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

Slavica Aleksić¹, Sonja Ristić², Ivan Luković³

¹ University of Novi Sad, Faculty of Technical Sciences, Department of Computing and Control
Trg Dositeja Obradovića 6
21000 Novi Sad, Serbia
¹slavica@uns.ac.rs
³ivan@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
²srnistic@uns.ac.rs

ICIT 2011 The 5th International Conference on Information Technology

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 scheme \( N_j \) 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, \( r(N_j) \) is called referencing relation, while \( r(N_i) \) 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 \( N_j \), where \( R_C N_j[Y] \subseteq N_i[X] \) 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 constrains of the following types: domain constrains, key constrains, unique constrains, tuple constrains, native and extended referential integrity constrains, referential integrity constrains inferred from nontrivial inclusion dependencies, native inverse referential integrity constrains, and inverse referential integrity constrains inferred from nontrivial inclusion dependencies ([6], [13]). Constrains 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 constrains 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:

\[
\text{Faculty}(\{\text{FacId}, \text{FacShortName}, \text{facName}, \text{Dean}\}, \\
\quad \{\text{DepId}\}), \\
\text{Department}(\{\text{FacId}, \text{DepId}, \text{Name}\}, \\
\quad \{\text{FacId+DepId}\}), \\
\text{IND1: Department} \subseteq \text{Faculty}[\text{FacId}], \\
\text{IND2: Faculty}[\text{FacId}] \subseteq \text{Department}[\text{FacId}].
\]

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.

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 \( N_1 \subseteq N_2 \) it comes towards the bogus mutual „locking“ of the instances of the relation schemas \( N_1 \) and \( N_2 \). The notion „locking“ is used to illustrate the following situation: (i) it is not possible to insert new tuple into relation \( r(N_1) \) with not null values for all attributes \( A \in X \), unless there is the tuple in the relation \( r(N_2) \) 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 \( r(N_1) \) with a certain \( Y \) value, unless there is the tuple in the relation \( r(N_2) \) 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 \( IND_1 \) it is not possible to insert the tuple (2, D2, 'Dentistry') into the relation Department. But, due to the specified inverse referential integrity \( IND_2 \) 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.

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 \( v \).

![Fig. 1 The IIS*Case closure graph diagram of a University db schema](image1)

![Fig. 2 A University database instance](image2)

![Fig. 3 An algorithm for insertion control](image3)

![Fig. 4 An algorithm for deletion control](image4)

![Fig. 5 An algorithm for update control](image5)
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 \( r(N) \), 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 pseudoinstruction `EXECUTE ACTIVITY` will be replaced with an appropriate program code for the selected action.

### Trigger: DELETION CONTROL IN THE PRESENCE OF IRICs

<table>
<thead>
<tr>
<th>Definition area:</th>
<th>Relation schemes: ( N, N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes: ( X = (A_1, \ldots, A_m) X \in R, Y = (B_1, \ldots, B_n) Y \in R )</td>
<td></td>
</tr>
<tr>
<td>( X = { Y } \land (\forall , l \in {1, \ldots,</td>
<td>X</td>
</tr>
<tr>
<td>Specification of the constraint: ( \forall , N \in</td>
<td>X</td>
</tr>
<tr>
<td>Specification of the operation: Time: AFTER OPERATION Operation: DELETE</td>
<td></td>
</tr>
<tr>
<td>From DB</td>
<td>( r(N), r(N) )</td>
</tr>
<tr>
<td>Input</td>
<td>tuple ( u ) - tuple that would be modified ( r(N) )</td>
</tr>
<tr>
<td>Local declarations:</td>
<td>( ind )</td>
</tr>
<tr>
<td>( (ind = 1 ) – constraint is satisfied, ( ind = 0 ) – constraint is violated)</td>
<td></td>
</tr>
</tbody>
</table>

### Pseudo code:

#### BEGIN PROCESS

Delete_inv_ref_int

\[ \begin{align*} & \text{SET} \ ind \leftarrow 0 \\
& \text{DO} \ Search\_Null\_value \ \forall \ A \in X \text{ WHILE } ind = 0 \\
& \quad \text{IF} \ u[A] = \emptyset \text{ THEN} \\
& \quad \quad \text{SET} \ ind \leftarrow 1 \\
& \quad \text{ENDIF} \\
& \text{ENDDO} \ Search\_Null\_value \\
& \text{IF} \ ind = 0 \ \text{THEN} \\
& \quad \text{DO} \ Search_t \ \forall \ t \in r(N) \ \text{WHILE} \ ind = 0 \\
& \quad \quad \text{IF} \ \{K_j(R_j) \} \cup \{K_j(R_j) \} \cup \{X\} = \{X\} \text{ THEN} \\
& \quad \quad \quad \text{SET} \ ind \leftarrow 1 \\
& \quad \quad \text{ENDIF} \\
& \quad \text{ENDDO} \ Search_t \\
& \text{ENDIF} \\
& \text{IF} \ ind = 0 \ \text{THEN} \\
& \quad \text{EXECUTE ACTIVITY} \\
\end{align*} \]

#### ENDPROCESS PROCESS Delete_inv_ref_int

![Fig. 4 An algorithm for deletion control](image)

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**

<table>
<thead>
<tr>
<th>Definition area:</th>
<th>Relation schemes: ( N, N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes: ( X = (A_1, \ldots, A_m) X \in R, Y = (B_1, \ldots, B_n) Y \in R )</td>
<td></td>
</tr>
<tr>
<td>( X = { Y } \land (\forall , l \in {1, \ldots,</td>
<td>X</td>
</tr>
<tr>
<td>Specification of the constraint: ( \forall , N \in</td>
<td>X</td>
</tr>
<tr>
<td>Specification of the operation: Time: AFTER OPERATION Operation: UPDATE</td>
<td></td>
</tr>
<tr>
<td>Data Inputs</td>
<td>From DB</td>
</tr>
<tr>
<td>Input</td>
<td>tuple ( u ) - tuple that would be modified ( r(N) )</td>
</tr>
<tr>
<td>Local declarations:</td>
<td>( ind )</td>
</tr>
<tr>
<td>( (ind = 1 ) – constraint is satisfied, ( ind = 0 ) – constraint is violated)</td>
<td></td>
</tr>
</tbody>
</table>

### Pseudo code:

**BEGIN PROCESS**

Update_inv_ref_int

\[ \begin{align*} & \text{IF} \ u[X] \neq \emptyset \text{ THEN} \\
& \quad \text{SET} \ ind \leftarrow 0 \\
& \quad \text{DO} \ Search\_Null\_value \ \forall \ A \in X \text{ WHILE } ind = 0 \\
& \quad \quad \text{IF} \ u[A] = \emptyset \text{ THEN} \\
& \quad \quad \quad \text{SET} \ ind \leftarrow 1 \\
& \quad \quad \text{ENDIF} \\
& \quad \text{ENDDO} \ Search\_Null\_value \\
& \text{IF} \ ind = 0 \ \text{THEN} \\
& \quad \text{DO} \ Search_t \ \forall \ t \in r(N) \ \text{WHILE} \ ind = 0 \\
& \quad \quad \text{IF} \ \{K_j(R_j) \} \cup \{K_j(R_j) \} \cup \{X\} = \{X\} \text{ THEN} \\
& \quad \quad \quad \text{SET} \ ind \leftarrow 1 \\
& \quad \quad \text{ENDIF} \\
& \quad \text{ENDDO} \ Search_t \\
& \text{ENDIF} \\
& \text{IF} \ ind = 0 \ \text{THEN} \\
& \quad \text{CANCEL\_OPERATION(`Error description`) } \\
& \quad \text{ENDIF} \\
& \text{ENDDO} \ ENDPROCESS PROCESS Update_inv_ref_int \\
\end{align*} \]

![Fig. 5 An algorithm for modification control](image)

**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 database 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 $R_i(N_i)$ and $R_j(N_j)$ 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_<Nj>, called in this trigger is shown in Fig. 9.

![Figure 6 A pattern of the trigger over view](image1)

```
CREATE TRIGGER TRG_<Const_Name> INV_View
ON View_<Nj> INSTEAD OF INSERT
AS
DECLARE @ldt int, @Count int, <Decl_Var_For_Ni_i_Nj>
SELECT <Var_array_for_Ni_i_Nj> FROM Inserted
SET @ldt = @@SPID
exec dbo.Trigger_Ex 0, 'WriteRI_<Nj>',@Idt
INSERT INTO <Nj> VALUES (<Var_array_for_Nj>)
INSERT INTO <Nj> VALUES (<Var_array_for_Nj>)
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
```

![Figure 7 A SQL procedure for trigger execution control](image2)

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

![Figure 8 A tuple insertion control pattern](image3)

```
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>_<Nj> or procedure
Insert_<Const_Name>',16,1)
ROLLBACK TRAN
END
```

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> 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>_i, called in this trigger is shown in Fig. 14.

CREATE OR REPLACE TRIGGER TRG_<Const_Name>_View
    INSTEAD OF INSERT ON View_<N>_<N>
    FOR EACH ROW
DECLARE
    i NUMBER;
    Exc EXCEPTION;
BEGIN
    <Const_Name>_PCK.Trigger_Ex := FALSE;
    INSERT INTO <N> VALUES (<Attr_Value_From_N>);
    SELECT COUNT(*) INTO i FROM <N>
    WHERE (<Selection_Cond>);
    IF i <> 0 THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
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 TrRec<N> IS RECORD (<Attr_Decl_Re_X>);
    TYPE TTabForDelUpd IS TABLE OF TrRec<N> INDEX BY BINARY_INTEGER;
    FOR_<N> TTabForDelUpd;
    Count_IRI NUMBER(8,0);
    Trigger_Ex BOOLEAN;
END;
Fig. 13 A pattern of IRICs’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.

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.

CREATE OR REPLACE TRIGGER TRG_<Const_Name>_INS
    BEFORE INSERT ON <N> FOR EACH ROW
BEGIN
    IF <Const_Name>_PCK.Trigger_Ex = TRUE THEN
        RAISE_APPLICATION_ERROR(-20004, 'Data have to be inserted via view <View_Name> 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 <N>
BEGIN
    <Const_Name>_PCK.Count_IRI := 0;
    <Const_Name>_PCK.For_<N>.DELETE;
END;
Fig. 16 A pattern of the first delete trigger

CREATE OR REPLACE TRIGGER TRG_<Const_Name>_DEL2
    BEFORE DELETE <N>
FOR EACH ROW
DECLARE u <N>%ROWTYPE;
BEGIN
    <Initialization_u>;
    <Name_P>.Count_IRI := <Name_P>.Count_IRI + 1;
    <Name_P>.For_<N> (<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.

```sql
CREATE OR REPLACE TRIGGER TRG_<Const_Name>_DEL3
AFTER DELETE ON <N>
DECLARE
  u <N>%.ROWTYPE;
  i NUMBER;
BEGIN
  FOR j IN 1.. <Const_Name>_PCK.Count_IRI LOOP
    SELECT COUNT(*) INTO i FROM <N>
    WHERE <Selection_Cond>;
    IF i <> 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>` 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