An Approach to Generating Server Implementation of the Inverse Referential Integrity Constraints

Slavica Aleksić [#1], Sonja Ristić [*2], Ivan Luković [#3]

[#] *University of Novi Sad, Faculty of Technical Sciences,*
Department of Computing and Control
Trg Dositeja Obradovića 6
21000 Novi Sad, Serbia
[1] slavica@uns.ac.rs
[3] ivan@uns.ac.rs

[*] *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 inclusion dependencies (INDs) convey much information on the data structure and data semantics. There are two basic kinds of INDs: key-based INDs and non-key-based INDs. The inverse referential integrity constraints (IRICs) are special case of non-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 (as well as IRICs as their special case) 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 the native IRICs and IRICs inferred from nontrivial inclusion dependencies integrated in the SQL Generator tool that we developed as integral part of the IIS*Case development environment.**

***Keywords*— Inclusion Dependencies, Non-key-based IND, Key-based IND, Inverse Referential Integrity Constraint, Declarative Constraint Specification.**

I. Introduction

A common approach to database design is to describe the structure and constraints of the Universe of Discourse in a semantically rich conceptual data model. The obtained conceptual database schema is subsequently translated into a logical, relational database schema, representing a design specification of the future database. The most fundamental integrity constraints that arise in practice in relational databases are functional dependencies (FDs) and inclusion dependencies (INDs). Both are fundamental to the conceptual and logical database design and are supported by the SQL standard. The inclusion dependencies convey much information on the data structure and data semantics. Let $N_i(R_i, C_i)$ and $N_j(R_j, C_j)$ be two relation schemes, where N_i and N_j are theirs names, R_i and R_j, corresponding sets of attributes, and C_i and C_j corresponding sets of relation schemes' constraints. An inclusion dependency is a statement of the form $N_i[X] \subseteq N_j[Y]$, where X and Y are non-empty sets of attributes from R_i and R_j respectively. Having the inclusion operator orientated from left to right ($\subseteq$) we say that relation scheme N_i is on the left-hand side of the IND, while the relation schema N_j is on the right-hand side of the IND. In order to define the satisfaction of the IND we use the following notation: the relation $r(N_i)$ is the set of tuples $u(R_i)$ (or just u) satisfying all constraints from the constraint set C_i, X-value is the projection of a tuple u on the set of attributes X and, according to the aforementioned orientation of the inclusion operator, $r(N_i)$ is called referencing relation, while $r(N_j)$ is called referenced relation. Informally, a database satisfies the inclusion dependency if the set of X-values in the referencing relation $r(N_i)$ is a subset of the set of Y-values in the referenced relation $r(N_j)$. There are two basic kinds of INDs: key-based INDs and non-key-based INDs. The IND is said to be key-based if the set of attributes Y is key of the relation scheme N_j, and non-key-based otherwise. More often key-based INDs are called referential integrity constraints (RICs). Non-key-based INDs with X that is a key of the relation scheme N_i, where RIC $N_j[Y] \subseteq N_i[X]$ is specified as well, are called inverse referential integrity constraints (IRICs). Referential integrity constraints may be fully enforced by most current relational database management systems (RDBMSs). On the contrary, non-key-based INDs (as well as IRICs as their special case) 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.

In order to provide an efficient transformation of design specifications into error free SQL specifications of relational database (db) schemas we developed the SQL Generator [2]. 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. SQL Generator is integrated in Integrated Information Systems*Case (IIS*Case), a software tool aimed to provide the information system (IS) design and generating executable application prototypes. It is an integral part of 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. A case study illustrating main features of IIS*Case is given in [8], the methodological aspects of its usage may be found in [9] and the description of information system design and prototyping using form types is given in [16]. IIS*Case generates 3NF relational db schemas with all the relation scheme keys, null value constrains, unique constrains, referential and inverse referential integrity constraints. These schemas are stored in the IIS*Case repository. The specification of the IIS*Case repository is given in [16]. The input into SQL Generator is a database schema stored in the repository.

Using SQL Generator, a user may produce SQL scripts for the creation of tables, views, indexes, sequences, procedures, functions and triggers, even without knowing SQL syntax and mechanisms for the implementation of constrains of a selected DBMS. SQL Generator may produce scripts for implementing a new db schema, or modify an already existing one in the following three ways: (i) by creating SQL scripts in files only for a later execution, (ii) by creating and immediately executing SQL scripts under a selected db server with an established connection, and (iii) by creating and immediately executing SQL scripts on a selected data source with an established connection via an ODBC driver. In all three cases, generated SQL scripts are stored in one or more files.

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, native inverse referential integrity constraints, and inverse referential integrity constraints inferred from nontrivial inclusion dependencies ([6], [13]). Constraints are implemented by the declarative DBMS mechanisms, whenever it is possible. However, the expressiveness of declarative mechanisms of commercial DBMSs may be limited and therefore, SQL Generator implements a number of constraints through the procedural mechanisms [3].

In this paper we present the SQL Generator's feature of an automated implementation of the native IRICs and IRICs inferred from nontrivial inclusion dependencies. Systems adhering to the SQL standard allow specifying of RICs using the FOREIGN KEY clause, but the IRICs are disregarded by actual RDBMSs.

There are numerous contemporary software tools aimed at an automated conceptual database schema design and its implementation under different database management systems, such as: DeKlarit, ERwin Data Modeler, Oracle Designer, Power Designer etc. Some of them are described in [4], [5], [15], [17]. All of them enable setting the relationship minimal multiplicity (cardinality) to one. Therefore, they support the specification of the existential dependency between two entity types. However, all of them ignore this specification when generate the SQL code to implement a 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.

II. Inverse Referential Integrity Constraint

The business rules that would be modeled with the inverse referential integrity constraints are not rare in the real world. They are the consequence of the mutual existential dependency of the entities of two entity classes in the 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 must have at least one department. The relational database schema of a very simplified and hypothetical university information system, beyond the others, has two relation schemes (RS) *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 the relation scheme *Faculty*, *IND1* is the key-based inclusion dependency, i.e. the referential integrity constraint. It is modeling 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. The *FacId* is the key of the relation scheme *Faculty*, which is on the left 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 modeling the business rule that faculty must have at least one department. Fig. 1 represents the University database schema using the IIS*Case closure graph. The arrow from the *Department* to the *Faculty* rectangle represents referential integrity constraint, while the arrow from the *Faculty* to the *Department* rectangle represents inverse referential integrity constraint. □

Database systems adhering to the SQL standard allow specifying of RICs using the FOREIGN KEY clause, but the IRICs are disregarded by actual RDBMSs. Programmers are obliged to manage them via procedural mechanisms (procedures and triggers). That is the reason why the IRICs are mostly implemented on the middle layer instead on the db server. Still, the validation of the IRICs on 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.

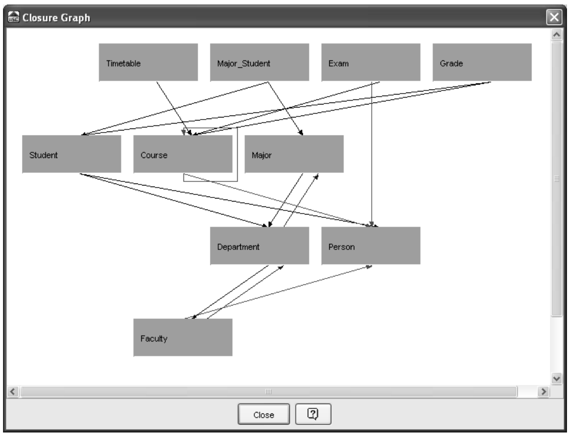

Fig. 1 The IIS*Case closure graph diagram of a University db schema

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 [7], (ii) DBMS Microsoft (MS) SQL Server 2000/2008 with MS T-SQL [10], [11], and (iii) DBMS Oracle 9i/10g with Oracle PL/SQL [14].

III. Algorithms for IRIC Validation

By specifying of the IRICs $N_j[Y] \subseteq N_i[X]$ it comes towards the bogus mutual „locking" of the instances of the relation schemas N_i and N_j. The notion „locking" is used to illustrate the following situation: (i) it is not possible to insert new tuple into relation $r(N_i)$ with not null values for all attributes $A \in X$, unless there is the tuple in the relation $r(N_j)$ with the Y value same as the X value of the inserted tuple; and, as well, (ii) it is not possible to insert new tuple into relation $r(N_j)$ with a certain Y value, unless there is the tuple in the relation $r(N_i)$ with the X value same as the aforementioned Y value [12].

Example 2. Fig. 2 shows a database instance of the database schema from Example 1. Due to the specified referential integrity *IND1* it is not possible to insert the tuple (2, D2, 'Dentistry') into the relation *Department*. But, due to the specified inverse referential integrity *IND2* it is not possible to insert the tuple (2, 'FOM', 'Faculty of Medicine', 'Simpson') into the relation *Faculty*. These tuples are said to be mutually locked. □

Faculty

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

Department

FacId	*DeptId*	*DeptName*
1	D1	Geometry

Fig. 2 A University database instance

Because of that mechanisms for IRIC's validation require deferred trigger consideration during the transaction. Albeit SQL standards allow deferred check constraint, most of the contemporary DBMSs do not support it.

In this Section the common algorithms for controlling the IRIC validation during the insert, update and delete operations are given. The algorithms for insertion, deletion and modification control in the presence of inverse referential integrity constraints are presented in Fig 3, Fig 4 and Fig 5, respectively. In the following text these algorithms will be described in more details.

An IRIC can be violated in three cases: when tuple is inserted into the referencing relation, when tuple is deleted from the referenced relation or when tuple's X-value is modified in the referenced relation.

An algorithm for the control of insertions (Fig. 3) 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:	INSERTION CONTROL IN THE PRESENCE OF IRICs
Definition area: **Relation schemes**: N_i, N_j **Attributes:** $X = (A_1, \dots, A_{\lvert X\rvert})$ $X \in R_i$, $Y = (B_1, \dots, B_{\lvert Y\rvert})$ $Y \in R_j$ $\lvert X\rvert = \lvert Y\rvert \wedge (\forall l \in \{1,\dots,\lvert X\rvert\})(dom(A_l) \subseteq dom(B_l))$	
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	tuple 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←0
  DO Search_in ∀ u∈r(Ni) WHILE ind = 0
    IF v[Y] = u[X] THEN
      SET ind←1
    ENDIF
  ENDDO Search_in
  IF ind = 0 THEN
    CANCEL_OPERATION('Error description')
  ENDIF
ENDPROCESS Insert_inv_ref_int
```

Fig. 3 An algorithm for insertion control

An algorithm for the control of deletions (Fig. 4) 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.

Trigger:	DELETION CONTROL IN THE PRESENCE OF IRICs
Definition area: **Relation schemes**: N_i, N_j **Attributes:** $X = (A_1, \ldots, A_{\|X\|})\ X \in R_i$, $Y = (B_1, \ldots, B_{\|Y\|})\ Y \in R_j$ $\|X\| = \|Y\| \wedge (\forall\, l \in \{1,\ldots,\|X\|\})(dom(A_l) \subseteq dom(B_l))$	
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	tuple *u* - tuple that would be deleted into $r(N_i)$
Local declarations:*ind* (*ind* = 1 – constraint is satisfied, *ind* = 0 – constraint is violated)	

```
Pseudo code:
BEGIN PROCESS  Delete_inv_ref_int
  SET ind ← 0
  DO Search_Null_value ∀ A∈X  WHILE  ind = 0
    IF u[A] = ω THEN
      SET ind ← 1
    ENDIF
  ENDDO Search_Null_value
  IF ind = 0 THEN
    DO Search_t ∀ t∈r(Ni) WHILE ind =0
      IF t[Kp(Ri)] ≠ u[Kp(Ri)] ∧ u[X] = t[X]  THEN
        SET ind ← 1
      ENDIF
    ENDDO  Search_t
  ENDIF
  IF ind = 0 THEN
    EXECUTE ACTIVITY
  ENDIF
ENDPROCESS  PROCESS  Delete_inv_ref_int
```

Fig. 4 An algorithm for deletion control

An algorithm for the control of modifications (Fig. 5) 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; (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 tuple *u*) with *X*-value matching the original *X*-value. The explanation for the second condition is analog to the explanation for the first condition in the previous paragraph.

Trigger:	MODIFICATION CONTROL IN THE PRESENCE OF IRICs
Definition area: **Relation schemes**: N_i, N_j **Attributes:** $X = (A_1, \ldots, A_{\|X\|})\ X \in R_i$, $Y = (B_1, \ldots, B_{\|Y\|})\ Y \in R_j$ $\|X\| = \|Y\| \wedge (\forall\, l \in \{1,\ldots,\|X\|\})(dom(A_l) \subseteq dom(B_l))$	
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	tuple *u* - tuple that would be modified $r(N_i)$
Local declarations:*ind* (*ind* = 1 – constraint is satisfied, *ind* = 0 – constraint is violated)	

```
Pseudo code:
BEGIN PROCESS  Update_inv_ref_int
  IF  u'[X] ≠ u[X]  THEN
    SET  ind←0
    DO Search_Null_value ∀ A∈X  WHILE  ind = 0
      IF u[A] = ω THEN
        SET ind ← 1
      ENDIF
    ENDDO  Search_Null_value
    IF ind = 0 THEN
      DO  Search_t ∀ t∈r(Ni) WHILE  ind =0
        IF t[Kp(Ri)] ≠ u[Kp(Ri)] ∧ u[X] = t[X]  THEN
          SET ind ← 1
        ENDIF
      ENDDO  Search_t
    ENDIF
    IF ind = 0 THEN
      CANCEL_OPERATION('Error description')
    ENDIF
  ENDIF
ENDPROCESS  PROCESS  Update_inv_ref_int
```

Fig. 5 An algorithm for modification control

IV. Implementation of IRICs by Procedural Mechanisms

The process of 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].

In this Section, we present the parameterized patterns of the algorithms from Section 3 for DBMSs MS SQL Server 2008 [11] and Oracle 10g [14]. Since the parameterized patterns for implementation of modification and deletion are similar, only the patterns for insertion and deletion will be presented.

In order to keep the db consistency checking under the database management system, in the presence of the IRICs a special mechanism has to be developed. Namely, mutually locked 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) i r(N_j) may be used for the double insertion; and (ii) a custom db procedure for double insertion may be developed. In the following subsections the first way will be shown. The patterns for the custom procedures, both for the MS SQL Server and Oracle may be found in [1].

A. IRIC Implementation for MS SQL Server 2008

The pattern of the trigger using views for tuple insertion is presented in Fig. 6. Procedure *Trigger_Ex* in Fig 7 is aimed at the trigger's execution control. In the suggested solution an auxiliary db relation *Trigger_Stat* is used. This relation contains the information would the observed trigger be executed or not in previously specified transaction. If the relation contains the tuple with given trigger name and transaction ID, trigger procedure will not be executed. The pattern of the db function *ContainmentIRI_*<N_j>, called in this trigger is shown in Fig. 9.

```
CREATE TRIGGER TRG_<Const_Name>_INV_View
ON View_<Nj>_<Ni> INSTEAD OF INSERT
AS
  DECLARE
    @Idt int, @Count int, <Decl_Var_For_Ni_i_Nj>
    SELECT <Var_array_For_Ni_i_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. 6 A pattern of the trigger over view

```
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. 7 A SQL procedure for trigger execution control

```
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. 8 A tuple insertion control pattern

```
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

```
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. 10 A SQL function for trigger execution control

```
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. 11 A pattern of the delete trigger

SQL code for view creation is trivial, and therefore it is omitted here. We only emphasize that it should contain all attributes from both relation schemes: N_i and N_j.

In order to prevent the IRIC violation due to the separate insertion of mutually locked tuples a trigger adhering the pattern in Fig. 8 should be created.

Finally, the pattern for SQL function for trigger execution is presented in Fig. 10.

The pattern of the trigger for tuple deletion is presented in Fig. 11. Depending on the selected activity, *<Execute_Activity>* is replaced with *CascadeIRI_Del_<N$_i$>* procedure call (Cascade delete) or with SQL code for activity restriction. Aforementioned code could be found in [1].

B. IRIC Implementation for Oracle 10g

SQL syntax for different DBMSs is not the same. Therefore, we present the parameterized patterns for triggers and procedures implementing algorithms from Section 3, for Oracle db Server. The pattern of the trigger using views for tuple insertion is presented in Fig. 12. The pattern of the db function *ContainmentIRI_<N$_j$>*, called in this trigger is shown in Fig. 14.

```
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
  <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. 12 A pattern of the trigger over view

In Oracle Server *Trigger_Ex* is a global variable defined in special 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. 13.

```
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. 13 A pattern of IRIC's package

In order to prevent the IRIC violation due to the separate insertion of mutually locked tuples a trigger adhering the pattern in Fig. 15 should be created.

For Oracle 10g three triggers should 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 first trigger is shown in Fig. 16.

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

Fig. 14 A pattern of the ContainmentIRI_<N$_j$> function

```
CREATE OR REPLACE TRIGGER
TRG_<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. 15 A tuple insertion control pattern

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

Fig. 16 A pattern of the first delete trigger

```
CREATE OR REPLACE TRIGGER
                              TRG_<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. 17 A pattern of the second delete trigger

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. 17.

The third trigger (Fig. 18) is run on the statement level after the tuple deletion. It uses the auxiliary data set by the second trigger.

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

Fig. 18 A pattern of the third delete trigger

Depending on the selected activity, *<Execute_Activity>* is replaced with *CascadeIRI_Del_*<N_i> procedure call (Cascade delete) or with SQL code for activity restriction. Aforementioned code could be found in [1].

V. Conclusions

In order to provide an efficient transformation of design specifications into error free SQL specifications of relational db schema we developed the SQL Generator, as an integral part of the development environment IIS*Studio. IIS*Studio generates 3NF relational db schema with all the relation scheme keys, null value constrains, unique constrains, referential and inverse referential integrity constraints. These schemas are stored in the IIS*Studio repository. The input into SQL Generator is a database schema specification stored in the repository. 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, native inverse referential integrity constraints, and inverse referential integrity constraints inferred from nontrivial inclusion dependencies.

In the paper we deal with the inverse referential integrity constraints. We presented the algorithms that control the insertion, modification and deletion database operations under the presence of IRICs. 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.

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, as it is illustrated in [8].

Acknowledgment

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] S. Aleksić, “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, Nov. 2006.

[2] S. Aleksić, I. Luković, P. Mogin, and M. Govedarica, “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, pp. 77-96, 2007.

[3] S. Aleksić and I. Luković, "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, pp. 36-43, 2007.

[4] (2007) ARTech. *DeKlarit*™ (*The Model-Driven Tool for Microsoft Visual Studio 2005*), Chicago, U.S.A. [Online]. Available: http://www.deklarit.com/

[5] (2008) *CA ERwin Data Modeler r7.3*, [Online]. Available: https://support.ca. com/irj/

[6] M. Govedarica, “Design the Set of Implementation Database Schema Constraints,” M. Eng. (Mr.) thesis, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia, 1998.

[7] *ANSI SQL:2003*, American National Standards Institute, USA, ISO/IEC Std. 9075-{1, 2, 11}, 2003.

[8] I. Luković, P. Mogin, J. Pavićević, and S. Ristić, “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, pp. 1621-1656, 2007.

[9] I. Luković, S. Ristić, P. Mogin, and J. Pavicević, “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, pp. 115-140, 2006.

[10] *Microsoft SQL Server 2000*, 2000.

[11] *Microsoft SQL Server 2008*, 2008.

[12] P. Mogin, I. Luković, and M. Govedarica, *Database Design Principles*, 2nd Edition, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia, ISBN: 86-80249-81-5, 2004.

[13] P. Mogin, I. Luković, and M. Govedarica, “Extended Referential Integrity”, *Novi Sad Journal of Mathematics*, Novi Sad, Serbia, ISSN: 1450-5444, Vol. 30, No. 3, pp. 111-122, 2000.

[14] *Oracle DBMS 10g*, 2004.

[15] *Oracle Designer 9i*, 2000.

[16] J. Pavićević, I. Luković, P. Mogin, and M. Govedarica, “Information System Design and Prototyping Using Form Types,” *INSTICC I International Conference on Software and Data Technologies*, Setubal, Portugal, September 11-14, Proceedings, Vol. 2, pp. 157-160, 2006.

[17] *Sybase PowerDesigner 15*, 2009.