DELTA*

A tool for database refactoring

Patrick Stünkel

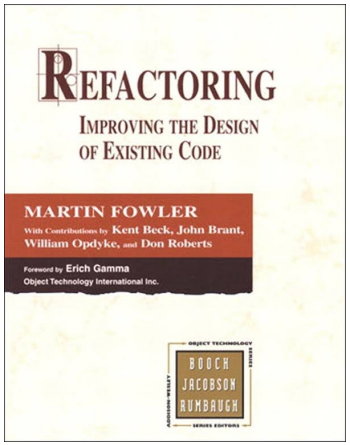
Western Norway University of Applied Sciences
*Supported by the "Modern Refactoring"bilateral SIU/CAPES project

November 21, 2018

About Me

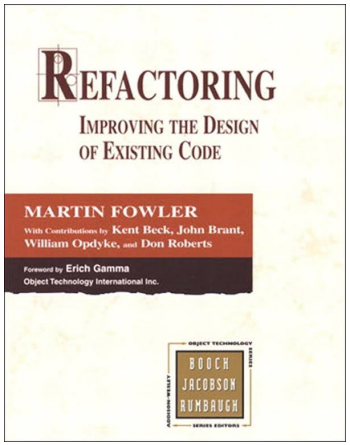
- B.Sc. and M.Sc. from FHDW Hannover in Germany
  - There *DELTA* was developed.
- 3 years in industry as a Software Engineer
- Since September 2017: *PhD research fellow* at *Western Norway University of Applied Sciences*
  - Topic: Interoperability in Model Driven Software Engineering (MDSE)
  - Areas: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

Motivation



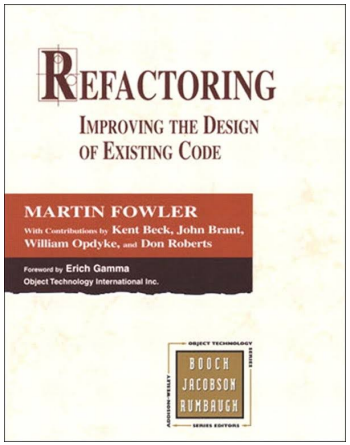
- Software refactoring is widely adopted and is performed automatically...

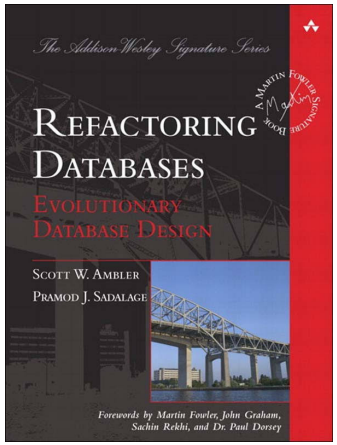
Motivation



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

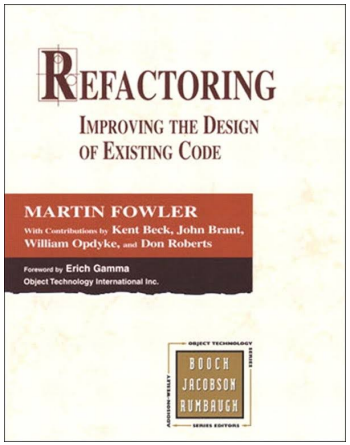
Motivation

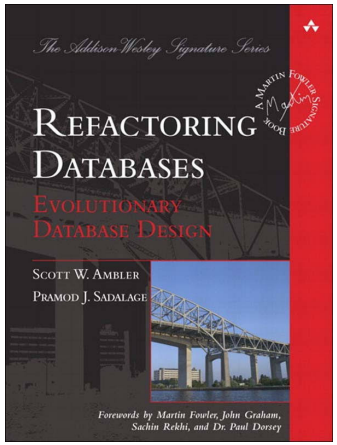




- Software refactoring is widely adopted and is performed automatically...
- ... whereas database refactoring is not.
- Ambler/Sadalage „Database Refactoring“, 2006 [1]

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.

Distinction: code refactoring - database refactoring

What makes Database refactoring different ?

Distinction: code refactoring - database refactoring

What makes Database refactoring different ?

- deals with behavioural and informational aspects,

Distinction: code refactoring - database refactoring

What makes Database refactoring different ?

- deals with behavioural and informational aspects,
- affects actual stored data,

Distinction: code refactoring - database refactoring

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.

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.

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

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.

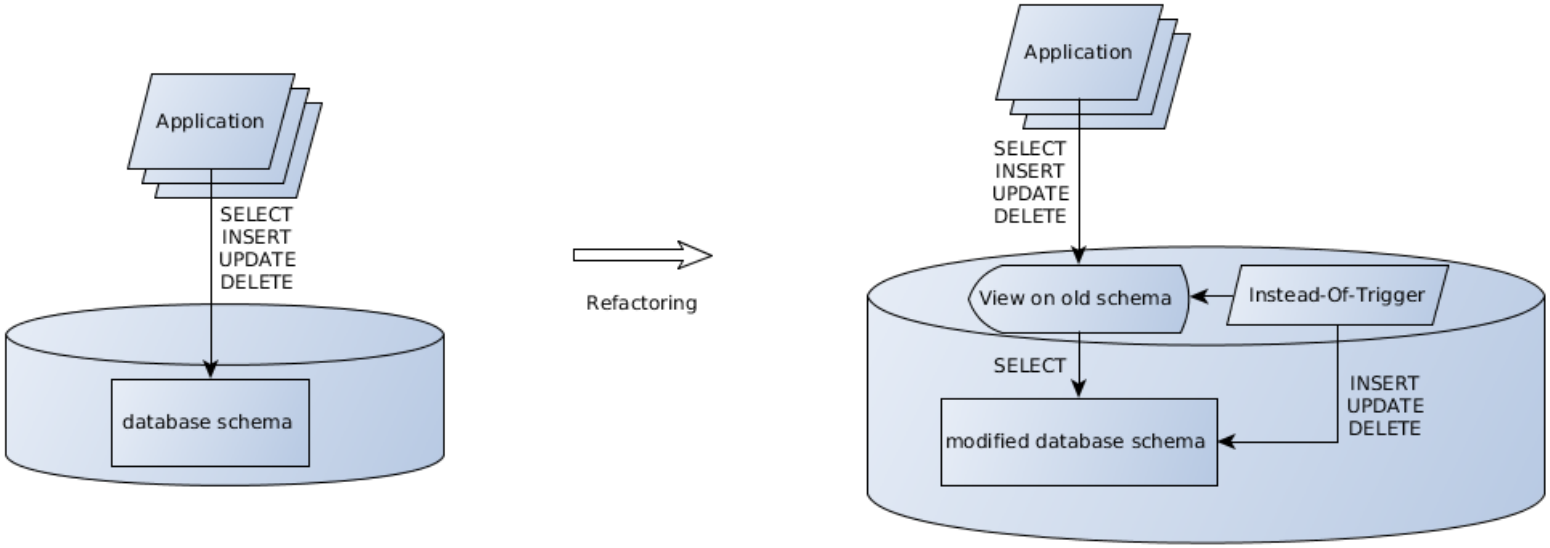
Maintaining compatibility

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

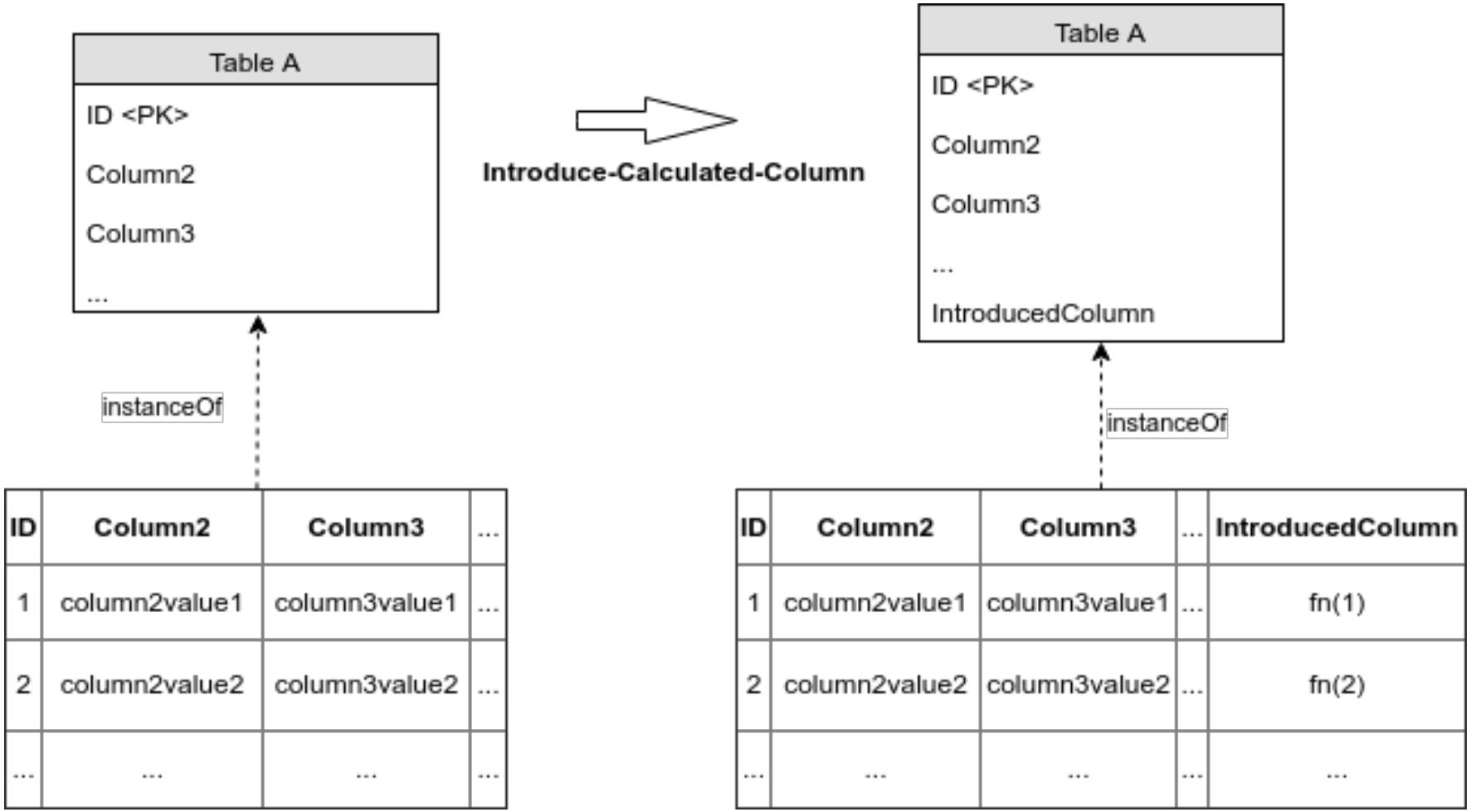
Assumptions

- Database schema represents an object-oriented-model, i. e. set of related entities.
- Object-Relational-Mapping
  - EntityType $\rightarrow$ Table with surrogate ID
  - Attribute $\rightarrow$ Column
  - *:1-Association $\rightarrow$ Foreign-Key-Constraint
- $\Rightarrow$ Schema satisfies 2NF.

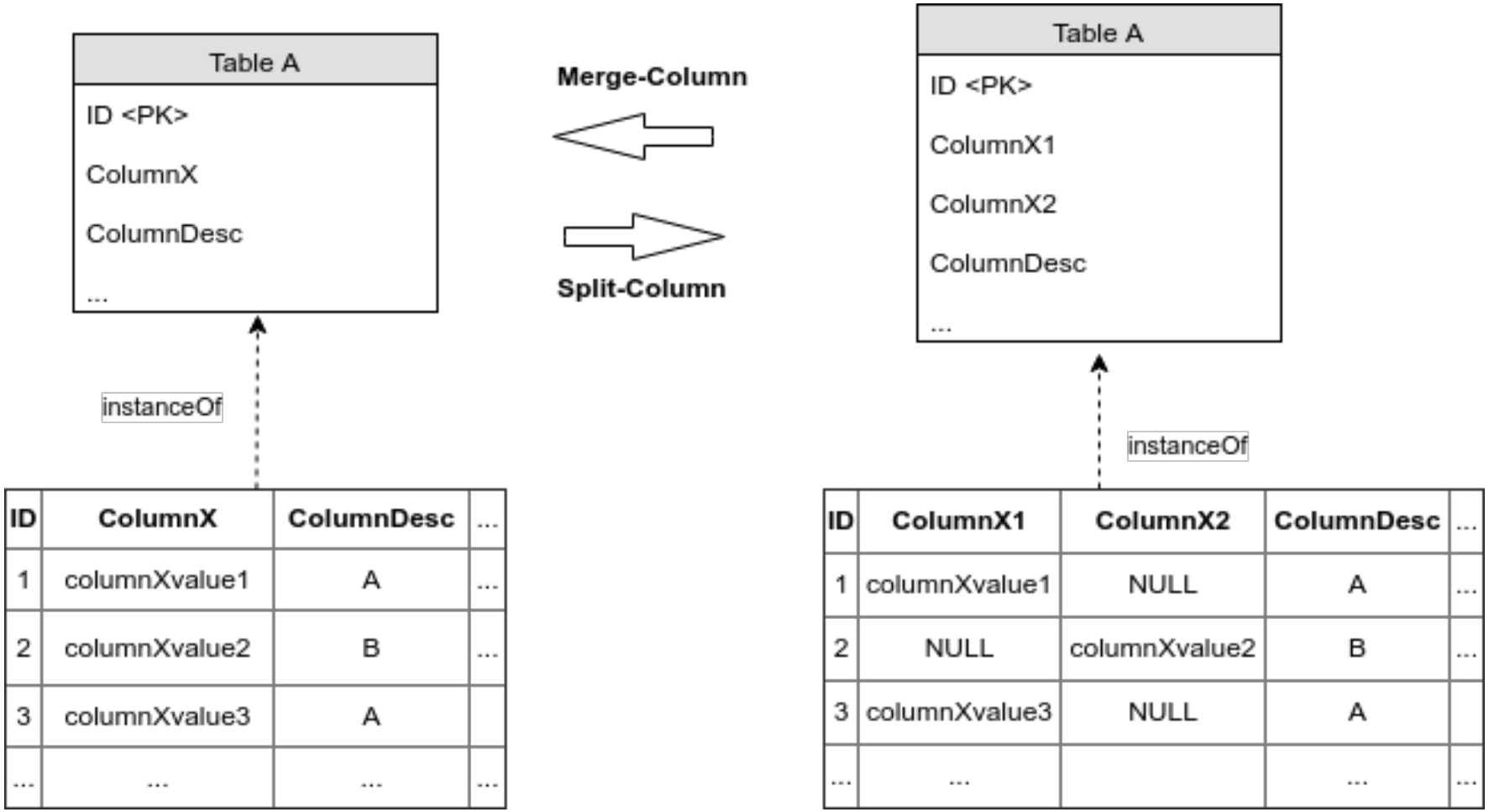
Catalogue

- 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

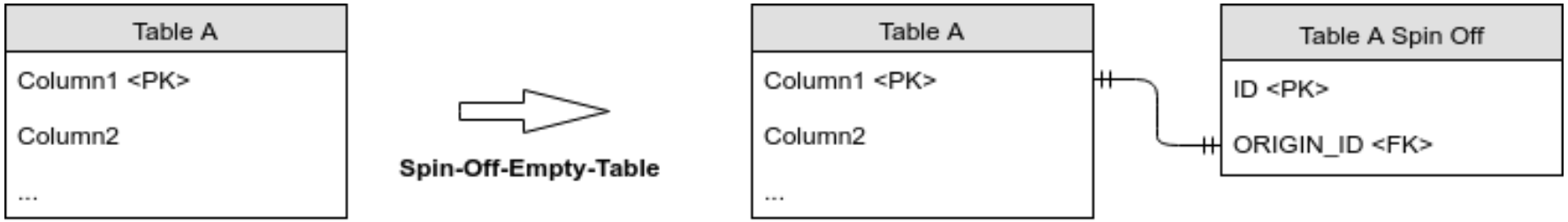
Introduce Calculated Column

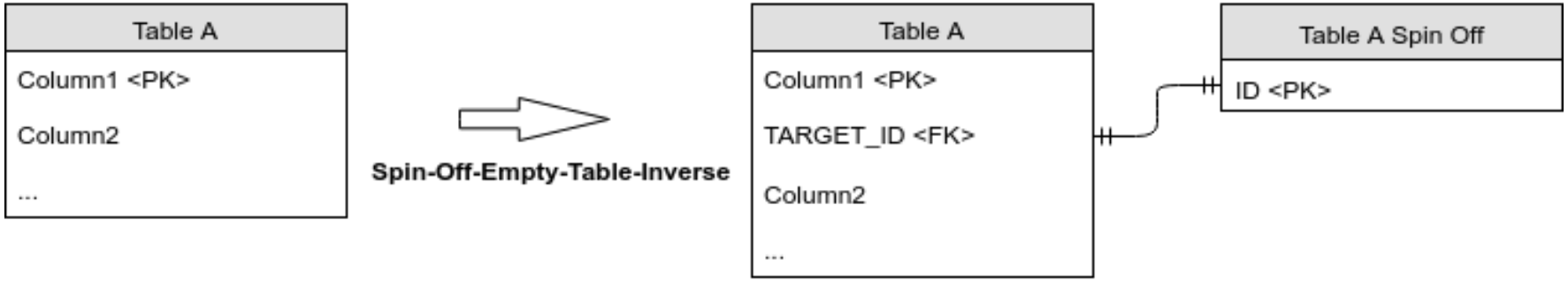


Merge and Split Columns



Spin-Off Empty Table





Move Column

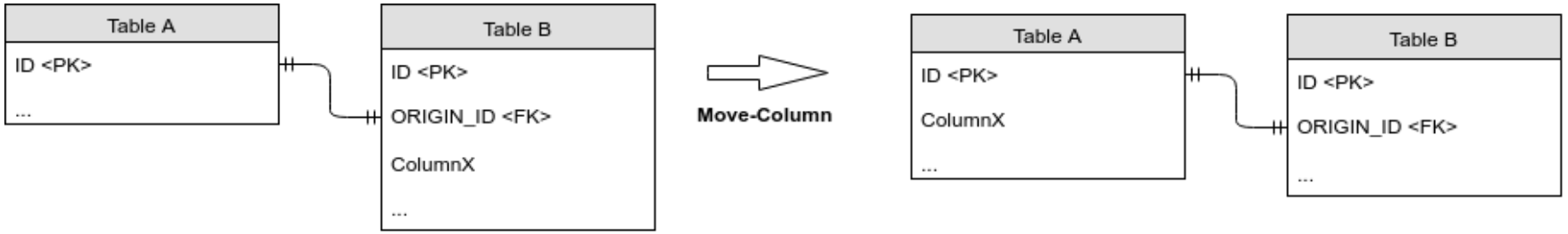


Table A
ID <PK>
...
Table B
ID <PK>
ORIGIN_ID <FK>
ColumnX
...
Move-Column
Table A
ID <PK>
ColumnX
...
Table B
ID <PK>
ORIGIN_ID <FK>
...

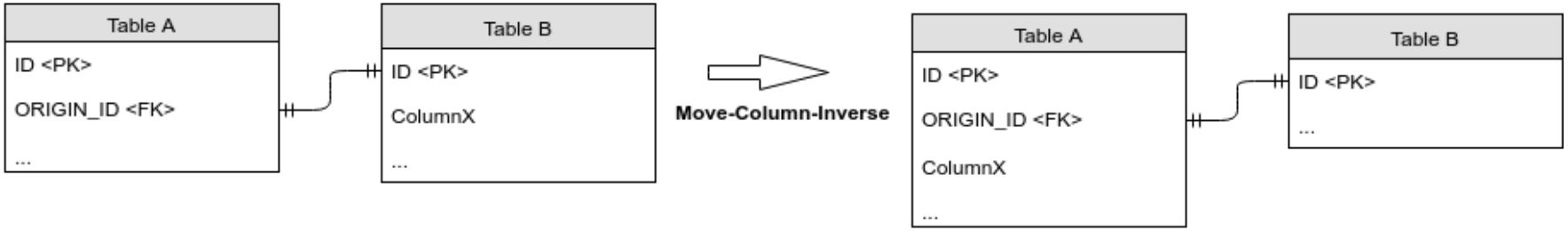
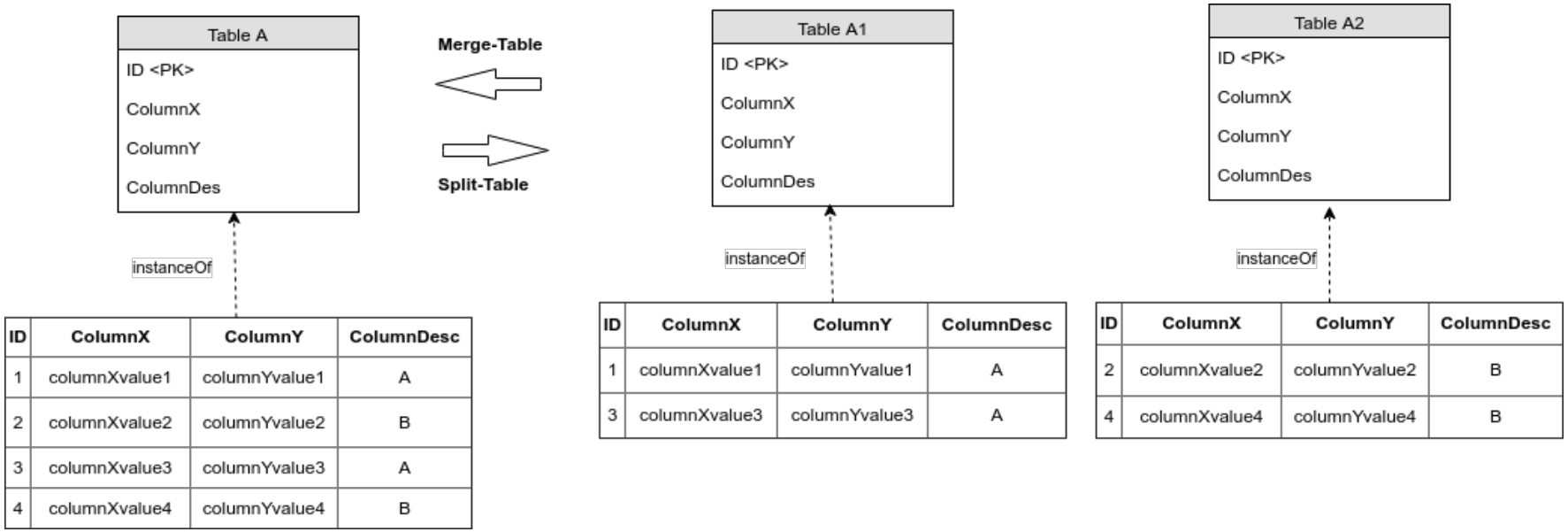


Table A
ID <PK>
ORIGIN_ID <FK>
...
Table B
ID <PK>
ColumnX
...
Move-Column-Inverse
Table A
ID <PK>
ORIGIN_ID <FK>
ColumnX
...
Table B
ID <PK>
...

Merge and Split Table

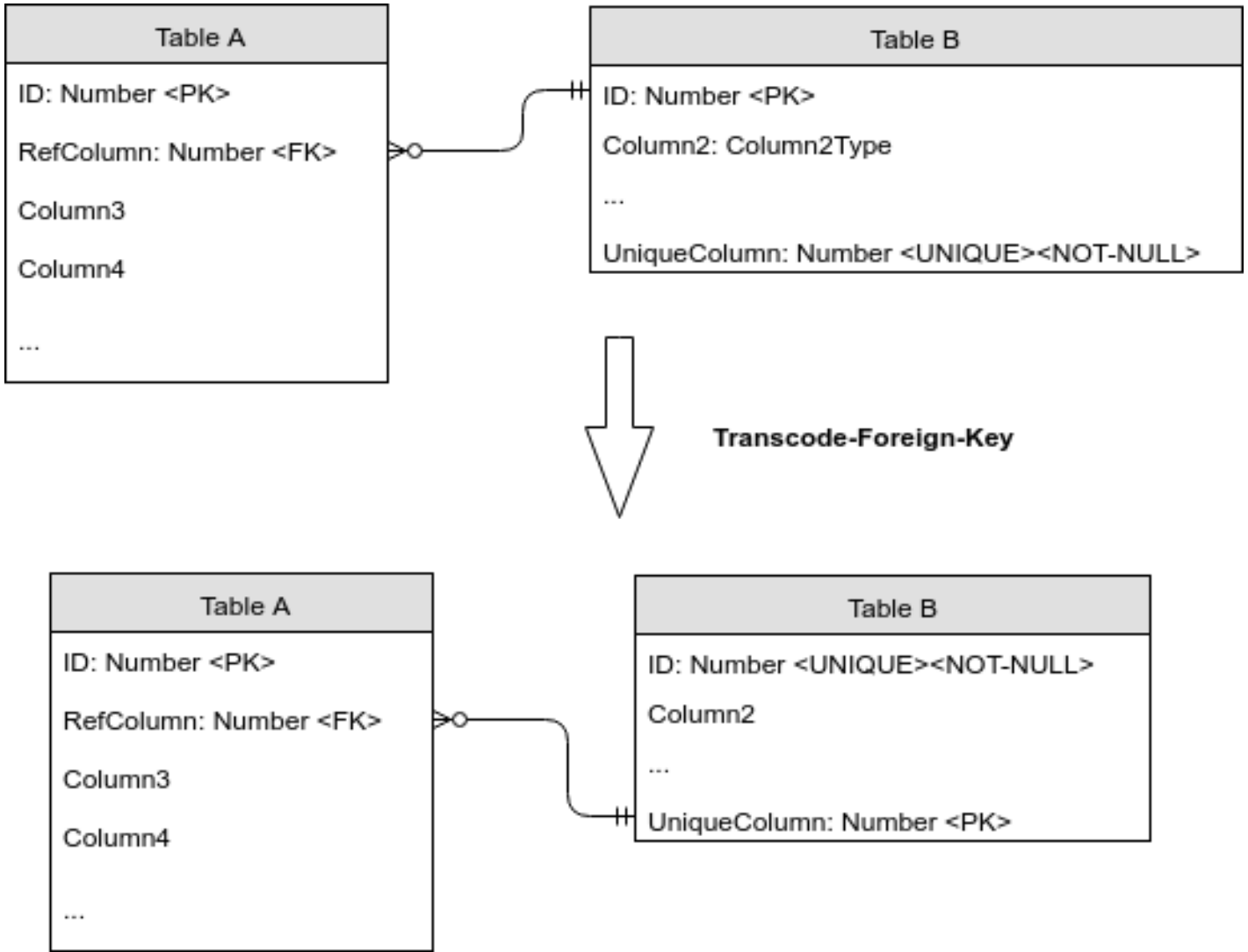


ID	ColumnX	ColumnY	ColumnDesc
1	columnXvalue1	columnYvalue1	A
2	columnXvalue2	columnYvalue2	B
3	columnXvalue3	columnYvalue3	A
4	columnXvalue4	columnYvalue4	B

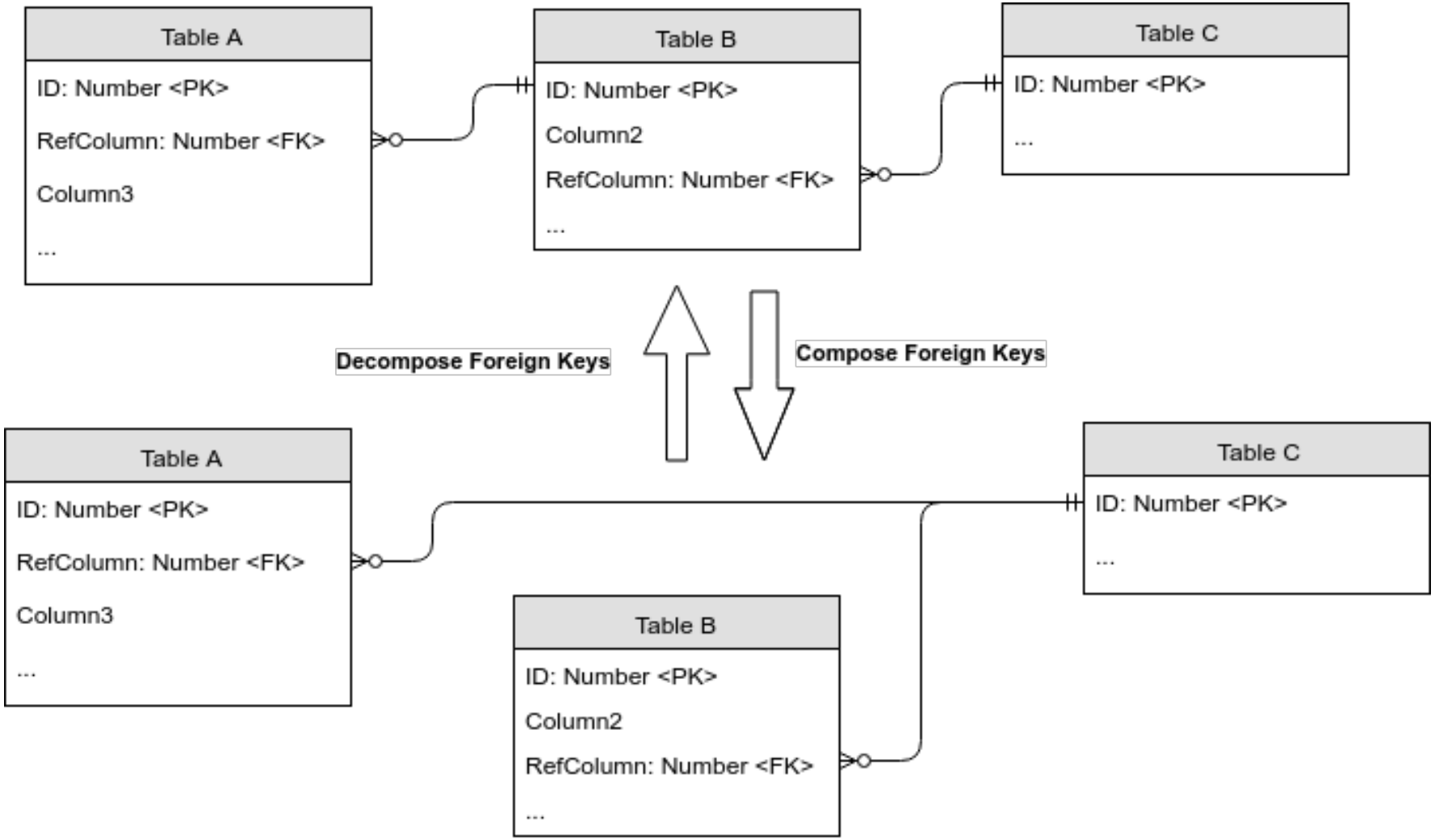
ID	ColumnX	ColumnY	ColumnDesc
1	columnXvalue1	columnYvalue1	A
3	columnXvalue3	columnYvalue3	A

ID	ColumnX	ColumnY	ColumnDesc
2	columnXvalue2	columnYvalue2	B
4	columnXvalue4	columnYvalue4	B

Transcode Foreing Key



Compose Foreign Keys



Offline Batch-Migration

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

Advantages	**Disadvantages**
Easy to implement	May cause long downtimes on large data sets

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.

Advantages	**Disadvantages**
Long downtimes avoided	Higher complexity due to aggregation
	Temporary trigger required

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.

Advantages	**Disadvantages**
Long downtimes avoided	Higher complexity due to merge
Concurrent migration	Temporary trigger for logging required

Design Goals

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!

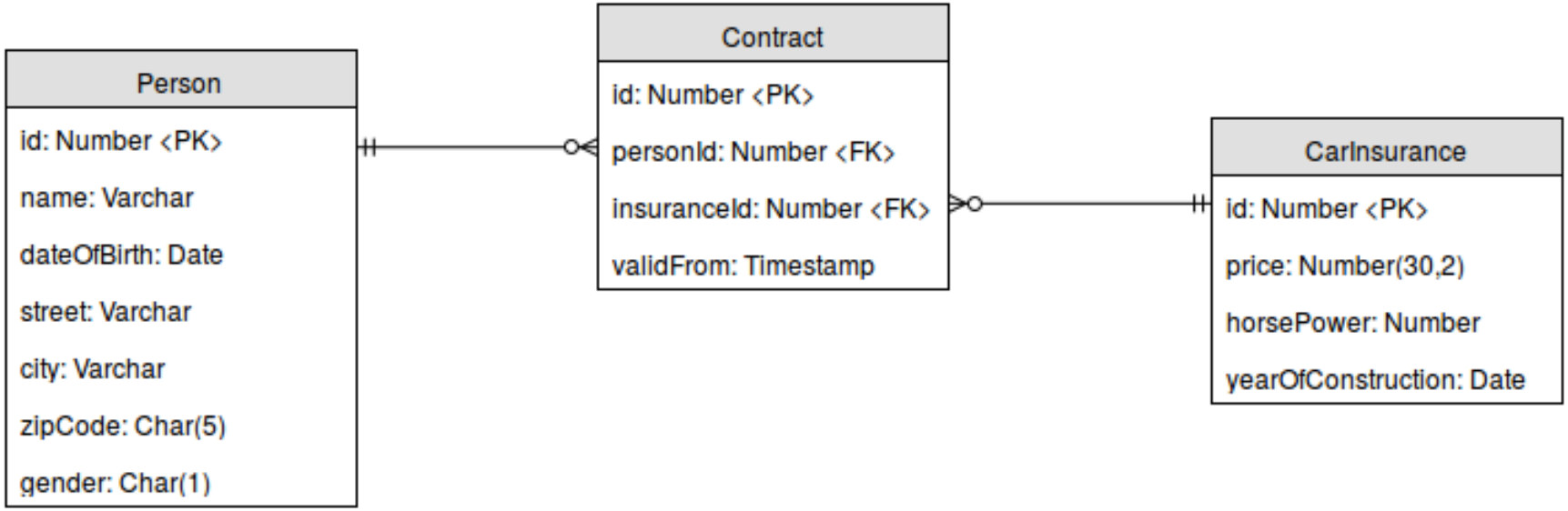
Design Goals

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

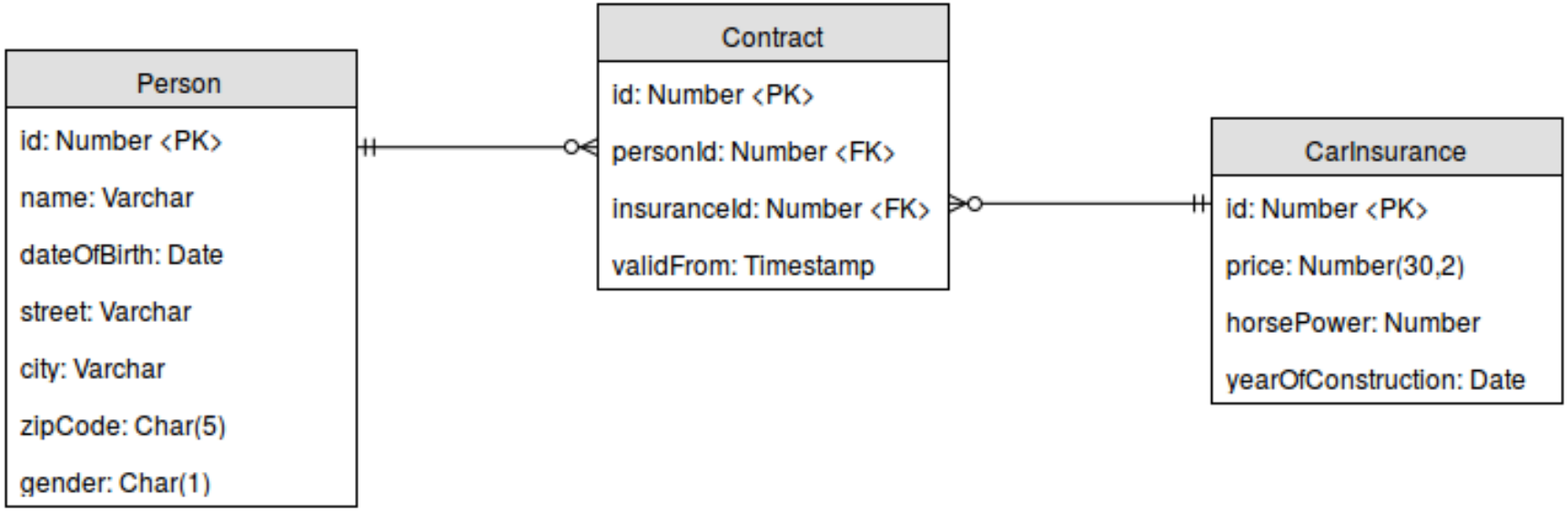
DELTA:

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

Starting point

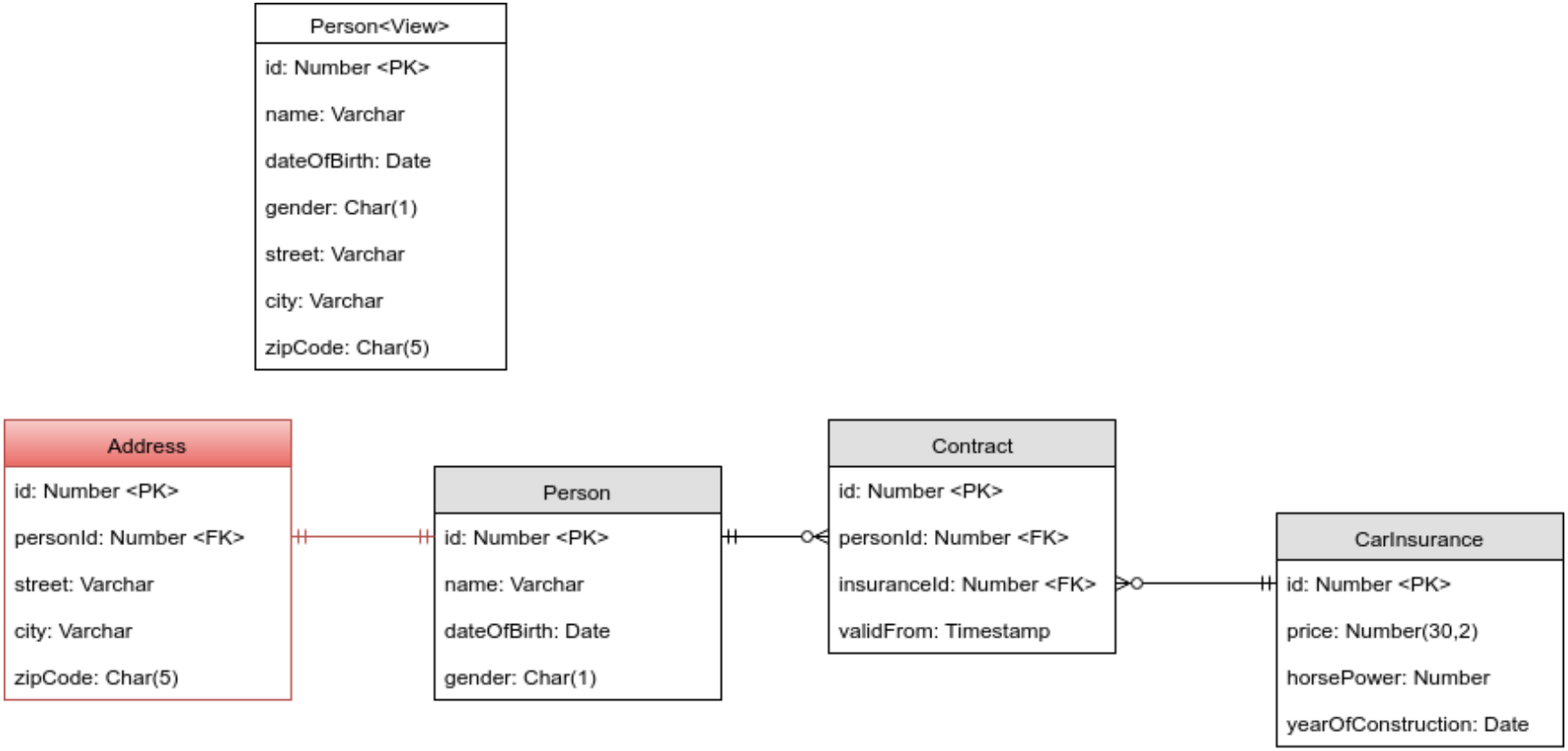


Starting point

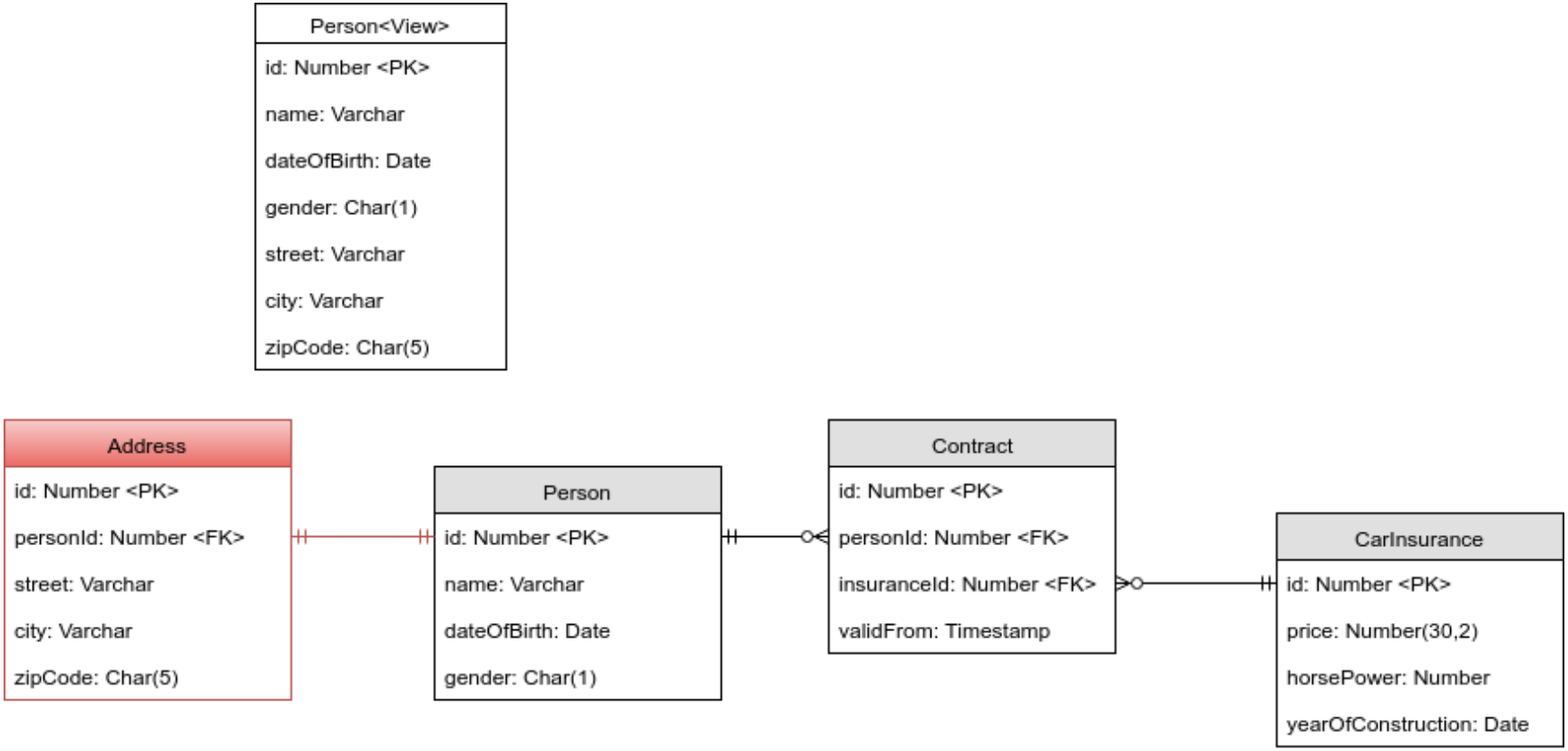


- Address is part of *Person*

Step 1: Spin-Off the Address

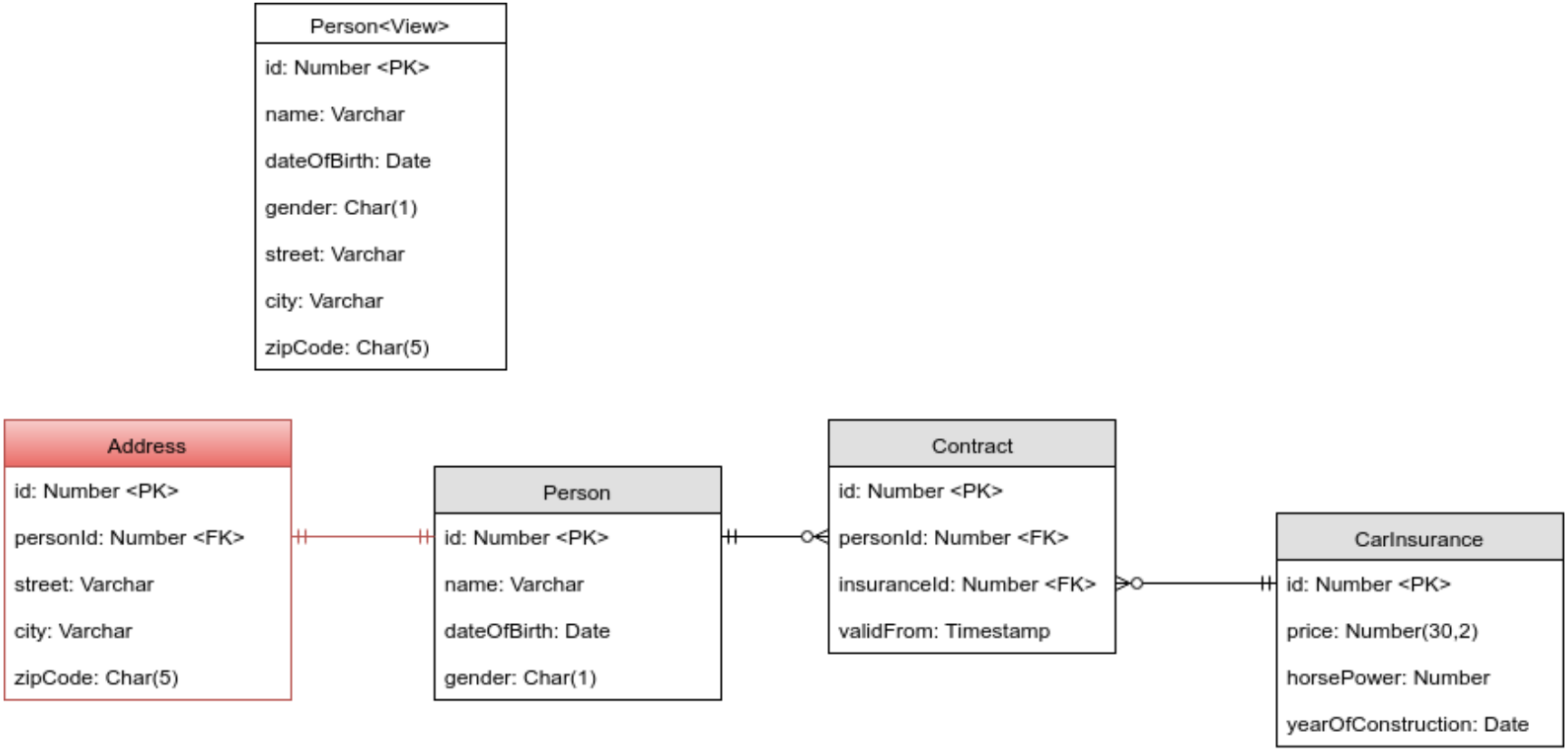


Step 1: Spin-Off the Address



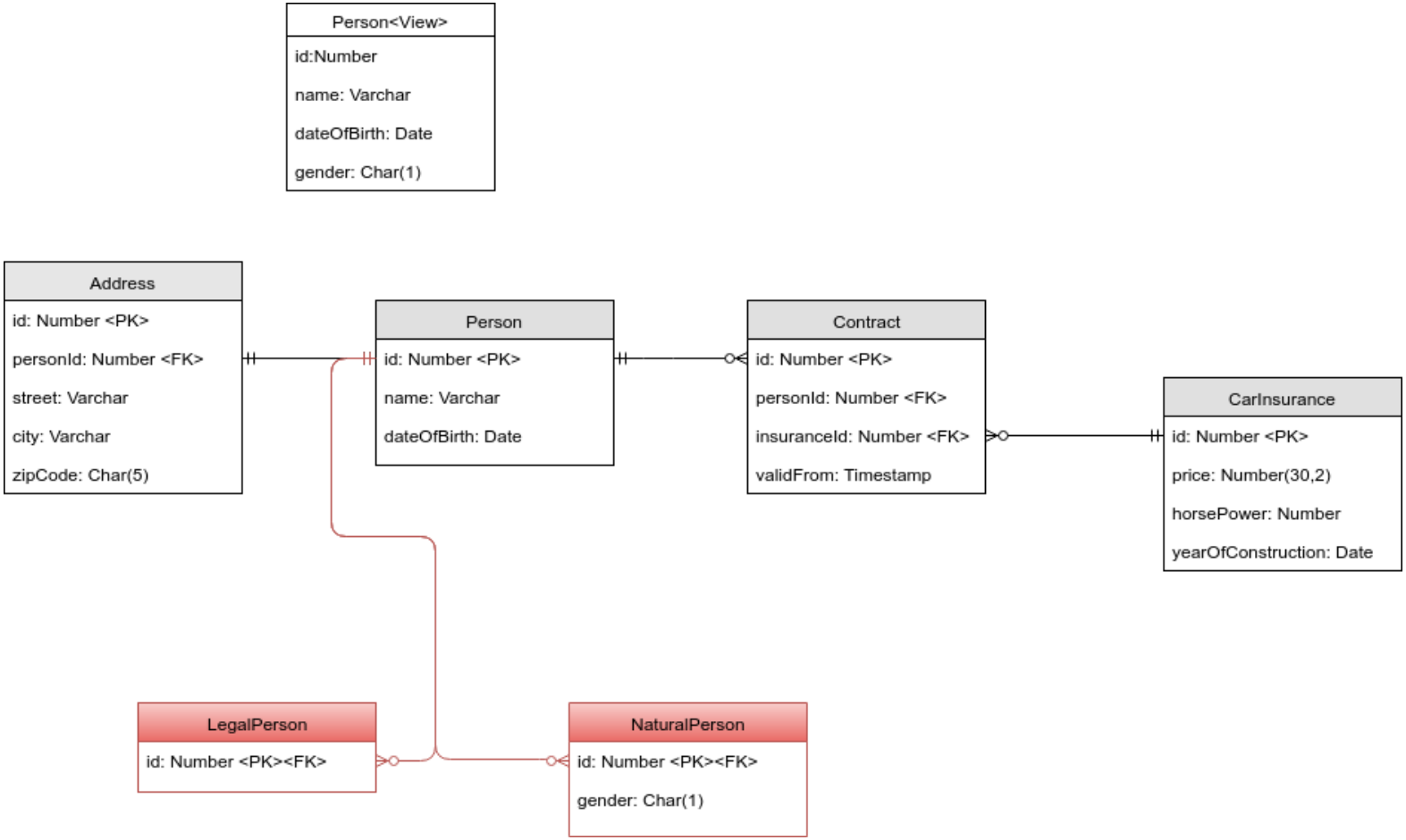
- transition phase

Step 1: Spin-Off the Address

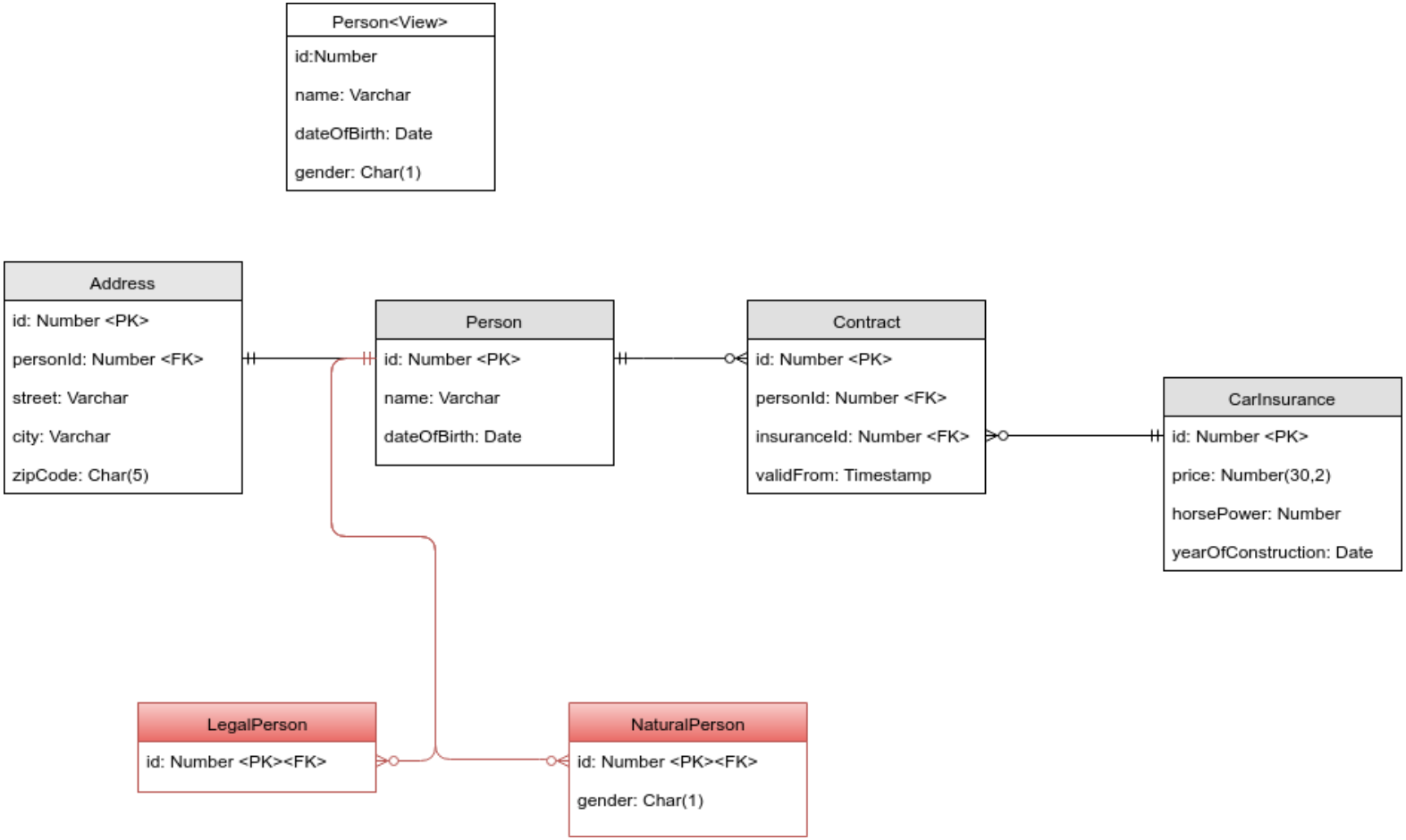


- transition phase
- no distinction between *legal* and *natural* persons

Step 2: Split in natural and legal persons

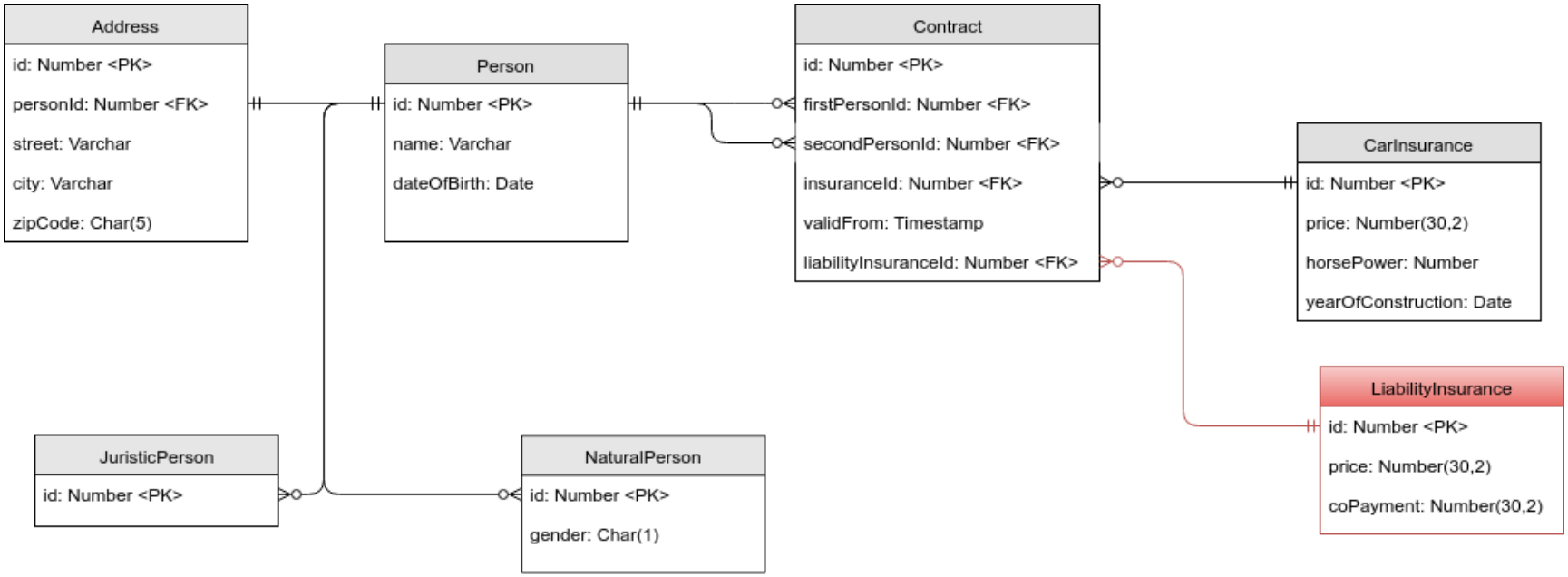


Step 2: Split in natural and legal persons

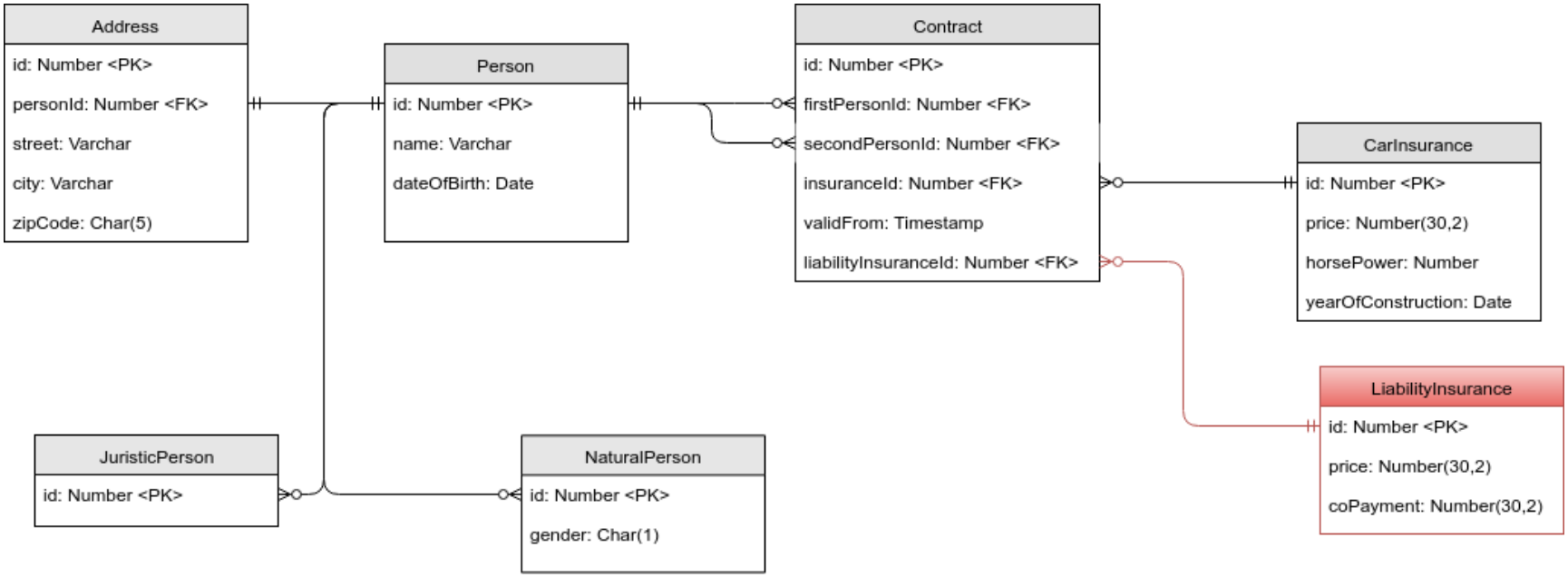


- no support for multiple tenants

Step 3: Implement Multi-Tenancy

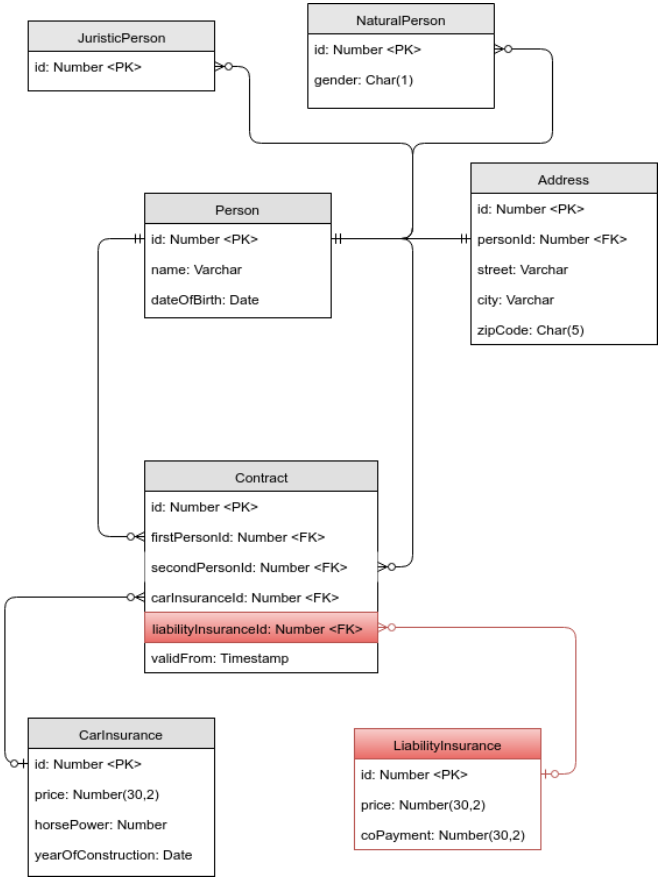


Step 3: Implement Multi-Tenancy

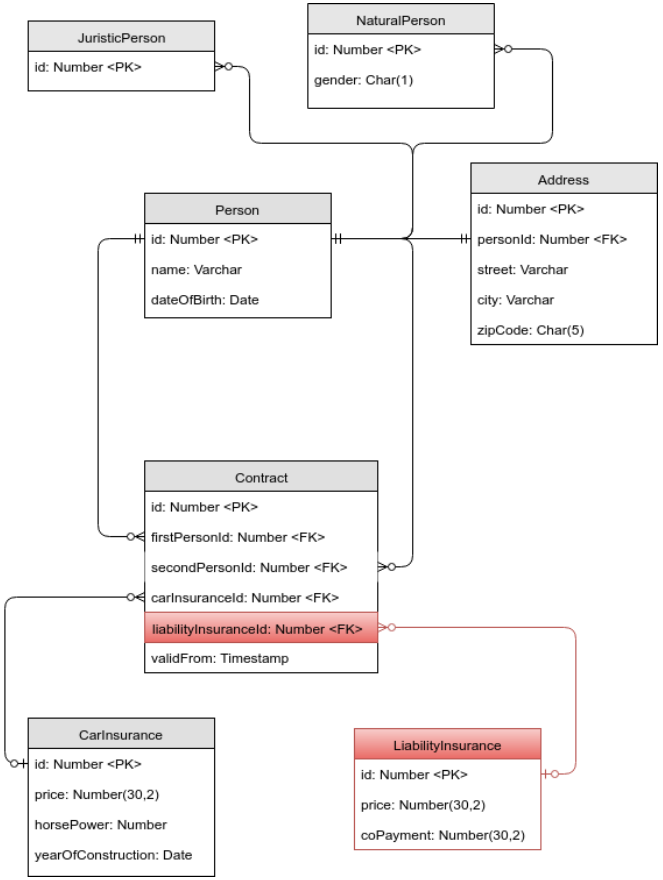


- transition phase

Company Merger



Company Merger



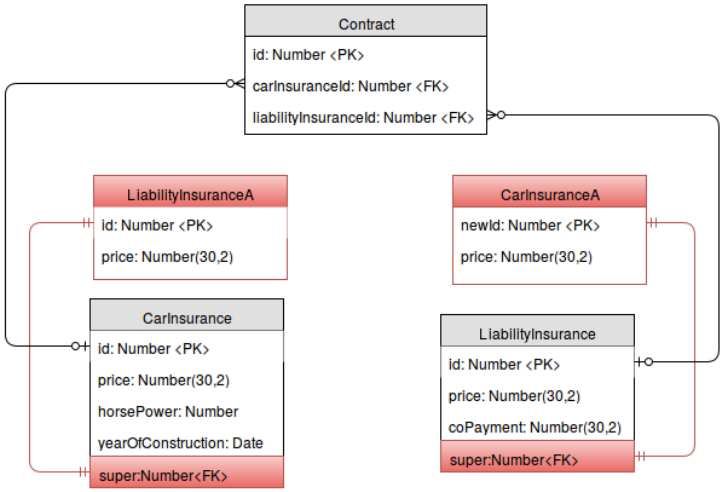
- bad design: `insuranceId` and `liabilityInsuranceId` are disjoint

Schritt 4: Concrete-Table-Inheritance → Class-Table-Inheritance

=>

Step1:

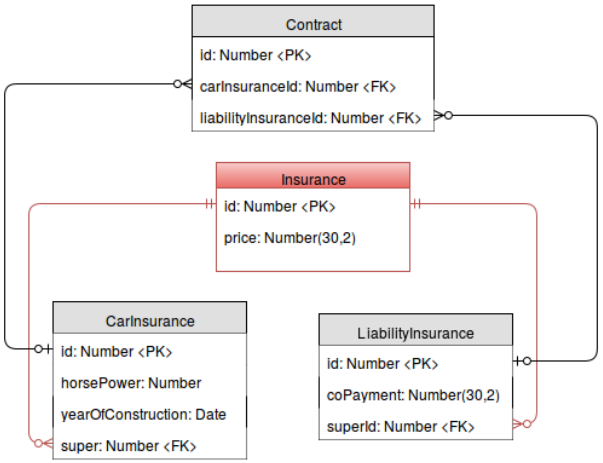
- Spin Off Empty Table
- Move Column



=>

Step2:

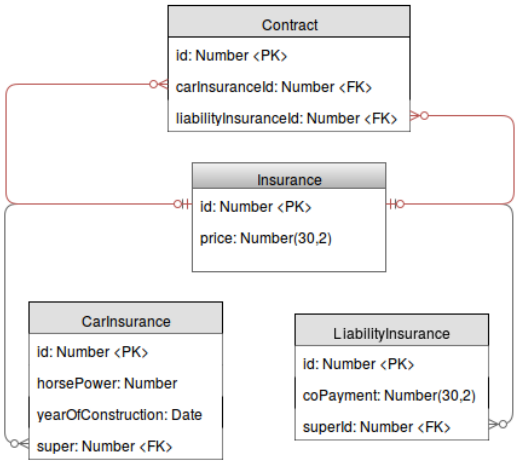
- Union Tables



=>

Step3:

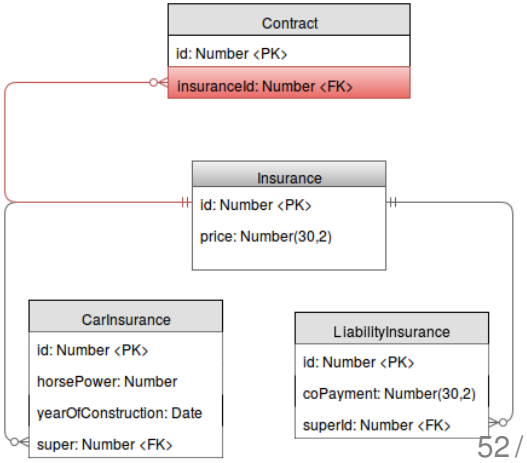
- Change Foreign Key



=>

Step4:

- 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 $\rightarrow$ 1:* $\rightarrow$ *:*
- Split entities: Single-Table-Inheritance $\rightarrow$ 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

SQL Prompt VisualStudio-PlugIn, supports renaming and split-table

ApexSQL Only MS-SQL-Server, supports renaming, split-table and introduce association-table.

Flyway Version control and management of migration scripts, no implemented refactorings

Liquibase Same as Flyway but with SQL-Abstraction and some built-in-refactorings (rename, split-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.
- M. Pereira et al. *Evolution of databases using petri nets* [6]. More general approach on how to check dependencies between refactorings.

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 refactoring

THANK YOU FOR YOUR ATTENTION!

[1] Scott W Ambler and Pramod J Sadalage
Refactoring databases: Evolutionary database design. Pearson Education, 2006.

[2] Martin Fowler
Refactoring : improving the design of existing code. Addison-Wesley, 1999.

[3] Michael Löwe
Refactoring information systems - association folding and unfolding. FHDW Hannover, 2013.

[4] Ioannis Skoulis, Panos Vassiliadis, Apostolos V. Zarras
Growing up with stability: How open-source relational databases evolve. Information Systems,53:363-385, 2015

[5] Carlo A. Curino, Hyun J. Moon, Carlo Zaniolo
Graceful database schema evolution: the prism workbench. Proceeding of the VLDB Endowment, 1:761-772, 2008

[6] Marcia Beatriz Carvalho Pereira, Jorge Rady de Almeida Junior, Jose Reinaldo Silva
Evolution of databases using petri nets. Anais de XIX Congresso Brasileiro de Automatica, CBA 2012