UML/OCL für die Integritätssicherung in Datenbankanwendungen

Zwischenbericht Diplom

Sten Löcher

Inhalt des Vortrages

- Motivation der Arbeit
- Konzept für die Integritätssicherung
- Musterkatalog → Probleme/Lösungen
- Tool-Gesamtkonzept
- Noch zu erledigende Aufgaben

I. Motivation der Arbeit

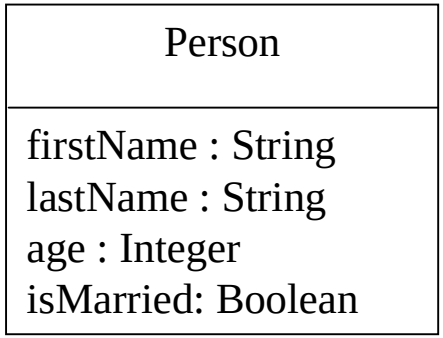
Grundlage:

Verwendung von UML bei der objektorientierten Modellierung persistenter Daten, die in relationalen Datenbanken gespeichert werden sollen

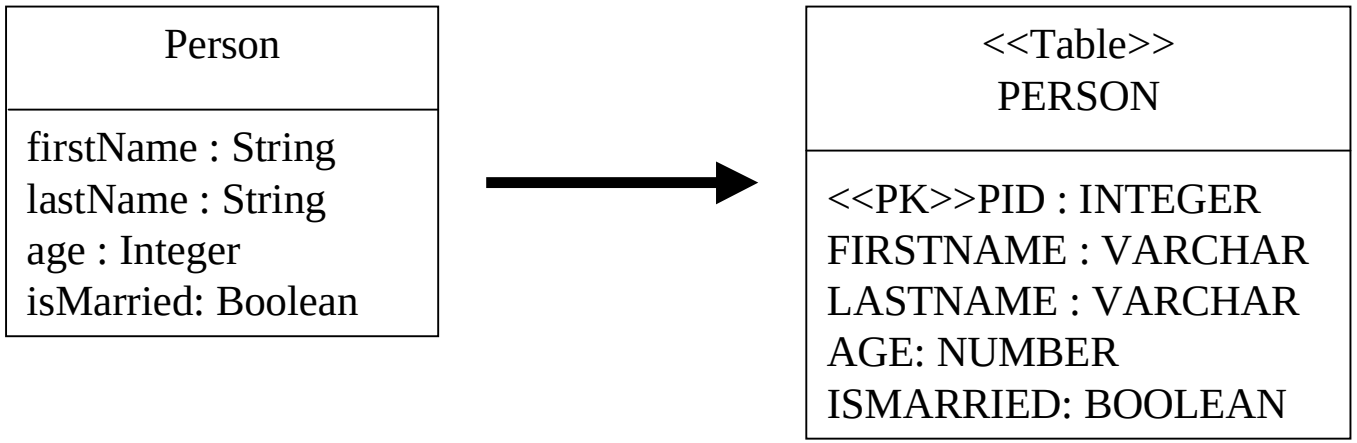
Ziel:

Überprüfung von Integritätsbedingungen, die mit Hilfe von OCL formuliert wurden, mit den Mechanismen der Datenbanken.

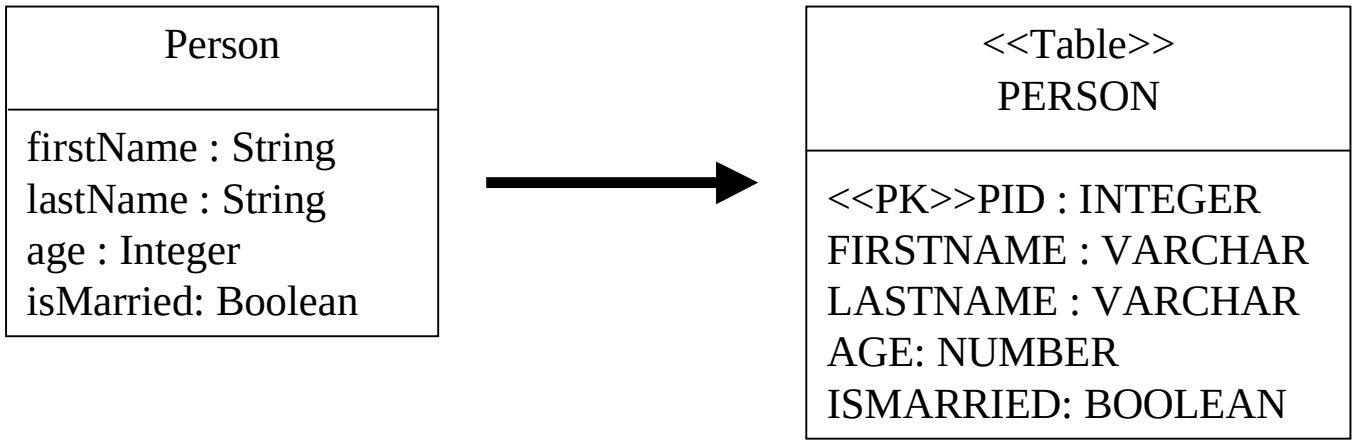
Beispiel:


Person
firstName : String
lastName : String
age : Integer
isMarried: Boolean

Beispiel:

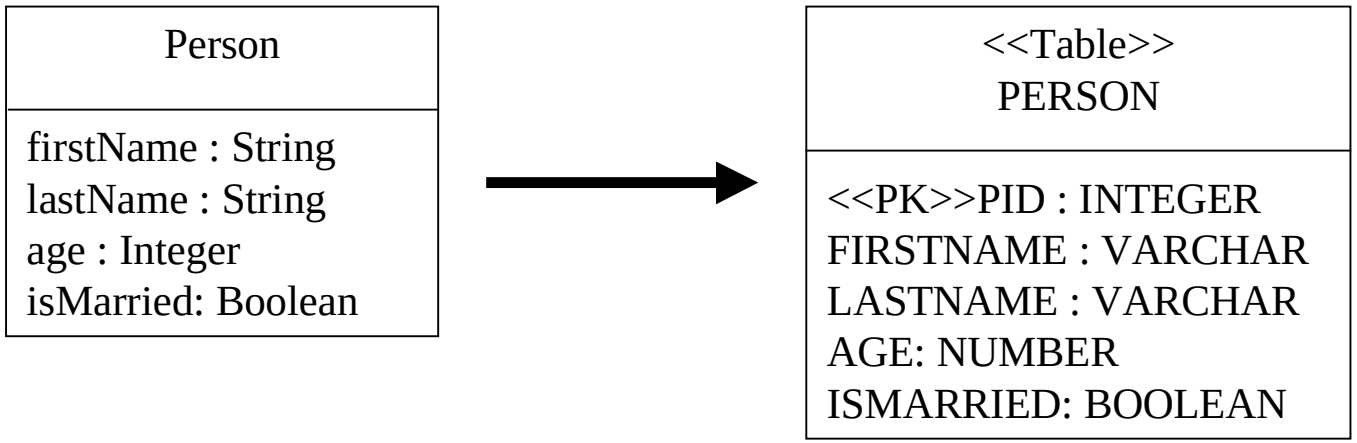

Person
firstName : String
lastName : String
age : Integer
isMarried: Boolean
<<Table>>
PERSON
<<PK>>PID : INTEGER
FIRSTNAME : VARCHAR
LASTNAME : VARCHAR
AGE: NUMBER
ISMARRIED: BOOLEAN

Beispiel:



context Person inv ageOfMarriage:
(isMarried = true) implies (age>=18)

Beispiel:



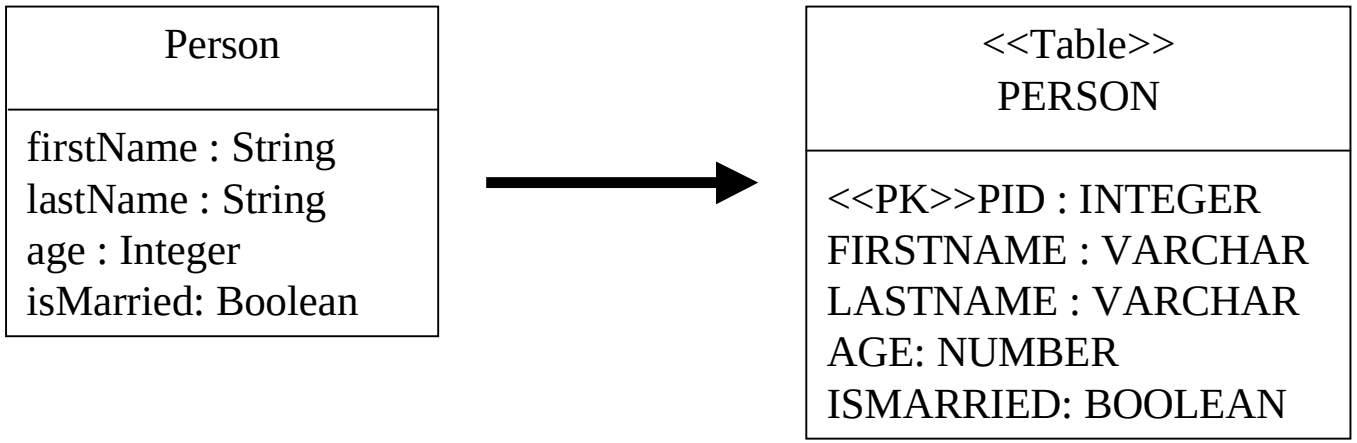
context Person inv ageOfMarriage:
(isMarried = true) implies (age>=18)

?

Ansätze:

- Übersetzung in prozeduralen Code für Datenbanken wie z.B. Transact SQL oder PL/SQL
  - Abbildung sehr komplex
  - Abbildung nur für ausgewähltes DBMS
- Übersetzung in deklarativen Code
  - strukturell relativ einfach
  - existierender Standard → SQL92

Beispiel:



context Person inv ageOfMarriage:
(isMarried = true) implies (age>=18)

```
create assertion AGEOFMARRIAGE
check ( not exists
(select * from PERSON
 where not (not (isMarried = true)
                or (age>=18))))
```

nächster Schritt:

- Implementation eines OCL-to-SQL-Compilers
  - Großer Beleg
  - Verwendung des OCL-Compilers aus DA Frank Finger
  - Erzeugung von SQL92-Code + Test auf Oracle 8i
- Ergebnis:
  - Prototyp zur Erzeugung von SQL92-Assertions aus OCL-Invarianten

Problem:

- Code nicht lauffähig auf Oracle 8i (und anderen DBMS)
- Gründe:
  - SQL92 definiert drei Level an Funktionalität
  - Oracle implementiert nur Entry-Level vollständig, höhere Levels nur teilweise → unzureichend für unseren Ansatz (Assertions Bestandteil der Full Level Spezifikation)
  - andere DBMS zeigen ähnliche Merkmale

Frage:

Ist der deklarative Ansatz unrealistisch bzw. überhaupt in absehbarer Zeit anwendbar ?

II. Konzept für die Integritätssicherung

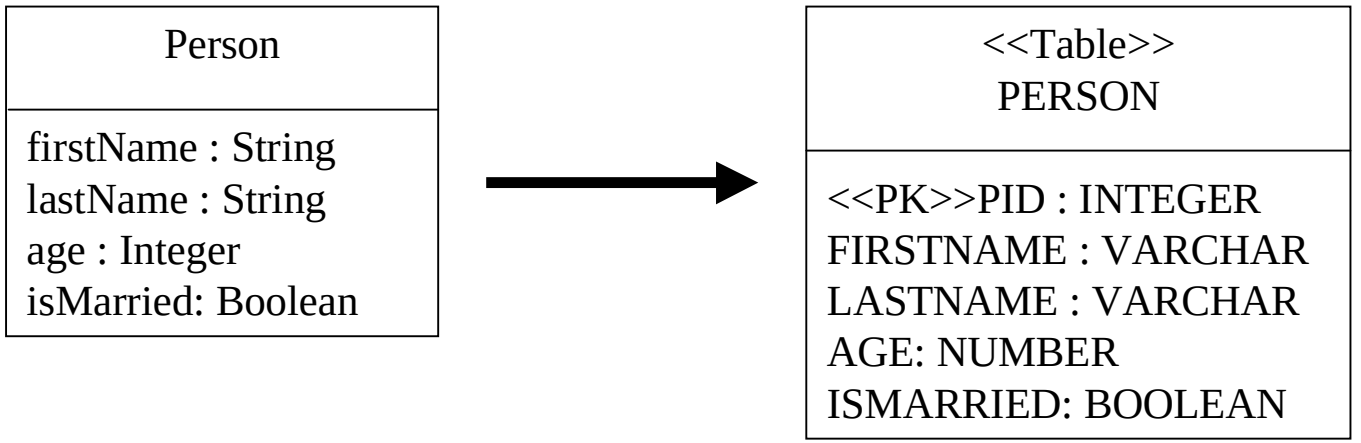
- Analyse der Anforderungen von DB-Anwendern
  - Anforderungen unterschiedlichster Art
  - Zeitpunkt der Integritätsprüfung
    - sofortige Prüfung nach Datenmanipulation
    - unabhängige Prüfung zu bestimmten Zeitpunkten
  - Auslöser der Überprüfung
    - extern: Anwendungsprogamme, Middleware, ...
    - intern: automatisch durch Datenbankmechanismen
  - Was passiert mit fehlerhaften Daten ?
    - werden vom DBS abgewiesen
    - werden behandelt und trotzdem gespeichert

View-Konzept

- Grundgedanke:

  Für jede OCL-Invariante wird ein View generiert, der alle in Bezug auf diese Invariante ungültigen Tupel der Datenbank ermittelt.

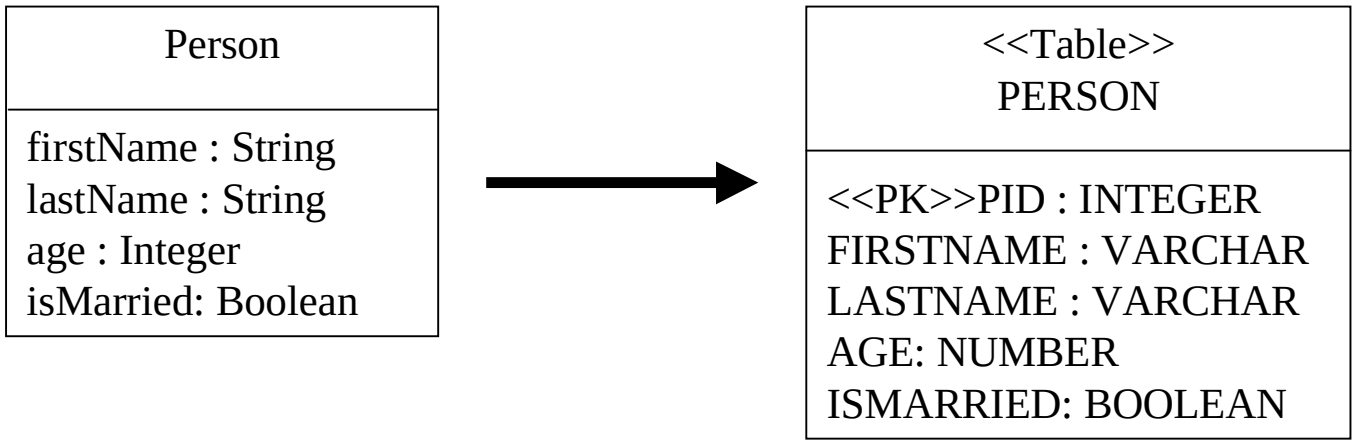
Beispiel:



context Person inv ageOfMarriage:
(isMarried = true) implies (age>=18)

```
create assertion AGEOFMARRIAGE
check ( not exists
(select * from PERSON
 where not (not (isMarried = true)
                or (age>=18))))
```

Beispiel:



context Person inv ageOfMarriage:
(isMarried = true) implies (age>=18)

```
create view AGEOFMARRIAGE as
(select * from PERSON
 where not (not (isMarried = true)
                or (age>=18)))
```

Verwendungszweck der Views:

1. Anwendungsgesteuerte Evaluation der Views:
   - Zeitpunkt der Integritätskontrolle von Anwendung festgelegt → Kontrolle über Performanceprobleme
   - z.B. objektrelationale Middleware mit eigenem Transaktionsmodell → Prüfung vor commit
   - Monitor-Anwendungen für Datenbanken

Beispiel:

```
ResultSet rs= theStatement.executeQuery(
        select nvl(count(*), 0) from AGEOFMARRIAGE
);
if (rs.next().getInt(0) > 0) {
        // integrity error handling
}
```

2. Assertion Ersatzmechanismus:

- Evaluation der Views nach jeder kritischen Datenmanipulationsoperation mit Hilfe von Triggern
- Rollback der aktuellen Transaktion bei Anzahl der Tupel > 0 und Fehlermeldung an aufrufende Anwendung

Beispiel:

```
create trigger TR_AGEOFMARRIAGE
after insert or update or delete on PERSON
begin
   if (select nvl(count(*), 0) from AGEOFMARRIAGE ) > 0 then
          raise_application_error(“Integrity error !“, 20900);
  end if;
end;
```

3. ECA-Trigger-Template

- Paradigma der aktiven Datenbanken (ADBMS)
- **E**vent-**C**ondition-**A**ction-Rules
- Event: eine kritische Datenmanipulationsoperation
- Condition: es existieren Tupel, die die OCL-Invariante verletzen
- Action: muß entsprechend implementiert werden → z.B. Behandlung der kritischen Tupel

Beispiel:

```
create trigger TR_AGEOFMARRIAGE
after insert or update or delete on PERSON
begin
   if (select nvl(count(*), 0) from AGEOFMARRIAGE ) > 0 then
            // todo: add action code here
  end if;
end;
```

III. Musterkatalog → Probleme/Lösungen

- durch GB erste Erfahrungen mit dem Musterkatalog gesammelt
- dadurch Schwächen bzw. Kritikpunkte entdeckt
- Überarbeitung notwendig, um Anwendbarkeit zu erhöhen und Fehler zu beseitigen

Struktur des Musterkataloges:

- Grundidee:

  Menge von Mustern, wobei jedes einzelne Muster die Übersetzung eines OCL-Sprachkonzeptes erklärt.

- insgesamt acht Muster:
  - OCL INVARIANT
  - BASIC TYPE
  - CLASS AND ATTRIBUTE
  - NAVIGATION
  - ...

- z.B. beschreibt BASIC TYPE die Abbildung der Grunddatentypen (Real, Integer, String, Boolean) und Operationen über diesen auf SQL → Entsprechungen + äquivalente Ausdrücke
- oder NAVIGATION beschreibt die Abbildung von Navigationsausdrücken →SQL Anfrage mit geschachtelten Unteranfragen

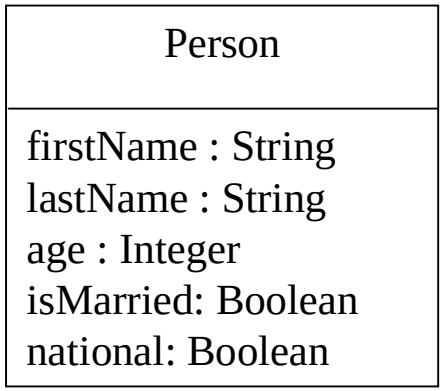
Hauptprobleme des Musterkataloges:

- keine Beachtung verschiedener objektrelationaler Abbildungsvarianten
- Zugriff auf Daten- und Metadaten des Anwendungsmodells unzureichend geklärt
- Behandlung von OCL-Sequences bei der Abbildung auf SQL

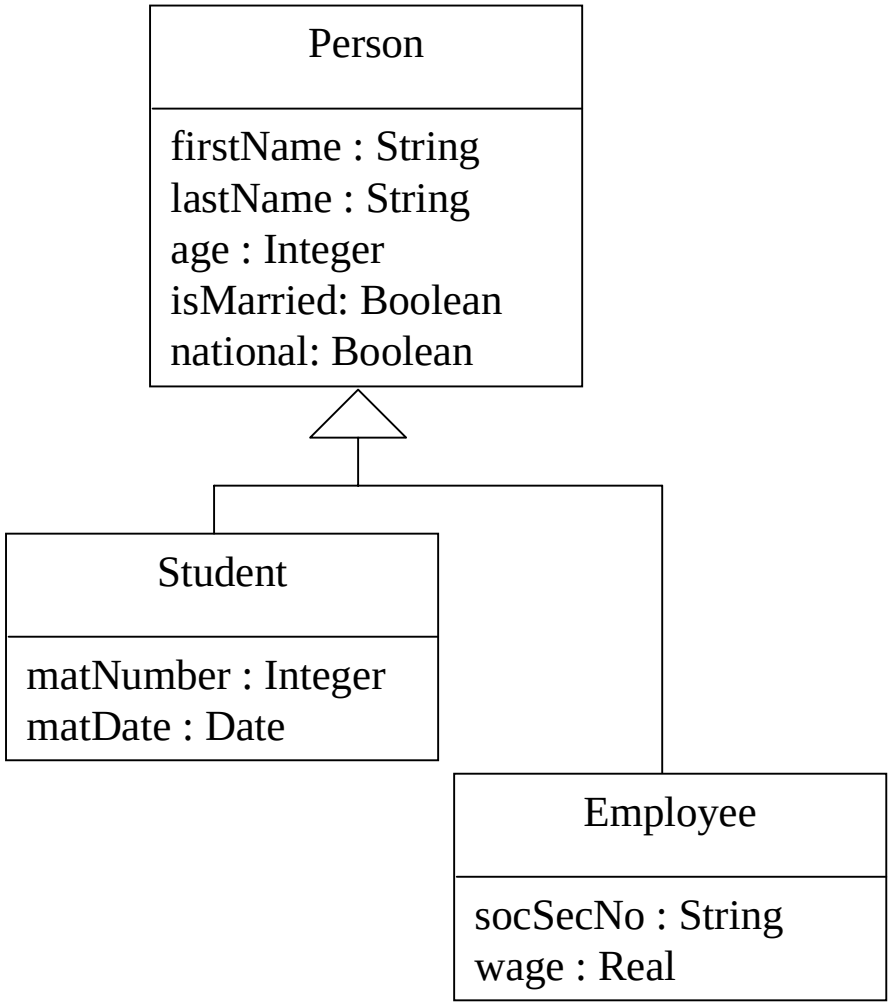
objektrelationale Abbildungsvarianten:

- Klassen können auf beliebig viele Tabellen abgebildet werden
- komplexe Attribute auf beliebige Anzahl von Spalten und separate Tabellen
- Abbildung von Assoziationen verschiedenartig möglich

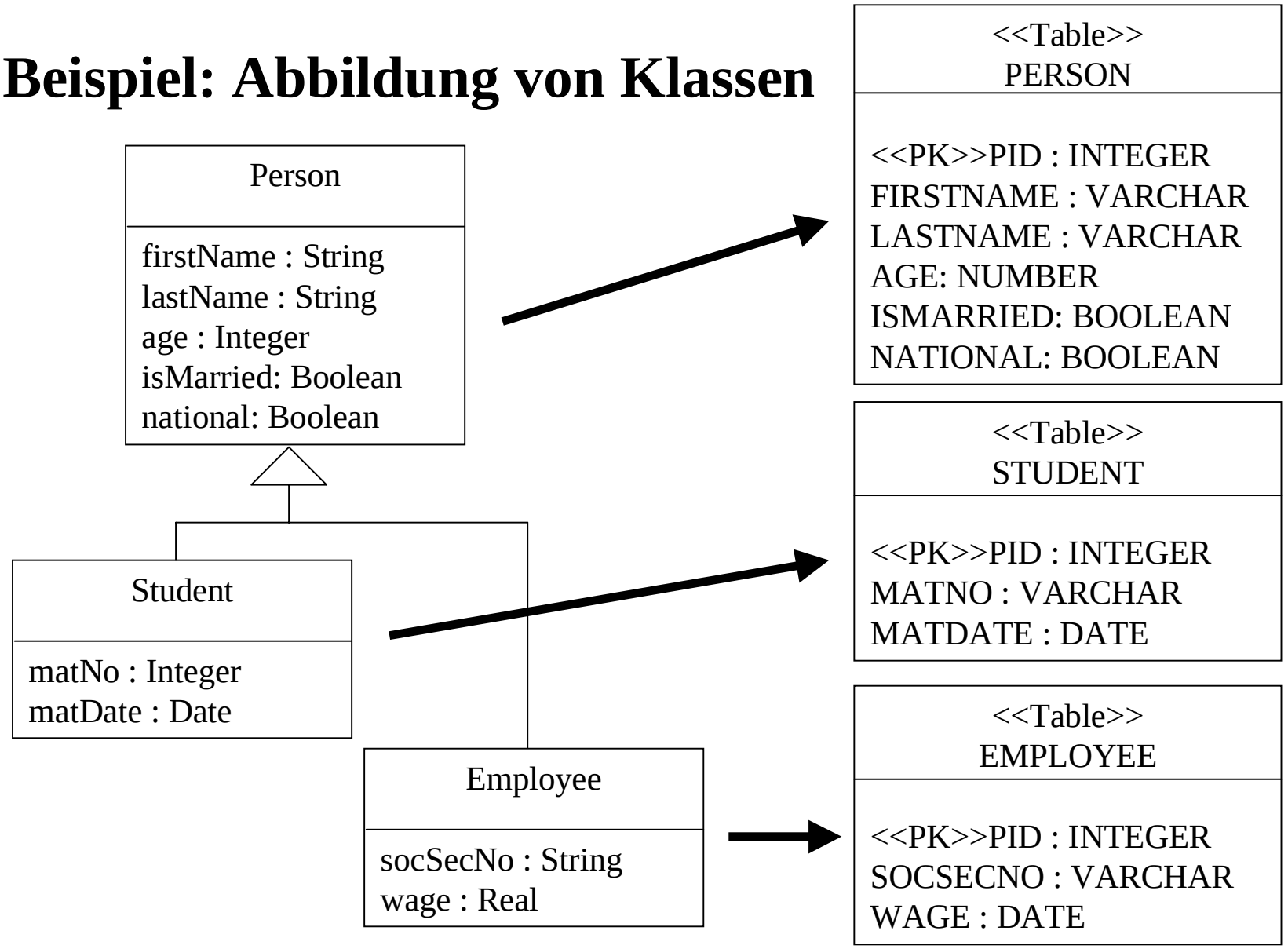
Beispiel: Abbildung von Klassen



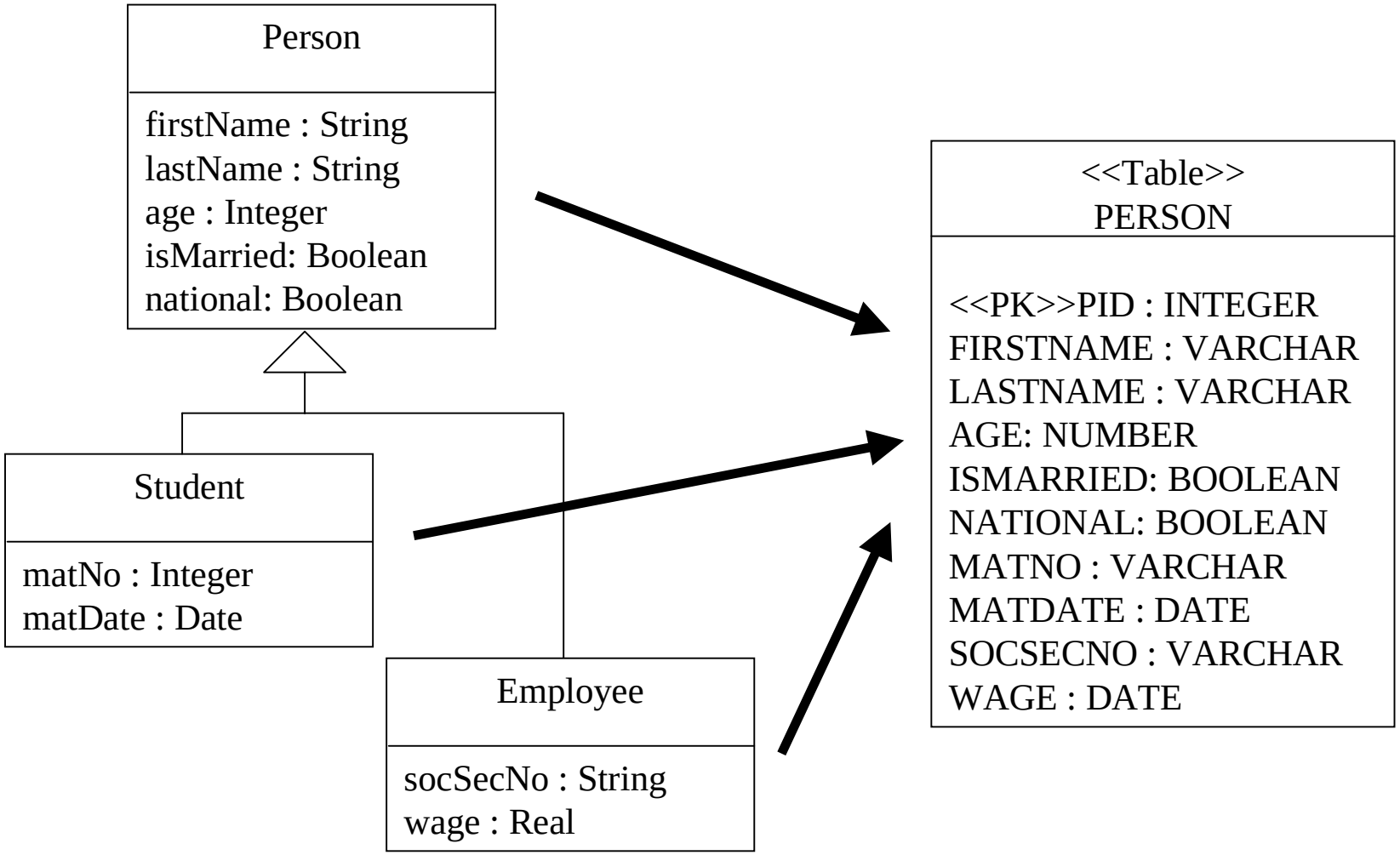
Beispiel: Abbildung von Klassen



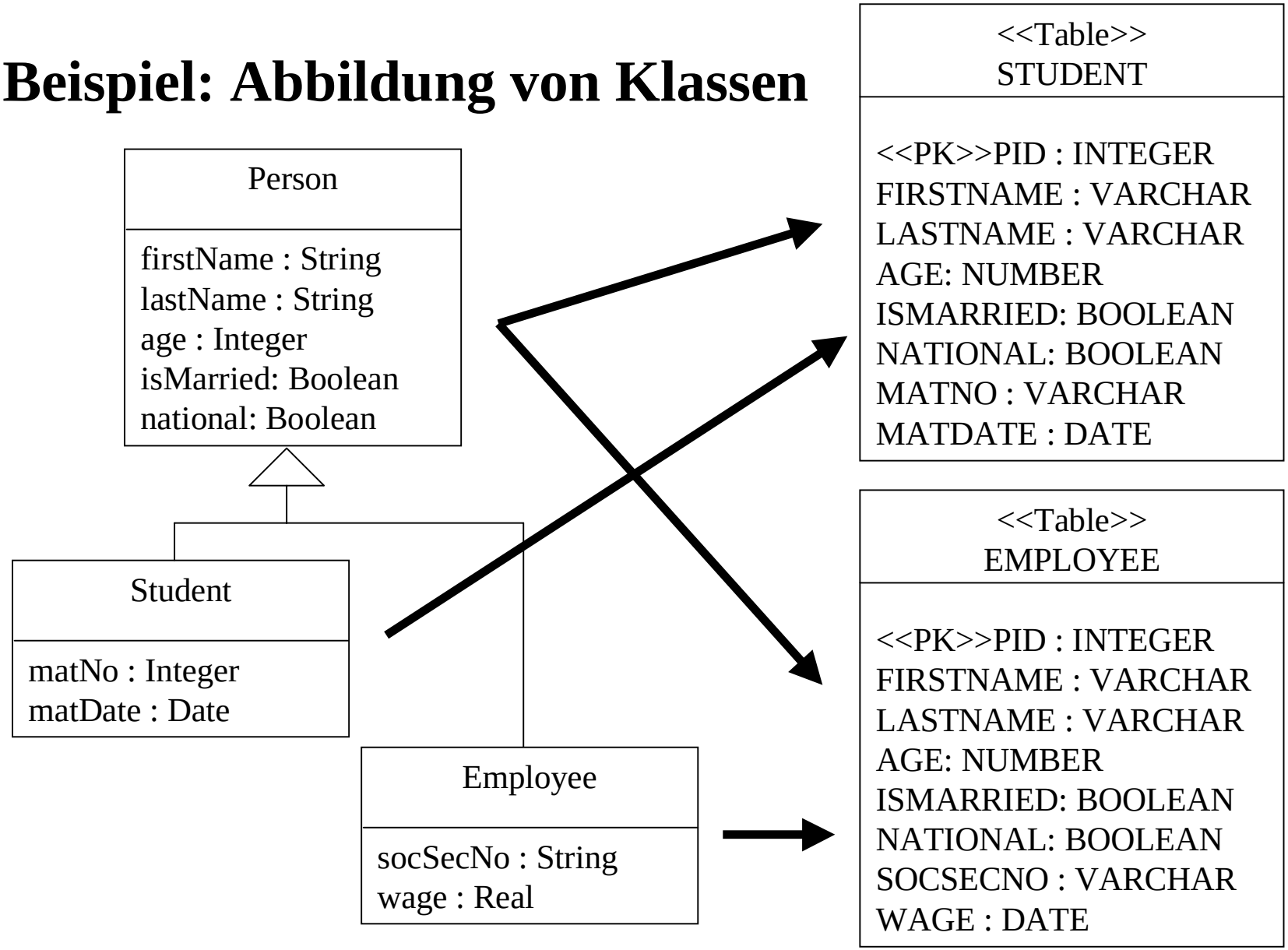
Beispiel: Abbildung von Klassen



Beispiel: Abbildung von Klassen



Beispiel: Abbildung von Klassen



Auswirkung auf Übersetzung OCL-SQL:

context Employee inv bsp :
(national = false) implies (wage>100000)

horizontale Unterteilung

```
create view BSP as
(select * from EMPLOYEE SELF
 where not (not (NATIONAL =false ) or
                 (WAGE > 100000))
```

Auswirkung auf Übersetzung OCL-SQL:

context Employee inv bsp :
(national = false) implies (wage>100000)

eine Tabelle pro Klasse

```
create view BSP as
(select * from EMPLOYEE SELF
 where not (not ((select NATIONAL from PERSON
                        where PID = SELF.PID) = false ) or
                  (WAGE > 100000))
```

Konsequenzen:

- erhöht Komplexität des Codegenerators wegen vielen Sonderfällen
- weitere Probleme z.B. bei vererbten Assoziationen
- Musterkatalog muß die Sonderfälle alle behandeln

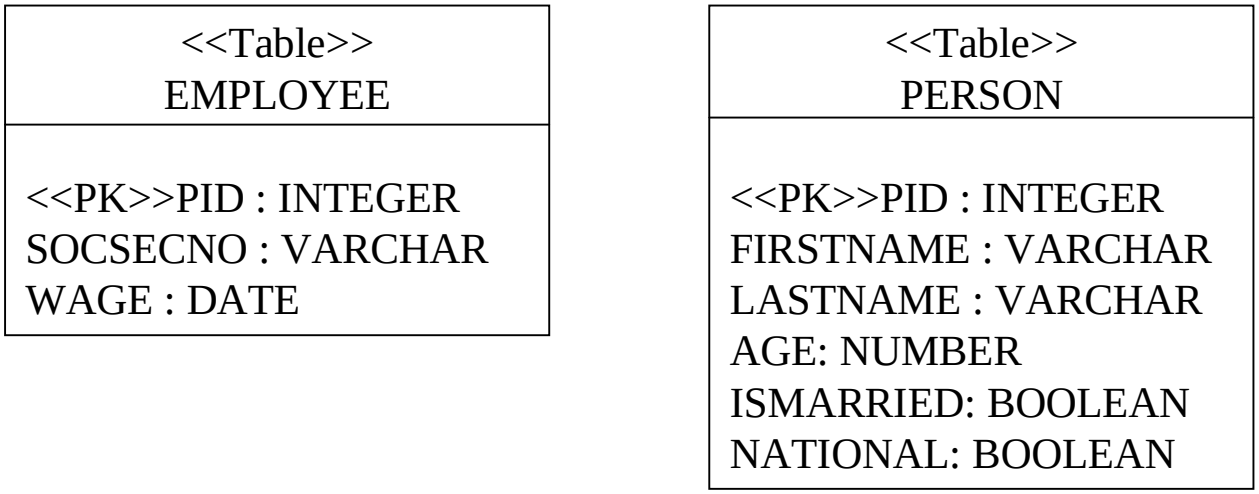
Deshalb:

Suche nach einfacher Lösung, die alle Sonderfälle für die Übersetzung OCL nach SQL transparent macht.

Objekt-View-Konzept:

- Einführung von Objekt-Views als Zwischenschicht zwischen Tabellenschema und Integritätsprüfung.
- Pro Klasse wird ein View erzeugt, der alle Tupel (Objekte) zu einer Klasse liefert.
- Vorteil:
  - Übersetzung von OCL nach SQL bleibt einfach
  - Erzeugung der Objekt-Views ist relativ simpel

Beispiel:



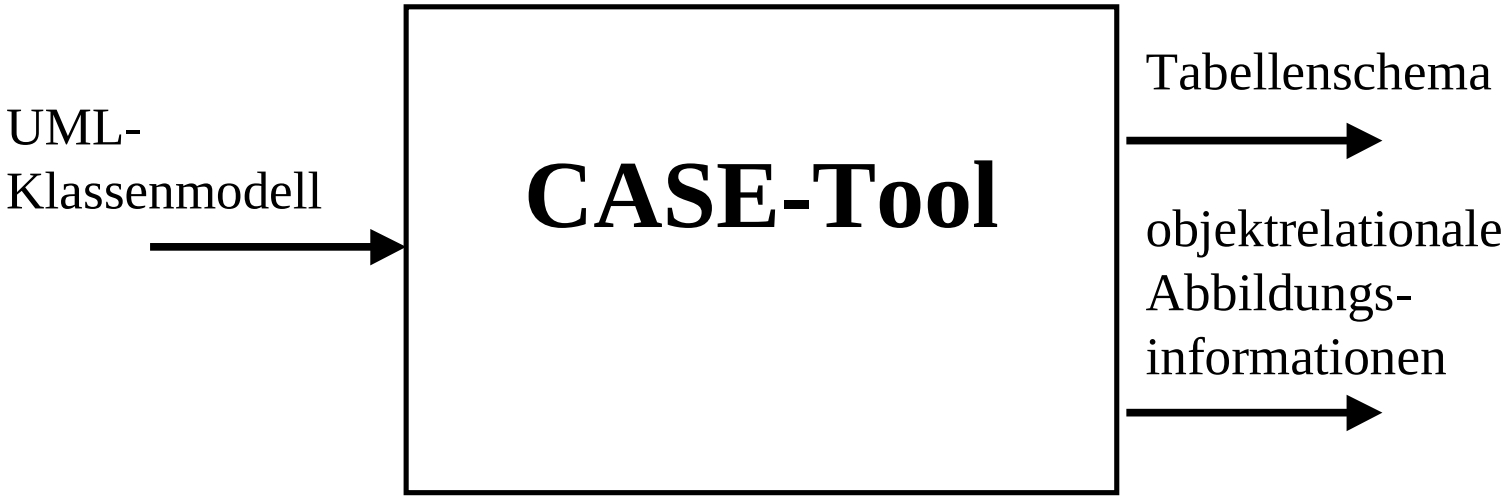

```
create view OV_EMPLOYEE as
(select PERSON.PID, ... , PERSON.NATIONAL, ... , EMPLOYEE.WAGE
 from PERSON, EMPLOYEE
 where PERSON.PID = EMPLOYEE.PID)
```

context Employee inv bsp :
(national = false) implies (wage>100000)

eine Tabelle pro Klasse

```
create view BSP as
(select * from OV_EMPLOYEE
 where not (not (NATIONAL =false ) or
                 (WAGE > 100000))
```

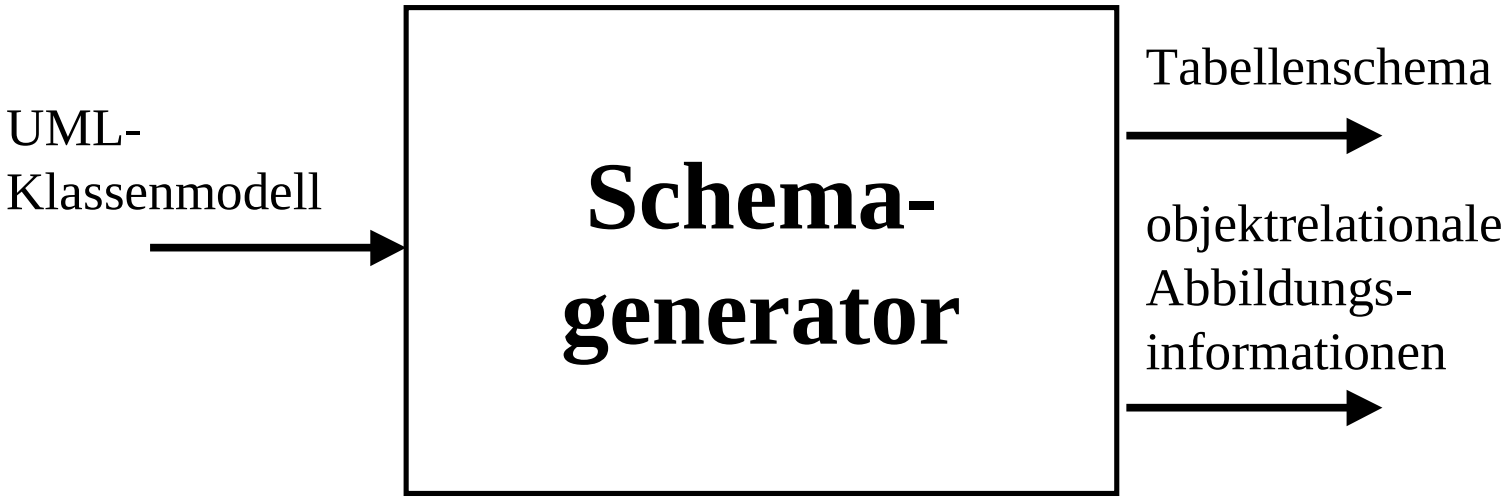
IV. Tool - Gesamtkonzept

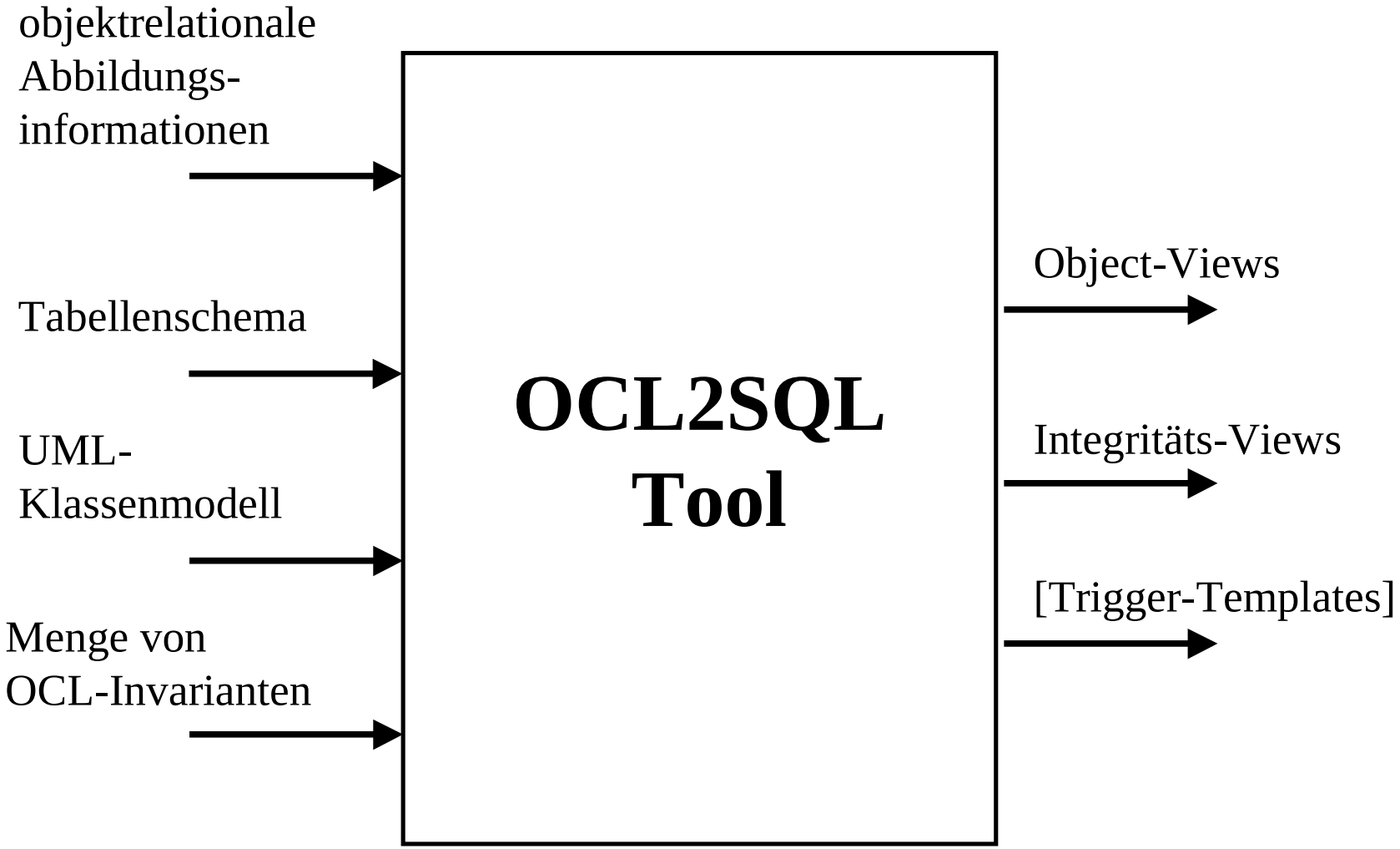


Problem:

- **Qualität der objektrelationalen Abbildung**
- **Schnittstelle zur Abfrage der Abbildungs-informationen**

deshalb:


UML-
Klassenmodell
Schema-
generator
Tabellenschema
objektrelationale
Abbildungs-
informationen


objektrelationale
Abbildungs-
informationen
Tabellenschema
UML-
Klassenmodell
Menge von
OCL-Invarianten
OCL2SQL
Tool
Object-Views
Integritäts-Views
[Trigger-Templates]

V. Noch zu erledigende Aufgaben

- SQL-Codegenerator noch nicht vollständig
- Trigger-Template-Generator
- Überarbeitung des Musterkataloges noch nicht abgeschlossen
- Test des Konzeptes an einer kleinen Beispielanwendung