Comparative Study of SQL Server and Mimer SQL in Mission-Critical Systems

JENNY OLSSON

KTH Numerical Analysis
and Computer Science

Master's Degree Project
Stockholm, Sweden 2004

TRITA-NA-E04133

Numerisk analys och datalogi
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Comparative Study of SQL Server and Mimer SQL in Mission-Critical Systems

JENNY OLSSON

TRITA-NA-E04133

Master's Thesis in Computer Science (20 credits)
at the School of Computer Science and Engineering,
Royal Institute of Technology year 2004
Supervisor at Nada was Serafim Dahl
Examiner was Stefan Arnborg

Comparative study of SQL Server and Mimer SQL
in mission-critical systems

Abstract

This thesis discusses a number of differences between Microsoft SQL Server and Mimer SQL Engine. Tests were performed to investigate performance differences in the two database management systems as well as the possibilities to store intermediate results in for example temporary tables. Other tests investigated some differences in syntax if you migrate from Microsoft SQL Server to Mimer SQL Engine.

When running programs concurrently, Mimer SQL Engine proved to be more time efficient with its optimistic concurrency control than Microsoft SQL Server with its pessimistic concurrency control. Microsoft SQL Server is more time efficient than Mimer SQL Engine when running programs non-concurrently. Other tests established that it is a disadvantage that Mimer SQL Engine has no temporary result storage, which can be created in stored procedures and have session handling that are comparable to SQL Server's temporary tables and table data type. The conclusion of the syntax differences test were that it is almost impossible to write non-database specific code for the two databases if you need to perform more complicated tasks than simple select-statements.

Jämförande studie av SQL Server och Mimer SQL
i affärskritiska system

Sammanfattning

I denna rapport görs en jämförelse mellan Microsoft SQL Server och Mimer SQL Engine. Skillnader i prestanda i de två databashanteringssystemen jämförs och även möjligheterna att spara temporära resultat i till exempel temporära tabeller. Skillnader i syntax när man migrerar från Microsoft SQL Server till Mimer SQL Engine jämfördes också.

När program körs samtidigt, visade tester att Mimer SQL Engine var snabbare med sin optimistiska låsningskontroll än Microsoft SQL Server med sin pessimistiska låsningskontroll. Microsoft SQL Server är snabbare när man inte kör program samtidigt. Andra tester visade att det är en nackdel att Mimer SQL Engine inte har något möjlighet till temporär lagring som man kan skapa i stored procedures och använda sessionskontroll i som är motsvarande till Microsoft SQL Servers temporära tabeller eller table datatyp. Slutsatsen av testerna om syntaxskillnader är att det nästan är omöjligt att skriva kod som är databasoberoende om man vill utföra mer komplicerade saker än en enkel "select".

Contents

1 Introduction

The purpose of this thesis is to investigate some differences between two database management systems (DBMS): SQL Server 2000 and Mimer SQL Engine. The two DBMS were chosen because they use different transaction concurrency controls.

To investigate the difference between the DBMS, the work was divided into three parts. The first part deals with performance issues, both when programs, which access the database, are executed non-concurrently and concurrently. The second part investigates differences in storing intermediate results in the databases, and finally, the third part investigates differences in syntax. In the third part the possibility to write non-database specific code and if it is possible to use the same application with different DBMS in different installations is studied.

The work with this thesis was performed at Ntier Solutions AB, a company that offers solutions in system development, IT-infrastructure and administrative systems.

2 Background

Different DBMS solve certain tasks in different ways and different DBMS have different functionality. When deciding on which DBMS is best suitable for a specific project these differences must be taken into account as well as the ability to reuse and maintain code used in the project. In some cases it is also interesting to use different DBMS in two different instances of the application.

2.1 Database basics

2.1.1 Structured Query Language

The need for a standardized query language resulted in the Structured Query Language (SQL) when The American National Standards Institute (ANSI) standardized it in 1986 and the International Standards Organization (ISO) standardized it in 1987. SQL is a standard, open language without corporate ownership. Two years later, in 1989, a revised standard known as SQL-89 or SQL1 was published. This standard was strengthened into a new standard, SQL-92 or SQL2 in 1992. SQL-92 was about six times as long as its predecessor and due to this, the authors defined three different levels of SQL-92 compliance:

- Entry-level conformance (only the barest improvements to SQL89)
- Intermediate-level conformance (most of the major advancements)
- Full conformance (total compliance with the SQL92 features)

In 1999 the ANSI/ISO released the SQL-99 or SQL3. This standard also came in different versions: Core SQL-99 and Enhanced SQL-99. [19]

Today, most major DBMS support the SQL-92 standard. Unfortunately, in a standardization point of view, they have also added extra functionality such as new data types and their own methods. This makes it hard to migrate from these DBMS to another, without doing extensive code rewriting. If a DBMS supports a standard (for example SQL-92 Full) and has no extra functionality that is not standard or the extra functionality is not used, it is easy to migrate from this DBMS to another, provided that the other DBMS also supports the same standard.

2.1.2 Stored Procedures

A stored procedure is a group of SQL statements that are saved together in the database as a single unit of code. A stored procedure can have input- as well as output parameters. [14]

The first time a stored procedure is run it is compiled and an execution plan is produced. This execution plan is then cached and used the next time the stored procedure is run. [3]

An advantage of this way to handle a group of SQL statements is that it saves time because the code is already compiled. It also reduces redundant code and simplifies code updating, since all the code is stored in one place.

2.1.3 Indexes

Indexes in a database are similar to indexes in a book. In a book, the index allows you to find what you are searching for without reading the entire book and an index in a database allows you the find the rows you are searching for without scanning the whole table. Indexes can be created on one or more columns in a table. [16]

However, there are some disadvantages with using indexes. Tables with indexes require more storage space in the database. Commands that insert, update or delete data can also take longer time to execute, since the indexes need to be updated after the execution of the command. When deciding which columns to have indexes on a table it is important that the indexes are essential, so that the performance benefits outweigh the extra storage space and processing resources. [16]

An index can be clustered or non-clustered. A clustered index determines in what order the data should be stored, it is analogue to a telephone directory, where items are ordered by last name. A clustered index can contain many columns, like in the telephone directory where data can be organized by both last name and first name. But a table can only have one clustered index since the index decides the physical storage order of the data. A non-clustered index does not store the data in a specific order. [20]

2.1.4 Cursors

A cursor is a database object, which is used to manipulate data in a set on row-by-row basis, instead of the typical SQL commands that operate on all rows in a set at one time. When you use a cursor, you can fetch a row and make multiple operations

on that row. [4]

According to Kumar [7] cursors are "fundamentally evil" as they force the database engine to repeatedly fetch rows, negotiate blocking, manage locks and transmit results. Before using cursors you need to consider if they are really necessary or if you can solve the problem in a different way.

2.1.5 Views

A view is a virtual table and is a subset of data from one or more tables. The view is not stored in the database, but the select statement that specifies which rows and columns that should be in the view. This means that if the data in the base table is changed, these updates will reflect in the view. Also, any updates of the view will reflect in the base table. [16] [21]

2.2 Reuse and maintainability

It is always good to reuse code. One reason for this is that it is better to have code in one place instead of many different places. This makes it easier to maintain the code, since you only need to change in one place. Another reason to reuse code is that you have less code to write, so the application development will go faster. Also if code is reused, the final application will be smaller and therefore probably have fewer bugs. This also improves maintainability since it is easier to debug the application.

2.3 Mimer SQL Engine

Mimer SQL Engine 9.2 (from now on called Mimer SQL) that is used in the testing in this thesis is a relational DBMS developed by Mimer Information Technology. Besides this version they also offer two other DBMS versions. Mimer SQL Embedded, which is a tailor-made DBMS developed to suit specific environments and Mimer SQL Mobile, which is a DBMS for mobile devices. The company resides in Uppsala, Sweden and their ambition is to create "intelligent, maintenance-free DBMS for embedded and mobile solutions." [8] [10]

2.3.1 Standards

Mimer SQL has full support for SQL-92 transitional. The transitional SQL standard covers complete Entry, most of Intermediate and many important components of Full SQL-92. [11]

2.3.2 Optimistic Concurrency Control

Mimer SQL handles transaction concurrency by using Optimistic Concurrency Control (from now on called OCC). OCC is when users do not lock data in the database. When an update is performed, the system checks if there has been an update from another user since the data was read. If there has not been an update since data was read, the update is performed. If there has, an error is raised and the transaction is rolled back. It is the applications responsibility to determine an

appropriate action to handle the error. In some cases it is best to let the transaction try again until it is successful and in other cases it is best to re-prompt the data to the user and let them decide what actions that should be performed. [18] [6]

The optimistic locking is rather easy to implement since there never occurs any deadlocks (see 2.4.2 Pessimistic Concurrency Control), but there needs to be a check if a rollback must be performed. This control is a little more complex to implement.

2.4 Microsoft SQL Server

SQL Server 2000 (from now on called SQL Server), which is used in the testing in this thesis is a relational database and developed by Microsoft and is a commonly used DBMS.

2.4.1 Standards

SQL Server supports the Entry Level of SQL-92 and supports many additional features from the Intermediate and Full levels of the standard. The dialect of SQL supported by SQL Server is called Transact-SQL (T-SQL). T-SQL also supports many other features, which is not included in any standard. [16]

2.4.2 Pessimistic Concurrency Control

Pessimistic concurrency control (from now on called PCC) uses locks to limit or prevent other users from working with the data the owner is working with. There are several kinds of locks: a database lock locks the entire database, a table lock locks the entire table, a page lock locks a collection of rows and a row lock locks a single row. [1]

An advantage with using PCC is that changes in the database are made consistently and safely. A disadvantage with using PCC in a system is that the system cannot have many users or long lived transactions since this increases the chance of having to wait for a lock to be released. [1]

It is easy to implement the pessimistic locking, but it is more difficult to discover and handle the deadlocks.

SQL Server enforces locking automatically and uses a dynamic locking strategy to determine the most cost-effective locks. You can also customize the locking to suit your application. [16]

In DBMS supporting PCC, deadlocks can occur. A deadlock is when two user processes have locks on separate objects and each process tries to lock the object of the other process. When this happens in SQL Server, the system chooses one process to continue and aborts the other. The transaction that is aborted is rolled back and an error message is sent to the user whom is the owner of the transaction. [17]

2.4.3 Optimistic Concurrency Control

SQL Server also offers the possibility to use OCC but only if you use cursors. However, the DBMS is built for supporting PCC and it is this locking strategy that should be used, except for very special cases.

2.5 Mimer Provider Manager

The Mimer Provider Manager is a tool you can use when you have one application, which needs to communicate with different DBMS or if you want to change DBMS. You can use the Mimer Provider Manager without using any of Mimer's DBMS. To use the tool you change the code in the application, for example if you connect to a SQL Server database you probably use the Sql-connection classes, all occurrences of Sql is then changed to Mpm, for example SqlConnection is changed to MpmConnection and SqlCommand is changed to MpmCommand. When this is done, you need to change the connection string and create a data source in the Mimer Provider Manager Administrator. Then the application can be used to connect to any database, provided that the correct data source with the correct plugin is used. [9] [5]

2.6 Traci

Traci is a time report system created at NTier Solutions. The system uses a SQL Server 2000 database and has all its code in stored procedures, except for the user interface code. The stored procedures use temporary tables to store intermediate results.

2.7 Development environment

All tests performed during this master's project were run on a computer using Microsoft Windows XP. The test programs written and executed were written for the .NET platform. The Microsoft .NET Framework is a platform for building, deploying and running applications and web services. The .NET Framework consists of three main parts: the common language runtime, a hierarchical set of unified class libraries, and a componentized version of Active Server Pages called ASP.NET.

3 Comparison of SQL Server and Mimer SQL

To be able to investigate the different qualities in the two DBMS, such as syntax, performance, scalability and maintainability, the work was divided into three parts. The first part mainly focuses on performance issues. In this part test programs were produced, tested and run, then runtimes were compared. The second part deals with temporary tables and also the table data type, which exists in SQL Server. Finally, in the third part, some stored procedures are taken from Traci and rewritten to work in Mimer. In the third part, focus is on differences in syntax and what to consider when migrating from SQL Server to Mimer SQL. Also, in the last part, maintainability and the ability to write non-specific database code are tested.

The actual work was performed in the same order as presented here. Each of the following three parts are divided into three subparts; method, results and conclusion.

3.1 Performance

3.1.1 Method

In order to test performance issues in both DBMS, tests were performed to investigate runtimes when simple insert, select, update and delete programs were run. Also, tests were performed when combinations of these programs were run concurrently. This was done to test the PCC in SQL Server and the OCC in Mimer SQL. To take advantage of the OCC in SQL Server, you need to use cursors when performing the statements and to test this; the update program in SQL Server was also run using cursors.

To test the two DBMS, exactly the same table was created in both databases. This table consists of three columns (see table 1). The first column consists of an ID; this column is the primary key of the table and is an incremental counter. This column is never modified in any of the programs, only the database changes and creates values in this column. The second column is a date-column. Here are the date and time of the moment the row was created (see insert program) or changed (see update program). The last column consists of text strings of maximum length 900 characters. An example of a table used in the databases can be seen in table 2.

Table 1. The design of the table used in both databases.

Name	Type	Length
ID	Integer	4
Date	timestamp / datetime	16
Text	varchar	900

Table 2. An example of the table used in both databases.

ID	Date	Text
1	2004-05-16 12:28:03	AGEKEJSBNGRFJLXZOGJES
2	2004-06-23 14:23:00	GESWOLGPDPLWOQJHHIYKCNG ROAD

When you create a new table in SQL Server and Mimer SQL no indexes are created automatically, except for the clustered index, which is created on the primary key (ID column). Since most databases are less effective without indexes, tests where also performed with an index on the date column and also with an index on the text column.

3.1.1.1 Test programs

The programs were written in C# for the .NET platform and developed in Microsoft Visual Studio .NET. The programs that communicated with SQL Server used an SqlConnection and the programs that communicated with Mimer SQL used an OdbcConnection. The reason why different connections were used is that the chosen ones were expectedly the fastest for each database.

The update, select and delete programs were run with around 10000 rows in the database. Every program was run around 100 times and then the average of the measured times was calculated. When the time measured in a program was too small to measure (<0.000 seconds), the statement was executed several times instead and the program measured the time it took to execute all statements. This time was then divided with how many times the statement was executed to get an estimation of how fast one statement took to execute.

3.1.1.1.1 The insert program

The insert program is a program, which inserts rows in the database with the current date and time as well as a string of random length, from 1 to 900 characters. SQL Server limited the maximum length of 900, since that database only can use indexes on columns, which have a width of 900 bytes. The characters in the string are also randomly chosen. This is done before the timing starts and put in an array.

3.1.1.1.2 The update program

The update program is a program where the table is updated. This program can be run in four different versions. All versions update the date in the date column to the current date and time where the condition is satisfied. The first version is where the condition of which row who should be updated is where the text in the text column begins with two randomly chosen letters. Version number two of the program is very similar to the first, only this time the condition is to update rows which have a text in the text column which contains a randomly chosen string of two letters. The third version updates the rows where the date in the datecolumn is older than *x* seconds. The fourth version updates all rows in the table. An example of how the conditions were written in SQL can be seen in table 3.

Table 3. An example of how the conditions were written in SQL code.

Version	Database	Condition
1	both	`where text like 'XY%'`
2	both	`where text like '%XY%'`
3	SQL Server	`where datediff(second, date, getdate()) >= 5`
3	Mimer SQL	`where date < (LOCALTIMESTAMP - interval '5' second)`
4	both	*no condition*

The update program using cursors has the same versions as the update program that not uses cursors. This program was only written and tested in SQL Server, since the goal of this test was to test the OCC in SQL Server. The cursor can be chosen to support OCC or PCC, in the test case the cursor was chosen to support OCC.

3.1.1.1.3 The select program

The select program can be run in four different versions. The first three versions of the select program selects rows which has the same conditions and updates the same column as the first three versions of the update program (see 3.1.1.1.2 The update program). The last version of the program has no condition, it selects the number of rows in the table using the count function, which exists in both databases. The statement looked like this: "`select count(*) from table`".

3.1.1.1.4 The delete program

The delete program also has four different versions. The first three versions also have conditions similar to the ones in the update program. If the condition is satisfied, the row is deleted. In the fourth version, all rows in the table are deleted.

3.1.1.2 Non-concurrent testing

The insert, select, update and delete programs were run non-concurrently and each of these programs measures the time it takes to perform the chosen statement (select, delete, insert or update). The test programs only measures the time it takes to perform the statement, not the time it takes to open the connection to the database or write to file etc. This because the times searched are not how long it takes to set up a connection etc., but to measure how long time the statement takes to execute.

3.1.1.3 Concurrent testing

Combinations of the programs where run concurrently in order to test how the database handles situations where two different programs want access to the same table. This tests the OCC in Mimer SQL as well as the OCC and PCC in SQL Server. When combinations of the programs were run, no indexes were on the table, except

for the clustered index on the ID column. No non-clustered indexes were used since these make a search in the table faster. It was better that the search executed slower so the risk that two connections wanted access to the same row in the table increased. Thus, testing the databases' ability to handle many users at the same time.

When the select- and update programs were run concurrent with another program they both were run in the second version, which updated or selected rows that had a text string in the text column that contained a string of two chosen letters.

The select and update programs were run concurrent in SQL Server. When this combination was run, both programs ran continuously.

The combination of the insert, delete- and select program was run in both databases with the insert- and select programs run continuously. The delete program was run every one and a half second with the condition in SQL Server to delete all rows, which were older than 135 seconds and the condition in Mimer SQL to delete all rows older than 250 seconds. This combination held a constant number of rows of about 10000 rows in each database.

The insert-, delete- and update program were run concurrently in SQL Server. In the test, the insert- and update program were run continuously and the delete program was run every one and a half second with the condition to delete all rows, which had a date older than 160 seconds. This combination held a constant number of rows of about 10000 rows in the database.

3.1.2 Results

When the update program was run in Mimer SQL, very strange results were found. The times measured were not reasonable and after being in contact with Mimer Information Technology they ran similar tests and also received unreasonable results. They gave me a possible explanation to why this happened. When you execute a transaction, the database (Mimer SQL) wants to give back the control to the user as soon as possible. This means that the database only does what is necessary at that point and the table is not updated on disk, only in memory and the transactions are written to a transaction file. Then the control is given back to the user and then the background processes will eventually move the transactions from the transaction file (in memory) to disk. If you during this time continue to execute new transactions, they seem to interfere with the other transactions. Due to this, the update program in Mimer was not further tested and no results are in the following tables. The combinations, which included the update program, were thus not performed in Mimer SQL.

With the condition to choose rows that have a text in the text column that begins with two chosen letters, about 16 rows of the 10000 are chosen. If the condition is to choose rows, which have a text in the text column which contains a string of two chosen letters, about 4700 rows are chosen. The condition to choose rows, which have a date older than *x* seconds, chooses all 10000 rows. The reason for that all rows were chosen with the older condition, was that since the program updates the column where also the condition is tested and the programs ran fast, there was no

specific period of time, where about the same number of rows could be chosen all times the programs were run.

3.1.2.1 Non-concurrent testing

The results of the testing when the programs were run non-concurrently can be seen in table 4. The boxes marked grey represent tests not performed as they were considered being unnecessary. For example, there is no idea to test the conditions where the text in the text column is searched for a specific text string, on a table, which has a date index on the date column.

Table 4. Results of the testing. XY stands for two randomly chosen letters. All times are in seconds.

			SQL Server			**Mimer SQL**		
program	version	condition	no index	date index	text index	no index	date index	text index
insert	1		0.001	0.001	0.002	0.004	0.004	0.012
update	1	text begins with *XY*	0.054		0.020			
	2	text contains *XY*	1.537		2.001			
	3	date older than *x*	0.782	1.418				
	4	*no condition*	0.972	1.537	1.055			
select	1	text begins with *XY*	0.036		0.015	0.055		0.003
	2	text contains *XY*	1.049		0.898	0.105		0.104
	3	date older than *x*	0.011	0.006		0.081	0.081	
	4	*select count(*)*	0.006	0.003	0.006	0.055	0.055	0.055
delete	1	text begins with *XY*	0.083		0.041	0.099		0.095
	2	text contains *XY*	1.181		2.880	2.296		7.202
	3	date older than *x*	0.801	1.418		6.265	6.728	
	4	*no condition*	0.748	3.300	6.247	6.088	6.093	17.840

3.1.2.1.1 The insert program

When comparing the results of the insert program you can see in table 4 that in both databases it takes longer time to insert when you use an index on the text-column. This is because after inserting in a column with a text index, the database has to reorganize the index according to the new text data. When you insert in a column with a date index, the data is inserted last and also in the correct index order, making it easier for the database to calculate the new index. Also when you use a text index, there is an alphabetical operation to compare data, whilst when you use a date index there is a numerical operation instead, meaning that there are fewer comparisons to do. This is why the time it takes to insert when not using indexes and using a date index are about the same. However, it takes longer time to do an insert in Mimer SQL than in SQL Server.

3.1.2.1.2 The update program

Since the update program only could be run in SQL Server, no comparison can be made, but you can in table 4 see the same pattern in the results when it comes to the use of indexes. When you use a text index, the condition to update if the text string in the text column starts with two chosen letters, goes faster. Also, when performing the older-condition, which updates the date column and use date index it takes longer time since the database needs to update the indexing.

The results of the update program with cursors that supported OCC can be seen in table 5. It is apparent that cursors are not that efficient in this particular case (also see 2.1.4 Cursors).

Table 5. Comparison of update runs in SQL Server with and without cursors. XY stands for two randomly chosen letters. All times are in seconds.

version	condition	update cursors	update no cursors
1	text begins with *XY*	0.084	0.054
2	text contains *XY*	3.917	1.537
3	date older than *x*	4.722	0.782
4	*no condition*	4.617	0.972

3.1.2.1.3 The select program

In the results of the select program you can see in table 4 that when you use a text index and perform the select with the condition to select rows where the text in the text column begins with two chosen letters, the program runs much faster, especially in Mimer SQL. When using a text index and perform the select program with the condition to select rows where the text in the text column contains a string of two chosen letters, the results are about the same as when not using an index, since the database has no use of the index. But the program, with this condition, in Mimer SQL runs about ten times faster than the program in SQL Server. Mimer SQL is about ten times slower when performing the statement that counts the rows in the table.

3.1.2.1.4 The delete program

The results of the delete program shows in table 4 that, over all, SQL Server is faster when it comes to executing delete statements. An explanation for the slow times, compared to SQL Server, which were measured in Mimer SQL when performing the delete program, can be that the same problems, which were encountered in the update program, could have occurred in the delete program as well.

3.1.2.2 Concurrent testing

3.1.2.2.1 Select and update

The results of when the select- and update program were run concurrently in SQL Server, can be seen in table 6.

Table 6. Run in SQL Server. All times are in seconds.

select - update		
	select	update
run concurrently	2.085	2.126
run non concurrently	1.049	1.537

As can be seen in table 6, the select took almost twice as long to run compared to when it was run non-concurrently. An update took about 40% longer to execute. This difference in runtimes can be explained by that when both programs are run at the same time, sometimes both statements want access to the same row in the table, forcing a lock of the row. When the statement that first wanted access to the row executes its statement, the other statement has to wait for the first one to finish, which takes time.

When update with cursors was run concurrently with the select program in SQL Server no measurable results was received. The update-statement timed out (timeout set to 30 seconds). An explanation to this strange result which you receive when you run an update program, using cursors and supporting OCC, concurrent with a select program which supports PCC can be that when an update is executed, a select is also executed and it selects one of the rows being changed in the update. This leads to a dirty read and one of the statements, update or select, has to be rolled back and tried again. Since the select supports PCC it cannot be rolled back and tried again, so the update is rolled back and tried again. This leads us back to where we started and the whole thing starts again, eventually leading to a timeout of the update statement.

3.1.2.2.2 Insert, delete and select

The results of when the combination of insert, delete and select programs was run concurrently in SQL Server can be seen in table 7.

Table 7. Run in SQL Server. All times are in seconds.

insert - delete - select			
	insert	delete	select
run concurrently	0.011	1.466	1.693
run non concurrently	0.001	0.070	1.049

With this combination the insertion of a row took about 700% longer to execute (since 0.001 is such a small number, one extra decimal can make a big difference, this is why

in this case one extra decimal is taken in account: 0,0013), the delete took about 2000% longer to execute and the select about 60% longer to execute than when run non-concurrently. When comparing these results to the ones from the test with the update- and select programs there is a difference in how fast the select is performed. When the select- and update programs are run concurrently it takes longer time to do a select, this is probably because there are more rowlocks, and more waiting time in this combination.

The only combination of programs running concurrently in Mimer SQL was the combination of the insert, delete and select programs. The results of this test can be seen in table 8.

Table 8. Run in Mimer SQL. All times are in seconds

insert - delete - select			
	insert	delete	select
run concurrently	0.022	0.170	0.130
run non concurrently	0.004	0.080	0.105

When run concurrently, to insert a row took about 500% longer to execute, the delete statement took about 100% longer to execute and the select statement took just 20 % longer to execute compared to when they were run non-concurrently.

3.1.2.2.3 Insert, delete and update

This combination of insert, delete and update program was only run in SQL Server and the differences in time when run concurrently or not are significant. The results can be seen in table 9.

Table 9. Run in SQL Server. All times are in seconds

insert - delete - update			
	insert	delete	update
run concurrently	0.015	2.882	3.058
run non concurrently	0.001	0.070	1.537

The insert program takes about 1000% longer to execute (here is also one extra decimal taken in account), the delete program about 4000% longer to execute and the update 100% longer to execute. The explanation for that this test takes longer to execute than the test with insert - delete - select is that the update takes longer time to perform than the select since it not just has to find the rows which fulfills the condition, it also has to change the date in the date column.

3.1.3 Conclusions

The fact that the update program in Mimer SQL gave strange results, limited the testing. But this also gave some information, for example that when statements are performed, they are not always written to disk immediately. Instead they are saved in

memory and in a transaction file on disk to be written to disk later. Assuming that it was this quality that made me receive the strange results when running the update program, I question the efficiency of this quality.

When running the programs non-concurrently, in most cases SQL Server is more efficient than Mimer SQL. However, when running programs concurrently, Mimer SQL is more efficient, this can for example be seen in the results of the test when select-, delete- and insert programs were run concurrently. In Mimer SQL the time increases were much lower than in SQL Server. For example, in the test, the delete program in SQL Server took about 2000% longer time to execute when run concurrently and in Mimer SQL the corresponding number was just 100%.

The possibility to use OCC in SQL Server is limited to only be used when having cursors. In this way SQL Server can support both OCC and PCC, but the DBMS is built to support PCC, not OCC. Cursors are far from always being the best way to perform a statement. In my tests you can see that when you run the update program without cursors concurrently with the select program it runs faster than when you run the update program with cursors non-concurrently. This shows that in my test cases, the PCC in SQL Server is more efficient than the OCC in SQL Server both when it comes to running programs concurrently or not.

If you have a database, which is mainly used as storage for data, and the data is not changed often I would say that the best database to use, of the two compared, is SQL Server. On the other hand, if you have a database, which is changed often or accessed by many programs concurrently, the best database to use is Mimer SQL. However, this is only valid when there is not very much data being selected, updated and deleted in the database at the same time, since then you have the problem with the transaction file in Mimer SQL. Both DBMS have their own strengths and when choosing which DBMS to use, it is important to consider what the database will be used for.

3.2 Table vs. Temporary Table (vs. Table Data type)

3.2.1 Method

In this part the goal was to test the temporary storage possibilities in both SQL Server and Mimer SQL. One temporary storage possibility in both SQL Server and Mimer SQL is to use views. However, if you choose to use a view, in both SQL Server and Mimer SQL you can only update and insert rows in a view if the select list contains specifications of column specifications. This implies that you cannot add an extra column that contains for example a number or a timestamp, which can be used for session handling. Session handling is necessary if for example several processes run the same stored procedures, which uses the view at the same time. Also, a view cannot be created within a stored procedure in SQL Server or Mimer SQL. These two properties that views have make them not suitable to use in my tests for temporary storage possibilities for any of the two DBMS, because my tests use stored procedures and the temporary storage type must be able to handle several processes accessing it at the same time. Another alternative to use as temporary

storage in SQL Server is the temporary table or the table data type. Mimer SQL does not support these two storage possibilities. This means that in Mimer, if you need to store temporary results, you have to use something else. You can either use cursors, but this is not the best alternative since cursors are often not that efficient (see 2.1.4 Cursors). Another alternative is to move the logic from the database to the application, however this has many disadvantages (see 2.1.2 Stored Procedures). A third alternative is to make a large select statement with many sub queries, but this makes the code hard to understand and difficult to maintain. A fourth alternative is to use a real table instead, and this is the alternative I chose to use in Mimer because it was an easy solution to create, maintain and understand.

To test performance and differences in the two databases when you need to use temporary tables the following tests were performed. First, two stored procedures: called coarseFilter and fineFilter were each created in four different versions. The first version was for Mimer SQL, which used a real table to store results, the other three versions were created for SQL Server. The second version used a real table for results (same as the first version, but for SQL Server instead), the third a temporary table and the fourth the table data type.

3.2.1.1 The stored procedures

The stored procedure coarseFilter does the following. It selects rows from a table, which is designed in the same way as the table used in the performance testing (3.1 Performance), has no indexed and has 10000 rows in it, where the text string in the text column contains a string of two randomly chosen letters. The selected rows are stored in different kinds of tables, depending on which version of the stored procedure that is run. Here is pseudo code for the stored procedure coarseFilter.

```
insert into tempTable
select * from myTable where text = '%XY%';
```

The stored procedure fineFilter calls the other stored procedure coarseFilter, which inserts rows into some kind of temporary table. Then fineFilter selects rows from this table where the text string in the text column begins with two letters. These two letters are the same as the one used in coarseFilter. Here is pseudo code for the stored procedure fineFilter:

```
create tempTable
call stored procedure coarseFilter
select * from tempTable where text = 'XY%'
drop tempTable
```

In the versions, which use a real table to store intermediate results, the table is not created or dropped in the stored procedure. The table is created before the stored procedure is run and never dropped. This because if you use a real table you only need to create it once, since it is not automatically dropped after the stored procedure is out of scope. In this way you can use the table several times. But if you use a real table, it also implies that several stored procedures can use it to store intermediate results at the same time. To prevent errors that can occur when this happens, some kind of control must be implemented to check which rows in the real table belong to

which stored procedure.

The coarseFilter in SQL Server, which uses a table data type to store the selected rows could not be a stored procedure, instead it had to be a function. The reason for this is that only a function can return a table data type. Also, the table data type is not created in fineFilter. It is instead created in coarseFilter.

3.2.1.2 Performance

Since the table data type is better for smaller datasets and the temporary table is better for larger datasets [12], tests were performed to examine where the limit is when you should use a table data type instead of a temporary table. These test were obviously only performed in SQL Server. To test this the stored procedure coarseFilter, was changed to only return the first *x* rows of the result set. In SQL Server this is done by using the top keyword. When not using the top keyword, about 4700 rows are returned to fineFilter. The top was performed with the following values: 50, 100, 250, 500, 750, 1000, 2000 and 4000, and in all three versions in SQL Server.

3.2.2 Results

The results of when the four versions of fineFilter were run in their respective database can be seen in table 10. The time it takes to create a real table has not been taken in consideration in the times where these are used, since the table is reused and never dropped. The time it takes to create a temporary table and a table data type are less than 0,1 ms and are neglected.

When you use a real table the drop row in table 10 stands for the time it takes to delete the rows in the table, which were used to store the results. When you use a temporary table, delete represents the time it takes to drop the temporary table. When you use a table data type it is dropped automatically and thus the time is 0.000 seconds. I also made some extra tests to confirm that the time it takes to drop the table data type is not significant.

Table 10. Results of the execution of fineFilter. All times are in seconds.

	Mimer	**SQL Server**		
	real table	real table	temp table	table datatype
execute coarseFilter	6.738	1.526	1.265	1.538
select in fineFilter	0.645	0.014	0.015	0.013
drop	5.006	0.750	0.002	0.000
sum	12.389	2.290	1.282	1.551

In Mimer almost half of the total time it takes to execute fineFilter is the deleting of the rows in the table. But the execution of coarseFilter also takes long time compared to the times in SQL Server. In SQL Server the fastest alternative is the temporary

table, which is a little faster than the table data type. Slowest in SQL Server is when you use a real table.

The results of the testing of where the limit is when you should use a temporary table instead of a table data type can be seen in diagram 1.

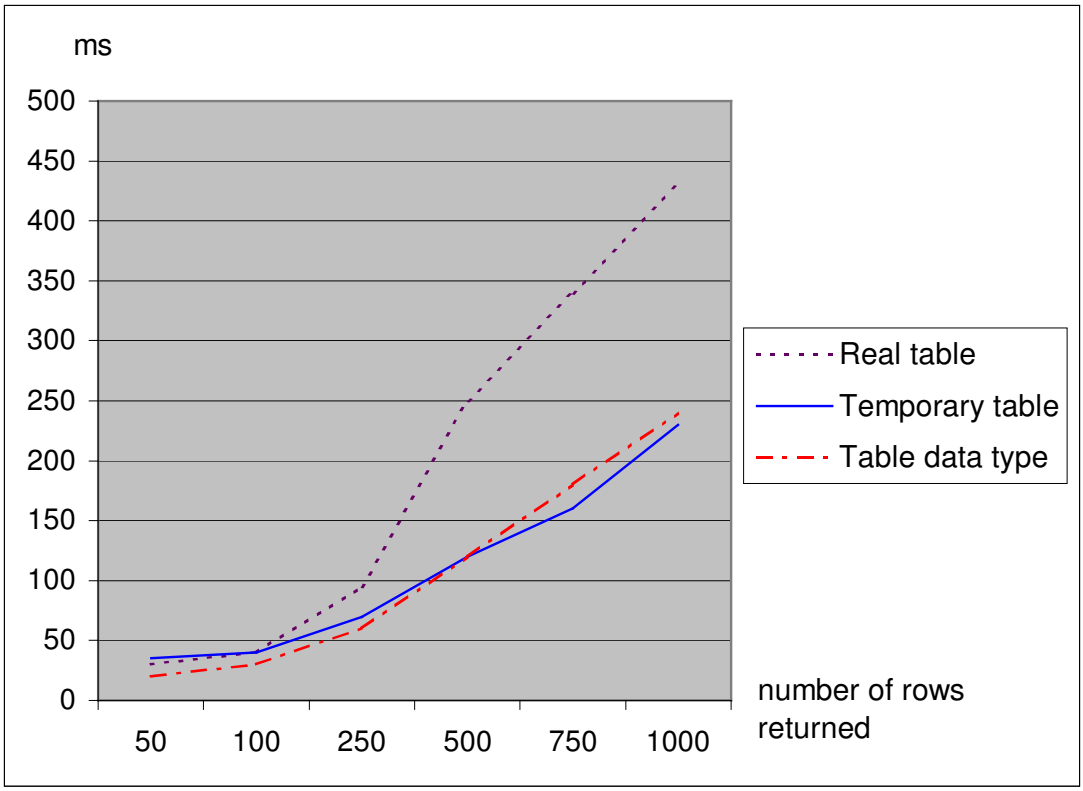

Diagram 1. Differences in runtimes of fineFilter in SQL Server when coarseFilter returns different number of rows to fineFilter.

The diagram shows that the limit is around 500 rows. Meaning that if you have less than 500 rows in the result set it is faster to use a table data type, and otherwise it is faster to use a temporary table. However the difference between the table data type and the temporary table is small.

3.2.3 Conclusions

Since you need to use a function instead of a stored procedure when you want to return a table data type, there are some differences which need to be considered:

- A function can only return one result set, whilst a stored procedure can return many result sets.

- Stored procedures are called by using the execute statement and the in- and output parameters can be specified by parameter names. If the parameters are specified

by parameter names they can come in a different order than specified in the definition of the stored procedure. A function is called by specifying the function name followed by parentheses. Within the parentheses, the input parameters, called arguments, are specified. The arguments cannot be parameter names, they must be values and specified in the order defined in the create function statement.

From the results of my testing it is apparent that the use of a real table when you really need a temporary table is not as effective as when you use a temporary table or a table data type. The fact that Mimer SQL does not support temporary tables or the table data type has some drawbacks:

- If you have a real table, the table is never dropped, thus taking more disk space, taking memory and lowers performance.
- If multiple users try to run the same stored procedure at the same time, to avoid errors you need to use some kind of session handling, which makes the code more complex and harder to maintain.
- If multiple users try to run the same stored procedure at the same time (and you use session handling to avoid errors) it takes longer time since the table has twice as many rows or more in it, which decreases performance.

There are also some advantages with using a temporary table instead of a real table.

- When you use a temporary table, you can cache values from a complex query in the temporary table, which leads to better performance. [13]
- The temporary table is only temporary, once the statement has been executed, the table is dropped, saving memory and increasing performance.
- Multiple users can run the same stored procedure without any conflicts at the same time without making the code more complex.

When it comes to choosing between to use a temporary table or a table data type some factors needs to be considered. Here are some advantages with using a temporary table instead of a table data type in SQL Server.

- Temporary tables are often faster than table data types, when you use large datasets. [12]
- Temporary tables can have non-clustered indexes, which improves performance. [15]
- SQL Server maintains statistics for queries that use temporary tables. If you do not have statistics, SQL Server might choose a less effective processing plan for a query. [12]

- You can change the table definition of a temporary table after it has been created. [15]
- Temporary table can be used in an INSERT EXEC or SELECT INTO statement. [15]

But there are also some advantages with using a table data type instead of a temporary table in SQL Server.

- When you have small datasets, the table data type is faster. [12]
- The table data type result in fewer recompilations of a stored procedure, which improves concurrency. [2]
- The table data type is not affected by a transaction rollback. A transaction rollback erases all data modifications since the start of the transaction. [15] [16]
- The table data type does not require that you lock multiple records, which implies better performance due to fewer locks to create, manage and release. [2]

Views could be an alternative to use as temporary storage, but the drawbacks are that you cannot create a view in a stored procedure and you should not use a view if several processes access it at the same time, since you cannot have any kind of session handling on a view. An advantage with views are that they are supported by both SQL Server and Mimer SQL and that the syntax for creating views are basically the same for both, which makes it easier to migrate from one of the DBMS to the other.

It is unfortunate that Mimer SQL does not support temporary tables, the table data type or a similar data type, but temporary tables will be supported in the next major release of Mimer SQL Engine. But now SQL Server has better support when you need to store temporary results in a table.

As a final conclusion it can be said that temporary tables and the table data type are better to use than a real table, when it comes to storing temporary results. It is also better to use the table data type than a temporary table when you have smaller result sets.

3.3 Migration

3.3.1 Method

In order to test the ability to write non-database specific code and if it is possible to use the same application with different DBMS, a test was performed. In this test, a stored procedure and its sub-stored procedures were taken from *Traci* (a total of 22 stored procedures). These stored procedures were chosen because they performed many calculations, used temporary tables and were a good representation of all functions and qualities used in *Traci*. The intention was to rewrite these stored procedures to work with both SQL Server and Mimer. But I soon discovered that this

was not possible, since the both DBMS has different syntax for many things, including simple things as how to declare a variable. Instead of writing stored procedures, which would work with both databases, I instead translated the stored procedures in SQL Server to work in Mimer.

To investigate the possibility to use different DBMS in two different instances of an application or to change DBMS, I also studied and tested the Mimer Provider Manager.

3.3.2 Results

When I performed this part, I discovered many differences in syntax. I also noticed differences in user friendliness and how it was to work with the two DBMS.

The SQL Query Program that is delivered with Mimer SQL Engine 9.2, called SQL Query Builder has no help pages, which makes it difficult to learn how the program works in an easy way. Also, Mimer SQL Engine Documentation which is the Help pages for Mimer SQL are not as detailed described as the ones in SQL Books Online, which are the Help pages for SQL Server. This makes it much easier for an inexperienced searcher to get help in SQL Server than in Mimer SQL. I found it really frustrating when I got an error message when running a stored procedure in Mimer and then trying to search in the Mimer SQL Engine Documentation to find what the problem was. Often when this occurred I had to search for a long time before I found what I was searching for, if I ever found it.

When I tried to move the tables, which were used in the stored procedures, from SQL Server to Mimer SQL I had some problems. After exporting the tables and its data from SQL Server, I tried to import them into Mimer SQL. However, only the tables were created without any data in it. Since there was no documentation of how to import a text file with data into Mimer SQL in the Mimer SQL Engine Documentation and after trying many different ways to import the data without any success, I decided I had to import the data in a different way. The easiest and fastest way for me was to write my own program, which took the data from the tables in SQL Server and inserted it into Mimer SQL. In this way the data was copied from SQL Server into Mimer SQL.

There are some differences in syntax when writing SQL-statements in Mimer SQL and SQL Server. Here follows the ones I discovered when translating the stored procedures:

- In Mimer SQL a semicolon is added after each statement, this is optional in SQL Server.

- The datediff function in SQL Server does not exist in Mimer SQL. Here is an example of how to convert to Mimer SQL.

```
SQL Server:     datediff (month, date1, date2)
Mimer SQL:      (date2 - date1) month(3);
```

- The dateadd function in SQL Server does not exist in Mimer SQL. Here is an example of the differences:

```
SQL Server:   dateadd (day, 3, myDate)
Mimer SQL:    myDate + interval '3' day;
```

- The datepart function in SQL Server does not exist in Mimer SQL. The same functionality can be achieved in this way:

```
SQL Server:   datepart (year, myDate)
Mimer SQL:    extract (year from myDate);
```

  In Mimer SQL the possible values to extract are: year, month, day, hour, minute or second.

- If you need to find out the day of a week, it is done differently. An example:

```
SQL Server:   datepart (weekday, myDate)
Mimer SQL:    dayofweek (myDate);
```

  Both of the above functions returns an integer, which indicates the day of the week, for example 1 = Monday, 7 = Sunday. In SQL Server the first day of a week can be set with the 'set datefirst' statement.

- To get the current date and time, the syntax in the two DBMS are:

```
SQL Server:   current_timestamp or getdate()
Mimer SQL:    localtimestamp(0);
```

  In Mimer SQL, the zero indicates the number of milliseconds precision, a number between 0 and 9 is allowed, 6 is default.

- Some commonly used data types have different names, see table 11.

Table 11. Different datatypes.

SQL Server	Mimer SQL
bit	OdbcBit
datetime	timestamp

- Declarations of variables are done in a different way. In Mimer SQL you do not use the prefix '@' to identify a variable. An example:

```
SQL Server:   declare @myVariable  numeric(9)
Mimer SQL:    declare myVariable    numeric(9);
```

- Comments in Mimer SQL are done with the prefix '--' on every row, which is a comment. The /* comment */ syntax is not valid in Mimer SQL.

- You create stored procedures differently as well. An example:

SQL Server:
```
create procedure myStoredProcedure
(
    @personID       numeric(18,0) = NULL,
    @name           varchar(10) = NULL OUTPUT,
    @address        varchar(20) = NULL OUTPUT
)
as
begin
    ...
end
```

Mimer SQL:
```
create  procedure myStoredProcedure
(
    personID        numeric(18,0),
    OUT name        varchar(10),
    INOUT address   varchar(20)
)
reads sql data
begin
    ...
end
```

The differences are that when you have output parameters in Mimer they are either OUT or INOUT parameters. If neither of these is specified, IN is default. If a variable is specified to be IN, then it is read-only, if OUT it is write only. If a parameter is INOUT in a sub procedure then the parameter also has to be INOUT in its procedure above. In Mimer SQL, it is not possible to have a default value if the parameter is not present, which can be done in SQL Server (see above '= NULL'), since in Mimer SQL you have to specify all parameters. This implies that if you run a stored procedure, which has five parameters, then five parameters have to be sent into the stored procedure. Another difference is that in Mimer SQL following the parameter specification you have to have one of the three following access options:

- **Contains sql**: The procedure cannot perform data-manipulation statements and it can only call other functions and procedures which has the access option Contains sql. With this option the procedure can neither write nor read from the database
- **Reads sql data**: This option allows the procedure to perform any procedural-sql-statements, except for updates i.e. insert, delete and update. The procedure can only call other functions and procedures, which has the access option Contains sql or Reads sql data. This option prevents the procedure to write data into the database.

- **Modifies sql data:** With this option all procedural-sql-statements are permitted and any procedure or function can be called. Thie option allows a procedure to perform both read and write operations on data in the database.

- The stored procedures in Mimer SQL cannot return a result set, unless you use a Result Set Procedure. These Result Set Procedures can return result sets, but they cannot have the access option modifies sql data and this means that the sub-procedures neither can have that access option.

- When you run a stored procedure in Mimer SQL you use 'call' instead of 'exec' (or 'execute'), which is used in SQL Server. Also, in Mimer SQL you cannot have NULL as an input parameter. In Mimer SQL it is not possible to name an input parameter, thus the parameters must be in the correct order. In SQL Server it is possible to name the parameters and specify them in a different order than when they were declared. SQL Server even allows you not to specify all parameters. An example:

```
SQL Server: exec myStoredProcedure      @address = @addressOUT,
                                        @personID = 1

Mimer SQL: call myStoredProcedure (     1,
                                        nameOUT
                                        addressOUT );
```

  In the example above you can see the differences described. In SQL Server you can specify the parameters by name, you do not need to specify the parameters in a specific order and you do not have to specify all parameters.

- The function isnull, which exists in SQL Server, does not exist in Mimer SQL. The function works like this:

```
isnull ( x1, x2 ) == case when x1 is not null then x1
                                        else x2
```

  In Mimer SQL there exists a function with the same functionality, which is called coalesce. The function coalesce exists in both SQL Server and Mimer SQL. The only difference from isnull is that coalesce can handle more values than two. Coalesce works like this:

```
coalesce (x1, x2,...,xn) == case when x1 is not null then x1
                                        else coalesce (x2,...,xn)
```

- The 'set nocount' statement, which controls if the number of affected rows affected by a statement should be returned or not in SQL Server, does not exist in Mimer SQL.

- The comparison operator for 'not equal to' is in both Mimer SQL and SQL Server '<>', but in SQL Server you can also use '!='. The operator '<>' is standard SQL-99.

- In Mimer SQL you cannot have variable names called 'value' or 'exists'. The reason for that you can have variables with these names in SQL Server, is that the variables in SQL Server has the prefix '@' and the string '@value' is not equal to the string 'value'.

- If-conditions are handled different in the two DBMS. Here is an example:

```
SQL Server:   if ( condition )
              begin
                  ...
              end
              else
                  ...

Mimer SQL:    if ( condition ) then
                  ...;
              else
                  ...;
              end if;
```

- When you need to cast a variable to another data type you can use the cast function, for example cast (myVariable as varchar(20)) in both Mimer SQL and SQL Server. In SQL Server you can also use the convert function, for example convert (varchar(20), myVariable), but this function does not exist in Mimer SQL.

- There are also differences in syntax when selecting some values from a table into variables, here is an example:

```
SQL Server:   select @internalConst = NumericValue
              from TraciConstants
              where name = 'Intern tid'
Mimer SQL:    select "NumericValue"
              into internalConst
              from "TraciConstants"
              where "Name" = 'Intern tid';
```

  As seen in the example above, in Mimer SQL you use 'select x into y' instead of 'select y = x'. You also need to use " " around the column names, to separate the column names from the variable names.

- The handlings of cursors are a bit different. How you close and deallocate a

cursor is a bit different. An example:

```
SQL Server:   close myCursor
              deallocate myCursor
Mimer SQL:    close myCursor release;
```

After fetching from the cursor, in SQL Server the function @@fetch_status is executed, this function returns -1 if the cursor is at the end and 0 if it is not. In Mimer SQL you instead use an exception handler. If the cursor is at the end, a 'not found' exception is raised. Here is an example of how you handle an 'not found' exception:

```
Mimer SQL:    declare exit handler for NOT FOUND
              begin
                  leave myCursor_loop;
              end;
```

The above statements indicate that when the cursor is at the end, a 'not found' exception is raised and the next statement to be run is the one after the myCursor_loop has ended in the sql code.

In SQL Server's query program SQL Query Analyzer (I used version 8.00.760) you can see and edit a stored procedure by right clicking on the stored procedure in the object tree. The stored procedure is then fetched from the database and displayed to the user. You also have the possibility to save the written stored procedure in a file. An advantage with SQL Query Analyzer is that if you right click on the stored procedure in the object tree, you can always be sure that the stored procedure shown to you is the one that is in the database. You also have the possibility to use a versioning system on your stored procedures if you save them on file. In Mimer SQL's query program QBE Vision Query Builder (I used version 7.1.1.001) you need to save the written stored procedure in a file, to be able to see it later. You can use a versioning system, since you save the stored procedures on file. But you cannot be sure that the stored procedure in the file is the one that actually is in the database.

The possibilities offered by the two DBMS query programs to debug and the see query plans and statistics are different. Mimer's QBE Vision Query Builder offers a debug option where you can see the queries in a new window before they are sent to the server. But there is no possibility the step row by row and check where an eventual error is located. The query program also offers an SQL statistics option, this option is however not available for some DBMS, including Mimer SQL and SQL Server. This means that with QBE Vision Query Program, which is delivered with Mimer you have poor possibilities to debug and see query plans. However, with Mimer SQL you also get an extra program called Mimer PSM Debugger, which is a program where you can debug your stored procedures. You can have breakpoints to stop in the stored procedure and you can see the values of the variables. When I tested this program I discovered some difficulties. First, you can only have breakpoints on certain rows decided by the program and second, when I tried to test my biggest stored procedures, the program could not handle it, it simply froze or

returned error messages. SQL Server's SQL Query Analyzer offers much better possibilities to debug and see statistics. In the program it is easy to find and use all different options. For example you can see the execution plan, server trace and client statistics. The server trace is really useful if you need to make the code more efficient, since with this option activated you can see how much time it takes to complete a statement and also how many reads and writes the statement performed.

I think that the Mimer Provider Manager is a usable tool to use. Even if you do not plan to change DBMS or plan to use the application with different DBMS, you never know what might happen. To me, using the Mimer Provider Manager is being a step ahead. If you want to change DBMS in the future, you do not have to change any code in the application.

In Mimer SQL Engine Documentation you can see if a function or specification is standard SQL and which version it supports. This is an advantage if you would like to write a program, which only supports standard SQL.

3.3.3 Conclusions

The major conclusion found during this phase, was that if you have an application, which has SQL that does more than some simple select statements, you couldn't communicate with both SQL Server and Mimer SQL since the syntax differs too much.

In this part I discovered during my tests that the control of what a stored procedure has permissions to perform is different in the two DBMS. In Mimer SQL you have to specify whether a stored procedure shall have permission to read and/or write in the database. In SQL Server there is no such control, all stored procedures can read and write into the database without having to specify this permission. According to me, the control of this in Mimer SQL has some drawbacks. One is that if you add functionality, which requires another access option, you need to change the access option before the stored procedure can be executed. Another is that a Result Set Procedure cannot have the access option 'modifies sql data'. This fact makes Mimer SQL not suitable for using as DBMS in *Traci*, since some stored procedures in *Traci* need to write data into the database and return a result set. This is not possible in Mimer SQL since if you need to return a result set you need to use a Result Set Procedure, and these cannot write data into the database.

In a developer's point of view, I have to say that during my testing I have found that SQL Server is user-friendlier with its more extensive help and more functions to use in the SQL-code. These extra functions are though a drawback when it comes to using the same SQL-code with different databases, since most extra functions are only supported by one vendor. In some cases SQL Server has support for two different syntaxes, one which is standard (and probably the syntax used in Mimer SQL) and one, which is SQL Server-specific, an example for this, is the syntax for 'not equal to'. Standard is '<>' and SQL Server also supports '!=', which is the common syntax in many other programming languages for example, C and Java. This is to me an example of SQL Server's will to help the developer.

6. Summary

In the first part I discovered that SQL Server is faster when it comes to running programs non-concurrently and as expected, Mimer SQL is faster when programs are running concurrently. However, a surprise in this part was that Mimer SQL is so slow. In many cases Mimer SQL was much slower than SQL Server, even when performing simple statements.

In the second part it was established that Mimer SQL has no temporary result storage possibilities comparable to SQL Server's temporary tables or table data type if you want to use the temporary storage in a stored procedures or want to use session handling on the temporary storage. If you do not have to use temporary storage in stored procedures or use session handling, an alternative to use is a view. Both SQL Server and Mimer SQL support views. An advantage with using views is that the syntax for creating views is basically the same in both SQL Server and Mimer SQL.

The third part discussed migrating issues, the possibility to write non-specific database code, syntax differences and also how it is to work with the two DBMS. The tests showed that it is almost impossible to write code that can communicate with both the chosen databases; a major reason for this is that the syntax for declaration of variables is different. Also, I found it easier to work with SQL Server because of its extensive help and many functions in the SQL Query Analyzer. The Mimer Provider Manager is a very helpful and effective tool, which can be used when you have an application, which needs to communicate with different DBMS.

7. Recommendations

As a general recommendation to NTier Solutions, I would not recommend them to use Mimer SQL as DBMS in their system *Traci*. Because of the lack of temporary tables and that you cannot return result sets from the stored procedures and do changes in the database within the same stored procedure.

I would not recommend them, in the general case, to rewrite any code from one application, which uses SQL Server as DBMS to use Mimer SQL instead, since this extensive rewriting work does not provide much in time gains. However, if they had an application, which had many simultaneous users that accessed the database, a migration to Mimer SQL would be suitable. Also if they have an application, which does not have much of its logic in stored procedures, a migration could also be suitable.

It is my opinion that SQL Server is much more user friendly than Mimer SQL. This is because SQL Server has a much more extensive help and better possibilities to debug and see statistics in the SQL Query Analyzer. Thus, I would recommend NTier Solutions to use SQL Server if they want a DBMS, which is easy to work with and has many usable functions.

I would also recommend NTier Solutions to use the Mimer Provider Manager with their applications, since this makes it much easier to change from one DBMS to

another, or to have use an application with different DBMS.

8. Acknowledgements

I would like to thank NTier Solutions for giving me the opportunity to do my master's project at their company. I would especially like to thank Emil Gustafsson, who was my supervisor at the company, for helping me and answering my questions. Thanks also to my family and friends for listening to my thoughts about the thesis. Also, thanks to Serafim Dahl who has been my supervisor at KTH.

9. Resources

[1] Ambler, S, (2004) *Concurrency Control*
http://www.agiledata.org/essays/concurrencyControl.html

[2] Bromberg, P. *Using SQL Server 2000 Table Variables to eliminate the need for Cursors*
http://www.eggheadcafe.com/articles/20010823.asp

[3] Carpentiere, C. (2004). *An Evaluation of Stored Procedures for the .NET Developer*
http://msdn.microsoft.com/netframework/default.aspx?pull=/library/en-us/dndotnet/html/storedprocsnetdev2.asp

[4] Chigrik, A. *Using SQL Server Cursors*
http://www.mssqlcity.com/Articles/General/UseCursor.htm

[5] Danielsson, L (2004). Mimer dömer ut Microsoft. *Computer Sweden*, 2004 April 2nd, page 16

[6] Elmasri, R & Navathe, S (1994). *Fundamentals of Database Systems* 2nd edition. The Benjamin/Cummings Publishing Company, Inc, Redwood City. ISBN 0-8053-1753-9

[7] Kumar, *SQL Server 2000 Best Practices*
http://www.extremeexperts.com/sql/articles/BestPractices.aspx

[8] *Mimer Information Technology*
http://www.mimer.com/ImageBottom.asp?secId=181

[9] *Mimer Provider Manager Documentation*

[10] *Mimer SQL*
www.mimer.com

[11] *Mimer SQL Engine*
http://www.mimer.com/imageBottom.asp?secId=200

[12] Moran, B. (2003) *Table Variables vs. Temporary Tables*
http://www.winnetmag.com/SQLServer/Article/ArticleID/40404/40404.html

[13] Sengupta, J. *Performance Enhancements Using Temporary Tables*
http://www.devx.com/tips/Tip/14077

[14] *Stored Procedures*
http://java.sun.com/docs/books/tutorial/jdbc/basics/storedprocedures.html

[15] *SQL Server 2000 - Table Variables*
http://support.microsoft.com/default.aspx?scid=kb;en-us;305977&Product=sql2k

[16] *SQL Server Books Online*

[17] *Tips for Reducing SQL Server Deadlocks*
http://www.sql-server-performance.com/deadlocks.asp

[18] *Transaction Concurrency - Optimistic Concurrency Control*
http://developer.mimer.com/features/feature_15.htm

[19] *Understanding SQL*
http://www.faqs.org/docs/ppbook/c1164.htm

[20] *Using clustered indexes*
http://www.schemamania.org/jkl/booksonline/SQLBOL70/html/8_des_05_2.htm

[21] *Views, managing*
http://www.comp.nus.edu.sg/~ooibc/courses/sql/ddl_view.htm

All links worked as of 2004-08-18