Jakub OTRZĄSEK, Wojciech MUELLER, Krzysztof KOSZELA
Uniwersytet Przyrodniczy w Poznaniu, Instytut Inżynierii Rolniczej
ul. Wojska Polskiego 50, 60-627 Poznań
e-mail: koszela@up.poznan.pl

METHODOLOGY OF COMPARING THE PERFORMANCE OF SQL INSERT OPERATIONS IN SELECTED RDBMS

Summary

Currently, the majority of the existing IT systems and those that are under development are based on applications that work with database servers. The ever wider range of functionalities that such servers are able to provide results in an increase in the number of their users and in the amount of data processing. This leads to performance problems. Such problems are addressed through both programming and hardware solutions. In terms of programmatic solutions the authors' interest is focused on table structures in RDBMS's and their performance in terms of write operations on data. So far analyses have been conducted relating to the impact of the presence of a primary key and the way it is generated on data write operations in a sample of relational databases.
***Key words**: computer systems; functionality; relational databases*

METODYKA PORÓWNYWANIA WYDAJNOŚCI OPERACJI SQL INSERT W WYBRANYCH RDBMS

Streszczenie

Aktualnie większość istniejących i tworzonych systemów informatycznych to aplikacje współpracujące z serwerami bazodanowymi. Oferowana przez nie coraz szersza gama funkcjonalności skutkuje zarówno wzrostem liczby użytkowników, jak i przetwarzanych danych. Powoduje to pojawienie się problemów z wydajnością. Rozwiązywane są one zarówno poprzez działania programistyczne, jak i sprzętowe. Przedmiotem zainteresowania autorów z obszaru działań programistycznych stała się definicja (struktura) samej tabeli w RDBMS z perspektywy wydajności zapisu danych. Dotychczas przeanalizowano wpływ klucza podstawowego i sposobu jego generowania na zapis danych w przykładowo wybranych bazach relacyjnych.
***Słowa kluczowe**: systemy informatyczne; funkcjonalność; relacyjne bazy danych*

1. Introduction

A key issue of growing relevance that is taken into account during the stages of design, development and operation of IT database systems is the problem of application performance and scalability [1]. Generally, during the multi-staged process of designing data structures, our attention is focused on mapping data correctly in accordance with any requirements or architectural data model that have been identified or adopted. The issue of performance in itself is a complex one as it can be analyzed in the context of the various operations performed on a production database or in the context of using it to develop analytical databases [1] [7] [8]. Within this wide area of issues as outlined above the authors' attention is concentrated on the impact of the existence of a primary key and the way it is generated on performance of data write operations within Relational Database Management Systems (RDBMS). The approach adopted here reflects the perspective of the users of RDBMS as a product. Therefore, the issues pertaining to the processes of spatial distribution of data in the drive were ignored.

2. Methodology

Our attempt to answer the question as to what extent the presence of a primary key and the way it is generated affects the process of writing data in relational databases required the creation of the methodological framework and programmatic tools to be developed.

At the initial stage, a number of assumptions were set down which, if adhered to, should eliminate the impact of any factors that might affect measurements. Among other things, it was decided that the application should be run solely on the database server side. This would eliminate the impact of any programming interfaces used to provide access to data such as ADO, ADO.NET, etc. on the time of rows writing. The consequence of this decision was that our attention was focused only on those RDBMS's whose internal languages are of the procedural type.

Any resulting procedure or procedures to be used as our basic research tool should be able to create a table structure, generate data rows and then save them. At the same time what should be recorded are the start and end times of write operations on the data and the number of rows inserted in the database. This second piece of information, which will be required for analysis, should be stored in a separate table.

Since the authors' primary goal was to answer the question as to what extent the presence of a primary key and the way it is generated affects the speed of write operations it was decided to write a series of separate independent procedures for each scenario under examination, where the common element would be a record of the experiment results stored in the same table. The following scenarios were considered based on the type of information stored in the database:

- data without a primary key,
- data with a primary key whose values are generated by the RDBMS,
- data with a primary key, where the values are generated on the basis of an algorithm provided by the user.

Procedures to provide this solution should be implemented under at least two RDBMS's. It was decided that these would be SQL Server 2008, Oracle XE and MySQL. The deciding factor was the widespread use of these IT tools. For example, the free version of MSSQL is used in Poland with a number of well-known programs including Płatnik and MicroSubiekt. Furthermore, its extended (paid) version is used in CDN XL, one of the most popular ERP programs on the Polish domestic market. MySQL, in turn, dominates the market of Internet applications, where most of the scripts that are used to run forums are based on the Apache, PHP and MySQL trio. Oracle's database software enjoys an excellent reputation as a favoured solution among the IT community, which is why its free version (XE) was also tested in the same way as MSSQL and MySQL.

Another methodological assumption, which is frequently adopted in database research, was that a there was a simple primary key which assumed int. values.

Although the methodology adopted here does not entirely reflect real operations, the present authors believe that it is a proper tool for comparative research. The present research and analysis purposefully forwent specialized tools for data feeding (ELT) used in the creation of data warehouses, and instead focused on OLTP database solutions.

3. Research tools - procedures

The above mentioned test procedures were first developed in T-SQL, which is an in-built language available as part of SQL Server 2008R2. It allows the user not only to manipulate data (components of the DML language - Data Modification Language) but also define structures (components of the DDL language - Data Definition Language). These functionalities of T-SQL were made use of in the following procedure which covers the case of writing data in a table without a pre-defined primary key. A decision was also taken to have partial results of the experiment saved in an additional table. To create the aggregates that would form the partial results stored in the table, a grouping query was used along with the DateDiff function. The group consisted of rows that had been saved in the database within the same time interval of one second. This method of grouping was possible because one of the fields of the primary table (simple_insert_table) contained the system date and time returned by the GetDate function.

```
CREATE PROCEDURE [dbo].[simple_insert]
@numerow int=1000;
@numtries int= 100;
AS
BEGIN
SET NOCOUNT ON;
declare @testb datetime;
declare @tests datetime;

create table simple_insert_table (a int, b varchar(10), c datetime );

declare @numtries1 int=@numtries;
declare @numerow1 int=@numerow;

set @testb = GETDATE();
while @numtries >0
begin
set @numerow = @numerow1;
while @numerow >0
begin
insert into simple_insert_table values (1,'1234567890',GETDATE());
set @numerow = @numerow -1;
end;
set @numtries = @numtries -1;
end;
set @tests=GETDATE();

insert into test_results (test_name,start_d,stop_d,param1,param2)
values ('simple insert test',@testb,@tests,@numerow1,@numtries1);

set @numerow = @@IDENTITY;
insert into test_subresult (test_id, ins_num, time_agr)
select @numerow,count (DATEDIFF(S, '19700101', c)) e, DATEDIFF(S, '19700101', c) from
simple_insert_table
group by DATEDIFF(S, '19700101', c);
insert into test_results (test_name,param1,param2,param3,param4,param5)
select 'statistic for simple insert: '+ cast(@numerow as varchar),exec_per,max_ins,avg_ins,min_ins,stddev_ins from
dbo.simple_insert_stat;
drop table simple_insert_table;

END
```

With minor changes made to the code shown above we were able to create new procedures to be used as testing tools for two subsequent cases. In the first scenario, the key was generated by an algorithm included in the procedure. The variable used to forward the value of the generated key to the query was at the same time a component of the condition affecting the number of loops performed. In the second case, the relevant tools of the database server were used to generate the key. In the case of the SQL Server 2008R2 the identity procedure was used, and at the level of MySQL autoincrement was used. In RDBMS Oracle the sequence procedure was used. The above tools were manipulated only in terms of their initial settings, including the initial value and jump, which were identical in all the cases. In order to ensure that generalizations could be made the algorithm was written in the PL/SQL language and in the procedural extension of the SQL language for MySQL 5.5.8, which allowed for tests to be performed using other RDBMS's.

Although at the time the present research was conducted a new version of the SQL Server appeared (SQL Server 2011), which this time featured two procedures enabling the automatic creation of a primary key, this version is not final and was therefore not subject to experimentation. The additional tool facilitating key generation at the level of RDBMS is the sequence procedure [8].

4. Tests and results

The tests were conducted in a virtual environment set up under Windows XP Home Edition with the latest updates installed. The installation was of the standard type as was the subsequent installation of RDBMS (with all default settings of the wizard accepted). After the completion of testing on an individual RDBMS the research environment was reinstalled. Each test was run on a server machine attached to an array in the Windows Server 2008 environment (except for the Oracle tests). The procedures for the Oracle database were tested on a server with the Ubuntu Server 11 operating system also attached to an efficient array. The results from the server environments were analogous with the research machine (relations of test results for a given RDBMS).

The next step was to create for each RDBMS to be tested an empty database also with the default settings of the wizard preserved. This allowed us to have testing procedures embedded in it and create a relational structure for the purpose of collecting test results.

Tests were conducted for each RDBMS for the following scenarios described in detail below during which 10^6 data rows were inserted into the database:

- insertion of data without a primary key,
- insertion of data with a primary key generated based on an algorithm proposed by the authors,
- insertion of data with a primary key generated using the generator of the RDBMS.

The results of the tests are presented in tabular form in Tables 1, 2, 3 and 4. The tables show relative values benchmarked against the results obtained for the first scenario under examination. This procedure was applied to the results obtained for each RDBMS tested. In the case of MySQL 5.5.8 tests were performed with two different engines INNODB and MyISAM, which are available for this particular RDBMS. However, the authors found it inadvisable to include the relative values obtained for the case where write operations were performed on data rows without a primary key.

Table 1. Characteristics of write operations under SQL Server 2008R2

Type of data	Relative duration of write operation for all rows	Relative write speed on data rows [1/s]	Maximum relative write speed on data rows [1/s]	Minimum relative write speed on data rows [1/s]	Modal value of relative write speed [1/s]	Median of relative write speed [1/s]
with a primary key (scenario 2)	1.013	0.986	1.150	0.591	0.998	0.988
with a primary key (scenario 3)	1.017	0.982	1.073	1.226	0.973	0.972

Table 2. Characteristics of write operations under Oracle XE 10.2.0

Type of data	Relative duration of write operation for all rows	Relative write speed on data rows [1/s]	Maximum relative write speed on data rows [1/s]	Minimum relative write speed on data rows [1/s]	Modal value of relative write speed [1/s]	Median of relative write speed [1/s]
with a primary key (scenario 2)	1.374	0.729	0.459	0.009	0.918	0.796
with a primary key (scenario 3)	1.237	0.810	0.818	0.169	0.768	0.800

Table 3. Characteristics of write operations under MySQL 5.5.8 (INNODB)

Type of data	Relative duration of write operation for all rows	Relative write speed on data rows [1/s]	Maximum relative write speed on data rows [1/s]	Minimum relative write speed on data rows [1/s]	Modal value of relative write speed [1/s]	Median of relative write speed [1/s]
with a primary key (scenario 2)	0.864	1.157	0.997	-	0.993	1.119
with a primary key (scenario 3)	0.955	1.048	0.984	-	0.985	1.048

Table 4. Characteristics of write operations under MySQL 5.5.8 (MyISAM)

Type of data	Relative duration of write operation for all rows	Relative write speed on data rows [1/s]	Maximum relative write speed on data rows [1/s]	Minimum relative write speed on data rows [1/s]	Modal value of relative write speed [1/s]	Median of relative write speed [1/s]
with a primary key (scenario 2)	1.432	0.708	0.439	0.391	0.775	0.796
with a primary key (scenario 3)	1.455	0.699	0.431	0.720	0.699	0.776

5. Discussion of results

The above test results for write operations on data with and without a primary key do not differ significantly from each other in the case of SQL Server 2008 R2. This applies equally to situations when the key is generated by the RDBMS itself and by an algorithm embedded in the testing procedure. The delay on write operations on data containing a primary key for a write speed of 5000 rows per second and where the number of data rows equals 10^6 is 10 seconds. An important note to be made here is that during the tests no decrease was observed in write speeds on rows that would correspond to any subsequent increase in the number of rows in the table.

A similar observation was made in the case of MySQL 5.5.8 with the InnoDB engine with the exception that the system proved more sensitive to the way a primary key was generated. Another surprising dependence that can be incurred from the results obtained in the test conducted on this RDBMS is an increase in the relative speed of write operations on data with a primary key. This applies equally to situations where the primary key is generated by the database system and by a procedure. Another fact that was revealed during the study and which requires explanation is the zero minimum speed values obtained during the tests carried out on MySQL 5.5.8 [5]. This resulted in a significant increase in standard deviation values and unspecified values of minimum relative speed.

These kinds of regularities were not found to exist earlier when the engine MyISAM was used under MySQL 5.5.8 [4]. The trends that were found to exist in this case, based on the results of measurements, remain consistent with the results obtained for Oracle XE, which confirm the previous belief that the introduction of a key will slow down the process of writing new rows into the table. However, significant drawbacks of the MyISAM engine should be borne in mind, i.e. its inability to create transactions and define referential integrity constraints [6].

6. Summary

Modern RDBMS are very different in terms of implementation of the relational model, which makes it difficult to formulate generalizations and rules regarding the performance of these IT systems. In some cases, an intuitive solution or a solution based on past experience with RDBMS's may prove to be ineffective when working with new types of database systems, so the authors recommend that any proposed solutions be tested beforehand, in particular, before the start of development work on software (especially in situations where OLAP structures are created). Our research and analysis of its results in relation to three different RDBMS's prompt us to formulate the following observations and conclusions:

- When data is provided with a primary key this generally reduces write performance on data rows in each of the RDBMS's tested with the exception of MySQL 5.5 with the InnoDB engine enabled. The amount of decrease in performance is varied but from the perspective of SQL Server 2008R2 it can be said to be insignificant.
- The primary key generation mechanism that is built into a RDBMS will generally provide a better solution in terms of write performance on new rows than *bespoke* software solutions embedded within procedures.
- It seems advisable to undertake further research in this area in order to explain the unusual behaviour of MySQL 5.5.8 with the InnoDB engine enabled in terms of write speeds on data with and without a primary key.
- The overall times of writing data in the analyzed cases and in different RDBMS that were recorded given the adopted methodology suggest that from the perspective of OLTP applications efficiency gains attributable to not using a simple primary key are insignificant.

7. Bibliography

[1] Beynon-Davies P.: Systemy baz danych. Warszawa: WNT, 2003.
[2] Celko J.: SQL for Smarties, Advanced SQL Programming, 3 Edition, 2005.
[3] MySQL Reference Manual - http://dev.mysql.com/doc/refman/5.5/en/mysql-nutshell.html 2011
[4] MySQL MyISAM Storage Engine Manual - http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html
[5] MySQL InnoDB - http://dev.mysql.com/doc/refman/5.5/en/innodb-storage-engine.html
[6] MySQL refman - http://dev.mysql.com/doc/refman/5.0/en/ansi-diff-foreign-keys.html
[7] Ponniah P.: Data Warehousing Fundamentals for IT Professionals, 2010.
[8] SQL Server perfomance - http://www.sql-server-performance.com/articles/dev/sequence_sql_server_2011_p1.aspx 2011.