Recommendations for Evolving Relational Databases: Technical Report

Julien Delplanque, Anne Etien, Nicolas Anquetil, Stéphane Ducasse

Recommendations for Evolving Relational Databases
Technical Report

Julien Delplanque[1,2], Anne Etien[1,2], Nicolas Anquetil[1,2], and Stéphane Ducasse[2,1]

[1] Univ. Lille, CNRS, Centrale Lille, Inria UMR 9189 - CRIStAL
[2] INRIA Lille Nord Europe, Villeneuve d'Ascq, France
{firstname}.{lastname}@inria.fr

Abstract. This report contains technical details that could not be included in the article "Recommendations for Evolving Legacy Databases" submitted to the 32nd International Conference on Advanced Information Systems Engineering (CAISE'20).

1 Operators

This section provides additional information related to operators used by DBEvolution in the context of CAISE'20 article experiment.

1.1 Definition

An operator is defined as follows:

- *A signature*: Defines the unique name and input parameters of an operator.
- *A description*: The description briefly explains the purpose of the operator.
- *One or multiple categories of entities*: A change may *impact* entities (instances of the meta-model concepts) of different kinds (*e.g.,* column, table, view, column reference, table reference, ...) and each kind may be handled differently. Moreover, within a given kind, entities with different properties may also be handled differently. For example, columns may carry different constraints (foreign key, *default*, or *not null*) requiring different treatments. Each entity kind and property subgroup defines a *category of entities*. Each *category of entities* has an identifier C_n.
- *One or multiple recommendations for each category*: For each C_n, at least one *recommendation* should be provided. The *recommendation* consists in the application of one or multiple operator(s) on entities of C_n and a textual description of its meaning in natural language. This description provides some context to justify why an operator is needed. If multiple recommendations are provided, the database architect needs to choose the operator fitting their needs.

2 Operators Catalog

This section provides specifications only for operators used in the context of the experiment described in the paper.

2.1 Rename Local Variable

RenameLocalVariable renames a local variable of a stored procedure. This local variable might be used inside the stored procedure source code.
Signature: RenameLocalVariable(localVariable, newName)
Description: Renames a local variable of a stored procedure.
Impact: Let I be the set of entities impacted by the change;

- $C_1(I) = I$ Deals with references to the local variable from inside the stored procedure holding it.

Recommendations:

- $C_1 \rightarrow$ ChangeReferenceTarget: Since the local variable is renamed, the references need to be adapted to target the reference the local variable via its new name.

2.2 Rename Function

RenameFunction renames a stored procedure in the database. This stored procedure might be called by behavioral entities in the database.
Signature: RenameFunction(function, newName)
Description: Renames a stored procedure in the database.
Impact: Let I be the set of entities impacted by the change;

- $C_1(I) = \{e \in I | has\_type(source(e), SelectClause)\}$ Deals with references to the stored procedure from select clause in SELECT query.
- $C_2(I) = \{e \in I | \neg has\_type(source(e), SelectClause)\}$ Deals with other references to the stored procedure.

Recommendations:

- $C_1 \rightarrow$ Choice(ChangeReferenceTargetInSelectClause, AliasColumnDeclaration): Calls made to the stored procedure in a select clause cause the name of the associated column to be the same as the procedure name if no aliasing is done. Thus, the architect needs to choose between changing call target and aliasing the column with the old stored procedure name to stop the impact propagation or only changing call target and thus propagating further the impact.
- $C_2 \rightarrow$ ChangeReferenceTarget: Since the stored procedure is renamed, the calls need to be adapted to target the reference the stored procedure via its new name. This recommendation is similar to what is done when renaming a function in procedural programming languages.

2.3 Remove Function

RemoveFunction removes a stored procedure from the database. This stored procedure might be used by behavioral entities of the database.
Signature: RemoveFunction(function)
Description: Removes a stored procedure from the database.
Impact: Let I be the set of entities impacted by the change;

- $C_1(I) = I$ Deals with calls to the stored procedure.

Recommendations:

- $C_1 \rightarrow$ HumanIntervention: The removal of the method induces complex changes in the source code which need to be evaluated and performed locally by the architect on each entity that was calling the stored procedure.

2.4 Remove View

RemoveView removes a view from the database. This view might be used by behavioral entities of the database.
Signature: RemoveView(view)
Description: Removes a view from the database.
Impact: Let I be the set of entities impacted by the change;

- $C_1(I) = I$ Deals with all references to the view.

Recommendations:

- $C_1 \rightarrow$ HumanIntervention: The removal of the view induces complex changes in the source code which need to be evaluated and performed locally by the architect on each entity that was referencing the view.

2.5 Rename Column

RenameColumn renames a columns in a table of the database. This column might be referenced by other entities in the system.
Signature: RenameColumn(view)
Description: Renames a column in a table of the database
Impact: Let I be the set of entities impacted by the change;

- $C_1(I) = \{e \in I | is\_wildcard\_reference(e)\}$ Deals with references to the column through a wildcard ($\star$).

- $C_2(I) = \{e \in I | has\_type(source(e), SelectClause)\}$ Deals with references to the column from select clause in SELECT query.
- $C_3(I) = \{e \in I | \neg has\_type(source(e), SelectClause)\}$ Deals with other references to the column.

Recommendations:

- $C_1 \rightarrow$ DoNothing: If the column to be renamed is referenced through SQL's wildcard ($\star$), nothing needs to be done. Indeed, as the wildcard does not refer to the column via its name but indirectly, the reference is not impacted by the renaming.
- $C_2 \rightarrow$ Choice(ChangeReferenceTargetInSelectClause, AliasColumnDeclaration): References to columns made in a `SELECT` clause define the names and types of the resulting table. Thus, when updating the reference made to a column in the `SELECT` clause, the architect needs to decide if the impact should be propagated to the resulting table users (by changing the reference target) or not (by changing the reference target and aliasing the column declaration with the old referenced entity name).
- $C_3 \rightarrow$ ChangeReferenceTarget: Other references need to be updated to refer to the new name of the column.

2.6 ChangeReferenceTarget

ChangeReferenceTarget changes the entity targetted by a reference by rewritting the corresponding source code. This operator must not receive a reference that occurs in a `SELECT` clause.
Signature: ChangeReferenceTarget(reference, newTarget)
Description: Changes the entity targetted by a reference by rewriting the corresponding source code.
Impact: None
Recommendations: None

2.7 ChangeReferenceTargetInSelectClause

ChangeReferenceTargetInSelectClause changes the target of a reference that occurs in a `SELECT` clause. This operator propagate the impact of this reference change to users of the table resulting from the `SELECT` query execution.
Signature: ChangeReferenceTargetInSelectClause(reference, newTarget)
Description: Changes the target of a reference that occurs in a `SELECT` clause.
Impact: Let I be the set of entities impacted by the change;

- $C_1(I) = I$ Deals with the column of the view defined by the reference.

Recommendations:

- $C_1 \rightarrow$ RenameColumn: As the reference made in the `SELECT` clause is updated without aliasing, the change is equivalent to renaming the column of the table resulting from the `SELECT` query.

2.8 AliasColumnDeclaration

AliasColumnDeclaration changes the target of the reference in the `SELECT` clause and create an alias with the old reference name. Because of the aliasing, this operator stops the propagation of impact.
Signature: AliasColumnDeclaration((view, reference, newTarget))
Description: Changes the target of the reference in the SELECT clause and create an alias with the old reference name.
Impact: None
Recommendations: None

2.9 DoNothing

DoNothing Does nothing, it is the null-operator.
Signature: DoNothing()
Description: Does nothing.
Impact: None
Recommendations: None

2.10 HumanIntervention

HumanIntervention Asks the database architect to perform the intervention on the source code of an entity. This operator generates no impact as we consider that the DBA modify the source code of the entity provided as argument in a way that there is no additional impact.
Signature: HumanIntervention(entity)
Description: Let the architect edit the source code of an entity manually.
Impact: None
Recommendations: None

3 Implementation of the Approach

This approach was implemented in a tool based on Moose[3], a software and data-analysis platform. Figure 1 shows a screen capture of the tool user interface that guides the architect through the steps of choosing recommendations.

Panel 1 shows the list of operators selected by the user and the tree of impacts resulting from the architect's choices. When an operator is inserted or clicked in panel 1, panel 2 shows the entities potentially impacted by the operator. The UI allows unfolding these impacted entities to show one or many choices the architect needs to make on them. The "gear and spanner" icon means that the architect still needs to choose a *recommendation*. The green check icon means that the architect already chose a *recommendation*. When one of the entries in panel 2 is clicked, two things happen: First, panel 3 shows the different recommendations the architect can choose; second, panel 4 shows the source code of the entity that is concerned (if it holds source code) with the reference to the entity that created the impact highlighted. In panel 3, the "Use this operator" button allows one to choose a *recommendation*.

Finally, once all the choices were made (all operators in panel 1 have the green check icon), the architect can click the "Generate patch" button (top left of the tool) to generate the SQL script.

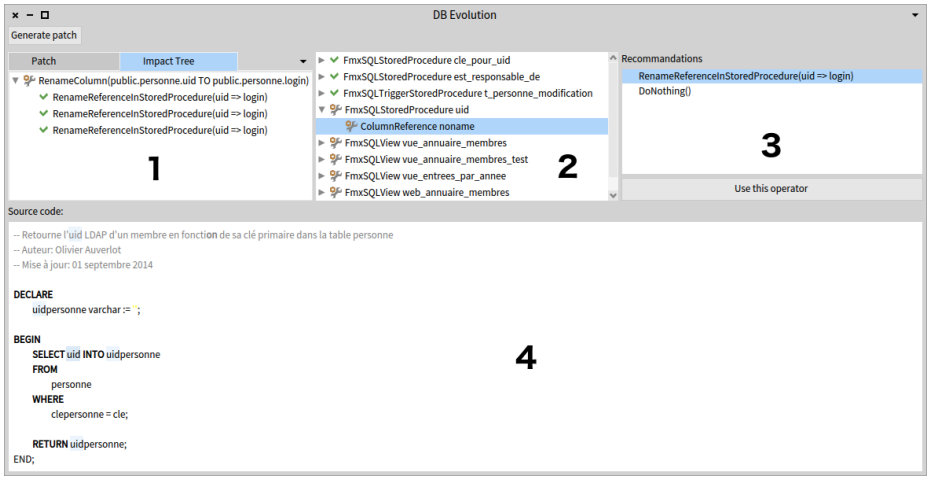

Fig. 1. User interface for the implementation of the approach.

[3] http://www.moosetechnology.org