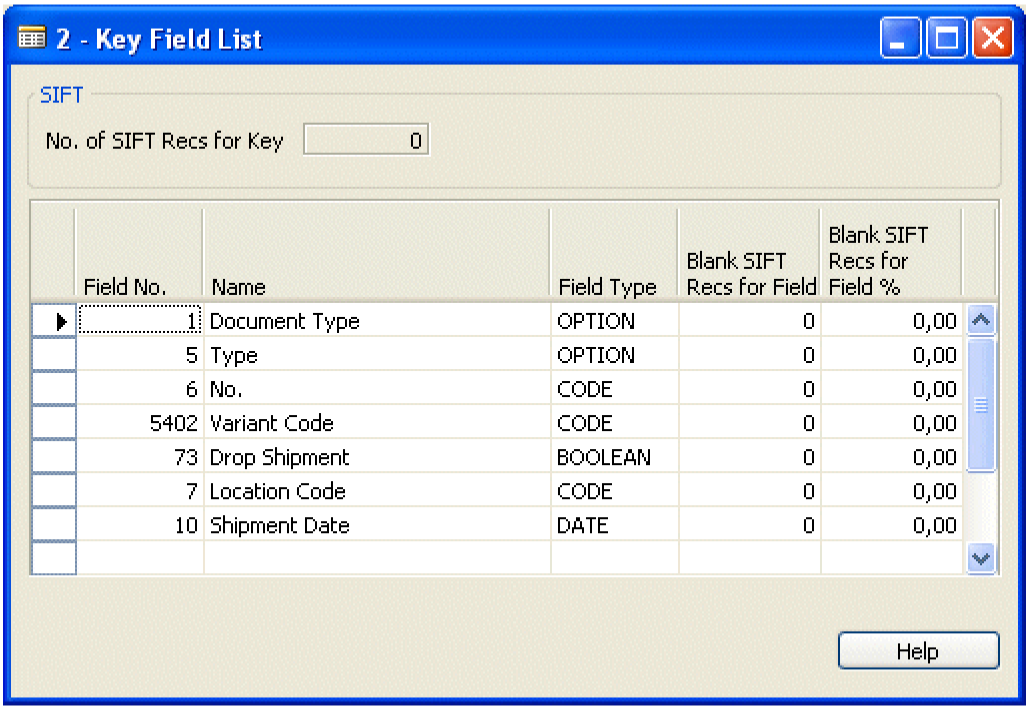
Date 24-06-04
Time 12:00:00
Version List NAVW14.00
Caption. . . . .
Modified . . . . ☐
Data Per Company . . . ☑

Key Information Properties Tab

This information is retrieved from the ***Objects*** virtual table and is not updated dynamically.

To analyze the information gathered by the Key Information tool:

1. In the ***General*** tab of the ***Key Information*** window, select the key that you want to analyze and click the **Key Fields** field.
2. In the **Key Fields** field, click the AssistButton and the ***Key Field List*** window appears:



No. of SIFT Recs for Key 0

Field No.	Name	Field Type	Blank SIFT Recs for Field	Blank SIFT Recs for Field %
1	Document Type	OPTION	0	0,00
5	Type	OPTION	0	0,00
6	No.	CODE	0	0,00
5402	Variant Code	CODE	0	0,00
73	Drop Shipment	BOOLEAN	0	0,00
7	Location Code	CODE	0	0,00
10	Shipment Date	DATE	0	0,00

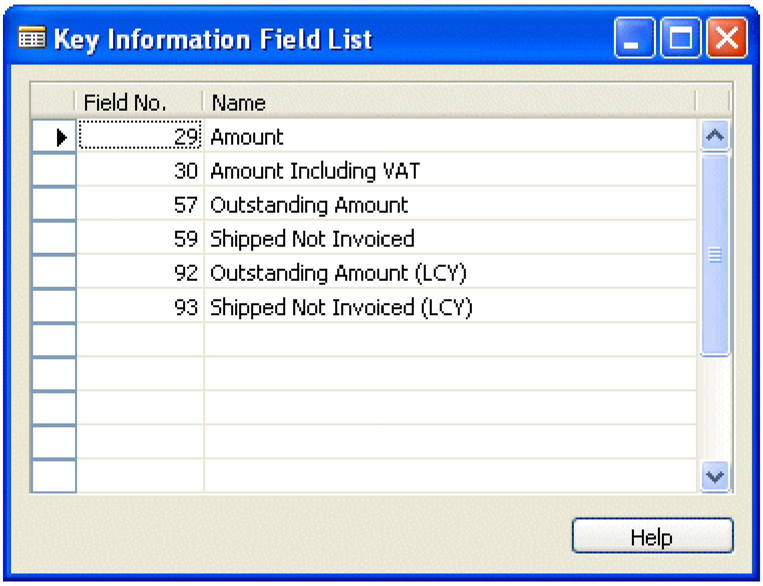
Key Field List

This form displays the SIFT record information for each individual field in the key. The **Blank SIFT Recs for Field %** field gives you an indication of whether or not the field is contributing to the key. If a field is not contributing to a key, it does not help SQL Server to optimize data retrieval. The key is therefore not as efficient as it could be.

Note
In certain types of field (*Option*, *Boolean*, *Integer* and so on) a blank value is not necessarily a blank value. You must analyze the field more carefully to determine whether or not the field really is blank.

3. In the ***Key Information*** window, select the key that you want to analyze and click the **Sum Index Fields** field.

4. In the **Sum Index Fields** field, click the AssistButton and the ***Key Information Field List*** window appears:



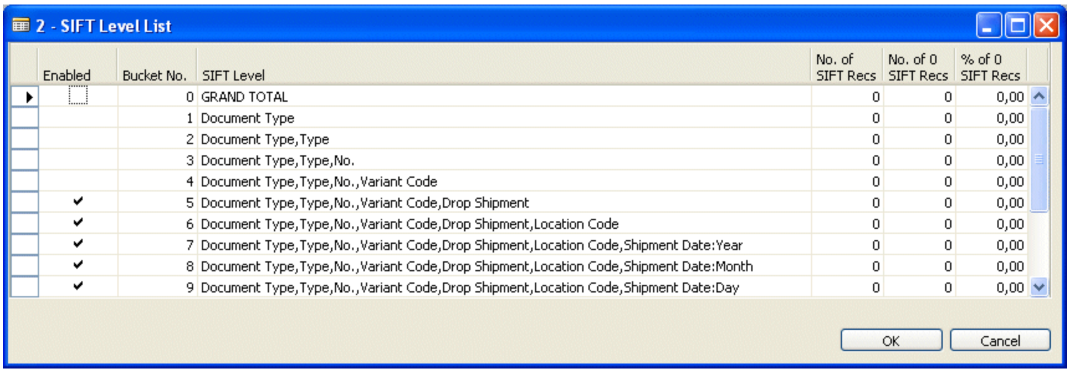
Field No.	Name
29	Amount
30	Amount Including VAT
57	Outstanding Amount
59	Shipped Not Invoiced
92	Outstanding Amount (LCY)
93	Shipped Not Invoiced (LCY)

Key Information Field List

This window lists the SumIndexFields associated with this key.

5. In the ***Key Information*** window, select the key that you want to analyze and click the **SIFT Levels Enabled** field.

6. In the **SIFT Levels Enabled** field, click the AssistButton and the ***SIFT Level List*** window appears:



Enabled	Bucket No.	SIFT Level	No. of SIFT Recs	No. of 0 SIFT Recs	% of 0 SIFT Recs
	0	GRAND TOTAL	0	0	0,00
	1	Document Type	0	0	0,00
	2	Document Type,Type	0	0	0,00
	3	Document Type,Type,No.	0	0	0,00
	4	Document Type,Type,No.,Variant Code	0	0	0,00
✓	5	Document Type,Type,No.,Variant Code,Drop Shipment	0	0	0,00
✓	6	Document Type,Type,No.,Variant Code,Drop Shipment,Location Code	0	0	0,00
✓	7	Document Type,Type,No.,Variant Code,Drop Shipment,Location Code,Shipment Date:Year	0	0	0,00
✓	8	Document Type,Type,No.,Variant Code,Drop Shipment,Location Code,Shipment Date:Month	0	0	0,00
✓	9	Document Type,Type,No.,Variant Code,Drop Shipment,Location Code,Shipment Date:Day	0	0	0,00

SIFT Level List

This window lists all the SIFT levels that are defined for the current key. It also lists the number of records per SIFT level and the number of blank records per SIFT level. These values are then use to calculate the **% of 0 SIFT Recs**.

You can also open this window by clicking the AssistButton in the **SIFT Recs** field and in the **Empty SIFT Recs** field in the ***Key Information*** window.

The Key Information tool also contains two reports that summarize all the information gathered by the tool:

- Key Information – contains all the information gathered by the Key Information tool, except the list of fields that have been defined for the keys.
- Key Field Information – contains the list of fields that have been defined for the keys

Both of these reports can be filtered so that they only show the records that meet or exceed the values that you specified in the ***Key Information Setup*** window.

The Navision Database Sizing Tool

If your application has been running for several months you might have a good idea of how fast the database grows – how many records are added per day/month per table. If you do not have an idea, you can use the Navision Database Sizing Tool (NDST) to estimate the growth rate of your database.

The Navision Database Sizing Tool is designed to help you evaluate your hard disk requirements by estimating the growth rate of your database. The tool can be used to estimate the growth rate of new implementations as well as existing installations.

Scope of NDST

NDST can be used with both database options – Navision and SQL.

The NDST only gives you estimates. The results it generates can be used as guidelines for how much your database will grow. The exact growth rate of the database depends on a number of customer specific factors. No two customers will have the exact same record sizes, or create the exact same number of new records per order.

Database growth depends on factors such as:

- **Application areas used** - A few examples: Customers using Warehouse Management create additional records such as Picks, Putaways, Warehouse entries and so on. If the customer uses Analysis Views, then this will increase database growth; synchronizing contacts with customers will add to the space used, and so on.
- **Working practice -** The more fields you enter in a record, the more space it takes up. A customer who enters every possible piece of information (including comments) in all the records will use more space than a customer with the same number of records, who only enters the most basic information.
- **Record size –** This is based on the size of the table compared to the number of records in that table. The size of a table includes indexes (keys). So record sizes will be affected by the number and size of the indexes.

The record sizes that NDST uses for calculations are only estimates and are based on a very simple setup. The real numbers may well be different.

Contents of NDST

NDST contains:

- `DBSizingWorksheet.xls`,
- `NDST.fob`,
- `SpaceUsed.sql`,

Each of these components is described in the following sections.

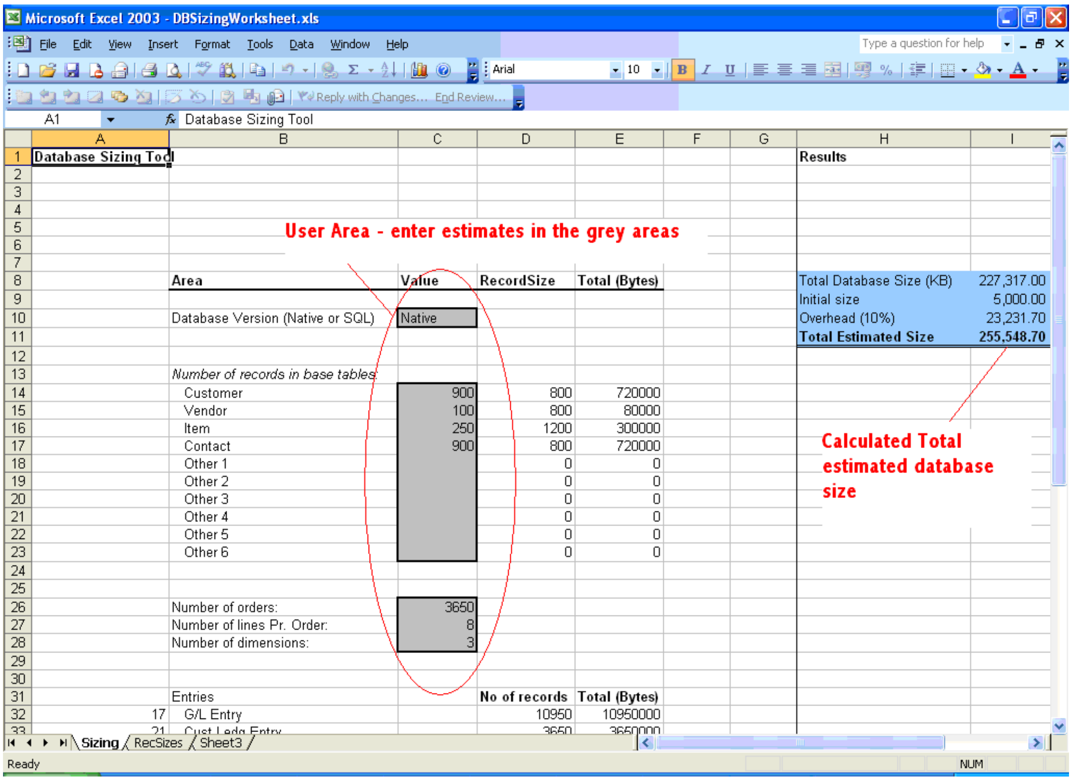
How to Use NDST

The NDST can be used before implementing a new system, or it can be used on an existing system to forecast future database growth.

Estimating Database Size of a New System

To estimate the appropriate database size for a new system, you only need to use the `DBSizingWorksheet.xls` spreadsheet.

1. Open `DBSizingWorksheet.xls`:


Microsoft Excel 2003 - DBSizingWorksheet.xls
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial 10
Reply with Changes... End Review...
A1
Database Sizing Tool
User Area - enter estimates in the grey areas
Calculated Total estimated database size
Sizing RecSizes Sheet3
Ready
NUM

	A	B	C	D	E	F	G	H	I
1	Database Sizing Tool							Results	
8		Area	Value	RecordSize	Total (Bytes)			Total Database Size (KB)	227,317.00
9								Initial size	5,000.00
10		Database Version (Native or SQL)	Native					Overhead (10%)	23,231.70
11								Total Estimated Size	255,548.70
13		*Number of records in base tables*							
14		Customer	900	800	720000				
15		Vendor	100	800	80000				
16		Item	250	1200	300000				
17		Contact	900	800	720000				
18		Other 1		0	0				
19		Other 2		0	0				
20		Other 3		0	0				
21		Other 4		0	0				
22		Other 5		0	0				
23		Other 6		0	0				
26		Number of orders:	3650						
27		Number of lines Pr. Order:	8						
28		Number of dimensions:	3						
31		Entries		No of records	Total (Bytes)				
32	17	G/L Entry		10950	10950000				
33	21	Cust Ledg Entry		3650	3650000				

Database Sizing Worksheet 1

2. In the grey column, enter your estimate of the size that the table should have. In this example, the estimate is for 900 customers, 100 vendors and so on. They place an estimated 3650 orders (Purchase and Sales) generating an average of 8 lines each).

The factors you can enter are:

Database Version	Navision Database Server or SQL Server. The tool does not allow other values.
Customer	The number of customers you expect.
Vendor	The number of vendors you expect.

Item	The number of items you expect.
Contact	The number of contacts you expect.
Other1 .. Other2	These lines are user defined. For example, if you want to include the size of your ***Fixed Assets*** table, you can enter a line here called Fixed Assets and then enter the number of Fixed Assets you expect. If you want to enter any user defined tables, then you also have to edit the underlying factors as described later.
Number of orders	The number of orders you expect. This includes sales as well as purchase orders.
Number of lines pr order	The average number of lines in an order.
Number of dimensions	The average number of dimensions in an order.

The spreadsheet calculates the total size of the data based on the numbers you entered.

It then adds two extra values:

- Initial Size: Even an empty database has a size.
- Overhead: The spreadsheet calculates the database growth based on the amount of data in the most commonly used tables. To account for other tables and fields, for example, comments, G/L Accounts, Budget Entries, To-dos, and so on, it adds an estimated overhead. This is set to 10%.

Changing the Underlying Factors

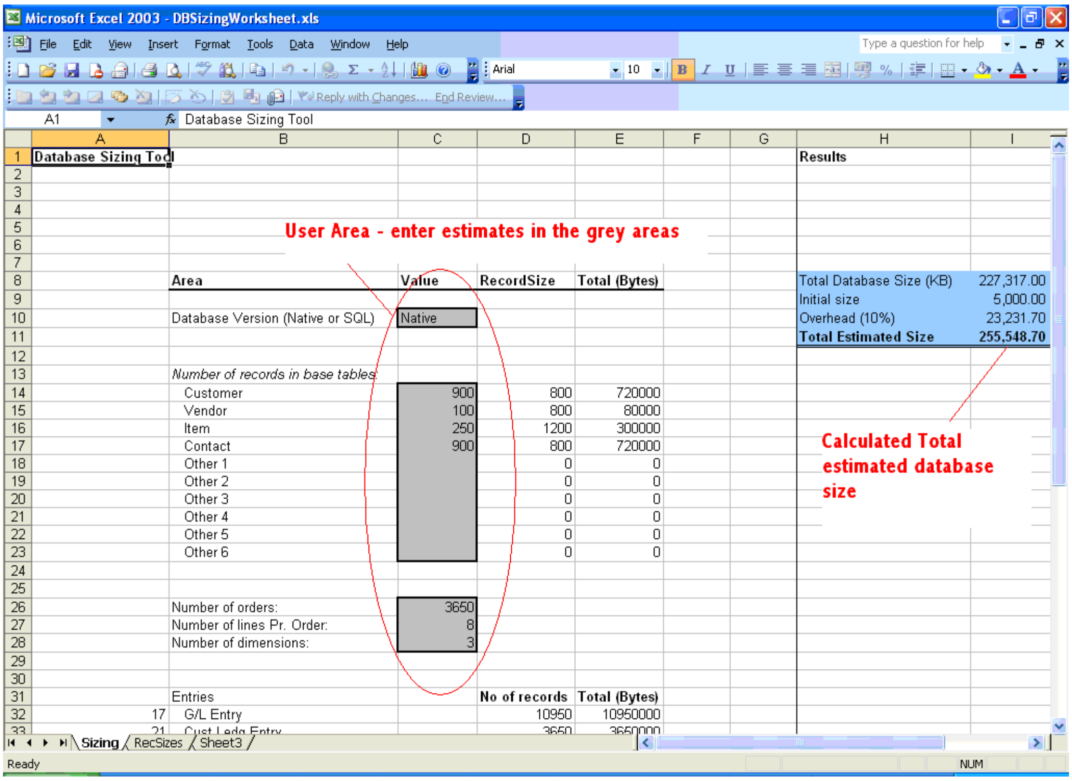
NDST uses estimated record sizes to calculate database growth. You can change these estimates. Reasons for changing the estimates could be, that you have experienced that the real record size for some table is different to the ones used in NDST, or you might have customized some tables.

The `DBSizingWorksheet.xls` consists of two worksheets: ***Sizing*** and ***RecSizes***. Both worksheets are protected, except for the grey cells. Before you can modify anything outside of the grey cells, you must unprotect the worksheet:

To unprotect the worksheet:

In Excel, click Tools, Protection, Unprotect Sheet.

The ***RecSizes*** sheet contains the estimated average size of a record in all the tables that are used to estimate database growth.


Microsoft Excel 2003 - DBSizingWorksheet.xls
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial
10
Reply with Changes... End Review...
A1
Database Sizing Tool
Sizing
RecSizes
Sheet3
Ready
NUM

Database Sizing Tool

User Area - enter estimates in the grey areas

	Area	Value	RecordSize	Total (Bytes)
	Database Version (Native or SQL)	Native		
	Number of records in base tables			
	Customer	900	800	720000
	Vendor	100	800	80000
	Item	250	1200	300000
	Contact	900	800	720000
	Other 1		0	0
	Other 2		0	0
	Other 3		0	0
	Other 4		0	0
	Other 5		0	0
	Other 6		0	0
	Number of orders:	3650		
	Number of lines Pr. Order:	8		
	Number of dimensions:	3		
	Entries		**No of records**	**Total (Bytes)**
17	G/L Entry		10950	10950000
21	Cust Ledg Entry		3650	3650000

Results

Total Database Size (KB)	227,317.00
Initial size	5,000.00
Overhead (10%)	23,231.70
Total Estimated Size	**255,548.70**

Calculated Total estimated database size

Database Sizing Worksheet 2

At the top is the base tables, and lower down you have the dynamic tables.

Because record size often differs depending on the database option, the average size is specified for both Navision Database Server, and SQL Server. Average Size (Bytes) is the average size of *one* record.

For dynamic tables, you also have the columns **Records pr order**, and **Records pr line**. These specify how many records are created when you post a sales or purchase order. For instance, we estimate that posting one order creates three G/L Entries and three Item Ledger Entries pr. line. These values can depend on the settings in Navision. For instance, if you have activated Automatic Cost Posting, then one order will create at least two additional G/L Entries, and “Discount Posting” in Sales & Receivables setup will also affect the number of entries created per order. Finally, of course customizations can affect the number of records created.

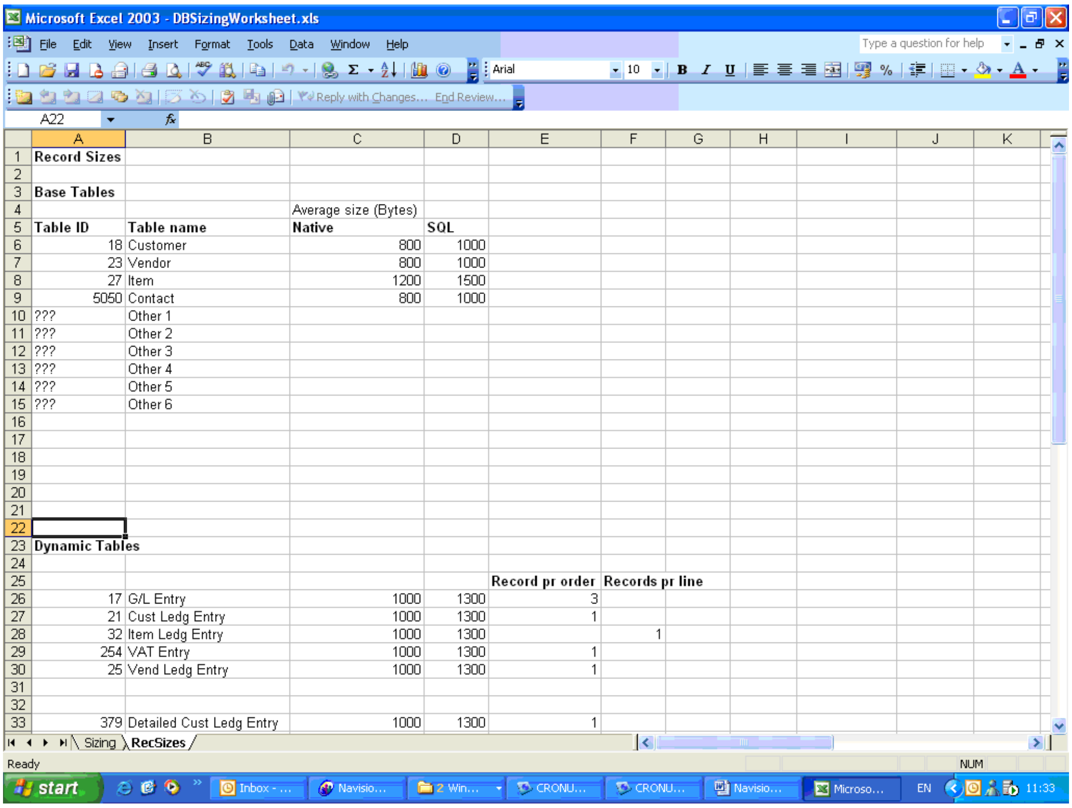
Warning
You can change the values on this sheet, but do not move the existing rows and columns. The formulas on the sizing sheet refer to many of these cells. If you move them, the calculations will probably be wrong.

If you specify any user-defined tables (Other 1, Other 6), then you must also specify the record size on the ***RecSizes*** sheet.

Finally, on the right hand side of the ***Sizing*** sheet (in the blue square), you can modify the Initial Size and the Overhead if you wish.

Details

The lower half of the sizing sheet contains calculated values for a number of tables.



	A	B	C	D	E	F
1	**Record Sizes**					
2						
3	**Base Tables**					
4			Average size (Bytes)			
5	**Table ID**	**Table name**	**Native**	**SQL**		
6	18	Customer	800	1000		
7	23	Vendor	800	1000		
8	27	Item	1200	1500		
9	5050	Contact	800	1000		
10	???	Other 1				
11	???	Other 2				
12	???	Other 3				
13	???	Other 4				
14	???	Other 5				
15	???	Other 6				
16						
17						
18						
19						
20						
21						
22						
23	**Dynamic Tables**					
24						
25					**Record pr order**	**Records pr line**
26	17	G/L Entry	1000	1300	3	
27	21	Cust Ledg Entry	1000	1300	1	
28	32	Item Ledg Entry	1000	1300		1
29	254	VAT Entry	1000	1300	1	
30	25	Vend Ledg Entry	1000	1300	1	
31						
32						
33	379	Detailed Cust Ledg Entry	1000	1300	1	

Database Sizing Worksheet 3

These details show how much each of the tables are expected to grow, based on the values you entered.

The detailed values are calculated, and should not be overwritten manually. They are calculated based on the Average Size and Records pr order/line, which are set up in the ***RecSizes*** sheet.

Forecasting Database Growth for an Existing System

The NDST also contains functionality that allows you forecast the growth rate of an existing system. However, you can also use it to estimate the size of a new system by finding out how much a single operation causes the database to grow and then multiplying that by the number of expected operations.

It works by taking snapshots of the size of the database at different times and comparing the two snapshots.

The NDST contains two files: `NDST.fob` and `SpaceUsed.sql`.

- `NDST.fob` can be used on both Navision Database Server and on the SQL Server Option.
- `SpaceUsed.sql` can only be used to get more detailed information about the SQL Server Option.

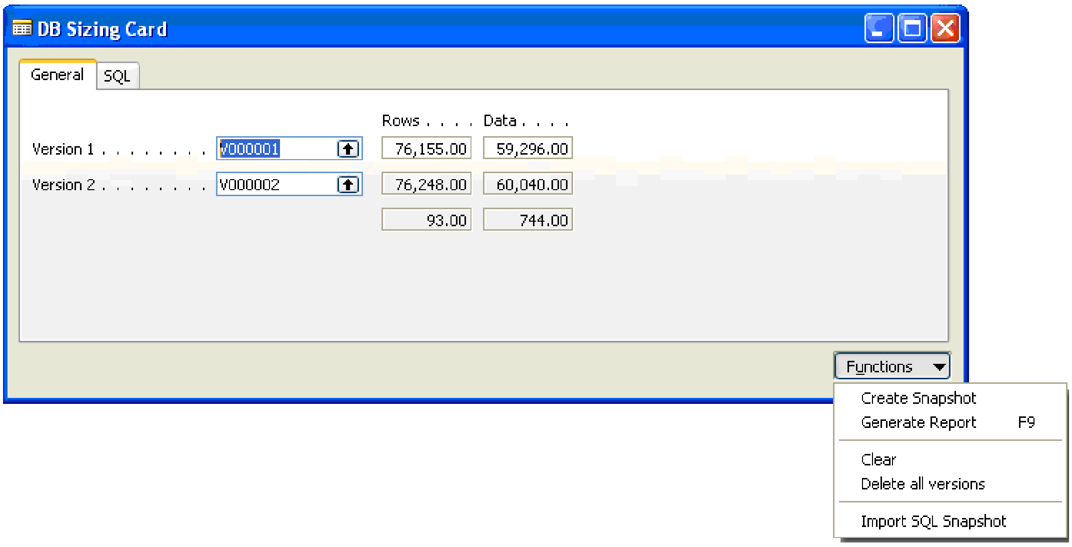
`NDST.fob` contains 8 objects:

Type	No.	Name	Version List
Table	72000	***sp_spaceused file Line***	NDST
Table	72001	***Space Used Version***	NDST
Form	72000	***Space Used Version Lines***	NDST
Form	72001	***Space Used Version List***	NDST
Form	72005	***DB Sizing Card***	NDST
Codeunit	72000	***Snapshot from files (SQL)***	NDST
Codeunit	72001	***Make Spreadsheet***	NDST
Codeunit	72003	***Snapshot from DB Info***	NDST

Creating Snapshots to Evaluate Database Growth

To create snapshots and estimate database growth:

1. In Navision, open the **Object Designer** and click File, Import to import `NDST.fob`. You can use both Navision Database Server and the SQL Server Option.
2. In the **Object Designer**, run form 72005, **DB Sizing Card**.



		Rows	Data
Version 1	V000001	76,155.00	59,296.00
Version 2	V000002	76,248.00	60,040.00
		93.00	744.00

Database Sizing Card

1. Click Functions, Clear to clear the form.
2. Click Functions, Create Snapshot. This creates a new version, and enter the new version number as Version 1.
3. Close the **DB Sizing Card** window.
4. Create a sales order and post it (or perform any other activity that will create new records).
5. Open the **DB Sizing Card** and click Functions, Generate Snapshot again. This creates a new version, and automatically specifies this new version as Version 2. With Version 1 and Version 2 specified on the card, you can see the number of new records and the amount of new data (KB) that has been generated between the two different versions.
6. To get more details, click Functions, Generate Report. This creates an Excel Spreadsheet that contains details about each table that was involved in the transactions that you performed.

The DB Sizing Card

All the functionality in the objects that were imported in the `NDST.fob` can be run from the **DB Sizing Card**. The Functions button gives you access to a number of functions:

Create Snapshot:

This creates a snapshot of the database. Every time you create a new snapshot, NDST automatically creates a new version. On the **DB Sizing Card**, you can select one or two versions, then immediately see the number of rows, and the amount of data in that version, as well as the difference between the two versions.

By comparing the two versions (snapshots), you can see how much the database grew in the time that elapsed between the two snapshots. This enables you to forecast database growth based on such things as:

- Running a batch job
- Posting one order
- Synchronizing contacts and so on
- Growth per month / quarter / and so on

Generate Report:

This creates a report that gives you a more detailed overview than the **DB Sizing Card** does. It creates a new Excel workbook which shows you the changes in each individual table. The workbook contains one line for each table which has been changed (and displays either the number of records added or the size it has grown by).

It generates two spreadsheets:

- **Changes**, which shows you data for version 2 as well as the changes between version 1 and 2.
- **Details**, which shows you version 1 and version 2.

Finally, at the top of the ***Changes*** sheet, you see a summary.

If you have problems making this Excel integration work, please refer to the Troubleshooting section.

Clear:

This clears the contents of the form. It does not delete any data it only blanks out the form. If you clear the form before creating a snapshot, then the form will automatically enter the newly created version as Version 1 (or Version 2 if Version 1 is already specified).

Delete all versions:

Deletes all the versions (all the snapshots) and clears the form.

Import SQL Snapshot:

This is another way of creating a snapshot. It only applies to the SQL Server Option, and gives you a bit more information than the Create Snapshot function. This means that with the SQL Server Option, you can choose between using Create Snapshot and Import SQL Snapshot. However, you can only compare like with like.

The Import SQL Snapshot function splits the amount of space used into Data and Indexes. The Create Snapshot function shows Data and Indexes as one field. So if you compare a version created by “Create Snapshot” with a version created by “Import SQL Snapshot”, the result will not be very precise.

The advantages of the Import SQL Snapshot function compared to the Create Snapshot function are:

- It displays separate values for the amount of database space used by the data and the indexes.
- You get individual results for the SIFT tables.
- For example: Create Snapshot creates one record for the, **G/L Entry** table. This record contains the accumulated values for the **G/L Entry** table and it’s two SIFT tables **17$0** and **17$1**. The Import SQL Snapshot function creates three individual records – one for each table.

You must run a SQL query from Query Analyzer before you can run the Import SQL Snapshot function.

To run this query:

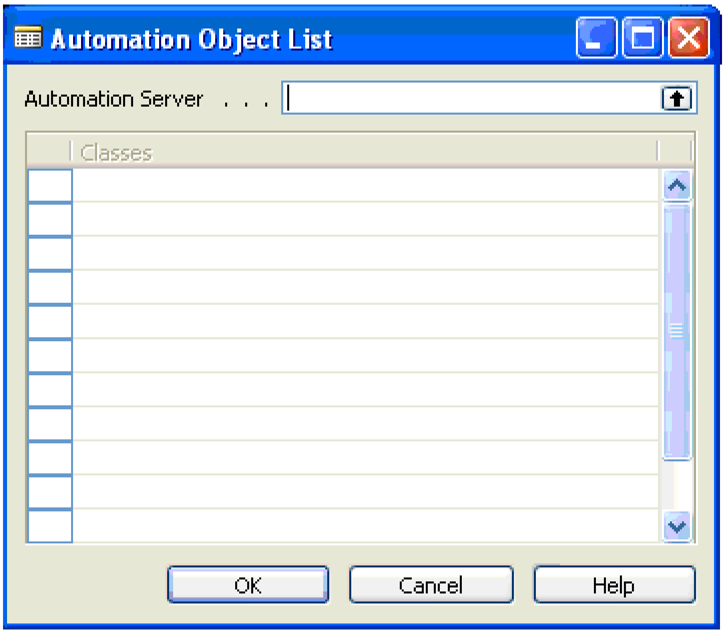
1. Open Query Analyzer.
2. Log in, and select your Navision database.
3. Open the script `SpaceUsed.sql`. Set Execute Mode to Results to File… (Ctrl+Shift+F), then execute the query. This batch uses the stored procedure `sp_spaceused` to export the details for each table.
4. In Navision open the **DB Sizing Card** window and on the SQL tab, specify the file that was created in Query Analyzer.
5. Run Import SQL Snapshot.

Troubleshooting

The Generate Report function was developed using Microsoft Office 2003 but it has also been tested with Office 2000. If you have problems making this work:

1. Make sure you have Office installed.
2. Even though Office 2003 if backwards compatible, previous versions may have problems. If this is the case, you may have to check the design of codeunit 72001, **Make Spreadsheet**.
3. Click View, Globals, and check the four Automation variables – they will most likely say *Unknown Automation Server*. Even if they say *Unknown Automation Server*, it may still work. If it does not work, try to re-set the variables to the installed version of Office.

For each Automation variable, click the AssistButton in the **SubType** field to open the ***Automation Object List*** window:



Automation Object List

4. Click the AssistButton in the **Automation Server** field to open the ***Automation Server List*** window and select the latest version of Office in the list.

5. In the ***Automation Object List*** window, select the corresponding class for each of the variables:

   - XApp.Application
   - XBook_Workbook
   - XSheet_Worksheet
   - xRange.Range

   Note
   For Workbook and Worksheet, select the classes that are preceded by an underscore.

6. Save and close and compile the codeunit.

Factors that Affect Database Growth

As mentioned earlier, a large number of customer specific factors can affect the pace at which the customer's database grows.

This means, that the results generated by the `DBSizingWorksheet.xls` tool must not be taken at face value. This section discusses some of the most common factors that must be considered and provides guidelines on how to deal with them.

As explained in the section "Changing the Underlying Factors" you can change the factors that the `DBSizingWorksheet.xls` tool uses to calculate the estimated database size. For instance, if you activate Automatic Cost Posting in your application, Navision will create 5 instead of 3 G/L entries for each order you post. You can easily update `DBSizingWorksheet.xls` to reflect this.

Use NDST to Make Customer Specific Estimates:

There is only one way to find out exactly how much any particular action (for example, posting an order) affects a database, and that is to run the action and see what happens. This is one of the things that NSDT can help you do – see the section “The DB Sizing Card.”

Example:

1. Set up Navision in the same the way as it is for the customer, and create a typical order.
2. Create a snapshot, post the order and then create another snapshot.
3. Run “Generate Report”, and you can now see exactly how many entries were created in each table.
4. Now you can update the `DBSizingWorksheet.xls` tool with these values.

Typical Factors that Affect Database Growth:

Apart from the most obvious factors, such as having the ***Contact*** table synchronized with the ***Customer*** table, a number of other typical but less obvious factors will have an effect on database growth:

Number of Posting Groups

Posting an order creates two G/L entries per *Combination* of Gen. Business Posting Group and Gen. Product Posting Group *plus* one balancing entry. If, for example you post an order with two items which belong to separate Gen. Product Posting Groups, Navision will create 5 G/L entries.

Part Shipping:

Every time you part ship an order, Navision creates posted documents, entries, dimension entries and so on. This means that if you part ship/invoice orders, the database will grow more per order than it will if you only ship and invoice full orders.

Once you have a good idea of how many records are created when an order is posted, you can update the `DBSizingWorksheet.xls` tool to get a more precise estimate of database growth.

Modifying the Key in an Index

If you modify a key in an index that you use for browsing through the records, the read ahead mechanism is switched off. We therefore recommended that you restore the key values just before the `NEXT` statement, or use one record variable for browsing through the set and another one for modifying the records.

Example of bad code

```
Customer.SETCURRENTKEY(“Currency Code”);
Customer.SETRANGE(“Currency Code”,’GBP’);
IF Customer.FIND(‘-‘) THEN
  REPEAT
    Customer.“Currency Code” := ’EUR’;
    Customer.MODIFY;
    //The above modifies the key value
    UNTIL Customer.NEXT = 0;
```

Example of good code 1 (restoring key value)

```
  Customer.SETCURRENTKEY(“Currency Code”);
  Customer.SETRANGE(“Currency Code”,’GBP’);
  IF Customer.FIND(‘-‘) THEN
    REPEAT
      Customer.“Currency Code” := ’EUR’;
      Customer.MODIFY;
      Customer.“Currency Code” := ’GBP’;
      //The above restores the original value
      //of the key, just before NEXT statement
    UNTIL Customer.NEXT = 0;
```

Example of good code 2 (restoring key value)

```
  Customer.SETCURRENTKEY(“Currency Code”);
  Customer.SETRANGE(“Currency Code”,’GBP’);
  IF Customer.FIND(‘-‘) THEN
    REPEAT
      Customer2 := Customer;
      Customer.“Currency Code” := ’EUR’;
      Customer.MODIFY;
      Customer.“Currency Code” := Customer2.“Currency Code”;
      //The above restores the original value
      //of the key, just before NEXT statement
    UNTIL Customer.NEXT = 0;
```

Example of good code 3 (separate variables for browsing)

```
  Customer1.SETCURRENTKEY(“Currency Code”);
  Customer1.SETRANGE(“Currency Code”,’GBP’);
  IF Customer1.FIND(‘-‘) THEN
    REPEAT
      Customer2 := Customer1;
      Customer2.“Currency Code” := ’EUR’;
      Customer2.MODIFY;
    UNTIL Customer1.NEXT = 0;
```

Form Design and Performance

This section describes how to create forms that perform optimally in Navision. You should pay particular attention to these three areas:

- SIFT
- *SourceTablePlacement* property
- Find As You Type feature

SIFT

SIFT technology is a great feature that allows you to retrieve and maintain cumulated sums. However, it has a performance overhead.

The best practices when using SIFT are:

Display on Demand

Do not place (or at least minimize the number of) FlowFields on normal forms, such as the ***Customer Card***, ***Item Card***, and so on. Use special forms such as ***Customer Statistics***, ***Item Statistics***, ***Customer Entry Statistics***, ***Item Entry Statistics***, ***Customer Sales***, ***Item Turnover***, and so on. The basic principle is to display these FlowFields on demand rather than by default when the user is not even interested in the information provided.

Never place FlowFields on List Forms

When you place FlowFields on list forms you delay the retrieval time because each line must calculate the FlowField value(s). You should also be aware that if you hide a FlowField on a form, C/SIDE still calculates the FlowField value. You must delete the field from the form to avoid the overhead.

SourceTablePlacement Property

The *SourceTablePlacement* property is used to tell the system which record it should display when a particular form is opened. The default value is ‘Saved’, which means that C/SIDE tries to position the cursor on the record that was last displayed. If the form is based on a big table, this has a large performance overhead (as a rule of thumb a big table contains more than 1 million records). The best practice is to find these forms and change the *SourceTablePlacement* property to ‘First’ or ‘Last’. Remember that tables grow, so try to estimate how many records they will contain in a year or two.

Find As You Type Feature

The *Find As You Type* feature allows you to type over a non-editable field in a list form, and while you are typing C/SIDE ‘jumps through’ the underlying table. This feature can have a massive impact on performance because it is impossible to predict the number of times that the cursor must reposition itself in the displayed table. We therefore recommended that this feature is disabled in large installations. You can use the Find (Ctrl+F) or Field/Table Filter (F7 or Ctrl+F7) features instead, and train the users to always change the key (index) to an index which includes the field they are searching and/or filtering on.

To disable the *Find As You Type* feature:

1. Click File, Database, Alter to open the ***Alter Database*** window.
2. Click the ***Options*** tab.
3. Make sure that the “Allow Find As You Type” is unchecked.

To minimize the dissatisfaction that can be caused by disabling the feature after users have gotten used to using it, we recommend that you always disable it when you create a new database and only enable it when requested.

Locking and Deadlocks

This section outlines some best practices and techniques that you can use to avoid what might otherwise be unpreventable locks and deadlocks. It must be stressed that application developers should focus on performance before looking into locks because improving performance might minimize locking.

Deadlocks

Deadlock situations arise when two processes have locked data and each process cannot release the lock until the other process has released its lock. The following logic would generate a deadlock:

- Session A locks table A
- Session B locks table B
- Session A tries to lock table B (but is blocked and must wait for session B to finish)
- Session B tries to lock table A (but is blocked and must wait for session A to finish)
- Result – Deadlock.

SQL Server resolves the deadlock by terminating one of the transactions. However detecting the deadlock can take some time and consumes valuable resources.

Preventing Deadlocks

There are two well-known approaches to preventing deadlocks;

- Always lock tables in the same order.

  Application developers must agree on the order in which they lock tables. For example, when processing a sales order we always lock the following tables in this order: table 39, ***Purchase Line***, table 38, ***Purchase Header***, table 37, ***Sales Line*** and table 36 ***Sales Header***.

- Lock an agreed “master resource” first.

  It might be impossible or just too complicated to agree the locking order. As a workaround, developers must agree that they will always lock a *master resource* before they process any tables (in any order). An example of this approach can be found in Codeunit 80, **Sales-Post**, which locks the last record in the ***G/L Entry*** table before processing anything else:

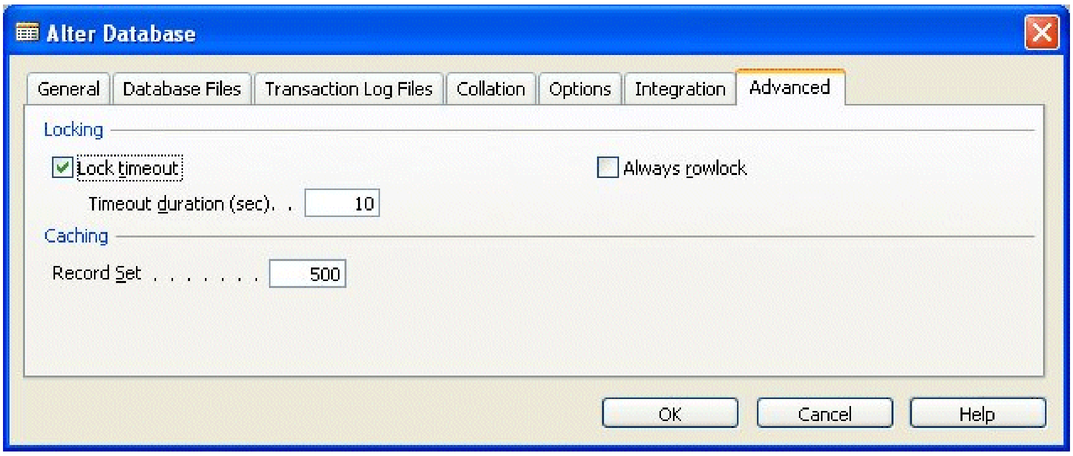
  ```
  GLEntry.LOCKTABLE;
  IF GLEntry.FIND('+') THEN;
  ```

  However, it must be stressed that even though this mechanism is used, the first principle (locking tables in the same order) is also used.

LOCKTIMEOUT

Navision contains a database property that allows you to specify whether or not a session will wait to place a lock on a resource that has already been locked by another session. You can let the session wait indefinitely or you can specify how long the session will wait before timing out and failing to place a lock. This is the SQL property *LockTimeout* that is exposed in Navision.

To set this property in Navision, click File, Database, Alter and click the ***Advanced*** tab:



☑ Lock timeout

Timeout duration (sec). . 10

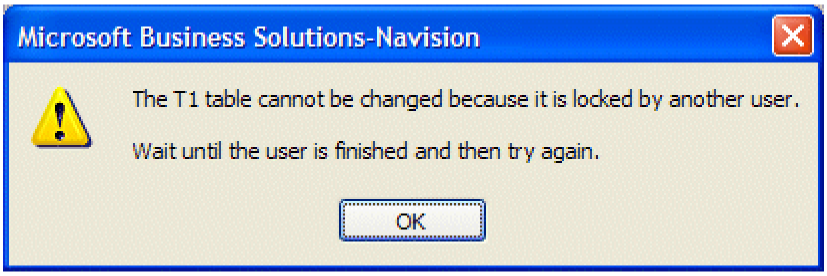
☐ Always rowlock

Record Set 500

Alter Database – Advanced tab

This property is designed to improve performance by ensuring that the users and transactions that are waiting to place locks don't have to wait indefinitely.

After a session has waited the specified number of seconds and failed to place a lock the user receives a message informing them that the lock could not be placed:



The T1 table cannot be changed because it is locked by another user.

Wait until the user is finished and then try again.

Lock Timeout Message

The user can then perform another task instead of waiting indefinitely for the lock to be released.

This is a database property and once set it remains set until you remove the check mark in this window. However, you can also use the C/SIDE function `LOCKTIMEOUT` to temporarily enable or disable this property in the application.

Always Rowlock

The ***Advanced*** tab also contains a property called *Always rowlock*.

By default this property is not selected and SQL Server uses its default locking behavior. This can improve performance by allowing SQL Server to determine the best locking granularity.

In Navision 4.00 and earlier, rowlock hints were always issued to SQL Server. With Navision 4.01 and later, Navision takes advantage of the SQL Server locking behavior by default.

When you upgrade to Navision 4.01, the new default functionality might change the behavior of your application and could have an adverse effect on concurrency. If you find this unacceptable, you should select this property and force Navision to issue rowlock hints.

Minimizing the Duration of Locks

You should always try to place a lock as late as possible and lock for as short a time as possible. If you have a lot of records to process, give other users a chance to access the system as well, for example when if you are importing 1000 orders from a web site, you could create a loop which would:

- import 5 records (use `LOCKTABLE` commands in the right order, see deadlock section later)
- write a log
- `COMMIT` the changes
- `SLEEP` for 1 minute (just an example)
- `SELECTLATESTVERSION`
- repeat this sequence

This enables other users to access the database in the slots of time when the system is not locked by the import routine.

Never Allow User Input during a Transaction

Consider the worst scenario; after you have started a transaction (locked the records) the system displays a form so that you can select some options, answer yes/no, click OK button, or some such user interaction. However, before you do this, you leave your desk to fetch a cup of coffee and get distracted by a colleague who needs some assistance and in the meantime all the other users are blocked from using the same or possibly some other functionality.

C/SIDE automatically detects this when you use, for example, the `Form.RUNMODAL` command after you have locked a resource. You would get the following error message:

The following C/AL functions can be used only to a limited degree during write transactions (because one or more tables will be locked).

`Form.RunModal()` is not allowed in write transactions.

`CodeUnit.Run()` is allowed in write transactions only if the return value is not used. For example, `OK := CodeUnit.Run()` is not allowed.

`Report.RunModal()` is allowed in write transactions only if `RequestForm = FALSE`. For example, `Report.RunModal(...,FALSE)` is allowed.

`DataPort.RunModal()` is allowed in write transactions only if `RequestForm = FALSE`. For example, `DataPort.RunModal(...,FALSE)` is allowed.

Use the `COMMIT` function to save the changes before this call, or structure the code differently.

Unfortunately C/SIDE doesn't detect other commands which can be used for user input such as `Window.INPUT` and `IF [NOT] CONFIRM` constructs.

Application Setup

It is also worthwhile reviewing some of the application setup.

For example, if you enable the **Automatic Cost Posting** field in the ***Inventory Setup*** table, every stock transaction automatically generates an entry in the General Ledger.

Another example is the **Update on Posting** field in the ***Analysis View*** table, which tells the system to generate analysis view entries for the view when you post to the General Ledger.

It could be a disaster if both of these fields are enabled because when you post a sales order or a purchase order with items, the system will post (lock) to the ***Item Ledger*** table as well as to the ***General Ledger*** table and it will update (lock) the ***Analysis View*** table.

Overnight Processing

You might consider performing some heavy tasks outside of normal working hours (overnight) by using the Navision CRM Job Scheduling functionality. Batch jobs such as *Adjust Cost - Item Entries* and *Post Inventory Cost to G/L* are typical examples.

Tools

The *Performance Troubleshooting Guide* manual contains tools and instructions that explain how to analyze the record locking order of transactions and compare them with the locking order rules. The locking order rules must be defined in the ***Locking Order Rules*** form. The *Performance Troubleshooting Guide* contains only a sample locking rule.

The most practical way of identifying the locking order rules is to focus on the key procedures and document the order in which they lock the various tables.

To identify the locking order rules for a procedure:

1. Start the Client Monitor.
2. Perform an isolated task (procedure).
3. Stop the Client Monitor.
4. Run the ***Transactions*** form.
5. Run the ***Transaction Locking Order*** form.
6. Use the locking order as manual input for the ***Locking Order Rules*** form.

After you have defined the locking order rules, you can perform the same procedure for the next task. You can then check that the second procedure does not violate the locking order rules that you identified for the first procedure.

You must:

1. Start the Client Monitor.
2. Perform an isolated task.
3. Stop the Client Monitor.
4. Run the ***Transactions (Locking Rules)*** form.
5. In the ***Transactions (Locking Rules)*** window, you must check the **Locking Rule Violations** field.
6. Click Transaction, Locking Rules Violations to check the violation.
7. In ***Locking Rules Violations*** window, you can see if and how your transaction violated the locking order rules. You can now fix the code, amend the locking order rules or decide that the probability of the two processes running concurrently is minimal and ignore the violation.

Needles to say, you can run the Code Coverage tool during the test and this will help you identify the code that is causing the conflicts. However the Code Coverage tool is quite a ‘heavy’ tool and should not be used for multiple transactions, or run for more than 10 minutes. It takes a long time to perform complex procedures when the Code Coverage tool is running and it also takes some time to process the information in the tool afterwards. On the other hand, if you only use the Client Monitor without the Code Coverage tool you will normally get enough information to identify the code that is causing the conflict.

In summary, you can run the Client Monitor on its own in a real-life environment without experiencing any major decrease in performance and still get the information needed to resolving incorrect locking order problems.

Important
You must perform all these steps for both server options because the locking order could look very different in the two options.

The only way to lock a record on SQL Server is to actually read it, while on Navision Database Server you can use the `LOCKTABLE` command to dictate and define the locking order. The `LOCKTABLE` command on SQL Server Option does not do anything except flag internally that locks will be placed later when records are read.

Using PINTABLE for Small Hot Tables

Some Navision application tables might be:

- small
- temporary
- accessed all the time (hot)
- over indexed

A typical example could be the ***Warehouse Activity Line*** table. In a busy warehousing application, Picks and Put-Away documents are created/registered (deleted) frequently in this table. However, it would not exceed more than several hundred records, and would have many indexes that are similar (use `sp_helpindex` to see the standard application indexes).

The following adjustments can improve performance:

- Use the *MaintainSIFTIndex* property to disable all the SIFT tables that are based on the source table.
- Use the *MaintainSQLIndex* property to disable all but a few of the SQL Server Indexes.
- Study the changes and if reading the tables is still too slow, pin the table to memory using `DBCC PINTABLE`.

Note
Pinning the table doesn't mean that it is read to memory automatically. You can ensure that the table is cached in memory by scheduling a job that reads the entire table (using `SELECT *`…) to run every time SQL Server starts.

For further information about the `DBCC PINTABLE` command, see SQL Server Books Online.

The following information is taken from SQL Server Books Online.

DBCC PINTABLE

Marks a table to be pinned, which means SQL Server does not flush the pages for the table from memory.

Syntax

```
DBCC PINTABLE ( database_id , table_id )
```

Arguments

database_id

This is the database identification (ID) number of the table to be pinned. To determine the database ID, use the `DB_ID` function.

table_id

This is the object identification number of the table to be pinned. To determine the table ID, use the `OBJECT_ID` function.

Remarks

`DBCC PINTABLE` does not read the table into memory. As the pages from the table are read into the buffer cache by normal Transact-SQL statements, they are marked as pinned pages. SQL Server does not flush pinned pages when it needs space to store a new page. SQL Server still logs updates to the page and, if necessary, writes the updated page back to disk. SQL Server does, however, keep a copy of the page in the buffer cache until the table is unpinned with the `DBCC UNPINTABLE` statement.

`DBCC PINTABLE` is useful for keeping small, frequently used tables in memory. The pages for the small table are read into memory once and then all future references to this data do not require a disk read.

Important

Although `DBCC PINTABLE` can give performance improvements, it must be used with care. If a large table is pinned, it can start using a large portion of the buffer cache and not leave enough cache to service the other tables in the system adequately. If a table larger than the buffer cache is pinned, it can fill the entire buffer cache. To unpin the table, a member of the *sysadmin* fixed server role must shut down SQL Server and restart SQL Server. Pinning too many tables can cause the same problems as pinning a table larger than the buffer cache.

Warning

Pinning tables should be carefully considered. If a pinned table is larger, or grows larger, than the available data cache, the server may need to be restarted and the table unpinned.

DBCC execution completed. If DBCC printed error messages, contact your system administrator.

Permissions

`DBCC PINTABLE` permissions default to members of the *sysadmin* fixed server role and are not transferable.

Examples

This example pins the **Authors** table in the Pubs database:

```
DECLARE @db_id int, @tbl_id int
USE pubs
SET @db_id = DB_ID('pubs')
SET @tbl_id = OBJECT_ID('pubs..authors')
DBCC PINTABLE (@db_id, @tbl_id)
```

SQL Server Maintenance

This section focuses on some of the most important maintenance tasks on SQL Server. These include updating statistics, defragmenting indexes and optimizing fill factors. You can use a Database Maintenance Plan in SQL Server to manage this, as well as other tasks such as backup, disaster recovery, etc.

Updating SQL Server Statistics

When you create a Navision database on SQL Server, statistical information is created automatically. In order for the SQL Server Option for Navision to function optimally, you must update these statistics regularly.

You can use two commands to update statistics:

- `update statistics [table name]` to update the statistics for a single table.
- `sp_updatestats` to update the statistics for all the tables.

For more information about these functions, see SQL Server Books Online.

Notes:

- Updating the statistics for Navision tables can be a time-consuming task depending on the number of records that the tables contain. The tables that are updated most often are the tables whose statistics must be updated most regularly. Therefore, we recommend that you create a job on SQL Server that performs regular updates of all the tables when the system is not in use. If performing the update is still too time-consuming, you can divide it into smaller jobs that update the statistics for some of the tables. Creating a SQL job allows you to automate the task and generate reports containing details about the success of the job.
- If you update statistics regularly using SQL jobs, turn off the *Auto Update Statistics* and *Auto Create Statistics* options otherwise you may have unpredictable behavior when this process starts during normal working hours.

  To turn off these options:

  1. Open SQL Server Enterprise Manager and select the Navision database.
  2. Right–click the database and select Properties.
  3. In the ***Properties*** window, click the ***Options*** tab, and make sure that the *Auto Update Statistics* and *Auto Create Statistics* options are not selected.
- If you use SQL jobs to update statistics regularly, make sure that when you create, modify or delete a table or index in Navision your SQL jobs are updated accordingly.

Index Fragmentation

The following is an excerpt from SQL Server Books Online:

When you create an index in the database, the index information used by queries is stored in index pages. The sequential index pages are chained together by pointers from one page to the next. When changes are made to the data that affect the index, the information in the index can become scattered in the database. Rebuilding an index reorganizes the storage of the index data (and table data in the case of a clustered index) to remove fragmentation. This can improve disk performance by reducing the number of page reads required to obtain the requested data.

There are a number of alternative ways to reduce the fragmentation of an index, the most common being:

- *`DBCC INDEXDEFRAG`*
- *`DBCC DBREINDEX`*

Unlike `DBCC DBREINDEX` (or the index building operation in general), `DBCC INDEXDEFRAG` is an online operation. It does not hold locks for a long time and therefore does not block running queries or updates. A relatively unfragmented index can be defragmented faster than a new index can be built because the time it takes to defragment is related to the amount of fragmentation. A very fragmented index might take considerably longer to defragment than to rebuild. Furthermore, the defragmentation is always fully logged, regardless of the database recovery model setting (see ALTER DATABASE). Defragmenting a very fragmented index can generate a larger log than even a fully logged index creation. The defragmentation, however, is performed as a series of short transactions and therefore does not require a large log if log backups are taken frequently or if the recovery model setting is SIMPLE.

Whichever method is chosen, it is possible to detect the fragmentation of an index by using the `DBCC SHOWCONTIG` command. It is recommended that indexes are maintained more frequently on volatile tables, perhaps via a scheduled job or a maintenance plan.

In general, if the level of Extent switches is much higher than Extents scanned, or a low scan density is detected, defragmenting or rebuilding the index may be beneficial. See `DBCC SHOWCONTIG` in Books Online for further details, and a sample script that you can use to detect a fragmented index and rebuild it if the specified threshold is exceeded.

Index Defrag Tool

This tool gives you easy access to some SQL Server functionality from within Navision. It contains a simple user interface that allows you to quickly identify the tables in your application that contain fragmented indexes or indexes that need to be rebuilt. It also allows you to generate a report containing details of the tables that you have analyzed.

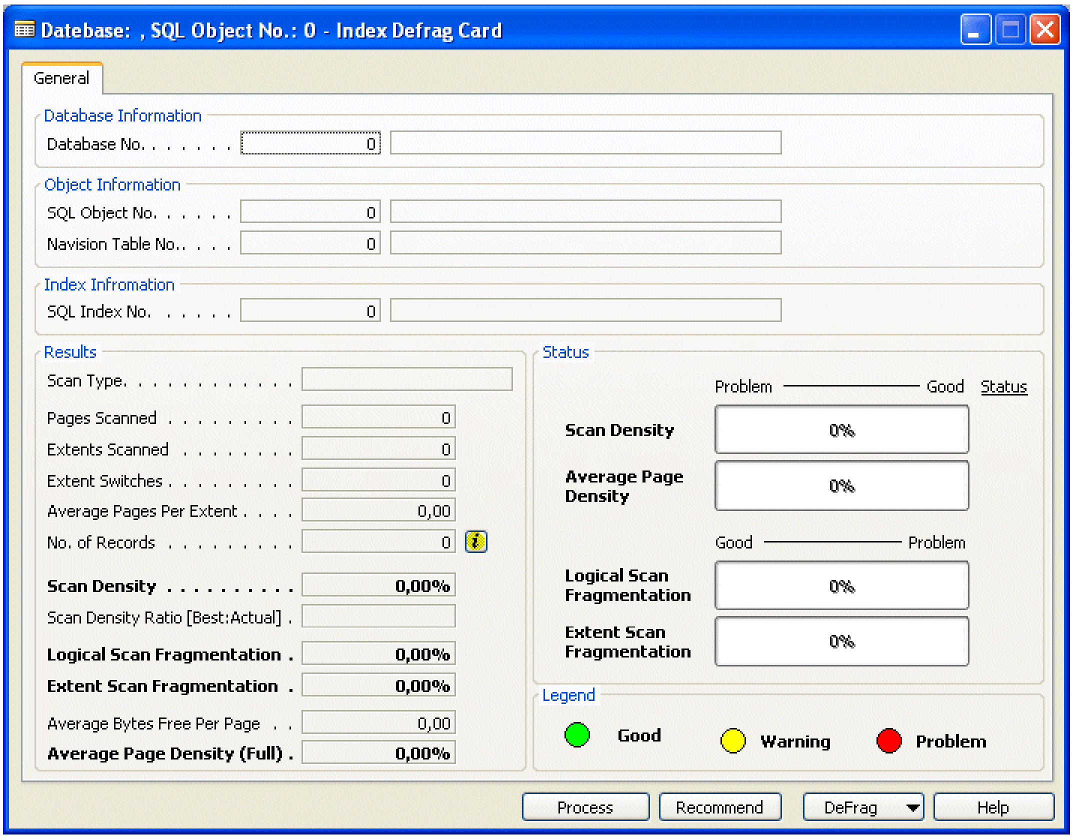
This tool calls a SQL statement called `DBCC SHOWCONTIG` that gathers and displays fragmentation information for the data and indexes of the specified table. For more information about this procedure, see SQL Server Books Online.

Setting Up the Index Defrag Tool

To set up the Index Defrag Tool:

1. In Navision, open the Object Designer and import Index Defrag 50090.fob.

2. Open form 50090, ***Index Defrag Card***:


Datebase: , SQL Object No.: 0 - Index Defrag Card

General

Database Information

Database No. 0

Object Information

SQL Object No. 0

Navision Table No.. . . . 0

Index Infromation

SQL Index No. 0

Results

Scan Type. . . . .

Pages Scanned 0

Extents Scanned 0

Extent Switches 0

Average Pages Per Extent 0,00

No. of Records 0

Scan Density 0,00%

Scan Density Ratio [Best:Actual] .

Logical Scan Fragmentation . 0,00%

Extent Scan Fragmentation . 0,00%

Average Bytes Free Per Page . . 0,00

Average Page Density (Full) . 0,00%

Status

Problem —— Good Status

Scan Density 0%

Average Page Density 0%

Good —— Problem

Logical Scan Fragmentation 0%

Extent Scan Fragmentation 0%

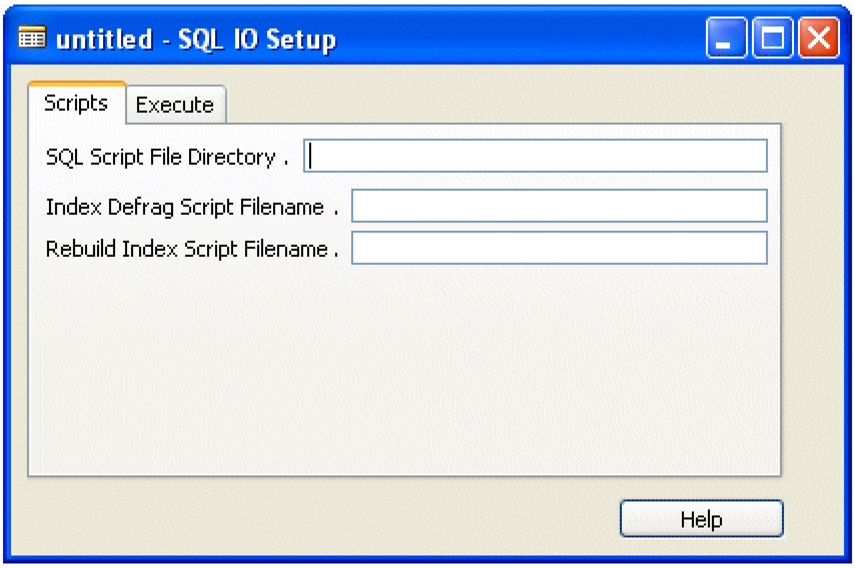
Legend

🟢 **Good** 🟡 **Warning** 🔴 **Problem**

Index Defrag Card

For more information about any of the fields in this window, see SQL Server Books Online.

3. Click Defrag, Setup, File Locations to open the ***SQL IO Setup*** window:


untitled - SQL IO Setup

Scripts Execute

SQL Script File Directory .

Index Defrag Script Filename .

Rebuild Index Script Filename .

SQL IO Setup

4. In the **SQL Script File Directory** field, enter the path to the folder where you want to store the scripts that are generated by this tool.

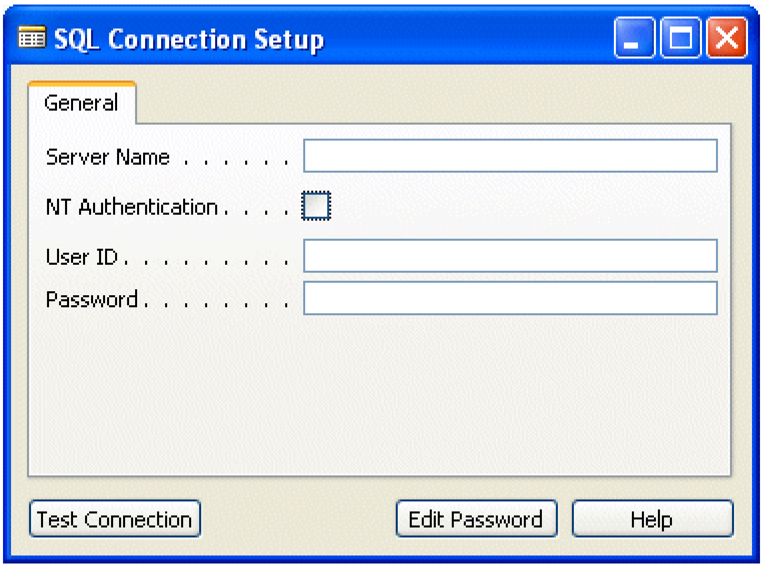
5. In the **Index Defrag Script Filename** field, enter the name that you want to give the script that you use to run the `DBCC SHOWCONTIG` SQL statement in your database. You can edit this file so that it always uses the same database.
6. In the **Rebuild Index Script Filename** field, enter the name that you want to give this file. This is the file that you use to store the information about which tables contain indexes that should be defragmented or rebuilt.
7. Click the ***Execute*** tab and in the **Query Analyzer** field, click the AssistButton and browse to the folder where the `.exe` file for the SQL Server Query Analyzer is stored. This could be on another computer. If Query Analyzer is installed on your local computer, you can just enter *isqlw* – the name of the exe file for Query Analyzer.

Connecting with the Server

The Index Defrag Tool allows you to specify which instance of SQL Server you want to connect with, which kind of authentication you want to use and change the password for you database login.

To connect with an instance of SQL Server:

1. In the ***Index Defrag Card*** window, click DeFrag, Setup, SQL Connection to open the ***SQL Connection Setup*** window:



SQL Connection Setup

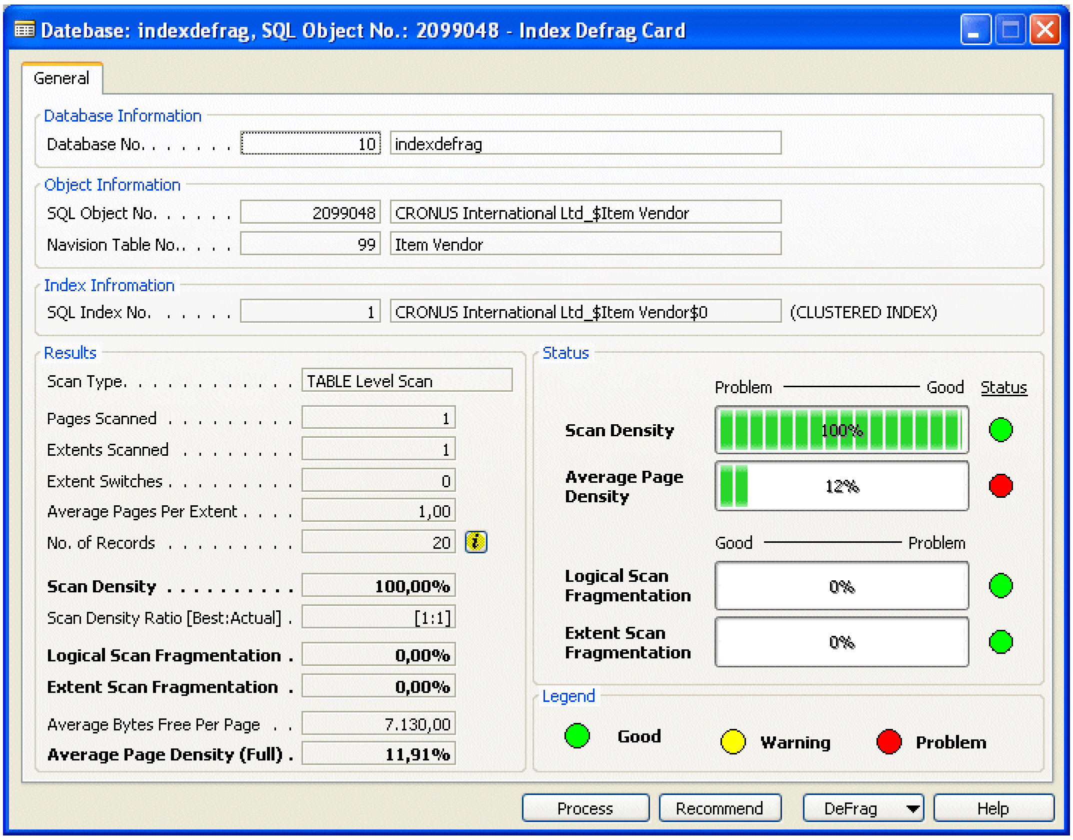
2. In the **Server Name** field, enter the name of the SQL Server that you want to connect with.
3. In the **NT Authentication** field, enter a checkmark if you want to use NT authentication when connecting to the server. If you want to use a database login to connect to the server, enter the user ID and password of the database login that you want to use.
4. If you want to change the password of a database login, click Edit Password.
5. Click Test Connection to see whether or not you can connect with the server.

Running the Index Defrag Tool

Now that you have set up the tool, you can start to run it.

To run the Index Defrag tool:

1. In the ***Index Defrag Card*** window, click Process. The tool now runs the `DBCC SHOWCONTIG` SQL statement. When it is finished, it populates the Index Defrag Card with the results:



Datebase: indexdefrag, SQL Object No.: 2099048 - Index Defrag Card

Database Information
Database No. . . . . . . 10 indexdefrag

Object Information
SQL Object No. . . . . . 2099048 CRONUS International Ltd_$Item Vendor
Navision Table No.. . . . 99 Item Vendor

Index Infromation
SQL Index No. . . . . . 1 CRONUS International Ltd_$Item Vendor$0 (CLUSTERED INDEX)

Results
Scan Type. . . . . TABLE Level Scan
Pages Scanned 1
Extents Scanned 1
Extent Switches 0
Average Pages Per Extent 1,00
No. of Records 20
Scan Density 100,00%
Scan Density Ratio [Best:Actual] . [1:1]
Logical Scan Fragmentation . 0,00%
Extent Scan Fragmentation . 0,00%
Average Bytes Free Per Page . . 7.130,00
Average Page Density (Full) . 11,91%

Status
Scan Density 100% 🟢
Average Page Density 12% 🔴
Logical Scan Fragmentation 0% 🟢
Extent Scan Fragmentation 0% 🟢

Legend
🟢 **Good** 🟡 **Warning** 🔴 **Problem**

Index Defrag Card

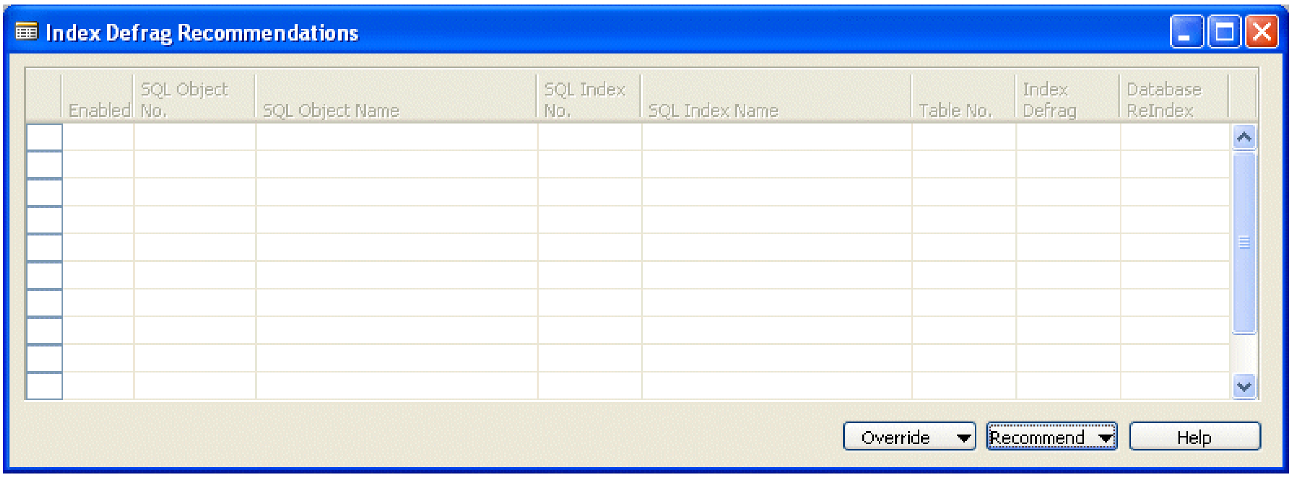
You can also print a report that summarizes the information contained in this window. To print this report click Defrag, DBCC Showcontig, Print Results.

The Index Defrag tool also contains functionality that allows you to get a better overview of the data that has been collected. This makes it much easier for you to identify the tables that contain fragmented indexes.

Identifying Fragmented Indexes

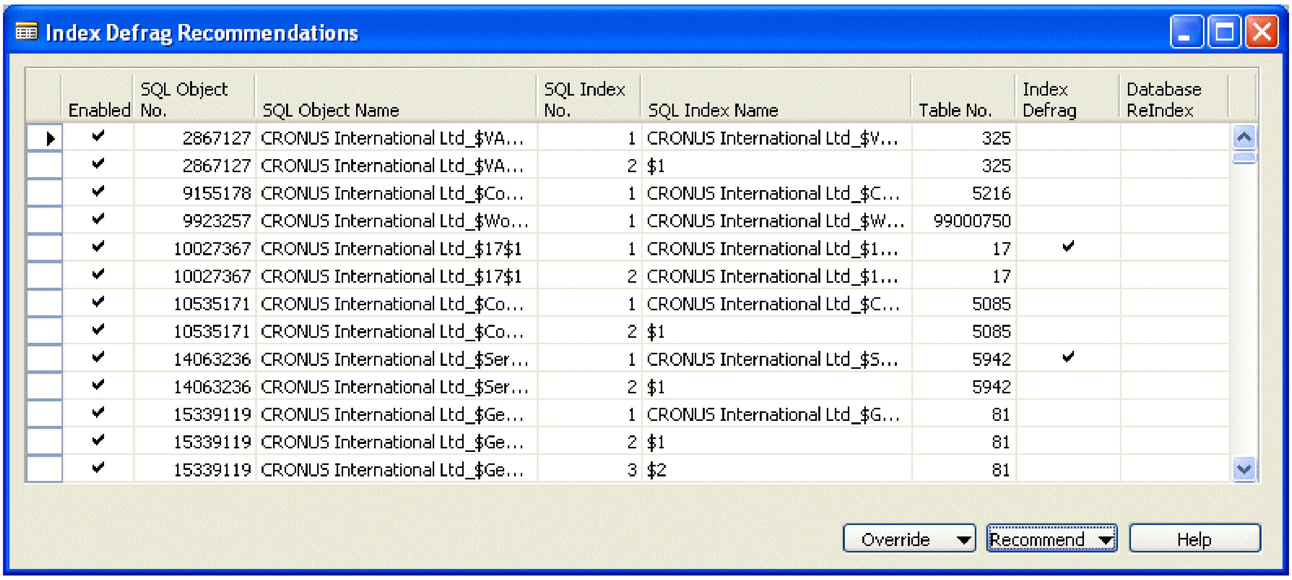
To identify the fragmented indexes:

1. Open the ***Index Defrag Card*** window and click Recommend and the ***Index Defrag Recommendations*** window opens:



Index Defrag Recommendations

2. Click Recommend, Generate and the ***Index Defrag Recommendations*** window is populated with data telling you which indexes the tool recommends should be defragmented and which should be re-indexed:



Enabled	SQL Object No.	SQL Object Name	SQL Index No.	SQL Index Name	Table No.	Index Defrag	Database ReIndex
✓	2867127	CRONUS International Ltd_$VA...	1	CRONUS International Ltd_$V...	325		
✓	2867127	CRONUS International Ltd_$VA...	2	$1	325		
✓	9155178	CRONUS International Ltd_$Co...	1	CRONUS International Ltd_$C...	5216		
✓	9923257	CRONUS International Ltd_$Wo...	1	CRONUS International Ltd_$W...	99000750		
✓	10027367	CRONUS International Ltd_$17$1	1	CRONUS International Ltd_$1...	17	✓	
✓	10027367	CRONUS International Ltd_$17$1	2	CRONUS International Ltd_$1...	17		
✓	10535171	CRONUS International Ltd_$Co...	1	CRONUS International Ltd_$C...	5085		
✓	10535171	CRONUS International Ltd_$Co...	2	$1	5085		
✓	14063236	CRONUS International Ltd_$Ser...	1	CRONUS International Ltd_$S...	5942	✓	
✓	14063236	CRONUS International Ltd_$Ser...	2	$1	5942		
✓	15339119	CRONUS International Ltd_$Ge...	1	CRONUS International Ltd_$G...	81		
✓	15339119	CRONUS International Ltd_$Ge...	2	$1	81		
✓	15339119	CRONUS International Ltd_$Ge...	3	$2	81		

Index Defrag Recommendations

If the tool has found any tables that contain fragmented indexes or indexes that need to be rebuilt, there is a check mark in the appropriate field for that table.

You can add or remove check marks as you see fit.

If you would like to defragment the indexes in every table or rebuild every index in the database, click Override, DBReIndex – All or IndexDefrag – All depending on what you want to do.

Important

Rebuilding indexes is a complicated and time consuming process that locks many resources and should only be done when there are no users accessing the database. Index defragmentation can be performed at any time.

To defragment the indexes:

1. When you have decided which indexes need defragmenting, click Recommend, Create Scripts to create the SQL scripts that will defragment and rebuild the indexes.
2. Click Recommend, Edit/Execute Scripts and the scripts are opened in the Query Analyzer. You can now edit the scripts if you want to.
3. In Query Analyzer, run the script and the indexes are defragmented.

You can verify that the indexes have been defragmented by repeating the entire process described in this section.

You can also print a report that lists all the tables and tells you which ones should be defragmented and which ones should be rebuilt. To generate this report click Recommend, Print Report.

Maintenance Plan

Many of the topics that we have discussed in this document can be addressed by creating a SQL Server maintenance plan. A maintenance plan can be created that addresses such issues as automating backups and rebuilding indexes on a scheduled basis. You can use a wizard to create a maintenance plan and this is fully documented in SQL Server Books Online.

Excerpt from SQL Server Books Online:

The Database Maintenance Plan Wizard can be used to help you set up the core maintenance tasks necessary to ensure that your database performs well, is regularly backed up in case of system failure, and is checked for inconsistencies. The Database Maintenance Plan Wizard creates a SQL Server 2000 job that performs these maintenance tasks automatically at scheduled intervals.

The maintenance tasks that can be scheduled to run automatically are:

- *Reorganizing the data on the data and index pages by rebuilding indexes with a new fill factor. This ensures that database pages contain an equally distributed amount of data and free space, which allows future growth to be faster. For more information, see Fill Factor.*
- *Compressing data files by removing empty database pages.*
- *Updating index statistics to ensure the query optimizer has up-to-date information about the distribution of data values in the tables. This allows the query optimizer to make better judgments about the best way to access data because it has more information about the data stored in the database. Although index statistics are automatically updated by SQL Server periodically, this option can force the statistics to be updated immediately.*
- *Performing internal consistency checks of the data and data pages within the database to ensure that a system or software problem has not damaged data.*
- *Backing up the database and transaction log files. Database and log backups can be retained for a specified period. This allows you to create a history of backups to be used in the event that you need to restore the database to a time earlier than the last database backup.*

- *Setting up log shipping. Log shipping allows the transaction logs from one database (the source) to be constantly fed to another database (the destination). Keeping the destination database in synchronization with the source database allows you to have a standby server, and also provides a way to offload query processing from the main computer (source server) to read-only destination servers.*

*The results generated by the maintenance tasks can be written as a report to a text file, HTML file, or the **sysdbmaintplan_history** tables in the msdb database. The report can also be e-mailed to an operator.*

Optimizing Navision Tables

Navision includes a feature that you can use to optimize your database tables.

To optimize a table:

- Click File, Database, Information, and the ***Database Information*** window opens.
- Click Tables and the ***Database Information (Tables)*** window opens.
- Select the table or tables that you want to optimize and click Optimize. You can select a single table, several tables or all the tables.

What Does Table Optimization Do?

Navision executes the following statement for each index that is maintained on SQL Server:

```
CREATE …. INDEX …. WITH DROP_EXISTING
```

If any SIFT tables are maintained, the zero entries are removed from the SIFT tables.

What Are the Benefits?

The benefits of table optimization include:

- Better performance as a result of defragmenting the indexes.
- Fewer records in the SIFT tables. There is a simple reason for this – Navision does not delete SIFT records. If you, for example, maintain some SIFT indexes on temporary tables such as the **Warehouse Activity Line** table, the SIFT records are not deleted from the SIFT tables when you delete (process) the records from the source table.

How Long Does It Take to Optimize?

It takes the same amount of time as it takes to create the indexes from scratch.

Can the Process Be Scheduled on SQL Server?

You can automate the first part using `DBCC REINDEX` or `DBCC INDEXDEFRAG`, however rebuilding SIFT tables cannot be automated easily and is not without its risks.

You can design custom SQL queries that delete records that contain zero values from the SIFT tables – all the ‘*s*’ columns contain zero values. However, those queries must be modified whenever you change the design of the SIFT indexes in Navision.

Server Performance Counters to Monitor

The following table contains a list server counters that you should monitor and lists the recommended action that you should take if the counters are not at their optimum value:

Performance Object	Counter Name	Instances	Best Value	Recommendation (when the Best Value is not met)
Memory	Available MBytes	SQL Server TS Servers	>5MB	Add more memory Reserve less memory for SQL Server
	Pages/sec	SQL Server TS Servers	<25	Add more memory Reserve less memory for SQL Server
Physical Disk	Avg. Disk Read Queue length	SQL Server Disks	<2	Change disk system
	Avg. Disk Write Queue length	SQL Server Disks	<2	Change disk system
Processor	% Processor Time	SQL Server TS Servers	0-80	Add more CPUs
System	Processor queue Length	SQL Server TS Servers	<2	Add more CPUs
	Context Switches/sec	SQL Server TS Servers (multi-processors)	<8000	Set Affinity Mask
Network Interface	Output Queue Length	SQL Server TS Servers	<2	Increase network capacity
SQL Server Access Methods	Full Scans/sec	SQL Server		Review Navision C/AL code
	Page Splits/sec	SQL Server	0	Defrag SQL Server indexes Review Navision C/AL Keys
SQL Server Buffer Manager	Buffer cache hit ratio	SQL Server	>90	Add more memory
SQL Server Databases	Log Growths	SQL Server	O (during peak times)	Increase and set the size of the transaction log
SQL Server General Statistics	User Connections	SQL Server		
SQL Server Locks	Lock Requests/sec	SQL Server		Review Navision C/AL code
	Lock Waits/sec	SQL Server		Review Navision C/AL code

Troubleshooting

The Performance Troubleshooting Guide that is available on the Navision tools CD includes tools and documentation that help Navision developers identify code bottlenecks, parts of the code that execute slowly as well as identify blocking and deadlocking situations.

SQL Diagnostic Utility

The sqldiag utility gathers and stores diagnostic information and the contents of the query history trace (if it is running). The output file includes error logs, output from `sp_configure` and additional version information. If the query history trace is running when the utility is invoked, the trace file will contain the last 100 SQL events and exceptions.

sqldiag is intended to expedite and simplify information gathering by Microsoft Product Support Services. If you have any problem with your SQL Server you will be asked to provide this information.

Running with the default options:

Run `sqldiag.exe` in the `MSSQL\Binn` folder. The default path is:

```
C:\Program Files\Microsoft SQL Server\MSSQL\Binn\sqldiag.exe
```

Collecting the results:

Collect `SQLdiag.txt` in the `MSSQL\LOG` folder. The default path is:

```
C:\Program Files\Microsoft SQL Server\MSSQL\LOG\SQLdiag.txt
```

For more information:

http://msdn.microsoft.com/library/en-us/coprompt/cp_sqldiag_96k9.asp

Typical Performance Problems

The following table lists some of the problems that have been dealt with by Microsoft Business Solutions support teams:

Area	Problem	Examples and Comments
Hardware	Badly configured disks	using RAID5, incorrect stripe size, incorrect cache policy, database files and transaction log files on same disk, too many disks on the same channel, database files not spread across the disks,
	RAM	4GB RAM on machine, 2GB only used by SQL Server – must use Windows Advanced Server and enable 3GB switch.
	Parallelism	Reducing the Max Degree of Parallelism to 1 increased performance of heavy batch jobs.
Platform	Maintenance	No maintenance plan
		No archiving plans – tables can grow very large over the years.

Area	Problem	Examples and Comments
	Old version of C/SIDE	It is a highly recommended that you keep your C/SIDE up to date – there are potentially big performance improvements and upgrading C/SIDE is nowhere near as costly as upgrading the application.
Application	SIFT	Large ‘empty’ SIFT tables – Optimize the tables to remove the entries which have zero sums or are based on nonexistent source tables.
		Too may SIFT indexes on hot tables.
		Too many SIFT levels maintained on big composite indexes.
		SIFT indexes on ‘temporary’ tables such as 36, 37, 38, 39 and so on. The record sets in these tales are typically small and only live in the database for several days.
		SIFT indexes on the ***Warehouse Activity Line*** table can cause deadlocks.
	Indexes	Too many indexes on ‘hot’ tables such as the ledger entry tables.
	Users	User input screen left hanging after processing has been started thereby locking the tables involved.
		Renaming an object during working hours and causing massive blocks because the application needs to update (lock) all the relevant tables.
	Business Logic and Timing	Enabling “Automatic Cost Posting” and/or “Expected Cost Posting to G/L” in Inventory Setup. These functions extend the processing time of any stock transaction by posting to the General Ledger and locking the ***G/L Entry*** table. It makes more sense to run this as a batch job outside normal working hours.
		Enabling “Update on Posting” in Analysis Views. This function extends processing times of any G/L transaction by updating the view. It makes more sense to run this as a batch job outside normal working hours.
		Posting to large journals during work hours.
		Inventory cost adjustment during work hours.
		Batch posting stock to G/L during work hours.
		Running MRP during work hours.
	Calcfields	Enabling automatic credit limit warnings (but not using the feature) – the code updates a lot of calcfields.
		Enabling automatic stock out warnings (but not using the feature) – the code updates a lot of calcfields.
	Processing Lengthy Transactions	One lengthy transaction can, for example, fetch all the sales lines, create and release picks, update associated purchase orders and so on. While this activity is run, almost all the users are blocked.
	Dimensions	Too many dimensions. The more dimensions you have the more indexes and SIFT tables need to be updated and maintained.
	General	Enabling the Find as you Type feature.
		Running ad hoc queries on large tables.

Useful Links

SQL Server 2000 Operations Guide

http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlops0.mspx

The contents of this guide are drawn from the knowledge and best practice guidelines drawn up by Microsoft Consulting Services (MCS), Microsoft's Internal Technical Support Group, and the SQL Server Development Team.

How To: Use the SQLIOStress Utility to Stress a Disk Subsystem Such As SQL Server

http://support.microsoft.com/default.aspx?scid=kb;en-us;231619

Assessing the New Microsoft SQL Server 2000 Resource Kit

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsqlpro01/html/sql01f1.asp

Microsoft SQL Server 2000 Resource Kit CD

http://www.microsoft.com/resources/documentation/sql/2000/all/reskit/en-us/sql2krk.mspx

Microsoft SQL Server 2000 Best Practices Analyzer 1.0 Beta

http://www.microsoft.com/downloads/details.aspx?FamilyID=B352EB1F-D3CA-44EE-893E-9E07339C1F22&displaylang=en

Microsoft SQL Server Best Practices Analyzer is a database management tool that lets you verify the implementation of best practices on your servers.

How to View SQL Server 2000 Performance Data

http://support.microsoft.com/default.aspx?scid=kb;en-us;283886

Identifying Common Administrative Issues for Microsoft SQL Server 2000

http://support.microsoft.com/default.aspx?scid=kb;en-us;322322

How to Monitor SQL Server 2000 Blocking

http://support.microsoft.com/default.aspx?scid=kb;en-us;271509

About Microsoft Business Solutions

Microsoft Business Solutions, a division of Microsoft, offers a wide range of integrated, end-to-end business applications and services designed to help small, midmarket and corporate businesses become more connected with customers, employees, partners and suppliers. Microsoft Business Solutions' applications optimize strategic business processes across financial management, analytics, human resources management, project management, customer relationship management, field service management, supply chain management, e-commerce, manufacturing and retail management. The applications are designed to provide insight to help customers achieve business success. More information about Microsoft Business Solutions can be found at http://www.microsoft.com/BusinessSolutions/