Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Adaptive XML to relational mapping: an integrated approach*

Tian-lei HU†, Gang CHEN

(*School of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*)

†E-mail: htl@zju.edu.cn

Received Dec. 5, 2007; revision accepted Feb. 25, 2008

Abstract: Storing and querying XML (eXtensible Markup Language) data in relational form can exploit various services offered by modern relational database management systems (RDBMSs). Due to structural complexity of XML, there are many equivalent relational mapping schemes for the same XML data and queries. In this paper, we propose the adaptive XML to relational mapping (AX2RM) system, which considers finding optimal XML to relational (X2R) mapping as four separate but correlated procedures: logical database design, data scale estimation, workload transformation, and physical database design. We view the whole process as an autonomic computing problem and formalize the adaptive X2R mapping problem. Search spaces for each procedure are investigated individually, and five approaches for finding the optimal mapping are studied. We propose an integrated approach with greedy pruning (IT-GP), which views the mapping procedures as a whole and exploits heuristic rules in each procedure to prune impossible mappings as early as possible. Evaluation of these approaches shows the validity and high efficiency of IT-GP.

Key words: XML, XML to relational (X2R) mapping, Autonomic computing
doi:10.1631/jzus.A0720103 **Document code:** A **CLC number:** TP315

INTRODUCTION

In the era of Internet, XML becomes an important medium for storing, representing, retrieving and exchanging data. There are growing interests in mapping XML data into relational databases, in order to exploit a complete set of data management services (including query optimization, concurrency control, crash recovery, and scalability) offered by modern relational database management systems (RDBMSs). However, being a tree structure, XML data is naturally far more complex than flat, two-dimensional relational data. There are many equivalent means to store and query XML data in relational form. This paper proposes an integrated system, the adaptive XML to relational mapping (AX2RM) system, for finding the best XML to relational (X2R) mapping.

Schema mapping between XML data and relational data, as well as query mapping between XML queries and SQL queries, has been heavily investigated (Florescu and Kossmann, 1999; Krishnamurthy *et al.*, 2003; Chen, 2004; Atay, 2006) recently. However, schema mapping and query mapping are correlated, therefore they should not be studied separately. Moreover, concerning schema mapping only, there are two related phases, logical design and physical design. Chaudhuri *et al.*(2005) have shown that independently considering these two phases will also lead to sub-optimal mappings.

In the AX2RM system, we solve the mapping problem in an integrated and iterative manner. AX2RM views the mapping procedure as four separate but correlated procedures: logical database design, data scale estimation, workload transformation, and physical database design. We formally define the related concepts and formalize the AX2RM problem. We investigate search spaces for each procedure and present five approaches for finding the optimal mapping. The integrated approach with greedy pruning (IT-GP), the main contribution of this paper, views the mapping procedures as a whole and exploits heuristic rules in each procedure to prune impossible mappings

* Project supported by the National Natural Science Foundation of China (No. 60603044), the China Postdoctoral Science Foundation (No. 20070411179), and the Program for Changjiang Scholars and Innovative Research Team in University of China (No. IRT0652)

as early as possible. Evaluation shows that IT-GP will produce relatively good mappings in a short time.

RELATED WORKS

Since 1999, database research literature has seen an explosion of publications aiming at using RDBMS to store and query XML data (Florescu and Kossmann, 1999; Krishnamurthy *et al.*, 2003; Chen, 2004; Atay, 2006). To store XML in relational form, we need to map the XML schema to the relational schema, and further to the physical schema; and to query XML stored in relational form, we need to translate XML queries into SQL queries based on schema mapping. Thus, X2R mapping problem contains three sub-problems: the problem of mapping XML schema to relational schema (logical mapping problem), the problem of mapping relational schema to physical schema (physical mapping problem), and the problem of translating XML query to SQL query (workload translation problem).

For solving the logical mapping problem, there are two distinct classes of techniques. One class is schema-oblivious XML storage (Krishnamurthy *et al.*, 2003), which exploits the same relational schema to store different XML documents/schemas. A typical schema oblivious approach is the edge approach (Florescu and Kossmann, 1999), which views an XML document as a graph, with each edge in the graph corresponding to a tuple in a single relation. The other class is schema-based XML storage (Krishnamurthy *et al.*, 2003), which chooses different relational schemas for different XML schemas. Inlining is a typical schema-based XML storage technique, and there are three kinds of inlining schemas proposed in (Shanmugasundaram *et al.*, 1999)—basic inlining, shared inlining and hybrid inlining. The basic idea of these schemas is to inline all elements that occur at most one time per parent element into parent relation. Lee and Chu (2000) study a constraint-preserving algorithm for transforming XML DTD to the relational schema. The authors choose hybrid-inlining algorithm from (Shanmugasundaram *et al.*, 1999) and show how semantic constraints can be generated.

The physical mapping problem, a topic in RDBMS domain, is studied a lot as the classical automatic physical database design problem (Rozen and Shasha, 1991; Zilio *et al.*, 2004; Agrawal *et al.*, 2004a; 2004b; Hu *et al.*, 2006). Appropriately choosing physical features, such as indexes, materialized views, vertical/horizontal partitions, compression schemas, replications, and clusters, is the key to improve the RDBMS performance. As we store XML documents in RDBMS, selection of physical features will heavily affect the performance of X2R mapping (Chaudhuri *et al.*, 2005).

For workload translation, most literature focuses on translation correctness (Krishnamurthy *et al.*, 2003; 2004). Beyond correctness, we need to find translation with the "best" quality. There are roughly two categories of quality-concerning approaches: SQL optimization and intelligent query translation. SQL optimization approaches exploit optimization ability of RDBMS to promote the quality after translation (Deutsch and Tannen, 2003). Besides them, intelligent query translation approaches leverage semantics in XML documents when executing translation (Krishnamurthy *et al.*, 2004; Mani *et al.*, 2006).

As far as we know, there is no publication providing a sophisticated solution to address all these problems thoroughly. Research efforts most similar to ours are the LegoDB approach (Bohannon *et al.*, 2002) and the study on the interplay of logical and physical X2R mappings by Microsoft research (Chaudhuri *et al.*, 2005). LegoDB views the problem of choosing a good relational schema as an optimization problem and exploits a greedy algorithm for this purpose, omitting details on workload transformation and physical database design. The latter concerns the interaction of logical and physical designs, however, it neglects query translation and postpones the query translation to the optimization phase in RDBMS, which also leads to suboptimal mappings (Krishnamurthy *et al.*, 2003).

Without integrally addressing these issues, optimal X2R mapping cannot be achieved. In the above-mentioned research, due to neglecting the mapping from the relational schema to the physical schema, it is impossible to utilize the physical design features and the optimizing capabilities of RDBMS, while neglecting translation query problems makes it impossible to exploit semantic information in XML queries. We will introduce the AX2RM system which addresses these problems in an integrated and iterative manner, and will show its feasibility and validity.

ARCHITECTURE AND RATIONALE OF AX2RM SYSTEM

Architecture overview

Fig.1 illustrates the architecture of the AX2RM system. Inputs of the system are XML schema (DTD/XML schema), XML workload (path expressions/XPath/XQuery and their frequency), and XML data. The system generates an optimal relational schema and a physical schema to store the XML data, together with optimal translation of the XML workload. The AX2RM system contains an integrated and iterative X2R mapping procedure, which is divided into four correlated sub-procedures: logical database design (LD), data scale estimation (SE), workload translation (WT), and physical database design (PD).

In the LD procedure, XML schema is mapped into an equivalent relational form. Since there are many possible mappings, XML workload and XML data are taken into account to determine the optimal one. Note that the "optimal mapping" costs the lowest when executing the translated SQL workload on the produced relational database schema. The optimal mapping is hence determined by the workload and the physical schema, which are outcomes of the other two procedures.

Concerning a specific logical design, the relational schema will be used for both estimating the relational database scale in the SE procedure and translating XML workload into SQL workload in the WT procedure. The SE procedure analyzes the input XML data and generates pseudo statistics for each relation and each attribute. However, for the same relational mapping schema and XML query, there are many equivalent but different query translations (Krishnamurthy *et al.*, 2003). The WT procedure tries to find the best translation.

After collecting information on the relational schema, data scale, and workloads, the problem of finding the optimal X2R mapping evolves into a classical problem of finding the optimal physical database design, i.e., the automatic physical database design problem in RDBMS domain. The PD procedure finds the optimal physical database design schema (indexes, materialized views, partitions, compressions, etc.) to maximize the performance.

All these procedures are optimization procedures except the SE, and each procedure depends on consequent procedures in that the objective function of a procedure is the optimization procedure of the consequent ones. For instance, the objective function of the LD procedure is the minimum cost of the workload in a specific relational schema, where the minimum cost is decided by the WT and the PD. AX2RM architecture is a flexible framework, and current X2R mapping techniques can incorporate seamlessly into it. We will exploit merits of different techniques to generate the best X2R mapping iteratively and adaptively.

Rationales for XML to relational mapping

For further discussion, we formalize the concepts of X2R mapping in this subsection. Since describing all kinds of schema-mapping methods and XML query representation based on a single infra-

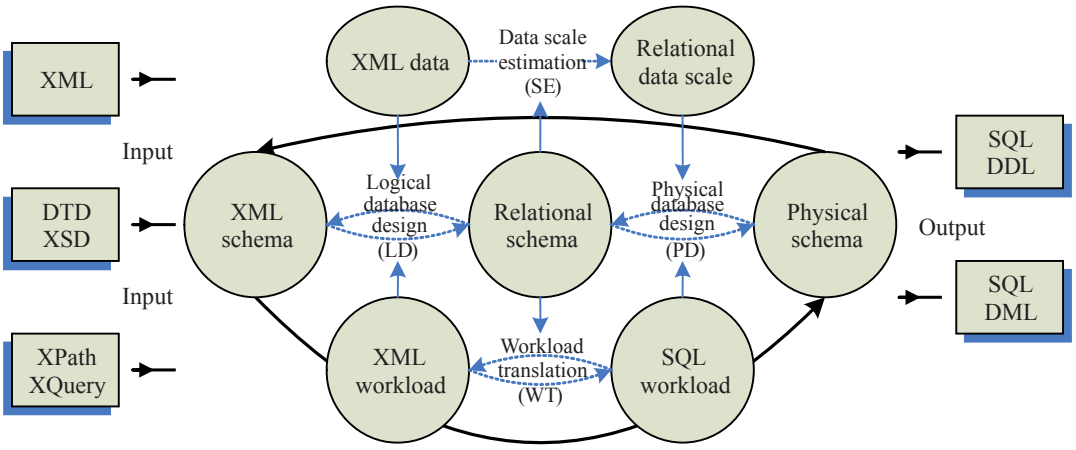


Fig.1 Architecture of the adaptive XML to relational mapping system. There are four correlated procedures in the system: the logical database design procedure, the data scale estimation procedure, the workload translation procedure, and the physical database design procedure

structure is too complicated, we limit our discussion to relatively small categories: only schema-based X2R mapping methods and XPath queries containing descendant/child axes with simple conditions being considered.

1. XML schema and workloads

We formalize XML schemas as XS-trees, which is a modification of the schema tree proposed in (Ramanath *et al.*, 2003).

Definition 1 (XS-tree) An XML schema X_S is a tree (V, e), where $V=\{v_i\}$ denotes an array of nodes in the tree and $v_1 \in V$ is the root element. Edge function e: $\mathbb{N}\times\mathbb{N}\rightarrow\{0, 1, ?, *, +\}$ determines directed edges and their multiplicity, where $e_{ij}=1$ $(1\leq i, j\leq|X_S.V|)$ indicates an directed edge from node v_i to v_j with multiplicity 1.

Definition 2 (XS-tree nodes) Each node v_i $(1\leq i\leq|X_S.V|)$ is a quad $(id, type, name, tag)$, where $v_i.id=i$, $v_i.name$ denotes the node name and $v_i.tag$ is a label for transformation. $v_i.type$ takes values from {"S", "C", "|", ","}, which individually represents "simple type" node, "complex type" node, union connector node and sequence connector node, respectively.

Initially "complex type" nodes are tagged by their names, and "simple type" nodes with an inbound edge whose multiplicity takes values from {*, +} are also tagged by their names. Union/sequence connector nodes with a {*, +} inbound edge are tagged by their ids prefixed with "R". All other nodes are tagged by nil.

Fig.2 shows an example of the XS-tree, constructed based on a modified snippet (Fig.2a) of the DBLP DTD (DBLP Bibliography, http://www.informatik.uni-trier.de/*ley/db/). Construction of the initial XS-tree for an XML schema is straightforward; however, there could be many equivalent XS-trees for the same XML schema. Details on it will be analyzed in the next subsections.

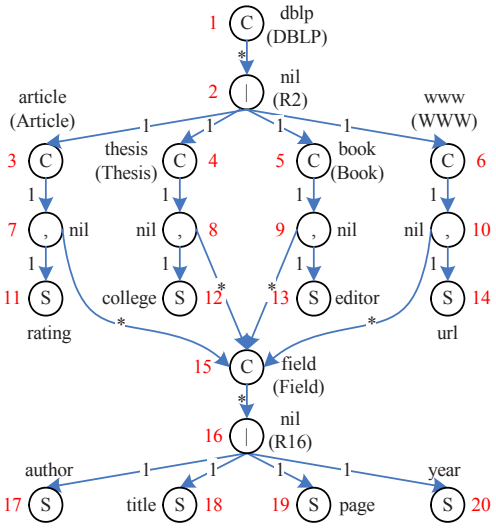
Definition 3 (XML query) An XML query is a sequence of items $\{p_i\}$, where each item p_i $(1\leq i\leq n)$ is a quad (s, T, cs, cp). $p_i.s$ takes values from {"/", "//"}, which individually means parent-child traversal and ancestor-descendant traversal, respectively. $p_i.T$ is an array of elements to traverse, and $p_i.cs$ is the sub-element name (or attribute name) to apply the condition predicate $p_i.cp$. If no condition is applied, $p_i.cs$ and $p_i.cp$ are neglected.

Following the definition, XML query "/dblp/article/field/(author|title)" is represented as Q_1={(/, [dblp]), (/, [article]), (/, [field]), (/, [author, title])}, which returns the "author" and "title" elements of all "field" elements belonging to "article". Query "//thesis (college='ZJU')/field/author" is represented as Q_2={(//, [thesis], college, ='ZJU'), (/, [field]), (/, [author])} which returns the "author" element in "thesis" subtree where the "college" element is "ZJU". Query "/dblp//field (year>2006)" is represented as Q_3={(/, [dblp]), (//, [field], year, >2006)} which returns all "field" nodes satisfying year>2006.

Definition 4 (XML workload) XML workload is a set $W_X=\{W_i\}$ $(1\leq i\leq|W_X|)$, and $W_i=\{q, f\}$, where $W_i.q\in Q_X$ is an XML query and $W_i.f\in(0, 1]$ is the normalized frequency of each query $(\sum_i W_i.f=1)$. We denote the cost of the XML workload as $C_X(W_X)$.

```
<!DOCTYPE dblp [
<!ELEMENT dblp (article|thesis|book|www)*>
<!ELEMENT article (rating, field*)>
<!ELEMENT thesis (college, field*)>
<!ELEMENT book (editor, field*)>
<!ELEMENT www (url, field*)>
<!ELEMENT field (author|title|pages|year)>
<!ELEMENT rating (#PCDATA)>
<!ELEMENT college (#PCDATA)>
<!ELEMENT editor (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT pages (#PCDATA)>
<!ELEMENT year (#PCDATA)>
]>
```

(a)



(b)

Fig.2 (Partial) DBLP DTD (a) and the corresponding XS-tree (b)

2. Relational schema and workloads

Definition 5 (Logical schema and data scale) Logical schema is a set of relations $L=\{L_i\}$ ($1\le i\le|L|$), and $L_i=\{name, A, num\}$. $L_i.name$ is the relation name, $L_i.A$ is a set of attributes with information about the attributes' names, types and average length, and $L_i.num$ is the approximate number of items in L_i relation, which indicates the data scale.

Definition 6 (Physical schema) Physical schema is a set of physical feature sets $P=\{P_i\}$, where $|P|=|L|$. P_i ($1\le i\le|L|$) denotes the physical schema for relation L_i, and L_i has a set of physical features $\{p_{ij}\}$ ($1\le i\le|L|$, $1\le j\le|P_i|$). Here $p_{ij}=(type, param, A)$, where $p_{ij}.type$ takes values from {I, M, H, V, C, R}, which respectively mean indexes, materialized views, horizontal partitions, vertical partitions, compression, and replication; $p_{ij}.param$ denotes parameters of the physical feature; $p_{ij}.A$ is a set of related attributes in the physical feature.

With the logical schema L and the physical schema P, we can estimate the required storage space. Since the estimation is straightforward, we omit details and denote the estimation function as $C_S(L, P)$.

Definition 7 (SQL query) SQL query is an SQL statement $q_S\in Q_S$, and we denote the cost of q_S under a certain logical schema L and a certain physical schema P as $C_Q(q_S, L, P)\in\mathbb{R}^+$, which is an estimated cost generated by the optimizer in RDBMS. Modern RDBMS with a "what-if" optimizer (Chaudhuri and Narasayya, 1998) can estimate the cost without physically creating the schemas.

Definition 8 (SQL workload) SQL workload is a set $W_S=\{S_i\}$ ($1\le i\le|W_S|$), and $S_i=\{q, f\}$, where $S_i.q\in Q_S$, and $S_i.f\in(0, 1]$ is the normalized frequency of each statement ($\sum_i S_i.f=1$). Cost of a whole workload is calculated as

$$C_W(W_S)=\sum_{i=1}^{|W_S|}\left(C_Q(S_i.q, L, P)\times S_i.f\right). \quad (1)$$

3. The AX2RM problem

With the above definitions, mapping functions in the AX2RM system are:

(1) Logical database design (with data scale estimation): a mapping function Ψ_L to derive a relational schema L from an XML schema X_S. We denote the function as $\Psi_L: X_S\to L$.

(2) Workload translation: a mapping function Ψ_W to derive an SQL workload W_S from an XML workload W_X, based on an XML schema X_S and a derived relational schema L. We denote the function as Ψ_W: $W_X \xrightarrow{X_S, L} W_S$.

(3) Physical database design: a mapping function Ψ_P to derive a physical schema P from a relational schema L, based on the workload information, W_S. We denote the function as Ψ_P: $L \xrightarrow{W_S} P$.

The criterion to judge a mapping (good or bad) is the workload cost, which is equivalent to the execution cost of the translated SQL workload on the designed relational and physical schemas. We have

$$C_X(W_X)=C_W(W_S)=C_W(\Psi_W(W_S)). \quad (2)$$

Finally, the AX2RM problem is defined as follows:

Definition 9 (The AX2RM problem) Given an XML schema X_S, an XML workload W_X and the space limitation L_S, find the mapping function triple (Ψ_L, Ψ_W, Ψ_P) to minimize the cost of W_X, i.e., to find

$$\arg\min_{\Psi_L,\Psi_W,\Psi_P} C_X(W_X) \quad \text{s.t. } C_S(L, P)\le L_S. \quad (3)$$

SEARCH SPACE OF APPROPRIATE X2R MAPPING

Before diving into details of optimization algorithms, we study the search space for all procedures in the AX2RM system.

Logical database design

1. XML schema to logical schema mapping

In the AX2RM system, an XML schema is an XS-tree, thereupon, the following XS-tree to relational mapping algorithm is presented as a basic X2R schema mapping method:

Algorithm 1 XS-tree mapping

Given an XS-tree X_S, the logical schema L is derived as follows:

```
Set L=∅;
Foreach v in X_S.V
  If (v.tag!=nil)
    rel_cur=L.find_or_create_relation(v.tag);
    v_anc=X_S.find_closest_tagged_ancestor(v);
    If (v_anc==nil)
      rel_cur.A∪={v.tag+"_id"};
    Else
```

$rel_{cur}.A \cup = \{v.tag + \text{“\_id”}, v_{anc}.tag + \text{“\_id”}\}$;
If ($v.type$==“s”)
$rel_{cur}.A \cup = v.name$;
End If
If (*not_all_descendants_tagged*(v))
$rel_{cur}.A \cup = \{v_d.name | v_d \in descendants(v)$ and $v_d.type$==“s”$\}$;
End If
End If
End Foreach

Algorithm 1 creates a relation for all tagged nodes and a primary key column—the “id” column—for each relation. It also creates a foreign key column named after the tag of the closest ancestor node plus a postfix “_id”. For a simple type node, a column named by the node name is appended; otherwise, columns named by the names of the untagged simple-typed descendants are added. With Algorithm 1, the XS-tree described in Fig.2b is mapped into the following relations: DBLP (dblp_id), R2 (r2_id, dblp_id), Article (article_id, rating, r2_id), Thesis (thesis_id, college, r2_id), Book (book_id, editor, r2_id), WWW (www_id, url, r2_id), Field (field_id, article_id, thesis_id, book_id, www_id), R16 (r16_id, author, title, page, year, field_id). We denote the mapping algorithm as $L=M_L(X_S)$.

2. Equivalent XS-tree transformation

As the XS-tree to relational schema transformation is fixed in Algorithm 1, flexibility of the logical database design goes to equivalent transformations of the XS-tree. Fig.3 shows some equivalent transformations on the XS-tree (Bohannon *et al.*, 2002; Chaudhuri *et al.*, 2005).

(1) Outlining/Inlining (Fig.3a). Outlining takes a node $v \in X_S.V$ which is not tagged, and simply tags it, while inlining simply clears tags on the tagged nodes. Note that, as long as a node has one inbound edge with multiplicity in {+, *}, it cannot be inlined unless all its direct descendants are tagged. Proper outlining and inlining will enhance the locality of frequently accessed correlated elements. In Fig.3a, the outlining form results in two relations: Article (article_id, parent_id), Rating (rating_id, rating, article_id), while the inlining results in only one relation: Article (article_id, rating, parent_id).

(2) Type split/Type merge (Fig.3b). Type split will split a shared node into multiple distinct nodes with distinct tags, while type merge does the reverse. Proper type split and merge will enhance the locality of frequently accessed elements with a shared type.

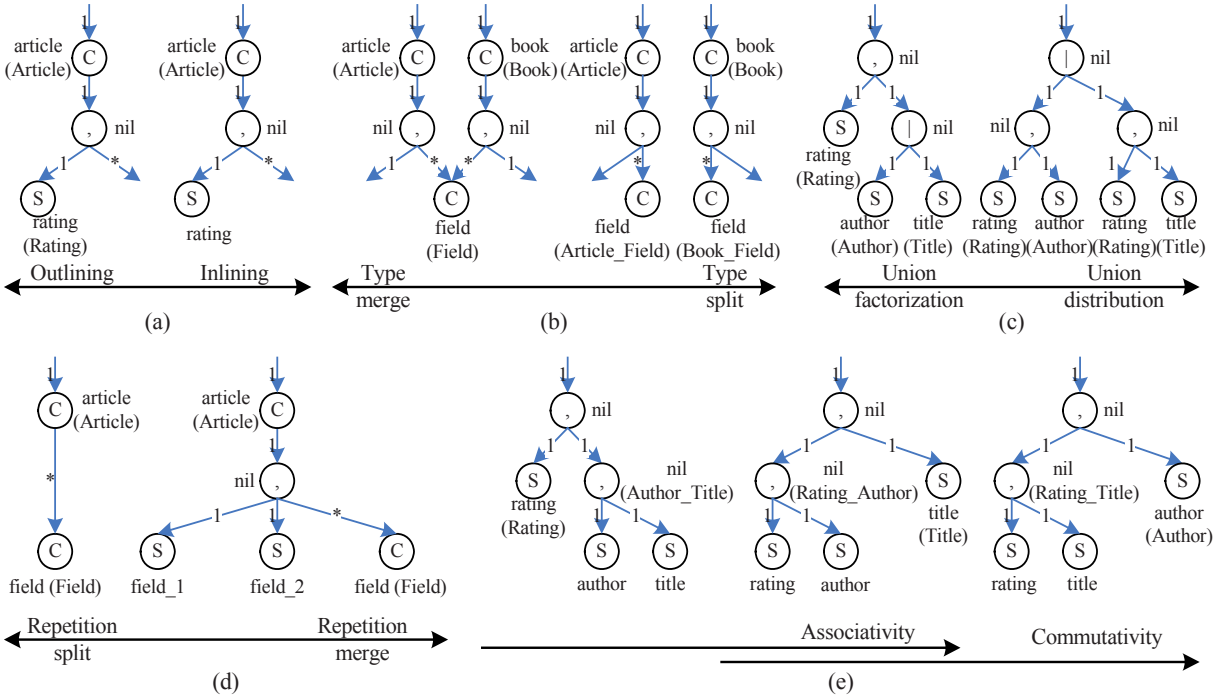


Fig.3 Equivalent XS-tree transformations. (a) Outlining/Inlining; (b) Type split/Type merge; (c) Union distribution/Union factorization; (d) Repetition split/Repetition merge; (e) Associativity and Commutativity

(3) Union distribution/Union factorization (Fig.3c). Union distribution converts a schema fragment from $(a, (b|c))$ to $(a, b)|(a, c)$, while union factorization does the reverse. In the conversion, c can be empty, which means b is optional. In Fig.3c, (Rating, (Author|Title)) is converted to (Rating, Author)|(Rating, Title).

(4) Repetition split/Repetition merge (Fig.3d). Repetition split splits a node with multiplicity * to k nodes with multiplicity 1 and one node with multiplicity *, followed by inlining the k newly created nodes. Repetition merge does the reverse. Proper repetition split/merge will remarkably reduce the cost of relational join. Fig.3d shows an example for repetition split/merge.

(5) Associativity and Commutativity (Fig.3e). Associativity groups several types into one relational table, while commutativity changes the order of the types. Both of them can be combined with other transformations to generate new mappings. Transformation in Fig.3e exploits associativity and commutativity of the "," nodes.

Workload translation

Based on a derived logical schema, we translate XML workload to equivalent SQL workload. Since frequencies are fixed during the translation, query translation, i.e., the XML query to SQL query mapping, becomes the key challenge.

1. Basic XML query to SQL query mapping

To meet the challenge, we first consider a simpler form of XML queries, the NXQ (normalized XML query). An XML query q_X is an NXQ, iff $\forall p_i \in q_X$, $p_i.s$="/" and for $i \in [1, |q_X|-1]$, $|p_i.T|=1$, i.e., q_X is an NXQ if and only if no descendant traversal is used and no "or" predicate exists except in the last projection.

Algorithm 2 Normalized XML query translation

Given an XS-tree X_S and an NXQ q_X, the relational algebra *ra* for the equivalent SQL query is derived as follows:

```
Function TransNXQ(q_X, X_S):
  ra=v_1.tag; v_last=X_S.v_1; proj=nil;
  For i=1 to |q_X|        // process internal paths
    p=q_X.p_i; v_cur=X_S.get_nodes_by_names(p.T);
    v_seq=X_S.get_tagged_nodes_on_path(v_last, v_cur);
    ra=(*(v.tag∈v_seq))*ra;
    v_last=v_seq.last();
    If (p.cs!=nil)      // applying constraints
      v_child=X_S.get_child_node_by_name(v_cur, p.cs);
      If (v_child.tag!=nil)
        ra=σ_(v_child.tag.(v_child.name) satisfy p.cp)(v_child.tag*ra);
      Else
        ra=σ_(v_last.tag.(v_child.name) satisfy p.cp)(ra);
      End If
    End If
  End For
  Foreach v in v_cur              // process the last item
    If (v.tag!=nil)
      ra=v.tag*ra; proj+=v.tag.(*);
    Else If (v.type=="s")
      proj+=v_last.tag.(v.name);
    Else
      proj+=v_last.tag.(X_S.names_of_untagged_descendants(v));
    End If
  End Foreach
  Return merge_duplicate_join(π_(proj)(ra));
End Function
```

Exploiting *TransNXQ*() function defined in Algorithm 2, we translate the general XML queries defined in Definition 3 as follows:

Algorithm 3 Basic XML query translation

Given an XS-tree X_S and an XML query q_X, the relational algebra for the equivalent SQL query $base(q_X)$ is derived as follows:

```
Q={q_X}; ra=nil;
Do      // transfer q_X to an equivalent set of NXQs
  cnt_old=|Q|;
  Foreach q in Q
    i=q.find_an_or_item_id();
    If (p_i is not nil)        // eliminate the internal "OR"
      Q'=q.clone_by_split_item(p_i);
    End If
    i=q.find_a_descendant_traversal();
    If (p_i is not nil)    // eliminate "//" by splitting it into "/"
      P=find_all_child_traversal_on_path(p_i−1, p_i);
      Q''=q.clone_by_split_traversal(p_i, P);
    End If
    Q=Q∪Q'∪Q'';
  End Foreach
While (|Q|>cnt_old)
Foreach q in Q
  ra=ra∪TransNXQ(q);
End Foreach
```

Fig.4 shows three examples translating the XML query to the SQL query based on the XS-tree in Fig.2b. Note that $base(q)$ is not an optimal but a correct mapping for the input XML query (for instance, the optimal SQL query for Q_3 is "**select** *f*.* **from** R16 *r*

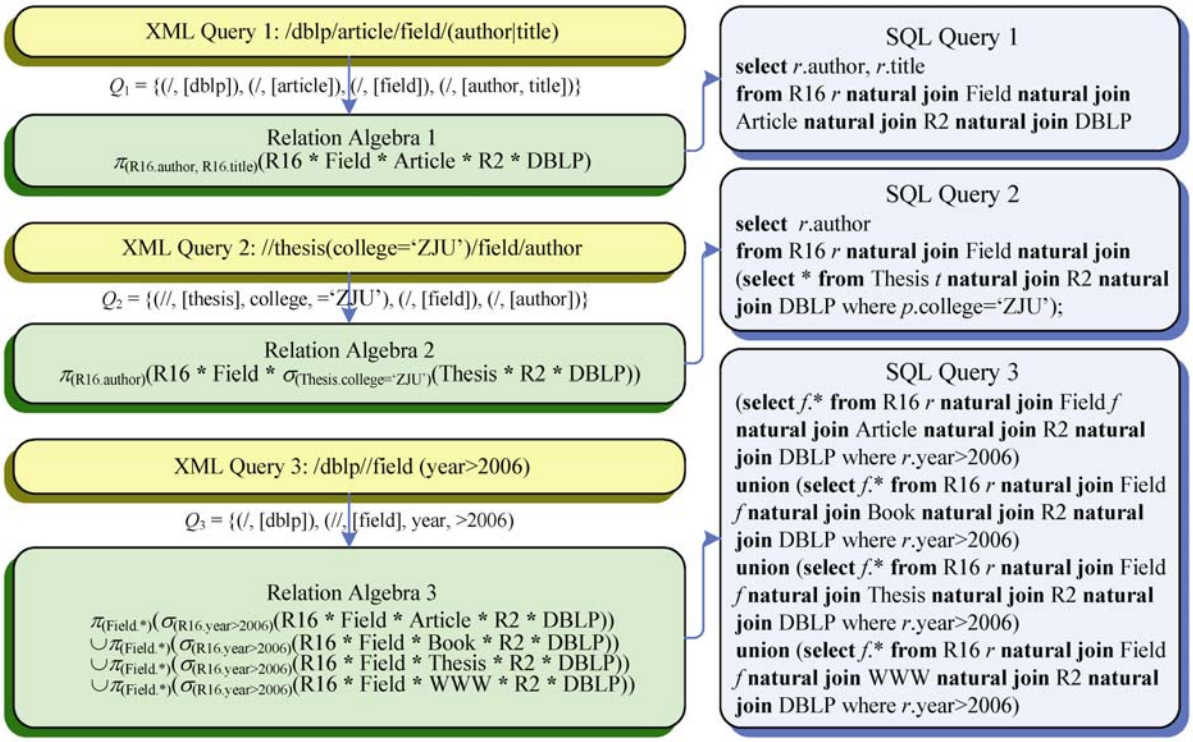


Fig.4 Equivalent XS-tree transformations: three XML queries, their NXQs, the equivalent relation algebras, and the equivalent SQL queries

natural join Field *f* where *r*.year>2006"). Algorithm 3 transforms the input XML query to a set of NXQs which eliminate internal ORs and descendant traversals, and it constructs *base*(*q*) as a union of translation results of each NXQ. For each NXQ, items in the query are sequentially taken into account. For each item, Algorithm 2 finds the only path from the last processed item to the current one (uniqueness is due to the normalization of the query, and only parent-child traversal exists). Natural joins are generated on the path, and constraints are applied afterwards. When the last item in the query sequence is reached, a projection is applied on the generated relational algebras.

2. Equivalent X2R query translations

In order to find optimal translations, we need to find all equivalent translations which generate the same result as *base*(*q*). Self-evidently, algebraic-equivalent transformations of *base*(*q*) are equivalent queries; however, valuable information for finding optimal translations such as structure information and logical constraints is lost during the execution of Algorithms 2 and 3, which results in sub-optimal mappings (Krishnamurthy *et al.*, 2004). We leverage the optimizer in RDBMS to perform algebraic equivalent transformations when optimizing SQL queries, and focus on logical equivalent transformations exploiting the structure of the XS-tree. There are two categories of logical equivalent transformations: intra-NXQ transformations which are applied within each NXQ (in Algorithm 2), and inter-NXQ transformations which are executed among NXQs in a single query (in Algorithm 3). The following transformations are identified:

(1) Path elimination (Intra-NXQ). For $p_k \in q_X$, if there is only one path from the root node $v[1]$ to $v[\text{node}(p_k.T[0])]$ and each node in p_i ($1 \le i \le k$) is on the path with all $p_i.cs$ being nil, we can merge the first k unnecessary natural joins. With this approach, the algebra of Q_1 will be simplified to "$\pi_{(R16.author, R16.title)}$(R16*Field*Article)" and the result of Q_2 will be simplified to "$\pi_{(R16.author)}$(R16*Field*$\sigma_{(Thesis.college='ZJU')}$(Thesis*R2*DBLP))".

(2) Natural join processing (Intra-NXQ). Since each relation has a foreign key pointing to the parent relation id, we can attach all natural joins with constraints on "foreign key column is not null". Moreover, with the constraint, the last natural join in a query can

be directly eliminated. Thus, the result of Q_1 can be improved to "$\pi_{(R16.author,\ R16.title)}(\sigma_{(Field.article\_id\ is\ not\ null)}(R16*Field))$".

(3) Same prefix merge (Inter-NXQ). For NXQs with the same prefix sequence $q_p=\{p_1, p_2, \ldots, p_i\}$, we can derive a prefix sequence algebra on q_p and re-execute Algorithm 2 on each NXQ starting from p_i. Result algebra is the union of all remaining NXQ algebras applying on the common prefix algebra. The prefix sequence q_p of Q_3 is {(/, [dblp])}. Thus, we can transform the result algebra to "$\pi_{(Field.*)}((\pi_{(Field.*,Article.r2\_id)}(\sigma_{(R16.year>2006)}(R16*Field*Article))\cup\pi_{(Field.*,Book.r2\_id)}(\sigma_{(R16.year>2006)}(R16*Field*Book))\cup\pi_{(Field.*,Thesis.r2\_id)}(\sigma_{(R16.year>2006)}(R16*Field*Thesis))\cup\pi_{(Field.*,WWW.r2\_id)}(\sigma_{(R16.year>2006)}(R16*Field*WWW)))*(R2*DBLP))$".

(4) Same postfix merge (Inter-NXQ). For NXQs with the same postfix sequence $q_p'=\{p_j, p_{j+1}, \ldots, p_n\}$, we can execute Algorithm 2 on each NXQ ended at p_j and re-execute Algorithm 2 for the postfix algebra modifier. Result algebra is the common postfix algebra modifier applying on the union of all remainder NXQ algebras. The postfix sequence q_p' of Q_3 is {(/, [field]), year, >2006}; thus, the result algebra is transformed to "$\pi_{(Field.*)}(\sigma_{(R16.year>2006)}(R16*(\pi_{(Field.*)}(Field*Article*R2*DBLP)\cup\pi_{(Field.*)}(Field*Book*R2*DBLP)\cup\pi_{(Field.*)}(Field*Thesis*R2*DBLP)\cup\pi_{(Field.*)}(Field*WWW*R2*DBLP))))$".

(5) Full postfix merge (Inter-NXQ). A special case of the same postfix merge is the full postfix merge, which merges all prefixes when the paths and conditions from the root to the node in NXQs cover all paths in the XS-tree. With the merge, Q_3 will result in "$\pi_{(Field.*)}(\sigma_{(R16.year>2006)}(R16*Field))$".

Optimally and iteratively applying these transformations, SQL queries in Fig.4 can be improved to:

SQL Query 1: **select** *r*.author, *r*.title **from** R16 *r* **natural join** Field *f* where *f*.article_id **is not null**;

SQL Query 2: **select** *r*.author **from** R16 *r* **natural join** Field *f* **natural join** (**select** * **from** Thesis *t* where *t*.college='ZJU');

SQL Query 3: **select** Field.* **from** R16 *r*, Field *f* where *r*.year>2006.

Physical database design

For a specific logical schema, there are large numbers of physical designs, i.e., solution space of correct physical designs is extraordinarily huge. First we need to reduce the solution space from "correct designs" to "useful designs", which are constituted by useful physical features that can decrease the cost for at least one SQL statement in the workload. This procedure is described as the "feature generation" in automatic physical database design literature (Rozen and Shasha, 1991). Details on how to generate useful physical features are intensively discussed in related publications (Rozen and Shasha, 1991; Agrawal *et al.*, 2004a; 2004b; Zilio *et al.*, 2004; Hu *et al.*, 2006).

APPROACHES FOR SEARCHING OPTIMAL X2R MAPPING

We need to explore the solution space studied in the previous section for optimal X2R mappings. The approaches for the optimal search are investigated one by one.

Naïve approach with brute force (NV-BT)

With NV-BT, the solution spaces are searched one by one, independently and exhaustedly. NV-BT first searches for the best logical design with basic workload transformation (using *base*(*q*)) and default (no additional) physical database design. Under a specific best logical design, NV-BT next bruteforcely searches for the best workload transformation with no additional physical database design. Finally, the best physical design is exhaustedly searched under certain logical schemas and workloads.

NV-BT views all procedures in the AX2RM as separated procedures, and traditional X2R mapping researches fall into this category. As mentioned, these approaches lead to sub-optimal mappings. In addition, since all searches are executed in an exhausted manner, the whole procedure is time-consuming.

Naïve approach with greedy manner (NV-GR)

The same as in NV-BT, we view the procedures as separate ones in NV-GR. However, we exploit greedy algorithms when searching for the best solution in each procedure. Detailed algorithm is omitted here due to its simplicity. The approach also leads to sub-optimal mappings (possibly worse than NV-BT); however, execution time will be dramatically reduced due to the greedy manner.

Integrated approach with brute force (IT-BT)

On account of executing each search procedure independently, naïve approaches lead to sub-optimal mappings. Thus, we need an integrated approach to solve the problem. IT-BT traverses all possible triples (logical schema, physical schema, and workload translation) to find the best mapping. IT-BT is an integrated approach which exhaustedly explores the combined search spaces for an optimal mapping. However, the approach is not practicable since the combined search space is too huge.

Integrated approach with greedy manner (IT-GR)

IT-GR is an improvement on IT-BT, which exploits greedy heuristic in optimization. In this approach we use the greedy algorithm instead of the exhausting algorithm in IT-BT in each procedure. Despite of the possibility of generating sub-optimal mappings, the greedy approach will reduce the time complexity markedly.

Nevertheless, the approach is still time-consuming, since the workload cost estimation exploiting RDBMS's "what-if" optimizer is relatively costly, and it will be executed repetitiously for every triple (logical schema, physical schema and workload translation).

Integrated approach with greedy pruning (IT-GP)

We improve IT-GR to a pruning based IT-GP to eliminate the mappings that are unlikely to be optimal as early as possible. The following three categories of heuristics are identified to generate possible good mappings: the logical scehma generation heuristic, the workload translation heuristic, and the physical schema geneation heuristic. Mappings other than these three categories are considered unlikely to be optimal and are thus simply pruned.

1. Logical schema generation heuristic (H_L) to generate possible good transformations on the XS-tree.

(1) Inlining maximization. Try inlining when the space limit is not reached.

(2) Repetition split generation. Do repetition split as calculated according to the actual XML data by analyzing multiplicity between two elements.

(3) Union distribution generation. If an XML query accesses less than half partitions in the union, do a union distribution; if an XML query accesses more than half partitions, try a union factorization.

(4) Union distribution merge. Generate a factorization on multiple union partitions produced by the previous steps.

(5) Type split generation. Almost the same as union distribution generation.

(6) Type split merge. Almost the same as union distribution merge.

2. Workload translation heuristic (H_W) to generate possible good transformations on $base(q)$.

(1) Suppress as many natural joins as possible.

(2) Merge NXQs to eliminate additional queries.

3. Physical schema generation heuristic (H_P) to generate possibly useful physical designs for each SQL query in the generated workload on the generated logical schema.

(1) Condition index generation. Generate multiple indexes when a "where" clause is specified in an SQL query for improving the query performance.

(2) Natural join index generation. Generate indexes on keys for the natural join to improve join performance.

(3) Access column index generation. Generate an index on all involved columns in a query for using-index optimizaiton.

(4) Access column vertical partition generation. Generate a vertical partition for all involved columns in a query to improve locality.

(5) Condition horizontal partition generation. Generate a horizontal partition on a condition specified in a query.

(6) Rarely-used column compression generation. Compress all columns rarely involved in queries.

(7) Large-size column compression generation. Compress all columns with relatively large size in total.

With these rules, we only concern a relatively small number of solutions, which dramatically reduce the time needed to execute the greedy algorithm. Due to space limitation, we omit the detailed description of above rules and the detailed algorithm of IT-GP.

EXPERIMENTAL EVALUATION

This section experimentally evaluates the five approaches proposed in the previous section and two approaches previously published by other researchers.

We find that IT-GP drastically reduces the running time with preferable results in optimality.

Experimental setup

Environment: We use a self-developed "what-if" optimizer on a revised version of PostgreSQL (supporting vertical partition, horizontal partition, and column based compression) on a Windows XP laptop as the cost estimation module.

Dataset: A real data set from DBLP (http://dblp.uni-trier.de/xml/dblp.xml.gz, Oct. 5th, 2007) is used for demonstration. The size of the original data set—the dblp.xml—is 404 MB. Four workloads containing 10 queries with the same frequency are considered: in W1(LS, LP), selectivity varies in 0.1%~1%, while the projected attribute number varies in 1~3; in W2(LS, HP), the selectivity is similar, but the projection number is from 10 to 20; W3(HS, LP)/W4(HS, HP) is similar to W1/W2 in projection, however, the selectivity is in 50%~100%.

Algorithms: All the five approaches mentioned in this paper are evaluated to generate relational mappings. The IT-BT does not stop after very long running time (1 d) for all workloads, so we simply ignore it in consequent comparisons. Other approaches are compared with the hybrid inlining (Shanmugasundaram *et al.*, 1999) based approach (Hybrid for short) and the integrated mapping approach proposed by Chaudhuri *et al.*(2005) (Chaud for short). Hybrid inlines all subelements with the in-degree equal to 1, and uses default workload translation and fully functional physical database designs. The Chaud approach does not consider equivalent query translations when recommending mappings.

Results: We compare both algorithm execution time and workload execution time of these six approaches. For clarity, these two indicators are normalized to those of Hybrid.

Results and analysis

Fig.5a compares the algorithm execution time of each approach. We find that all the approaches presented in this paper and the Chaud approach run longer than the Hybrid approach. Execution time of NV-BT and IT-GR is much longer than that of the others, since their search spaces are extremely huge. IT-GP prunes the search space of IT-GR and reduces the execution time by 50%~80%; execution of NV-GR requires the shortest time. Normally, IT-GP costs almost the same (a little longer) execution time as Chaud, which means that the query translation procedure costs relatively short execution time. Comparing different workloads, algorithm executions on workloads with bigger projection number take longer time, since many more mappings need to be concerned in each procedure. However, differences of selectivity are less significant.

Fig.5b is the comparison of the workload execution time on mappings suggested by each approach. The results show that NV-BT and NV-GR lead to much less optimal mappings, even compared with Hybrid and Chaud. However, IT-GR and IT-GP generate relatively good mappings compared with other approaches, by orders of magnitude.

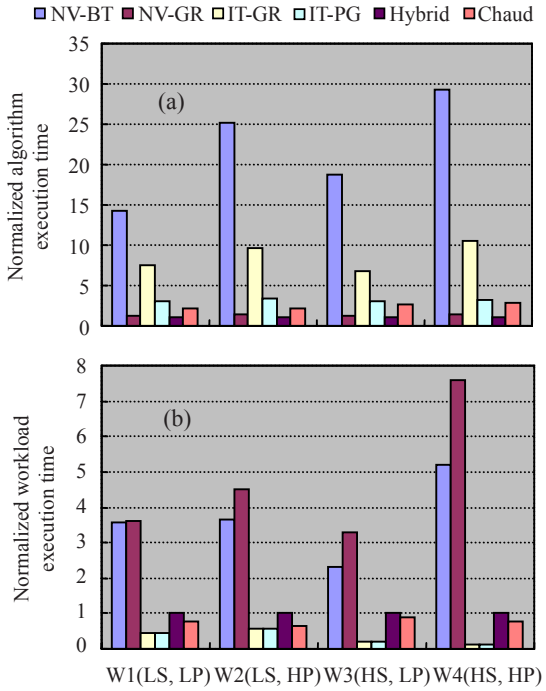


Fig.5 Comparison of algorithm execution time (a) and workload execution time (b) for the six approaches NV-BT, NV-GR, IT-GR, IT-PG, Hybrid, and Chaud

CONCLUSION

In this paper, we deem that to find an optimal XML to relational mapping, procedures including logical database design, physical database design, and workload translation should be considered integrat-

edly. Main contributions of this paper are:

(1) To present concepts and related issues of fully integrated X2R mapping design, and to introduce the AX2RM system to address these issues;

(2) To propose a pruning based integrated approach, the IT-GP, and to evaluate its validity and performance compared with other common approaches.

For further study, we need to expand the scope of supported XML queries and consider more logical transformation and workload translation schemes.

References

Agrawal, S., Chaudhuri, S., Kollar, L., Marathe, A.P., Narasayya, V.R., Syamala, M., 2004a. Database Tuning Advisor for Microsoft SQL Server 2005. Proc. 30th Int. Conf. on Very Large Databases, p.1110-1121.

Agrawal, S., Narasayya, V.R., Yang, B., 2004b. Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. Proc. Int. Conf. on Management of Data, p.359-370. [doi:10.1145/1007568.1007609]

Atay, M., 2006. XML2REL: An Efficient System for Storing and Querying XML Documents Using Relational Databases. Ph.D Thesis, Wayne State University.

Bohannon, P., Freire, J., Roy, P., Simeon, J., 2002. From XML Schema to Relations: A Cost-Based Approach to XML Storage. Proc. 18th Int. Conf. on Data Engineering. IEEE Computer Society, p.64-75. [doi:10.1109/ICDE.2002.994698]

Chaudhuri, S., Narasayya, V., 1998. AutoAdmin what-if index analysis utility. *ACM SIGMOD Record*, **27**(2):367-378. [doi:10.1145/276304.276337]

Chaudhuri, S., Chen, Z.Y., Shim, K., Wu, Y.Q., 2005. Storing XML (with XSD) in SQL databases: interplay of logical and physical designs. *IEEE Trans. on Knowl. Data Eng.*, **17**(12):1595-1609. [doi:10.1109/TKDE.2005.204]

Chen, Y.W., 2004. XQuery Query Processing in Relational Systems. Ph.D Thesis, University of Waterloo.

Deutsch, A., Tannen, V., 2003. MARS: A System for Publishing XML from Mixed and Redundant Storage. Proc. 29th Int. Conf. on Very Large Data Bases, p.201-212.

Florescu, D., Kossmann, D., 1999. Storing and querying XML data using an RDBMS. *Bull. Tech. Committ. on Data Eng.*, **22**(3):27-34.

Hu, T.L., Chen, G., Li, X.Y., Dong, J.X., 2006. Automatic relational database compression scheme design based on swarm evolution. *J. Zhejiang Univ. Sci. A*, **7**(10):1642-1651. [doi:10.1631/jzus.2006.A1642]

Krishnamurthy, R., Kaushik, R., Naughton, J.F., 2003. XML-to-SQL query translation literature: the state of the art and open problems. *LNCS*, **2824**:1-18.

Krishnamurthy, R., Kaushik, R., Naughton, J.F., 2004. Efficient XML-to-SQL Query Translation: Where to Add the Intelligence? Proc. 30th Int. Conf. on Very Large Databases, p.144-155.

Lee, D.W., Chu, W.W., 2000. Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema. Proc. Int. Conf. on Conceptual Modeling, p.323-338.

Mani, M., Wang, S., Dougherty, D.J., Rundensteiner, E.A., 2006. Join minimization in XML-to-SQL translation: an algebraic approach. *ACM SIGMOD Record*, **35**(1):20-25. [doi:10.1145/1121995.1121999]

Ramanath, M., Freire, J., Haritsa, J.R., Roy, P., 2003. Searching for efficient XML-to-relational mappings. *LNCS*, **2824**:19-36.

Rozen, S., Shasha, D., 1991. A Framework for Automating Physical Database Design. Proc. 17th Int. Conf. on Very Large Databases, p.401-411.

Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D.J., Naughton, J.F., 1999. Relational Databases for Querying XML Documents: Limitations and Opportunities. Proc. 25th Int. Conf. on Very Large Databases, p.302-314.

Zilio, D.C., Zuzarte, C., Lightstone, S., Ma, W.B., Lohman, G.M., Cochrane, R., Pirahesh, H., Colby, L.S., Gryz, J., Alton, E., *et al*., 2004. Recommending Materialized Views and Indexes with IBM DB2 Design Advisor. Proc. 1st Int. Conf. on Autonomic Computing, p.180-188. [doi:10.1109/ICAC.2004.47]