Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 1357–1364

ISBN 978-83-60810-51-4

Transformation of Special Multiplicity Constraints - Comparison of Possible Realizations

Zdenek Rybola* and Karel Richta†‡
*Faculty of Information Technology, Czech Technical University in Prague, Email: rybolzde@fit.cvut.cz
†Faculty of Mathematics and Physics, Charles University in Prague, Email: richta@ksi.mff.cuni.cz
‡Faculty of Electrical Engineering, Czech Technical University in Prague, Email: richta@fel.cvut.cz

***Abstract*—This paper deals with the transformation of binary relationships from Platform Independent Model (PIM) in UML to Platform Specific Model (PSM) for relational databases from the point of view of Model Driven Development (MDD). The paper summarizes the transformation of a binary relationship with multiplicity constraints and focuses on problems of the current approach for such transformations. In this paper, we deal with the source class optionality that is usually omitted during the transformation. We propose three various possible realizations for the optionality constraint to be used in the transformation of a PIM to a PSM. We also present some experiments to demonstrate that our approach to the optionality constraint realization is equivalent in execution time to the current approach of omitting this constraint during the transformation while providing better consistency control. Finally, we generalize our proposed solution to be used for special multiplicity values used in a PIM.**

I. Introduction

MODEL Driven Development (MDD) is a development process based on modeling and transformations. It is based on the Model Driven Architecture (MDA) developed by the Object Management Group (OMG) [1], [2]. The process usually consists of creating a set of models of various abstraction levels and points of view along with transformations between these models. These transformations enable forward engineering and reverse engineering – processes of transforming abstract models into more specific models or source code, and specific models into more abstract models, respectively.

This paper deals with the transformation of binary relationships along with their multiplicity constraints from a Platform Independent Model (PIM) to a Platform Specific Model (PSM) for relational databases. As a running example, we use UML to express the models and SQL as the domain specific language of a relational database for the implementation. Unified Modeling Language (UML) [3], [4] is a language for creating and maintaining variety of models using diagrams and additional components. Additional constraints for UML models are defined in Object Constraint Language [5]. OCL is a specification language used to define restrictions such as invariants – conditions that must be satisfied by all instances of the element –, pre- and post-conditions for connected model elements and can be also used as a general object query language.

In particular, we deal with a special case of a multiplicity constraint used in many-to-one relationships where the minimal multiplicity value of the *many* entity is equal to *one*. In [6], this constraint is called the *inverse referential integrity constraint*. This constraint is often used in models when we need to restrict the required existence of both related entities in such a relationship – none of them can exist without the other one. An example of this situation could be a register of students' accommodation – we register only students with an address and keep only addresses where someone lives (see Fig. 2).

Many CASE tools provide support for the modeling in UML and the MDD approach with transformation of models, for instance Enterprise Architect [7]. However, most of the tools do not provide full support for MDD and omit some aspects such as the source class optionality during the transformations. Therefore we try to bring these aspects to the attention by defining them in another formalism by OCL invariants. Thanks to this approach, various OCL tools such as DresdenOCL Toolkit [8] can be used to transform them to an implementation. Finally, we generalize our proposed mechanism for special multiplicity values. An example for this situation could be a part of a university information system where each year consists of exactly two or three terms. These special multiplicity constraints are usually not realized at all although they are defined in the model.

In our approach, according to the MDD approach, we suggest transforming a PIM to a PSM for a relational database together with OCL constraints for such multiplicity constraints that cannot be restricted just by the FOREIGN KEY, UNIQUE and NOT NULL constraints. The PSM is then transformed to SQL scripts for database creation. The OCL constraints can be transformed by an OCL tool. We also plan to develop a tool that utilizes our approach and proposes the transformation of a PIM and the PSM along with the OCL constraints to the realizations we propose.

Our approach was already published in [9], [10]. In this particular paper, we propose three various possible realizations of the constraint to be used in the transformation process by a database view, a check constraint or a trigger. We also present some experimental comparison of our proposed realizations to the common realization where the constraint is not realized.

The paper is structured as follows: In section II, we discuss related work and existing tools and their problems. In section III, the transformation of binary relationship and multiplicity constraints is defined. The possible realizations of the special constraint for source entity optionality are defined in section

IV. Experiments and their results are given in section V. Generalization of the mechanism for special multiplicity values is given in section VI. In section VII, the conclusions are given.

II. Related work

The problem of transforming data PIM to relational database schema is nothing new. There is a lot of books such as Rob and Coronel [11] describing the principles of data modeling and transformation to database schema, and tools such as DresdenOCL [8] and Enterprise Architect [7] providing support for such modeling and transformations. Rob and Coronel [11] suggest transforming entities to database tables with each entity attribute transformed to a table column. Instances of such entity are stored as rows in the corresponding table. A relationship between two entities – the source and the target entity, identified according to the direction and multiplicities of the relationship in the PIM – is transformed in a FOREIGN KEY constraint using the primary key of the table for the target entity and a special column in the table of the source entity. Using this mechanism we are able to restrict the optionality of the target entity with a *NOT NULL* constraint and the cardinality of the source entity with a *UNIQUE* constraint on the table column referring to the target entity table. The target entity cardinality is automatically ensured by the mechanism of foreign key as any single table row can refer only to a single target row. However, there is no way to restrict the optionality of the source entity using just the foreign key. Rob and Coronel [11] also suggest using an *ON DELETE RESTRICT* clause for the foreign key constraint to prevent break of the target entity optionality constraint if required. However, they suggest no solution to restrict the source entity optionality. Therefore, a special constraint has to be defined and realized to check this restriction.

In [9], [10], we presented a more exact definition of the transformation of a data PIM to a PSM for a relational database from the point of view of binary relationships. We also defined the minimal and maximal multiplicities of binary relationships using OCL invariants and suggested transformation rules of these binary relationships according to the defined multiplicities. Finally, we defined a special constraint in OCL to ensure minimal source multiplicity (i.e. optionality of the source entity) and suggested a simple method how to implement it in SQL on the PSM level. In our current paper, we present other possibilities for transformation of the special constraint for checking and ensuring the source entity optionality. We also provide some experiments to compare performance of the suggested implementations.

In [12], Cabot and Teniente identify various limitations of current tools regarding code generation of the integrity constraints defined in PIMs including OCL constraints and multiplicity constraints. In our paper, we focus on the multiplicity constraints and propose possible realizations of such constraints in relational databases. In addition to the tools compared in [12], we also identify another CASE tool with similar limitations. Enterprise Architect (EA) [7] is a complex commercial CASE tool for maintaining models, transformation of models, source code generation and reverse engineering process from a source code to a PSM. Beside others, it provides a transformation from a PIM data model to a specific database PSM model and generation of SQL source code from such a PSM model. However, the default transformations of Enterprise Architect do not consider optionality of relationships to determine neither the direction of the relationship implementation using the foreign key constraint nor the required multiplicity restrictions. It does not support special multiplicity values either. Although EA allows definition of OCL constraints, the constraints are not realized by the transformations.

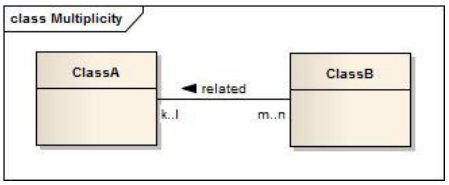

Fig. 1. Labeling of the multiplicities of relationship between two classes

In [6], the authors also identify a problem of current relational databases in the realization of the source class optionality constraint – they call this constraint an inverse referencial integrity constraint (IRIC). The authors also present an approach to the automated implementation of the IRICs by database triggers in a tool called IIS*Case. This tool is designed to provide complete support for developing database schemas including checking consistency of constraints embedded into the DB [6] and integration of subschemas into a relational DB schema [13].

III. Transformation of PIM to PSM for Relational Databases

Our approach to the transformation of a data PIM to a PSM for relational databases has been introduced in [9], [10]. This section brielfy summarizes our approach.

A. Platform Independent Model

In the platform independent model (PIM), each object of a problem domain is represented by a class – in some models called *an entity* – with a set of attributes and its instances [4]. The classes are connected together by associations to represent relationships between objects. Each association has its name to describe the meaning of the relationship and multiplicities to define the number of instances of each entity related together. Fig. 1 shows the general form of modeling a binary relationship using UML class diagram [4].

The *minimal multiplicity* defines the minimal number of instances of one entity related to a single instance of the other entity. However, this constraint is usually used only to define the obligation of the entity to be related to the other entity by using the values *0* for optional or *1* for required occurrence, respectively. Therefore the minimal multiplicity is sometimes called *optionality*. The *maximal multiplicity* – also called *cardinality* – defines the maximal number of instances of one entity related to a single instance of the other entity.

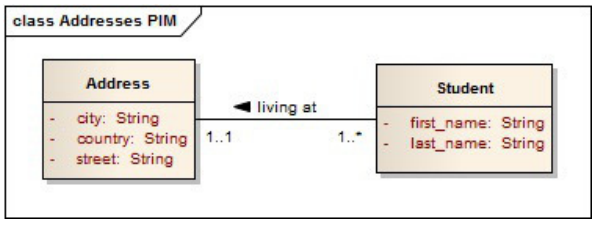

Fig. 2. PIM of one-to-many relationship of addresses and students

However, this constraint is usually used only to restrict the number of instances to a single instance by using the values of * for unrestricted number of instances or *1* for a single instance, respectively. In Fig. 1, k and l define the minimal and maximal multiplicities of instances of ClassA related to a single instance of ClassB, while m and n define the minimal and maximal multiplicities of instances of ClassB related to a single instance of ClassA. In the following, we focus on the definition and realization of constraints for the optionality and cardinality constraints, i.e. multiplicity values *0*, *1* and *. The generalization of the constraints for special multiplicity values – i.e. other values of multiplicities – is mentioned in section VI.

B. Transformation to Platform Specific Model

In general, data is stored as rows in database tables with a set of columns to store specific data in a relational database. Therefore, the classes of a PIM are transformed to database tables with columns corresponding to the classes' attributes. Each row in a database table is identified by a primary key.

Associations defined in a PIM are realized by a mechanism called foreign key [11]. This mechanism adds a special column to one of the related tables – the *source table* in the terms of the direction of the relationship realization – that links a row in that table to a single row in the other – target – table using the target's primary key. Using this mechanism, each row can refer only to a single target row so we can only realize one-to-one and one-to-many relationships, and therefore the target cardinality is always restricted to one. However, many-to-many relationships can be transformed into two many-to-one relationships and then transformed the usual way using the foreign key mechanism as well [11].

In fact, this restriction is the most important key to determine the direction of the relationship realization. The Fig. 2 shows one-to-many relationship between classes Address and Student with the meaning of many students being able to live at the same address. The cardinality n = * requires the realization by the foreign key referring from the table Student to the table Address as shown in Fig. 3, and therefore restricting the target cardinality $l = 1$. Furthermore, the model defines the source optionality $m = 1$.

By defining the FOREIGN KEY constraint, we automatically restrict the target cardinality to $l = 1$ because a single foreign key value always refers to a single target row [14]. A NOT NULL constraint can be used to restrict the value to be entered for each row and so make each row to be related to some target row, determining the target optionality $k = 1$. Furthermore, a UNIQUE constraint would prevent from entering more rows in the source table with the same foreign key value, determining the source cardinality $n = 1$ – however, this is not our case.

The only multiplicity value we have not restricted yet is the source optionality $m = 1$. There is no possible way to restrict the source entity optionality using just the foreign key mechanism. As mentioned earlier, the usual method is to omit this restriction and provide the constraint check by the application using the database schema [11], [9]. However, we present a method to create a special constraint realization to restrict the source optionality using special constraint to keep the database schema consistent independently of the application. This constraint can be expressed using OCL on the PSM level like this:

```
context a:Address inv minStudents:
Student.allInstances()
->exists(s | s.address_id = a.address_id)
```

IV. Realization of the Source Entity Optionality Constraint

The special constraint for optionality restriction of the source entity defined above can be transformed into several various realizations. This section deals with these possible realizations and their pros and cons. The proposed realizations are given in the Oracle SQL syntax.

A. Database views

The most straightforward realization of the constraint are database views [9], [10]. Each constraint is transformed into a database view to filter only valid data stored in a table. This approach is inspired by DresdenOCL Toolkit [8] which transforms constraints defined in OCL into database views. These views contain only the rows that satisfy the defined constraint using the WHERE clause. The realization of the constraint for the optionality of Student in Fig. 3 can be defined as follows:

```
CREATE VIEW valid_addresses AS
SELECT a.* FROM Address a WHERE EXISTS
(SELECT 1 FROM Student s
WHERE s.address_id = a.address_id)
```

However, this approach does not ensure the data stored in the database are consistent. We are still able to insert invalid data violating the defined multiplicity constraints. The application itself must use the view to work only with the valid data and provide support to correct the invalid data. For this process, an inverse view can be useful to detect the invalid data violating the constraints. Such an inverse view can be defined as follows:

```
CREATE VIEW invalid_addresses AS
SELECT a.* FROM Address a WHERE NOT EXISTS
(SELECT 1 FROM Student s
WHERE s.address_id = a.address_id)
```

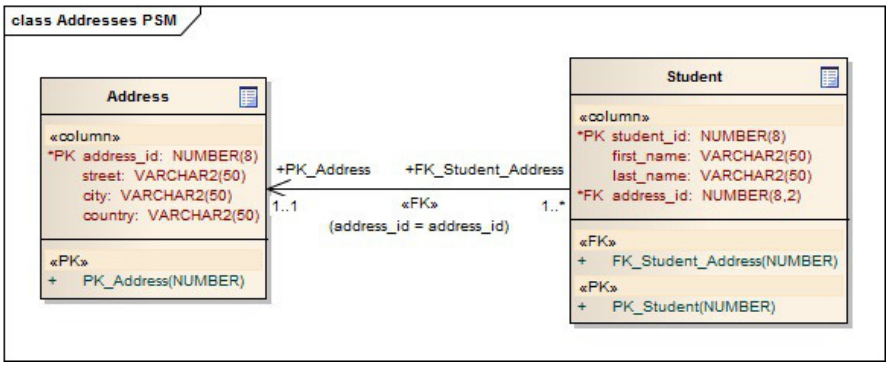

Fig. 3. PSM of one-to-many relationship of addresses and students

The realization using database views does not increase the time required for inserting new entries in the tables. However, the selection of valid data consists of evaluating the condition of the view and thus increasing the time required to get the data.

B. CHECK constraint

In relational databases, CHECK constraint can be used to restrict values in a column of a table [14]. The constraint is checked whenever a value is inserted or updated in the column and the operation is rolled back if the constraint fails. Such a constraint can restrict range for numeric values or provide a list of valid values. Using this approach, we can define a CHECK constraint to allow only the primary key values of addresses that are referred by rows in the students' table. According to the SQL:1999 specification [14], the constraint for the situation in Fig. 3 can be defined as follows:

```
ALTER TABLE Address
ADD CONSTRAINT address_check
CHECK (address_id IN
(SELECT address_id FROM Student))
```

However, after adding this constraint to the database schema we would not be able to insert new data because of two inverse checks – the CHECK constraint requiring existing students for an address, and the FOREIGN KEY constraint requiring an existing address for a student. This conflict can be resolved by deferring one of the constraints [14]. Defining a constraint deferrable makes the database engine to check the constraint at the end of the transaction instead of checking it in the time of inserting the data. Using the deferred FOREIGN KEY constraint, we can insert students referring to an address not yet inserted and then insert that address. The CHECK constraint is evaluated when inserting the address but there already exist students referring to it. On the other hand, the FOREIGN KEY constraint is not evaluated while inserting students – when it would fail –, instead it is evaluated in the end of the transaction when the address is already inserted. The deferred FOREIGN KEY constraint can be defined as follows:

```
ALTER TABLE Student
ADD CONSTRAINT student_address
FOREIGN KEY (address_id)
REFERENCES Address
DEFERRABLE INITIALLY DEFERRED;
```

Using this realization, the data consistence is ensured because it is impossible to insert invalid data. However, there are some problems with this realization. One of the problems is that if a violation is detected by the deferred constraint the whole transaction is rolled back as it can not be determined which command have caused the violation [14]. Another important problem of this realization is the fact that, although specified by the SQL:1999 specification [14], none of current common database engines support this kind of CHECK constraints as it contains a subquery. Therefore we can not use this realization until the database engines provide the support for this specification.

C. Trigger

Triggers are special procedures available in many relational databases [14] connected to some special events on a table. Each trigger can be defined to be executed BEFORE or AFTER such event while the event can be any statement to insert new rows, update rows or delete rows, including its combination. Furthermore, triggers can be defined to be executed for each affected row or for all rows affected by the statement at once. During the execution of the trigger, the original row data and the new row data can be accessed using special keywords.

In context of constraints checking, a trigger can be defined to check the validity of the updated data. Such a trigger would throw an exception if the updated data is invalid or make the other data invalid. For the situation in Fig. 3, the trigger would check existence of students for the inserted or updated address. Similar trigger should be created before updating or deleting students to prevent removing the last student of an address. The insert and update trigger for Oracle 10g database can be defined as follows:

```
CREATE OR REPLACE TRIGGER check_students
BEFORE INSERT OR UPDATE ON Address
FOR EACH ROW
DECLARE
  l_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO l_count
  FROM Student s
  WHERE s.address_id = :new.address_id;
  IF l_count = 0 THEN
    raise_application_error
     (-20910,
      'student not found for address');
  END IF;
END
```

This trigger is executed before each insert or update statement executed for the address table. Students referring the inserted address by its primary key are searched. If no students are found, exception is thrown causing the statement to be rolled back. Because this trigger is always executed in the time of address insertion, the students must be inserted before this statements. To enable this the FOREIGN KEY constraint must be defined deferrable as described in the previous section.

The trigger ensures the data stored in the database is consistent as it does not allow to insert invalid data violating the multiplicity constraint. However, the check is executed for each address insertion or update searching for the related students. This search takes the longer time the more records are already stored in the table. However, this searching time can be decreased by defining an index on the foreign key value in the referring table. For the situation in Fig. 3, the index can be defined as follows:

```
CREATE INDEX students_addresses_index
ON Student (address_id);
```

V. Experiments

To prove that our proposed realizations are suitable for wide usage in relational databases, we made some experiments. These experiments compare our proposed realizations to the commonly used realization without realizing such constraint in two areas - inserting new entries with the triggers and selecting through the view.

The suggested realization by triggers requires select operations executed while inserting new entries to the table. Therefore we made an experiment to compare our suggested realization by triggers to the commonly used realization omitting this constraint. In this experiment, the realization by the check contraint should have been also tested but it cannot be implemented in Oracle database as it does not support queries in check constraints.

On the other hand, the suggested realization by views to be used to select only valid data requires additional condition evaluation while selecting. Therefore, we also made experiments to compare the time of selection of entries from the *Address* table directly and using the view.

TABLE I
Variants of create scripts for various constraint realizations
(+ implemented, * implemented deferrable, - not implemented)

Variant	primary keys	foreign key	index	trigger
Foreign key	+	+	-	-
Trigger	+	*	-	+
Trigger with index	+	*	+	+

```
1  CREATE PROCEDURE insert_values (p_count)
2  IS
3  BEGIN
4    FOR i IN 1..p_count/10
5      -- insert students referencing incorrect address
6      INSERT INTO students_table REFERENCING address 0;
7
8    FOR i IN 1..p_count
9      -- insert new address
10     INSERT INTO addresses_table WITH ID i;
11     -- insert mod(i,5) students referencing the new address
12     FOR j IN 1..mod(i,5)
13       INSERT INTO students_table REFERENCING address i;
14     -- commit the inserted group
15     COMMIT
16 END
```

Fig. 4. Pseudo-SQL code of experimental insert scripts

We used Oracle 10g XE database for our experiments.

A. The Insertion Experiment

The experiment presents evaluation time comparison of inserting new entries for various implementations of a one-to-many relationship in a relational database for various number of entries already stored in the tables. We developed several creation scripts for creating the database tables with constraints and appropriate insert scripts for each implementation to simulate the process of inserting new entries in the database schema.

Table I presents the constraint implementation for each variant. The *Foreign key* variant is the standard realization of one-to-many relationship with a primary key in both tables and a foreign key referring from the table *Student* to the table *Address*. This variant does not restrict minimal multiplicity for students living at an address. The *Trigger* variant adds a trigger to check existing student for each address. In this variant, the trigger prevents inserting addresses where no students live. Finally, the *Trigger with index* variant adds an index on the address id in the table *Student* to speed up the search of students by their address.

The pseudo-SQL code of the insert script is given in Fig. 4. First, the insert script tries to insert students with incorrect reference to an address. This constraint violation is checked by the foreign key in each of the variants. Second, the script inserts several students with reference to a new address. The number of students referring to the same address differs to check the options of inserting no, one or more students for an address, respectively. A commit operation comes after each group of students with the same address to apply the constraints check. In the case of the constraint implementation

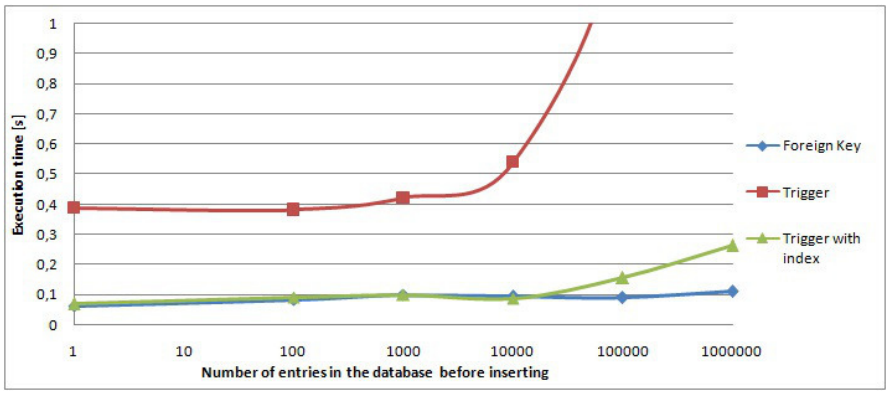

Fig. 5. Execution time of insertion of new entries for various implementation variants

TABLE II
THE RESULTS OF THE INSERTION EXPERIMENT - EXECUTION TIMES OF NEW ENTRIES INSERTION FOR VARIOUS IMPLEMENTATIONS IN SECONDS.

Number of entries	Foreign key	Trigger	Trigger with index
0	0.061	0.387	0.068
100	0.082	0.381	0.090
1000	0.098	0.422	0.100
10000	0.093	0.540	0.088
100000	0.089	1.460	0.155
1000000	0.110	12.579	0.262

by a trigger, the students are inserted before the address because the trigger checks the value immediately while the foreign key is deferred.

Fig. 5 presents execution time of the insertion of 100 new entries for each variant in a database already containing various number of entries. As we can see, the *Foreign key* variant results to be the fastest as there are no constraints to check when inserting. However, the optionality of students for each address is not checked and addresses without students are inserted, too. The *Trigger* variant enforces only valid addresses to be inserted, however, the constraint check slows down the evaluation the more entries already exist in the tables. However, the *Trigger with index* variant results to be only slightly slower than the *Foreign key* while enforcing only valid data inserted to the database. The measured data is summarized in the Table II.

B. The Selection Experiment

This experiment presents a comparison of the execution time of a SELECT operation from the table *Address* directly and using the view for accessing only the valid data. Two different types of SELECT queries were measured – a *select* using the primary key value to get a single result and a *select* using not-indexed column of unique values to get a single result. Three various implementations were measured for each select. The *Table* variant presents selects from the table *Address* directly without checking the constraint. The *View* variant presents selects from the view *ValidAddresses* defined over the table *Address* to check the existent entries in the *Student* table and to select only from valid addresses. The *View with index* variant presents selects from the view *ValidAddresses* defined over the table *Address* with an index defined on the address id in the table *Student* to speed up the search of students by their address while checking the existence. All select variants are summarized in Table III.

Fig. 6 presents results of the experiment. It shows the execution time of each variant for various number of addresses stored in the *Address* table together with associated students in the *Student* table. The *Table* variants result to be fastest as there is no additional condition to check the constraint. However, the query returns both valid and invalid data according to the constraint. The *View* variants become much slower when more entries are stored in the tables because of the additional constraint with a subquery to check the valid addresses. However, only valid addresses according to the constraint are returned. The index defined over the foreign key value in the *Student* table speeds up the subquery execution rapidly as shown by the *View with index* variant results. Therefore, the *View with index* variant results to be equivalent in the execution time to the selection from the table directly while returning only the valid data. The measured data is summarized in the Table IV.

VI. SPECIAL MULTIPLICITIES

Usually, multiplicities of binary relationships are used to restrict obligation of the relationship existence and to define if the relationship is limited to only one entity. In other words, the target minimal multiplicity defines if there has to be a related target entity ($m = 1$) or not ($m = 0$) while target maximal multiplicity defines if there can be a collection of related target entities for each source entity ($n = *$) or just one target entity ($n = 1$). The same stands for source multiplicities.

However, in practise, we can use multiplicities to restrict the numbers of related entities more strictly – we can define a range of numbers. In Fig. 7, a small part of university information system is shown. We define class Year consisting of exactly two terms – winter term and summer term. Using the common mechanism of foreign key, the model is transformed to table Year and table Term as shown in Fig. 8. A foreign

TABLE III
VARIANTS OF SELECTS EXECUTED AND MEASURED

Select	Variant	Condition	Source	Index in the Student table
S1A	Table	address id	table Address	not defined
S1B	View	address id	view ValidAddresses	not defined
S1C	View with index	address id	view ValidAddresses	defined
S2A	Table	street	table Address	not defined
S2B	View	street	view ValidAddresses	not defined
S2C	View with index	street	view ValidAddresses	defined

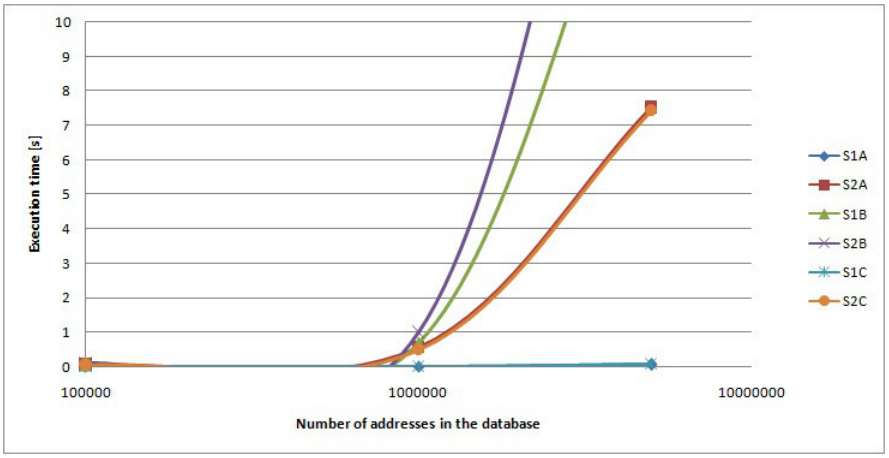

Fig. 6. Execution time of selection of entries for various implementation variants

TABLE IV
THE RESULTS OF THE SELECTION EXPERIMENT - EXECUTION TIMES OF SELECT OPERATIONS FOR VARIOUS IMPLEMENTATIONS IN SECONDS.

Number of entries	S1A	S2A	S1B	S2B	S1C	S2C
1	0.001	0.004	0.003	0.001	0.001	0.001
100	0.001	0.002	0.001	0.001	0.001	0.001
1000	0.001	0.001	0.001	0.002	0.001	0.001
10000	0.001	0.006	0.005	0.009	0.001	0.006
100000	0.001	0.055	0.044	0.089	0.001	0.054
1000000	0.003	0.552	0.691	0.994	0.005	0.495
5000000	0.051	7.523	17.168	24.547	0.065	7.426

key constraint is created in the table Term referring an entry in the table Year. The foreign key value can be restricted by NOT NULL constraint to enforce each term having a year related. No UNIQUE contraint will be defined for the foreign key value to enable having more terms referrencing the same year. However, there is no way to restrict the number of terms related to a single year to exactly two as defined in the model just by the foreign key mechanism.

These restrictions are usually omitted during transformation and constraints are not at all realized or must be implemented manually. However, we can generalize our proposed mechanism to restrict the number of terms in PSM [9], [10]. Using this mechanism, OCL invariants can be automatically defined for the multiplicities in PIM and realized by views and triggers in PSM of relational database. This way, we can easily check the consistency of data in the schema and enforce inserting only valid data according to the restrictions in defined in PIM.

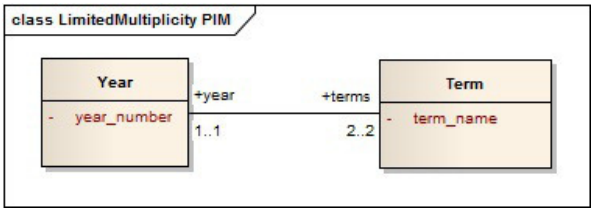

Fig. 7. PIM of year and terms with special multiplicities

VII. CONCLUSIONS

In this paper, we summarized the currently used method for modeling binary relationships in data models using UML class diagram. We showed the way to define multiplicity constraints in the model. Furthermore, we showed the usual transformation of the model from PIM to PSM for relational database and showed the usual transformations for multiplicity

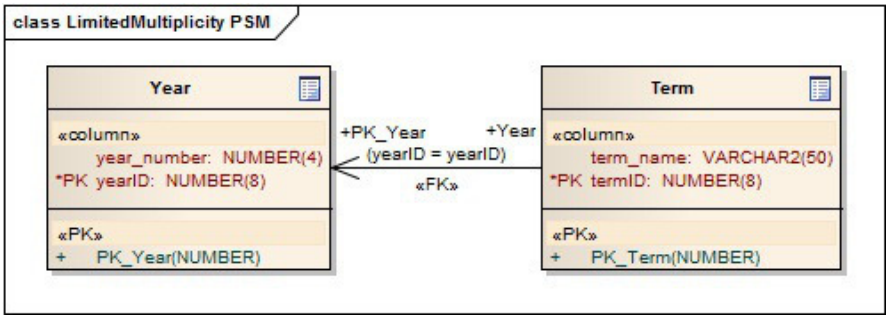

Fig. 8. PSM of year and terms with special multiplicities

constraints using FOREIGN KEY, NOT NULL and UNIQUE constraints in SQL.

We pointed out the constraint for the source entity optionality. This constraint is often used in the model but not realized in the database because the foreign key is insufficient to realize it. Therefore, we defined this constraint in another formal way by an OCL invariant and suggested three methods of realization of this special constraint.

We also compared the suggested realizations to the currently used approaches in context of the execution time while inserting new data to the tables and selecting existing data from the table. The trigger realization of the special constraint with an index created on the foreign key column was proved in the insertion experiment to be able to prevent the insertion of invalid data in the database schema and, in the same time, to be still comparable in the execution time to the realization omitting this constraint. On the other hand, the view implementation with the index created on the foreign key column was proved in the selection experiment to be able to filter our the invalid data and, in the same time, to be equivalent in the execution time to selection from the table directly. Therefore, we suggest the constraint to be realized by transformations in CASE tools' transformations of data models to relational databases either by the trigger to prevent inserting invalid data or the view to filter invalid data from selecting. This realization will definitelly support the analysis and design processes to create consistent database schemas using the MDD approach.

Furthermore, as we have shown, our proposed mechanism can be easily used to automatically generate constraints for special multiplicity values used in PIM. These constraints are transformed to database views and triggers in the same way enforcing the data to be valid according to the multiplicities in the PIM.

VIII. Acknowledgements

We would like to thank for financial support of Student Grant Competition of CTU in Prague, grant number SGS12/093/OHK3/1T/18 and also to AVAST Foundation in Prague.

References

[1] OMG, "Object management group," http://www.omg.org/, Dec. 2011.
[2] OMG, J. Miller, and J. Mukerji, "MDA guide version 1.0.1," http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, Jun. 2003.
[3] OMG, "UML 2.3," http://www.omg.org/spec/UML/2.3/, Feb. 2011.
[4] J. Arlow and I. Neustadt, *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition)*. Addison-Wesley Professional, 2005.
[5] OMG, "Object constraint language, version 1.3," http://www.omg.org/spec/OCL/2.2/PDF, Feb. 2010.
[6] S. Aleksić, S. Ristić, and I. Luković, "An approach to generating server implementation of the inverse referential integrity constraints," in *Proceedings*. Amman, Jordan: AL-Zaytoonah University of Jordan, May 2011.
[7] Sparx Systems, "Enterprise architect - UML design tools and UML CASE tools for software development," http://www.sparxsystems.com.au/products/ea/index.html, Mar. 2011.
[8] B. Demuth, "DresdenOCL," http://www.reuseware.org/index.php/DresdenOCL, Jan. 2011.
[9] Z. Rybola and K. Richta, "Transformation of binary relationship with particular multiplicity," in *DATESO 2011*, vol. 11. Písek, Czech Republic: Department of Computer Science, FEECS VSB - Technical University of Ostrava, Apr. 2011, pp. 25–38. [Online]. Available: http://www.informatik.uni-trier.de/~ley/db/conf/dateso/dateso2011.html
[10] K. Richta and Z. Rybola, "Transformation of relationships from UML/OCL to SQL," in *ITAT 2011: Zborník príspevkov prezentovaných na konferencii ITAT*, vol. 11. Terchová, Slovakia: University of P. J. Šafárik, Košice, Slovakia, Sep. 2011. [Online]. Available: http://itat.ics.upjs.sk/proceedings/itat2011-zbornik.pdf
[11] P. Rob and C. Coronel, *Database Systems: Design, Implementation, and Management*, 2nd ed. Boyd & Fraser, 1995.
[12] J. Cabot and E. Teniente, "Constraint support in MDA tools: A survey," in *Model Driven Architecture Foundations and Applications*, ser. Lecture Notes in Computer Science, A. Rensink and J. Warmer, Eds., vol. 4066. Springer Berlin / Heidelberg, 2006, pp. 256–267. [Online]. Available: http://www.springerlink.com/content/4902321654674181/abstract/
[13] I. Luković, P. Mogin, J. Pavićević, and S. Ristić, "An approach to developing complex database schemas using form types," *Software: Practice and Experience*, vol. 37, no. 15, p. 16211656, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1002/spe.v37:15
[14] J. Melton, *Advanced SQL:1999*. Morgan Kaufmann Publishers, 2003.