Faculty of Information Technology

Transformation of Kinds and Subkinds into Relational Databases: A Running Example

Zdeněk Rybola, Robert Pergl

Technical Report TR-FIT-17-01

Czech Technical University in Prague

Technical Report TR-FIT-17-01

Transformation of Kinds and Subkinds into Relational Databases: A Running Example

Zdeněk Rybola, Robert Pergl

January 2017

Contents

Transformation of Kinds and Subkinds into Relational Databases:

A Running Example [*]

Zdeněk Rybola, Robert Pergl
Department of Software Engineering
Faculty of Information Technology
Thákurova 9
160 00 Praha 6
Czech Republic

Abstract

This technical report contains a complete running example documenting the transformation of an OntoUML Platform Independent Model (PIM) into an SQL Implementation Specific Model (ISM). It does not provide the theory, explanations and discussion, the reader is advised to refer to the referenced resources.

Keywords OntoUML, UML, Model-driven Engineering, Relational Databases

[*]This research was partially supported by grant by Student Grant Competition No. SGS16/120/OHK3/1T/18.

1 Goal and Methodology

The goal of this document is to present a running example documenting our approach to the transformation of a PIM in OntoUML into its realization in a relational database. The transformation consists of three steps:

1. A transformation of an OntoUML PIM into a UML PIM: Section 3
2. A transformation of the UML PIM into an RDB PSM: Section 4
    a) by a single table containing columns for all the attributes of the superclass and all the subclasses;
    b) by individual tables for each of the subclasses, containing the columns for the attributes of the respective subclass and the superclass;
    c) by a table for the superclass and individual tables for all the subclasses.
3. A transformation of the RDB PSM into an SQL ISM: Section 5.

It should hold that no information should be lost when transforming from a more abstract model into a more specific one. As OntoUML applies certain constraints based on the OntoUML type used for an entity, these constraints should be carried over to the other models. In our approach, we use OCL to define such constraints in the UML models that cannot be expressed directly in the diagrams.

The SQL examples are coded using the Oracle SQL dialect used by Oracle Database 12c [1].

We do not discuss the theory of the transformation here, nor do we discuss the results, as this has been published in [2] and consecutive papers.

2 Running Example

Our approach to the transformation of Kinds, Subkinds and their generalization sets from the OntoUML PIM into SQL ISM is illustrated on the running example shown in Figure 1. The model shows an excerpt of the domain of transportation company. The company uses various vehicles for transportation of various types of load, ranging from persons to documents to heavy cargo. Therefore, the main entity of such model is the type `Vehicle`. As it is the type defining the identity principle for its instances, it is classified as *Kind*. For each vehicle used by the company, its manufacturer, model and plate number is needed, represented by the respective attributes of the `Vehicle` type.

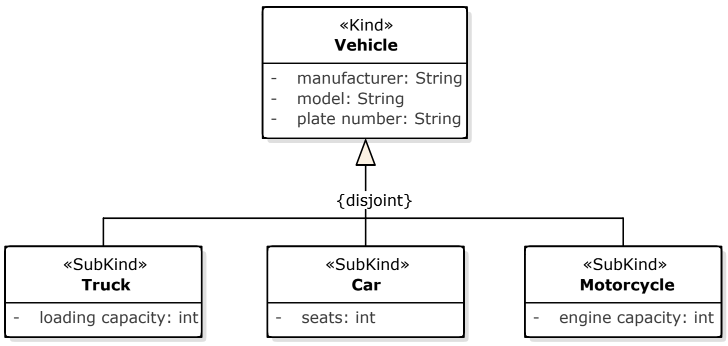


Figure 1: Example of an OntoUML model with Kinds and Subkinds

Furthermore, the company distinguishes three special types of vehicles – trucks, cars and motorcycles – for which additional attributes are important. Each of these types of vehicles is represented by its own type in the OntoUML PIM with the appropriate attributes. As all these types represent the specialization of the general concept of a vehicle, their are classified as *Subkinds* forming a generalization set specializing the type `Vehicle`.

Moreover, as the types of the vehicles are disjoint – clearly, one vehicle cannot be a truck and a motorcycle at the same time – the generalization set is defined `disjoint`. On the other hand, there might be other types of vehicles, for which no special attributes need to be recorded. Therefore, the generalization set is not defined `complete`, but rather left `incomplete`.

3 Transformation of OntoUML PIM into UML PIM

The resulting transformed PIM into UML for the vehicle domain shown in Figure 1 is shown in Figure 2. Each of the $\ll Kind \gg$ and $\ll Subkind \gg$ classes has been transformed into standard UML class.

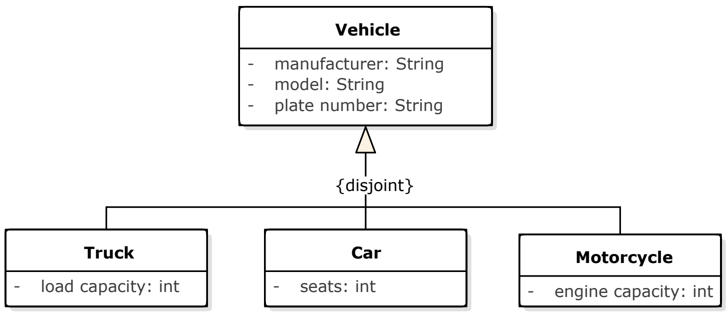
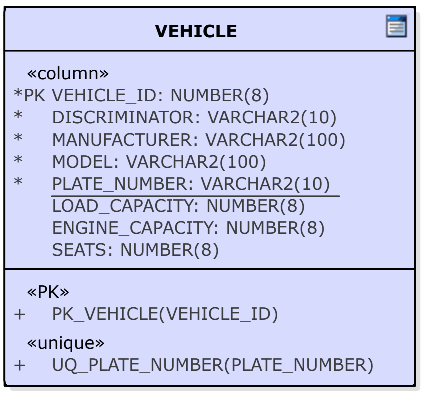


Figure 2: UML PIM with the transformed Kinds and Subkinds

4 Transformation of PIM into PSM

4.1 Single Table

In Figure 3, the RDB PSM is shown with the table `VEHICLE` realizing the transformed generalization set of `Truck`, `Car` and `Motorcycle` classes specializing the `Vehicle` class shown in Figure 2.



```
VEHICLE
«column»
*PK VEHICLE_ID: NUMBER(8)
*   DISCRIMINATOR: VARCHAR2(10)
*   MANUFACTURER: VARCHAR2(100)
*   MODEL: VARCHAR2(100)
*   PLATE_NUMBER: VARCHAR2(10)
    LOAD_CAPACITY: NUMBER(8)
    ENGINE_CAPACITY: NUMBER(8)
    SEATS: NUMBER(8)
«PK»
+   PK_VEHICLE(VEHICLE_ID)
«unique»
+   UQ_PLATE_NUMBER(PLATE_NUMBER)
```

Figure 3: RDB PSM with the generalization set realized by a single table

In Constraint 1, the OCL constraint is defined for the table `VEHICLE` shown in Figure 3, realizing the {`disjoint,incomplete`} generalization set of classes `Truck`, `Car` and `Motorcycle` as shown in Figure 2. In the case of other values of the meta-properties of the generalization set, the OCL constraint would be defined with small differences, as shown in Constraint 2 for a {`disjoint,incomplete`} generalization set, in Constraint 3 for a {`overlapping,complete`} generalization set, and in Constraint 4 for a {`overlapping,incomplete`} generalization set.

Constraint 1 OCL invariant for the {`disjoint,incomplete`} generalization set realized by a single table

```
context v:VEHICLE inv GS_Vehicle_Types:
def Vehicle_Instance: Boolean =
  v.DISCRIMINATOR = 'Vehicle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Truck_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Vehicle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Motorcycle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid

Vehicle_Instance
  OR Truck_Instance
  OR Car_Instance
  OR Motorcycle_Instance
```

Constraint 2 Example of the OCL invariant for a {`disjoint,complete`} generalization set realized by a single table

```
context v:VEHICLE inv GS_Vehicle_Types:
def Truck_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Vehicle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Motorcycle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid

Truck_Instance
  OR Car_Instance
  OR Motorcycle_Instance
```

Constraint 3 Example of the OCL invariant for a {`overlapping,complete`} generalization set realized by a single table

```
context v:VEHICLE inv GS_Vehicle_Types:
def Truck_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Vehicle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Motorcycle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid
def Truck_Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Car'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Truck_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Motorcycle'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid
def Car_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Car_Motorcycle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid
def Truck_Car_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Car_Motorcycle'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid

Truck_Instance
  OR Car_Instance
  OR Motorcycle_Instance
  OR Truck_Car_Instance
  OR Truck_Motorcycle_Instance
  OR Car_Motorcycle_Instance
  OR Truck_Car_Motorcycle_Instance
```

Constraint 4 Example of the OCL invariant for a {`overlapping,incomplete`} generalization set realized by a single table

```
context v:VEHICLE inv GS_Vehicle_Types:
def Vehicle_Instance: Boolean =
  v.DISCRIMINATOR = 'Vehicle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Truck_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Vehicle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Motorcycle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid
def Truck_Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Car'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY = OclVoid
def Truck_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Motorcycle'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS = OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid
def Car_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Car_Motorcycle'
    AND v.LOAD_CAPACITY = OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid
def Truck_Car_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Car_Motorcycle'
    AND v.LOAD_CAPACITY <> OclVoid
    AND v.SEATS <> OclVoid
    AND v.ENGINE_CAPACITY <> OclVoid

Vehicle_Instance
  OR Truck_Instance
  OR Car_Instance
  OR Motorcycle_Instance
  OR Truck_Car_Instance
  OR Truck_Motorcycle_Instance
  OR Car_Motorcycle_Instance
  OR Truck_Car_Motorcycle_Instance
```

In case that values of some of the attributes of the superclass or some of the subclasses should be unique, the realization of such constraint in the RDB PSM is simple. As all data are stored in the same table, the column representing the particular attribute can be simply restricted by the UNIQUE constraint, as shown in Figure 3 where the uniqueness is defined by the `UQ_PLATE_NUMBER` constraint.

4.2 Individual Tables

In Figure 4, the RDB PSM is shown with the transformed generalization set shown in Figure 2. The individual subclasses `Truck`, `Car` and `Motorcycle` are transformed into the individual tables `TRUCK`, `CAR` and `MOTORCYCLE`. The attributes of the class specialized by their generalization set are transformed into columns in each of these tables.

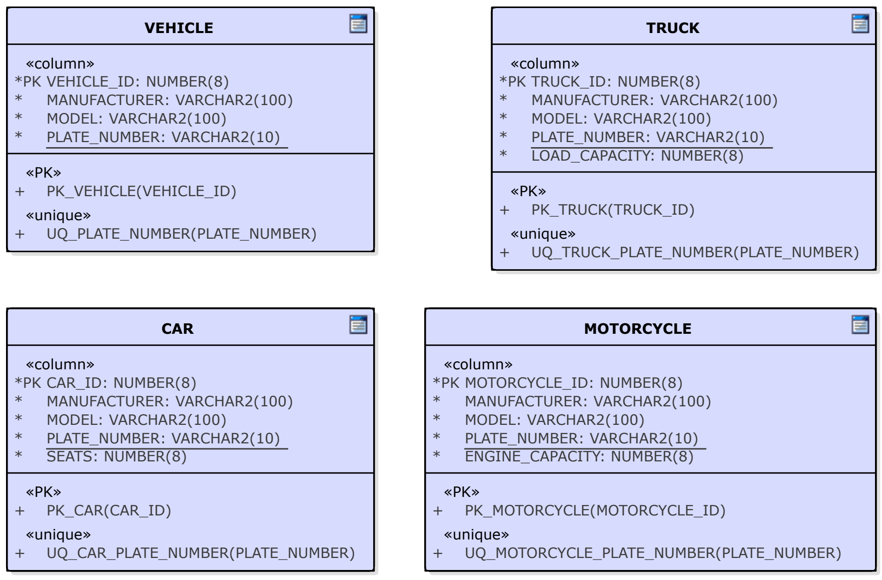

VEHICLE
«column»
*PK VEHICLE_ID: NUMBER(8)
* MANUFACTURER: VARCHAR2(100)
* MODEL: VARCHAR2(100)
* PLATE_NUMBER: VARCHAR2(10)
«PK»
+ PK_VEHICLE(VEHICLE_ID)
«unique»
+ UQ_PLATE_NUMBER(PLATE_NUMBER)
TRUCK
«column»
*PK TRUCK_ID: NUMBER(8)
* MANUFACTURER: VARCHAR2(100)
* MODEL: VARCHAR2(100)
* PLATE_NUMBER: VARCHAR2(10)
* LOAD_CAPACITY: NUMBER(8)
«PK»
+ PK_TRUCK(TRUCK_ID)
«unique»
+ UQ_TRUCK_PLATE_NUMBER(PLATE_NUMBER)
CAR
«column»
*PK CAR_ID: NUMBER(8)
* MANUFACTURER: VARCHAR2(100)
* MODEL: VARCHAR2(100)
* PLATE_NUMBER: VARCHAR2(10)
* SEATS: NUMBER(8)
«PK»
+ PK_CAR(CAR_ID)
«unique»
+ UQ_CAR_PLATE_NUMBER(PLATE_NUMBER)
MOTORCYCLE
«column»
*PK MOTORCYCLE_ID: NUMBER(8)
* MANUFACTURER: VARCHAR2(100)
* MODEL: VARCHAR2(100)
* PLATE_NUMBER: VARCHAR2(10)
* ENGINE_CAPACITY: NUMBER(8)
«PK»
+ PK_MOTORCYCLE(MOTORCYCLE_ID)
«unique»
+ UQ_MOTORCYCLE_PLATE_NUMBER(PLATE_NUMBER)

Figure 4: RDB PSM with the {`disjoint,incomplete`} generalization set realized by individual tables

As the generalization set shown in Figure 2 is {`disjoint,incomplete`}, the table for the instances of the superclass `VEHICLE` is also generated to store instances of just the superclass.

The OCL constraint for the unique values of the `PLATE_NUMBER` column distributed across the tables shown in Figure 4 is shown in Constraint 5. Each of the invariants defines that only such value, which is not existing in the tables representing the other subclasses, is a valid value for the column in that particular table.

Constraint 5 OCL invariants for the distributed unique column `PLATE_NUMBER`

```
context v:VEHICLE inv UQ_VEHICLE_PLATE_NUMBER:
NOT(TRUCK.allInstances()->exists
   (t|t.PLATE_NUMBER = c.PLATE_NUMBER))
 AND
  NOT(CAR.allInstances()->exists
     (c|c.PLATE_NUMBER = t.PLATE_NUMBER))
 AND
  NOT(MOTORCYCLE.allInstances()->exists
     (m|m.PLATE_NUMBER = t.PLATE_NUMBER))

context t:TRUCK inv UQ_TRUCK_PLATE_NUMBER:
NOT(VEHICLE.allInstances()->exists
   (v|v.PLATE_NUMBER = c.PLATE_NUMBER))
 AND
  NOT(CAR.allInstances()->exists
   (c|c.PLATE_NUMBER = t.PLATE_NUMBER))
 AND
  NOT(MOTORCYCLE.allInstances()->exists
     (m|m.PLATE_NUMBER = t.PLATE_NUMBER))

context c:CAR inv UQ_CAR_PLATE_NUMBER:
NOT(VEHICLE.allInstances()->exists
   (v|v.PLATE_NUMBER = c.PLATE_NUMBER))
 AND
  NOT(TRUCK.allInstances()->exists
   (t|t.PLATE_NUMBER = c.PLATE_NUMBER))
 AND
  NOT(MOTORCYCLE.allInstances()->exists
     (m|m.PLATE_NUMBER = c.PLATE_NUMBER))

context m:MOTORCYCLE
   inv UQ_MOTORCYCLE_PLATE_NUMBER:
NOT(VEHICLE.allInstances()->exists
   (v|v.PLATE_NUMBER = c.PLATE_NUMBER))
 AND
  NOT(TRUCK.allInstances()->exists
   (t|t.PLATE_NUMBER = m.PLATE_NUMBER))
 AND
  NOT(CAR.allInstances()->exists
     (c|c.PLATE_NUMBER = m.PLATE_NUMBER))
```

4.3 Related Tables

An example of this realization of the generalization in the RDB PSM for the generalization set shown in Figure 2 is shown in Figure 5. Each of the classes is transformed into a separate table, containing only the columns for the attributes of the respective class and the `ID` column for the unique instance identifier. The generalization relations are realized by the references from the tables of the subclasses to the table of the superclass. These references are realized by the FOREIGN KEY constraints defined on the ID columns with the PRIMARY KEY constraints.

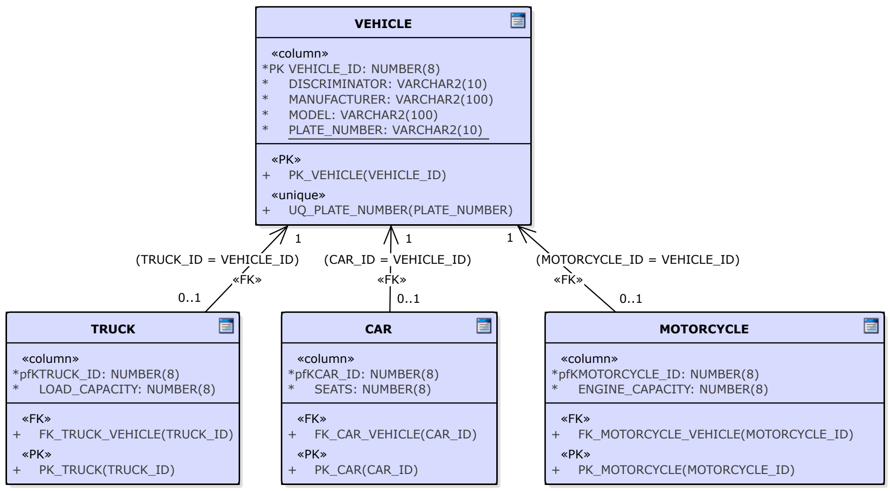

VEHICLE
«column»
*PK VEHICLE_ID: NUMBER(8)
* DISCRIMINATOR: VARCHAR2(10)
* MANUFACTURER: VARCHAR2(100)
* MODEL: VARCHAR2(100)
* PLATE_NUMBER: VARCHAR2(10)
«PK»
+ PK_VEHICLE(VEHICLE_ID)
«unique»
+ UQ_PLATE_NUMBER(PLATE_NUMBER)
1
1
1
(TRUCK_ID = VEHICLE_ID)
«FK»
0..1
(CAR_ID = VEHICLE_ID)
«FK»
0..1
(MOTORCYCLE_ID = VEHICLE_ID)
«FK»
0..1
TRUCK
«column»
*pfKTRUCK_ID: NUMBER(8)
* LOAD_CAPACITY: NUMBER(8)
«FK»
+ FK_TRUCK_VEHICLE(TRUCK_ID)
«PK»
+ PK_TRUCK(TRUCK_ID)
CAR
«column»
*pfKCAR_ID: NUMBER(8)
* SEATS: NUMBER(8)
«FK»
+ FK_CAR_VEHICLE(CAR_ID)
«PK»
+ PK_CAR(CAR_ID)
MOTORCYCLE
«column»
*pfKMOTORCYCLE_ID: NUMBER(8)
* ENGINE_CAPACITY: NUMBER(8)
«FK»
+ FK_MOTORCYCLE_VEHICLE(MOTORCYCLE_ID)
«PK»
+ PK_MOTORCYCLE(MOTORCYCLE_ID)

Figure 5: RDB PSM with the generalization set realized by related tables

In Constraint 6, the definition of the OCL constraint for the {disjoint,incomplete} generalization set of the running example shown in Figure 2 is shown. In the cases of other values of the generalization set meta-properties, these constraints differ only slightly, as shown in Constraint 7 for a {disjoint,complete} generalization set, in Constraint 8 for a {overlapping,complete} generalization set and in Constraint 9 for a {overlapping,incomplete} generalization set.

Constraint 6 OCL invariant for the {`disjoint,incomplete`} generalization set realized by related tables

```
context v:VEHICLE inv GS_Vehicle_Types:
def Vehicle_Instance: Boolean =
  v.DISCRIMINATOR = 'Vehicle'
    AND NOT (TRUCK.allInstances()->exists
       (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Truck_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck'
    AND TRUCK.allInstances()->exists
       (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Car'
    AND NOT (TRUCK.allInstances()->exists
        (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Motorcycle'
    AND NOT (TRUCK.allInstances()->exists
       (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)

Vehicle_Instance
  OR Truck_Instance
  OR Car_Instance
  OR Motorcycle_Instance
```

Constraint 7 Example of the OCL invariant for a {`disjoint,complete`} generalization set realized by related tables

```
context v:VEHICLE inv GS_Vehicle_Types:
def Truck_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck'
    AND TRUCK.allInstances()->exists
       (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Car'
    AND NOT (TRUCK.allInstances()->exists
        (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Motorcycle'
    AND NOT (TRUCK.allInstances()->exists
       (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)

Truck_Instance
  OR Car_Instance
  OR Motorcycle_Instance
```

Constraint 8 Example of the OCL invariant for a {`overlapping,complete`} generalization set realized by related tables

```
context v:VEHICLE inv GS_Vehicle_Types:
def Truck_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck'
    AND TRUCK.allInstances()->exists
       (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Car'
    AND NOT (TRUCK.allInstances()->exists
        (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Motorcycle'
    AND NOT (TRUCK.allInstances()->exists
       (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)
def Truck_Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Car'
    AND TRUCK.allInstances()->exists
       (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Truck_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Motorcycle'
    AND TRUCK.allInstances()->exists
        (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)
```

```
def Car_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Car_Motorcycle'
    AND NOT (TRUCK.allInstances()->exists
      (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)
def Truck_Car_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Car_Motorcycle'
    AND TRUCK.allInstances()->exists
      (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)

Truck_Instance
  OR Car_Instance
  OR Motorcycle_Instance
  OR Truck_Car_Instance
  OR Truck_Motorcycle_Instance
  OR Car_Motorcycle_Instance
  OR Truck_Car_Motorcycle_Instance
```

Constraint 9 Example of the OCL invariant for a {`incomplete,overlapping`} generalization set realized by related tables

```
context v:VEHICLE inv GS_Vehicle_Types:
def Vehicle_Instance: Boolean =
  v.DISCRIMINATOR = 'Vehicle'
    AND NOT (TRUCK.allInstances()->exists
      (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Truck_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck'
    AND TRUCK.allInstances()->exists
      (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Car'
    AND NOT (TRUCK.allInstances()->exists
        (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
def Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Motorcycle'
    AND NOT (TRUCK.allInstances()->exists
      (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)
def Truck_Car_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Car'
    AND TRUCK.allInstances()->exists
      (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      NOT (MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID))
```

```
def Truck_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Motorcycle'
    AND TRUCK.allInstances()->exists
        (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      NOT (CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID))
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)
def Car_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Car_Motorcycle'
    AND NOT (TRUCK.allInstances()->exists
      (t|t.TRUCK_ID = v.VEHICLE_ID))
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)
def Truck_Car_Motorcycle_Instance: Boolean =
  v.DISCRIMINATOR = 'Truck_Car_Motorcycle'
    AND TRUCK.allInstances()->exists
      (t|t.TRUCK_ID = v.VEHICLE_ID)
    AND
      CAR.allInstances()->exists
        (c|c.CAR_ID = v.VEHICLE_ID)
    AND
      MOTORCYCLE.allInstances()->exists
        (m|m.MOTORCYCLE_ID = v.VEHICLE_ID)

Vehicle
  OR Truck_Instance
  OR Car_Instance
  OR Motorcycle_Instance
  OR Truck_Car_Instance
  OR Truck_Motorcycle_Instance
  OR Car_Motorcycle_Instance
  OR Truck_Car_Motorcycle_Instance
```

5 Transformation of PSM into ISM

In the following sections, the transformation of the resulting PSMs from Section 4 is shown. Only the OCL constraints defined in the RDB PSM for the {`disjoint,incomplete`} generalization set shown in Figure 2 are discussed, as the other variants with the other meta-properties would be realized similarly.

5.1 Single table

When the generalization set is transformed into a single database table, then a special OCL constraint is defined to ensure the meta-properties of the generalization set, as shown in SQL 1.

5.1.1 Database view

The transformed OCL constraint realized by the database view is shown in SQL 1.

SQL 1 Database view to query valid data from the combined `Vehicle` table

```
CREATE VIEW GS_VEHICLE_TYPES_VIEW AS
SELECT * FROM VEHICLE v WHERE
   (v.DISCRIMINATOR = 'Vehicle'
     AND v.LOAD_CAPACITY IS NULL
     AND v.SEATS IS NULL
     AND v.CONTENT IS NULL)
  OR
   (v.DISCRIMINATOR = 'Truck'
     AND v.LOAD_CAPACITY IS NOT NULL
     AND v.SEATS IS NULL
     AND v.CONTENT IS NULL)
  OR
   (v.DISCRIMINATOR = 'Car'
     AND v.LOAD_CAPACITY IS NULL
     AND v.SEATS IS NOT NULL
     AND v.CONTENT IS NULL)
  OR
   (v.DISCRIMINATOR = 'Motorcycle'
     AND v.LOAD_CAPACITY IS NULL
     AND v.SEATS IS NULL
     AND v.CONTENT IS NOT NULL)
WITH CHECK OPTION;
```

5.1.2 CHECK constraint

The variant using a *CHECK constraint* checked after each operation on the table is shown in SQL 2.

SQL 2 CHECK constraint for the combined `Vehicle` table

```
ALTER TABLE VEHICLE ADD CONSTRAINT GS_VEHICLE_TYPES_CHECK CHECK (
   (DISCRIMINATOR = 'Vehicle'
     AND LOAD_CAPACITY IS NULL
     AND SEATS IS NULL
     AND CONTENT IS NULL)
  OR
   (DISCRIMINATOR = 'Truck'
     AND LOAD_CAPACITY IS NOT NULL
     AND SEATS IS NULL
     AND CONTENT IS NULL)
  OR
   (DISCRIMINATOR = 'Car'
     AND LOAD_CAPACITY IS NULL
     AND SEATS IS NOT NULL
     AND CONTENT IS NULL)
  OR
   (DISCRIMINATOR = 'Motorcycle'
     AND LOAD_CAPACITY IS NULL
     AND SEATS IS NULL
     AND CONTENT IS NOT NULL)
);
```

5.1.3 Trigger

The trigger checking the OCL constraint defined in SQL 1 is shown in SQL 3.

SQL 3 Trigger for the combined `Vehicle` table

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_TRIGGER
AFTER INSERT OR UPDATE ON VEHICLE
FOR EACH ROW
DECLARE
  l_valid BOOLEAN;
BEGIN
  l_valid :=
    (:new.DISCRIMINATOR = 'Vehicle'
      AND :new.LOAD_CAPACITY IS NULL
      AND :new.SEATS IS NULL
      AND :new.CONTENT IS NULL)
    OR
    (:new.DISCRIMINATOR = 'Truck'
      AND :new.LOAD_CAPACITY IS NOT NULL
      AND :new.SEATS IS NULL
      AND :new.CONTENT IS NULL)
    OR
    (:new.DISCRIMINATOR = 'Car'
      AND :new.LOAD_CAPACITY IS NULL
      AND :new.SEATS IS NOT NULL
      AND :new.CONTENT IS NULL)
    OR
    (:new.DISCRIMINATOR = 'Motorcycle'
      AND :new.LOAD_CAPACITY IS NULL
      AND :new.SEATS IS NULL
      AND :new.CONTENT IS NOT NULL);

  IF NOT l_valid THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END
```

5.2 Individual tables

When the generalization set is transformed using the *individual tables* approach, a special OCL constraint as shown in SQL 5 is needed for unique attributes of the superclass.

5.2.1 Database views

The OCL constraints shown in SQL 5 transformed into database views are presented in SQL 4.

SQL 4 Database views to query valid data from the individual tables

```
CREATE VIEW UQ_VEHICLE_PLATE_NUMBER_VIEW AS
SELECT * FROM VEHICLE v WHERE (
  NOT EXISTS (SELECT 1 FROM TRUCK t
    WHERE t.PLATE_NUMBER = v.PLATE_NUMBER)
  AND NOT EXISTS (SELECT 1 FROM CAR c
    WHERE c.PLATE_NUMBER = v.PLATE_NUMBER)
  AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
    WHERE m.PLATE_NUMBER = v.PLATE_NUMBER)
)
WITH CHECK OPTION;

CREATE VIEW UQ_TRUCK_PLATE_NUMBER_VIEW AS
SELECT * FROM TRUCK t WHERE (
  NOT EXISTS (SELECT 1 FROM VEHICLE v
    WHERE v.PLATE_NUMBER = t.PLATE_NUMBER)
  AND NOT EXISTS (SELECT 1 FROM CAR c
    WHERE c.PLATE_NUMBER = t.PLATE_NUMBER)
  AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
    WHERE m.PLATE_NUMBER = t.PLATE_NUMBER)
)
WITH CHECK OPTION;

CREATE VIEW UQ_CAR_PLATE_NUMBER_VIEW AS
SELECT * FROM CAR c WHERE (
  NOT EXISTS (SELECT 1 FROM VEHICLE v
    WHERE v.PLATE_NUMBER = c.PLATE_NUMBER)
  AND NOT EXISTS (SELECT 1 FROM TRUCK t
    WHERE t.PLATE_NUMBER = c.PLATE_NUMBER)
  AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
    WHERE m.PLATE_NUMBER = c.PLATE_NUMBER)
)
WITH CHECK OPTION;

CREATE VIEW UQ_MOTORCYCLE_PLATE_NUMBER_VIEW AS
SELECT * FROM MOTORCYCLE m WHERE (
  NOT EXISTS (SELECT 1 FROM VEHICLE v
    WHERE v.PLATE_NUMBER = m.PLATE_NUMBER)
  AND NOT EXISTS (SELECT 1 FROM TRUCK t
    WHERE t.PLATE_NUMBER = m.PLATE_NUMBER)
  AND NOT EXISTS (SELECT 1 FROM CAR c
    WHERE c.PLATE_NUMBER = m.PLATE_NUMBER)
)
WITH CHECK OPTION;
```

5.2.2 CHECK constraints

In SQL 5, the CHECK constraints realizing the transformed OCL constraints shown in SQL 5 are shown. However, although valid according to the SQL:1999 specification [3], the CHECK constraints are not applicable in the current common database engines (including Oracle Database 12c), as they do not support subqueries in the CHECK constraint statements.

SQL 5 CHECK constraints for the individual tables

```
ALTER TABLE VEHICLE
  ADD CONSTRAINT UQ_VEHICLE_PLATE_NUMBER_CHECK
    CHECK (
      NOT EXISTS (SELECT 1 FROM TRUCK t
        WHERE t.PLATE_NUMBER = PLATE_NUMBER)
      AND NOT EXISTS (SELECT 1 FROM CAR c
        WHERE c.PLATE_NUMBER = PLATE_NUMBER)
      AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
        WHERE m.PLATE_NUMBER = PLATE_NUMBER)
);

ALTER TABLE TRUCK
  ADD CONSTRAINT UQ_TRUCK_PLATE_NUMBER_CHECK
    CHECK (
      NOT EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.PLATE_NUMBER = PLATE_NUMBER)
      AND NOT EXISTS (SELECT 1 FROM CAR c
        WHERE c.PLATE_NUMBER = PLATE_NUMBER)
      AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
        WHERE m.PLATE_NUMBER = PLATE_NUMBER)
);

ALTER TABLE CAR
  ADD CONSTRAINT UQ_CAR_PLATE_NUMBER_CHECK
    CHECK (
      NOT EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.PLATE_NUMBER = PLATE_NUMBER)
      AND NOT EXISTS (SELECT 1 FROM TRUCK t
        WHERE t.PLATE_NUMBER = PLATE_NUMBER)
      AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
        WHERE m.PLATE_NUMBER = PLATE_NUMBER)
);

ALTER TABLE MOTORCYCLE
  ADD CONSTRAINT UQ_MOTORCYCLE_PLATE_NUMBER_CHECK
    CHECK (
      NOT EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.PLATE_NUMBER = PLATE_NUMBER)
      AND NOT EXISTS (SELECT 1 FROM TRUCK t
        WHERE t.PLATE_NUMBER = PLATE_NUMBER)
      AND NOT EXISTS (SELECT 1 FROM CAR c
        WHERE c.PLATE_NUMBER = PLATE_NUMBER)
);
```

5.2.3 Triggers

In SQL 6, the trigger defined on the `VEHICLE` table is shown. The triggers for the other tables are defined in the similar way, as shown in SQL 7 for the `TRUCK` table, in SQL 8 for the `CAR` table and in SQL 9 for the `MOTORCYCLE` table.

SQL 6 Trigger for the `VEHICLE` table for the individual tables realization

```
CREATE OR REPLACE TRIGGER UQ_VEHICLE_PLATE_NUMBER_TRIGGER
AFTER INSERT OR UPDATE ON VEHICLE
FOR EACH ROW
DECLARE
  l_count NUMBER := 0;
BEGIN
  SELECT count(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM TRUCK t
      WHERE t.PLATE_NUMBER = :new.PLATE_NUMBER)
    OR
    EXISTS (SELECT 1 FROM CAR c
      WHERE c.PLATE_NUMBER = :new.PLATE_NUMBER)
    OR
    EXISTS (SELECT 1 FROM MOTORCYCLE m
      WHERE m.PLATE_NUMBER = :new.PLATE_NUMBER)
  );

  IF l_count > 0 THEN
    raise_application_error (-20101, 'OCL constraint
      UQ_Vehicle_Plate_number violated!');
  END IF;
END;
```

SQL 7 Trigger for the TRUCK table for the individual tables realization

```
CREATE OR REPLACE TRIGGER UQ_TRUCK_PLATE_NUMBER_TRIGGER
AFTER INSERT OR UPDATE ON TRUCK
FOR EACH ROW
DECLARE
  l_count NUMBER := 0;
BEGIN
  SELECT count(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM VEHICLE v
      WHERE v.PLATE_NUMBER = :new.PLATE_NUMBER)
    OR
    EXISTS (SELECT 1 FROM CAR c
      WHERE c.PLATE_NUMBER = :new.PLATE_NUMBER)
    OR
    EXISTS (SELECT 1 FROM MOTORCYCLE m
      WHERE m.PLATE_NUMBER = :new.PLATE_NUMBER)
  );

  IF l_count > 0 THEN
    raise_application_error (-20101, 'OCL constraint
      UQ_Truck_Plate_number violated!');
  END IF;
END;
```

SQL 8 Trigger for the `CAR` table for the individual tables realization

```
CREATE OR REPLACE TRIGGER UQ_CAR_PLATE_NUMBER_TRIGGER
AFTER INSERT OR UPDATE ON CAR
FOR EACH ROW
DECLARE
   l_count NUMBER := 0;
BEGIN
   SELECT count(1) INTO l_count FROM DUAL WHERE (
      EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.PLATE_NUMBER = :new.PLATE_NUMBER)
      OR
      EXISTS (SELECT 1 FROM TRUCK t
        WHERE t.PLATE_NUMBER = :new.PLATE_NUMBER)
      OR
      EXISTS (SELECT 1 FROM MOTORCYCLE m
        WHERE m.PLATE_NUMBER = :new.PLATE_NUMBER)
   );

   IF l_count > 0 THEN
      raise_application_error (-20101, 'OCL constraint
      UQ_Car_Plate_number violated!');
   END IF;
END;
```

SQL 9 Trigger for the `MOTORCYCLE` table for the individual tables realization

```
CREATE OR REPLACE TRIGGER UQ_MOTORCYCLE_PLATE_NUMBER_TRIGGER
AFTER INSERT OR UPDATE ON MOTORCYCLE
FOR EACH ROW
DECLARE
  l_count NUMBER := 0;
BEGIN
  SELECT count(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM VEHICLE v
      WHERE v.PLATE_NUMBER = :new.PLATE_NUMBER)
    OR
    EXISTS (SELECT 1 FROM TRUCK t
      WHERE t.PLATE_NUMBER = :new.PLATE_NUMBER)
    OR
    EXISTS (SELECT 1 FROM CAR c
      WHERE c.PLATE_NUMBER = :new.PLATE_NUMBER)
  );

  IF l_count > 0 THEN
    raise_application_error (-20101, 'OCL constraint
      UQ_Motorcycle_Plate_number violated!');
  END IF;
END;
```

5.3 Related tables

When the generalization set is transformed using the approach of related tables, it is necessary to define the OCL constraint shown in SQL 6 to check that records in the valid combination of tables are stored according to the metaproperties of the generalization set – {`disjoint,incomplete`} in this case.

To enable the implementation of the constraints, the FOREIGN KEY constraints must be defined as DEFERRABLE, as shown in SQL 10.

SQL 10 Deferrable FOREIGN KEY constraints for the subclass tables

```
ALTER TABLE "TRUCK" ADD CONSTRAINT "FK_TRUCK_VEHICLE"
  FOREIGN KEY ("TRUCK_ID")
    REFERENCES "VEHICLE" ("VEHICLE_ID")
  DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "CAR" ADD CONSTRAINT "FK_CAR_VEHICLE"
  FOREIGN KEY ("CAR_ID")
    REFERENCES "VEHICLE" ("VEHICLE_ID")
  DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE "MOTORCYCLE" ADD CONSTRAINT "FK_MOTORCYCLE_VEHICLE"
  FOREIGN KEY ("MOTORCYCLE_ID")
    REFERENCES "VEHICLE" ("VEHICLE_ID")
  DEFERRABLE INITIALLY DEFERRED;
```

5.3.1 Database view

The database view realizing the OCL constraint in SQL 6 is shown in SQL 11.

SQL 11 Database view to query only valid data from the superclass of the related tables

```
CREATE OR REPLACE VIEW GS_VEHICLE_TYPES_VIEW AS
SELECT * FROM VEHICLE v WHERE
   (v.DISCRIMINATOR = 'Vehicle'
     AND NOT EXISTS (SELECT 1 FROM TRUCK t
       WHERE t.TRUCK_ID = v.VEHICLE_ID)
     AND NOT EXISTS (SELECT 1 FROM CAR c
       WHERE c.CAR_ID = v.VEHICLE_ID)
     AND NOT EXISTS
       (SELECT 1 FROM MOTORCYCLE m
       WHERE m.MOTORCYCLE_ID = v.VEHICLE_ID))
  OR
     (v.DISCRIMINATOR = 'Truck'
     AND EXISTS (SELECT 1 FROM TRUCK t
       WHERE t.TRUCK_ID = v.VEHICLE_ID)
     AND NOT EXISTS (SELECT 1 FROM CAR c
       WHERE c.CAR_ID = v.VEHICLE_ID)
     AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
       WHERE m.MOTORCYCLE_ID = v.VEHICLE_ID))
  OR
     (v.DISCRIMINATOR = 'Car'
     AND NOT EXISTS (SELECT 1 FROM TRUCK t
       WHERE t.TRUCK_ID = v.VEHICLE_ID)
     AND EXISTS (SELECT 1 FROM CAR c
       WHERE c.CAR_ID = v.VEHICLE_ID)
     AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
       WHERE m.MOTORCYCLE_ID = v.VEHICLE_ID))
  OR
     (v.DISCRIMINATOR = 'Motorcycle'
     AND NOT EXISTS (SELECT 1 FROM TRUCK t
       WHERE t.TRUCK_ID = v.VEHICLE_ID)
     AND NOT EXISTS (SELECT 1 FROM CAR c
       WHERE c.CAR_ID = v.VEHICLE_ID)
     AND EXISTS (SELECT 1 FROM MOTORCYCLE m
       WHERE m.MOTORCYCLE_ID = v.VEHICLE_ID))
WITH CHECK OPTION;
```

5.3.2 CHECK constraints

In SQL 12, the CHECK constraint realizing the transformed OCL constraints shown in SQL 6 is shown.

However, although valid according to the SQL:1999 specification [3], the CHECK constraints are not applicable in the current common database engines (including Oracle Database 12c), as they do not support subqueries in the CHECK constraint statements.

SQL 12 CHECK constraint for the VEHICLE table for the related tables realization

```
ALTER TABLE VEHICLE ADD CONSTRAINT GS_VEHICLE_TYPES_CHECK CHECK (
  (DISCRIMINATOR = 'Vehicle'
    AND NOT EXISTS (SELECT 1 FROM TRUCK t
      WHERE t.TRUCK_ID = VEHICLE_ID)
    AND NOT EXISTS (SELECT 1 FROM CAR c
      WHERE c.CAR_ID = .VEHICLE_ID)
    AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
      WHERE m.MOTORCYCLE_ID = VEHICLE_ID))
  OR
    (DISCRIMINATOR = 'Truck'
    AND EXISTS (SELECT 1 FROM TRUCK t
      WHERE t.TRUCK_ID = VEHICLE_ID)
    AND NOT EXISTS (SELECT 1 FROM CAR c
      WHERE c.CAR_ID = VEHICLE_ID)
    AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
      WHERE m.MOTORCYCLE_ID = VEHICLE_ID))
  OR
    (DISCRIMINATOR = 'Car'
    AND NOT EXISTS (SELECT 1 FROM TRUCK t
      WHERE t.TRUCK_ID = VEHICLE_ID)
    AND EXISTS (SELECT 1 FROM CAR c
      WHERE c.CAR_ID = VEHICLE_ID)
    AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
      WHERE m.MOTORCYCLE_ID = VEHICLE_ID))
  OR
    (DISCRIMINATOR = 'Motorcycle'
    AND NOT EXISTS (SELECT 1 FROM TRUCK t
      WHERE t.TRUCK_ID = VEHICLE_ID)
    AND NOT EXISTS (SELECT 1 FROM CAR c
      WHERE c.CAR_ID = VEHICLE_ID)
    AND EXISTS (SELECT 1 FROM MOTORCYCLE m
      WHERE m.MOTORCYCLE_ID = VEHICLE_ID)));
```

5.3.3 Triggers

For the OCL constraint shown in SQL 6, the trigger for the INSERT and UPDATE operations on the superclass table `VEHICLE` is shown in SQL 13. The trigger for the INSERT operation on the subclass table `TRUCK` is shown in SQL 14, the trigger for the UPDATE operation in SQL 15 and the trigger for the DELETE operation in SQL 16. Similarly, the triggers shown in SQL 17, SQL 18, SQL 19, SQL 20, SQL 21 and SQL 22 are defined to check the INSERT, UPDATE and DELETE operations on the other subclass tables.

SQL 13 Trigger for the INSERT and UPDATE operation on the `VEHICLE` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_VEHICLE_TRG
BEFORE INSERT OR UPDATE ON VEHICLE
FOR EACH ROW
DECLARE
  l_count NUMBER(1);
BEGIN
  SELECT COUNT(*) INTO l_count FROM DUAL WHERE (
    (:new.DISCRIMINATOR = 'Vehicle'
      AND NOT EXISTS (SELECT 1 FROM TRUCK t
        WHERE t.TRUCK_ID = :new.VEHICLE_ID)
      AND NOT EXISTS (SELECT 1 FROM CAR c
        WHERE c.CAR_ID = :new.VEHICLE_ID)
      AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
        WHERE m.MOTORCYCLE_ID = :new.VEHICLE_ID))
    OR
     (:new.DISCRIMINATOR = 'Truck'
      AND EXISTS (SELECT 1 FROM TRUCK t
        WHERE t.TRUCK_ID = :new.VEHICLE_ID)
      AND NOT EXISTS (SELECT 1 FROM CAR c
         WHERE c.CAR_ID = :new.VEHICLE_ID)
      AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
        WHERE m.MOTORCYCLE_ID = :new.VEHICLE_ID))
    OR
     (:new.DISCRIMINATOR = 'Car'
      AND NOT EXISTS (SELECT 1 FROM TRUCK t
        WHERE t.TRUCK_ID = :new.VEHICLE_ID)
      AND EXISTS (SELECT 1 FROM CAR c
        WHERE c.CAR_ID = :new.VEHICLE_ID)
      AND NOT EXISTS (SELECT 1 FROM MOTORCYCLE m
        WHERE m.MOTORCYCLE_ID = :new.VEHICLE_ID))
    OR
     (:new.DISCRIMINATOR = 'Motorcycle'
      AND NOT EXISTS (SELECT 1 FROM TRUCK t
        WHERE t.TRUCK_ID = :new.VEHICLE_ID)
      AND NOT EXISTS (SELECT 1 FROM CAR c
        WHERE c.CAR_ID = :new.VEHICLE_ID)
      AND EXISTS (SELECT 1 FROM MOTORCYCLE m
        WHERE m.MOTORCYCLE_ID = :new.VEHICLE_ID)));

  IF l_count = 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 14 Trigger for the INSERT operation on `TRUCK` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_TRUCK_TRG_INSERT
BEFORE INSERT ON TRUCK
FOR EACH ROW
DECLARE
  l_count NUMBER(1) := 0;
BEGIN
  SELECT COUNT(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM VEHICLE v
      WHERE v.VEHICLE_ID = :new.TRUCK_ID)
  );

  IF l_count > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 15 Trigger for the UPDATE operation on `TRUCK` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_TRUCK_TRG_UPDATE
BEFORE UPDATE ON TRUCK
FOR EACH ROW
DECLARE
  l_count_old NUMBER(1) := 0;
  l_count_new NUMBER(1) := 0;
BEGIN
  IF :old.TRUCK_ID <> :new.TRUCK_ID THEN
    SELECT COUNT(1) INTO l_count_old FROM DUAL WHERE (
      EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.VEHICLE_ID = :old.TRUCK_ID)
    );
    SELECT COUNT(1) INTO l_count_new FROM DUAL WHERE (
      EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.VEHICLE_ID = :new.TRUCK_ID)
    );
  END IF;

  IF l_count_old > 0 OR l_count_new > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 16 Trigger for the DELETE operation on the `TRUCK` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_TRUCK_TRG_DELETE
BEFORE DELETE ON TRUCK
FOR EACH ROW
DECLARE
  l_count NUMBER(1);
BEGIN
  SELECT COUNT(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM VEHICLE v
      WHERE v.VEHICLE_ID = :old.TRUCK_ID)
  );

  IF l_count > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 17 Trigger for the INSERT operation on `CAR` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_CAR_TRG_INSERT
BEFORE INSERT ON CAR
FOR EACH ROW
DECLARE
  l_count NUMBER(1) := 0;
BEGIN
  SELECT COUNT(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM VEHICLE v
      WHERE v.VEHICLE_ID = :new.CAR_ID)
  );

  IF l_count > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 18 Trigger for the UPDATE operation on `CAR` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_CAR_TRG_UPDATE
BEFORE UPDATE ON CAR
FOR EACH ROW
DECLARE
  l_count_old NUMBER(1) := 0;
  l_count_new NUMBER(1) := 0;
BEGIN
  IF :old.CAR_ID <> :new.CAR_ID THEN
    SELECT COUNT(1) INTO l_count_old FROM DUAL WHERE (
      EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.VEHICLE_ID = :old.CAR_ID)
    );
    SELECT COUNT(1) INTO l_count_new FROM DUAL WHERE (
      EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.VEHICLE_ID = :new.CAR_ID)
    );
  END IF;

  IF l_count_old > 0 OR l_count_new > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 19 Trigger for the DELETE operation on the `CAR` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_CAR_TRG_DELETE
BEFORE DELETE ON CAR
FOR EACH ROW
DECLARE
  l_count NUMBER(1);
BEGIN
  SELECT COUNT(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM VEHICLE v
      WHERE v.VEHICLE_ID = :old.CAR_ID)
  );

  IF l_count > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 20 Trigger for the INSERT operation on `MOTORCYCLE` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_MOTORCYCLE_TRG_INSERT
BEFORE INSERT ON MOTORCYCLE
FOR EACH ROW
DECLARE
  l_count NUMBER(1) := 0;
BEGIN
  SELECT COUNT(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM VEHICLE v
      WHERE v.VEHICLE_ID = :new.MOTORCYCLE_ID)
  );

  IF l_count > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 21 Trigger for the UPDATE operation on `MOTORCYCLE` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_MOTORCYCLE_TRG_UPDATE
BEFORE UPDATE ON MOTORCYCLE
FOR EACH ROW
DECLARE
  l_count_old NUMBER(1) := 0;
  l_count_new NUMBER(1) := 0;
BEGIN
  IF :old.MOTORCYCLE_ID <> :new.MOTORCYCLE_ID THEN
    SELECT COUNT(1) INTO l_count_old FROM DUAL WHERE (
      EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.VEHICLE_ID = :old.MOTORCYCLE_ID)
    );
    SELECT COUNT(1) INTO l_count_new FROM DUAL WHERE (
      EXISTS (SELECT 1 FROM VEHICLE v
        WHERE v.VEHICLE_ID = :new.MOTORCYCLE_ID)
    );
  END IF;

  IF l_count_old > 0 OR l_count_new > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

SQL 22 Trigger for the DELETE operation on the `MOTORCYCLE` table of the related tables realization

```
CREATE OR REPLACE TRIGGER GS_VEHICLE_TYPES_MOTORCYCLE_TRG_DELETE
BEFORE DELETE ON MOTORCYCLE
FOR EACH ROW
DECLARE
  l_count NUMBER(1);
BEGIN
  SELECT COUNT(1) INTO l_count FROM DUAL WHERE (
    EXISTS (SELECT 1 FROM VEHICLE v
      WHERE v.VEHICLE_ID = :old.MOTORCYCLE_ID)
  );

  IF l_count > 0 THEN
    raise_application_error
      (-20101, 'OCL constraint GS_Vehicle_Types violated!');
  END IF;
END;
```

Bibliography

[1] Oracle. Oracle Database 12c. Available from: `\url{http://www.oracle.com/us/corporate/features/database-12c/index.html}`

[2] Rybola, Z.; Pergl, R. Towards OntoUML for Software Engineering: Transformation of Rigid Sortal Types into Relational Databases. In *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems*, edited by M. P. M. Ganzha, L. Maciaszek, number 8 in ACSIS, Gdansk, Poland, 2016, ISBN 978-83-60810-90-3, ISSN 2300-5963, p. 1581–1591, doi:10.15439/2016F250, doi: 10.15439/2016F250.

[3] Melton, J. *Advanced SQL:1999*. Morgan Kaufmann Publishers, 2003.