EFFICIENTLY PUBLISHING DATA AS XML DOCUMENTS
Introduction

XML – Into the Basics

SQL-Based Language Specification

Implementation Alternatives
  - Early Tagging, Early Structuring
  - Late Tagging, Late Structuring
  - Late Tagging, Early Structuring

Performance Comparison of Alternatives for Publishing XML

Conclusion
INTRODUCTION

- XML is emerging as a standard for exchanging business data on the WWW.

- Most operational business data is stored in relational database systems.

- Unlikely to change: reliability, scalability, tools and performance are associated with relational database systems.
How to publish relational data in the form of XML documents?

Two main requirements:
- A **language** to specify the conversion from relational data to XML documents.
- An **implementation** to efficiently carry out the conversion.
Extensible Markup Language (XML) is a hierarchical format for information exchange in the World Wide Web.

A document consists of:
- Nested element structures starting with a root element;
- Each element has a tag associated with it;
- An element can have attributes and values or sub-attributes;
- Elements can be ordered.
Root Element: <customer>

Id attribute is a special kind that uniquely identifies an element in a XML document.

The type of the attribute acct is IDREF and logically point to an element having the same value as its ID.
Key requirement to convert relational data to XML documents.

Suggested approach:
- Harness and extend the SQL.
- Nested SQL statements are used to specify nesting;
- SQL functions are used to specify XML element construction.
SQL-BASED LANGUAGE SPECIFICATION

Customer (id integer, name varchar(20))

Account (id varchar(20), custId integer, acctnum integer)

PurchOrder (id integer, custid integer, acctId varchar(20), date varchar(10))

Item (id integer, poid integer, desc varchar(10))

Payment (id integer, poid integer, desc varchar(10))
The overall query consists of several correlated queries.

This query produces both SQL and XML data.
- Each result tuple contains a customer’s name together with the XML representation of the customer.
Conceptually is like a scalar function returning XML.

Input: customer’s name, account information (as XML form) and purchase order information (as xml form).

Output: customer XML element.

ACCT, PORDER, ITEM and PAYMENT had similar constructors.
XMLAGG

- Is an aggregating function.
- Concatenates all the XML fragments (ITEM, PAYMENT...) produced by XML constructors.
- Needs to work on ordered inputs.
Main difference between relational tables and XML documents:

- XML documents have tags.
- XML documents have nested structure.

In converting from relational tables to XML documents, tags and structure have to be added somewhere along the way.
Early Tagging and Late Structuring is not a viable alternative because physically tagging an XML document without structure makes no sense.
1. The Stored Procedure Approach

- Simple and iterative technique.
- **Outside Engine:**
  - queries for each nested structure within the desired XML document.
- **Early Structuring:**
  - queries that are issued mimic the structure of the result.
- **Early Tagging:**
  - tagging is done as soon as each nested structure becomes available.
1. The Stored Procedure Approach

- Commonly used today.

- Problems:
  - One or more queries issued per tuple causing overhead.
  - Serious performance inefficiencies.
  - This approach dictates a particular join order.
2. The Correlated CLOB Approach

- **Inside Engine:**
  - Way to eliminate the overhead of issuing many queries.
  - Process one large query with sub-queries rather than many top level queries.
2. The Correlated CLOB Approach

- **Challenge:**
  - Have the relational engine tag and build nested structures

- **Solution:**
  - Add engine support for the XML constructors and XMLAGG function.
  - Represent XML document fragments (arbitrary size) as large objects – Character Large Objects (CLOBs) – inside the relational engine.
2. The Correlated CLOB Approach

**Problems:**

- CLOBs can lead to performance problems because they are large objects and they are stored separately from the tuples they belong to.

- In parallel environments fetching CLOBs can lead to significant performance degradation.

- Each invocation of XML constructor copies its input to a new CLOB. This repeated creation and copy can be costly.
3. The *De-Correlated CLOB* Approach

- Perform query de-correlation inside the relational engine to give to the relational optimizer more flexibility.

- Inside Engine:
  - Each path from the root level table to a leaf level table is computing by joining the tables along the way.
3. The *De-Correlated CLOB* Approach

- **Outer joins:**
  - Preserve parent information even if it has no children.

- The set of leaf level XML elements is built up by grouping on the id columns on the path from the root level to the leaf level.
3. The *De-Correlated CLOB Approach*

- **Advantages:**
  - Flexible approach allowing the engine to explore join strategies.

- **Problems:**
  - Repeated copying, parallelism and materialization of CLOBs.
  - Creates opaque intermediate objects.
The construction of an XML document splits in two phases:

- Content creation – where relational data is produced
- Tagging and Structuring – where the relational data is structured and tagged to produce the XML document.

We consider only inside engine approaches.
1. Content Creation: *Redundant Relation Approach*

- Simple way to produce content:
  - Join all the source tables.

```sql
Select cust.*, acct.*, porder.*, pay.*, item.*
From Customer cust
  left join Account acct on cust.id = acct.custId
  left join PurchOrder porder on cust.id = porder.custId
  left join Item item on porder.id = item.pold
  left join Payment pay on porder.id = pay.pold
```
1. Content Creation: *Redundant Relation Approach*

- **Advantages:**
  - Using regular, set-oriented relational processing.

- **Problems:**
  - Content and processing redundancy.
  - Multi-data dependencies are created. This increases both the size and the amount of processing.
2. Content Creation: *Outer Union* Approach

- Reduce redundancy.

- How?
  - Separate the representation of a given child of a parent from the representation of the other children of the same parent;
  - Example: one tuple of the relational result should represent *either* an account *or* a purchase order associated with the customer, not both.
2. Content Creation: *Outer Union* Approach

- Each path, from root to leaf, is computed by means of joins.
- Produces one tuple per data item with all information about its ancestors.
- Final step: glue all together.
- *Outer Union* is an analogy to *Outer Join*.

![Diagram](image)
2. Content Creation: *Outer Union* Approach
2. Content Creation: *Outer Union* Approach
2. Content Creation: *Outer Union* Approach
2. Content Creation: *Outer Union* Approach

- **Problems:**
  - This approach eliminates much of the data redundancy but it increased the number of tuples in the result.
  - Number of columns in the result increases with the depth and width of the XML document. This may lead to an increased processing overhead.
3. Structuring/Tagging: Hash-Based Tagger

- Can be done inside or outside the relational engine.

**Inside Engine:**
- Implement as an aggregate function that is invoked as the last processing step, after the content has been produced.
- The function would logically perform the function of all the XML constructors and XMLAGGs in the user query.
3. Structuring/Tagging: Hash-Based Tagger

- Tag and Structure results (inside or outside the engine):
  - Group all siblings in the desired XML document under the same parent.
  - Extract the information from each tuple and tag it to produce the XML result.
3. Structuring/Tagging: Hash-Based Tagger

- **Group siblings:**
  - Use main-memory hash table to look up the parent of a node, given the parent’s type and id information.
  - Whenever a tuple containing information about an XML element is seen, it is hashed on the element’s type and the ids of its ancestors in order to verify if its parent is already present in the hash table.
3. Structuring/Tagging: Hash-Based Tagger

- Group siblings:
  - If parent is present: a new XML element is created and added as a child of the parent.
  - If parent is not present: search for grandparent.
    - If grandparent exists, the parent is created and then the child is created.
    - If grandparent does not exist, the procedure is repeated until an ancestor is present in the hash table or root of the document is reached.
3. Structuring/Tagging: Hash-Based Tagger

- After all the input tuples have been hashed, the entire tagged structured can be written out as an XML file.

- Limitation:
  - Performance can degrade rapidly when there is insufficient memory to hold the hash table and the intermediate result.
The main limitation with Late Tagging, Late Structuring approach is that complex memory management needs to be performed when memory is scarce.

The solution:
- Use relational engine to produce “structured content” which can be tagged using a constant space tagger.
1. Structured Content Creation: *Sorted Outer Union* Approach

- The key to structuring relational content is to order it the same way that it needs to appear in the result XML.

- This can be achieved by ensuring that:
  - Parent information occurs before, or with, child information.
  - The information about a particular node and its descendants is not mixed in with the information about the non-descendant nodes.
1. Structured Content Creation: Sorted Outer Union Approach

- To ensure the previous conditions is required:
  - Sort the result of the Node Outer Union on its id field, with the ids of parent nodes occurring higher in the sort order than the ids of children node.
  - Ensure that result is in document order.
  - Tuples having null values in the sort fields occur before tuples having non-null values.
1. Structured Content Creation: Sorted Outer Union Approach

- This approach has the advantage of scaling to large data volumes because relational database sorting is disk-friendly.

- It can also ensure user-specified orderings with little additional cost.
2. Tagging Sorted Data: *Constant Space Tagger*

- Once structured content is created, the next step is to tag and construct the result XML document.

- Tuples arrive in document order
  - They can be immediately tagged and written.
2. Tagging Sorted Data: Constant Space Tagger

- Tagger only requires memory to remember the parent ids of the last tuple seen.
  - The ids are used to detect when all the children of a particular node have been seen in order to close the tag and be written out.

- The storage required is proportional only to the level of nesting and is independent of the size of the XML document.
## PERFORMANCE COMPARISON

### PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range of Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Fan Out</td>
<td>2, 3, 4</td>
<td>2</td>
</tr>
<tr>
<td>Query Depth</td>
<td>2, 3, 4</td>
<td>2</td>
</tr>
<tr>
<td># Roots</td>
<td>1, 50, 500, 5000, 40000</td>
<td>5000</td>
</tr>
<tr>
<td># Leaf Tuples</td>
<td>160000, 320000, 480000</td>
<td>320000</td>
</tr>
</tbody>
</table>

Parameter Settings for Experiments
Experimental Setup

- Database Engine
  - DB2 Universal Database System

- Machine
  - Pentium 366MHz processor with 256Mb of main memory running Windows NT4.0

- Implementations
  - The XML constructors and XMLAGG were implemented as new built-in functions
  - The Stores Procedure approach was implemented as an stored procedure
  - The “outside the engine” approaches were implemented as local embedded-SQL programs
The worst result in the “inside engine” approaches takes approximately one third of the time that the worst result in the “outside engine” approaches.

Unsorted and Sorted Outer Union “inside engine” version takes less than half the time to execute that the correspondent “outside engine” versions
Constructing as XML document should be done inside the engine to maximize performance,
The execution time for all the approaches increases with query depth.

The dramatic increase for the De-Correlated CLOB approach, is related with the relational query optimizer.
Performance Comparison

RESULTS

Varying Number of Roots (Inside Engine)
Constructing an XML document inside the relational engine is more efficient than doing so outside the engine.

When processing can be done in main memory, a stable approach that is always among the very best is the Unsorted Outer Union approach SQL-Based Language Specification.

When processing cannot be done in main memory, the Sorted Outer Union approach is the approach of choice.