Module 5

Implementation of XQuery
(Rewrite, Indexes, Runtime System)
XQuery: a language at the cross-roads

• Query languages
• Functional programming languages
• Object-oriented languages
• Procedural languages
• Some new features: context sensitive semantics

• Processing XQuery has to learn from all those fields, plus innovate
XQuery processing: old and new

• **Functional programming**
  + Environment for expressions
  + Expressions nested with full generality
  + Lazy evaluation
  - Data Model, schemas, type system, and query language
  - Contextual semantics for expressions
  - Side effects
  - Non-determinism in logic operations, others
  - *Streaming execution*
    - Logical/physical data mismatch, appropriate optimizations

• **Relational query languages (SQL)**
  + High level construct (FLWOR/Select-From-Where)
  + Streaming execution
  + Logical/physical data mismatch and the appropriate optimizations
  - *Data Model, schemas, type system, and query language*
  - Expressive power
  - Error handling
  - 2 values logic
XQuery processing: old and new

• **Object-oriented query languages (OQL)**
  + Expressions nested with full generality
  + Nodes with node/object identity
  - Topological order for nodes
    - *Data Model, schemas, type system, and query language*
  - *Side effects*
  - *Streaming execution*

• **Imperative languages (e.g. Java)**
  + Side effects
  + Error handling
    - *Data Model, schemas, type system, and query language*
  - Non-determinism for logic operators
  - *Lazy evaluation and streaming*
    - *Logical/physical data mismatch and the appropriate optimizations*
    - *Possibility of handling large volumes of data*
Major steps in XML Query processing

- Parsing & Verification
- Code rewriting
- Code generation
- Executable code

Compilation

Data access pattern (APIs)

Query

Internal query/program representation

Lower level internal query representation
(SQL) Query Processing 101

```
SELECT  *
FROM      Hotels h, Cities c
WHERE   h.city = c.name;
```

Parser & Query Optimizer

Execution Engine

Hash Join

Scan(Hotels)  Scan(Cities)

<Ritz, Paris, ...>
<Rитsser Hase, Passau, ...>
<Edgewater, Madison, ...>

Indexes & Base Data

Catalogue

Schema info, DB statistics
(SQL) Join Ordering

• Cost of a Cartesian Product: \(n \times m\)
  – \(n, m\) size of the two input tables
• \(R \times S \times T; \quad \text{card}(R) = \text{card}(T) = 1; \quad \text{card}(S) = 10\)
  – \((R \times S) \times T\) costs \(10 + 10 = 20\)
  – \((R \times T) \times S\) costs \(1 + 10 = 11\)
• For queries with many joins, join ordering responsible for orders of magnitude difference
  – Millisecs vs. Decades in response time
• How relevant is join ordering for XQuery?
(SQL) Query Rewrite

SELECT *
FROM   A, B, C
WHERE  A.a = B.b AND B.b = C.c
is transformed to
SELECT *
FROM   A, B, C
WHERE  A.a = B.b AND B.b = C.c  AND A.a = C.c

• Why is this transformation good (or bad)?
• How relevant is this for XQuery?
Code rewriting

• Code rewritings goals
  – Reduce the level of abstraction
  – Reduce the execution cost

• Code rewriting concepts
  – Code representation
    • \texttt{db}: algebras
  – Code transformations
    • \texttt{db}: rewriting rules
  – Cost transformation policy
    • \texttt{db}: search strategies
  – Code cost estimation
Code representation

• Is “algebra” the right metaphor? Or expressions? Annotated expressions? Automata?
• Standard algebra for XQuery?
• Redundant algebra or not?
  – Core algebra in the XQuery Formal Semantics
• Logical vs. physical algebra?
  – What is the “physical plan” for 1+1?
• Additional structures, e.g. dataflow graphs? Dependency graphs?

See Compiler transformations for High-Performance computing
Bacon, Graham, Sharp
Automata representation

- Path expressions
  \$x/chapter//section/title

- \([Yfilter’03, Gupta’03, etc]\)

- NFA vs. DFA vs. AFA

- one path vs. a set of paths

- Problems
  - Not extensible to full XQuery
  - Better suited for push execution, pull is harder
  - Lazy evaluation is hard

```xml
<book>
  <chapter>
    <section>
      <title/>
    </section>
  </chapter>
</book>
```
TLC Algebra
(Jagadish et al. 2004)

- XML Query tree patterns (called *twigs*)
- Annotated with predicates
- Tree matching as basic operation
  - Logical and physical operation
- Tree pattern matching => tuple bindings (i.e. relations)
- Tuples combined via classical relational algebra
  - Select, project, join, duplicate-elim., ...
XQuery Expressions (BEA implementation)

- Expressions built during parsing
- (almost) 1-1 mapping between expressions in XQuery and internal ones
  - Differences: Match (expr, NodeTest) for path expressions
- Annotated expressions
  - *E.g. unordered* is an annotation
  - Annotations exploited during optimization
- Redundant algebra
  - *E.g. general FLWR, but also LET and MAP*
  - *E.g. typeswitch, but also instanceof and conditionals*
- Support for dataflow analysis is fundamental
Expressions

- Constants
- Complex Constants
- Variable
  - ForLetVariable
  - Parameter
  - CountVariable
- ExternalVariable
- CastExpr
- TreatExpr
- IfThenElseExpr
- InstanceOfExpr
Expression representation example

for $line in $doc/Order/OrderLine
where $line/SellersID eq 1
return <lineItem>{$line/Item/ID}</lineItem>

for $line in $doc/Order/OrderLine
where xs:integer(fn:data($line/SellersID)) eq 1
return <lineItem>{$line/Item/ID}</lineItem>
Dataflow Analysis

• Annotate each operator (attribute grammars)
  – Type of output (e.g., BookType*)
  – Is output sorted? Does it contain duplicates?
  – Has output node ids? Are node ids needed?

• Annotations computed in walks through plan
  – Intrinsinc: e.g., preserves sorting
  – Synthetic: e.g., type, sorted
  – Inherited: e.g., node ids are required

• Optimizations based on annotations
  – Eliminate redundant sort operators
  – Avoid generation of node ids in streaming apps
Dataflow Analysis: Static Type

Match("book")

elem book of BookType

FO:children

elem book of BookType
or elem thesis of BookType

FO:children

elem bib of BibType

doc of BibType

validate as "bib.xsd"

doc("bib.xml")

item*
XQuery logical rewritings

• Algebraic properties of comparisons
• Algebraic properties of Boolean operators
• LET clause folding and unfolding
• Function inlining
• FLWOR nesting and unnesting
• FOR clauses minimization
• Constant folding
• Common sub-expressions factorization
• Type based rewritings
• Navigation based rewritings
• “Join ordering”
(SQL) Query Rewrite

SELECT *  
FROM    A, B, C  
WHERE  A.a = B.b AND B.b = C.c  
is transformed to  
SELECT *  
FROM    A, B, C  
WHERE  A.a = B.b AND B.b = C.c AND A.a = C.c  

• Why is this transformation good (or bad)?  
• How relevant is this for XQuery?
(SQL) Query Rewrite

SELECT A.a
FROM    A
WHERE  A.a in (SELECT x FROM X);

is transformed to (assuming x is key):

SELECT A.a
FROM    A, X
WHERE  A.a = X.x

• Why is this transformation good (or bad)?
• When can this transformation be applied?
Algebraic properties of comparisons

• General comparisons not reflexive, transitive
  – (1,3) = (1,2) (*but also !=, <, >, <=, >=*)
  – Reasons
    • implicit existential quantification, dynamic casts

• Negation rule does not hold
  – fn:not($x = $y) is not equivalent to $x != $y

• General comparison not transitive, not reflexive

• Value comparisons are *almost* transitive
  – Exception:
    • xs:decimal due to the loss of precision

*Impact on grouping, hashing, indexing, caching!!!*
What is a correct Rewriting

• E₁ -> E₂ is a legal rewriting iff
  – Type(E₂) is a subtype of Type(E₁)
  – FreeVar(E₂) is a subset of FreeVar(E₁)
  – For any binding of free variables:
    • If E₁ must return error (acc. Semantics), then E₂ must return error (not mandatory the same error)
    • If E₂ can return a value (non error) then E₂ must return a value among the values accepted for E₁, or error
    • Note: Xquery is non-deterministic

• This definition allows the rewrite E₁->ERROR
  – Trust your vendor she does not do that for all E₁
Properties of Boolean operators

• *Among of the most useful logical rewritings: PCNF and PDNF*

• *And, or* are commutative & allow short-circuiting
  – For optimization purposes
• But are non-deterministic
  – Surprise for some programmers :(
    • If (($x$ castable as xs:integer) and (($x$ cast as xs:integer) eq 2) ) .....  
• 2 value logic
  – () is converted into fn:false() before use
• Conventional distributivity rules for and, not, or do hold
LET clause folding

• Traditional FP rewriting
  let $x := 3$                        $3+2$
  return $x +2$

• Not so easy !
  let $x := <a/>$                (<a/>, <a/> )          NO. Side effects. (Node identity)
  return ($x, $x$ )

  declare namespace ns=“uri1”                     NO. Context sensitive
  let $x := <ns:a/>$                                    namespace processing.
  return <b xmlns:ns=“uri2”>{$x}</b>

    declare namespace ns:=“uri1”
    <b xmlns:ns=“uri2”>{<ns:a/>}{<ns:a/>}</b>

XML does not allow cut and paste
LET clause folding (cont.)

• Impact of unordered{..} /* context sensitive*/

\[
\text{let } x := (y/a/b)[1] \quad \text{the c’s of a specific b parent}
\]
\[
\text{return unordered } \{ x/c \} \quad \text{(in no particular order)}
\]

not equivalent to

\[
\text{unordered } \{(y/a/b)[1]/c\} \quad \text{the c’s of “some” b}
\]
\[
\text{(in no particular order)}
\]
LET clause folding: fixing the node construction problem

- Sufficient conditions

\[
\begin{align*}
\text{let } x & := \text{expr1} \\
\text{return expr2'}
\end{align*}
\]

where expr2' is expr2 with substitution \{$x/$expr1\}

- Expr1 does never generate new nodes in the result
- OR $x$ is used (a) only once and (b) not part of a loop and (c) not input to a recursive function
- Dataflow analysis required
LET clause folding: fixing the namespace problem

• Context sensitivity for namespaces
  ¹. Namespace resolution during query analysis
  ². Namespace resolution during evaluation

• (1) is not a problem if:
  – Query rewriting is done after namespace resolution

• (2) could be a serious problem (***)
  – XQuery avoided it for the moment
  – Restrictions on context-sensitive operations like string -> Qname casting
LET clause unfolding

• Traditional rewriting

\[
\begin{align*}
  &\text{for } x := (1 \text{ to } 10) \quad \text{let } y := (\text{input}+2) \\
  &\quad \text{return } (\text{input}+2)+x \\
  &\text{for } x \text{ in } (1 \text{ to } 10) \quad \text{return } y+x
\end{align*}
\]

• Not so easy!

  – Same problems as above: side-effects, NS handling and unordered/ordered{..}

  – Additional problem: error handling

\[
\begin{align*}
  &\text{for } x \text{ in } (1 \text{ to } 10) \quad \text{let } y := (\text{input} \text{idiv } 0) \\
  &\quad \text{return } \text{if}(x \text{ lt } 1) \\
  &\quad \quad \text{then } (\text{input} \text{idiv } 0) \\
  &\quad \quad \text{else } x \\
  &\quad \text{return } \text{if } (x \text{ lt } 1) \\
  &\quad \quad \text{then } y \\
  &\quad \quad \text{else } x
\end{align*}
\]

Guaranteed only if runtime implements consistently lazy evaluation. Otherwise dataflow analysis and error analysis required.
Function inlining

• Traditional FP rewriting technique
  
  define function f($x as xs:integer) as xs:integer
  
  {$x+1}

  f(2)

• Not always!
  
  – Same problems as for LET (NS handling, side-effects, unordered {...})
  
  – Additional problems: *implicit operations (atomization, casts)*
    
    define function f($x as xs:double) as xs:boolean
    
    {$x instance of xs:double}

    f(2)

    (2 instance of xs:double) NO

• Make sure this rewriting is done after normalization
FLWR unnesting

• Traditional database technique
  
  for $x$ in (for $y$ in $input/a/b$
  where $y/c$ eq 3
  return $y/d$)
  where $x/e$ eq 4
  return $x$

• Problem simpler than in OQL/ODMG
  – No nested collections in XML

• Order-by, count variables and unordered{...} limit the limits applicability
FLWR unnesting (cont.)

• Another traditional database technique

  for $x$ in $input/a/b$
  where $x/c$ eq 3
  return (for $y$ in $x/d$
          where $x/e$ eq 4
          return $y$
  )

  for $x$ in $input/a/b$,
  $y$ in $x/d$
  where ($x/e$ eq 4) and ($x/c$ eq 3)
  return $y$

• Same comments apply
FOR clauses minimization

• Yet another useful rewriting technique

for $x$ in $\text{input}/a/b$, $y$ in $\text{input}/c$
  where ($x/d$ eq 3)
return $y/e$

for $x$ in $\text{input}/a/b$, $y$ in $\text{input}/c$
  where ($x/d$ eq 3) and $y/f$ eq 4
return $y/e$

for $x$ in $\text{input}/a/b$
  $y$ in $\text{input}/c$
where ($x/d$ eq 3)
return $\text{input}/c/e$

for $x$ in $\text{input}/a/b$
  $y$ in $\text{input}/c$
where ($x/d$ eq 3) and $\text{input}/c/f$ eq 4
return $\text{input}/c/e$

for $x$ in $\text{input}/a/b$
  $y$ in $\text{input}/c$
where ($x/d$ eq 3)
return $<e>{\{x, y\}}</e>$
Constant folding

• Yet another traditional technique

for $x$ in (1 to 10)                         for $x$ in (1 to 10)                         YES
  where $x$ eq 3                               where $x$ eq 3                return $x+1$                                    return (3+1)
return $x$+1

for $x$ in $input/a                             for $x$ in $input/a
  where $x$ eq 3                                 where $x$ eq 3                     NO
  return <b>{$x}</b>                          return <b>{3}</b>
return ${x}</b>

for $x$ in (1.0,2.0,3.0)                       for $x$ in (1.0,2.0,3.0)          NO
  where $x$ eq 1                                  where $x$ eq 1
  return ($x$ instance of xs:integer)     return (1 instance of xs:integer)
  return (1 instance of xs:integer)
Common sub-expression factorization

• Preliminary questions
  – *Same* expression?
  – *Same* context?
  – *Error* “equivalence”?  
  – Create the same *new nodes*?

for $x$ in $input/a/b$
where $x/c$ lt 3
return if ($x/c$ lt 2)
  then if ($x/c$ eq 1)
    then (1 idiv 0)
    else $x/c$+1
  else if($x/c$ eq 0)
    then (1 idiv 0)
    else $x/c$+2
else if($x/c$ eq 0)
  then $y$ = (1 idiv 0)
  for $x$ in $input/a/b$
  where $x/c$ lt 3
  return if($x/c$ lt 2)
    then if ($x/c$ eq 1)
      then $y$
      else $x/c$+1
    else if($x/c$ eq 0)
      then $y$
      else $x/c$+2
  else $x/c$+2
Type-based rewritings

• Type-based optimizations:
  – Increase the advantages of lazy evaluation
    • $input/a/b/c ((($input/a)[1]/b[1])/c)[1]
  – Eliminate the need for expensive operations (sort, dup-elim)
    • $input//a/b $input/c/d/a/b
  – Static dispatch for overloaded functions
    • e.g. min, max, avg, arithmetics, comparisons
    • Maximizes the use of indexes
  – Elimination of no-operations
    • e.g. casts, atomization, boolean effective value
  – Choice of various run-time implementations for certain logical operations
Dealing with backwards navigation

• Replace backwards navigation with forward navigation

```xml
for $x$ in $\text{input/a/b} $
return <c>{$x/.., $x/d}</c>
```

```xml
for $y$ in $\text{input/a,}$
$x$ in $\text{y/b}$
return <c>{$y, $x/d}</c>
```

```
for $x$ in $\text{input/a/b}$
return <c>{$x//e/..}</c>
```

• Enables streaming

YES

??
More compiler support for efficient execution

- Streaming vs. data materialization
- Node identifiers handling
- Document order handling
- Scheduling for parallel execution
- Projecting input data streams
When should we materialize?

• Traditional operators (e.g. sort)
• Other conditions:
  – Whenever a variable is used multiple times
  – Whenever a variable is used as part of a loop
  – Whenever the content of a variable is given as input to a recursive function
  – In case of backwards navigation
• Those are the **ONLY** cases
• In most cases, materialization can be *partial* and *lazy*
• Compiler can detect those cases via dataflow analysis
How can we minimize the use of node identifiers?

- Node identifiers are required by the XML Data model but onerous (time, space)
- Solution:
  - Decouple the node construction operation from the node id generation operation
  - Generate node ids only if really needed
    - Only if the query contains (after optimization) operators that need node identifiers (e.g. sort by doc order, is, parent, <<) OR node identifiers are required for the result
- Compiler support: dataflow analysis
How can we deal with path expressions?

- Sorting by document order and duplicate elimination required by the XQuery semantics but very expensive

- Semantic conditions
  - $document / a / b / c
    - Guaranteed to return results in doc order and not to have duplicates
  - $document / a // b
    - Guaranteed to return results in doc order and not to contain duplicates
  - $document // a / b
    - NOT guaranteed to return results in doc order but guaranteed not to contain duplicates
  - $document // a // b $document / a / .. / b
    - Nothing can be said in general
Parallel execution

\[ ns1:WS1($input) + ns2:WS2($input) \]

for \( x \) in \((1\) to \(10)\)

\[ return ns:WS($i) \]

• Obviously certain subexpressions of an expression can (and should...) be executed in parallel
  – Scheduling based on data dependency
• Horizontal and vertical partitioning
• Interaction between errors and parallelism

See David J. DeWitt, Jim Gray:
XQuery expression analysis

• How many times does an expression use a variable?
• Is an expression using a variable as part of a loop?
• Is an expression a map on a certain variable?
• Is an expression guaranteed to return results in doc order?
• Is an expression guaranteed to return (node) distinct results?
• Is an expression a “function”?
• Can the result of an expression contain newly created nodes?
• Is the evaluation of an expression context-sensitive?
• Can an expression raise user errors?
• Is a sub expression of an expression guaranteed to be executed?
• Etc.
Compiling XQuery vs. XSLT

• Empiric assertion: it depends on the entropy level in the data (see M. Champion xml-dev):
  – XSLT easier to use if the shape of the data is totally unknown (entropy high)
  – XQuery easier to use if the shape of the data is known (entropy low)

• Dataflow analysis possible in XQuery, much harder in XSLT
  – Static typing, error detection, lots of optimizations

• Conclusion: less entropy means more potential for optimization, unsurprisingly.
Data Storage and Indexing
Major steps in XML Query processing

Compilation

Query

Parsing & Verification

Code rewriting

Internal query/program representation

Code generation

Lower level internal query representation

Data access pattern (APIs)

Executable code
Questions to ask for XML data storage

• *What* actions are done with XML data?
• *Where* does the XML data live?
• *How* is the XML data processed?
• *In which* granularity is XML data processed?

• There is no one fits all solution !?!
(This is an open research question.)
What?

• Possible uses of XML data
  – ship (serialize)
  – validate
  – query
  – transform (create new XML data)
  – update
  – persist

• Example:
  – UNICODE reasonably good to ship XML data
  – UNICODE terrible to query XML data
Where?

• Possible locations for XML data
  – wire (XML messages)
  – main-memory (intermediate query results)
  – disk (database)
  – mobile devices

• Example
  – Compression great for wire and mobile devices
  – Compression not good for main-memory (?)
How?

• Alternative ways to process XML data
  – materialized, all or nothing
  – streaming (on demand)
  – anything in between

• Examples
  – trees good for materialization
  – trees bad for stream-based processing
Granularity?

• Possible granularities for data processing:
  – documents
  – items (nodes and atomic values)
  – tokens (events)
  – bytes

• Example
  – tokens good for fine granularity (items)
  – tokens bad for whole documents
Scenario I: XML Cache

- Cache XHTML pages or results of Web Service calls

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ship</td>
<td>yes</td>
<td></td>
<td></td>
<td>wire</td>
<td>yes</td>
</tr>
<tr>
<td>validate</td>
<td>maybe</td>
<td>m.-m.</td>
<td>yes</td>
<td>stream</td>
<td>maybe</td>
</tr>
<tr>
<td>query</td>
<td>no</td>
<td>disk</td>
<td>yes</td>
<td>granularity</td>
<td>docs/items</td>
</tr>
<tr>
<td>transform</td>
<td>maybe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scenario II: Message Broker

- Route messages according to simple XPath rules
- Do simple transformations

<table>
<thead>
<tr>
<th></th>
<th>ship</th>
<th>wire</th>
<th>materialize</th>
<th>validate</th>
<th>m.-m.</th>
<th>stream</th>
<th>query</th>
<th>disk</th>
<th>granularity</th>
<th>transform</th>
<th>update</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td>yes</td>
<td>m.-m.</td>
<td>stream</td>
<td>yes</td>
<td></td>
<td>docs</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scenario III: XQuery Processor

- apply complex functions
- construct query results

<table>
<thead>
<tr>
<th></th>
<th>ship</th>
<th>wire</th>
<th>materialize</th>
<th>validate</th>
<th>m.-m.</th>
<th>stream</th>
<th>transform</th>
<th>granularity</th>
<th>update</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

query: yes, disk: maybe, granularity: item
Scenario IV: XML Database

- Store and archive XML data

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ship</td>
<td>yes</td>
<td>wire</td>
<td>no</td>
<td>materialize</td>
</tr>
<tr>
<td>validate</td>
<td>yes</td>
<td>m.-m.</td>
<td>yes</td>
<td>stream</td>
</tr>
<tr>
<td>query</td>
<td>yes</td>
<td>disk</td>
<td>yes</td>
<td>granularity</td>
</tr>
<tr>
<td>transfor m</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Object Stores vs. XML Stores

• **Similarities**
  – nodes are like objects
  – identifiers to access data
  – support for updates

• **Differences**
  – XML: tree not graph
  – XML: everything is ordered
  – XML: streaming is essential
  – XML: dual representation (lexical + binary)
  – XML: data is context-sensitive
XML Data Representation Issues

• Data Model Issues
  – InfoSet vs. PSVI vs. XQuery data model

• Storage Structures basic Issues
  1. Lexical-based vs. typed-based vs. both
  2. Node identifiers support
  3. Context-sensitive data (namespaces, base-uri)
  4. Data + order : separate or intermixed
  5. Data + metadata : separate or intermixed
  6. Data + indexes : separate of intermixed
  7. Avoiding data copying

  Storage alternatives: trees, arrays, tables
  Indexing
  APIs

• Storage Optimizations
  – compression?, pooling?, partitioning?
Lexical vs. Type-based

• Data model requires both properties, but allows only one to be stored and compute the other

• Functional dependencies
  – string + type annotation -> value-based
  – value + type annotation -> schema-norm. string

  Example
  „0001“ + xs:integer -> 1
  1 + xs:integer -> „1“

• Tradeoffs:
  – Space vs. Accuracy
  – Redundancy: cost of updates
  – indexing: restricted applicability
Node Identifiers Considerations

• XQuery Data Model Requirements
  – identify a node uniquely (implements identity)
  – lives as long as node lives
  – robust to updates

• Identifiers might include additional information
  – Schema/type information
  – Document order
  – Parent/child relationship
  – Ancestor/descendent relationship
  – Document information

• Required for indexes
Simple Node Identifiers

• **Examples:**
  - **Alternative 1 (data: trees)**
    • id of document (integer)
    • pre-order number of node in document (integer)
  - **Alternative 2 (data: plain text)**
    • file name
    • offset in file

• **Encode document ordering (Alternative 1)**
  - identity: doc1 = doc2 AND pre1 = pre2
  - order: doc1 < doc2
    OR (doc1 = doc2 AND pre1 < pre2)

• Not robust to updates
• Not able to answer more complex queries
Dewey Order
Tatrinov et al. 2002

• Idea:
  – Generate surrogates for each path
  – 1.2.3 identifies the third child of the second child of the first child of the given root

• Assessment;
  – **good**: order comparison, ancestor/descendent easy
  – **bad**: updates expensive, space overhead

• Improvement: ORDPath Bit Encoding
  O’Neil et al. 2004 (Microsoft SQL Server)
Example: Dewey Order
XML Storage Alternatives

• Plain Text (UNICODE)
• Trees with Random Access
• Binary XML / arrays of events (tokens)
• Tuples (e.g., mapping to RDBMS)
Plain Text

- Use XML standards to encode data
- Advantages:
  - simple, universal
  - indexing possible
- Disadvantages:
  - need to re-parse (re-validate) all the time
  - no compliance with XQuery data model (collections)
  - not an option for XQuery processing
Trees

• XML data model uses tree semantics
  – use Trees/Forests to represent XML instances
  – annotate nodes of tree with data model info

• Example

  `<f1>`
  `<f2>..</f2>  <f3>..</f3>`
  `<f4>  <f7/>  <f8>..</f8>  </f4>`
  `<f5/>  <f6>..</f6>`
  `</f1>`
Trees

• Advantages
  – natural representation of XML data
  – good support for navigation, updates index built into the data structure
  – compliance with DOM standard interface

• Disadvantages
  – difficult to use in streaming environment
  – difficult to partition
  – high overhead: *mixes indexes and data*
  – index everything

• Example: DOM, *others*

• Lazy trees possible: minimize IOs, able to handle large volumes of data
Natix (trees on disk)

- Each sub-tree is stored in a *record*
- Store records in blocks as in any database
- If record grows beyond size of block: *split*
- Split: establish *proxy nodes* for subtrees
- Technical details:
  - use B-trees to organize space
  - use special concurrency & recovery techniques
Natix

<bib>
  <book>
    <title>...</title>
    <author>...</author>
  </book>
</bib>
Binary XML as a flat array of „events“

• Linear representation of XML data
  – pre-order traversal of XML tree
• Node -> array of events (or tokens)
  – tokens carry the data model information
• Advantages
  – good support for stream-based processing
  – low overhead: separate indexes from data
  – logical compliance with SAX standard interface
• Disadvantages
  – difficult to debug, difficult programming model
Example Binary XML as an array of tokens

```xml
<?xml version=„1.0“>
<order id=„4711“ >
  <date>2003-08-19</date>
  <lineitem xmlns = „www.boo.com“ >
  </lineitem>
</order>
```
<xml version="1.0">  
<order id="4711">  
  <date>2003-08-19</date>  
  <lineitem xmlns="www.boo.com">  
  </lineitem>  
</order>  
</xml>
BeginDocument()
BeginElement(„order“, „rn:PO“, 1)
BeginAttribute(„id“, „xs:Integer“, 2)
CharData(„4711“)
Integer(4711)
EndAttribute()
BeginElement(„date“, „Element of Date“, 3)
Text(„2003-08-19“, 4)
Date(2003-08-19)
EndElement()
BeginElement(„www.boo.com:lineitem“, „xs:untypedAny“, 5)
NameSpace(„www.boo.com“, 6)
EndElement()
EndElement()
EndDocument()
Binary XML

• Discussion as part of the W3C
• Processing XML is only one of the target goals
• Other goals:
  – Data compression for transmission: WS, mobile
• Open questions today: can we achieve all goals with a single solution? Will it be disruptive?
• Data model questions: Infoset or XQuery Data Model?
• Is streaming a strict requirement or not?
• More to come in the next months/years.
Compact Binary XML in Oracle

• Binary serialization of XML Infoset
  – Significant compression over textual format
  – Used in all tiers of Oracle stack: DB, iAS, etc.

• Tokenizes XML Tag names, namespace URIs and prefixes
  – Generic token table used by binary XML, XML index and in-memory instances

• *(Optionally)* Exploits schema information for further optimization
  – Encode values in native format (e.g. integers and floats)
  – Avoid tokens when order is known
  – For fully structured XML (relational), format very similar to current row format *(continuity of storage !)*

• Provide for schema versioning / evolution
  – Allow any backwards-compatible schema evolution, plus a few incompatible changes, without data migration
XML Data represented as tuples

• Motivation: Use an RDBMS infrastructure to store and process the XML data
  – transactions
  – scalability
  – richness and maturity of RDBMS

• Alternative relational storage approaches:
  – Store XML as Blob (text, binary)
  – Generic shredding of the data (edge, binary, ...)
  – Map XML schema to relational schema
  – Binary (new) XML storage integrated tightly with the relational processor
Mapping XML to tuples

• **External** to the relational engine
  – Use when:
    • The structure of the data is relatively simple and fixed
    • The set of queries is known in advance
  – Processing involves hand written SQL queries + procedural logic
  – Frequently used, but not advantageous
    • Very expensive (performance and productivity)
    • Server communication for every single data fetch
    • Very limited solution

• **Internally** by the relational engine
  – A whole tutorial in Sigmod’05
XML Example

<person, id = 4711>
  <name> Lilly Potter </name>
  <child> <person, id = 314>
    <name> Harry Potter </name>
    <hobby> Quidditch </hobby>
  </child>
</person>

<person, id = 666>
  <name> James Potter </name>
  <child> 314 </child>
</person>
<person, id = 4711>
   <name> Lilly Potter </name>
   <child> <person, id = 314>
      <name> Harry Potter </name>
   </child>
</person>

<person, id = 666>
   <name> James Potter </name>
   <child> 314 </child>
</person>
### Edge Approach
(Florescu & Kossmann 99)

#### Edge Table

<table>
<thead>
<tr>
<th>Source</th>
<th>Label</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>person</td>
<td>4711</td>
</tr>
<tr>
<td>0</td>
<td>person</td>
<td>666</td>
</tr>
<tr>
<td>4711</td>
<td>name</td>
<td>v1</td>
</tr>
<tr>
<td>4711</td>
<td>child</td>
<td>i314</td>
</tr>
<tr>
<td>666</td>
<td>name</td>
<td>v2</td>
</tr>
<tr>
<td>666</td>
<td>child</td>
<td>i314</td>
</tr>
</tbody>
</table>

#### Value Table (String)

<table>
<thead>
<tr>
<th>Id</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>Lilly Potter</td>
</tr>
<tr>
<td>v2</td>
<td>James Potter</td>
</tr>
<tr>
<td>v3</td>
<td>Harry Potter</td>
</tr>
</tbody>
</table>

#### Value Table (Integer)

<table>
<thead>
<tr>
<th>Id</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v4</td>
<td>12</td>
</tr>
</tbody>
</table>
Binary Approach
Partition *Edge* Table by *Label*

<table>
<thead>
<tr>
<th>Person Tabelle</th>
<th>Name Tabelle</th>
<th>Child Tabelle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source</strong></td>
<td><strong>Target</strong></td>
<td><strong>Source</strong></td>
</tr>
<tr>
<td>0</td>
<td>4711</td>
<td>4711</td>
</tr>
<tr>
<td>0</td>
<td>666</td>
<td>666</td>
</tr>
<tr>
<td>i314</td>
<td>314</td>
<td>314</td>
</tr>
</tbody>
</table>

*Age Tabelle*

<table>
<thead>
<tr>
<th><strong>Source</strong></th>
<th><strong>Target</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>314</td>
<td>v4</td>
</tr>
</tbody>
</table>
Tree Encoding
(Grust 2004)

• For every node of tree, keep info
  – pre: pre-order number
  – size: number of descendants
  – level: depth of node in tree
  – kind: element, attribute, name space, ...
  – prop: name and type
  – frag: document id (forests)
### Example: Tree Encoding

<table>
<thead>
<tr>
<th>pre</th>
<th>size</th>
<th>level</th>
<th>kind</th>
<th>prop</th>
<th>frag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>0</td>
<td>elem</td>
<td>person</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>attr</td>
<td>id</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>elem</td>
<td>name</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>elem</td>
<td>child</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
<td>elem</td>
<td>person</td>
<td>1</td>
</tr>
</tbody>
</table>
XML Triple (R. Bayer 2003)

<table>
<thead>
<tr>
<th>Pfad</th>
<th>Surrogat</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author[1]/FN[1]</td>
<td>2.1.1.1</td>
<td>Rudolf</td>
</tr>
<tr>
<td>Author[1]/LN[1]</td>
<td>2.1.2.1</td>
<td>Bayer</td>
</tr>
</tbody>
</table>
DTD -> RDB Mapping
Shanmugasundaram et al. 1999

• Idea: Translate DTDs into Relations
  – Element Types -> Tables
  – Attributes -> Columns
  – Nesting (= relationships) -> Tables
  – „Inlining“ reduces fragmentation
• Special treatment for recursive DTDs
• Surrogates as keys of tables
• (Adaptions for XML Schema possible)
DTD Normalisation

• Simplify DTDs
  
  \[(e1, e2)^* \rightarrow e1^*, e2^* \quad (e1, e2)^? \rightarrow e1?, e2?\]
  
  \[(e1 | e2) \rightarrow e1?, e2? \quad e1^{**} \rightarrow e1^*\]
  
  \[e1^{*?} \rightarrow e1^* \quad e1^{??} \rightarrow e1?\]
  
  \[... \rightarrow a^*, ... \quad ... \rightarrow a^*, ....\]

• Background
  
  – regular expressions
  
  – ignore order (in RDBMS)
  
  – generalized quantifiers (be less specific)
Example

<!ELEMENT book (title, author)>  
<!ELEMENT article (title, author*)>  
<!ATTLIST book price CDATA>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT author (firstname, lastname)>  
<!ELEMENT firstname (#PCDATA)>  
<!ELEMENT lastname (#PCDATA)>  
<!ATTLIST author age CDATA>
Example: Relation „book“

```xml
<!ELEMENT book (title, author)>  
<!ELEMENT article (title, author*)>  
<!ATTLIST book price CDATA>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT author (fname, lname)>  
<!ELEMENT firstname (#PCDATA)>  
<!ELEMENT lastname (#PCDATA)>  
<!ATTLIST author age CDATA>  

```
Example: Relation „article“

<!ELEMENT book (title, author)>  
<!ELEMENT article (title, author*)>  
<!ATTLIST book price CDATA>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT author (fname, lname)>  
<!ELEMENT firstname (#PCDATA)>  
<!ELEMENT lastname (#PCDATA)>  
<!ATTLIST author age CDATA>  

article(artID, art.title)  
artAuthor(artAuthorID, artID, art.author.fname, art.author.lname, art.author.age)
Example (continued)

• Represent each element as a relation
  – element might be the root of a document

\[
\begin{aligned}
\text{title}(\text{titleId}, \text{title}) \\
\text{author}(\text{authorId}, \text{author.age}, \text{author.fname}, \text{author.lname}) \\
\text{fname}(\text{fnameId}, \text{fname}) \\
\text{lname}(\text{lnameId}, \text{lname})
\end{aligned}
\]
Recursive DTDs

```xml
<!ELEMENT book (author)>
<!ATTLIST   book title CDATA>
<!ELEMENT author (book*)>
<!ATTLIST   author name CDATA>
```

```plaintext
author(authorId, author.name)
author.book(author.bookId, authorId, author.book.title)
```
XML Data Representation Issues

• Data Model Issues
  – InfoSet vs. PSVI vs. XQuery data model

• Storage Structures Issues
  1. Lexical-based vs. typed-based vs. both
  2. Node identifiers support
  3. Context-sensitive data (namespaces, base-uri)
  4. Order support
  5. Data + metadata: separate or intermixed
  6. Data + indexes: separate or intermixed
  7. Avoiding data copying

• Storage alternatives: trees, arrays, tables

• Storage Optimizations
  – compression?, pooling?, partitioning?

• Data accesses APIs
Major steps in XML Query processing

- Parsing & Verification
- Code rewriting
- Code generation
- Executable code
- Data access pattern (APIs)
- Internal query/program representation
- Lower level internal query representation

Compilation
XML APIs: an overview

- DOM (any XML application)
- SAX (low-level XML processing)
- JSR 173 (low-level XML processing)
- TokenIterator (BEA, low level XML processing)
- XQJ / JSR 225 (XML applications)
- Microsoft XMLReader Streaming API

1. For reasonable performance, the data storage, the data APIs and the execution model have to be designed together!

2. For composability reasons the runtime operators (ie. output data) should implement the same API as the input data.
Classification Criteria

• Navigational access?
• Random access (by node id)?
• Decouple navigation from data reads?
• If streaming: push or pull ?
• Updates?
• Infoset or XQuery Data Model?
• Target programming language?
• Target data consumer? application vs. query processor
Decoupling

• Idea:
  – methods to *navigate* through data (XML tree)
  – methods to *read properties* at current position (node)

• Example: DOM (tree-based model)
  – *navigation*: firstChild, parentNode, nextSibling, …
  – *properties*: nodeName, getNamedItem, …
  – *(updates)*: createElement, setNamedItem, …)

• Assessment:
  – *good*: read parts of document, integrate existing stores
  – *bad*: materialize temp. query results, transformations
Non Decoupling

• Idea:
  – Combined navigation + read properties
  – Special methods for fast forward, reverse navigation

• Example: BEA‘s TokenIterator (token stream)
  Token getNext(), void skipToNextNode(), …

• Assessment:
  – **good**: less method calls, stream-based processing
  – **good**: integration of data from multiple sources
  – **bad**: difficult to wrap existing XML data sources
  – **bad**: reverse navigation tricky, difficult programming model
## Classification of APIs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DOM</td>
<td>InfoSet</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>SAX</td>
<td>InfoSet</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>Java</td>
</tr>
<tr>
<td>JSR173</td>
<td>InfoSet</td>
<td>(no)</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>Java</td>
</tr>
<tr>
<td>TokIter</td>
<td>XQuery</td>
<td>(no)</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>Java</td>
</tr>
<tr>
<td>XQJ</td>
<td>XQuery</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Java</td>
</tr>
<tr>
<td>MS</td>
<td>InfoSet</td>
<td>(no)</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>.Net</td>
</tr>
</tbody>
</table>
XML Data Representation Issues

• Data Model Issues
  – InfoSet vs. PSVI vs. XQuery data model

• Storage Structures basic Issues
  1. Lexical-based vs. typed-based vs. both
  2. Node identifiers support
  3. Context-sensitive data (namespaces, base-uri)
  4. Data + order: separate or intermixed
  5. Data + metadata: separate or intermixed
  6. Data + indexes: separate or intermixed
  7. Avoiding data copying

Storage alternatives: trees, arrays, tables

Indexing

APIs

• Storage Optimizations
  – compression?, pooling?, partitioning?
Classification (Compression)

- XML specific?
- Queryable?
- (Updateable?)
Compression

• Classic approaches: e.g., Lempel-Ziv, Huffman
  – decompress before queries
  – miss special opportunities to compress XML structure

• Xmill: Liefke & Suciu 2000
  – Idea: separate data and structure -> reduce entropy
  – separate data of different type -> reduce entropy
  – specialized compression algo for structure, data types

• Assessment
  – Very high compression rates for documents > 20 KB
  – Decompress before query processing (bad!)
  – Indexing the data not possible (or difficult)
Xmill Architecture

XML

Parser Path Processor

Cont. 1 → Compr.
Cont. 2 → Compr.
Cont. 3 → Compr.
Cont. 4 → Compr.

Compressed XML
Xmill Example

<book price="69.95">  
  <title> Die Wilde Wutz </title>  
  <author> D.A.K. </author>  
  <author> N.N. </author>  
</book>

– Dictionary Compression for Tags:  
  book = #1, @price = #2, title = #3, author = #4

– Containers for data types:  
  ints in C1, strings in C2

– Encode structure (/ for end tags) - skeleton:  
  gzip( #1 #2 C1 #3 C2 / #4 C2 / #4 C2 / / )
Querying Compressed Data
(Buneman, Grohe & Koch 2003)

• Idea:
  – extend Xmill
  – special compression of skeleton
  – lower compression rates,
  – but no decompression for XPath expressions
XML Data Representation Issues

• Data Model Issues
  – InfoSet vs. PSVI vs. XQuery data model

• Storage Structures basic Issues
  1. Lexical-based vs. typed-based vs. both
  2. Node identifiers support
  3. Context-sensitive data (namespaces, base-uri)
  4. Data + order : separate or intermixed
  5. Data + metadata : separate or intermixed
  6. Data + indexes : separate of intermixed
  7. Avoiding data copying

Storage alternatives: trees, arrays, tables

Indexing

APIs

• Storage Optimizations
  – compression?, pooling?, partitioning?
XML indexing

- No indexes, no performance
- *Indexing and storage*: common design
- *Indexing and query compiler*: common design
- Different kind of indexes possible
- Like in the storage case: **there is no one size fits all**
  - it all depends on the use case scenario: type of queries, volume of data, volume of queries, etc
Kinds of Indexes

1. **Value Indexes**
   - index atomic values; e.g., `//emp/salary/fn:data(.)`
   - use B+ trees (like in relational world)
   - (integration into query optimizer more tricky)

2. **Structure Indexes**
   - materialize results of path expressions
   - (pendant to Rel. join indexes, OO path indices)

3. **Full text indexes**
   - Keyword search, inverted files
   - (IR world, text extenders)

- Any combination of the above
Value Indexes: Design Considerations

• What is the **domain** of the index? (Physical Design)
  – All database
  – Document by document
  – Collection

• What is the **key** of the index? (Physical Design)
  – e.g., //emp/salary/fn:data(.) , //emp/salary/fn:string(.)
  – singletons vs. sequences
  – string vs. typed-value
  – which type? homogeneous vs. heterogeneous domains
  – composite indexes
  – indexes and errors

• Index for what **comparison**? (Physical Design)
  – =: problematic due to implicit cast + exists
  – eq, leq, ... less problematic

• When is a value index **applicable**? (Compiler)
Index for what comparison?

- Example: $x := <age>37</age>$ unvalidated
- Satisfies all the following predicates:
  - $x = 37$
  - $x = \text{xs:double}(37)$
  - $x = "37"$
- Indexes have to keep track of all possibilities
  - Index 37 as an integer, double and string
- Penalty on indexing time, indexes size
SI Example 1: Patricia Trie
Cooper et al. 2001

• Idea:
  – Partitioned Patricia Tries to index strings
  – Encode XPath expressions as strings
    (encode names, encode atomic values)

```xml
<book>
  <author>Whoever</author>
  <author>Not me</author>
  <title>No Kidding</title>
</book>
B A 1 Whoever
B A 2 Not me
B T No Kidding
```
Example 2: XASR
Kanne & Moerkotte 2000

• Implement axis as self joins of XASR table

<book>
  <author>Whoever</author>
  <author>Not me</author>
  <title>No Kidding</title>
</book>

<table>
<thead>
<tr>
<th>type</th>
<th>min</th>
<th>max</th>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
<td>4</td>
<td>null</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 3: Multi-Dim. Indexes
Grust 2002

- pre- and post order numbering (XASR)
- multi-dimensional index for window queries
Oracle’s XML Index

• Universal index for XML document collections
  – Indexes paths within documents
  – Indexes hierarchical information using dewey-style order keys
  – Indexes values as strings, numbers, dates
  – Stores base table rowid and fragment “locator”

• No dependence on Schema
  – Any data that can be converted to number or date is indexed as such regardless of Schema

• Option to index only subset of XPaths

• Allows Text (Contains) search embedded within XPath
XML Index Path Table (Oracle)

<table>
<thead>
<tr>
<th>BaseRid</th>
<th>Path</th>
<th>OrderKe</th>
<th>Value</th>
<th>Locator</th>
<th>NumValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rid1</td>
<td>po</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rid1</td>
<td>po.data</td>
<td>1</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Rid1</td>
<td>po.data.item</td>
<td>1.1</td>
<td>“foo”</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Rid1</td>
<td>po.data.pkg</td>
<td>1.2</td>
<td>“123”</td>
<td>39</td>
<td>123</td>
</tr>
<tr>
<td>Rid1</td>
<td>po.data.item</td>
<td>1.3</td>
<td>“bar”</td>
<td>58</td>
<td></td>
</tr>
</tbody>
</table>
Summary for XML data storage

• Know what you want
  – query? update? persistence? ...
• Understand the usage scenario right
• Get the big questions right
  – tree vs. arrays vs. tuples?
• Get the details right
  – compression? decoupling? indexes? identifiers?
• Open question:
  – Universal Approach for XML data storage ??
XML processing benchmark

• We cannot really compare approaches until we decide on a *comparison basis*

• XML processing *very* broad

• Industry not mature enough

• Usage patterns not clear enough

• Existing XML benchmarks (Xmark, Xmach, etc.) limited

• Strong need for a TP benchmark
Runtime Algorithms
Query Evaluation

• **Hard to discuss special algorithms**
  – Strongly depend on algebra
  – Strongly depends on the data storage, APIs and indexing

• **Main issues:**
  1. Streaming or materializing evaluations
  2. Lazy evaluation or not
Lazy Evaluation

• Compute expressions on demand
  – compute results only if they are needed
  – requires a pull-based interface (e.g. iterators)

• Example:

  declare function endlessOnes() as integer*
    { (1, endlessOnes()) };

  some $x$ in endlessOnes() satisfies $x$ eq 1

The result of this program should be: true
Lazy Evaluation

• Lazy Evaluation also good for SQL processors
  – e.g., nested queries
• Particularly important for XQuery
  – existential, universal quantification (often implicit)
  – top N, positional predicates
  – recursive functions (non terminating functions)
  – if then else expressions
  – match
  – correctness of rewritings, ...
Stream-based Processing

• Pipe input data through query operators
  – produce results before input is fully read
  – produce results incrementally
  – minimize the amount of memory required for the processing

• Stream-based processing
  – online query processing, continuous queries
  – particularly important for XML message routing

• Traditional in the database/SQL community
Stream based processing issues

• Streaming burning questions:
  – push or pull?
  – Granularity of streaming? Byte, event, item?
  – Streaming with flexible granularity?

• Pure streaming?
  – Processing Xquery needs some data materialization
  – Compiler support to detect and minimize data materialization

• Notes:
  – Streaming + Lazy Evaluation possible
  – Partial Streaming possible/necessary
Token Iterator
(Florescu et al. 2003)

• Each operator of algebra implemented as iterator
  – `open()`: prepare execution
  – `next()`: return next token
  – `skip()`: skip all tokens until first token of sibling
  – `close()`: release resources

• Conceptionally, the same as in RDMBS ...
  – pull-based
  – multiple producers, one consumer

• ... but more fine-grained
  – good for lazy evaluation; bad due to high overhead
  – special tokens to increase granularity
  – special methods (i.e., `skip()`) to avoid fine-grained access
XML Parser as TokenIterator

<book>
  <author>Whoever</author>
  <author>Not me</author>
  <title>No Kidding</title>
</book>
XML Parser as TokenIterator

open()

XML Parser

<Book>
  <Author>Whoever</Author>
  <Author>Not me</Author>
  <Title>No Kidding</Title>
</Book>
XML Parser as TokenIterator

```xml
<book>
  <author>Whoever</author>
  <author>Not me</author>
  <title>No Kidding</title>
</book>
```
XML Parser as TokenIterator

```
<book>
  <author>Whoever</author>
  <author>Not me</author>
  <title>No Kidding</title>
</book>
```
XML Parser as TokenIterator

next()

XML Parser

<book>
  <author>Whoever</author>
  <author>Not me</author>
  <title>No Kidding</title>
</book>

BE(book)

BE(author)

TEXT(Whoever)

…
$x[3]$

next()

top3

$x$
$x[3]$
$x[3]$
$x[3]$
\$x[3]\$

```
ext()

\top3

\skip()

\$x$
```
$x[3]$
$x[3]$

```
next()

top3

next()

$x$
```

```
[Diagram of a sequence with 'next()' calls and a 'top3' label]
```
$x[3]$
Common Subexpressions

Buffer Iterator Factory

next()

buffer scan

top3

next()

result of common sub-expression
Common Subexpressions

next() → top3

next()*/skip()*/ buffer scan

Buffer Iterator Factory

next() → result of common sub-expression
Common Subexpressions

Buffer Iterator Factory

result of common sub-expression

next()
Iterator Tree

for $line in $doc/Order/OrderLine
where xs:integer(fn:data($line/SellersID)) eq 1
return <lineItem>{$line/Item/ID}</lineItem>
Streaming: push vs. pull

• Pull:
  – Data consumer requests data from data producer
  – Similar in spirit with the iterator model (SQL engines)
  – Lazy evaluation easier to integrate

• Push:
  – Data triggers operations to be executed
  – More natural for evaluating automata
  – Control is still transmitted from data consumer to data producer

• See Fegaras’04 for a comparison

• Remark: pull and push can be mixed, adapters and some buffering required
Memoization  
(Diao et al. 2004)

• Memoization: cache results of expressions
  – common subexpressions (intra-query)
  – multi-query optimization (inter-query)
  – semantic caching (inter-process)

• Lazy Memoization: Cache partial results
  – occurs as a side-effect of lazy evaluation
  – cache data and state of query processing
  – optimizer detects when state needs to be kept
XQuery implementations

• Extensions of existing data management systems
  - Relational: e.g. DB2, Oracle 10g, Yukon (Microsoft)
  - Non-relational: e.g. SleepyCat

2. New, specialized XML stores and XML processors
  - Open source: e.g. dbXML, eXist, Saxon,
  - Commercial: e.g. MarkLogic, BEA
  - Data stores vs. query processors only

• Integrators
  1. do not store data per se, but they do aggregate XML data coming from multiple data sources
     – E.g. LiquidData (BEA), DataDirect

“Native XML database !!??”
XQuery implementations (cont.)

- **BEA**: [http://edocs.bea.com/liquiddata/docs10/prodover/concepts.html](http://edocs.bea.com/liquiddata/docs10/prodover/concepts.html)
- Bluestream Database Software Corp.'s XStreamDB: [http://www.bluestream.com/dr/?page=Home/Products/XStreamDB/](http://www.bluestream.com/dr/?page=Home/Products/XStreamDB/)
- Ipedo's XML Database v3.0: [http://www.ipedo.com](http://www.ipedo.com)
- IPSI's IPSI-XQ: [http://ipsi.fhg.de/oasys/projects/ipsi-xq/index_e.html](http://ipsi.fhg.de/oasys/projects/ipsi-xq/index_e.html)
- Microsoft's XML Query Language Demo: [http://XQuerieservices.com](http://XQuerieservices.com)
- Politecnico di Milano's XQBE: [http://dbgroup.elet.polimi.it/XQuery/xqbedownload.html](http://dbgroup.elet.polimi.it/XQuery/xqbedownload.html)
- QuiLogic's SQL/XML-IMDB: [http://www.quilogic.cc/xml.htm](http://www.quilogic.cc/xml.htm)
XQuery implementations (cont.)

- Software AG's Tamino XML Server:
  http://www.softwareag.com/tamino/News/tamino_41.htm
  Tamino XML Query Demo:
  http://tamino.demozone.softwareag.com/demoXQuery/index.html
- Sonic Software's Stylus Studio 5.0 (XQuery, XML Schema and XSLT IDE):
  http://www.stylusstudio.com
  Sonic XML Server:
  http://www.sonicsoftware.com/products/additional_software/extensible_information_server/
- Sourceforge's XQuery Lite: http://sourceforge.net/projects/phpxmlclasses/. See also documentation and description. PHP implementation, open-source.
- X-Hive's XQuery demo: http://www.x-hive.com/XQuery
- XQuark Group and Université de Versailles Saint-Quentin's:
  XQuark Fusion and XQuark Bridge, open-source (see also the XQuark home page)
Outline of the Presentation

• Why XML?
• Processing XML
• XQuery: the good, the bad, and the ugly
  – XML data model, XML type system, XQuery basic constructs
  – Major XQuery applications
• XML query processing
  – Compilation issues
  – Data storage and indexing
  – Runtime algorithms
• Open questions in XML query processing
• The future of XML processing (as I see it)
Some open problems

- XQuery equivalence
- XQuery subsumption
- Answering queries using views
- Memoization for XQuery
- Caching for XQuery
- Partial and lazy indexes for XML and XQuery
- XQueries independent of updates
- Xqueries independent of schema changes
- Reversing an XML transformation
- Data lineage through XQuery
- Keys and identities on the Web
Some open problems (cont.)

1. Declarative description of data access patterns; query optimization based on such descriptions
2. Integrity constraints and assertions for XML
3. Query reformulation based on XML integrity constraints
4. XQuery and full text search
5. Parallel and asynchronous execution of XQuery
6. Distributed execution of XQuery in a peer-to-peer environment
7. Automatic testing of schema verification
8. Optimistic XQuery type checking algorithm
9. Debugging and explaining XQuery behavior
10. XML diff-grams
11. Automatic XML Schema mappings
Research topics (1)

• XML query equivalence and subsumption
  
  Dan Suciu

• Algebraic query representation and optimization
  
  – Algebraic XML Construction and its Optimization in Natix, Thorsten Fiebig
    Guido Moerkotte
  
  – TAX: A Tree Algebra for XML, H. V. Jagadish, Laks V. S. Lakshmanan, Divesh
    Srivastava, et al.
  
  – Honey, I Shrunk the XQuery! --- An XML Algebra Optimization Approach, Xin
    Zhang, Bradford Pielech, Elke A. Rundensteiner
  
  – XML queries and algebra in the Enosys integration platform, the Enosys
    team
  
  – An Efficient Compressor for XML Data, Hartmut Liefke, Dan Suciu
  
  – Path Queries on Compressed XML, Peter Buneman, Martin Grohe, Christoph
    Koch
  
  – XPRESS: A Queriable Compression for XML Data, Jun-Ki Min, Myung-Jae Park,
    Chin-Wan Chung
Research topics (2)

• Views and XML
  – On views and XML, Serge Abiteboul
    Alon Halevy  Dan Suciu

• Query cost estimations
  – Using histograms to estimate answer sizes for XML, Yuqing Wu, MI Jignesh M. Patel, MI H. V. Jagadish
  – Selectivity Estimation for XML Twigs, Neoklis Polyzotis, Minos Garofalakis, and Yannis Ioannidis
  – Estimating the Selectivity of XML Path Expressions for Internet Scale Applications, Ashraf Aboulnaga, Alaa R. Alameldeen, and Jeffrey F. Naughton
Research topics (3)

• Full Text search in XML
  – XRank: Ranked Keyword Search over XML Documents, L. Guo, F. Shao, C. Botev, Jayavel Shanmugasundaram
  – Phrase matching in XML, Sihem Amer-Yahia, Mary F. Fernandez, Divesh Srivastava and Yu Xu
  – XIRQL: A language for Information Retrieval in XML Documents, N. Fuhr, K. Grbjoann
  – Integration of IR into an XML Database, Cong Yu
  – FleXPath: Flexible Structure and Full-Text Querying for XML, Sihem Amer-Yahia, Laks V. S. Lakshmanan, Shashank Pandit
Research topics (4)

• XML Query relaxation/approximation
  – Approximate matching of XML Queries, AT&T, Sihem Amer-Yahia, Nick Koudas, Divesh Srivastava
  – Approximate XML Query Answers, Sigmod’04 Neoklis Polyzotis, Minos N. Garofalakis, Yannis E. Ioannidis
  – Approximate Tree Embedding for Querying XML Data, T. Schlieder, F. Naumann.
  – Co-XML (Cooperative XML) -- UCLA
Research topics (5)

• Security and access control in XML
  – LockX: A system for efficiently querying secure XML, SungRan Cho, Sihem Amer-Yahia, Laks V. S. Lakshmanan and Divesh Srivastava
  – Cryptographically Enforced Conditional Access for XML, Gerome Miklau Dan Suciu
  – Author-Chi - A System for Secure Dissemination and Update of XML Documents, Elisa Bertino, Barbara Carminati, Elena Ferrari, Giovanni Mella
  – Compressed accessibility map: Efficient access control for XML, Ting Yu, Divesh Srivastava, Laks V.S. Lakshmanan and H. V. Jagadish
  – Secure XML Querying with Security Views, Chee-Yong Chan, Wenfei Fan, and Minos Garofalakis
Research topics (6)

• Indexes for XML
  – Accelerating XPath Evaluation in Any RDBMS, Torsten Grust, Maurice van Keulen, Jens Teubner
  – Index Structures for Path Expressions, Dan Suciu, Tova Milo
  – Indexing and Querying XML Data for Regular Path Expressions, Quo Li and Bongki Moon
  – Covering Indexes for Branching Path Queries, Kaushik Philip Bohannon, Jeff Naughton, Hank Korth
  – A Fast Index Structure for Semistructured Data, Brian Cooper, Nigel Sample, M. Franklin, Gisli Hjaltason, Shadmon
  – Anatomy of a Native XML Base Management System, Thorsten Fiebig et al.
Research topics (7)

• Query evaluation, algorithms
  – From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery. Z. Chen, H. V. Jagadish, Laks V. S. Lakshmanan, S. Paparizos
  – Holistic twig joins: Optimal XML pattern matching, Nicolas Bruno, Nick Koudas and Divesh Srivastava.
  – Navigation- vs. index-based XML multi-query processing, Nicolas Bruno, Luis Gravano, Nick Koudas and Divesh Srivastava
  – Efficiently supporting order in XML query processing, Maged El-Sayed Katica Dimitrova Elke A. Rundensteiner
Research topics (8)

- Streaming evaluation of XML queries
  - Projecting XML Documents, Amelie Marian, Jerome Simeon
  - Processing XML Streams with Deterministic Automata, Todd J. Green, Gerome Miklau, Makoto Onizuka, Dan Suciu
  - Stream Processing of XPath Queries with Predicates, Ashish Gupta, Dan Suciu
  - Query processing of streamed XML data, Leonidas Fegaras, David Levine, Sujoe Bose, Vamsi Chaluvadi
  - Query Processing for High-Volume XML Message Brokering, Yanlei Diao, Michael J. Franklin
  - Attribute Grammars for Scalable Query Processing on XML Streams, Christoph Koch and Stefanie Scherzinger
  - XPath Queries on Streaming Data, Feng Peng, Sudarshan S. Chawathe
  - An efficient single-pass query evaluator for XML data streams, Dan Olteanu Tim Furche, François Bry
Research topics (9)

• Graphical query languages
  – XQBE: A Graphical Interface for XQuery Engines, Daniele Braga, Alessandro Campi, Stefano Ceri

• Extensions to XQuery
  – Grouping in XML, Stelios Paparizos, Shurug Al-Khalifa, H. V. Jagadish, Laks V. S. Lakshmanan, Andrew Nierman, Divesh Srivastava and Yuqing Wu
  – Merge as a Lattice-Join of XML Documents, Kristin Tufte, David Maier.
  – Active XQuery, A. Campi, S. Ceri

• XML integrity constraints
  – Keys for XML, Peter Buneman, Susan Davidson, Wenfei Fan, Carmem Hara, Wang-Chiew Tan
  – Constraints for Semistructured Data and XML, Peter Buneman, Wenfei Fan, Jérôme Siméon, Scott Weinstein
Some DB research projects

• **Timber**
  – Univ. Michigan, At&T, Univ. British Columbia
  – http://www.eecs.umich.edu/db/timber/

• **Natix**
  – Univ. Manheim
  – http://www.dataexmachina.de/natix.html

• **XSM**
  – Univ. San Diego
  – http://www.db.ucsd.edu/Projects/XSM/xsm.htm

• **Niagara**
  – Univ. Madison, OGI
  – http://www.cs.wisc.edu/niagara/