Analysis of the Blocking Behaviour of Schema Transformations in Relational Database Systems

Lesley Wevers ✉, Matthijs Hofstra, Menno Tammens,
Marieke Huisman, and Maurice van Keulen

University of Twente, Enschede, the Netherlands
l.wevers@utwente.nl,{m.hofstra,m.j.tammens}@student.utwente.nl
m.huisman@utwente.nl,m.vankeulen@utwente.nl

Abstract. In earlier work we have extended the TPC-C benchmark with basic and complex schema transformations. This paper uses this benchmark to investigate the blocking behaviour of online schema transformations in PostgreSQL, MySQL and Oracle 11g. First we discuss experiments using the data definition language of the DBMSs, which show that all complex operations are blocking, while we have mixed results for basic transformations. Second, we look at a technique for online schema transformations by Ronström, based on triggers. Our experiments show that pt-online-schema-change for MySQL and *DBMS_REDEFINITION* for Oracle can perform basic transformations without blocking, however, support for complex transformations is missing. To conclude, we provide a solution outline for complex non-blocking transformations.

1 Introduction

Software is in constant need of maintenance, adaptation and extension. For applications storing and maintaining data in a database, a software change often involves restructuring of data, i.e., a schema change with an accompanying conversion of the data. To ensure that no concurrency conflicts occur, many relational database systems block access to the data during a schema change. The effect is that concurrent transactions completely halt until the execution of the schema change has finished, which could take many hours to days for large databases. This is a real problem for systems that need 24/7 availability, such as telecommunication systems, payment systems and control systems [5],[7].

Goals We experimentally investigate the blocking behaviour of online schema transformations in current DBMSs. We look at the capabilities provided by the standard SQL data definition language (DDL) as implemented by the DBMSs, and we investigate a method developed by Ronström [6], which can perform non-blocking schema changes on any DBMS that supports triggers. We investigate basic transformations provided by the SQL DDL such as adding columns and indexes, and we look at complex transformations that require multiple DDL operations, such as changing the cardinality of a relationship, or changing the primary key of a table. While the basic transformations are the most common, these complex transformations are often needed in realistic transformations.

Contenders We investigate PostgreSQL, MySQL and Oracle 11g, which represent a large fraction of the DBMSs used in industry. We now provide a brief overview of their capabilities for online schema transformations. First, PostgreSQL does not provide non-blocking DDL, but it is interesting as it can perform many DDL operations instantaneously. Next, MySQL has recently added support for online DDL[1]. In addition, a number of tools have been developed in industry to perform online schema changes on MySQL using Ronström's method, including pt-online-schema-change[2], oak-online-alter-table[3], and the online-schema-change tool developed at Facebook[4]. As these tools have similar capabilities, we investigate pt-online-schema-change in our experiments as a representative. Finally, Oracle 11g does not provide online DDL, but it can perform non-blocking schema changes using the *DBMS_REDEFINITION* package[5].

Approach For our experiment we have developed a benchmark [8] that extends the standard TPC-C benchmark[6] with basic and complex schema transformations. We run the standard TPC-C workload, while concurrently executing a schema transformation, and measure the impact on the TPC-C throughput. An important aspect of our benchmark is that schema transformations should be correct, i.e., they should satisfy the ACID properties, they should be composable to allow the execution of complex transformations, and ideally, transformations should be specified declaratively. We briefly discuss our requirements and the benchmark in Section 2, and we discuss our experimental setup in Section 3.

Results In Section 4 we discuss our experimental results for online transformations using the DDL provided by the DBMSs. We see mixed results for basic transformations, while all complex transformations block the TPC-C workload. In Section 5 we discuss the experimental results for Ronström's approach using pt-online-schema-change for MySQL and Oracle's *DBMS_REDEFINITION* package. We see that pt-online-schema-change can perform all basic DDL operations without blocking, but it can not perform complex transformations. Oracle can perform some complex transformations, but is limited to operations on a single table. We summarize our results in Section 6, and in Section 7 we discuss a solution outline to support complex non-blocking schema transformations.

Contributions The contributions of this paper are:

- An experimental investigation of the blocking behaviour of basic and complex schema transformations using the DDL in PostgreSQL, MySQL and Oracle 11g, and using Ronström's method as implemented by pt-online-schema-change and Oracle's *DBMS_REDEFINITION*.
- A solution outline for complex non-blocking schema transformations.

---

[1] `http://dev.mysql.com/doc/refman/5.6/en/innodb-online-ddl.html`
[2] `http://www.percona.com/doc/percona-toolkit/2.1/pt-online-schema-change.html`
[3] `http://openarkkit.googlecode.com/svn/trunk/openarkkit/doc/html/oak-online-alter-table.html`
[4] `https://www.facebook.com/notes/mysql-at-facebook/online-schema-change-for-mysql/430801045932`
[5] `http://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_redefi.htm`
[6] `http://www.tpc.org/tpcc/spec/tpcc_current.pdf`

2 Benchmark

In an earlier paper we have defined requirements for non-blocking schema transformations, based on which we have extended the standard TPC-C benchmark to measure the impact of various types of schema transformations on the TPC-C workload. In this section we briefly discuss the requirements and the benchmark. More details can be found in our earlier paper [8].

Requirements We have defined requirements on the functionality of schema transformations, and on their performance characteristics.

In terms of functionality, we assert that a schema transformation should satisfy the ACID properties like any other transaction that updates the database. Moreover, ideally, schema transformations should be specified declaratively. Similar to queries, a user should not have to be concerned with *how* a transformation is executed, but only *what* the result of a transformation should be. For instance, an implementation of the DDL satisfies this requirement if it provides ACID guarantees for transactionally composed DDL operations. Moreover, the system should provide a mechanism to update applications as part of the schema transformation, e.g., by replacing stored procedures transactionally.

In terms of performance, a schema transformation should have minimal impact on the performance of concurrent transactions. In particular, regular transactions should not be blocked, should not experience excessive slowdown, and should be able to complete without aborting. Moreover, the schema transformation itself should be able to commit while concurrent transactions are running, and the time to commit from the start of the transformation should be minimal. In our benchmark we measure the impact of schema transformations on the OLTP throughput, and the time-to-commit of the transformation.

Transformations Our benchmark contains basic transformations as provided by the SQL data definition language. Additionally, we also investigate bulk data updates without changing the schema, which is required in many complex transformations. Furthermore, our benchmark also contains a number of complex transformations, which generally consist of multiple DDL statements. In particular, we look at creating a column derived from another column, changing the cardinality of a relationship, and changing a primary key. Most transformations involve the largest table in the TPC-C schema, and update the stored procedures to allow the TPC-C workload to keep running on the transformed schema. A detailed description of the benchmark cases can be found in our earlier paper [8].

Benchmark Process The execution of a benchmark case is done in four phases. First, during the *setup* phase, we create a TPC-C database. Some benchmark cases require a modification to the TPC-C schema, which we also perform in this phase. Next, during the *intro* phase, we start the TPC-C benchmark load. We wait for 10 minutes before starting the transformation, while measuring the baseline TPC-C performance. Next, we start the *transformation* phase, where we execute the benchmark transformation. We wait for it to complete, while logging

the begin and end time of the transformation. Finally, we wait for another 10 minutes while measuring the TPC-C throughput in the *outro* phase.

Benchmark Results As seen in Figure 1, we present the result of a benchmark as a line graph that plots the TPC-C transaction execution rate over time. We mark the start and commit time of the transformation with vertical lines, and we show the time-to-commit under the x-axis. Moreover, we plot aborted and failed transactions in red. The y-axis starts at zero transactions per second, which corresponds to blocking behaviour. We do not show the absolute TPC-C throughput as we are only interested in blocking behaviour and the relative performance of TPC-C during and after a schema transformation compared to the intro phase.

3 Experimental Setup

An implementation of our benchmark, and all experimental results can be found on our website[7]. We use the TPC-C implementation HammerDB[8] to create the TPC-C database and to provide stored procedures. We use HammerDB to generate one database for each DBMS, which we backup once, and then restore in the setup phase of every experiment. Before starting the introduction phase of the experiment, we let the TPC-C benchmark run for ten seconds, as to give the DBMS some time to warm up. To generate load on the system, and to measure the TPC-C performance, HammerDB provides a driver script. However, as this script does not perform logging of transactions, we have ported the script to Java and we have added logging facilities. For all experiments, we generate a database of 30 warehouses, and we use 64 threads of load on the database. We do not spawn new threads to start other transactions while a thread is blocked. For the experiments we have used a quad-core Intel i7 machine with 16GB of RAM and a solid-state drive. For the software we used Ubuntu Linux kernel 3.20.0, PostgreSQL version 9.1.14, MySQL version 5.6.20, pt-online-schema-change version 2.2.11, oracle 11g release 11.2.0.3.0, and HammerDB version 2.14.

Stored Procedures Many of our benchmark cases update the TPC-C stored procedures so that the workload can keep running after the transformation. As such, we need support from the DBMS to change stored procedures as part of a schema transformation. PostgreSQL provides transactional DDL which also supports transactional upgrades of stored procedures. In contrast, MySQL does not have transactional DDL, and does not provide a mechanism to upgrade stored procedures safely. This means that stored procedure upgrades in our MySQL experiments are not atomic. Oracle provides *editions*, which allow switching between different versions of stored procedures safely. However, we found it difficult to automate our tests using editions, and chose to use non-atomic updates of stored procedures. This does not affect the results of our experiments.

[7] `http://wwwhome.ewi.utwente.nl/~weversl2/?page=ost`

[8] `http://hammerora.sourceforge.net/`

4 Experimental Results: Data Definition Language

This section shows our experimental results for online schema transformations using the data definition language in PostgreSQL, MySQL and Oracle 11g. First, we look at basic operations, including column operations, index operations and bulk data updates. To conclude, we investigate composition of DDL statements to perform complex transformations.

4.1 Basic Transformations

Adding and Removing Columns Figure 1 shows the impact of basic column operations on the TPC-C workload. Both PostgreSQL and Oracle can add a column instantaneously, without noticeably interrupting the TPC-C workload. MySQL can not add a column instantaneously, but uses its online schema change functionality. Despite this, MySQL still shows a short period of blocking at the start of the operation, and we see a significant reduction in throughput. When adding a column with a default value, PostgreSQL and Oracle now materialize the column being created, which results in a period of blocking. For MySQL we see the same behaviour as the previous case. When removing a column, PostgreSQL can perform this operation instantaneously, and MySQL can use its online schema change feature. Interestingly, DROP COLUMN causes Oracle to block. Oracle also allows a column to be marked as unused, which effectively removes the column without reclaiming disk space. Disk space can be reclaimed using DROP UNUSED COLUMNS, however, this is still a blocking operation.

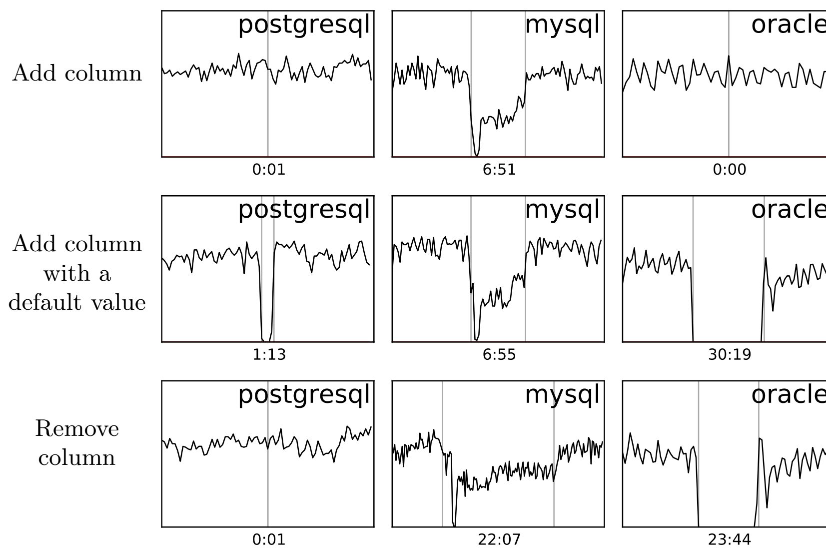


Fig. 1: Adding and removing columns.

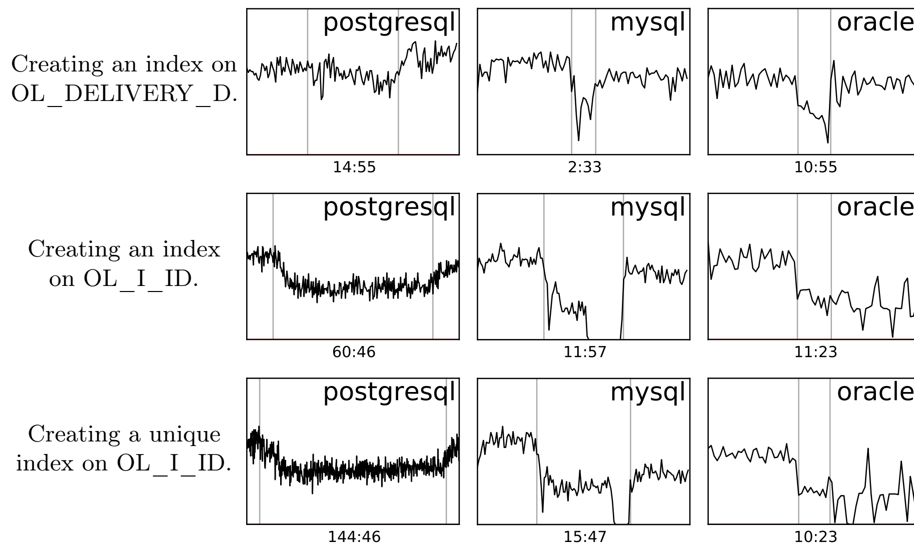


Fig. 2: Creating normal and unique indexes.

Creating Indexes Figure 2 shows the impact of creating indexes on the TPC-C workload. We have created indexes on two columns with different workload: the OL_DELIVERY_D column which is nullable and is not written on insertion, while the OL_I_ID is being written to on insertion. All tested DBMSs allow online creation of indexes. PostgreSQL shows a small impact on TPC-C throughput, but behaves well. Oracle commits more quickly than PostgreSQL, but shows periods of significant blocking after the commit, suggesting that Oracle is creating the index in the background. We have run the experiment for three hours after the commit, and have seen that this behaviour persists during this period. Despite supporting online index creation, MySQL blocks for a significant amount of time on when indexing the OL_I_ID column. We see that creating a unique index has similar characteristics to creating a regular index, but the time to commit for PostgreSQL and MySQL is longer. Removing indexes is an instantaneous operation in all three DBMSs, so we don't show their results.

Bulk Data Transformations For some transformations it is essential that we can update data in bulk. An update statement differs from an ALTER TABLE statement in that the schema is not changed. However, semantically it is a schema transformation. Updating prices in a database to use a different currency is an example of such a transformation. Moreover, bulk data operations are important in many complex transformations to transform data or to move data between tables. Where stored procedures may simply fail on a schema that it does not expect, for bulk data updates this is not the case. As such, it is important that bulk data transformations satisfy the ACID properties.

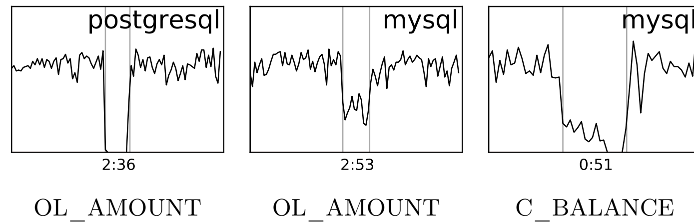


Fig. 3: Bulk data transformations in PostgreSQL and MySQL.

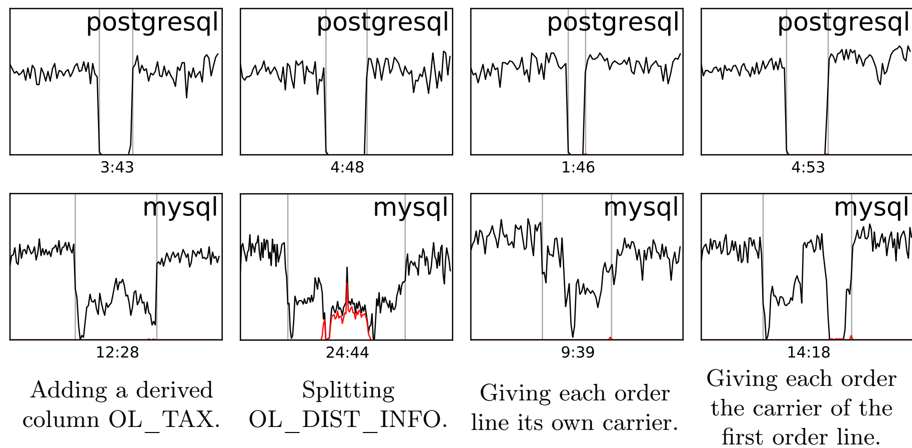
Figure 3 shows the impact of a bulk data update on the column in the caption using PostgreSQL and MySQL. We do not show results for Oracle, because it could not execute the bulk update due to concurrency conflicts. In both cases, we use the serializable transaction level to guarantee correctness. We see that PostgreSQL takes a table lock to guarantee serializability, and blocks the TPC-C workload. Interestingly, MySQL does not block the workload when updating the *OL_AMOUNT* column, because it only locks the OL_AMOUNT column, which is not being updated by the TPC-C workload. We ran the experiment on the *C_BALANCE* column, which is being updated, and see that MySQL now blocks toward the end of the operation. During the transformation, transactions can still execute, as MySQL doesn't take a complete table lock.

4.2 Complex Transformations

Transactional Composition A natural way to construct a complex transformation from DDL operations is to wrap them into a transaction. If every DDL operation is non-blocking, commits instantaneously, and does not block other transformations from starting after committing, then the composed transformation can also be non-blocking and instantaneous. However, all complex transformations that we have considered involve bulk data updates, which, as we have seen in the previous section, is blocking in current DBMSs. When composing an instantaneous transformation with a bulk data update, the instantaneous operation can take a table lock, which is held during the bulk data update.

We see this behaviour in PostgreSQL, as shown in Figure 4 (top row). In the leftmost experiment we have added a column OL_TAX whose value is derived from an existing column. First, we add the new column, which is non-blocking and instantaneous, and then we fill the column using UPDATE, which results in a table lock. We see the same behaviour in all complex cases that we have tested.

Non-transactional Composition MySQL and Oracle 11g do not provide transactional DDL: they auto-commit after each DDL operation. However, MySQL does support online DDL. Can we use this to perform complex transformations correctly? As many operations require bulk data operations that can not be performed without blocking, this is not possible in general.



Adding a derived column OL_TAX. | Splitting OL_DIST_INFO. | Giving each order line its own carrier. | Giving each order the carrier of the first order line.

Fig. 4: Complex transformations in PostgreSQL and MySQL.

When composing non-blocking transformations non-transactionally, intermediate schemas are visible to concurrent transactions. If we keep using the original stored procedures on these intermediate states, they can fail to execute, perform erroneous operations, or encounter lost updates, which could damage the integrity of the database. We could update the stored procedures directly after the commit of each transformation step to handle intermediate states. However, this does not work for bulk data transformation, as the original stored procedures will keep executing while the bulk data is in progress, which results in concurrency conflicts. For instance, if we want to add a derived column, we can first create a new column, and then fill it using a bulk update statement. However, while the update statement is in progress, the original transactions can continue executing on the source column, which updates are not reflected in the derived column, thus resulting in lost updates. To solve this, we could attempt to update the stored procedures *before* the transformation starts, but this does not solve the problem, as the new transactions can be blocked from writing to the derived column while the bulk update is in progress.

Figure 4 (bottom row) shows results when performing complex transformations using the online DDL in MySQL, where we only update the stored procedures after the transformation. While the transformations are mostly non-blocking, their results are incorrect in all cases because the TPC-C transactions keep executing on intermediate transformation states, which results in lost updates. In the second transformation, we also see many erroneous transactions because we do not update the stored procedures after every transformation step.

4.3 Conclusions

Our experiments with MySQL, PostgreSQL and Oracle show that support for non-blocking transformations using the DDL is rather weak. Most problematic

are adding columns with default values and performing bulk data updates. As complex transformations regularly require bulk data updates, non-blocking complex transformations are currently not possible at all. If non-blocking bulk updates where possible, many complex transformations could in principle be performed by adapting the stored procedures to intermediate states. However, this would also be very costly to implement in terms of development effort.

5 Experimental Results: Ronström's Method

Ronström proposed a method that allows changing of columns, adding indexes, and horizontally and vertical splitting and merging of tables by using database triggers [6]. The method works as follows. First, an *interim* table that matches the desired schema is created. Next, triggers are created on the original table that propagate any changes on the original table to the interim table. Next, data is copied to the new tables in small batches, while performing the desired schema transformation on the data. Finally, after copying the data, the original table is replaced by the interim table. A benefit of Ronström's method is that it can be implemented on top of existing database systems that support triggers.

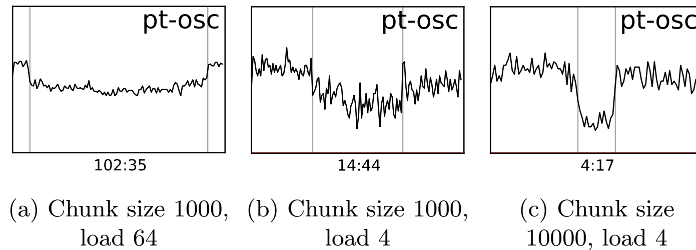
In this section, we investigate pt-online-schema-change, which implements Ronström's method for MySQL, and we look at the *DBMS_ REDEFINITION* package provided by Oracle 11g. While tools similar to pt-online-schema-change could be implemented for PostgreSQL, to our knowledge, at the time of writing no such tools are available.

5.1 pt-online-schema-change for MySQL

The pt-online-schema-change tool from the Percona Toolkit implements Ronström's method for MySQL. The tool accepts a single ALTER TABLE statement, which it executes transactionally. Multiple transformations can be performed using a single ALTER TABLE statement, but multi-table transformations and data transformations are not supported. It creates a new table with the new schema, and copies the rows from the source table to this new table. Copying is done in chunks of a certain size, which can be configured using two strategies. First, a fixed chunk size can be specified, and second, a fixed time per chunk can be specified. While a chunk is being copied, the copied rows are locked for writing. A larger chunk size impacts concurrent transactions more due to locking, while a shorter chunk size slows down the schema transformation.

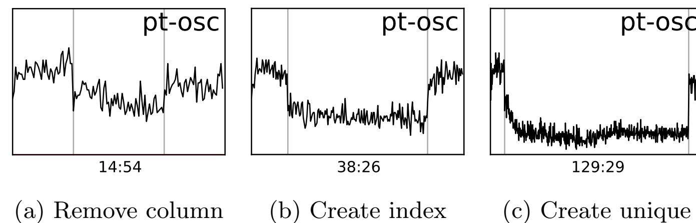
Effect of Chunk Size and Load The chunk size and the TPC-C benchmark load have a large effect on the performance of pt-online-schema-change. This is because pt-online-schema-change executes transactions in *LOW PRIORITY* mode, to minimize slowdown for concurrent transactions. Figure 5a shows the behaviour of MySQL when pt-online-schema-change is used to add a column with a TPC-C load of 64 threads and chunk size 1,000. The time to commit is very long, about 102 minutes, much longer than the 6:51 used by MySQL to perform the same

operation. If we lower the TPC-C load from 64 threads to only 4 threads, and keep the chunk size at 1,000, pt-online-schema-change commits in only 14:44, as shown in Figure 5b. If we increase the chunk size to 10,000, pt-online-schema-change completes in 4:17, as shown in Figure 5c, however, we also see a reduction in TPC-C performance.



(a) Chunk size 1000, load 64 (b) Chunk size 1000, load 4 (c) Chunk size 10000, load 4

Fig. 5: Adding a column using pt-online-schema-change.



(a) Remove column (b) Create index (c) Create unique

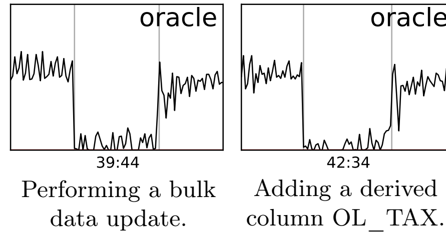
Fig. 6: Experimental results for pt-online-schema-change.

Results Figure 6 shows experimental results for pt-online-schema-change on several basic transformations. All basic DDL operations could be performed using pt-online-schema-change, and impact on performance is generally acceptable. Interestingly, pt-online-schema-change does not suffer from the initial period of blocking that we have seen in experiments using MySQL's online DDL. However, pt-online-schema-change does not support bulk data operations, and can not be used to perform transformations consisting of multiple DDL statements.

5.2 DBMS_REDEFINITION for Oracle

Since version 9i, Oracle provides the *DBMS_REDEFINITION* package, which allows schema transformations to be performed using Ronström's method. To use *DBMS_REDEFINITION*, the following steps have to be followed. First, an interim table has to be created with the desired schema. Next, the transformation is started by defining a mapping from fields in the original table to fields in the interim table, and by specifying a key that must be present in both the original and interim table, which is used to propagate updates on the original table to the interim table. Next, after the transformation is complete, objects such

as indexes, constraints and stored procedures can be added to the table. The package provides a method to copy all existing objects from the original table to the interim table. Finally, the transformation can be finished to replace the original table with the interim table. This is a blocking operation, and takes longer if the interim table is not synchronized with the original table.



Performing a bulk data update. Adding a derived column OL_TAX.

Fig. 7: Experimental results for Oracle's DBMS_REDEFINITION.

Figure 7 shows experimental results where we use *DBMS_REDEFINITION* to perform a bulk data update, and to add a column whose value is derived from another column. In general, we see that transactions can continue executing during the transformation, but performance is poor, and there are periods during the transformation where the throughput drops to zero. Despite this, *DBMS_REDEFINITION* allows any single-table transformation to be performed. Both of these cases can not be handled by pt-online-schema-change. However, transformations that involve multiple tables can not be performed, as the source of the transformation must be a single table. Compared to using the data definition language, this approach is more verbose, as the interim table must be defined, and all objects on the table must be copied.

6 Analysis Results

Our experiments with basic DDL operations in PostgreSQL, MySQL and Oracle 11g show mixed results. PostgreSQL can add and remove columns instantaneously and it can create indexes online, but blocks when adding a column with a default value, and when performing bulk updates. MySQL provides online DDL for adding and removing columns, but blocks for a significant period of time at the start of the transformation. MySQL also supports online creation of indexes, but our experiments show long periods of blocking at the end of the transformation. Similar to PostgreSQL, Oracle 11g can add columns instantaneously, however, adding columns with default values and removing columns takes very long, and blocks concurrent transactions. Bulk data updates are a problem in all tested DBMSs. PostgreSQL and MySQL simply block, while Oracle 11g can not execute the operation due to concurrency conflicts.

Using the DDL for complex non-blocking transformations is not possible in any of the DBMSs. Using transactional DDL, PostgreSQL can generally perform all operations correctly, but blocks access to all affected tables during the

transformation. MySQL and Oracle do not support transactional DDL. Composing non-blocking DDL operations non-transactionally is possible in general by updating stored procedures after each transformation step, however, MySQL and Oracle can not perform non-blocking data updates, which prevents us from performing most complex transformations. Moreover, such an approach is non-declarative, and can be costly to implement.

As an alternative to the DDL provided by the DBMSs, we have investigated Ronström's method. This method is interesting, as it can perform non-blocking schema transformations based on blocking transformations in any DBMS that implements triggers. The pt-online-schema-change tool shows that Ronström's method is a promising approach for basic online transformations: it can perform all basic schema transformations without blocking, with the exception of bulk data updates. However, complex transformation cases can not be handled by pt-online-schema-change, as it only supports a single ALTER TABLE statement at a time, and there is no support for UPDATE statements. Oracle's *DBMS_REDEFINITION* shows that Ronström's method can also be used for more complex single-table operations, but its implementation shows a significant amount of blocking.

7 Solution Outline

Native Support With the existence of Ronström's method, it could be argued that DBMSs do not need to provide native support for online schema transformations, but only have to provide support for triggers and atomic updates of schema meta-data. This is the approach that Oracle has taken with edition-based redefinition[9] and the *DBMS_PARAL-LEL_EXECUTE* package[10]. Edition-based redefinition allows atomic updates of schema meta data and provides cross-edition triggers that can transform data between versions of the schema. The *DBMS_PARALLEL_EXECUTE* package can be used to avoid full table locks while transforming data between versions.

A drawback of Ronström's approach is that transformations can take a long time to execute. Native implementations of Ronström's method in DBMSs can potentially be more efficient than external tooling. For instance, Løland and Hvasshovd present Log Redo as an alternative implementation for Ronström's approach that avoids the use of triggers, and has minimal impact on performance [3]. However, while more efficient implementations of Ronström's method could reduce execution time, this does not scale to very large databases.

On-the-fly Transformations An interesting alternative to Ronström's method is to perform transformations lazily, or on-the-fly. The basic idea is to commit a transformation before transforming the data, and transform the data before it is accessed. From the viewpoint of the user, this allows a transformation to

[9] http://docs.oracle.com/cd/E11882_01/appdev.112/e41502/adfns_editions.htm
[10] http://docs.oracle.com/cd/E11882_01/appdev.112/e40758/d_parallel_ex.htm

be executed instantaneously. Moreover, data can be transformed in the background during idle time. Lazy transformations have already been investigated in the context of Object-Oriented database systems [1]. Additionally, Neamtiu has shown that many relational schema changes can be performed on-the-fly [4].

Depending on the implementation, on-the-fly transformations have two advantages over Ronström's method. First, they compose naturally: two on-the-fly operations executed in sequence form an on-the-fly operation. Second, on-the-fly transformations can be implemented to execute in-place or incrementally: reusing storage space or garbage collecting parts of the original table that are already transformed. This avoids the problem of additional memory consumption for intermediate tables as seen in Ronström's method.

A drawback of on-the-fly transformations is that they must be implemented in the DBMS. Moreover, they provide overhead on data access, which increases latency. Most importantly, instantaneous transformations and on-the-fly transformations are limited to operations that can produce results on-the-fly. For instance, it is not possible to create an index or check a constraint instantaneously, and lookups in very large tables that do not have an index can not be done instantaneously. Consequently, composing a blocking transformations with an on-the-fly transformation leads to a blocking transformation. This shows that on-the-fly transformation by themselves are not a full solution to the problem.

Complex Operations using Ronström's Method In his original paper, Ronström has proposed the use of SAGAs to compose basic transformations into more complex transformations [6]. The idea of SAGAs is to execute the individual operations of a transaction as a sequence of transactions, where for each operation an undo operation is provided that can be used to rollback the complete sequence of operations [2]. While SAGAs provide failure atomicity for composed operations, they expose intermediate states of the transformation to concurrent transactions. This requires applications that use the database to handle these states, which is non-declarative and requires additional development effort.

However, we think that almost any relational transformation can be performed atomically using Ronström's method without the use of SAGAs. The following is a sketch of the solution. First, to compose transformations, we can chain interim tables, i.e., triggers on the original table propagate updates to the first interim table, while triggers on the first interim table propagate updates to a second interim table, and so on. Using multiple interim tables can require a lot of memory. However, sequential transformations could potentially be combined to use a single interim table. Second, we can define triggers on multiple tables to propagate updates to one or more tables. This allows multi-table transformations. Finally, update propagation is inefficient for operations that require lookups on tables that are not indexed. This can be solved by dividing a transformation into two steps, where indexes are constructed in the first step, and where the transformation is performed in the second step using these indexes.

From a practical viewpoint, manually implementing transformations using this approach is quite complex, and optimizing such transformations even more so. One has to reason about updates on all involved tables, and how these should

be propagated to interim tables. Data could be lost if certain triggers are missing or wrongly implemented. To solve this, tooling could be developed to transform declarative transformation specification into optimized execution plans.

Solution Outline Rönstrom's method is essentially an optimistic concurrency control method: it performs operations on a snapshot of the state, and repairs any conflicts that arise from concurrent operations. As such, Ronström's method never blocks access to the state, but it requires additional memory to maintain multiple versions of the state. Moreover, it can only commit after the transformation has been completely executed. On the other hand, an on-the-fly method is essentially a pessimistic concurrency control method: it avoids conflicts by transforming data before access, i.e., it blocks access to parts of the database until the transformation for that part has been executed. However, on-the-fly methods can commit immediately, and require less memory compared to Ronström's method as they can perform transformations in-place or incrementally.

A solution to minimize time to commit would combine both approaches by first using Ronström's method to check constraints and prepare indexes, and then performing the remainder of the transformation using on-the-fly methods. However, if time to commit is not crucial, Ronström's method could be preferable in situations where predictable low-latency access to data is crucial.

Similar to declarative query support, we envision that DBMSs allow us to perform arbitrary schema transformations declaratively. As such, a DBMS should provide a schema transformation optimizer that can construct a non-blocking execution plan from a declarative specification of a schema transformation with the goal of minimizing throughput reduction, access latency, memory consumption and time to commit.

References

1. Ferrandina, F., Meyer, T., Zicari, R.: Implementing Lazy Database Updates for an Object Database System. In: VLDB '94. pp. 261–272 (1994)
2. Garcia-Molina, H., Salem, K.: Sagas. In: SIGMOD '87. pp. 249–259. ACM (1987)
3. Løland, J., Hvasshovd, S.O.: Online, Non-blocking Relational Schema Changes. In: EDBT '06. pp. 405–422. Springer-Verlag, Berlin, Heidelberg (2006)
4. Neamtiu, I., Bardin, J., Uddin, M.R., Lin, D.Y., Bhattacharya, P.: Improving Cloud Availability with On-the-fly Schema Updates. In: COMAD '13. pp. 24–34. Computer Society of India (2013)
5. Neamtiu, I., Dumitras, T.: Cloud software upgrades: Challenges and opportunities. In: MESOCA '11. pp. 1–10. IEEE (2011)
6. Ronström, M.: On-Line Schema Update for a Telecom Database. In: ICDE '00. pp. 329–338. IEEE (2000)
7. Sockut, G.H., Iyer, B.R.: Online reorganization of databases. ACM Comput. Surv. 41(3), 14:1–14:136 (Jul 2009)
8. Wevers, L., Hofstra, M., Tammens, M., Huisman, M., van Keulen, M.: A Benchmark for Online Non-Blocking Schema Transformations. In: DATA '15 (2015)