STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LX**, Number 1, 2015

EXTENDING OBJECT-RELATIONAL MAPPING WITH CHANGE DATA CAPTURE

RADOSLAV RADEV

Abstract. The most critical disadvantage of the Change Data Capture functionality provided by the relational database management systems is the impossibility to add custom, application-specific data to the log. In this paper an approach is proposed how to implement custom, application-specific Change Data Capture functionality with the help of database triggers and O/RM framework. There are listed clear problem requirements and concrete steps are proposed how to resolve it. The two layers of the solution are identified and described database level and application level. All details of the solution are provided with example source code. It is also suggested an approach for quality assurance of the solution.

1. Introduction

One of the open issues of object-oriented programming in the last two decades is the object-relational impedance mismatch [13], [7]. It concerns the usage of relational database management systems (RDBMS) with object-oriented programming languages. In 2002 Fowler summarized the concept of object-relational mapping (O/RM) [3] and in the last years several O/RM frameworks were developed, thus bringing object-relational technology to the mainstream of application programming [10]. Among them most known and used are Hibernate (for Java) [18] and Microsoft Entity Framework (for .NET) [11]. They are based on the entity data model conception and definition of mappings between object-oriented classes and relational database tables [6, 18].

Beside the latest development of object-oriented and other non-relational database systems, the relational databases are still widely used for enterprise and data-driven applications. This is due mostly to their simplicity, proven

Received by the editors: October 23, 2014.

2010 *Mathematics Subject Classification.* 68P05, 68P20.

1998 *CR Categories and Descriptors.* D.2.12 [**Software Engineering**]: Interoperability - *Data mapping*; H.2.4 [**Database Management**]: Systems - *Relational databases*.

Key words and phrases. relational database, change data capture, log-trigger, object-relational mapping.

reliability for almost 40 years of development and support by big software vendors like Oracle, Microsoft and IBM [3].

In 2002 Oracle introduced a new relational database functionality called Change Data Capture [20]. It identifies and captures data that has been added to, updated or removed from, relational tables, and makes the change data available for use by applications. This allows all changes made to the database records to be tracked, persisted and retrieved later. This is very useful for building statistics, reviewing logs or revert the changes made. In the last case the records affected could be restored to their previous state, because the Change Data Capture feature keeps the changes made to the records along with their previous versions.

In 2007 Popfinger gives a more precise definition of Change Data Capture architecture [22]. This architecture uses rules and triggers to identify data that has been changed since the last extraction. Triggers copy updated data to a specially created change table (log-table) where they can be accessed by subscribers using individual views to access the information it is interested in and allowed to read.

In 2008 Microsoft also implemented Change Data Capture in Microsoft SQL Server 2008 [15]. Change Data Capture functionality is very useful for tracking and keeping the changes made to database records. The whole process is actually very easy to use, because all the tracking and persistence of the changes is made by the database server itself (of course we assume it supports this functionality like in the examples above). But the following problem arises: we cannot associate additional, custom data with the changes made. Let us give one very common example. Imagine an enterprise or data-driven application [3], [12] that uses a relational database as a persistent storage. It allows different users to log in to the system and manipulate the data via user interface. The users can work simultaneously and perform different operations that result in changes of the database records. But with the default Change Data Capture functionality provided by RDBMSs we cannot distinguish which change is made by which user.

Why this is important? Because if we find a way to attach a user identifier to the Change Data Capture logs, we not only will be able to distinguish which change is made by which user, but also to revert the changes of a particular user without affecting the other users changes.

In this paper we propose an approach how to implement a custom Change Data Capture functionality that supports logging not only of the database record changes, but also of additional, custom, application-specific data for every record. To solve the upper problem we will use O/RM framework. We will centralize all the access to the relational database through the O/RM framework and will ensure that the additional data is saved correctly to the

Change Data Capture logs. In advance, our approach is not tied to a particular O/RM framework or a RDBMS, but suggests a general solution that could be implemented for different O/RM frameworks and RDBMSs according to their specific features. Finally, we will illustrate one practical application of our approach as a proof of concept - how to use this extended Change Data Capture functionality to revert a set of changes made to the database by a particular user in an enterprise application. In this way, we are trying to solve some current problems in database usage.

It has to note, that Change Data Capture technique is extensively used in other areas of computer science and is of particular importance for data warehouse maintenance (see [20], [8]), which is beyond of the scope of this study.

2. Requirements

Here we will specify in more details the problem and the requirement for its solution. We seek to define a general approach how to:

- Implement our own Change Data Capture functionality because the default one provided by RDBMSs (Oracle, Microsoft, etc.) cannot be easily extended to support custom data added to the log. That means we have to define a mechanism how to record all changes made to the database records and how to persist these changes.
- Along with the database records changes we want to persist also additional, application-specific data to the log for every changed record.
- The approach should not be tied to any particular O/RM framework or a RDBMS. It must suggest a general solution that could be implemented for different O/M frameworks and RDBMSs.
- Ensure a possibility for quality assurance of our solution. That means a way to ensure all preconditions in the database (tables, triggers, etc.) are satisfied and so our solution is guaranteed to work correctly.
- As prove of concept, revert a set of changes by a given criteria without affecting all other changes. For example, reverting only changes made in an enterprise application by a particular user, but keep all other changes made meanwhile by other users.

Our approach consists of concepts applied to two levels database level (RDBMS) and application level (O/RM). We will discuss the database level in Section 3 and the application level in Section 4.

3. Change Data Capture - Database Level

Let us call every table we want to track changes of data-table. We will create a common log-table to store the logs needed for Change Data Capture

functionality. This means we will store all the logs in a single log-table, not in a separate log-table for every data-table. For every change made to a record in any data-table in the database we will insert a log-record in the log-table with information about the change. This information consists of:

- Unique identifier of the operation.
- Table identifier (in which data-table the changes is made).
- Date and time of the operation (when the change is made).
- Old record data (record values before the change). In case of insert operation, it will be empty (NULL).
- New record data (record values after the change). In case of delete operation, it will be empty (NULL).
- Type of the operation (insert/update/delete) optional, because it could be figured out from the previous two columns. If old record data is NULL, it is insert operation. If new record data is NULL, it is delete operation. If both are NOT NULL, it is update operation.
- Transaction identifier (needed by the O/RM framework to add custom additional data later).
- Additional data it could be a single column, but if more complex additional data is to be persisted, it could be a single column in XML format or multiple different columns.

The old and new record data will be serialized in a common format. It is most convenient to use XML because some RDBMSs support serializing a record to XML, for example Oracle [21] and Microsoft [16]. If that is not the case with a particular RDBMS, another specific solution should be applied according to the functionalities provided by this specific RDBMS for record serialization.

Here is the DDL code for creating the log-table in Transact-SQL [4] for Microsoft SQL Server:

```
CREATE TABLE [dbo].[Log_Records]
(
    [ID] [uniqueidentifier] NOT NULL PRIMARY KEY DEFAULT
NEWSEQUENTIALID(), – Unique identifier of the operation automatically generated by RDMBS.
    [Old_Data] [xml] NULL, – Old recor values serialized in XML format.
    [New_Data] [xml] NULL, – New record values serialized in XML format.
    [Table_Name] [nvarchar](50) NOT NULL, – Data-table name of the changed record.
    [Operation] [varchar](10) NOT NULL, – Code of the operation. For clearance we will use
'INSERT', 'UPDATE' or 'DELETE' string values. It could equally well be an integer value, for
example 1 (insert), 2 (update), 3 (delete).
    [Transaction_ID] [bigint] NOT NULL, – Needed by O/RM framework to add additional data to
the log-record later.
    [User_ID] [uniqueidentifier] NULL, – Custom, additional, application-specific data added to the
Change Data Capture logs.
```

[Date_Time] [datetime] NOT NULL DEFAULT GETDATE() – *Date and time of the operation, by default the current datetime.*
)

To capture the changes made to the records we propose to use database triggers. They are routines executed before, instead, or after an event in a database [14]. In our case this event is an insert, update, or delete operation in a data-table. Triggers could be used for different purposes logging, validation, ensuring data integrity, or others. Another aim could be introduction of some business logic to the database [9]. In our case we use database triggers for the logging process of changes [1] and also for including some business-specific data to the log.

We will add log-trigger to every table for which we want to track the changes. It will be executed after every insert, update or delete operation. If the particular RDBMS allows it, the code of the trigger could be the same for all data tables, which makes it highly maintainable and testable. This is possible for Oracle and Microsoft SQL Server, because they support serializing a record to XML. Then we will have multiple triggers, one per data-table, but with the same DDL code, which makes them easy to maintain.

Here is our suggestion for a DDL code that creates a log-trigger for a data-table in Microsoft SQL Server. T-SQL, also known as Transact-SQL, is a proprietary extension to SQL, central to using Microsoft SQL Server [4]. All SQL and DDL examples in this paper will be given in T-SQL.

```
CREATE TRIGGER [dbo].[TR_Persons_Log]
    ON [dbo].[Persons] – in this case the data-table is dbo.Persons
    AFTER INSERT, UPDATE, DELETE
    AS
BEGIN
    – Serialize the old record values to a XML variable.
    DECLARE @old_data XML;
    SELECT @old_data = (SELECT * FROM deleted FOR XML PATH(''));
    /* Make notice that we do not use the column names of the data-table, which proves that
the log-trigger DDL code could be the same for all tables, because it does not depend on the table
metadata. */
    – Serialize the new record values to a XML variable.
    DECLARE @new_data XML;
    SELECT @new_data = (SELECT * FROM inserted FOR XML PATH(''));
    – Fill the operation column value as user-friendly text for clearance.
    DECLARE @operation VARCHAR(10);
    SELECT @operation = CASE
    WHEN (@old_data IS NULL) and (@new_data IS NOT NULL) THEN 'INSERT'
    WHEN (@old_data IS NOT NULL) and (@new_data IS NOT NULL) THEN 'UPDATE'
    WHEN (@old_data IS NOT NULL) and (@new_data IS NULL) THEN 'DELETE'
    END;
    – If both old and new data is null  nothing to do.
```

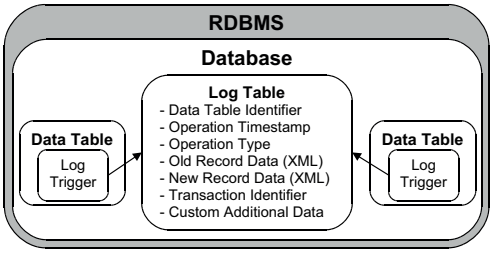

FIGURE 1. Scheme of Change Data Capture - Database level

```
    IF (@old_data IS NULL and @new_data IS NULL) RETURN;
    – Retrieve the current transaction identifier  this is needed by O/RM to add additional data later.
    DECLARE @transaction_id INT;
    SELECT @transaction_id = transaction_id
    FROM sys.dm_tran_current_transaction;
    – Get the current table name in format [schema].[table].
    DECLARE @table_name NVARCHAR(50);
    SELECT @table_name = s.name + '.' + t.name
    FROM sysobjects tr
    INNER JOIN sys.tables t ON tr.parent_obj = t.object_id
    INNER JOIN sys.schemas s ON s.schema_id = t.schema_id
    WHERE tr.id = @@procid;
    – Insert the new log-record in the log-table.
    INSERT INTO dbo.Log_Records (Old_Data, New_Data, Operation, Table_Name, Transaction_ID)
    VALUES (@old_data, @new_data, @operation, @table_name, @transaction_id);
END
```

All these operations proposed by us are implemented on the database server level, as shown on Figure 1. Make notice that so far we have implemented only the basic Change Data Capture functionality that tracks the changes made to the records. No additional data has been added to the log yet, only the storage for it has been prepared in the common log-table.

4. Change Data Capture with OR/M Framework - Application Level

On application level we will use O/RM framework to add custom additional data to the log. O/RM framework centralizes all database access from within the software application. It both separates and bridges the object-oriented classes or entities and the relational database.

O/RM frameworks frequently use Unit of Work design pattern, defined by Fowler [3]. This is the case for example with Hibernate [5] and Entity Framework [2]. Unit of Work design pattern ensures that all operations made to different entities are submitted to the database within a single transaction.

We will use this specific feature to encapsulate both logging the changes and adding the additional data to the log in this single transaction, as shown in Figure 2.

We propose the following sequence of operations made by O/RM framework to support Change Data Capture functionality and to add additional data to the log:

(1) Client/User manipulates the entities via O/RM framework.
(2) Client/User calls O/RM frameworks *SaveChanges* method and passes to it as parameters the additional data that is to be saved to the log.
(3) O/RM framework opens a connection to the database, starts a transaction and retrieves the transaction identifier from the database.
(4) O/RM framework generates SQL statements corresponding to the changes made to its entities and sends them to the database server.
(5) Database server executes these SQL statements consequently. Because every table has a log-trigger that is executed after every insert, update or delete operation, the log-trigger is fired for every inserted/updated/deleted record in any table we want to track changes of.
(6) On every execution, the log-trigger inserts a single record in the log-table (log-record) with the following data: timestamp, table identifier, type of the operation, old record data, new record data, transaction identifier.
(7) O/RM framework add the additional data to the log by generating and executing a SQL statement to update all log-records with the transaction identifier retrieved in step 3. This ensures that only the newly inserted log-records (in previous step 6) will be updated, so any other changes made simultaneously by other users/clients will not be affected.
(8) O/RM framework commits the transaction started in step 4 and closes the connection to the database.

Here is our suggestion for a commit method of the O/RM framework. It is given in C# for Entity Framework 4.0 [11].

```
/* The Object Context class  instance of the main O/RM framework class. In case of Entity Framework, this is the ObjectContext class, so we inherit it. */
public partial class DbRollbackContext : ObjectContext
{
    /* Pass the additional data that is to be saved to the log as parameters of the SaveChanges method  in our case, only the current users identifier, but there could be also other parameters. */
    public void SaveChanges(Guid userId)
    {
    this.Connection.Open(); // Open a connection to the database.
    try
```

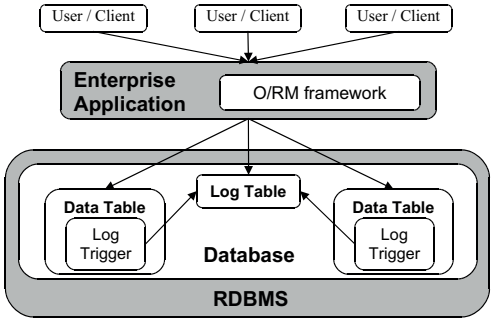

FIGURE 2. Scheme of Change Data Capture - Application level.

```
    {
     // Start a transaction to the database within the opened connection.
    using (DbTransaction transaction = this.Connection.BeginTransaction())
    {
    /* Retrieve the just started transaction identifier. This is the same value that will be saved by
the log-trigger in the Transaction_ID column of the log-table for all log-records that will be inserted
for the changes that are to be made in the next line of code. */
    long transationId = this.ExecuteStoreQuery<long>("SELECT
    transaction_id FROM sys.dm_tran_current_transaction").First();
    /* Call the base SaveChanges method of the ObjectContext class. It will generate and execute
SQL statements corresponding to the changes made to the entities tracked by the object context.
These SQL statements are insert/update/delete operations that will fire the log-trigger of the cor-
responding tables. The log trigger will insert log-records in the log-table with the current Transac-
tion_ID, which was retrieved in the previous line of code. */
    base.SaveChanges();
    /* Update all log-records with the same Transaction_ID, i.e. the just inserted ones and set
their additional data. */
    this.ExecuteStoreCommand(string.Format("UPDATE dbo.Log_Records SET User_ID = '0'
    WHERE User_ID IS NULL AND Transaction_ID = 1", userId, transationId));
    transaction.Commit(); // Commit the transaction.
    }
    }
    finally
    {
    this.Connection.Close(); // Close the connection to the database.
    }
    }

}
```

With this the implementation of our custom Change Data Capture functionality is complete. In the next Section we will discuss how to retrieve the inserted logs and how to use them to revert a set of operations.

5. An Example of Practical Application - Retrieve the Logs and Revert a Set of Changes

Once the log-records are in the log-table, it is easy to retrieve them with a simple SELECT SQL command. And because we have also the additional data stored in the log, we can filter the log-records accordingly. For example, we can retrieve the set of changes made by a user for a time period. Moreover, we can easily revert these changes. We have the order of the changes in the *Date_Time* column. For every log-record we can construct a corresponding revert SQL statement. Then we must execute these revert statements in the opposite order.

5.1. **Revert INSERT Operation.**

To revert an INSERT operation, we will generate a corresponding DELETE statement. In the WHERE clause of the DELETE statement we will include all old record values (before the change logged by the log-record we want to revert). This ensures that the newly generated DELETE statement will actually delete the record only if the current record values are the same as the original ones in the moment of its insertion. That is to say, if the record has been updated later by another user, its current record values will be different from the original ones. The WHERE clause of the DELETE statement will match no record, and accordingly no records will be deleted. So if another user has changed the record after that, our newly generated DELETE statement will not affect them actually it will not affect any records. What to do in this case is a decision that depends of the business logic of the enterprise application. It could regard such situation as an error or not; it could stop the whole process of reverting or to try to continue it. Different approaches are possible according to the concrete situations.

Here is an example code how to generate a DELETE statement that will revert an INSERT statement from a log-record.

```
private static string GenerateRevertInsertSqlStatement(LogRecord logRecord)
{
    /*Convert the old data XML to Dictionary with key the column name and value - the old
record value for this column (before the changed logged in this logged record). */
    IDictionary<string, string> newRecordValues =
    ConvertXmlToDictionary(logRecord.NewData);
    /*Delete the inserted record, but check if it has not been updated later. For the purpose include
the record values in the moment of insertion in the WHERE clause. This guarantees the record will
not be deleted if it has been updated later by another user.*/
    return string.Format("DELETE FROM {0} WHERE {1}", logRecord.TableName,
    string.Join(" AND ", newRecordValues.Select(v =>
    string.Format("{0} = '{1}'", v.Key, v.Value))));
}
```

5.2. **Revert UPDATE Operation.** To revert an UPDATE operation, we will generate another UPDATE SQL statement, taking the following guidelines into consideration:

- In the SET clause of the UPDATE statement, we will update only the changed columns values.
- If the records table contains computed column [5], they should not be included in the SET clause, because they will be computed by the RDBMS. Information about the computed columns is usually available in the O/RM framework meta-data.
- In the WHERE clause of the UPDATE statement we will include all old record values (before the change logged by the log-record we want to revert). This ensures that the newly generated UPDATE statement will actually update the record only if the current record values are the same as in the moment of inserting the log-record. That is to say, if the record has been updated again later by another user, its current record values will be different from the ones in the log-record. The WHERE clause of the UPDATE statement will match no record, and accordingly no records will be updated. So if another user has changed the record after that, our newly generated UPDATE statement will not affect them actually it will not affect any records. Again, what to do in this case is a decision that depends of the business logic of the enterprise application, as stated in Section 5.1.

Here is an example code how to generate an UPDATE statement that will revert an UPDATE statement from a log-record.

```
private static string GenerateRevertUpdateSqlStatement(LogRecord logRecord)
{
    /* Convert the old data XML to Dictionary with key the column name and value - the old
record value for this column (before the changed logged in this logged record). */
    IDictionary<string, string> oldRecordValues = ConvertXmlToDictionary(logRecord.OldData);
    /* Convert the new data XML to Dictionary with key the column name and value - the new
record value for this column (after the changed logged in this logged record). */
    IDictionary<string, string> newRecordValues = ConvertXmlToDictionary(logRecord.NewData);
    StringBuilder setStatement = new StringBuilder();
    StringBuilder whereStatement = new StringBuilder();
    /* If the table contains computed columns, they should be excluded from the SQL statement.
These special columns can be retrieved from O/RM mappings meta-data. */
    // Iterate through the new record values to build the UPDATE SQL statement.
    foreach (KeyValuePair<string, string> newRecordValue in newRecordValues)
    {
    string oldRecordValue = oldRecordValues[newRecordValue.Key];
     // If the new value is the same as the old one, do not include it in the SET clause.
    if (oldRecordValue ! = newRecordValue.Value)
    {
    if (setStatement.Length > 0) setStatement.Append(", ");
```

```
    setStatement.AppendFormat("0= '1'", newRecordValue.Key, oldRecordValue);
    }
    /* Include all old values in the WHERE clause to guarantee that the record will be updated
only if the current record values match the "new" ones (after the change). */
    if (whereStatement.Length > 0) whereStatement.Append(" AND ");
    whereStatement.AppendFormat("0 = '1'", newRecordValue.Key, newRecordValue.Value);
    }
    // No fields have been changed - nothing to revert.
    if (setStatement.Length == 0) return string.Empty;
    // Concatenate and return the whole SQL statement.
    return string.Format("UPDATE 0 SET 1 WHERE 2", logRecord.TableName, setStatement,
whereStatement);
}
```

5.3. **Revert DELETE Operation.** To revert a DELETE operation, we will generate INSERT SQL statement to re-insert the deleted record. In order to achieve this we have to take the following into consideration:

- If the records table contains computed column [24], they should not be included in the SET clause, because they will be computed by the RDBMS. Information about the computed columns is usually available in the O/RM framework meta-data.
- If the records table contains auto-generated columns [17], some RDBMSs do now allow inserting concrete values (in our case, the old ones in the moment of records deletion). An example of such a restriction is Microsoft SQL Server [17]. In this case the RDBMS must be explicitly instructed to allow insertion of auto-generated values with DDL statement (if it supports it). Because many inserts may be needed in the whole process of reverting records, it is best to execute these enable auto-generated values DDLs against the tables needed in the beginning of the whole revert process and undo them (i.e. restore the restriction of inserting auto-generated values) in the end of the whole revert process, after all SQL statements generated by the log-records have been executed. Example code will be shown in the next Section 5.4.

Here is an example code how to generate an INSERT statement that will revert a DELETE statement from a log-record.

```
private static string GenerateRevertDeleteSqlStatement(LogRecord logRecord)
{
    IDictionary<string, string> oldRecordValues = ConvertXmlToDictionary(logRecord.OldData);
    /* Re-insert the deleted record with its last values. If the table contains computed columns, they
should be excluded from the SQL statement. These special columns can be retrieved from O/RM
mappings meta-data. */
    return string.Format("INSERT INTO 0 (1) VALUES (2)", logRecord.TableName,
```

```
    string.Join(", ", oldDataValues.Select(v => v.Key)),
    string.Join(", ", oldDataValues.Select(v => string.Format("'0'", v.Value))));
}
```

5.4. **O/RM framework Revert Changes Method.** The O/RM framework Revert Changes method should execute the following operations:

(1) Retrieve a set of log-records filtered by a given criteria, for example for specific user and/or time interval.
(2) Identify the data-tables that need structural changes because of auto-generated columns, as explained in Section 5.3.
(3) Open a connection to the database.
(4) Generate and execute DDL statements for the tables identified in Step 2. to enable insertion of concrete values in auto-generated columns.
(5) Starts a transaction to the database all revert operations should be in a transaction.
(6) For every log-record generate and execute a revert SQL statement. If the revert SQL statement has not affected any records, handle the situation that the record we try to revert the changes of has been changed later, for example by another user. It is up to the enterprise application how to handle this situation. At least a few approaches are possible to abort the whole revert process; to ignore the situation and to continue with the next changes and to try to revert them; or to require a user interaction how to proceed
(7) Commit the transaction to the database.
(8) Generate and execute DDL statements for the tables identified in Step 2. to disable insertion of concrete values in auto-generated columns, i.e. to return the tables to their initial state.
(9) Close the connection to the database.

Here is an example revert changes method in C# for Entity Framework 4.0 [5].

```
public void RevertChanges(Guid userId, DateTime startDateTime, DateTime endDateTime)
{
    // Get the logs filtered by the given criteria.
    IEnumerable<LogRecord> logsToRevert = this.LogRecords
    .Where(r => (r.User_ID == userId) &&
    (r.Date_Time >= startDateTime) &&
    (r.Date_Time <= endDateTime))
    .OrderByDescending(r => r.Date_Time)
    .ToArray();
    /* Retrieve the tables with deleted records  because the revert operation is an INSERT operation,
we need to enable insert of identity/auto-generated values in this tables. */
    IEnumerable<string> tablesWithDeletedRecordsThatWillBeReinserted = logsToRevert
```

```
.OfType<DeleteLogRecord> ()
.Select(r => r.Table_Name)
.Distinct()
.ToArray();
this.Connection.Open(); // Open a connection to the database.
try
{
 /* Enable insert of identity/auto-generated values for each table with deleted records that will
be re-inserted. */
foreach (string tableName in tablesWithDeletedRecordsThatWillBeReinserted)
{ string cmdEnableIdentityInsertSql = string.Format("SET IDENTITY_INSERT 0 ON", table-
Name);
this.ExecuteStoreCommand(cmdEnableIdentityInsertSql); }
 // Start a transaction to the database within the opened connection.
using (DbTransaction transaction = this.Connection.BeginTransaction())
{
foreach (LogRecord logRecord in logsToRevert)
{ string revertSql = null;
 // Generate the revert SQL statement according to the operation.
switch (logRecord.Operation)
{
case "INSERT": revertSql = GenerateRevertInsertSqlStatement(logRecord); break;
case "UPDATE": revertSql = GenerateRevertUpdateSqlStatement(logRecord); break;
case "DELETE": revertSql = GenerateRevertDeleteSqlStatement(logRecord); break;
}
 // UPDATE statement that has made no changes - ignore it.
if (string.IsNullOrEmpty(revertSql)) continue;
try
{  // Execute the command against the database.
int recordsAffected = this.ExecuteStoreCommand(revertSql);
 // Check the result
if (recordsAffected == 0)
{
/* The revert statement has not affected any records, i.e. its WHERE clause is not matched
by any record in the database. This means that the record that is to be reverted has been changed
by another user after the change. Usually an exception should be thrown here, but a more complex
logic is possible according to the enterprise application specific needs. */
}
}
catch (Exception ex)
{
 /* Executing revert SQL statement has failed, due to constraints violation or some other
reason. More detailed information about the error can be retrieved from ex variable. */
}
}
transaction.Commit(); // Commit the transaction.
}
}
finally
{
 /* Disable insert of identity/auto-generated values for each table with deleted records that have
been be re-inserted, i.e. return the table to its initial state. */
foreach (string tableName in tablesWithDeletedRecordsThatWillBeReinserted)
```

```
    {
    string cmdDisableIdentityInsertSql = string.Format("SET IDENTITY_INSERT 0 OFF", table-
Name);
    this.ExecuteStoreCommand(cmdDisableIdentityInsertSql);
    }
    this.Connection.Close(); // Close the connection to the database.
    }
}
```

---

6. Quality Assurance

One of the fundamental concerns in software engineering is Quality Assurance (QA) [23]. Very common methodology for its realization is the unit tests [19]. Here we will describe how to implement unit tests to ensure the correct behavior of our solution. We propose to test a few things:

(1) All data-tables in the database must have log-triggers.
(2) The code for every log-trigger should be correct. If we need to change the log-trigger code in time, and because the log-trigger code is basically the same for all data-tables, we must be sure that all data-tables have the latest version of the log-trigger. So as correct we will regard the latest version of the log-trigger code.

There exists various options for implementation of these unit tests. We will give an example with implementation in the following way: We will store the correct (i.e. the latest) log-trigger code as an embedded file in the assembly that contains the O/RM object context. The O/RM framework will read this file. The code of the log trigger is the same for all tables, except the data-table name. So the O/RM framework will store a template log-trigger code and replace it with the given table name.

The unit test must execute the following actions to ensure all data-tables in the database have a correct log-trigger:

(1) Retrieve all data-tables from the RDBMSs metadata with the code of their log-trigger.
(2) Check if they have a log-trigger at all.
(3) Check if the code of the log-trigger is correct.

If there are other triggers in the database, it will be good to have a naming convention for them in order to distinguish the log-triggers from other triggers. An example for a naming convention is TR_TABLE_NAME_Log, but it could equally well be any other. Also, if we do not want to track changes for all data-tables, but only for specific ones, we should modify our unit test to retrieve from the database only the data-tables we want to track the changes of.

7. Conclusion and Further Work

In this paper an approach is proposed how to implement custom Change Data Capture functionality with the help of O/RM framework. The clear problem requirements are listed along with the proposed steps for our solution divided in two layers database level (RDBMS) and application level (O/RM framework). Fully-working and well-documented example source code is given for every part of the solution along with an approach for quality assurance. We proved empirically the correctness of our approach implementing it successfully in a real-world business application for *interAxio* project in Healthcare Research & Development department of the Belgium company Televic. The following technologies were used: Microsoft .NET Framework 4.0, C# 4.0, Entity Framework 4.0 [5], SQL Server 2008 R2.

As future perspectives for our approach we consider: (1) Examine and prove empirically its application for other RDBMSs and O/RM frameworks; (2) Measure and evaluate the performance impact of the whole system where it is integrated; (3) Explore how this logging could interfere with systems that are using triggers for other things (pre-insert/post-insert triggers as example), and where would the log trigger be inserted in the chain of called triggers; (4) Does this procedure of logging could infere with other transactions and how?

References

[1] A.A. Chuvakin, K.J. Schmidt, Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management, Syngress, Waltham, 2012.

[2] Developer Network, DbContext class documentation, Microsoft, http://msdn.microsoft.com/en-us/library/system.data.entity.dbcontext

[3] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Longman Publishing Co. Inc., Boston, 2002.

[4] K. Henderson, The Guru's Guide to Transact-SQL. Addison-Wesley Longman, Reading, MA, USA, 2000.

[5] Hibernate Community Documentation, Chapter 11. Transactions and Concurrency, http://docs.jboss.org/hibernate/core/3.3/reference/en/html/transactions.html, (2014).

[6] A.T.F. Hutt, Data mappings again, Software: Practice and Experience, 8(1978), pp. 483493.

[7] C. Ireland, D. Bowers, M. Newton, K. Waugh, A classification of object-relational impedance mismatch, in Proc. of First International Conference on Advances in Databases, Knowledge, and Data Applications, IEEE, 2009, pp. 36-43.

[8] R. Kimball, J. Caserta, The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data, John Wiley & Sons, Inc., 2004.

[9] G. Knolmayer, H. Herbst, M. Schlesinger, Enforcing Business Rules by the Application of Trigger Concepts, in Proc. of Priority Programme Informatics Research, Information Conference, Module 1, Swiss National Science Foundation, Berne, Switzerland, 1994, pp. 24-30.

[10] V. Krishnamurthy, S. Banerjee, A. Nori, Bringing object-relational technology to the mainstream, in Proc. of SIGMOD '99 ACM International Conference on Management of Data, ACM, New York, 1999, pp. 513-514.
[11] J. Lerman, Programming Entity Framework, Building Data Centric Apps with the ADO.NET Entity Framework, OReilly Media, Sebastopol, 2009.
[12] D.S. Linthicum, Enterprise Application Integration. Addison-Wesley Longman, Reading, MA, USA, 2000.
[13] O.L. Madsen, Open issues in object-oriented programming - A Scandinavian perspective, Software: Practice and Experience, 25(S4), (1995), pp. 343.
[14] J. Melton, A.R. Simon, SQL: 1999, Understanding Relational Language Components, Chapter 11 Active Databases and Triggers. Morgan Kaufmann, San Francisco, 2002.
[15] Microsoft MSDN documentation, Tracking Data Changes, Change Data Capture, http://msdn.microsoft.com/en-us/library/bb522489(v=sql.105).aspx, (2014).
[16] Microsoft SQL Server documentation, For XML (SQL Server), http://technet.microsoft.com/en-us/library/ms178107(v=sql.100).aspx, (2014).
[17] Microsoft SQL Server documentation, SET IDENTITY_INSERT (Transact-SQL), http://technet.microsoft.com/en-us/library/ms188059.aspx, (2014).
[18] E.J. O'Neil, Object/relational mapping 2008: hibernate and the entity data model, in Proc. of SIGMOD '08 ACM SIGMOD International Conference on Management of data, ACM, New York, 2008, pp. 1351-1356.
[19] M. Olan, Unit testing: test early, test often, J. Comput. Sci. in Colleges archive, 19(2003), pp. 319-328.
[20] Oracle9i Data Warehousing Guide, Release 2 (9.2), Part Number A96520-01. Oracle Corporation, 2002.
[21] Oracle9i Supplied PL/SQL Packages and Types Reference, Release 2 (9.2), Part Number A96612-01, Chapter 85: Functions and Procedures of DBMS_XMLGEN, Oracle Corporation, 2002.
[22] C. Popfinger, Enhanced active database for federated information systems, Doctoral Thesis/Dissertation, Heinrich-Heine-Universität Düsseldorf, GRIN Verlag, Düsseldorf, 2007.
[23] G. Schulmeyer, J.I. McManus, Handbook of Software Quality Assurance, Van Nostrand Reinhold Co., New York, 1987.
[24] N. Sharma, L. Perniu, R.F. Chong, A. Iyer, A.C. Mitea, C. Nandan, M. Nonvinkere, M. Danubianu, Database Fundamentals. IBM Corporation, 2010.

Department of Computer Systems, Faculty of Mathematics and Informatics, Plovdiv University 'Paisii Hilendarski', Tzar Asen str. 24, 4000 Plovdiv, Bulgaria

E-mail address: radoslav_radev@gbg.bg