DELTA*
A tool for database refactoring

Patrick Stünkel

Western Norway University of Applied Sciences
*Supported by the "Modern Refactoring" bilateral SIU/CAPES project

November 21, 2018
About Me

- B.Sc. and M.Sc. from FHDW Hannover in Germany
  - There *DELTA* was developed.
- 3 years in industry as a Software Engineer
- Since September 2017: *PhD research fellow at Western Norway University of Applied Sciences*
  - Topic: Interoperability in Model Driven Software Engineering (MDSE)
  - Areas: MDSE, Bidirectional Transformations (BX), Co-Evolution
Motivation

Software refactoring is widely adopted and is performed automatically...
Software refactoring is widely adopted and is performed automatically...

... whereas database refactoring is not.
Software refactoring is widely adopted and is performed automatically...

... whereas database refactoring is not.

Ambler/Sadalage „Database Refactoring“, 2006 [1]
Introduction

Theory

Application

Related Work

Conclusion and Outlook

Motivation

Software refactoring is widely adopted and is performed automatically...

... whereas database refactoring is not.

Ambler/Sadalage „Database Refactoring“, 2006 [1]

**Definition 1 (Ambler):** Database refactoring

A simple change to a database schema that improves its design while retaining both its behavioural and informational semantics.
What makes Database refactoring different?
Distinction: code refactoring - database refactoring

What makes Database refactoring different?

▶ deals with behavioural and informational aspects,
What makes Database refactoring different?

- deals with behavioural and informational aspects,
- affects actual stored data,
What makes Database refactoring different?

- deals with behavioural and informational aspects,
- affects actual stored data,
- requires manual effort,
Distinction: code refactoring - database refactoring

What makes Database refactoring different?

- deals with behavioural and informational aspects,
- affects actual stored data,
- requires manual effort,
- in general a database schema is shared between many different applications.
What makes Database refactoring different?

- deals with behavioural and informational aspects,
- affects actual stored data,
- requires manual effort,
- in general a database schema is shared between many different applications.

**Requirement 1: Transition periods**

The database has to be accessible through the new and the old schema after a refactoring for certain transition period because the different dependent applications need time to adopt the changed schema.
Further requirements

Requirement 2: Generated Migrations
As the manual development of migration procedures is error-prone and there are recurring patterns of migrations the migration code itself shall be generated.

Requirement 3: Revertible Migrations
As a refactoring must not change the behaviour of the data or cause information loss the refactoring can be reverted. Therefore we require an undo-feature for our migrations.

Requirement 4: Avoid downtimes
As long downtimes have negative impact and productivity and sale they shall be avoided whilst applying refactorings.
Further requirements

Requirement 2: Generated Migrations
As the manual development of migration procedures is error-prone and there are recurring patterns of migrations the migration code itself shall be generated.
Further requirements

Requirement 2: Generated Migrations
As the manual development of migration procedures is error-prone and there are recurring patterns of migrations the migration code itself shall be generated.

Requirement 3: Revertible Migrations
As a refactoring must not change the behaviour of the data or cause information loss the refactoring can be reverted. Therefore we require an undo-feature for our migrations.
Further requirements

**Requirement 2: Generated Migrations**
As the manual development of migration procedures is error-prone and there are recurring patterns of migrations the migration code itself shall be generated.

**Requirement 3: Revertible Migrations**
As a refactoring must not change the behaviour of the data or cause information loss the refactoring can be reverted. Therefore we require an undo-feature for our migrations.

**Requirement 4: Avoid downtimes**
As long downtimes have negative impact and productivity and sale they shall be avoided whilst applying refactorings.
Maintaining compatibility

**Trigger**  Old and new schema running in parallel, trigger replicate changes back and forth.

**Views**  Views, representing the old schema, provide backward compatibility.

**Batch Jobs**  Old and new schema running in parallel, Batch jobs replicate changes back and forth on a regular basis.
Maintaining compatibility

Ambler and Sadalage suggest Trigger. We chose Views with Instead-Of-Trigger.
Maintaining compatibility

- Ambler and Sadalage suggest *Trigger*.
Maintaining compatibility

- Ambler and Sadalage suggest *Trigger*.
- We chose *Views* with Instead-Of-Trigger.
Assumptions

- Database schema represents an object-oriented-model, i.e. set of related entities.
Assumptions

- Database schema represents an object-oriented-model, i.e. set of related entities.
- Object-Relational-Mapping
  - EntityType $\rightarrow$ Table with surrogate ID
  - Attribute $\rightarrow$ Column
  - *:1-Association $\rightarrow$ Foreign-Key-Constraint
Assumptions

- Database schema represents an object-oriented-model, i.e. set of related entities.

- **Object-Relational-Mapping**
  - EntityType → Table with surrogate ID
  - Attribute → Column
  - *:1-Association → Foreign-Key-Constraint

- ⇒ Schema satisfies 2NF.
Catalogue

- Rename Table
- Rename Column
- Introduce Calculated Column
- Merge and Split Columns
- Spin-Off Empty-Table
- Move Column
- Merge and Split Table
- Transcode Foreign-Key
- Compose Foreign-Keys
Introduce Calculated Column

Table A

<table>
<thead>
<tr>
<th>ID</th>
<th>Column2</th>
<th>Column3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>column2value1</td>
<td>column3value1</td>
</tr>
<tr>
<td>2</td>
<td>column2value2</td>
<td>column3value2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Introduce Calculated Column

Table A

<table>
<thead>
<tr>
<th>ID</th>
<th>Column2</th>
<th>Column3</th>
<th>IntroducedColumn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>column2value1</td>
<td>column3value1</td>
<td>fn(1)</td>
</tr>
<tr>
<td>2</td>
<td>column2value2</td>
<td>column3value2</td>
<td>fn(2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Merge and Split Columns

### Table A

| ID  | ColumnX | ColumnDesc | ...
|-----|---------|------------|-----
| 1   | columnXvalue1 | A          | ... |
| 2   | columnXvalue2  | B          | ... |
| 3   | columnXvalue3  | A          | ... |
| ... | ...       | ...        | ... |

**Merge-Column**

| ID  | ColumnX1 | ColumnX2 | ColumnDesc | ...
|-----|----------|----------|------------|-----
| 1   | columnXvalue1 | NULL    | A          | ... |
| 2   | NULL     | columnXvalue2 | B        | ... |
| 3   | columnXvalue3 | NULL    | A          | ... |
| ... | ...       | ...      | ...        | ... |

**Split-Column**
Spin-Off Empty Table

Table A
Column1 <PK>
Column2
...

Spin-Off-Empty-Table

Table A
Column1 <PK>
Column2
...

Table A Spin Off
ID <PK>
ORIGIN_ID <FK>

Table A
Column1 <PK>
Column2
...

Spin-Off-Empty-Table-Inverse

Table A
Column1 <PK>
TARGET_ID <FK>
Column2
...

Table A Spin Off
ID <PK>
Move Column

Table A
ID <PK>
... 

Table B
ID <PK>
ORIGIN_ID <FK>
ColumnX
...

Move-Column

Table A
ID <PK>
ColumnX
...

Table B
ID <PK>
ORIGIN_ID <FK>
...

Table A
ID <PK>
ORIGIN_ID <FK>
ColumnX
...

Table B
ID <PK>
... 

Move-Column-Inverse

Table A
ID <PK>
ORIGIN_ID <FK>
ColumnX
...

Table B
ID <PK>
...
Merge and Split Table

### Table A
<table>
<thead>
<tr>
<th>ID</th>
<th>ColumnX</th>
<th>ColumnY</th>
<th>ColumnDesc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>columnXvalue1</td>
<td>columnYvalue1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>columnXvalue2</td>
<td>columnYvalue2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>columnXvalue3</td>
<td>columnYvalue3</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>columnXvalue4</td>
<td>columnYvalue4</td>
<td>B</td>
</tr>
</tbody>
</table>

### Table A1
<table>
<thead>
<tr>
<th>ID</th>
<th>ColumnX</th>
<th>ColumnY</th>
<th>ColumnDesc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>columnXvalue1</td>
<td>columnYvalue1</td>
<td>A</td>
</tr>
</tbody>
</table>

### Table A2
<table>
<thead>
<tr>
<th>ID</th>
<th>ColumnX</th>
<th>ColumnY</th>
<th>ColumnDesc</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>columnXvalue2</td>
<td>columnYvalue2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>columnXvalue3</td>
<td>columnYvalue3</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>columnXvalue4</td>
<td>columnYvalue4</td>
<td>B</td>
</tr>
</tbody>
</table>
Transcode Foreign Key

Table A
ID: Number <PK>
RefColumn: Number <FK>
Column3
Column4
...

Table B
ID: Number <PK>
Column2: Column2Type
...
UniqueColumn: Number <UNIQUE><NOT-NULL>

Transcode-Foreign-Key

Table A
ID: Number <PK>
RefColumn: Number <FK>
Column3
Column4
...

Table B
ID: Number <UNIQUE><NOT-NULL>
Column2
...
UniqueColumn: Number <PK>
Compose Foreign Keys

Table A
ID: Number <PK>
RefColumn: Number <FK>
Column3
...

Table B
ID: Number <PK>
Column2
RefColumn: Number <FK>
...

Table C
ID: Number <PK>
...

Decompose Foreign Keys

Compose Foreign Keys

Table A
ID: Number <PK>
RefColumn: Number <FK>
Column3
...

Table B
ID: Number <PK>
Column2
RefColumn: Number <FK>
...

Table C
ID: Number <PK>
...
Offline Batch-Migration

- Shut down database.
- Apply schema changes and migrate data.
- Restart database.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to implement</td>
<td>May cause long downtimes on large data sets</td>
</tr>
</tbody>
</table>
Stepwise, Transactional Migration

- Create new schema (empty).
- Copy data to the new schema on-access or per batch.
- View layer on the new schema aggregates data from old and new schema.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long downtimes avoided</td>
<td>Higher complexity due to aggregation</td>
</tr>
<tr>
<td></td>
<td>Temporary trigger required</td>
</tr>
</tbody>
</table>
Two-Phase Migration

- Copy database.
- Apply schema changes to one copy (incl. data migration).
- Original schema receives triggers, which log every data manipulation.
- The logged actions are applied to the new schema copy bit by bit.
- Eventually all data is migrated to the new schema.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long downtimes avoided</td>
<td>Higher complexity due to merge</td>
</tr>
<tr>
<td>Concurrent migration</td>
<td>Temporary trigger for logging required</td>
</tr>
</tbody>
</table>
Design Goals

- Independent, encapsulated refactorings
- A refactorings is composed of Deltas
- Virtual preview of schema modification
- Goal in the long run: Merge of refactorings!
Design Goals

- Independent, encapsulated refactorings
Design Goals

- Independent, encapsulated refactorings
- A refactorings is composed of *Deltas*
Design Goals

- Independent, encapsulated refactorings
- A refactorings is composed of *Deltas*
- Virtual preview of schema modification
Design Goals

- Independent, encapsulated refactorings
- A refactorings is composed of *Deltas*
- Virtual preview of schema modification
- Goal in the long run: Merge of refactorings!
Design Goals

- Independent, encapsulated refactorings
- A refactorings is composed of *Deltas*
- Virtual preview of schema modification
- Goal in the long run: Merge of refactorings!

**DELTA:**
- Tool for database migration (currently Oracle)
- Java 8
- JavaFX-GUI
- Three-Layer-Architecture
- 50 Deltas (9 "real"refactorings)
Starting point

```
Person
id: Number <PK>
name: Varchar
dateOfBirth: Date
street: Varchar
city: Varchar
zipCode: Char(5)
gender: Char(1)
```

```
Contract
id: Number <PK>
personId: Number <FK>
insuranceId: Number <FK>
validFrom: Timestamp
```

```
CarInsurance
id: Number <PK>
price: Number(30,2)
horsePower: Number
yearOfConstruction: Date
```
Address is part of Person
Step 1: Spin-Off the Address

<table>
<thead>
<tr>
<th>Person&lt;View&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: Number &lt;PK&gt;</td>
</tr>
<tr>
<td>name: Varchar</td>
</tr>
<tr>
<td>dateOfBirth: Date</td>
</tr>
<tr>
<td>gender: Char(1)</td>
</tr>
<tr>
<td>street: Varchar</td>
</tr>
<tr>
<td>city: Varchar</td>
</tr>
<tr>
<td>zipcode: Char(5)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: Number &lt;PK&gt;</td>
</tr>
<tr>
<td>personId: Number &lt;FK&gt;</td>
</tr>
<tr>
<td>street: Varchar</td>
</tr>
<tr>
<td>city: Varchar</td>
</tr>
<tr>
<td>zipcode: Char(5)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: Number &lt;PK&gt;</td>
</tr>
<tr>
<td>personId: Number &lt;FK&gt;</td>
</tr>
<tr>
<td>name: Varchar</td>
</tr>
<tr>
<td>dateOfBirth: Date</td>
</tr>
<tr>
<td>gender: Char(1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contract</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: Number &lt;PK&gt;</td>
</tr>
<tr>
<td>personId: Number &lt;FK&gt;</td>
</tr>
<tr>
<td>insuranceId: Number &lt;FK&gt;</td>
</tr>
<tr>
<td>validFrom: Timestamp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CarInsurance</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: Number &lt;PK&gt;</td>
</tr>
<tr>
<td>price: Number(30,2)</td>
</tr>
<tr>
<td>horsePower: Number</td>
</tr>
<tr>
<td>yearOfConstruction: Date</td>
</tr>
</tbody>
</table>
Step 1: Spin-Off the Address

- transition phase
Step 1: Spin-Off the Address

- transition phase
- no distinction between legal and natural persons
Step 2: Split in natural and legal persons

- **Person**
  - Id: Number
  - Name: Varchar
  - DateOfBirth: Date
  - Gender: Char(1)

- **Address**
  - Id: Number <PK>
  - PersonId: Number <FK>
  - Street: Varchar
  - City: Varchar
  - ZipCode: Char(5)

- **Contract**
  - Id: Number <PK>
  - PersonId: Number <FK>
  - InsuranceId: Number <FK>
  - ValidFrom: Timestamp

- **CarInsurance**
  - Id: Number <PK>
  - Price: Number(30,2)
  - HorsePower: Number
  - YearOfConstruction: Date

- **LegalPerson**
  - Id: Number <PK><FK>

- **NaturalPerson**
  - Id: Number <PK><FK>
  - Gender: Char(1)
Step 2: Split in natural and legal persons

- no support for multiple tenants
Step 3: Implement Multi-Tenancy
Step 3: Implement Multi-Tenancy

- transition phase
Case study

Company Merger

- Introduction
- Theory
- Application
- Related Work
- Conclusion and Outlook

**Case study**

- **bad design:** `insuranceId` and `liabilityInsuranceId` are disjoint

**Entity Diagram**

- **Person**
  - `id`: Number (PK)
  - `name`: Varchar
  - `dateOfBirth`: Date

- **Address**
  - `id`: Number (PK)
  - `postalCode`: Number (PK)
  - `street`: Varchar
  - `city`: Varchar
  - `zipCode`: Char(5)

- **Contract**
  - `id`: Number (PK)
  - `firstNamePersonId`: Number (FK)
  - `secondPersonId`: Number (FK)

- **CarInsurance**
  - `id`: Number (PK)
  - `price`: Number (30.2)
  - `horsePower`: Number
  - `yearOfConstruction`: Date

- **LiabilityInsurance**
  - `id`: Number (PK)
  - `price`: Number (30.2)
  - `carPayment`: Number (30.2)
bad design: insuranceId and liabilityInsuranceId are disjoint
Schritt 4: Concrete-Table-Inheritance $\rightarrow$ Class-Table-Inheritance

Introduction
Theory
Application
Related Work
Conclusion and Outlook

Case study

Schritt 4: Concrete-Table-Inheritance $\rightarrow$ Class-Table-Inheritance

Class-Table-Inheritance

Step 1:
- Spin Off Empty Table
- Merge Columns

Step 2:
- Union Tables

Step 4:
- Merge Columns
Case Study: Wrap-Up

- Extract entities: transformation of associations $1:1 \rightarrow 1:* \rightarrow *:*$
Case Study: Wrap-Up

- Extract entities: transformation of associations $1:1 \rightarrow 1:* \rightarrow *:*$
- Split entities: Single-Table-Inheritance $\rightarrow$ Class-Table-Inheritance
Case Study: Wrap-Up

- Extract entities: transformation of associations 1:1 → 1:* → *:* 
- Split entities: Single-Table-Inheritance → Class-Table-Inheritance 
- Extend models (multi-tenancy) with calculated columns
Case Study: Wrap-Up

- Extract entities: transformation of associations $1:1 \rightarrow 1:* \rightarrow *:*$
- Split entities: Single-Table-Inheritance $\rightarrow$ Class-Table-Inheritance
- Extend models (multi-tenancy) with calculated columns
- Transformation of models (Merge, „Transcode“ etc.): Concrete-Table-Inheritance $\rightarrow$ Class-Table-Inheritance
### Existing tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SQL Prompt</strong></td>
<td>VisualStudio-PlugIn, supports renaming and split-table</td>
</tr>
<tr>
<td><strong>ApexSQL</strong></td>
<td>Only MS-SQL-Server, supports renaming, split-table and introduce association-table.</td>
</tr>
<tr>
<td><strong>Flyway</strong></td>
<td>Version control and management of migration scripts, no implemented refactorings</td>
</tr>
<tr>
<td><strong>Liquibase</strong></td>
<td>Same as Flyway but with SQL-Abstraction and some built-in-refactorings (rename, split-table)</td>
</tr>
</tbody>
</table>
Related research

- I. Skoulis et. al. *Growing up with stability: How open-source relational databases evolve* [4]: study of the VCS-history to analyse patterns in the development of the database model.

- C. Curino et. al. *Graceful database schema evolution: the prism workbench* [5]. Similar approach to DELTA, based on SQL rewriting.

Conclusion

- Catalogue of database refactorings
- Backward-compatibility with Views and Instead-Of-Trigger
- Presentation of the different migration scenarios
- Open-Source-Tool: DELTA
Outlook

- Evaluation of the prototype
- Support of the different migration scenarios
- Composition of refactorings
- Reordering of refactoring
Outlook

- Evaluation of the prototype
- Support of the different migration scenarios
- Composition of refactorings
- Reordering of refactorings

Thank you for your attention!
[1] Scott W Ambler and Pramod J Sadalage

Refactoring : improving the design of existing code. Addison-Wesley, 1999.

Refactoring information systems - association folding and unfolding. FHDW Hannover, 2013.

[4] Ioannis Skoulis, Panos Vassiliadis, Apostolos V. Zarras


Evolution of databases using petri nets. Anais de XIX Congresso Brasileiro de Automatica, CBA 2012