Optimizing Dynamics NAV on SQL Server

Part I - Application

Hynek Muhlbacher
SQL Perform Ltd

What is covered in this webcast?

- Monday January 29th 2007, 4:30-6:30 CET
- SQL Optimization for Microsoft Dynamics NAV
- Part I – Application
- Focus on design and troubleshooting at an application level to optimize SQL performance
- Include a walk through of
  - Performance Troubleshooting Guide
  - SQL Server Resource Kit
  - Application Benchmark Toolkit

What is covered in next webcast?

- Friday February 2nd 2007, 4:30-6:30 CET
- SQL Optimization for Microsoft Dynamics NAV
- Part II – Infrastructure
- Focus at infrastructure level to optimize SQL performance
- Include a walk through of
  - Hardware & Software Planning
  - Maintenance & Troubleshooting
  - Common performance problems

Versions/Names

- Attain, Navison, Dynamics NAV?
  - Referred here as NAV
- Microsoft SQL Server 2000 or Microsoft SQL Server 2005?
  - No difference from NAV perspective
  - Referred here as SQL Server
- NAV 2.60, 3.00, 3.10, 3.60, 3.70, 4.00 or 5.00?
  - No difference in principle
  - Before 3.10 no Client Monitor
  - 4.00 SP1 brings new SQL Features
  - 5.00 application changes for SQL Server

Agenda

- Introduction
- NAV ‘Native’ and SQL Server Differences
- NAV on SQL Server Basics
- Costs of Data Manipulation, Cursors, Blocks and Deadlocks
- Eliminating the costs
- Tools & Recourses
- Summary
- Q+A

SQL Perform Ltd

- Formerly HM Business Solutions
- Tools, education and services for partners and end-users
- 100% focus on performance
- International coverage
- Three years running
- 15 years NAV experience
- 6 years NAV on SQL experience
- SQL Perform director original author of the Navision SQL Resource Kit and parts of Troubleshooting Guide
- Projects helping medium to big companies including household names like Microsoft, GNN, Imerco, ACA International, Smyths Toys, Gamestop, Carpetright, FTC Kaplan, Danish State, Saxo Bank, Hilton Hotels, RYA, BMW, Royal Canin, etc.

Why do we need to know more?

We need to know more because

- Servers are different
- Knowledge is different
- Approaches are different

Dynamics NAV ‘Native’ and SQL Server differences

Servers Are Very Different

- Microsoft Dynamics NAV “Native” Server = ISAM
  - Indexes based – Indexed Sequential Access Method
  - Server is ‘dumb’, Client processes
  - Native SIFT
  - Native Version Principle
  - Table locking
- Microsoft SQL Server = Result Sets
  - Query Optimizer gets data, sort/hash/joints ‘inexpensive’
  - Can do server side processing, Microsoft Dynamics NAV does not use it
  - Need (very expensive) cursors to act as ISAM
  - SIFT and Version Principle are ‘Simulated’
  - Row locking

NAV on SQL Server Basics

Two-tier Infrastructure

- NAV Client (including NAS) process executes C/AL code
- Some C/AL Code needs data or object
- Call to NDBCS driver
- NDBCS driver request to SQL Server
- SQL Server reply
- NDBCS driver supply data

C/AL Reading Data

C/AL to T-SQL “Translation”

```
TableX.GET('ABC‘);

SELECT * FROM TableX
WHERE Code = 'ABC‘

either  WITH (READUNCOMMITED)
or      WITH (UPDLOCK)
(depends if we had implicit/explicit LOCKTABLE)
```

C/AL to T-SQL “Translation”

```
WITH TableX DO BEGIN
SETCURRENTKEY(Code);
SETRANGE(Code,'ABC‘);
FIND('-‘);

SELECT * FROM TableX
WHERE Code = 'ABC‘
ORDER BY "Code", ……,"Entry No."

either  WITH (READUNCOMMITED)
or      WITH (UPDLOCK)
(depends if we had implicit/explicit LOCKTABLE)
```

T-SQL Execution (simplified)

- Query received
  SELECT * FROM TableX
  WHERE Code = 'ABC‘
  ORDER BY Code, …
- Query Optimizer decides how to get the data (reading Clustered or Non-Clustered indexes), and performs sorting, and/or other operations such as SUMs
- INDEPENDENTLY!! Based on costs

Query Optimizer Execution Plan

- SQL Commands
  - sp_helpindex
  - dbcc show_statistics
- Show Estimated Execution Plan
- Let's have look - demo of the above

Query Optimizer Execution Plan Demo

Query Optimizer Execution Plan

- What have we seen?
- Based mainly on costs of data retrieval
- Based on statistics histogram
- Sorting a secondary cost

C/AL Data Modifications

C/AL Commands

```
WITH TableX DO BEGIN
INSERT;

SETRANGE(Code,'ABC');
FieldA := 'B';
MODIFY;
or
MODIFYALL(Code,'B');

SETRANGE(Code,'ABC');
DELETE;
or
DELETEALL;
```

C/AL to T-SQL “Translation”

- INSERT INTO TableX VALUES(….)
- UPDATE TableX SET FieldA = 'B'
  WHERE Code = 'ABC'
- DELETE FROM TableX
  WHERE Code = 'ABC'

Data Modification Costs

- There is an element of reading costs (WHERE statement)
- Record(s) need to be inserted/deleted/modified
- Indexes need to be updated
- SIFT Indexes need to be updated through SQL Triggers
- And no two processes may update table(s) in the same range

Data Modification Costs Demo

Data Modification Cost Demo

- What have we seen?
- Many indexes and SIFT tables updates
- Lots of Reads and CPU time
- Long insert/modify/delete durations

Summary

- Indexes help during Data Reading
- Indexes cost during Data Modifications

Summary

- Dynamics NAV tables have many indexes
- Some indexes have low selectivity
  - Boolean and Option fields at the index start
- Some indexes have ‘unused’ fields
  - 'Variant Code' if variants not used, etc
- Some indexes are similar to others
  - “slightly” different to support different sorting
- Some indexes are for a specific feature
  - Intercompany, Intrastat, Manufacturing, etc
- Some indexes are for sorting only
  - Warehouse Activity Line, and similar

Cursors

Cursors

- ‘Native’ server reads in B-Tree index and returns data record by record
- SQL Server is a set based engine – returns a set based on the WHERE clause
- NDBCS driver uses cursor to ‘simulate’ the ‘native’ server

T-SQL Cursor Syntax (simplified)

```
DECLARE cursor_name CURSOR FOR select_statement

OPEN cursor_name

FETCH NEXT FROM cursor_name

WHILE @@FETCH_STATUS = 0 BEGIN
  xxx
  FETCH NEXT FROM cursor_name
END
```

Cursor types

[LOCAL | GLOBAL]

[FORWARD_ONLY | SCROLL]

[STATIC | KEYSET | DYNAMIC |
FAST_FORWARD]

[READ_ONLY | SCROLL_LOCKS |
OPTIMISTIC]

Cursor Costs

- Data sets need to be read and kept somewhere
- Extra costs on CPU, memory, tempdb
- For FIND('Which') command NAV uses dynamic cursor – the most expensive cursor of all

Cursor limitations

- If a set is fetched into a cursor and the set definition is modified, and a NEXT record is requested;

- the cursor can no longer be re-used
- NAV needs to find NEXT record through series of individual SELECT statements
- a new cursor is created

- Results to suboptimum NEXT

Summary

- Cursors are not “evil”, but
- Cursors should be avoided if possible
- Should keep cursor set definition – avoid suboptimum NEXT

Locks, Blocks, Deadlocks

Locks

- No locking
  - “dirty” data reading, can read phantoms
- With locking
  - higher isolation level
  - SQL Server is asked to keep the data for you
  - SQL Server locks a range, plus neighbours
- LOCKTABLE on SQL differs with ‘Native’
  - SQL locks when reading, meaning “no read – no lock”
  - ‘Native’ locks whole table if implicit or explicit LOCKTABLE was used and command buffer is to be executed

NAV Optimistic Concurrency

- EXPLICIT Locking:
  - Used after LOCKTABLE
  - NAV uses UPDLOCK
  - If a modification of already read data then no extra read as we rely on SQL Server to protect the set
- IMPLICIT Locking:
  - NAV uses READUNCOMMITTED
    - reads data plus record timestamp
  - If a modification of already read data then a new read with UPDLOCK, and filter on the original timestamp
  - this means reading twice, a bit similar to LOCKTABLE(TRUE,TRUE), but not the same!

Locks, Blocks and Deadlocks

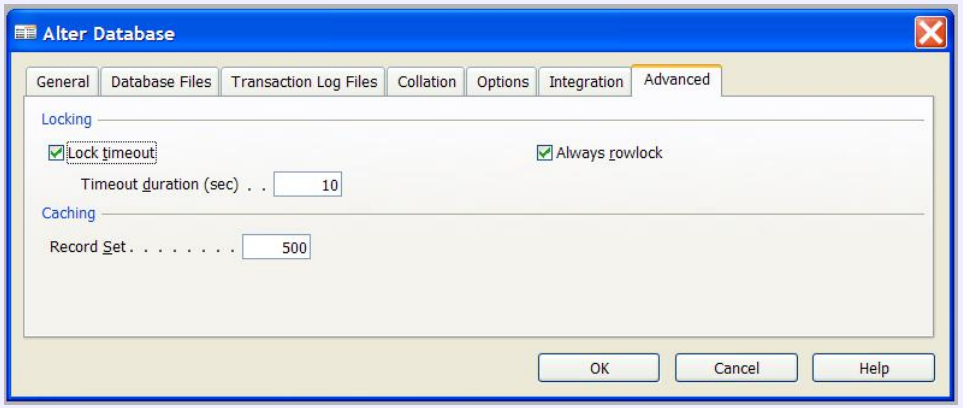
- No lock – “dirty” data reading
- Lock – data reading with higher isolation level
- Block
  - Session A locks resource X
  - Session B tries to lock resource X
- Deadlock
  1. Session A locks table X
  2. Session B locks table Y
  3. Session A tries to lock table Y (is blocked and waiting for session B to finish)
  4. Session B tries to lock table X (which is blocked by session B) – Deadlock.

Locks, Blocks, Deadlocks Costs

- Memory to keep the locks
- Users waiting for other processes
- Deadlocked users “not happy”
- Deadlocked application may have partial commit and roll-back only rolls-back partially, resulting to dirty data

Locking Options

- New in NAV 4.00 SP1
- Database-wide options, File, Database, Alter:
  - Lock Timeout
  - Always Rowlock



- C/AL command
  - LOCKTIMEOUT(FALSE);

Summary

- Locks are not “evil”, but
- Should be kept at minimum
- Should last as short as possible
- Should not lead to deadlocks

So these are the basics of NAV on SQL Server

How can we make it better?

How do we eliminate the impact?

How can we reduce the costs?

Eliminating costs

- Data access/manipulation costs
- Cursor costs
- Blocking, deadlocking
- Sub-optimum code

Eliminating data access and manipulation costs

OLTP/OLAP type of applications

- OLTP (On-line Transaction Processing)
  Day to day activities such as create and post sales order

- OLAP (On-line Analytical Processing)
  Reports, Analysis by Dimensions

Focus on OLTP, not OLAP

- Quick Insert/Modify/Delete !!
- Quickly insert sales order, quickly post sales order, ...

- Reading secondary !!
- Print a report, look at statistics, …

Focus on indexes

- Index maintenance costs
- SIFT Index maintenance costs
- Index efficiency

Optimizing Indexes

- Properties to use
  - Keygroups
  - MaintainSQLIndex
  - MaintainSIFTIndex
  - SIFTLevels
- New in 4.00 SP1
  - SQLIndex
  - Clustered
  - Key virtual table

KeyGroups

- Index(s) for a specific feature
- Can be turned on/off by user
- Use for features that can be isolated
- To be documented and used extensively in version 5.0

MaintainSQLIndex

- Turn on/off index maintenance
- Application still works without it
  (it uses ORDER BY, not INDEX hint)
- Use for “redundant” indexes

MaintainSIFTIndex

- Turn on/off SIFT table maintenance
- Application still works without it
  SELECT SUM FROM [Sales Line]
  rather than
  SELECT SUM FROM [37$0]

- Turn off for sums on small tables and/or if the filtered sets are small

SIFTLevels

- Turn on/off individual ‘buckets’
- Further eliminates the impact

SQLIndex

- New property in version 4.0 SP1
- Ability to create alternative SQL indexes to the ones defined in the Dynamics NAV Application
- Changing the order of the columns in the index (composite index/low selectivity issue)
- Taking out some columns in the index
- You must include all columns from the primary key in order to get an Unique Index created
- Non-Unique Index are allowed

Clustered

- Enables to change the clustered index to another key than the Primary
- Only use when you know what you doing
- Good example:
  - table had three fields in primary key, we change clustered to index with the same three fields but reorganized it to support better selectivity
- Bad example:
  - table had one field in primary key, we changed it to a three fields index because we read the table that way very often. The issue is that SQL Server will add all the clustered keys into all non-clustered keys.

Virtual table Key

- Added in 4.00 SP1
- Exposes all Key properties apart from KeyGroups and SIFTLevels
- Possible to change Enabled, MaintainSQLIndex, MaintainSIFTIndex and Clustered properties
- Change results in table redesign (changing Modified, Date and Time fields in the Object table)
- SETRANGE and FIND does not work properly on that table (bug reported)

Optimizing Dynamics NAV Indexes and SIFT Tables

- Focus on indexes on ‘hot’ tables
- Minimize number of SIFT indexes
  - If the set is small, doing SUM on normal table is quick enough
- Minimize number of indexes
  - Eliminate maintenance of indexes with sorting purpose only
  - Reorganize indexes, filtered fields first, eliminate ‘gaps’
- Redesign indexes for better selectivity
  - Boolean, Option and similar nature fields towards the end of the index, or not at all

Optimizing Dynamics NAV Indexes and SIFT Tables

- Avoid too granular indexes unless really needed
  - Date fields towards the end of the index
  - Avoid double dates in indexes, especially with SIFT
- Do not maintain SIFT on small/temporary tables
  - Sales Line, Purchase Line, Journal Lines, Warehouse Activity Line – have a look at that one!
- Minimize number of SIFT buckets
  - No need to maintain the same thing twice
- Archiving

Eliminating cursor costs

Focus on Cursor Costs

- Dynamic cursors costs
- Costs of suboptimum NEXT
- Suboptimum coding costs

Dynamic Cursor Costs

- FIND('Which') command is ambiguous
- NDBCS driver creates dynamic cursor
- NDBCS driver tries itself to eliminate the costs, for example
  - reading ahead (FETCH 10,…)
  - changing to fast_forward cursor
  - etc

Reducing the costs

- New C/AL Commands in 4.0 SP1
- FINDFIRST
- FINDLAST
- FINDSET

- …and the ‘old’ ISEMPTY command

ISEMPTY;

- ‘Forgotten’ command
- SELECT TOP 1 NULL
- (Returns TRUE or FALSE only)
- Can be used in almost all checking functions

- Makes obsolete
  IF FIND(‘-’) THEN ERROR(TextXYZ)

  ! be careful as you can easily create a bug !
  the function does not return record values…

FINDFIRST;

- No cursor
  - One server trip compared to several round-trips with FIND('-')
  - 'Native' SQL set retrieval
- SELECT TOP 1 *
- Use for a single record find

FINDLAST;

- No cursor
  - One server trip compared to several round-trips with FIND('+')
  - 'Native' SQL set retrieval
- SELECT TOP 1 *, DESC
- Use for a single record find

FINDSET;

- Optimized loops
- Code for

```
IF FIND('-') THEN
  REPEAT
  UNTIL NEXT = 0
```

- Now simply

```
IF FINDSET THEN
  REPEAT
  UNTIL NEXT = 0;
```

FINDSET syntax and settings

- [Ok :=] FINDSET([ForUpdate] [, UpdateKey])
- Database property ‘Record Set’. Sets the amount of records retrieved in default record set

FINDSET variations

- FINDSET
  - No cursor
  - SELECT TOP 500 * FROM … WITH(READUNCOMMITED)
- LOCKTABLE and FINDSET
  - No cursor
  - SELECT TOP 500 * FROM … WITH(UPDLOCK)
- FINDSET(TRUE)
  - Dynamic cursor for
  - SELECT * FROM … WITH(UPDLOCK)
  - Followed by FETCH 10, FETCH 25, or similar
  - Built in read ahead, FETCH 75, FETCH 150, or similar follows
- FINDSET(TRUE,TRUE)
  - Dynamic cursor for
  - SELECT * FROM … WITH(UPDLOCK)
  - Followed by FETCH 1 with no read ahead (“Fetch Fetch” mode)

Costs of suboptimum NEXT

- Probably the single biggest Dynamics NAV cursor issue
- Perform NEXT on
  - Changed filter
  - Changed sorting
  - Changed key value
  - Changed isolation level
  - In the middle of nowhere

Example: Modifying filtered values

```
SETRANGE(FieldA,Value);     //set based on filter
FIND('-');
REPEAT
  …
  FieldA := NewValue;       //first warning !
  …
  MODIFY;                   //alarm !
  …
UNTIL NEXT = 0              !!! NEXT on invalid set
```

Example: “Jumping Through Set”

```
SETRANGE(FieldA,Value);
FIND(‘-’);                          //set based on filter 1
…
REPEAT
  SETRANGE(FieldB,FieldB);  //new extra filter, new set
  …
  FIND(‘+’);                        //!!new cursor for this…
  …
  SETRANGE(FieldB);
..
UNTIL NEXT = 0                      !!! NEXT where?
                                     //!!new cursor for this…
```

Example: Non-Cursor to Cursor

```
SETRANGE(FieldA);
FINDFIRST;                    // Non-cursor fetch
REPEAT
  …
UNTIL NEXT = 0                !!! NEXT creates
                              a new cursor
```

Example: Changing Isolation Level

Rec.FIND('-');

New cursor with READUNCOMMITED

Rec.Field1 := Value;

Rec.MODIFY;

New read of the record with UPDLOCK isolation level and filter on timestamp

Rec.NEXT;

Series of T-SQL statements estimating if next record exists

Drops existing and creates new cursor with UPDLOCK isolation

Eliminating Suboptimum NEXT

- Browse sets nicely
- Use a separate looping variable
- Restore original key values, sorting and filters before the NEXT statement
- Read records to temp tables, modify within, and write back afterwards
- FINDSET(TRUE,TRUE)

Eliminate other cursor costs

- Never use WHILE FIND('Which')
- Do not use Find As You Type
- Use MODIFYALL and DELETEALL
- Use SourceTablePlacement = Last/First

Priorities

- Eliminate suboptimum NEXT
- Use FINDSET
  on read only sets
- Use LOCKTABLE and FINDSET
  on sets you want to modify
- Use ISEMPTY, FINDFIRST, FINDLAST
  to see if there are any or get single record
- Use FINDSET(TRUE)
  on sets you want to modify
- Use FINDSET(TRUE,TRUE)
  on sets you modifying key value

Preventing Blocks and Deadlocks

Preventing Locks and Deadlocks

- Always process tables in the same order
  - No guarantee but it helps
- Block an agreed “master resource” first
  - Serializing transactions – bad concurrency
- Dynamics NAV uses combination of both above
- but the trick is to get application speed fast enough so that it cannot occur…

Preventing Locks and Deadlocks

- Test conditions of validity before you process further.
  - TEST NEAR – testing Sales Header record we are processing (Blocked, Dates, etc)
  - TEST FAR – testing Sales Lines, G/L Setup, Customer blocked, Items blocked, etc.
  - DO IT – process
  - CLEAN UP - finish

Preventing Locks and Deadlocks

- Minimize the duration of locks
  - import 5 records (use LOCKTABLE commands in the right order, see deadlock section later in document)
  - write a log
  - COMMIT changes
  - SLEEP for 1 minute (just an example)
  - SELECTLATESTVERSION
  - go to the beginning

Preventing Locks and Deadlocks

- Never allow user input during transaction
- Application Setup
- Overnight processing
- NAS processing/ printing

OK now we know what to do

Is there any more information?

Are there any more tools?

Tools & Resources

Tools & Resources

- SQL Server Resource Kit
- Application Benchmark Toolkit
- Performance Troubleshooting Guide
- SQL Tools
- SQL Perform Offerings

SQL Server Resource Kit

- Part of the Tools CD
- \Implementation\SQL Server Resource Kit\
- Contains articles and tools for Dynamics NAV and SQL consultants
- Make them better equipped for trouble free Dynamics NAV implementation on SQL Server platform (*)

(*)Note that the resource kit is built for Dynamics NAV 3.70 and SQL Server 2000 platform, even though a lot of it can be used for previous or next versions of the products, or is based on previous versions of the products.

SQL Server RK Content

- Articles
  - Planning
  - Designing
  - Implementing
  - Maintenance
  - Troubleshooting
  - Miscellaneous
- Tools
  - Useful SQL scripts
  - Index Defrag tool
  - Key Information tool
  - Dynamics NAV database sizing tool

SQL Server Resource Kit

Quick Look - Demo

Application Benchmark Kit

- Part of the Tools CD
- \Implementation
  \Application Benchmark Toolkit

- Way how to simulate load
- Measure response times

Application Benchmark Kit

Quick Look - Demo

Performance Troubleshooting Guide

- Part of the Tools CD
- \Implementation
  \Performance Troubleshooting Guide

- Contains documentation and tools for NAV consultants
- Use for application changes once you know which key area code is to be focused on

Performance Troubleshooting Guide

- THE performance trouble shooting tools
- Client Monitor and the Code Coverage tool
- Setting up the test environment
- Identifying the clients that cause performance problems
- Profiling a task with the Client Monitor
- Identifying the worst server calls and the keys and filters that cause them
- Identifying the tasks that cause deadlocks on Dynamics NAV Server
- Using the SQL Error Log to identify the clients involved in deadlocks on SQL Server
- How to identify locking problems
- How to set up locking order rules and check whether or not your application follows these rules
- Identifying index problems on SQL Server
- How to identify bad C/AL NEXT statements on SQL Server
- How to use Excel pivot tables to get an overview of the data in the Client Monitor
- Etc, Etc

Performance Troubleshooting Guide

Quick Look
– Demo of Client Monitor

SQL Tools

- SQL Profiler
- Index Tuning Wizard (2000)
- Database Tuning Adviser (2005)

SQL Profiler

- Measuring Read/Write latencies
- Events:
  - Stored Procedure::SP:StmtCompleted
  - TSQL::SQL:BatchCompleted
- Data Columns:
  - TextData, Duration, Reads, Writes, CPU
  - ApplicationName,LoginName,DatabaseName
- Filters:
  - Duration > 30-50ms, Exclude System IDs

SQL Profiler

Quick Look - Demo

SQL Perform Offerings

- Perform-Tools
- Perform-Workshops
- Perform-Services
- Perform-Partners
- Perform-Consultants

Perform-Tools

- Perform-Tools for Dynamics NAV
- Two modules focusing on maintenance and analysis
  - Perform-Maintenance
  - Perform-Analysis

Perform-Analysis

- Table costs analysis
- Index costs analysis
- SIFT costs analysis
- Lock Monitor
- Block Monitor
- Cache Distribution
- Disk Response Times Analysis
- Cache analysis
- etc

Perform-Maintenance

- Does the proper Dynamics NAV maintenance for you automatically
- SIFT Tables Optimizer – deletes ‘zero’ SIFT records
- Fill Factor Optimizer – reindexes with optimum fill factors based on index volatility and level of splitting
- Delete Autostats – deletes unused statistics
- Database Options Tool - makes sure that you use recommended database settings
- Etc

Perform-Workshops

- 2 days in depth (compared to the two hours today)
- Theory and hands-on
- Application
- Infrastructure
- Problem Analysis & Resolution

Perform-Services

- Provided by
  - Perform-Partners
  - Perform-Consultants
  - SQL Perform Ltd
- Way to grow your business
- Wide range of new potentials
  - Performance Tuning
  - DBA Services
  - Etc
- Contact us to become a partner today!
- Contact us to request a partner to help you today!

Summary

- We went through
  - NAV ‘Native’ and SQL Server Differences
  - NAV on SQL Server Basics
  - Costs of Data Manipulation, Cursors, Blocks and Deadlocks
  - Eliminating the costs
  - Extra tools & resources

What is covered in next webcast?

- Friday February 2nd 2007, 4:30-6:30 CET
- SQL Optimization for Microsoft Dynamics NAV
- Part II – Infrastructure
- Focus at infrastructure level to optimize SQL performance
- Include a walk through of
  - Hardware & Software Planning
  - Maintenance & Troubleshooting
  - Common performance problems

Q&A

Questions & Answers

Contact

Ms Meral Bahcetepe
SQL Perform Ltd
90 Long Acre
Covent Garden
London, WC2E 9RZ
United Kingdom
info@sqlperform.com
Mobile +44 (0) 77 6276 9221
Office +44 (0) 20 7716 5812