Simple UML Representation of Relational Database Schema

Drazen Brdjanin, Slavko Maric
University of Banja Luka, Bosnia & Herzegovina
Presentation outline

• Motivation
• (Un)suitability of the standard UML for RDBS representation
• UML-profile based approaches for RDBS representation
• Disadvantages of the UML-profile based approaches
• Our approach for representation of (complex) keys
• Example of forward database engineering
• Conclusion and future work
Motivation

- A part of long-term research devoted to model-driven database design

- There is no single and standardized approach to (UML-based) RDBS representation

- The standard UML notation is not fully adapted for RDBS representation – RDBS represents a model containing some specific concepts which cannot be represented by the standard UML notation?! – UML profile is required?!

- Our research question: Is it possible to represent RDBS by standard UML?
(Un)suitability of the standard UML

- The UML-based RDBS representation has been the subject of research since the beginning of UML development
Representation of PK by “isID” property

It is possible to represent a PRIMARY KEY by setting isID=true for all attributes belonging to the primary key.

However:

- **Some modeling tools**, including some open source development platforms, **still** do not support the recent UML specifications and **do not allow representation of the primary key by applying the isID property**.
- **How to represent foreign keys and other RDBS specific concepts?**
UML-profile based approaches

- The UML-based RDBS representation has been the subject of research since the beginning of UML development

Typical UML stereotypes for RDBS representation (Naiburg & Maksimchuk)
Disadvantages of the existing approaches

Representation of complex keys

<table>
<thead>
<tr>
<th>Example1</th>
<th>Example2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;pk&gt;&gt; att1 : type</td>
<td>&lt;&lt;pk&gt;&gt; att1 : type {PK_SEG=1}</td>
</tr>
<tr>
<td>&lt;&lt;pk&gt;&gt; att2 : type</td>
<td>&lt;&lt;pk&gt;&gt; att2 : type {PK_SEG=2}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Order of the key segments is not visible in the diagram, but hidden (represented by additional property of the given stereotype)

Order of the key segments is visible in the diagram - depicted by tagged values (e.g. PK_SEG)

Even more difficulties with the existing approaches?

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;pk&gt;&gt; att1 : type1 {PK_SEG=1}</td>
</tr>
<tr>
<td>&lt;&lt;pk,fk&gt;&gt; att2 : type2 {PK_SEG=2} {FK=1} {FK_SEG=1}</td>
</tr>
<tr>
<td>&lt;&lt;fk&gt;&gt; att3 : type3 {FK=2} {FK_SEG=1}</td>
</tr>
<tr>
<td>&lt;&lt;fk&gt;&gt; att4 : type4 {FK=2} {FK_SEG=2}</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

D. Brdjanin and S. Maric: Simple UML Representation of Relational Database Schema
Representation of keys by class operations

- UML possesses another inherent (but not explicit) mechanism that also provides the possibility for very simple and efficient representation of keys as attribute sequences.

KEY is ordered sequence of columns.

\[ \text{key}(p_1, \ldots, p_k) \]

Operation parameters constitute the sequence.

\[ \text{operation}(\text{op}_1:\text{type}, \ldots, \text{op}_k:\text{type}) \]

KEYS can be modeled by OPERATIONS!

Key segments \( p_1, \ldots, p_k \) are represented by corresponding operation parameters \( \text{op}_1, \ldots, \text{op}_k \).

**Primary key:**

\[ \text{PK}(p_1, \ldots, p_k) \]

**Foreign key:**

\[ \text{FK}(f_1, \ldots, f_k) \]

**Table**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a1 : type1</td>
<td></td>
</tr>
<tr>
<td>a2 : type2</td>
<td></td>
</tr>
<tr>
<td>PK(a1:type1, a2:type2)</td>
<td></td>
</tr>
</tbody>
</table>
Representation of keys by class operations

**Example**

Existing approaches

<table>
<thead>
<tr>
<th>&lt;&lt;table&gt;&gt;</th>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;pk&gt;&gt; a1:t1 {PK_SEG=1}</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;pk,fk&gt;&gt; a2:t2 {PK_SEG=2} {FK=1} {FK_SEG=1}</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;fk&gt;&gt; a3:t3 {FK=2} {FK_SEG=1}</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;fk&gt;&gt; a4:t4 {FK=2} {FK_SEG=2}</td>
<td></td>
</tr>
</tbody>
</table>

...  

Our solution based on the operation signature semantics!

<table>
<thead>
<tr>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1 : t1</td>
</tr>
<tr>
<td>a2 : t2</td>
</tr>
<tr>
<td>a3 : t3</td>
</tr>
<tr>
<td>a4 : t4</td>
</tr>
</tbody>
</table>

| PK(a1:t1, a2:t2) |
| FK_1(a2:t2) |
| FK_2(a3:t3, a4:t4) |

Direct advantages of this approach:

- RDBS is represented by **standard UML notation**;
- There is **no need to define and apply the specific profile**;
- **Modeling is easier and faster** than using the specialized notation;
- **Visualization is better** (without specific stereotypes, order of operation parameters represents the key’s segments, order of the key segments is visible in the diagram).

---

Representation of keys by class operations

How to deal with multiple foreign keys?

Example 1:

```
T_12
| a1_T1 : t1 |
| a2_T2 : t2 |
PK(a1_T1:t1,a2_T2:t2)
FK_T1(a1_T1:t1)
FK_T2(a2_T2:t2)
```

Example 2:

```
T_1
| a2_T2_role1 : t2 |
| a2_T2_role2 : t2 |
FK_T2_role1(a2_T2_role1:t2)
FK_T2_role2(a2_T2_role2:t2)
```

T_1
```
| a2_T2_role1 : t2 |
| a2_T2_role2 : t2 |
FK_T2_role1(a2_T2_role1:t2)
FK_T2_role2(a2_T2_role2:t2)
```

T_2
```
| a2 : t2 |
PK(a2:t2)
```
Specification of referential actions

How to specify referential actions for foreign keys?

• UML allows specification of operation constraints.
• A set of constraints (ownedRule) can be specified for each operation.

• In our case, the appropriate constraint specifying the referential integrity actions, is to be defined for each operation representing the foreign key.
• Although constraints can be specified by using the OCL, they can be specified in another way, as well.

• The easiest way is to directly specify the DDL statement part, e.g.

  ON DELETE RESTRICT ON UPDATE CASCADE
Example of forward database engineering

ECLIPSE-TOPCASED plug-in

D. Brdjanin and S. Maric: Simple UML Representation of Relational Database Schema

Z. S. Pavkovic and D. Brdjanin, “A UML-based approach to forward engineering of SQLite database”, ZINC 2016 (Best Paper)
Example of forward database engineering
Example of forward database engineering

```plaintext
[comment encoding = UTF-8 ]
[module generate('http://www.eclipse.org/uml2/3.0.0/UML')]
[template public generateElement(aPackage : Package)]
[comment @main/]
[file (aPackage.name.concat('.ddl'), false, 'UTF-8')]
CREATE SCHEMA [aPackage.name/];
[for (aClass:Class | aPackage.ownedElement) after('\n')]
CREATE TABLE [aPackage.name.concat('.').concat(aClass.name)/]
(for (aProperty:Property | aClass.ownedAttribute) separator('\\n') after('\\n'))
(aProperty.name/) [aProperty.type.name/][if (aProperty.lower>0) NOT NULL][if]
(for (aOperation:Operation | aClass.ownedOperation->
    select(o | o.name.startsWith('FK') or o.name.equalsIgnoreCase('PK')) )
    separator('\\n', 'after('\\n'))
    [if (aOperation.name.equalsIgnoreCase('PK'))]
    PRIMARY KEY [(for op:Parameter | aOperation.ownedParameter)
    separator(', ', ')][op.name]()[for][if]
    [if (aOperation.name.startsWith('FK'))]
    CONSTRAINT [aOperation.name/]
    FOREIGN KEY [(for op:Parameter | aOperation.ownedParameter)
    separator(', ', ')][op.name]()[for]
    REFERENCES [(for aDependency:Dependency | aPackage.ownedElement)
    [if (aDependency.name.equalsIgnoreCase(aOperation.name))]
    [aPackage.name.concat('.')][aDependency.supplier.name][if][for]
    [for (r:Constraint | aOperation.ownedRule)] [r.name][if][for]
    );
[file] [if]
[template]
```
Example of forward database engineering

```sql
CREATE SCHEMA RDB_schema;
CREATE TABLE RDB_schema.Assessment
(
    student_id text NOT NULL,
    exam_course_id text NOT NULL,
    exam_date date NOT NULL,
    grade int NOT NULL,
    PRIMARY KEY (student_id, exam_course_id, exam_date),
    CONSTRAINT FK_Student
        FOREIGN KEY (student_id)
        REFERENCES RDB_schema.Student ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT FK_Exam
        FOREIGN KEY (exam_course_id, exam_date)
        REFERENCES RDB_schema.Exam ON UPDATE CASCADE ON DELETE RESTRICT
);
...
Conclusion

- **UML has a** **built-in mechanism for representation of primary keys**, but **there are no concepts for representation of other RDBS concepts** (foreign keys, indices, etc).

- In this presentation we presented a simple **representation of composite keys by using class operations**. The main idea of the proposed approach is based on the fact that the **standardized order of operation parameters can be used to represent the order of key segments** (primary, foreign, alternate).

- This approach has several direct advantages compared to the existing approaches:
  - RDBS is represented by standard UML;
  - there is no need to define and apply the specific profile;
  - modeling of RDBS is easier and faster than using the specialized notation;
  - visualization of RDBS is better;
  - forward database engineering is easier and more efficient.

- In the future work we will focus on:
  - further analysis of the suitability of standard UML notation to represent other important RDBS-related concepts (indices, views, queries, triggers, etc.),
  - integrating the implemented tool in the ADBdesign tool for automated MDD of RDBS.
Simple UML Representation of
Relational Database Schema

Thank You!