DOI:10.2298/CSIS111102003A

A Design Specification and a Server Implementation of the Inverse Referential Integrity Constraints

Slavica Aleksić[1], Sonja Ristić[2], Ivan Luković[1], and Milan Čeliković[1]

[1]University of Novi Sad, Faculty of Technical Sciences,
Department of Computing and Control
Trg Dositeja Obradovića 6
21000 Novi Sad, Serbia
{slavica, ivan, milancel}@uns.ac.rs
[2]University of Novi Sad, Faculty of Technical Sciences,
Department for Industrial Engineering and Management
Trg Dositeja Obradovića 6
21000 Novi Sad, Serbia
[2]sdristic@uns.ac.rs

Abstract. The inverse referential integrity constraints (IRICs) are specialization of non-key-based inclusion dependencies (INDs). Key-based INDs (referential integrity constraints) may be fully enforced by most current relational database management systems (RDBMSs). On the contrary, non-key-based INDs are completely disregarded by actual RDBMSs, obliging the users to manage them via custom procedures and/or triggers. In this paper we present an approach to the automated implementation of IRICs integrated in the SQL Generator tool that we developed as a part of the IIS*Studio development environment. In the paper the algorithms for insertion, modification and deletion control are presented, alongside with parameterized patterns for their implementation for DBMSs MS SQL Server 2008 and Oracle 10g. It is also given an example of generated procedures/triggers.

Keywords: Inclusion Dependencies, Inverse Referential Integrity Constraint, Declarative Constraint Specification.

1. Introduction

A common approach to database design is to describe the structure and constraints of the Universe of Discourse (UoD) in a semantically rich conceptual data model. The Entity-Relationship (ER) diagrams or the UML (Unified Modelling Language) class diagrams are widely used to represent the conceptual database schemas. The obtained conceptual database (DB) schema is translated latter on into a logical DB schema, representing a design specification of the future database. Such design specification is to be implemented by means of a database management system (DBMS). Contemporary DBMSs are mostly based on the relational or object-relational data models. Therefore, logical DB schemas are still expressed by the concepts of relational data model. Furthermore, logical DB schemas as the design specifications are normally transformed into error free SQL specifications of relational or object-relational DB schemas. In this way, a designed database may be implemented. These SQL specifications are implementations of the structure and constraints of UoD specified in the conceptual DB schema. A goal of this paper is to present an approach to the specification and implementation of a relational integrity constraint type called the **inverse referential integrity constraint** (IRIC).

The most fundamental integrity constraints that arise in practice in relational databases are functional dependencies (FDs) and inclusion dependencies (INDs). There are two basic kinds of INDs: key-based INDs and non-key-based INDs. More often key-based INDs are called referential integrity constraints (RICs). On the contrary, IRICs are a kind of non-key-based INDs. More details about INDs, as well as definitions of different kinds of INDs, including the IRICs, are given in Section 3.

In ER data model or UML class meta-model, cardinality or multiplicity constraints are used, among all, to express the existential dependency between two entity types, i.e. classes. Namely, the existential dependency is modelled by setting the minimal multiplicity to one. Such existential dependency between two entity types in a conceptual DB schema causes an IRIC to be specified in a relational DB schema, as its consequence. More precisely, an IRIC specification in a relational DB schema is caused by a minimal multiplicity set to one, together with the maximal multiplicity set to many on the same side of the association between the two entity types in a conceptual DB schema.

While the referential integrity constraints may be fully enforced by most current relational database management systems (RDBMSs), non-key-based INDs are completely disregarded by actual RDBMSs, obliging the users to manage them via stored program units and triggers. This implies an excessive effort to maintain integrity and develop applications.

There are numerous contemporary software tools aimed at an automated conceptual database schema design and its implementation under different (mostly relational or object-relational) database management systems, such as: DeKlarit, ERwin Data Modeler, Oracle Designer, Power Designer etc. Some of them are described in [7], [14], [24], [28]. All of them enable setting the relationship minimal multiplicity to one. Therefore, they support the specification of the existential dependency between two entity types in the conceptual database schema. However, all of them ignore this specification when generate the SQL code to implement a relational or an object-relational database schema. Even more, to the best of our knowledge, neither of the other CASE tools offers such functionality, as well. As a rule, they do not employ any procedural DBMS mechanisms to provide the automatic implementation of IRICs.

Our approach to the specification and implementation of the IRICs is implemented through the development environment IIS*Studio (IIS*Studio DE, current version 7.1). The development of IIS*Studio DE is spanned through a number of research projects lasting for several years, in which the authors of the paper are actively involved. One of its integral parts is Integrated Information Systems*Case (IIS*Case) – a software tool that supports a model driven approach to information system (IS) design. It supports conceptual modelling of database schemas and generating executable application prototypes. A case study illustrating main features of IIS*Case is given in [19]. Methodological aspects of its usage may be found in [20]. A description of information system design and prototyping using form types is given in [25].

Many commercial CASE tools, e.g. ERwin Data Modeler, Oracle Designer, Power Designer, use ER data model or UML class meta-models to express a conceptual schema. Unlike them, IIS*Case provides a specific platform independent meta-model that does not rely on the ER or UML meta-models. Among the other, this meta-model provides the concepts of form types, component types and their attributes, at the abstraction level of a conceptual DB schema.

The attribute and the form type concepts are explained in details in [19] and [26]. The multiplicity constraints are included in the set of constraints that may be specified by means of form types. IIS*Case uses the set of attributes and the set of form type specifications as the input data for database design to generate logical DB schemas as 3rd normal form (3NF) relational DB schemas with all the relation scheme keys, null value constrains, unique constrains, referential and inverse referential integrity constraints, derived from an IIS*Case conceptual data model. These schemas are stored in the IIS*Case repository. The specification of the IIS*Case repository is given in [25].

In order to provide an efficient transformation of design specifications into error free SQL specifications of relational database schemas we developed the SQL Generator [2]. It is a tool that utilizes SQL, as one of the most common domain-specific languages applied at the level of DB servers. One of the main reasons for the development of such a tool was to make DB designer's and developer's job easier, and particularly to free them from manual coding and testing of SQL scripts for the creation of tables, views, indexes, sequences, procedures, functions and triggers. The SQL Generator implements one transformation in the chain of all IIS*Case transformations from the conceptual model, which is platform independent, towards the executable program code. The input into SQL Generator is a relational database schema, obtained by a transformation of the conceptual DB schema and stored in the repository.

Our SQL Generator implements constraints of the following types: domain constraints, key constraints, unique constraints, tuple constraints, native and extended referential integrity constraints, referential integrity constraints inferred from nontrivial inclusion dependencies, and inverse referential integrity constraints ([18], [23]). Constraints are implemented by the declarative DBMS mechanisms, whenever it is possible. However, the expressiveness of declarative mechanisms of commercial DBMSs may be limited. Therefore, SQL Generator implements a number of constraints through the procedural mechanisms [3]. In this paper we present a feature of SQL Generator that provides an automated implementation of IRICs that are caused by the multiplicity specifications in the IIS*Case conceptual model.

Apart from the Introduction and Conclusion the paper has five sections. Section 2 presents the related work. In Section 3 the notion of an IRIC is explained, illustrated with a real life example to point out the necessity of IRICs implementation. The algorithms for insertion, modification and deletion control in the presence of IRICs are presented in Section 4. In Section 5 we present parameterized patterns of the aforementioned algorithms for DBMSs MS SQL Server 2008 [21] and Oracle 10g [24]. In [4] we introduce patterns for the insertion of mutually blocked tuples via a view created over the relations $r(N_i)$ and $r(N_j)$. Apart from these patterns, here in Section 5, we also present in details patterns for the insertion of mutually blocked tuples via custom db procedures. In Section 6 we present an example of an IRIC design specifications and transformation of design specifications into error free SQL specifications of relational DB schemas by means of IIS*Studio.

2. Related work

Integrity has always been an important issue for database design and implementation. Its importance grows with increasing demands according the quality and reliability of data. Integrity constraint specifications are translated into constraint enforcing mechanisms provided by the DBMS used to implement a database. Most of the commercial DBMSs offer efficient declarative support for the domain constraints, null value constraints, uniqueness constraints and foreign key constraints (key-based IND) [16]. For more complex constraints, using triggers and stored procedures as the procedural mechanisms instead of declarative ones is recommended. Türker and Gertz in [30] emphasize the importance of embedding integrity constraints in the database schema rather then in the application. They state that enforcing integrity constraints and rules identified in the application domain with declarative constraints and/or triggers often is less costly than enforcing the equivalent rules by issuing SQL statements in an application. Preserving of logical data independence is another important reason to embed integrity constraints into database schema. Attaulah and Tompa in [9] stress that the absence of a centralized policy and constraint management system within database systems leads to several problems like the lack of transparency, manageability and compliance of business rules. The approaches presented in [5], [6], [12], [15], [16], [27] and [31] comply with the aforementioned attitudes. We advocate a similar stance and this is an important reason why we develop our SQL Generator to implement the IRICs, besides other integrity constraints.

The growing interest in the Model-Driven Software Development (MDSD) approaches has largely increased the number of tools and methods including code-generation capabilities. Given a platform-independent model (PIM) of an application, these tools generate the application code either by defining an intermediate platform-specific model (PSM) or by executing a direct PIM to code transformation. A conceptual database schema may be seen as a PIM. A transformation of conceptual DB schema into a logical DB schema is a model-to-model (M2M) transformation, while the SQL script generation based on a logical DB schema is a model-to-text (M2T) transformation. Nowadays, almost all tools that support MDSD are able to generate the relational database schemas from PIMs. The major drawback of these tools is that most of them tend to ignore some of the integrity constraints specified in PIMs. Cabot and Teniente in [13] present a survey on the capabilities of current tools regarding the explicit definition of integrity constraints in a PIM and the code generation to enforce them. They classified the different tools in the four categories: CASE tools, MDA (Model-Driven Architecture) specific tools, MDSD tools and OCL (Object Constraint Language) tools. From CASE tools they selected: *Poseidon*, *Rational Rose*, *MagicDraw, Objecteering/UML* and *Together*. In the class of MDA tools *ArcStyler*, *OptimalJ* and *AndroMDA* are evaluated. *OO-Method, WebML* and *Executable UML* are selected beyond MDSD tools, while *Dresden OCL*, *OCLtoSQL*, *OCL2J*, *OCL4Java* and *BoldSoft* are evaluated in the OCL tool class. Most of them do not take the multiplicity constraints into account. The *Objecteering/UML* is an exception to the other tools reviewed in [13], since it allows the use of a trigger system to map the multiplicity constraints, including the minimal multiplicity equal to 1. However, in contrast to our approach, it ignores tuple deletions and updates.

Al-Jumaily, Cuadra and Martinez in [5] present a module to generate triggers for multiplicity constraints verification that is integrated into Rational Rose. In the paper they consider only Oracle DBMS. Although they are tackling similar problem as we are, they are not taking into account mutual dependencies caused by a RIC that exists simultaneously with a considered IRIC. We present the solution of that problem and consider it as the one of the contributions of the paper.

Berrabah and Boufarès in [11] recognize the triggers as a good mean to implement integrity constraints. They distinguish two classes of constraints specified on a UML class diagram: multiplicity and participation constraints. However, furthermore they consider the participation constraints only.

Badaway and Richta in [10] propose an extension to OCL for automatic translation of object level constraints in the modelling language to database level triggers and Zimbrão et al. in [31] proposed a mechanism for translating an OCL constraint to a SQL assertion.

Rybola and Richta in [27] define the multiplicity constraints in a formal way in OCL. They take into consideration both minimal and maximal multiplicity, like we do in our approach. The transformation of OCL specification of constraints into the relational database schema is presented. In the contrast to our approach, the OCL constraints are implemented in SQL as views selecting records violating the multiplicity restrictions. The authors were motivated with the solutions used in the Dresden OCL toolkit.

Some commercial DBMSs supported triggers before they were covered by the SQL-99 standard. Currently, all major relational DBMS vendors have some support for triggers. However, such support may vary from one to the other DBMS, showing typical deviations from the standard [15]. That is the main reason why we present here the implementation of IRICs for two DBMSs: Oracle 10g and MS SQL Server 2008. They are widely used commercial DBMSs. Besides the similarities, there are significant differences between them in the context of trigger mechanisms. We consider that the examples of IRICs implementation for these two platforms may guide practitioners in solving the similar problems. An automated generation of triggers for IRICs implementation may lead towards less error prone solutions compared to handcrafted database trigger.

Summarizing the related work we may say that we have found just a few approaches tackling the problem of automated implementation of IRICs. Some of them use SQL views to select records violating the multiplicity restrictions. Others use a trigger system, but neither of them consider mutual dependencies caused by a RIC that exists simultaneously with a considered IRIC. Because of that, mechanisms for IRIC's validation require deferred trigger consideration during the transaction. Unfortunately, most of the contemporary DBMSs do not support it and solely use the immediate trigger consideration. Oracle and MS SQL Server have different means that may be used to emulate deferred trigger consideration. In our approach we deal with these differences and suggest possible solutions for both of the DBMSs.

3. Inverse Referential Integrity Constraint

Here we give the definitions of IND, key-based IND, non-key-based IND and IRIC.

Let $N_l(R_l, C_l)$ and $N_r(R_r, C_r)$ be two relation schemes, where N_l and N_r are theirs names, R_l and R_r, corresponding sets of attributes, and C_l and C_r corresponding sets of relation scheme constraints. An inclusion dependency is a statement of the form $N_l[LHS] \subseteq N_r[RHS]$, where *LHS* and *RHS* are non-empty arrays of attributes from R_l and R_r respectively. Having the inclusion operator ($\subseteq$) orientated from the left to right we say that relation scheme N_l is on the left-hand side of the IND, while the relation scheme N_r is on its right-hand side. We use the indexes *l* and *r*, and the names of attribute arrays *LHS* and *RHS*, in order to indicate the left and right hand side of IND, respectively. To define a validation rule of IND we use the following notation: (i) the relation $r(N_l)$ is a set of tuples $u(R_l)$ (or just u) satisfying all constraints from the constraint set C_l; (ii) X-value is a projection of a tuple u on the set of attributes X; and (iii) according to the aforementioned orientation of the inclusion operator, $r(N_l)$ is called the referencing relation, while $r(N_r)$ is called the referenced relation. Informally, a database satisfies the inclusion dependency if the set of *LHS*-values in the referencing relation $r(N_l)$ is a subset of the set of *RHS*-values in the referenced relation $r(N_r)$.

There are two basic kinds of INDs: key-based INDs and non-key-based INDs. An IND is said to be key-based if *RHS* is a key[1] of the relation scheme N_r. Otherwise, it is a non-key-based. More often key-based IND is called referential integrity constraint. Non-key-based IND with *LHS* that is a key of the relation scheme N_l, where RIC $N_r[RHS] \subseteq N_l[LHS]$ is specified at the same time, is called inverse referential integrity constraint [22]. In Fig. 1 a UML class diagram is used to visually represent this classification of INDs. The associations between the different classes of INDs are also given. A key-based IND, as well as a non-key-based IND, may be seen as a specialization of IND, while an IRIC may be seen as a specialization of a non-key-based IND.

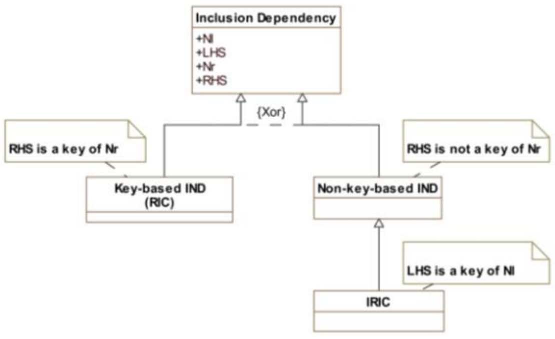


Fig. 1. A classification of different kinds of INDs

Business rules that are to be modelled by the inverse referential integrity constraints often exist in a real world. They are consequences of the mutual existential dependency of the entities of two entity types in a real system.

Example 1. According to the business rules of the university, a department can be established only as a part of a faculty, and a faculty has at least one department. The conceptual database schema expressed by UML class diagram is presented in Fig. 2. The minimal multiplicity of the association *Has* between the class *Faculty* and the class *Department* is one, while the maximal multiplicity is many. After mapping the conceptual DB schema to a

[1] According to [17], a key of a relation scheme *N*(*R*, *C*) is a minimal superkey. Informally, a superkey is any non-empty subset *S* of *R* such that no two distinct tuples in any relation *r*(*N*) can have the same *S*-value. In general, a relation scheme may have more than one key (each of them may be called a candidate key). It is common to designate one of them as the primary key. If a relation scheme has only one key it is, at the same time, the primary key of the relation scheme.

relational DB schema according to the transformation rules suggested in [17] we produce a relational database schema containing two relation schemes: *Faculty* and *Department*, with the keys *FacId* and *FacId+DepId* respectively, and two inclusion dependencies *IND1* and *IND2*:

Faculty({*FacId, FacShortName, FacName, Dean*}, {*FacId*}),
Department({*FacId, DepId, DepName*}, {*FacId+DepId*}),
*IND1***:** *Department* [*FacId*] $\subseteq$ *Faculty* [*FacId*],
*IND2***:** *Faculty*[*FacId*] $\subseteq$ *Department*[*FacId*].

Since that *FacId* is the key of relation scheme *Faculty*, *IND1* is the key-based inclusion dependency, i.e. the referential integrity constraint. It is modelling the business rule that a department can be established only as a part of a faculty. The constraint *IND2* is the non-key-based inclusion dependency, since that *FacId* is not the key of relation scheme *Department*. The *FacId* is the key of the relation scheme *Faculty*, which is on the left-hand side of the inclusion dependency's specification and the referential integrity constraint *IND1* is specified as well. Therefore, the constraint *IND2* is the inverse referential integrity constraint. It is modelling the business rule that faculty must have at least one department. □

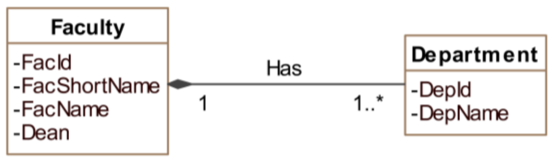


Fig. 2. A university conceptual database schema

Programmers are obliged to manage IRICs via procedural mechanisms (procedures and triggers). That is the reason why the IRICs are mostly implemented at the middle layer instead at the DB server. Still, the validation of the IRICs at the DB server: (i) cuts the costs of the application maintaining; (ii) provides better performances due to the less traffic in the typical client-server architecture; (iii) enables the same way of preventing the violation of a database consistency.

In this paper the methods for the implementation of IRICs, using the mechanisms provided by relational database systems are presented. These methods are implemented in the SQL Generator that provides creating SQL scripts according to the syntax of: (i) ANSI SQL:2003 standard [8], (ii) DBMS Microsoft (MS) SQL Server 2008 with MS T-SQL [21], and (iii) DBMS Oracle 10g with Oracle PL/SQL [24]. In the context of the approach presented in this paper, there are no crucial syntax differences in SQL languages of the DBMSs used in this paper, in comparison to the newer releases of the same DBMSs, i.e. MS SQL Server 2012 and Oracle 11g, respectively. Therefore, considerations given in this paper may be applied also at these DBMSs, without any limits.

4. Algorithms for IRIC Validation

By specifying the IRIC $N_j[Y] \subseteq N_i[X]$ it comes towards the bogus mutual insertion blocking of the instances of the relation schemes N_i and N_j, since the RIC $N_i[X] \subseteq N_j[Y]$ is also specified. The notion „mutual insertion blocking“ is used to illustrate the following situation: (i) it is not possible to insert a new tuple into the relation $r(N_i)$ with not null values for all attributes $A \in X$, unless there is a tuple in the relation $r(N_j)$ with the Y value same as the X value of the inserted tuple (due to the RIC $N_i[X] \subseteq N_j[Y]$); and, also (ii) it is not possible to insert a new tuple into the relation $r(N_j)$ with a Y value given, unless there is a tuple in the relation $r(N_i)$ with the X value same as the aforementioned Y value (due to the IRIC $N_j[Y] \subseteq N_i[X]$) [23].

Example 2. In Fig. 3 it is presented a database instance of the database schema from Example 1. Due to the existence of the referential integrity *IND1* it is not possible to insert the tuple (2, D2, ’Dentistry’) into the relation *Department*. However, due to the specified inverse referential integrity *IND2* it is even not possible to insert the tuple (2, ’FOM’, ’Faculty of Medicine’, ’Simpson’) into the relation *Faculty*. These tuples are said to be mutually blocked. □

Faculty

FacId	*FacShortName*	*FacName*	*Dean*
1	MAT	Mathematics	Smith

Department

FacId	*DepId*	*DepName*
1	D1	Geometry

Fig. 3. A University database instance

The algorithms for insertion, deletion and modification control in the presence of inverse referential integrity constraints are presented in Fig. 4, Fig. 5 and Fig. 6, respectively.

Apart from the notation already introduced at the beginning of Section 3, in the algorithms we use the following notation: $u[X]$ denotes X-value of a tuple u, $|X|$ denotes the cardinality of an array of attributes X, ω denotes the null value, and $K_p(R_i)$ denotes the primary key of a relation scheme N_i.

In the following text these algorithms are described in more details.

Let $N_j[Y] \subseteq N_i[X]$ is an IRIC. In the context of the IRIC, $r(N_j)$ is the referencing relation, while $r(N_i)$ is the referenced relation, since the relation scheme N_j is on the left-hand side and the relation scheme N_i is on the right-hand side of the IRIC. The IRIC may be violated in three cases: (i) when a tuple is inserted into the referencing relation, (ii) when a tuple is deleted from the referenced relation or (iii) when a tuple's X-value is modified in the referenced relation.

Trigger:	INSERTION CONTROL IN THE PRESENCE OF IRICs

Definition area:
Relation schemes: N_i, N_j
Attributes: $X = (A_1, ..., A_{|X|})$, $Y = (B_1, ..., B_{|Y|})$
$|X| = |Y| \wedge (\forall l \in \{1, \ldots, |X|\})(dom(A_l) \subseteq dom(B_l) \wedge A_l \in R_i \wedge B_l \in R_j)$

Specification of the constraint: i: $N_j[Y] \subseteq N_i[X]$

Specification of the operation:
Time: AFTER OPERATION
Operation: INSERT

Data Inputs

From DB	$r(N_i)$, $r(N_j)$
Input	v: tuple that would be inserted into $r(N_j)$

Local declarations:*ind*
(*ind* = 1 – constraint is satisfied,
ind = 0 – constraint is violated)

Pseudo code:
```
BEGIN PROCESS  Insert_inv_ref_int
  SET  $ind \leftarrow 0$
  FOR ALL $u \in r(N_i)$ DO          // Search in the relation $r(N_i)$ for $v[Y]$ value
    IF  $v[Y] = u[X]$  THEN
      SET  $ind \leftarrow 1$
      BREAK
    ENDIF
  ENDFOR
  IF  $ind = 0$  THEN
    CANCEL_OPERATION('Error description')
  ENDIF
ENDPROCESS  Insert_inv_ref_int
```

Fig. 4. An algorithm for insertion control

An algorithm for the control of insertions (Fig. 4) will reject the insert operation of the *v* tuple into the referencing relation if the referenced relation doesn't contain any tuple with *X*-value matching the *Y*-value of the tuple *v*.

Trigger:	DELETION CONTROL IN THE PRESENCE OF IRICs

Definition area:
Relation schemes: N_i, N_j
Attributes: $X = (A_1, ..., A_{|X|})$, $Y = (B_1, ..., B_{|Y|})$
$|X| = |Y| \wedge (\forall l \in \{1, \ldots, |X|\})(dom(A_l) \subseteq dom(B_l) \wedge A_l \in R_i \wedge B_l \in R_j)$

Specification of the constraint: i: $N_j[Y] \subseteq N_i[X]$

Specification of the operation:
Time: AFTER OPERATION
Operation: DELETE

Data Inputs	
From DB	$r(N_i)$, $r(N_j)$
Input	u: tuple that would be deleted from $r(N_i)$

Local declarations:*ind*
(*ind* = 1 – constraint is satisfied,
ind = 0 – constraint is violated)

Pseudo code:

```
BEGIN PROCESS  Delete_inv_ref_int
  SET $ind \leftarrow 0$
  FOR ALL $A \in X$ DO          // Search for null value in $X$-value
    IF $u[A] = \omega$ THEN
      SET $ind \leftarrow 1$
      BREAK
    ENDIF
  ENDFOR
  IF $ind = 0$ THEN
    FOR ALL $t \in r(N_i)$ DO       // Search in $r(N_i)$
      IF $t[K_p(R_i)] \neq u[K_p(R_i)] \wedge u[X] = t[X]$  THEN
        SET $ind \leftarrow 1$
        BREAK
      ENDIF
    ENDFOR
  ENDIF
  IF $ind = 0$ THEN
    EXECUTE ACTIVITY
  ENDIF
ENDPROCESS Delete_inv_ref_int
```

Fig. 5. An algorithm for deletion control

Trigger:	MODIFICATION CONTROL IN THE PRESENCE OF IRICs
Definition area: **Relation schemes**: N_i, N_j **Attributes:** $X = (A_1, ..., A_{\lvert X\rvert})$, $Y = (B_1, ..., B_{\lvert Y\rvert})$ $\lvert X\rvert = \lvert Y\rvert \wedge (\forall l \in \{1, \dots, \lvert X\rvert\})(dom(A_l) \subseteq dom(B_l) \wedge A_l \in R_i \wedge B_l \in R_j)$	
Specification of the constraint: i: $N_j[Y] \subseteq N_i[X]$	
Specification of the operation: **Time**: AFTER OPERATION **Operation**: UPDATE	
Data Inputs	
From DB	$r(N_i)$, $r(N_j)$
Input	u: original tuple to be modified in $r(N_i)$ u': new tuple obtained by the modification of tuple u
Local declarations:*ind* (*ind* = 1 – constraint is satisfied, *ind* = 0 – constraint is violated)	

Pseudo code:

```
BEGIN PROCESS  Update_inv_ref_int
  IF  $u'[X] \neq u[X]$  THEN
    SET  $ind \leftarrow 0$
    FOR ALL $A \in X$  DO        // Search for null value in X-value
        IF $u[A] = \omega$ THEN
         SET $ind \leftarrow 1$
          BREAK
       ENDIF
    ENDFOR
    IF $ind = 0$ THEN
      FOR ALL $t \in r(N_i)$ DO    // Search in $r(N_i)$
         IF $t[K_p(R_i)] \neq u[K_p(R_i)] \wedge u[X] = t[X]$  THEN
           SET $ind \leftarrow 1$
            BREAK
         ENDIF
        ENDFOR
    ENDIF
    IF $ind = 0$ THEN
      CANCEL_OPERATION('Error description')
    ENDIF
  ENDIF
ENDPROCESS Update_inv_ref_int
```

Fig. 6. An algorithm for modification control

An algorithm for the control of deletions (Fig. 5) detects an IRIC's violation when a tuple *u* from the referenced relation is deleted and if the conjunction of conditions is satisfied: (i) *X*-value of the tuple *u* doesn't contain null values; and (ii) the referenced relation doesn't contain another tuple *t* (strictly different from the tuple *u*) with *X*-value matching the *X*-value of the tuple *u*. The first condition needs additional explanation. Namely, *Y* is the key for the left-hand side relation scheme. Consequently, neither of the tuples from the referencing relation can contain null value in the *Y*-value sequence. Therefore, neither of the tuples from the referenced relation that contains null values can be referenced by some tuple from referencing relation. It may be concluded that by the deletion of such a tuple from $r(N_i)$, IRIC cannot be violated. If a constraint violation is detected, the algorithm will reject the delete operation or, alternatively it will delete all tuples from the referencing relation having the *Y*-value matching the *X*-value of the tuple *u*. During the IRIC implementation pseudo-instruction *EXECUTE ACTIVITY* will be replaced with an appropriate program code for the selected action.

An algorithm for the control of modifications (Fig. 6) will reject the update operation of the tuple *u* from the referenced relation if the conjunction of conditions is satisfied: (i) the update operation changes the tuple's *X*-value ($u'[X] \neq u[X]$, where *u* is the tuple before the modification and *u'* is the tuple after the modification); (ii) the original *X*-value (*X*-value of the tuple *u* before the modification) doesn't contain null values; and (iii) the referenced relation doesn't contain any other tuple *t* (strictly different from the original tuple *u*) with *X*-value matching the original *X*-value. The explanation for the second condition is analogous to the explanation for the first condition in the previous paragraph.

5. Implementation of IRICs by Procedural Mechanisms

DBMSs have different mechanisms for the implementation of relational database constraints (RDBCs). We are going to classify these mechanisms into two categories. The *core* mechanisms use the CONSTRAINT clause within the CREATE / ALTER TABLE statements of SQL, to implement a RDBC. The *additional* mechanisms use the CREATE ASSERTION statement or the CREATE TRIGGER statement to implement a RDBC. The fundamental mechanisms are declarative, while the additional mechanisms may be declarative (e.g. assertions) or procedural (e.g. triggers). An IRIC cannot be implemented by means of core mechanisms of contemporary DBMSs. Therefore, it has to be implemented via declarative assertions or procedural triggers. Albeit SQL standards allow assertions, most of the contemporary DBMSs do not support them. Therefore, we have to implement the IRICs via DBMS procedural mechanisms, by creating triggers, alongside with the required functions and procedures. Another problem to be solved occurs due to a mutual insertion blocking, caused by an IRIC specification. Because of that, mechanisms for IRIC's validation require deferred trigger consideration during the transaction. Some DBMSs support the deferred constraint consideration. Unfortunately, most of the contemporary DBMSs do not support deferred trigger consideration and provide the immediate trigger consideration only.

Our SQL Generator enables an automated implementation of the IRICs for DBMSs MS SQL Server 2008 [21] and Oracle 10g [24]. One of the reasons for their selection is that they are widely used commercial DBMSs. Another reason is that besides the similarities, there are significant differences between them.

In the context of IRICs implementation the main similarities are that: (i) both of them do not support assertions and deferred trigger consideration; and (ii) there are many similarities concerning trigger specification. The major difference in the same context is that Oracle and MS SQL Server have different means that may be used to emulate deferred trigger consideration. Oracle enables global variables declaration in packages. We can use them to pass the information that a trigger has to skip an IRIC checking. The global variables can't be declared in MS SQL Server. Instead, we use tuple in auxiliary table to pass the information that a trigger has to skip an IRIC checking. In this section we will illustrate the differences between IRICs' implementation techniques for MS SQL Server 2008 and Oracle 10g, used in our SQL Generator.

In our approach the procedural implementation of a constraint, can be unified. It consists of the following steps: (i) specifying a parameterized pattern of the algorithm for a specific DBMS, (ii) replacing the pattern parameters with real values, and (iii) generating an SQL script comprising necessary triggers, procedures and functions [1].

5.1. IRIC Implementation for MS SQL Server 2008

In this section, parameterized patterns of the algorithms for controlling the IRIC validation during the insert, update and delete operations for DBMS MS SQL Server 2008 (MS SQL) are given.

5.1.1. Patterns for tuple insertion in the presence of an IRIC

In order to keep the DB consistency in the presence of the IRICs, mutually blocked tuples (like those in Example 2) must be inserted in one transaction. There are two ways to do that: (i) a view created over the relations $r(N_i)$ and $r(N_j)$ may be used for the double insertion; or (ii) a custom DB procedure for double insertion may be developed.

The pattern of the trigger using views for tuple insertion is presented in Fig. 7. For each specified IRIC $N_j[Y] \subseteq N_i[X]$ the trigger based on that pattern would be generated. We use a special MS SQL Server table, named *Inserted*, that stores copies of the affected tuples during the execution of INSERT and UPDATE statements. The values of attributes from R_j and R_i are separated and two INSERT statements are specified for tuple insertion into relations $r(N_j)$ and $r(N_i)$, respectively. Since these tuples are mutually blocked, a trigger for tuple insertion in $r(N_j)$ (Fig. 10), will raise the error. To prevent that, we emulate the deferred trigger consideration, using an auxiliary DB relation *Trigger_Stat*. The tuple with given trigger name and transaction ID is to be written in the *Trigger_Stat* relation, by calling the procedure *Trigger_Ex* (Fig. 8) with 0 as the first argument. This tuple is aimed to pass the information that the trigger for tuple insertion in $r(N_j)$ (Fig. 10) has to skip an IRIC checking. In that way we emulate the deferred trigger consideration. Thus, the tuple insertion in $r(N_j)$ is enabled. Afterwards, the insertion of corresponding tuple in $r(N_i)$ is allowed, since it does not violate the RIC $N_i[X] \subseteq N_j[Y]$ any more. The next step is to re-enable IRIC checking in the trigger for tuple insertion in $r(N_j)$ (Fig. 10). In order to do it, the previously inserted tuple in the *Trigger_Stat* relation, with given trigger name and transaction ID, is to be deleted, by calling the procedure *Trigger_Ex* (Fig. 8) with 1 as the first argument. At the end, the function *ContainmentIRI_*<N_j> (Fig. 9) is called in order to check if the IRIC $N_j[Y] \subseteq N_i[X]$ is violated.

```
CREATE TRIGGER TRG_<Const_Name>_View
ON View_<Nj>_<Ni> INSTEAD OF INSERT
AS
  DECLARE @Idt int, @Count int, <Decl_Var_For_Ni_Nj>
  SELECT <Var_array_For_Ni_Nj> FROM Inserted
  SET @Idt = @@SPID
  exec dbo.Trigger_Ex 0, 'WriteRI_<Nj>', @Idt
  INSERT INTO <Nj> VALUES (<Var_array_For_Nj>)
  INSERT INTO <Ni> VALUES (<Var_array_For_Ni>)
  exec dbo.Trigger_Ex 1, 'WriteRI_<Nj>', @Idt
  IF dbo.ContainmentIRI_<Nj>(<Var_For_Y>) = 0
  BEGIN
     RAISERROR('IRIC violation!',16,1)
     ROLLBACK TRAN
  END
```

Fig. 7. A pattern of the trigger over view

The trigger generator creates the trigger from the pattern presented in Fig. 7. by replacing:

- <N_j> with the name of relation scheme N_j;
- <N_i> with the name of relation scheme N_i;
- <*Const_Name*> with IRI_<N_j>_<N_i>, where IRI marks that it is the inverse referential integrity constraint, and <N_j> and <N_i> will be replaced with the names of relation schemes N_j and N_i respectively;
- <*Decl_Var_For_Ni_Nj*> with the list of variable declarations of the form *@<Attribute_Name> data type*, for each attribute from R_i and R_j;
- <*Var_array_For_Ni_Nj*> with the list of variables declared by the list of declarations that replaced <*Decl_Var_For_Ni_Nj*>. These variables are

set to appropriate values from a tuple contained in the MS SQL Server table *Inserted*;

- <*Var_array_For_Nj*> and <*Var_array_For_Ni*> with the lists of variables containing the input value for each attribute from R_j and R_i, respectively; and
- <*Var_For_Y*> with the list of variables declared by the list of declarations that replaced <*Decl_Var_For_Ni_Nj*>, containing only those variables that are related to the attributes from *Y*. List elements are of the form @<*Name_of_Attribute_From_Y*>. The list of variables represents the argument of function *ContainmentIRI_*<N_j>.

The procedure *Trigger_Ex* and the pattern of the function *ContainmentIRI_*<N_j>, that are called from a trigger based on the pattern in Fig. 7, are presented in Fig. 8 and Fig. 9 respectively.

```
CREATE PROCEDURE dbo.Trigger_Ex
(@Stat int, @Trigger_Name varchar(50), @Idt int)
AS
  IF @Stat = 1
    DELETE FROM Trigger_Stat WHERE
    Trigger = @Trigger_Name AND IdTransaction = @Idt
  ELSE
    INSERT INTO Trigger_Stat (Trigger, IdTransaction)
    VALUES (@Trigger_Name, @Idt)
```

Fig. 8. A SQL procedure for trigger execution control

The procedure *Trigger_Ex* in Fig. 8 is used in the process of trigger's execution control. In the suggested solution an auxiliary DB relation *Trigger_Stat* is used, as we already explained, earlier in this section.

```
CREATE FUNCTION dbo.ContainmentIRI_<Nj>(<Decl_Var_For_Y>)
RETURNS int
AS
  BEGIN
    DECLARE @Count int, @Ret  int
    SELECT @Count = COUNT(*) FROM <Ni> u
    WHERE (<Selection_Cond>)
    IF @Count != 0
      SELECT @Ret =1
    ELSE
      SELECT @Ret =0
    RETURN @Ret
  END
```

Fig. 9. A pattern of the *ContainmentIRI_*<N_j> function

A DB function *ContainmentIRI_*<N_j> is to be called from a trigger based on the pattern in Fig. 7. It is generated from the function pattern in Fig. 9, by replacing:

- <N_j> with the name of relation scheme N_j;

- <*Decl_Var_For_Y*> with the list of parameter declarations of the form @<*Name_of_Attribute_From _Y*> *data type*, for each attribute from *Y*; and
- <*Selection_Cond*> with a conjunction of relational expressions of the form:
  u.<*Name_of_Attribute_From _X*> = @<*Name_of_Attribute_From _Y*>.

SQL code for view creation is trivial, and therefore it is omitted here. We only emphasize that it should contain all attributes from both R_i and R_j.

In order to prevent the IRIC violation due to the separate insertion of mutually blocked tuples, a trigger adhering the pattern in Fig. 10 is created. The replacement of the parameters during the trigger generation is analogous to the replacement of corresponding parameters in patterns from figures 7, 8 and 9.

Finally, the SQL function for trigger execution is presented in Fig. 11. This function is aimed to detect if there is a tuple, with given trigger name and transaction ID, in the auxiliary table *Trigger_Stat*. It is called from a trigger for tuple insertion in $r(N_j)$ (Fig. 10). The return value 0 (a tuple exists) indicates that IRIC check is to be done, while the return value 1 (a tuple doesn't exist) indicates that it is to be skipped.

```
CREATE TRIGGER TRG_<Nj>_<Const_Name>_INS
ON <Nj> FOR INSERT
AS
  IF dbo.ExecuteTrigger(TRG_<Nj>_<Const_Name>_INS)=0
  BEGIN
    RAISERROR('Data have to be inserted via view:
        View_<Nj>_<Ni> or procedure Insert_<Const_Name>',16,1)
    ROLLBACK TRAN
  END
```

Fig. 10. A tuple insertion control pattern

```
CREATE FUNCTION dbo.ExecuteTrigger(@Trigger_Name varchar(50))
RETURNS int
AS
  BEGIN
    DECLARE @Count int, @Idt  int, @Ret int
    SELECT @Idt = @@SPID
    SELECT @Count = COUNT(*) FROM Trigger_Stat
    WHERE (Trigger = @Trigger_Name) AND (IdTransaction = @Idt)
    IF @Count != 0
      SELECT @Ret =1
    ELSE
      SELECT @Ret =0
    RETURN @Ret
  END
```

Fig. 11. A SQL function for trigger execution control

```
CREATE PROCEDURE dbo.Insert_<Const_Name>
(<Decl_Var_For_Ni_Nj>)
AS
  DECLARE @Idt int
  BEGIN TRANSACTION
    SET @Idt = @@SPID
    exec dbo. Trigger_Ex 0, 'WriteRI_<Nj>', @Idt
    INSERT INTO <Nj> VALUES (<Var_array_For_Nj>)
    INSERT INTO <Ni> VALUES (<Var_array_For_Ni>)
    exec dbo.Trigger_Ex 1, 'WriteRI_<Nj>', @Idt
    IF dbo.ContainmentIRI_<Nj>(<Var_For_Y>) = 0
    BEGIN
      RAISERROR('IRIC violation!',16,1)
      ROLLBACK TRAN
    END
  COMMIT TRANSACTION
```

Fig. 12. A pattern of the procedure for tuple insertion

Another way for providing insertion of mutually blocked tuples in one transaction is by creating a custom DB procedure for double insertion. Parameterized pattern for such a procedure is given in Fig. 12. The meaning of parameters is similar to that in the pattern of the trigger for double insertion using views, and therefore is omitted here.

5.1.2. Patterns for tuple deletion in the presence of an IRIC

The pattern of the trigger for tuple deletion is presented in Fig. 13. For each specified IRIC $N_j[Y] \subseteq N_i[X]$ the trigger based on that pattern would be generated. The trigger generator creates the trigger from the pattern by replacing:

- <N_i> with the name of relation scheme N_i;
- <*Const_Name*> with IRI_<N_j>_<N_i>, where IRI marks that it is the inverse referential integrity constraint, and <N_j> and <N_i> will be replaced with the names of relation schemes N_j and N_i respectively;
- <*Decl_Var_For_X*> with the list of variable declarations of the form @<*Attribute_Name*> *data type*, for each attribute from *X*;
- <*Attr_from_X*> with the list containing the names of attributes from *X*;
- <*Condition*> with the conjunction of relational expressions of the form:

  @<Name_of_Attribute_From _X> IS NOT NULL;

- <*Selection_Cond*> with the conjunction of relational expressions of the form:

  u.<Name_of_Attribute_From_X> = *@<Name_of_Attribute_From_X>*; and

- <*Var_For_X*> with the list of variables' names declared by the list of declarations that replaced <*Decl_Var_For_X*>. List elements are of the form @<*Name_of_Attribute_From_X*>.

```
CREATE TRIGGER TRG_<Ni>_<Const_Name>_DEL
ON <Ni> FOR DELETE
AS
  DECLARE @Count int, <Decl_Var_For_X>
  DECLARE Cursor_<Ni> CURSOR
  FOR SELECT <Attr_From_X> FROM Deleted
  OPEN Cursor_<Ni>
  FETCH NEXT FROM Cursor_<Ni> INTO <Var_For_X>
  WHILE @@FETCH_STATUS=0
  BEGIN
    IF <Condition>
    BEGIN
      SELECT @Count = COUNT(*) FROM <Ni> u
      WHERE (<Selection_Cond>)
      IF @Count = 0
         <Execute_Activity>
    END
    FETCH NEXT FROM Cursor_<Ni> INTO <Var_For_X>
  END
  CLOSE Cursor_<Ni>
  DEALLOCATE Cursor_<Ni>
```

Fig. 13. A pattern of the delete trigger

Deleted table, used in a declaration of cursor *Cursor_*<N_i>, is a special MS SQL Server table that stores copies of the affected tuples during the execution of DELETE and UPDATE statements. Depending on the selected activity, <*Execute_Activity*> is replaced with *CascadeIRI_Del_*<N_i> (Fig. 14) procedure call (Cascade delete) or with SQL code for activity restriction (Fig. 15).

The procedure generator creates the procedure for cascade deletion (Fig. 14) from the pattern by replacing:

- <N_i> with the name of relation scheme N_i;
- <*Decl_Var_For_X*> with the list of variable declarations of the form @<*Attribute_Name*> *data type*, for each attribute from *X*; and
- <*Selection_Cond*> with the conjunction of relational expressions: *v.*<*Name_of_Attribute_From_Y*> = @<*Name_of_Attribute_From_X*>.

```
CREATE PROCEDURE dbo.CascadeIRI_Del_<Ni>(<Decl_Var_For_X>)
AS
  DELETE FROM <Nj> v WHERE (<Selection_Cond>)
```

Fig. 14. A pattern of procedure for cascade deletion

```
BEGIN
  RAISERROR('The tuple from relation <Ni> could not be deleted',16,1)
  ROLLBACK TRAN
END
```

Fig. 15. A pattern of SQL code for operation restriction

5.1.3. Patterns for tuple modification in the presence of an IRIC

The pattern of the trigger for tuple modification for MS SQL Server is presented in Fig. 16.

```
CREATE TRIGGER TRG_<Ni>_<Const_Name>_UPD
ON <Ni> FOR UPDATE
AS
  DECLARE @Count int, <Decl_Var_For_X>
  IF <Modification_Cond>
  BEGIN
    DECLARE Cursor_<Ni> CURSOR
    FOR SELECT <Attr_From_X> FROM Deleted
    OPEN Cursor_<Ni>
    FETCH NEXT FROM Cursor_<Ni> INTO <Var_For_X>
    WHILE @@FETCH_STATUS=0
    BEGIN
      IF <Condition>
      BEGIN
        SELECT @Count = COUNT(*) FROM <Ni> u
        WHERE (<Selection_Cond>)
        IF @Count = 0
        BEGIN
          RAISERROR('The tuple from relation <Ni> could not be updated',16,1)
          ROLLBACK TRAN
        END
      END
      FETCH NEXT FROM Cursor_<Ni> INTO <Var_For_X>
    END
    CLOSE Cursor_<Ni>
    DEALLOCATE Cursor_<Ni>
  END
```

Fig. 16. A pattern of the modification trigger

The replacement of the parameters is same as the replacement of parameters for the deletion trigger. The modification trigger has one more parameter, <*Modification_Cond*>. During the trigger generation it is replaced by the disjunction of SQL functions UPDATE(<*Name_of_Attribute_from_X*>) for each attribute belonging to the attribute set *X*.

5.2. IRIC Implementation for Oracle 10g

Existence of the SQL standard may be considered as one of the major reasons for the commercial success of relational databases. The RDBMSs' vendors make efforts to achieve high SQL standard compliance. Despite this, in practice, there are many differences between various RDBMSs. In the context of IRICs implementation, the differences concerning the means that may be used to emulate deferred trigger consideration are crucial. The global variables can’t be declared in MS SQL Server. Instead, we use a tuple in auxiliary table to pass the information that a trigger has to skip an IRIC checking, as it is shown in Section 5.1. Oracle DBMS enables global variables declaration in packages. They can be used to pass the information that a trigger has to skip an IRIC checking. Therefore, here we present the parameterized patterns for triggers and procedures implementing algorithms from Section 4, for Oracle DBMS. Some of parameters in the patterns for Oracle DBMS are same as the parameters in the corresponding patterns for MS SQL Server. The explanation of their replacement will be omitted in the following text. The replacement of the parameters those are specific for patterns for Oracle Server will be explained in details.

5.2.1. Patterns for tuple insertion in the presence of an IRIC

As well as for MS SQL Server, there are two ways to insert mutually blocked tuples in one transaction: (i) a view created over the relations $r(N_i)$ and $r(N_j)$ may be used for the double insertion; or (ii) a custom DB procedure for double insertion may be developed. The pattern of the trigger using views for tuple insertion is presented in Fig. 17.

Here we notify the basic differences between the patterns for tuple insertion in the presence of an IRIC for MS SQL Server and Oracle. The MS SQL Server has the *Inserted* table that stores copies of the affected tuples during the execution of INSERT and UPDATE statements. In Oracle notation key-words NEW and OLD are used for that purpose. NEW is used to refer to a newly inserted or newly updated tuple. OLD is used to refer to a deleted tuple or to a tuple before it was updated. The values of attributes from R_j and R_i are separated and two INSERT statements are specified for tuple insertion into relations $r(N_j)$ and $r(N_i)$, respectively. Since these tuples are mutually blocked, a trigger for tuple insertion in $r(N_j)$ (Fig. 20), will raise an error. To prevent that, we need to emulate the deferred trigger consideration. Unlike MS SQL Server, Oracle enables global variables declaration in packages. So, in a package (Fig. 18) the global variable *Trigger_Ex* is declared. Before the first INSERT statement in Fig. 17, the *Trigger_Ex* is set to FALSE, indicating that a trigger for tuple insertion in $r(N_j)$ (Fig. 20) has to skip an IRIC checking. In that way we emulate the deferred trigger consideration. Thus, the tuple insertion in $r(N_j)$ is enabled, without raising an application error. Afterwards, the insertion of corresponding tuple in $r(N_i)$ is allowed, since it does not violate the RIC $N_i[X] \subseteq N_j[Y]$ any more. The next step is to re-enable IRIC checking in the trigger for tuple insertion in $r(N_j)$ (Fig. 20). In order to do it in Oracle, the *Trigger_Ex* is set to TRUE, indicating that a trigger for tuple insertion in $r(N_j)$ (Fig. 20) has to enforce an IRIC checking. At the end, the function *ContainmentIRI_*<N_j> (Fig. 19) is called in order to check if the IRIC $N_j[Y] \subseteq N_i[X]$ is violated.

```
CREATE OR REPLACE TRIGGER TRG_<Const_Name>_View
INSTEAD OF INSERT ON View_<Nj>_<Ni>
FOR EACH ROW
DECLARE
  I NUMBER;
  Exc EXCEPTION;
  t <Nj>%ROWTYPE;
BEGIN
  SELECT COUNT(*) INTO I FROM <Ni> WHERE (<Selection_Cond>);
  IF I <> 0 THEN
    INSERT INTO <Ni> VALUES (<Attr_Value_From_Nj>);
  ELSE
    <Const_Name>_PCK.Trigger_Ex := FALSE;
    INSERT INTO <Nj> VALUES (<Attr_Value_From_Nj>);
    INSERT INTO <Ni> VALUES (<Attr_Value_From_Ni>);
    <Const_Name>_PCK.Trigger_Ex := TRUE;
    SELECT * INTO t
    FROM <Nj> WHERE (<Selection_Cond>);
    IF NOT Global_PCK.ContainmentIRI_<Nj>(t) THEN
      RAISE Exc;
    END IF;
  END IF:
  EXCEPTION WHEN Exc THEN
    RAISE_APPLICATION_ERROR (-20001,'IRIC violation!');
END;
```

Fig. 17. A pattern of the trigger over view for Oracle

The replacement of parameters, specific for Oracle patterns, during the trigger generation is done as follows:

- <*Selection_Cond*> is replaced by the conjunction of relational expressions (one expression per each attribute from *Y*) of the form:

<*Name_of_Attribute_From _Y*> = :NEW.<*Name_of_Attribute_From _Y*>;

- <*Attr_Value_From_Ni*> (<*Attr_Value_From_Nj*>) is replaced by the list of elements (one element per each attribute from R_i or R_j) of the form:

:NEW.<*Name_of_Attribute_From _Ni*>

(:NEW.<*Name_of_Attribute_From _Nj*>).

Trigger_Ex is a global variable defined in a package created for the appropriate constraint. The variable gets value TRUE if the trigger ought to be executed and gets value FALSE otherwise. The parameterized content of that package is presented in Fig. 18. The package parameter <*Attr_Decl_Rec_X*> is replaced with the list of elements of the form:

<N_i>.<*Name_of_Attribute_From_X*>*%TYPE*.

*For_*<N_i> and *Count_IRI* are variables declared in the package presented in Fig. 18. They are used in modification and deletion triggers, and will be explained in Section 5.2.2.

```
CREATE OR REPLACE PACKAGE <Const_Name>_PCK
IS
  TYPE TRec<Ni> IS RECORD (<Attr_Decl_Rec_X>);
  TYPE TTabForDelUpd IS TABLE OF TRec<Ni> INDEX BY BINARY_INTEGER;
  For_<Ni> TTabForDelUpd;
  Count_IRI NUMBER(8,0);
  Trigger_Ex BOOLEAN;
END;
```

Fig. 18. A pattern of IRIC's package

```
FUNCTION ContainmentIRI_<Nj> (v IN <Nj>%ROWTYPE)
RETURN BOOLEAN
IS
  I  NUMBER;
BEGIN
  SELECT COUNT(*) INTO I FROM <Ni> u
  WHERE (<Selection_Cond>);
  IF I <> 0 THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END;
```

Fig. 19. A pattern of the *ContainmentIRI_*<N_j> function

The pattern of the DB function *ContainmentIRI_*<N_j>, called from *TRG_*<*Const_Name*>*_View* trigger (Fig. 17) is shown in Fig. 19. The function is to be defined in global package *Global_PCK*. During the function generation process the parameter <*Selection_Cond*> is replaced by the conjunction of relational expressions (one expression per each attribute from *Y*) of the form:

u.<*Name_of_Attribute_From_X*> = *v*.<*Name_of_Attribute_From_Y*>.

In order to prevent the IRIC violation due to the separate insertion of mutually blocked tuples, a trigger adhering to the pattern in Fig. 20 is to be created.

```
CREATE OR REPLACE TRIGGER TRG_<Nj>_<Const_Name>_INS
BEFORE INSERT ON <Nj> FOR EACH ROW
BEGIN
  IF <Const_Name>_PCK.Trigger_Ex = TRUE THEN
    RAISE_APPLICATION_ERROR(-20004, 'Data have to
       be inserted via view:View_<Nj>_<Ni> or procedure Insert_<Const_Name>');
  END IF;
END;
```

Fig. 20. A tuple insertion control pattern

```
CREATE OR REPLACE PROCEDURE Insert_<Const_Name>
(v IN <Nj>%ROWTYPE, u IN <Ni>%ROWTYPE)
IS
  t <Nj>%ROWTYPE;
  Exc EXCEPTION;
BEGIN
  <Const_Name>_PCK.Trigger_Ex := FALSE;
  INSERT INTO <Nj> VALUES (<Attr_Value_From_Nj>);
  INSERT INTO <Ni> VALUES (<Attr_Value_From_Ni>);
  <Const_Name>_PCK.Trigger_Ex := TRUE;
   SELECT * INTO t
  FROM <Nj> WHERE (<Selection_Cond>);
  IF NOT Global_PCK.ContainmentIRI_<Nj>(t) THEN
     RAISE Exc;
  END IF;
  EXCEPTION WHEN Exc THEN
    RAISE_APPLICATION_ERROR (-20001,'IRIC violation!');
END;
```

Fig. 21. A pattern of the procedure for mutually blocked tuples insertion

Another way for providing insertion of mutually blocked tuples in one transaction is by creating a custom DB procedure for double insertion. Parameterized pattern for such a procedure is given in Fig. 21. The differences between a procedure for mutually blocked tuples insertion for MS SQL Server (see Fig. 12) and appropriate procedure for Oracle (see Fig. 21), are the same as the differences described at the beginning of this section (see Fig. 7 and Fig. 17). The meaning of parameters is similar to that in the pattern of the trigger for double insertion using views, and therefore is omitted here.

5.2.2. Patterns for tuple deletion in the presence of an IRIC for Oracle

Here we notify basic differences between the patterns for tuple deletion in the presence of an IRIC for MS SQL Server and Oracle. MS SQL Server provides the *Deleted* table that stores copies of the affected tuples during the execution of DELETE and UPDATE statements. In Oracle notation key-words

NEW and OLD are used for that purpose. Hereof, for Oracle 10g three triggers are to be created for the implementation of tuple deletion under the presence of IRICs. The first one is run at the statement level, before the tuple deletion. It has an assignment to set the auxiliary data structures, used by other triggers. The pattern for the first trigger is shown in Fig. 22.

```
CREATE OR REPLACE TRIGGER TRG_<Ni>_<Const_Name>_DEL1
BEFORE DELETE <Ni>
BEGIN
  <Const_Name>_PCK.Count_IRI := 0;
  <Const_Name>_PCK.For_<Ni>.DELETE;
END;
```

Fig. 22. A pattern of the first delete trigger

The variable *For_<N$_i$>* enables transfer of old values of attributes belonging to the attribute set *X* for all tuples that would be deleted. The variable *Count_IRI* is aimed to keep the number of tuples that would be deleted. Both of them are declared in the package presented in Fig. 18 and represent auxiliary data structures.

The second trigger is run just before the tuple deletion. It puts the attribute values from the tuple that would be deleted into the previously declared auxiliary data structures. The pattern for the second trigger is presented in Fig. 23.

```
CREATE OR REPLACE TRIGGER TRG_<Ni>_<Const_Name>_DEL2
BEFORE DELETE ON <Ni>
FOR EACH ROW
DECLARE
  u <Ni>%ROWTYPE;
BEGIN
  <Initialization _u>
  <Name_P>.Count_IRI := <Name_P>.Count_IRI + 1;
  <Name_P>.For_<Ni> (<Name_P>.Count_IRI).
                  <Attr_From_X> := u.<Attr_From_X>;
                  .
                  .
                  .
END;
```

Fig. 23. A pattern of the second delete trigger

Parameter <*Name_P*> is replaced by <*Const_Name*>*_PCK*. Parameter <*Initialization _u*> is replaced by list of value assignment statements (one for each attribute from *X*), separated with the semicolons, of form:

u.<*Name_of_Attribute_from_X*> := OLD.<*Name_of_Attribute_from_X*>.

Bolded statements are repeating for each attribute from *X*.

```
CREATE OR REPLACE TRIGGER TRG_<Ni>_<Const_Name>_DEL3
AFTER DELETE ON <Ni>
DECLARE
  u <Ni>%ROWTYPE;
  I NUMBER;
BEGIN
  FOR j IN 1.. <Const_Name>_PCK.Count_IRI LOOP
      <Initialization_u>
      SELECT COUNT(*) INTO I FROM <Nj>
      WHERE (<Selection_Cond>);
      IF I <> 0 THEN
        <Execute_Activity>
      END IF;
  END LOOP;
END;
```

Fig. 24. A pattern of the third delete trigger

The third trigger (Fig. 24) is run on the statement level after the tuple deletion. It uses the auxiliary data structures generated by the second trigger.

The replacement of parameters, specific for Oracle patterns, during the trigger generation is done as follows:

- <*Initialization_u*> is replaced by the list of value assignment statements (one for each attribute from *X*), separated with the semicolons, of form:

  u.<*Name_of_Attribute_from_X*> :=

  <*Const_Name*>*_PCK.For_*<N_i> (*j*).<*Name_of_Attribute_from_X*>;

- <*Selection_Cond*> is replaced by the conjunction of relational expressions of the form:

  <*Name_of_Attribute_From_Y*> = *u*.<*Name_of_Attribute_From_X*>.

Depending on the selected activity, <*Execute_Activity*> is replaced with *Cascade_IRI_Del_*<N_i>(*u*) procedure call (Cascade delete), that belongs to the global package *Global_PCK*, or with SQL code for activity. SQL code raises the error:

RAISE_APPLICATION_ERROR (-20003,'Tuple deletion is forbidden <N_i>').

Parameterized pattern of the procedure for cascade deletion for Oracle Server is presented in Fig. 25.

```
PROCEDURE CascadeIRI_Del_<Ni> (u IN <Ni>%ROWTYPE)
IS
BEGIN
   DELETE FROM <Nj> v WHERE (<Selection_Cond>);
END;
```

Fig. 25. A parameterized pattern of the procedure for cascade deletion

Parameter <*Selection_Cond*> is replaced by the conjunction of relational expressions of the form:

v.<*Name_of_Attribute_From_Y*> = *u*.<*Name_of_Attribute_From_X*>.

5.2.3. Patterns for tuple modification in the presence of an IRIC

Basic differences between the patterns for tuple modification in the presence of an IRIC for MS SQL Server and Oracle, are the same as the differences discussed in Section 5.2.2. Therefore, the discussion is omitted here.

Same as for tuple deletion, for Oracle 10g three triggers are to be created for the implementation of tuple modification under the presence of IRICs. The first one is run at the statement level, before the tuple modification. It has an assignment to set the auxiliary data structures, used by two other triggers. The pattern for first trigger is shown in Fig. 26.

The second trigger (Fig. 27) is run just before tuple modification. It is aimed at putting the values of attributes from *X*, for the tuple that would be modified, into the previously declared auxiliary data structures.

```
CREATE OR REPLACE TRIGGER TRG_<Ni>_<Const_Name>_UPD1
BEFORE UPDATE ON <Ni>
BEGIN
   <Const_Name>_PCK.Count_IRI := 0;
   <Const_Name>_PCK.For_<Ni>.DELETE;
END;
```

Fig. 26. A pattern for the first modification trigger

```
CREATE OR REPLACE TRIGGER TRG_<Ni>_<Const_Name>_UPD2
BEFORE UPDATE ON <Ni>
FOR EACH ROW
WHEN (<Cond>)
DECLARE
  u <Ni>%ROWTYPE;
BEGIN
  <Initialization_u>
  <Name_P>.Count_IRI := <Name_P>.Count_IRI + 1;
  <Name_P>.For_<Ni> (<Name_P>.Count_IRI). <Attribute_From_X> :=
          u.< Attribute_From _X>;
  .
  .
  .
END;
```

Fig. 27. A pattern for the second modification trigger

```
CREATE OR REPLACE TRIGGER TRG_<N_i>_<Const_Name>_UPD3
AFTER UPDATE ON <N_i>
DECLARE
  u <N_i>%ROWTYPE;
  l NUMBER;
BEGIN
  FOR j IN 1..<Const_Name>_PCK.Count_IRI LOOP
    <Initialization_u>;
    SELECT COUNT(*) INTO l FROM <N_j> WHERE (<Selection_Cond>);
    IF l <> 0 THEN
      RAISE_APPLICATION_ERROR
          (-20002,'Tuple modification is forbidden <N_i>');
    END IF;
  END LOOP;
END;
```

Fig. 28. A pattern for the third modification trigger

Unlike the second deletion trigger, the second modification trigger has one more parameter. That is parameter <*Cond*>. During the trigger generation process it would be replaced by the disjunction of relational expressions (one for each attribute from *X*) of the form:

NEW.<*Name_of_Attribute_from_X*> <> OLD.<*Name_of_Attribute_from_X*>.

The third trigger (Fig. 28) is run on the statement level after the tuple modification. It uses the auxiliary data structures generated by the second trigger. The replacement of the parameters is analogous to the replacement of the corresponding parameters in the third deletion trigger (Fig. 24).

6. An Example of IRIC Specification and Implementation in IIS*Studio DE

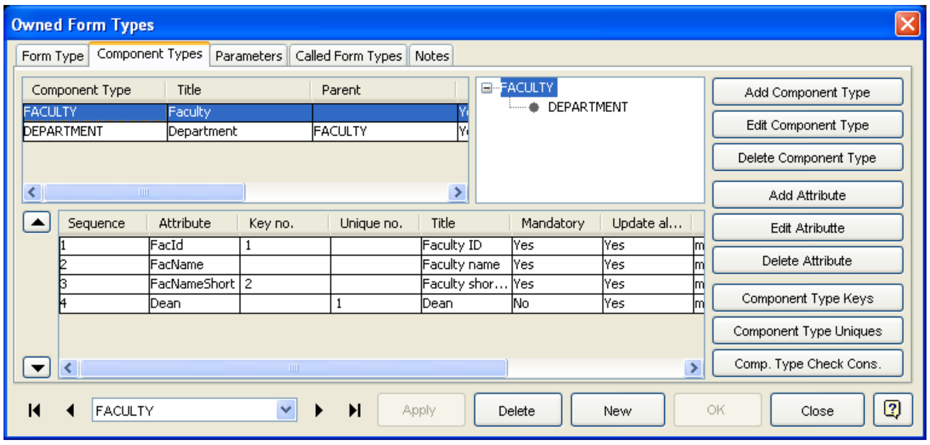
In this section, we present an example of an IRIC design specifications and transformation of design specifications into error free SQL specifications of relational DB schemas. We implement Examples 1 and 2 by means of IIS*Studio development environment.

In this section we present the processes of:

- A conceptual modelling of a DB schema;
- An automated design of relational DB schema in the 3rd normal form (3NF); and
- an automated generation of SQL/DDL code for chosen DBMSs.

A form type is the main modelling concept in IIS*Studio. Each form type is an abstraction of business documents, and therefore screens or report forms utilized by the end-users of the IS. IIS*Studio uses the set of form types to specify conceptual data model. From the set of form types, it generates the relational database schema ([19], [25]). In this way, by creating form types, a designer specifies: (i) a future database schema, (ii) functional properties of future transaction programs, (iii) and a look of the end-user interface, all at the same time. The detailed description of the structure and specification of a form type may be found in [19] and [25]. In Fig. 29 one of IIS*Studio forms for creating form type specifications is presented.

A form type is a hierarchical structure of form type components (Fig. 29). Each component type is identified by its name within the scope of a form type, and has non-empty sets of attributes and keys, a possibly empty set of unique constraints, and a specification of the check constraint. In Example 1 a faculty is composed of at least one department. Therefore, form type *Faculty* has at least one component type *Department*. This means that each *Faculty* instance is connected with at least one *Department* instance. In Fig. 30 the IIS*Studio form for specifying component type *Department* is given. The number of occurrences of component type *Department* on the form type *Faculty* may be specified. A designer has to choose between the options: 0-N and 1-N. If the option 0-N is chosen, the model describes a faculty organization that allows the existence of a faculty with no departments. The selection of the option 1-N, models a faculty organization that does not allow the existence of a faculty with no departments. Starting from the set of form types of an IS, IIS*Studio automatically generates a relational DB schema in 3NF with all relevant data constraints.



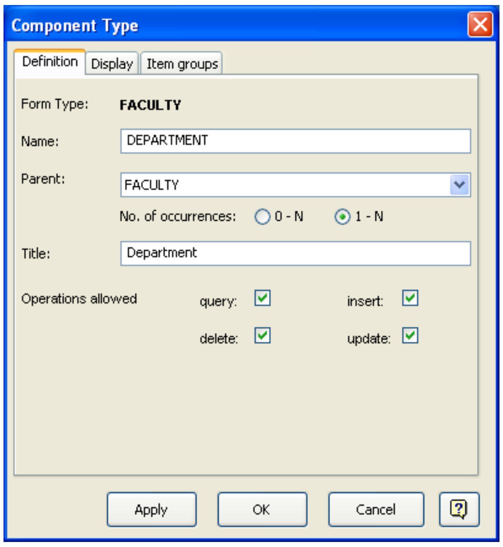
Component Type	Title	Parent	
FACULTY	Faculty		Y
DEPARTMENT	Department	FACULTY	Y

Sequence	Attribute	Key no.	Unique no.	Title	Mandatory	Update al...	
1	FacId	1		Faculty ID	Yes	Yes	m
2	FacName			Faculty name	Yes	Yes	m
3	FacNameShort	2		Faculty shor...	Yes	Yes	m
4	Dean		1	Dean	No	Yes	m

FACULTY

Fig. 29. The IIS*Studio form for specification of form type *Faculty*

Through the process of DB schema generation, the fact that component type *Department* is subordinated to the form type *Faculty* is recognized as the RIC *Department*[*FacId*] $\subseteq$ *Faculty*[*FacId*]. Furthermore, the selection of option 1-N for the number of occurrences of component type *Department* within the form type *Faculty* (Fig. 30) is recognized as the IRIC *Faculty*[*FacId*] $\subseteq$ *Department*[*FacId*] in the relational DB schema.



Form Type: FACULTY

Name: DEPARTMENT

Parent: FACULTY

No. of occurrences: 0 - N [✓ 1 - N]

Title: Department

Operations allowed query: ✓ insert: ✓ delete: ✓ update: ✓

Fig. 30. The form with a specification of the component type *Department*

IIS*Studio generates a directed graph called *Closure Graph*. A graph node represents a relation scheme and a graph directed edge (arc) between two relation schemes represents an IND between them. A closure graph diagram of University database schema (UDBS) is presented in Fig. 31. The relation schemes of UDBS are represented as rectangles and INDs between them as arrows. The arrow from the *Department* to the *Faculty* rectangle represents referential integrity constraint *IND1*, while the arrow from the *Faculty* to the *Department* rectangle represents inverse referential integrity constraint *IND2*.

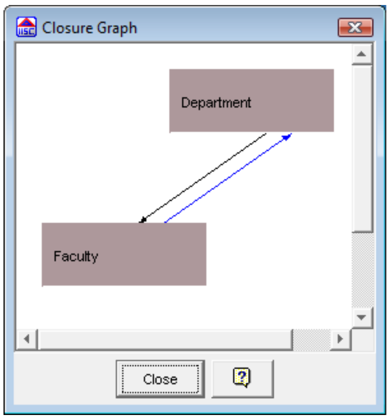
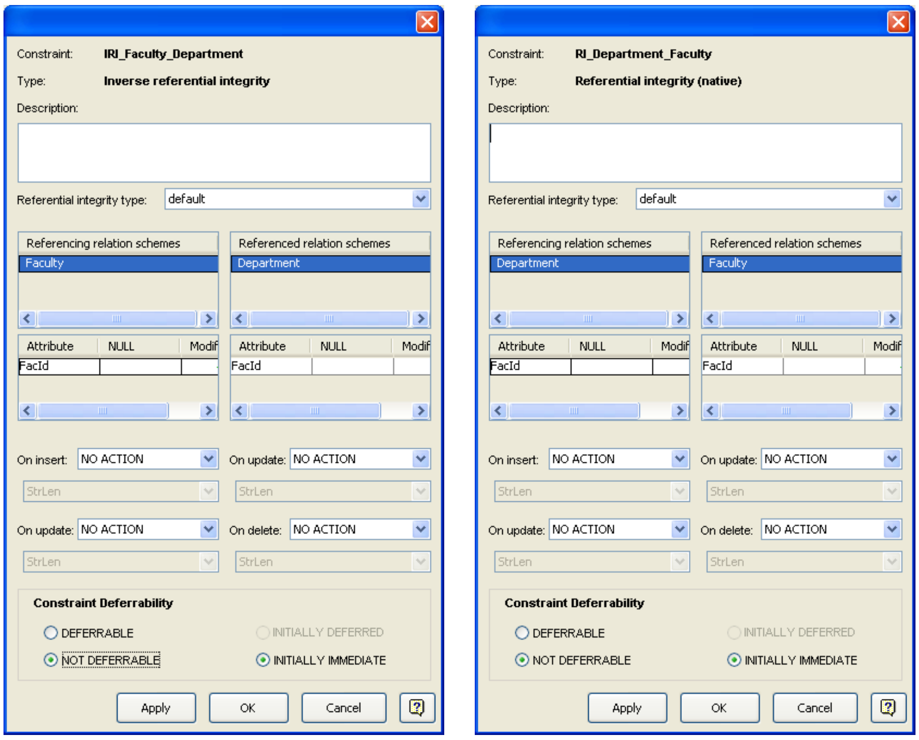


Fig. 31. A closure graph diagram of the University database schema

A designer may select an arrow representing an IND (RIC or IRIC) and invoke the appropriate form for further specifying of INDs (Fig. 32). Through this form possible actions for keeping the DB consistency on insert, update or delete operations are to be specified. A designer may select between No Action or Cascade actions in case of an IRIC specification. This selection will affect on the corresponding deletion or modification trigger, through the way of the replacement of <*Execute_Activity*> parameter.

An automated generation of SQL/DDL code for the chosen DBMS is the next step of DB generation by using IIS*Studio. An implementation (SQL) specification of relational DB schema is generated.


Constraint: **IRI_Faculty_Department**
Type: **Inverse referential integrity**
Description:
Referential integrity type: default

Referencing relation schemes	Referenced relation schemes
Faculty	Department

Attribute	NULL	Modif
FacId		

Attribute	NULL	Modif
FacId		

On insert: NO ACTION — On update: NO ACTION
On update: NO ACTION — On delete: NO ACTION

Constraint Deferrability
DEFERRABLE [✓ NOT DEFERRABLE]
INITIALLY DEFERRED [✓ INITIALLY IMMEDIATE]

Constraint: **RI_Department_Faculty**
Type: **Referential integrity (native)**
Description:
Referential integrity type: default

Referencing relation schemes	Referenced relation schemes
Department	Faculty

Attribute	NULL	Modif
FacId		

Attribute	NULL	Modif
FacId		

On insert: NO ACTION — On update: NO ACTION
On update: NO ACTION — On delete: NO ACTION

Constraint Deferrability
DEFERRABLE [✓ NOT DEFERRABLE]
INITIALLY DEFERRED [✓ INITIALLY IMMEDIATE]

Fig. 32. Forms for specifying IRIC and RIC, respectively

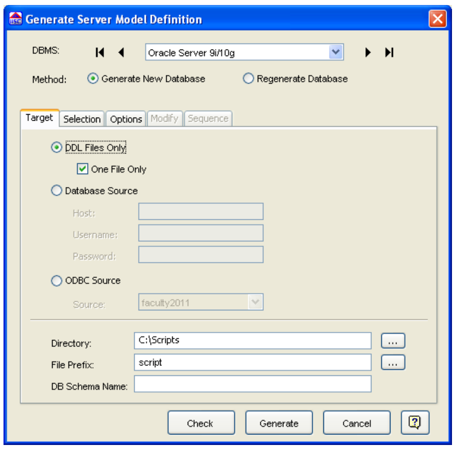
Two forms that are used to define values of SQL Generator input parameters are presented in Fig. 33. Firstly, we describe the form on the left-hand side of Fig. 33. The field *DBMS* enables the selection of the type and version of a target DB server. Oracle DBMS is chosen for the example. The radio button *DDL Files only* provides the creation of SQL scripts in files only. The radio button *Database Source* enables the selection of either Oracle or MS SQL DB server, establishing a connection, and immediate execution of generated SQL scripts. In this case, SQL Generator creates a script file, invokes the appropriate SQL tool, and passes necessary parameter values for the script execution. The radio button *ODBC Source* enables the creation and the immediate execution of SQL scripts in a selected ODBC data source. An appropriate ODBC driver for the target DB server must be installed and configured. SQL Generator supports the user authentication when it works via an established connection. The field *DB Schema Name* enables defining a DB name that is then included in an appropriate CREATE DATABASE command.

By means of *Selection* panel, a user picks relation schemes. SQL Generator will produce the appropriate SQL commands for the selected relation schemes only, and place them in script files.

By means of *Options* panel (right-hand side of Fig. 33) a user defines which types of DB objects are to be generated. By checking the appropriate check-box items, he or she may decide to generate: (i) indexes for primary, alternate and foreign keys, (ii) SQL CONSTRAINT clauses, (iii) triggers, and (iv) comments.

For inverse referential integrity constrains SQL Generator offers two ways of implementation: (i) by means of SQL views and the appropriate stored procedures, or (ii) by means of stored procedures only.

Not all possible combinations of the selected generator options are always valid. By pressing the *Check* button, a user initiates a check of the selected options. If some inconsistencies arise, a user gets the appropriate warnings. Pressing the button *Generate* initiates the generation of SQL scripts. Respecting the selected options, that can be seen on Fig. 33, the appropriate triggers are generated. The generated insertion trigger in the presence of IRIC *Faculty*[*FacId*] $\subseteq$ *Department*[*FacId*] for Oracle 10g is presented in Fig. 34. The examples of other generated procedures/triggers may be found in [1].



DBMS: Oracle Server 9i/10g

Method: [✓ Generate New Database] Regenerate Database

[✓ DDL Files Only]

✓ One File Only

Database Source

Host:

Username:

Password:

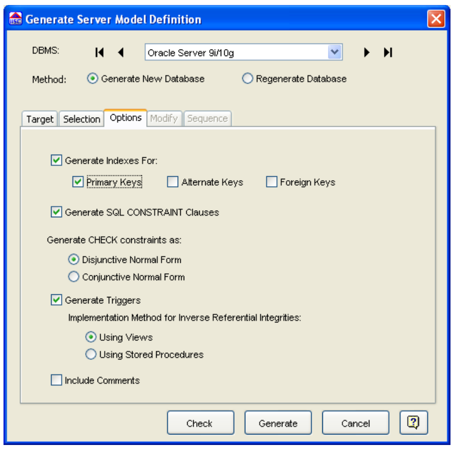
ODBC Source

Source: faculty2011

Directory: C:\Scripts

File Prefix: script

DB Schema Name:



DBMS: Oracle Server 9i/10g

Method: [✓ Generate New Database] Regenerate Database

✓ Generate Indexes For:

✓ Primary Keys ☐ Alternate Keys ☐ Foreign Keys

✓ Generate SQL CONSTRAINT Clauses

Generate CHECK constraints as:

[✓ Disjunctive Normal Form]

Conjunctive Normal Form

✓ Generate Triggers

Implementation Method for Inverse Referential Integrities:

[✓ Using Views]

Using Stored Procedures

☐ Include Comments

Fig. 33. A form for SQL Generator parameters specification

```
CREATE OR REPLACE TRIGGER TRG_IRI_Faculty_Department_View
INSTEAD OF INSERT ON View_Faculty_Department
FOR EACH ROW
DECLARE
   I NUMBER;
   Exc EXCEPTION;
   t Faculty%ROWTYPE;
BEGIN
   SELECT COUNT(*) INTO I
   FROM Faculty WHERE (FacId = :NEW.FacId);
   IF I <> 0 THEN
      INSERT INTO Department (FacId, DepId, DepName)
      VALUES (:NEW.FacId, :NEW.DepId, :NEW.DepName);
   ELSE
      IRI_Faculty_Department_PCK.Trigger_Ex := FALSE;
      INSERT INTO Faculty (FacId, FacName, Dean, FacShortName)
      VALUES (:NEW.FacId, :NEW.FacName, :NEW.Dean, :NEW.FacShortName);
      INSERT INTO Department (FacId, DepId, DepName)
      VALUES (:NEW.FacId, :NEW.DepId, :NEW.DepName);
      IRI_Faculty_Department_PCK.Trigger_Ex := TRUE;
      SELECT * INTO t
      FROM Faculty WHERE (FacId = :NEW.FacId);
      IF NOT Global_PCK.ContainmentIRI_Faculty(t) THEN
         RAISE Exc;
      END IF;
   END IF;
   EXCEPTION
   WHEN Exc THEN
      RAISE_APPLICATION_ERROR(-20005,'IRIC violation!');
END;
```

Fig. 34. The insertion trigger over the view for Oracle DBMS

7. Conclusion

In the paper we present an approach to the specification and implementation of IRICs. The algorithms that control the insertion, modification and deletion database operations under the presence of IRICs are shown. The patterns for triggers, as well as stored SQL functions and procedures, based on the aforementioned algorithms, are also presented. Proposed patterns provide generating SQL program code for DBMSs MS SQL Server 2008 and Oracle 10g. Our SQL Generator replaces the pattern parameters with real values obtained from a database specification stored in IIS*Case repository; then, it generates executable SQL scripts comprising necessary triggers, procedures and functions for a target DBMS platform.

While RICs are fully enforced by most current database systems, IRICs are completely disregarded by actual DBMSs, obliging users to manage them via procedural mechanisms (procedures and triggers). This implies an excessive effort to maintain integrity and develop applications. That is the reason why the IRICs are mostly implemented at the application logic (middle) layer instead at the DB server layer. Our approach enables the automated SQL scripts generation and moving of the IRICs validation at the

DB server. Thanks to that: (i) the costs of the application maintaining is cut; (ii) better performances due to the less traffic in the typical client-server architecture are provided; (iii) the same way of preventing the violation of a database consistency is enabled.

We propose two ways to insert mutually blocked tuples in one transaction: (i) a view created over the relations $r(N_i)$ and $r(N_j)$ may be used for the double insertion; or (ii) a custom DB procedure for double insertion may be developed. The first approach is more appropriate for causal end users that typically use high-level interactive query and data manipulation language. The second approach is aimed at embedding DML statements in general-purpose languages and making compiled transactions.

It is very hard to compare the features of MS T-SQL and Oracle PL/SQL in the course of implementation of IRICs. The evaluation of programmer's efforts during the program code preparing for SQL Generator strongly depends on programmer's previous knowledge, level of training and commitment to certain DBMS. We could say that according to our experience, Oracle enables easier parameters' transmission, partially due to the ROWTYPE data type, that does not exist in MS SQL Server. The ability of global variables declaration and grouping functions and procedures in packages enabled in Oracle is for sure advantage over MS SQL Server. We estimate that the existence of *Deleted* table in MS SQL Server facilitates the easier implementation of tuple deletion in the presence of IRIC, comparing with the Oracle.

Due to the fact that both Oracle and MS SQL Server, are widely used DBMSs, we decide to provide generating SQL program code for them in the first place.

Further development is directed towards extensions of SQL Generator's functionality to provide: (i) generating SQL scripts for a wider set of contemporary DBMSs and (ii) implementation of other, more complex constraints types, but often recognized in real database projects. One of typical examples is the extended referential integrity constraint (referential integrity constraint spanned over more then two relation schemes.

It is worth of emphasizing that IIS*Studio relies on the approach that conforms to the principles of model-driven (MD) approach. By means of IIS*Studio, a designer specifies only PIM models, because they are free of any implementation details. By the chain of consecutive transformations a set of different semi or fully platform specific models (PSMs) is generated. Consequently, a relational database schema that is generated by means of IIS*Studio is just one of the PSMs that we can get from PIM of the real system. Concerning the chain of model transformations, we recognize two main directions of our further research. The first one is to develop new transformations that will generate different PSMs like object-oriented model or XML model of database schemas. The second one is to develop reverse transformations from fully specific PSMs, through a series of semi PSMs towards a PIM model. One of them may be a transformation of legacy relational databases (fully PSMs) into logical database schemas expressed by the concepts of the relational data model (semi PSMs), or another, a transformation of relational database schemas into conceptual database schemas based on the form types (PIMs). Besides, we plan to investigate a possible usage of category theory in order to improve the performance of generated code [29].

Acknowledgements. Research presented in this paper was supported by Ministry of Science and Technological Development of Republic of Serbia, Grant III-44010, Title: *Intelligent Systems for Software Product Development and Business Support based on Models*.

References

1. Aleksić S.: An SQL Generator of Database Schema Implementation Specification in a CASE Toll IIS*Case. M. Eng. (Mr.) thesis, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia. (2006)
2. Aleksić S., Luković I., Mogin P., and Govedarica M.: A Generator of SQL Schema Specifications. Computer Science and Information Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, ISSN: 1820-0214, Vol. 4, No. 2, 77-96. (2007)
3. Aleksić S. and Luković I.: Generating SQL Specifications of a Database Schema for Different DBMSs. Info M - Journal of Information Technology and Multimedia Systems, Faculty of Organizational Sciences, Belgrade, Serbia, ISSN: 1451-4397, No. 23, 36-43. (2007)
4. Aleksić S., Ristić S., Luković I.: An Approach to Generating Server Implementation of the Inverse Referential Integrity Constraints. The 5th International Conference on Information Technologies ICIT 2011, May 11th – 13th, Amman, Jordan, Proceedings on CD. (2011)
5. Al-Jumaily, H.T., Cuadra, D., Martinez, P.: Plugging Active Mechanisms to Control Dynamic Aspects Derived from the Multiplicity Constraint in UML. In: The workshop of 7th International Conference on the Unified Modelling Language, Portugal. (2004)
6. Al-Jumaily H. T., Cuadra D., Martínez P.: OCL2Trigger: Deriving active mechanisms for relational databases using Model-Driven Architecture. Journal of Systems and Software, Vol. 81, No. 12, 2299-2314, ISSN 0164-1212, (2008)
7. ARTech. DeKlarit^{TM} (The Model-Driven Tool for Microsoft Visual Studio 2005), Chicago, U.S.A. [Online]. Available: http://www.deklarit.com/. (2007)
8. ANSI SQL:2003, American National Standards Institute, USA, ISO/IEC Std. 9075-{1, 2, 11}. (2003)
9. Ataullah A. A., Tompa F.: Business Policy Modelling and Enforcement in Databases. PVLDB Vol. 4, No. 11, 921-931. (2011)
10. Badaway, M., Richta, K.: Deriving triggers from UML/OCL specification, Information Systems Development: Advances in Methodologies, Components and Management, 305-315. (2003)
11. Berrabah D., Boufarès F.: Constraints Checking in UML Class Diagrams: SQL vs OCL. Database and Expert Systems Applications Lecture Notes in Computer Science, Vol. 4653/2007, 593-602, DOI: 10.1007/978-3-540-74469-6_58. (2007)
12. Bravo, L.: Handling Inconsistency in Databases and Data Integration Systems. Ph.D. Thesis, Carleton University. (2007)

13. Cabot J., Teniente E.: Constraint Support in MDA tools: A Survey. Proceedings of 2nd European Conference on Model Driven Architecture, LNCS, ECMDA-FA, 256-267. (2006)
14. CA ERwin Data Modeler r7.3. [Online]. Available: https://support.ca. com/irj/. (2008)
15. Ceri, S., Cochrane, R., and Widom, J.: Practical applications of triggers and constraints: success and lingering Issues. In VLDB 2000, 254-262, (2000)
16. Decker H., Martinenghi D.: Database Integrity Checking. In Database Technologies: Concepts, Methodologies, Tools, and Applications, ed. John Erickson, 212-220, doi:10.4018/978-1-60566-058-5.ch016. (2009)
17. Elmasri R., Navathe B. S.: Database Systems: Models, Languages, Design and Application Programming, Sixth Edition, Pearson Global Edition, ISBN 978-0-13-214498-8. (2011)
18. Govedarica M.: Design the Set of Implementation Database Schema Constraints. M. Eng. (Mr.) thesis, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia. (1998)
19. Luković I., Mogin P., Pavićević J., and Ristić S.: An Approach to Developing Complex Database Schemas Using Form Types. Software: Practice and Experience, John Wiley & Sons Inc, Hoboken, USA, ISSN: 0038-0644, DOI: 10.1002/spe.820 Vol. 37, No. 15, 1621-1656. (2007)
20. Luković I., Ristić S., Mogin P., and Pavicević J.: Database Schema Integration Process – A Methodology and Aspects of Its Applying, Novi Sad Journal of Mathematics, Faculty of Science, Novi Sad, Serbia, ISSN: 1450-5444, Vol. 36, No. 1, 115-140. (2006)
21. Microsoft SQL Server 2008. (2008)
22. Mogin P., Luković I., and Govedarica M.: Database Design Principles, 2nd Edition, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia, ISBN: 86-80249-81-5. (2004)
23. Mogin P., Luković I., and Govedarica M.: Extended Referential Integrity, Novi Sad Journal of Mathematics, Novi Sad, Serbia, ISSN: 1450-5444, Vol. 30, No. 3, 111-122. (2000)
24. Oracle DBMS 10g. (2004)
25. Pavićević J., Luković I., Mogin P., and Govedarica M.: Information System Design and Prototyping Using Form Types. International Conference on Software and Data Technologies, Setubal, Portugal, September 11-14, Vol. 2, 157-160. (2006)
26. Ristic S., Aleksic S., Lukovic I., Banovic J.: Form-Driven Application Generation: A Case Study, In Proceedings of the XI International Conference on Informatics, Roznava, Slovakia, 115 – 120. (2011)
27. Rybola Y., Richta K.: Transformation of binary relationship with particular multiplicity. In DATESO 2011, Vol. 11, 25–38. Czech Republic: Department of Computer Science, FEECS VSB – Technical University of Ostrava. [Online]. Available: http://www.informatik.uni-trier.de/ ˜ley/db/conf/dateso/dateso 2011.html. (2011)
28. Sybase PowerDesigner 15. (2009)
29. Slodičák V.: Some useful structures for categorical approach for program behaviour. Journal of Information and Organizational Sciences, Vol. 35, No. 1, 99-109, (2011)
30. Türker, C., Gertz, M.: Semantic Integrity Support in SQL-99 and Commercial (Object) Relational Database Management Systems. UC Davis Computer Science Technical Report CSE-2000-11, University of California. (2000)

31. Zimbrão, G., Miranda, R., de Souza, J., Estolano, M.H, Neto, F. P.: Enforcement of business rules in relational databases using constraints. In Proceedings of XVIII Simposio Brasileiro de Bancos de Dados/SBBD, 129-141, UFAM. (2003)

Slavica Aleksić received her M.Sc. degree from the Faculty of Technical Sciences at University of Novi Sad. She completed her Mr (2 year) degree at the University of Novi Sad, Faculty of Technical Sciences. Currently, she works as a teaching assistant at the Faculty of Technical Sciences, at University of Novi Sad, where she assists in teaching several Computer Science and Informatics courses. Her research interests are related to Database Systems, Theory of Data Models, System Design, Logical and Physical Database Design, Development and Usage of MDSE / CASE tools in Software Engineering and System Design, Reengineering of Information Systems and Model Transformations in MDA.

Sonja Ristić works as an associate professor at the University of Novi Sad, Faculty of Technical Sciences, Serbia. She received two bachelor degrees with honors from University of Novi Sad, one in Mathematics, Faculty of Science in 1983, and the other in Economics from Faculty of Economics, in 1989. She received her Mr (2 year) and Ph.D. degrees in Informatics, both from Faculty of Economics, University of Novi Sad, in 1994 and 2003. From 1984 till 1990 she worked with the Novi Sad Cable Company NOVKABEL– Factory of Electronic Computers. From 1990 till 2006 she was with High School of Business Studies -Novi Sad, and since 2006 she has been with the Faculty of Technical Sciences at University of Novi Sad. Her research interests are related to Database Systems and Software Engineering.

Ivan Luković received his M.Sc. degree in Informatics from the Faculty of Military and Technical Sciences in Zagreb in 1990. He completed his Mr (2 year) degree at the University of Belgrade, Faculty of Electrical Engineering in 1993, and his Ph.D. at the University of Novi Sad, Faculty of Technical Sciences in 1996. Currently, he works as a Full Professor at the Faculty of Technical Sciences at the University of Novi Sad, where he lectures in several Computer Science and Informatics courses. His research interests are related to Database Systems and Software Engineering. He is the author or co-author of over 90 papers, 4 books, and 30 industry projects and software solutions in the area.

Milan Čeliković graduated in 2009 at the Faculty of Technical Sciences, Novi Sad, at the Department of Computing and Control. Since 2009 he has worked as a teaching assistant at the Faculty of Technical Sciences, Novi Sad, at the Chair for Applied Computer Science. In 2010, he started his Ph.D. studies at the Faculty of Technical Sciences, Novi Sad. His main research interests are focused on: Domain specific modeling, Domain specific languages, Databases and Database management systems. At the moment, he is involved in the projects concerning application of DSLs in the field of software engineering.

Received: November 02, 2011; Accepted: December 08, 2012.