The Extended Referential Integrity Constraint Type – Specification and Implementation in Relational and XML Database Management Systems

Jovana Vidaković*, Sonja Ristić**, Slavica Kordić***, Ivan Luković***

* University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Novi Sad, Serbia
** University of Novi Sad, Faculty of Technical Sciences, Department of Industrial Engineering and Engineering Management, Novi Sad, Serbia,
*** University of Novi Sad, Faculty of Technical Sciences, Department of Computing and Control, Novi Sad, Serbia,
{jovana, sdristic, slavica, ivan}@uns.ac.rs

***Abstract*— The referential integrity constraint (RIC) type is one of the constraint types of relational data model. It is defined between at most two, not necessarily different, relation schemes. In the course of the third normal form database design, there may be necessary to define the referential integrity constraint between more than two relation schemes. Therefore, an extended referential integrity constraint (ERIC) type is defined. A constraint of this type spans more than two relation schemes. ERIC is defined in the relational data model and its implementation in relational database management systems can be done procedurally, by means of database triggers, which is presented in this paper. Unlike relational data model, XML data model does not recognize ERIC type. In this paper we specify the extended referential integrity constraint type in XML data model and propose techniques for the implementation of ERICs in XML database management systems.**

I. INTRODUCTION

Every database management system (DBMS) is based on a data model. It provides the means to express database schema and has a specific set of integrity constraint types. There are well-known constraint types that are common for most data models. The referential integrity constraint (RIC) type is one of the common constraint types in relational and eXtended Markup Language (XML) data model. It is a special case of inclusion dependency (IND) constraint type [1, 2, 3]. Relational database management systems (RDBMSs) support declarative mechanisms for the implementation of RICs. Therefore, they are broadly accepted and used in the relational database design and implementation. Besides the common constraint types, there are more complex ones, that are specific for a data model. These specific constraint types are very often difficult to define, implement and enforce. One of those constraint types is the extended referential integrity constraint (ERIC) type. It is an extension of the RIC type and, consequently, it is another special case of IND constraint type. Detail description of ERIC type is given in second section.

In this paper we explain the need for introducing the extended referential integrity constraint type in the relational data model and XML data model. RDBMSs do not support declarative mechanisms for the implementation of ERICs. That is the main reason why the database designers avoid to specify and to implement ERICs. Here, we propose an implementation of the ERIC in Oracle DBMSs. Oracle DBMS use triggers as a procedural mechanism for ERIC implementation. Since ERIC can be violated by critical operations of inserting, modifying or deleting, we present several triggers which are raised before these critical operations.

Unlike relational data model, XML data model does not recognize ERIC type at all. Therefore, here we specify the ERIC type in the XML data model and propose two techniques for implementation of ERIC in XML DBMSs, which depend on the characteristics of a selected XML DBMS. In presented implementations XQuery functions are used in eXist DBMS, and triggers are used in Sedna DBMS.

Apart from Introduction and Conclusion, this paper is organized as follows. In Section 2 we present an example to explain why it is important to recognize, specify and implement ERICs. In Section 3 an implementation of an ERIC by means of RDBMs is given. Aspects of ERIC specification and implementation in XML data model are discussed in Section 4. Related work is presented in Section 5.

II. THE EXTENDED REFERENTIAL INTEGRITY CONSTRAINT TYPE

An extended referential integrity constraint in relational database schema [4] is defined between more than two relation schemes. For example, let the relational database schema models suppliers, articles and price list for each article supplied by a supplier, orders made for suppliers and order items of these orders. The relational database schema contains following relation schemes:

Supplier({*SupID*, *SupName*}, {*SupID*}),
Article({*ArtID*, *ArtName*, *ArtUn*}, {*ArtID*}),
PriceList({*SupID*, *ArtID*, *Price*}, {*SupID*+*ArtID*}),
Order({*OrdID*, *SupID*, *Total*}, {*OrdID*}), and
OrderItem({*OrdID*, *SupID*, *ArtID*, *ItemNo*, *Quantity*}, {*OrdID*+*SupID*+*ArtID*}).

SupID and *ArtID* are primary keys of relation schemes *Supplier* and *Article*, respectively. The primary key of the relation scheme *PriceList* is composed of two foreign keys *SupID* and *ArtID*. The relational database schema contains two referential integrity constraints: *PriceList*[*SupID*] ⊆ *Supplier*[*SupID*] and *PriceList*[*ArtID*] ⊆ *Article*[*ArtID*]. Here we use the standard notation to specify inclusion dependency constraint as it is presented in [3]. *PriceList* and *Supplier* are names of relation schemes. An argument within the square brackets represents an array of attributes from the relation scheme attribute set. In case of the left side of IND the attribute *SupID* is from the attribute set of relation scheme *Pricelist* and on the right side of IND it is the attribute *SupID* from the attribute set of relation scheme *Supplier*. If the length of that array is greater than one, attributes are comma separated.

The relation scheme *Order* has a primary key *OrdID* and a foreign key *SupID*, so the referential integrity constraint has to be defined as: *Order*[*SupID*] ⊆ *Supplier*[*SupID*].

The relation scheme *OrderItem* has two foreign keys: *OrdId* and *SupID+ArtID*, which means that two referential integrity constraints have to be defined: *OrderItem*[*OrdID*] ⊆ *Order*[*OrdID*] and *OrderItem*[*SupID, ArtID*] ⊆ *PriceList*[*SupID, ArtID*]. However, in the relation scheme *Order*, the functional dependency *OrdID* → *SupID* is valid, which means that the attribute *SupID* is redundant in the relation scheme *OrderItem*. The supplier which supplies the article is the same supplier whose order is made. Therefore, the relation scheme is not in the third normal form and it would be normalized.

The normalized relational database schema contains following schemes:

Supplier({*SupID, SupName*}, {*SupID*}),

Article({*ArtID, ArtName, ArtUn*}, {*ArtID*}),

PriceList({*SupID, ArtID, Price*}, {*SupID+ArtID*}),

Order({*OrdID, SupID, Total*}, {*OrdID*}), and

OrderItem({*OrdID, ArtID, ItemNo, Quantity*}, {*OrdID+ArtID*}).

It still contains the referential integrity constraints *OrderItem*[*OrdID*] ⊆ *Order*[*OrdID*] and *Order*[*SupID*] ⊆ *Supplier*[*SupID*]. Unfortunately, in the normalized relational database schema it is not possible to define the referential integrity *OrderItem*[*SupID, ArtID*] ⊆ *PriceList*[*SupID, ArtID*], because *SupID* does not belong to the attribute set of *OrderItem* relation scheme anymore. Without such a constraint it could not be ensured that order items contain only articles supplied by the same supplier to whom the order is sent. Therefore, there is a need to extend the RIC type allowing a join of two or more relations to appear on referencing or referenced side of RIC specification. This constraint type is called the extended referential integrity constraint type. Now, it is possible to define an ERIC over the relation schemes *OrdetItem*, *Order* and *PriceList*:

$$\bowtie(\mathit{OrderItem}, \mathit{Order})[\mathit{SupID}, \mathit{ArtID}] \subseteq \mathit{PriceList}[\mathit{SupID}, \mathit{ArtID}],$$

where ⋈ is the join operator between two relations over the relation schemes *OrderItem* and *Order*, respectively. Hereinafter a relation over a relation scheme will be named after that relation scheme. Therefore, the notation „relation *Order*“ will be used to refer the relation over the relation scheme *Order*. In order to obey this constraint a tuple from ⋈(*OrderItem*, *Order*) relation must not contain a pair of *SupID* and *ArtID* values that is not already contained in a tuple of relation *PriceList*. In that way it is ensured that an order can contain only the articles supplied by its supplier.

In this example the constraint is characterized as an ERIC type because there is a join of two relations on the one side (in this case on the left side) of inclusion dependency. Generally speaking, join may appear on both sides of IND and it can be a join between more than just two relations. In general, the join may be a theta-join (θ-join), an equijoin or a natural join. Formally, we specify an ERIC as follows:

$$(N_{l1} \bowtie_{JCl1} N_{l2} \bowtie_{JCl2} \bowtie ... \bowtie_{JCl(n-1)} N_{ln})[X] \subseteq (N_{r1} \bowtie_{JCr1} N_{r2} \bowtie_{JCr2} \bowtie ... \bowtie_{JCr(m-1)} N_{rm})[Y],$$

where N_i is the name of relation scheme $N_i(R_i, C_i)$, R_i is set of relation scheme attributes, C_i set of relation scheme constraints (like domain, not null, key or unique constraints), JC_i is join condition of θ -join between relation schemes N_i and N_{i+1}, and X and Y are the arrays of attributes from the union $\bigcup_{i=1}^{m} R_{li}$ and from the union $\bigcup_{i=1}^{m} R_{ri}$, respectively. The specification of a join condition is missed if an ERIC is defined over a natural join. This constraint is validated over the projections of included relations:

$$\pi_X(r(N_{l1} \bowtie_{JCl1} N_{l2} \bowtie_{JCl2} \bowtie ... \bowtie_{JCl(n-1)} N_{ln})) \subseteq \pi_Y(r(N_{r1} \bowtie_{JCr1} N_{r2} \bowtie_{JCr2} \bowtie ... \bowtie_{JCr(m-1)} N_{rm})) .$$

III. The Implementation of Extended Referential Integrity Constraint in Relational Data Model

The definition of each constraint contains the list of the operations that can violate a constraint. Those operations are called critical operations. The extended referential integrity constraint in the example given in the section 2 can be violated in several cases. Therefore, the critical operations of that ERIC are: insertion of a new order item, modification of an order item, modification of an order, modification and deleting of an existing price list item. First, we will explain how these critical operations can violate the ERIC in relational database schema. To easy the further explanations we say that a tuple of relation *Order* matches to an *OrderItem* tuple if they have the same *OrderId* value. The critical operations are implemented in RDBMS using procedural mechanism, i.e. database triggers.

Since the relation scheme *OrderItem* does not contain an attribute *SupID*, if a new tuple is inserted in the relation *OrderItem*, we have to check if the supplier to whom the order is sent supplies the article in that order item. It means that in relation *PriceList* there must be a tuple with *ArtID* value same as the *ArtID* value in the tuple inserted in *OrderItem*. At the same time, *SupID* value of that tuple from *PriceList* has to be same as the *SupID* value in the *Order* tuple that matches to the inserted *OrderItem* tuple.

A critical modification of an existing tuple in the relation *OrderItem* referes to the changing of *ArtID* or *OrdID* values. In both cases, the existence of a tuple in relation *PriceList* with the *ArtID* value same as the *ArtID* value in modified *OrderItem* tuple, and with the *SupID* value same as the *SupID* value of the *Order* tuple that

matches modified *OrderItem* tuple, would be checked. If it does not exist the critical operation must be rejected.

A modification of a tuple in the relation *Order* refers to the modification of the attribute *SupID* value, since the attribute *OrdID* is the primary key of the relation scheme *Order* and its value cannot be updated. If we want to change the supplier, we have to check if a new supplier supplies all articles from that order, i.e. all items of that order. Therefore, for each *OrderItem* tuple that is matching with modified *Order* tuple, the existence of a tuple in relation *PriceList* with the *ArtID* value same as the *ArtID* value in *OrderItem* tuple, and with the *SupID* value same as the *SupID* value of the modified *Order* tuple should be checked. If it does not exist for at least one of these *Ordertem* tuples, the modification of the *Order* tuple must be rejected. In this paper we present the database trigger that is raised before the update of an *Order*. It is given in the Figure 1.

```
CREATE OR REPLACE TRIGGER
Cons_Order_PriceList_ExRefIn
   BEFORE UPDATE OF OrdID, SupID ON Order
     FOR EACH ROW
     WHEN (OLD. SupID!= NEW. SupID OR
     OLD. OrdID!= NEW. OrdID)
          DECLARE I_NumRow NUMBER(4);
          BEGIN
             IF UPDATING(OrdID)
             THEN
               Raise_Application_Error (-20000, '<Message>');
            ELSE
              SELECT COUNT(*) INTO I_ NumRow
              FROM OrderItem s
              WHERE s. OrdID = :OLD. OrdID
                    AND  (:NEW. SupID, s.ArtID) NOT IN
                    (SELECT SupID, ArtID FROM PriceList);
            IF I_ NumRow!= 0 THEN
              Raise_Application_Error (-20001, '<Message>');
                    END IF;
             END IF;
   END Cons_ Order_PriceList_ExRefIn;
```

Figure 1. Trigger before tuple update from *Order*

A modification of a tuple in relation *PriceList* refers only to updating a price, since the attributes *SupID* and *ArtID* are parts of the primary key of the relation scheme *PriceList* and their values cannot be updated. This operation is not critical in the context of the ERIC.

If we want to delete a tuple from the relation *PriceList*, first we have to check if there are any tuples from the join of relations *OrderItem* and *Order* that reference this tuple from the relation *PriceList*. If they exist, the deletion have to be forbidden or the cascade deletion has to be forced.

```
CREATE OR REPLACE TRIGGER
Cons_PriceList_OrderItem_ExRefIn
          BEFORE DELETE OR UPDATE OF SupId, ArtId
          ON PriceList
          FOR EACH ROW
          WHEN (OLD.SupId != NEW.SupId OR OLD.ArtId
!= NEW.ArtId)
          BEGIN
             IF UPDATING THEN
               Raise_Application_Error (-20000, '<Message>');
             ELSIF DELETING
                      AND        Cons_ExRefIn_CheckOrderItem
(:OLD.SupId, :OLD.ArtId) THEN
                      /* Deleting forbidden, if a tuple is referenced
in OrderItem */
                         Raise_Application_Error          (-20001,
'<Message>');
             END IF;
END Cons_PriceList_OrderItem_ExRefIn;
```

Figure 2. Trigger before tuple delete from *PriceList*

We have an additional function which has to check if there is a tuple in the *OrderItem* with the same values of *ArtId* and *SupId* like in the relation *PriceList*. If this is the case, the deletion of this tuple in *PriceList* will not be possible. This function is given in the Figure 3.

```
CREATE OR REPLACE FUNCTION
Cons_ExRefIn_CheckOrderItem
             (p_SupId IN PriceList.SupId%TYPE,
              p_ArtId IN PriceList.ArtId%TYPE)
          RETURN BOOLEAN IS
            I_exists NUMBER(1);
          BEGIN
            SELECT 0 INTO I_exists
            FROM dual
            WHERE EXISTS
                        (SELECT 0
                         FROM OrderItem s, Order p
                         WHERE  s.OdrId = p.OrdId
                         AND  s.ArtId = p_ArtId
                         AND  p.SupId = p_SupId);
            RETURN TRUE;
          EXCEPTION
            WHEN NO_DATA_FOUND THEN
                    RETURN FALSE;
   END Cons_ExRefIn_CheckOrderItem;
```

Figure 3. Function for tuple existence checking of *OrderItem*

IV. The Implementation of Extended Referential Integrity Constraint in XML Data Model

In our previous research [5, 6, 7] we have dealt with different types of constraints and proposed their specification and validation in XML databases since they do not completely support it. Like in the relational data model, there is a need to use ERIC type in XML data model, too. In this paper we specify this constraint type in the XML data model and propose two techniques for implementation of ERICs in XML DBMSs, used in two different XML DBMSs. We use XQuery functions in eXist DBMS, and triggers in Sedna DBMS.

The detected critical operations for an ERIC can be processed in XML DBMSs. Considerations are similar to the ones in relation database schema. We have used different XML DBMSs according to the level of support for triggers and tested the validation of the ERIC. We

have implemented constraints using triggers in XML DBMS Sedna, and XQuery functions in eXist.

The definition of the extended referential integrity constraint for XML data model is similar to the definition of this constraint for relational data model. ERIC is defined between two or more element types. Let E be an element type and $Attr(E)$ be a set of attributes specified within the element type E. The definition scope of an ERIC in XML model is an array of element types E_1,..., E_m. In [5] constraint taxonomy and constraint type formal specification for XML data model are proposed. According to [5] constraint type formal specification is named 5-tuple. Therefore, the formal specification of ERIC type for XML data model is given as follows:

ExRICon(*n*, *me*,
⋈(E_1, …, E_n)@Z⊆⋈(E_1, …, E_k)@Z,
⋈(e_1, …, e_n)@Z⊆⋈(e_1, …, e_k)@Z,
((e_i, *referencing*, {(*insert*, *Restrict*),
(*update*, *Restrict*)}),
(e_j, *referenced*, {(*delete*, *Restrict*, *cascade*),
(*update*, *Restrict*, *cascade*)})).

In the following text the explanations of the components of the aforementioned specification of ERIC type are given. *ExRICon* is the name (label) of ERIC type. First tuple element is definition scope of ERIC type and its value is „*n*“ (more), which means that at least two element types have to be included in the specification of ERIC type. Next tuple element indicates the scope of constraint type interpretation. For ERIC type it is "more elements" constraint (*me*) since it is interpreted over the values from more than one tuples (elements) gathered by joining of two or more elements. Third tuple element is formula pattern ⋈(E_1, …, E_n)@Z⊆⋈(E_1, …, E_k)@Z, which means that this constraint is defined over the join of two or more element types E_1, …, E_n, i.e. over the subset Z which belongs to the union of attribute sets $Attr(E_1)$, … $Attr(E_n)$. It is interpreted over the projection of tuples (e) on the subset Z, which belongs to the union of attribute sets $Attr(E1)$, … $Attr(Em)$, that is specified by forth tuple element. The last tuple element is 3-tuple containing element, role and set of pairs (critical operation, set of actions). For the ERIC role of an element can be referencing or referenced. Critical operations for the ERIC, are insert or update of one of the referencing elements, or delete or update of the referenced elements. Critical operations may violate database consistency in regard to an ERIC. For each critical operation actions can be defined that would be carried out to maintain database consistency when this violation occurs. For all ERIC type critical operations set of possible actions is singleton that contains action *Restrict* or *Cascade*. It means that if an ERIC is violated the critical operation will not be executed, or it will be cascaded.

In XML data model, we specify the ERIC over three elements: *OrderItem*, *Order* and *PriceList*, in the similar way like in the relational data model.

According to [5] the specification of a given constraint is named 3-tuple. Here we present the specification of XML ERIC constraint that is analogous to the relational ERIC constraint in the following way:

ExRefIntOrderItem_PriceList(*ExRefIntCon*,
(*OrderItem*⋈*Order*)@(*SupId*, *ArtId*) ⊆
PriceList@(*SupId*, *ArtId*),
((*Order*, *referencing*, {(*insert*, /),
(*update* (*OrdId*, *SupId*), *Restrict*)}),
(*OrderItem*, *referencing*, {(*insert*, *Restrict*),
(*update* (*OrdId*, *ArtId*), *Restrict*)}),
(*PriceList*, *referenced*, {(*delete*, *Restrict*),
(*update* (*SupId*, *ArtId*), *Restrict*)}))

ExRefIntOrderItem_PriceList is the name of the XML ERIC. The first element of the specification is the label of constraint type (*ExRefIntCon*). In the way the specification of this constraint have to be in line with the specification of ERIC type. ERIC is defined over the set of attributes from three element types: *Order*, *OrderItem* and *PriceList*. These element types are listed within the second element of 3-tuple. Attributes of interest are *SupId*, *ArtId* and *OrdId*. Critical operations, alongside with actions aimed at maintaining database consistency are specified in the third 3-tuple element. In the particular ERIC critical operations are insert of a new order item and update of the values of *OrdId* or *SupId* in the element *Order*, or *OrdId* or *ArtId* in the element *OrderItem*. Also, delete or update of the element *PriceList* are critical operations. These operations can be restricted or cascaded.

One of the critical operations is updating of the element *OrderItem*. In the Figure 4 we present the trigger in Sedna that is raised before update of the element *OrderItem*. If the supplier from the order does not supply any of the articles in order items, the extended referential integrity constraint is violated and the modification is not possible.

```
CREATE TRIGGER "ExRIOrderItemBeforeUpdate"
BEFORE REPLACE
ON collection('ExRI')/Orders/OrderItem
FOR EACH NODE
DO {
        if(exists(fn:doc('Orders', 'ExRI')/Orders/PriceList
        [@ArtID = $NEW/@ArtID
        and
        @SupID = fn:doc('Orders',  'ExRI')/Orders/Order
        [@OrdID = $NEW/@OrdID]/@SupID]))
        then
                ($NEW)
        else
        error(xs:QName("ExRIOrderItemBeforeUpdate"),"Or
derItem cannot be updated - Extended Referential Integrity
Constraint Violated!");
}
```

Figure 4. Trigger before update of tuple from *OrderItem*

If we use eXist XML DBMS then the XQuery functions have to be implemented. In the Figure 5 we present the XQuery function that is executed when the update of the element *OrderItem* has to be done. Since the primary key *OrdId* cannot be updated, only the update of the attribute *SupID* is possible. In this case, a new supplier has to provide the articles on that order, i.e. the items of that order.

```
declare function local:canUpdateOrderItem($NEW as
element(OrderItem))
    as xs:boolean{
      let $exists :=
```

```
    exists(doc('Orders.xml')/Orders/PriceList[@ArtId =
   $NEW/@ArtId and @SupId =
   doc('Orders.xml')/Orders/Order[@OrdId =
   $NEW/@OrdId]/@SupId])
     return ($exists)
   };
declare function local:doUpdateOrderItem($OLD as
element(OrderItem), $NEW as element(OrderItem))
      as xs:boolean{
        let $i := update replace
          doc('Orders.xml')/Orders/OrderItem[@OrdId=
          $OLD/@OrdId and @ArtId=$OLD/@ArtId]
          with $NEW
       return true()
   };

declare function local:updateOrderItem($OLD as
element(OrderItem), $NEW as element(OrderItem))
      as xs:boolean{
        let $res :=  if(local:canUpdateOrderItem($NEW))
        then local:doUpdateOrderItem($OLD, $NEW)
        else false()
        return $res
   };
```

Figure 5. XQuery function before update of tuple from *OrderItem*

V. Related Work

Maintaining compliance of data with respect to specified integrity constraints is a crucial database issue. Besides common integrity constraint types such as domain integrity, not null constraint type or referential integrity constraint type, real-world applications may involve nontrivial integrity requirements that capture complex data dependencies and business rules. Referential integrity (foreign key) constraint type, as a special case of inclusion dependency constraint type, plays an important role in data modeling. In relational data model, it is well studied and is widely used [2, 3]. Most of the commercial DBMSs, based on different data models like relational or XML data model, offer efficient declarative support for the domain constraints, null value constraints, uniqueness constraints and referential integrity constraints [8]. On the contrary, more complex constraint types are completely disregarded by actual relational and XML DBMSs, obliging the users to manage them in application. Türker and Gertz in [9] emphasize that enforcing integrity constraints and business rules with declarative constraints or via custom procedures or triggers often is less costly than enforcing the equivalent rules within an application. In that way logical data independence is preserved, too. Attaulah and Tompa in [10] stress that several problems like the lack of transparency and lack of manageability are caused by the absence of a centralized policy and constraint management system within database systems. Therefore, it is important to take these complex constraint types in the consideration, too.

The ERIC type is one of the complex constraint types that can be used to model nontrivial integrity requirements but to the best of our knowledge there are no papers that define and explain this type of constraint in relational or XML data model.

In recent years, XML has gained a wide acceptance as data representation and storage format. XML integrity constraints, just as in relational data model play an important role to keep XML dataset as consistent as possible. XML databases offer specification and implementation of several constraint types, like keys, foreign keys and unique constraints [11, 12]. Other constraints that exist in large database project are not declaratively supported in XML DBMSs.

Karlinger, Vincent, and Schrefl in [13] define an XML inclusion constraint (XIND), and show that it extends the semantics of a relational inclusion dependency. Shahriar and Liu in [14] propose XML Inclusion Dependency (XID) and XML foreign key (XFK) constraints and show how both XID and XFK can be defined over the Document Type Definition (DTD) and are satisfied by the XML documents. In [15] they discuss the preservation of referential integrity constraint in XML data. All of these papers discus simple non-extended inclusion dependencies and referential integrity constraint type.

In our paper we deal with ERIC type specification and implementation in XML databases. The usual way of implementing constraints in XML DBMSs is using triggers [16]. The XML DBMS that supports triggers in a way similar to relational databases is Sedna [17]. On the other hand, there are XML DBMSs which do not support triggers in the appropriate extent, such as eXist [18]. In this paper we have proposed two ways of ERIC implementation based on XQuery functions [19] and triggers, depending on the level of support for triggers in XML DBMSs.

VI. Conclusion

In this paper we discuss the need for using the extended referential integrity constraint. We present the definition of this constraint in the relational data model as well as the implementation in RDBMS using triggers. We also present two approaches to the extended referential integrity constraint specification and implementation in XML databases. One is based on the usage of XQuery functions, while the other is based on triggers. These approaches depend on the level of trigger support in a selected XML DBMS. A support of triggers in the eXist DBMS is a quite basic, so it cannot be used for the ERIC implementation. In this case, we have proposed a usage of the XQuery functions. XML DBMS Sedna offers the appropriate trigger support.

Future work will cover the generation of XQuery functions and triggers for constraint validation. This process could be fully automated and based on the paradigm of a model driven software engineering and the appropriate transformation specifications. A research work on a code generator development is in progress. This code generator would make database designer's and developer's job easier and free them from manual coding

of XQuery functions and triggers for validating constraints.

ACKNOWLEDGMENT

The research presented in this paper was supported by Ministry of Education, Science and Technological Development of Republic of Serbia, Grant OI-174023.

REFERENCES

[1] C. J. , Date, H. Darwen: Types and the Relational Model. The Third Manifesto, 3rd ed. Addison Wesley, 2006.

[2] R. Elmasri, B. S. Navathe: Fundamentals of Database Systems, Seventh Edition, Pearson Global Edition, 2015, ISBN 978-0133970777

[3] S. Ristić, S. Aleksić, M. Čeliković, I. Luković: Generic and Standard Database Constraint Meta-Models, Computer Science and Information Systems 11(2):679–696, 27 doi:10.2298/CSIS140216037R

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

[5] J. Vidaković, I. Luković, S. Kordić: Specification and Implementation of the Inverse Referential Integrity Constraint in XML Databases, 7th Balkan Conference in Informatic, 2015, ACM New York, USA, ISBN 978-1-4503-33351, DOI: 10.1145/2801081.2801111

[6] J. Vidaković, S. Ristić, S. Kordić, I. Luković: Extended Tuple Constraint Type in Relational and XML Data Model – Definition and Enforcement. In Proceedings of BCI '17, September 20–23, 2017, Skopje, Macedonia, 8 pages. https://doi.org/10.1145/3136273.3136294

[7] J. Vidaković, I. Luković, S. Kordić: Specification and Validation of the Referential Integrity Constraint in XML Databases, Proceedings of the 6th International Conference on Information Society and Technology, 2016, Kopaonik, Serbia, ISBN 978-86-85525-18-6, pp. 197-202.

[8] H. Decker, D. Martinenghi: Database Integrity Checking. In Database Technologies: Concepts, Methodologies, Tools, and Applications, 2009, ed. John Erickson, 212-220, doi:10.4018/978-1-60566-058-5.ch016.

[9] C. Türker, M. Gertz: 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)

[10] A. A. Ataullah, F. Tompa: Business Policy Modelling and Enforcement in Databases,2011, PVLDB Vol. 4, No. 11, 921-931, 2011

[11] P. Buneman, W. Fan, J. Simeon, S. Weinstein: Constraints for Semistructured Data and XML, SIGMOD Record, 2001, pp. 47-54

[12] W. Fan: XML Constraints: Specification, Analysis, and Applications, DEXA, 2005, pp.805-809.

[13] M. Karlinger, M. W. Vincent, M. Schrefl: Inclusion dependencies in XML: Extending relational semantics. In S. S. Bhowmick, J. Kung, and ¨ R. Wagner, editors, DEXA, volume 5690 of LNCS, pages 23–37. Springer, 2009.

[14] M. S. Shahriar, J. Liu: On Defining Referential Integrity for XML. In: IEEE International Symposium of Computer Science and Its Applications(CSA), pp. 286–291, 2008

[15] Md. S. Shahriar, J. Liu: Towards the Preservation of Referential Constraints in XML Data transformation for Integration, International Journal of Database Theory and Application, vol. 3, 2010, pp. 1-10.

[16] M. Grinev, M. Rekouts: Introducing Trigger Support to XML Database Systems, Proceedings of the Spring Young Researcher's Colloquium on Database and Information Systems SYRCoDIS, St. Petersburg, Russia, 2005.

[17] Sedna, Native XML Database System, www.sedna.org

[18] eXist, http://exist-db.org/exist/apps/homepage/index.html

[19] XQuery, http://www.w3.org/TR/xquery/