The nuts and bolts of DBMS construction: building your own prototype

Alberto Lerner  Ioana Manolescu
New York University  INRIA Futurs
USA  France

Part 1: Goal and methodology
Motivation

• **www.amazon.com:**
  – 48 books on compiler construction, several on optimizing compilers;
  – 1 book on DBMS system implementation!!
• Researchers wanting to test new concepts are forced to build entire systems from scratch
• Students learning how to prototype a DBMS have a hard time finding resources
• Haven’t we made construction of DBMS any easier in the last 20 years?

Goal of the tutorial

• **To outline techniques** for building
  – advanced research experimental systems
  – simple but useful query engines
• **To map resources** providing
  – Building blocks, or
  – Knowledge on how to build a building block if there is not a satisfactory one at hand
Method followed in the sequel

- Outline a realistic architecture
  - Identify components and interfaces
- Study the interfaces and define ways to combine the components
- Use the above components as building-blocks for simple but realistic case studies

DBMS architecture

- User
  - Query TEXT
  - Internal query form
  - Query execution plan
- Execution engine
  - Data storage
    - Data, indices
    - Materialized views
Recommended prototype design methodology

1. What is the component of interest (the one you want to study / implement / test) ?
2. Where would it fit into a functional system ? (What modules would it interact with ?)
3. Which of these interactions are interesting for you ?
4. How to get (satisfactory approximations of) the modules interacting with yours ?

Identifying your target

- Already a good step in clarifying ideas
- A new component:
  - A new XML indexing technique
  - A new (physical) join operator
  - A new query plan enumeration algorithm
- Sometimes a whole system is sought (framework rather than component)
  - A generic framework for query optimization
  - A framework for distributed XML management
  - An XML benchmark
Identifying your target

• Component vs. framework
  – software engineering issue, with a DB twist

Identifying module interactions

• Start by considering an "ideal, complete" system architecture

• Your component interacts with others
  – XML index: storage manager, execution engine, query optimizer…
  – Join algorithm: execution engine, optimizer..
  – QEP enumerator: parser, storage and statistics descriptions…
Identifying relevant module interactions

- Maybe the trickiest step
- Two ways out
  - Do it seriously:
    build, or find, a real-life (complex, tested, documented) system, and build on it.
  - Intelligent pruning:
    assume away a number of features, simulate or ignore them
    - Good enough for a paper
    - If you assume away too much, not even good for a paper

Module interactions for an XML indexing technique

- **MUST CONSIDER**
  - a real, disk-based, persistent storage
  - an execution model
  - notions of an algebra

- **MAY IGNORE**
  - query parsing (query rewriting, view unfolding…)
  - transaction management (index locking ?)
Module interactions for a new join algorithm

• **MUST CONSIDER**
  – A very precise execution model
  – Depending on the algorithm, memory management, or persistent storage of intermediary results
  – Notions of the optimizer

• **MAY IGNORE**
  – Storage management, query parsing, optimization details

Module interactions for a QEP enumerator

• **MUST CONSIDER**
  – Query language expressive power
  – Storage description language / generality
  – Algebra

• **MAY IGNORE** / abstract away
  – Storage manager
  – Query execution engine
  – Cost model
  (once characterized by assumptions)
Getting the right components

• Finding existing components
  – In general: public domain tools with user groups tend to be solid (and sometimes complex)
  – Alternatively: academic prototypes with open source code are usually easy to understand (but not always work)

• Devising new ones
  – In general: text books + lots of programming
  – Alternatively: implementing algorithms suggested on research literature

Part 2: A look into DBMS internals
Query Parsing and Analyzing

1. Scan – identify basic language elements (tokens)
2. Parse – identify language structures
3. Analyze – check whether identifiers refer to existing objects, whether expressions are sound, and whether they are used correctly
4. Generate intermediate representation of the query

Tokenizing

```
select salary*12
from employee
where depto="INF"
```

```
SELECT | IDENTIFIER | TIMES | LITERAL | FROM | IDENTIFIER | WHERE | IDENTIFIER |_EQUALS | LITERAL
select | salary | * | 12 | from | employee | where | depto | = | "INF"
```
Parsing

Expression

SELECT | IDENTIFIER | TIMES | LITERAL | FROM | IDENTIFIER | WHERE | IDENTIFIER | EQUALS | LITERAL

select | salary | * | 12 | from | employee | where | depto | 1<INF> | “INF”

From Clause

List of Tables

Select Statement

Where Clause

Search Condition

Analyzing (Type Checking)

identifier exist?
can salary be multiplied by
an integer?

tables/views exist?

expression is
bool?

Expression

TIMES

SELECT | IDENTIFIER | TIMES | LITERAL | FROM | IDENTIFIER | WHERE | IDENTIFIER | EQUALS | LITERAL

select | salary | * | 12 | from | employee | where | depto | 2<INF> | “INF”
Intermediate Representation

Annotated Syntax Tree

Logical Operator Tree

Query Execution

1. Plan Generation - transform the intermediate representation into an executable plan
2. Plan Execution – carry on the steps that the plan determines
Plan Generation

```
proj::proj(Operator op, List exprs)
Tuple proj::get_next() {
    /* get next tuple */
    /* compute every expression in exprs */
    /* return result */
}

scan::scan(Relation rel, Predicate p, Attributes att)
Tuple scan::get_next() {
    /* issue getpage() to the disk manager */
    /* return one tuple at a time */
}
```

Buffer and Storage Manager

```
getpage()
lookup()
page | slot
E1   | 8
E2   | 5
...  | ...

getfreepage()
lst of empty slots
0,1,2,3,4,6,...
```
Plan Execution

More Complex Queries

- The order of operators in an optimal plan may not resemble that of the query text
- Several possibilities of Query Optimization
  - Choosing more efficient access paths
  - Reducing the size of intermediate results
  - Choosing among several algorithms for a given operator
Query Optimizing

1. Internal representation now stores a graph of the relations involved
2. Optimization tries to order the operations in the graph in the best possible way
3. The best algorithms are chosen for each operation

Query Graphs

```
select ... from r,s,t
where r.a = s.a
      and s.b = t.b
      and t.c = r.c
```
Dynamic Programming Enumeration

- Start by choosing the best way to access each relation
- Then the best way to join every pair of relation
- Then the best way to join each pair with the third relation
- At each step, keep only the best performing alternative (and those that have “interesting orders”)

Dynamic Programming Enumeration

<table>
<thead>
<tr>
<th>Best way to join with third relation</th>
<th>r</th>
<th>m</th>
<th>s</th>
<th>m</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best way to join two relations</td>
<td>r</td>
<td>m</td>
<td>s</td>
<td>r</td>
<td>m</td>
</tr>
<tr>
<td>Best access path to a single relation</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transformation-Based Enumeration

- A transformation is a triple \(<\text{pattern}, \text{condition}, \text{substitution}>\)
- \text{Pattern} is a sub-tree with wildcards to be matched to the plan
- Once a pattern is bound to the plan, the \text{condition} is tested
- If the condition passed the \text{substitution} is done
- Repeat the cycle
Cost Model

- Estimates work and cardinality of the results for each operation
- Most difficult aspect of optimization (tens of thousands of code lines in commercial products)
- Very prone to errors because of estimates made over estimates

Part III: Case Studies
(Please, do try this at home!)
What if I wanted to…

Complex Components
Re-engineer
(Localized Impact)

Adaptable Components
Re-use

Number of components involved

(Re-)develop

(Widespread Impact)

Case 1

- Problem - LRU discipline may give poor cache hits
- Solution - Replace LRU discipline by MRU
- Implementation - Localized change in existing component of PostgreSQL
DBMS architecture

- User
  - Query TEXT
  - Query execution plan
    - Internal query form
      - Analyzer
        - Catalog schema information
        - Data and execution statistics
        - Optimizer
          - Data storage
            - Data, indices
            - Materialized views

Execution engine
Memory buffer

Problem

- Relation R being scanned

BUFFER MANAGER

DISK

list of available pages
0, 1, 2

R0 R1 S0 S1 S2
Problem

- Relation R being scanned

BUFFER MANAGER

DISK

list of available pages

LRU
Problem

- Relation R being scanned

BUFFER MANAGER

DISK

list of available pages

2, 0

R0 R1

S0 S1 S2

Problem

- Relation R being scanned

BUFFER MANAGER

DISK

list of available pages

2, 0, 1

R0 R1

S0 S1 S2
Problem

- Relation $S$ having statistics updated

BUFFER MANAGER

DISK

list of available pages $\{0, 1\}$

Problem

- Relation $S$ having statistics updated

BUFFER MANAGER

DISK

list of available pages $\{0, 1, 2\}$
Problem

- Relation S scan steals R’s pages! What if R is accessed again?

Most Recently Used Page Replacement

- If S won’t likely be scanned again, why is it stealing pages from R?
- Utilities such as statistics gathering could use MRU replacement
- In some cases big repeated scans can do too, so to avoid buffer flooding
Statistics using MRU

- When page S0 gets unpinned, the buffer #2 will be put in front of the available list.
Implementation Ingredients

• PostgreSQL
  – Open source database
  – Big user (and development) community
  – Complete implementation:
    • Fairly complete SQL 92
    • Concurrency
    • Recovery
    • Transactions

PostgreSQL Resources

• Download Sources
  – www.postgresql.org
• Development Documentation
  – developer.postgresql.org
• Discussion List for Developers
  – psql-hackers
**PostgreSQL Buffer Manager**

- **FreeListDescriptor**
- **Buffer Descriptors**
- **Shared Free List**
- **First Available Buffer**
- **End of Available List**
- **Actual Space for Disk Pages**

---

**Adding a Buffer**

```c
static void AddBufferToFreelist(BufferDesc *bf) {
    /* change bf so it points to inFrontOfNew and its successor */
    bf->freePrev = SharedFreeList->freePrev;
    bf->freeNext = Free_List_Descriptor;

    /* insert new into chain */
    BufferDescriptors[bf->freeNext].freePrev = bf->buf;
    BufferDescriptors[bf->freePrev].freeNext = bf->buf;
}
```

/src/backend/storage/buffer/freelist.c
Adding a Buffer

static void AddBufferToFreelist(BufferDesc *bf)
{
    /* change bf so it points to inFrontOfNew and its successor */
    bf->freePrev = SharedFreeList->freePrev;
    bf->freeNext = Free_List_Descriptor;

    /* insert new into chain */
    BufferDescriptors[bf->freeNext].freePrev = bf->buf;
    BufferDescriptors[bf->freePrev].freeNext = bf->buf;
}
Adding a Buffer

```c
static void AddBufferToFreelist(BufferDesc *bf) {
    /* change bf so it points to inFrontOfNew and its successor */
    bf->freePrev = SharedFreeList->freePrev;
    bf->freeNext = Free_List_Descriptor;
    /* insert new into chain */
    BufferDescriptors[bf->freeNext].freePrev = bf->buf;
    BufferDescriptors[bf->freePrev].freeNext = bf->buf;
}
```
Implementing MRU

FreeListDescriptor

Shared Free List

Buffer Descriptors

Insert Here

Actual Space for Disk Pages

Rather Than Here

DEMO
Case 2

- Problem – build a simple **persistent XML storage system**
- Solution: use an embedded database library and customize storage structures

Simple XML storage system

- **User**
  - Query TEXT
  - Internal query form
  - Result

- **Execution engine**
  - Query execution plan

- **Data storage**
  - Data, indices
  - Materialized views

- **XML data**
  - XML loader

- **Catalog schema information**
- **Data and execution statistics**
XML documents are data trees

Snippet of XML data (auction site) generated for the XMark benchmark:

```
<site>
  <samerica>
    <item id="item6">
      <location>UnitedStates</location>  <quantity>1</quantity>
      <name>chair</name>
      <payment>Creditcard, Cash</payment>
      <description>nice</description>
      <keyword>wooden</keyword>
      <window>See description for charges</window>
      <incategory category="category0" />
      <incategory category="category1" />
      <mailbox />
    </item>
  </samerica>
</site>
```
Simple data and storage model for XML

1. Add unique identifiers to all XML nodes

```
<site>
  <samerica>
    <item id="item6">
      <location>UnitedStates</location>
      <quantity>1</quantity>
      <name>chair</name>
      <payment>Creditcard, Cash</payment>
      <description>nice wooden chair</description>
      <shipping>See description for charges</shipping>
      <incategory category="category0"/>
      <incategory category="category1"/>
    </item>
  </samerica>
</site>
```

2. Separate tree structure (edges) from tree values (leaves)

<table>
<thead>
<tr>
<th>Edge(pID, tag, cID, attr)</th>
<th>Value(id, val)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- site</td>
<td>3 &quot;item6&quot;</td>
</tr>
<tr>
<td>1 samerica</td>
<td>5 &quot;United States&quot;</td>
</tr>
<tr>
<td>2 item</td>
<td>6 &quot;1&quot;</td>
</tr>
<tr>
<td>3 id</td>
<td>9 &quot;nice&quot; &quot;chair&quot;</td>
</tr>
<tr>
<td>4 X</td>
<td>10 &quot;wooden&quot;</td>
</tr>
<tr>
<td>2 name</td>
<td></td>
</tr>
</tbody>
</table>

63

64
Simple data and storage model for XML

3. Many variants exist

Simple variant: partition by tag

<table>
<thead>
<tr>
<th>Edge(pID, tag, cID, attr)</th>
<th>Value(id, val)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- site</td>
<td>3 &quot;item6&quot;</td>
</tr>
<tr>
<td>1 samerica</td>
<td>5 &quot;United States&quot;</td>
</tr>
<tr>
<td>2 item</td>
<td>6 &quot;1&quot;</td>
</tr>
<tr>
<td>3 id</td>
<td>9 &quot;nice&quot; &quot;chair&quot;</td>
</tr>
<tr>
<td>2 name</td>
<td>10 &quot;wooden&quot;</td>
</tr>
<tr>
<td>9 description</td>
<td></td>
</tr>
<tr>
<td>9 keyword</td>
<td></td>
</tr>
</tbody>
</table>

A small step forward: indexing elements by their content

Index `<item>` elements by their @id value
Implementation of a simple storage system for XML

• We need
  – An XML parser
    1. Give unique IDs to XML nodes
    2. Separate structure from content
  – A storage for
    • One edge sequence per tag
    • ID-value pairs
  – Indices (e.g., B+-trees)

Implementation of a simple storage system for XML

• We need
  – An XML parser org.apache.xerces
    1. Give unique IDs to XML nodes you do it
    2. Separate structure from content you do it
  – A storage BerkeleyDB www.sleepycat.com
    • One edge sequence per tag use BerkeleyDB
    • ID-value pairs use BerkeleyDB
  – Indices (e.g., B+-trees) use BerkeleyDB
    B+-trees
Parsing and loading XML documents

```java
import org.apache.xerces.parsers.*;
public class myParser extends DefaultHandler implements Parser {
    public void startElement(String nsURI, String tag,
        String qName, Attributes attrs)
    {
        // give it a number (unique ID), push on the stack
        // record the attributes
        // record the edge connecting to its father
    }

    public void characters(char[] buffer, int start, int end){
        // record the characters as composing a value
        // record the edge connecting to the stack top
    }

    public void endElement(String nsURI, String IName, String qName) {
        // pop the stack
    }
}
```
Loading XML data and structure into a persistent storage

- BerkeleyDB: embedded storage library
- Offers Java/C++ API for storing items in databases by keys
  - An item is a sequence of bytes
  - A key is a sequence of bytes
  - A database is a collection of (key, item) pairs
- Four database types:
  - B+ tree
  - Hashtable
  - Queue (fixed length)
  - RecordNumber (var. length)

Loading XML data and structure into a persistent storage

- Four Berkeley DB database types:
  - B+ tree
  - Hashtable
  - Queue (fixed length)
  - RecordNumber (var. length)
- What do we need?
  - Queue databases for edge sequences
    - One for edges starting in /site/samerica/item elements
  - RecordNumber databases for values
    - One RecordNumber database for /site/samerica/item/@id values
  - B+ tree indices on item IDs by item/@id value
    - One B+ tree database with (item/@id, item ID) pairs
Loading element identifiers into BerkeleyDB

```java
import com.sleepycat.db.*;

Db myDb = new Db();
myDb.open(null, "dataFile0", "item", Db.DB_QUEUE);
// transaction, data file name, database name, database type
Db.DB_WRITE|Db.DB_CREATE, 644, 4);
// operation flags, access rights, item length

IntDb myID = new IntDb(elementID);
// IntDb extends Db (wrapper BDB class for items and keys)
IntDb myKey = new IntDb(crt);
myDb.put(myKey, myID);
```

Case 3

- Problem – AQuery language works with array data types and array expressions that need to be checked
- Solution – Create type checker for the language
- Implementation - Use parser generation tools and integrate type checker
Motivational Query

- Create a log of flow information. A flow from src to dest ends after a 2-minutes silence.
- Grouping by source and destination alone does not do the job.
### A Query in Pictures

#### Packets

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
<th>len</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$s_2$</td>
<td>250</td>
<td>1</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_2$</td>
<td>270</td>
<td>20</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_1$</td>
<td>330</td>
<td>47</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_2$</td>
<td>235</td>
<td>141</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_1$</td>
<td>280</td>
<td>150</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_1$</td>
<td>305</td>
<td>155</td>
</tr>
</tbody>
</table>

```sql
SELECT src, dst, count(*), avg(len)
FROM Packets
ASSUMING ORDER src, dst, time
GROUP BY src, dst, sums (deltas(time) > 120)
```
AQuery in Pictures

<table>
<thead>
<tr>
<th>Packets</th>
<th>src</th>
<th>dst</th>
<th>len</th>
<th>time</th>
<th>c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>s2</td>
<td>250</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
<td>270</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
<td>235</td>
<td>141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>s1</td>
<td>330</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>s1</td>
<td>280</td>
<td>150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>s1</td>
<td>305</td>
<td>155</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SELECT src, dst, count(*), avg(len)
FROM Packets
ASSUMING ORDER src, dst, time
GROUP BY src, dst, sums (deltas(time) > 120)
### A Query in Pictures

<table>
<thead>
<tr>
<th>Packets</th>
<th>src</th>
<th>dst</th>
<th>len</th>
<th>time</th>
<th>c1</th>
<th>c2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s1</td>
<td>s2</td>
<td>250</td>
<td>1</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>s1</td>
<td>s2</td>
<td>270</td>
<td>20</td>
<td>F</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>s1</td>
<td>s2</td>
<td>235</td>
<td>141</td>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td>s1</td>
<td>330</td>
<td>47</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td>s1</td>
<td>280</td>
<td>150</td>
<td>F</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td>s1</td>
<td>305</td>
<td>155</td>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

**SELECT** `src, dst, count(*), avg(len)`  
**FROM** `Packets`  
**ASSUMING** `ORDER src, dst, time`  
**GROUP BY** `src, dst, sums (`deltas(time)` > 120)`

---

### A Query in Pictures

<table>
<thead>
<tr>
<th>Packets</th>
<th>src</th>
<th>dst</th>
<th>len</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s1</td>
<td>s2</td>
<td>250</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>s1</td>
<td>s2</td>
<td>270</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>s1</td>
<td>s2</td>
<td>235</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td>s1</td>
<td>330</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td>s1</td>
<td>280</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td>s2</td>
<td>s1</td>
<td>305</td>
<td>155</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
<th>[250,270]</th>
<th>[235]</th>
<th>[141]</th>
</tr>
</thead>
<tbody>
<tr>
<td>s2</td>
<td>s1</td>
<td>[330,280,305]</td>
<td>[47,150,155]</td>
<td></td>
</tr>
</tbody>
</table>

**SELECT** `src, dst, count(*), avg(len)`  
**FROM** `Packets`  
**ASSUMING** `ORDER src, dst, time`  
**GROUP BY** `src, dst, sums (`deltas(time)` > 120)`
A Query in Pictures

![Table of Packet Data]

Other Order Built-in Functions

- `Firstn(n,col), Lastn(n,col)`
  - Return a vector/array with the n first or n last elements

```sql
SELECT src, dest, avg(firstn(10,len))
FROM Packets
ASSUMING ORDER src, dst, time
GROUP BY src, dst, sums (deltas(time) > 120)
```

- Non-size preserving functions may form invalid expressions
Type-checking Problem

```sql
SELECT src, dst, count(*), avg(len)
FROM Packets
ASSUMING ORDER src, dst, time
GROUP BY src, dst, sums (deltas(time) > 120)
```

- How to make sure that expressions such as the one in the group clause are valid?
  - Checking the type of ‘time’ is just the tip of the iceberg… checking size of vectors is also necessary.
  - Can sums() be applied to a result of deltas()?
  - Can sums() be applied to a result of firstn()?
  - Can those two expressions participate in a group by?

Considering cardinality and dimension

- Type is actually a 3-tuple `<cardinality, dimension, type>`
  - Type of column ‘time’ is `<original-card, 1, integer>` meaning that time did not suffer cardinality reduction, and that it is a vector (dimension 1) of integers
  - Type of `deltas(<original-card, 1, integer>)` is `<original-card, 1, integer>` meaning that the function is size-preserving and that it returns a vector of integers
  - Type of `firstn(10, <original-card, 1, integer>)` is `<10, 1, integer>`

- Group by requires an expression that is `<original-card, ?, ?>` meaning that to every row of the table a value of group must be assigned
Implementation Ingredients

- Scanner Generator
  - Flex
    - Given a collection of regular expressions, it generates code to tokenize any text accordingly
    - Very sophisticated, yet very simple to use
- Parser Generator
  - LLGEN (www.cs.vu.nl/~cerei/LLgen.html)
    - Given a LL(1) grammar generates a recursive descent parser
    - Could have used yacc or bison (LALR parser generators) instead

More Ingredients

- K (www.kx.com)
  - APL descendant very used in Wall Street
  - Very robust language for manipulating vectors
  - Very terse syntax
  - Somewhat long learning curve
  - Then why bother using???
  - Productivity, productivity, productivity
AQuery Scanner Definitions

...  
AND { return AQ_AND; }  
AS { return AQ_AS; }  
ASC { return AQ_ASC; }  
ASSUMING { return AQ.Assuming; }  
ASSUMING{spc}ORDER { return AQ.Assuming; }  
BETWEEN { return AQ.BETWEEN; }  
DESC { return AQ.DESC; }  
DISTINCT { return AQ.DISTINCT; }  
EACH { return AQ.EACH; }  
FALSE { return AQ.FALSE; }  
FLATTEN { return AQ.FLATTEN; }  
FROM { return AQ.FROM; }  
GROUP{spc}BY { return AQ.GROUPBY; }  
...

AQuery Parser Definitions

query_specification : { setctx(initblk()); }  
AQ_SELECT  
{ setctx(SCTXSELECT); }  
[ AQ.DISTINCT ]?  
select_list  
table_expression  
[ AQ.ROC ]?  
;  
select_list : star_or_derived_column  
select_list_tail  
;  
...  
derived_column : value_expression  
{ appexpr( POP, getctx(), getisctx()); }  
;
Demo

Case 4

- Problem – build a simple XML querying system
- Solution: use structural identifiers and twig joins
- Implementation – Develop join algorithm following classical interfaces
Simple XML querying system

- Simple XML storage
  - The idea (already seen + revisited)
  - The implementation
- Performing XML tree traversals
  - The idea
  - The implementation
XML querying requires tree traversal

- For $i$ in site/regions/samerica/item
  $k$ in $i$/keyword
  return <artigo>
    <numero>$i/@id/value()</numero>
    <keyword> $k/text() </numero>
  </artigo>

- Path traversals:
  - Site/regions/samerica/item
  - Site/regions/samerica/item/@id
  - Site/regions/samerica/item//keyword
Tree traversal requires joins

- We want:
  all elements named "item"
  whose parents have the tag "samerica"
  whose parents have the tag "site"
- Select e3.cID
  from Edge_site e1, Edge_samerica e2, Edge_item e3
  where e1.cID=e2.pID and e2.cID=e3.cID

Implementing tree traversals

- site/samerica/item:
  - Edge_site ⊲ Edge_samerica ⊲ Edge_item
    – 3-way join…

- site/samerica/item/keyword/description
  - Edge_site ⊲ Edge_samerica ⊲ Edge_item ⊲ Edge_description ⊲ Edge_keyword
    – 5-way join……
What XML documents really look like

- A bigger fragment of an XMark-generated document (a very small one) on the next slide

- Real XMark generated documents also have
  - Asia, Australia, Europe, N America, Africa
  - /site/people/person...
  - /site/catgraph/
  - /site/open_auctions,
    /site/closed_auctions...
Implementing real tree traversals

- Item//keyword requires in practice a union of
  - 7-way join
  - 8-way join
  - 10-way join
- A good moment for changing the strategy.
Solution: better unique identifiers for XML nodes

Structural identifiers (beginTagNo, endTagNo, depth)

- Structural identifiers and structural relationships
  - Element \( x \) is an ancestor of element \( y \) iff:
    - \( x.\text{begin} < y.\text{begin} \) and \( x.\text{end} > y.\text{end} \)
  - Example:
    - Item \([3\ 3\ 11]\) is an ancestor of location \([4\ 4\ 1]\)
    - Location \([4\ 4\ 1]\) is not an ancestor of keyword \([9\ 5\ 5]\)
  - Element \( x \) is the parent of element \( y \) iff:
    - \( x.\text{begin} < y.\text{begin} \) and \( x.\text{end} > y.\text{end} \) and \( x.\text{depth} + 1 = y.\text{depth} \)
  - Example:
    - Item \([3\ 3\ 11]\) is the parent of location \([4\ 4\ 1]\)
The interest of structural joins (1/2)

- One join establishes ancestor-descendent connections of arbitrary length
  - No need to store edges any longer, just IDs
- site/regions/samerica/item:
  ID$_{site}$ $\Rightarrow$ ID$_{site/regions/samerica/item}$
The interest of structural joins (2/2)

- One join establishes ancestor-descendent connections of arbitrary length
  - No need to store edges any longer, just IDs
- site/regions/samerica/item:
  - IDsite IDsite/regions/samerica/item
- site/regions/samerica/item//keyword:
  - IDsite/regions.samerica/item (IDpathKeyword1 U IDpathkeyword2 U IDpathKeyword3)

Back to implementation

- Changing the data storage to use structural identifiers
  - Loader
  - Storage
- A simple execution engine for XML queries
  - PathAccess operator
  - Structural join
  - Value scan operator
Parsing and loading, revisited with structural identifiers

```java
import org.apache.xerces.parsers.*;
public class myParser extends DefaultHandler implements Parser {
    int depth;
    int endTagNo;
    XMLReader saxP = new SAXParser();
    saxP.setContentHandler(this);
    saxP.parse(XMLDocName);

    public void startElement(String nsURI, String tag, String qName, Attributes attrs) {
        // give it a number [start, depth, end], push start
        // record the attributes; update depth
    }
    public void characters(char[] buffer, int start, int end) {
        // same as before
    }
    public void endElement(String nsURI, String IName, String qName) {
        // same as before; update depth
    }
```
Loading XML data and structure, revisited with structural identifiers

- Four Berkeley DB database types:
  - B+ tree
  - Hashtable
- Queue (fixed length)
- RecordNumber (var. length)

- What do we need?
  - Queue databases for ID sequences
    - One for IDs of /site/samerica/item elements
  - RecordNumber databases for values
    - One RecordNumber database for /site/samerica/item/@id values
  - B+ tree indices on item IDs by item/@id value
    - One B+ tree database with (item/@id, item ID) pairs

Implementing a simple execution engine

- Most frequently used model: iterators
  - Interface Iterator {
    public void init();
    public boolean hasNext();
    public Tuple getNext();
  }
- Model also used in java.util.Iterator, java.sql.ResultSet...
A three-operators execution engine

- **PathAccess(String path):**
  - Returns [ID] tuples, where ID is an element identifier
  - Tuples are sorted in increasing order of the start number
- **StructuralJoin(Iterator i1, Iterator i2, int col1, int col2)**
  - Joins the tuples from Iterator 1 and Iterator 2 whenever i1[col1] is an ancestor of i2[col2]
- **ValueAccess(String path)**
  - Returns [ID, value] tuples, where ID is an element identifier, and value is a text value within such an element

Implementing PathAccess

```java
public class PathAccess implements Operator{
    public PathAccess(String path) {
        Tuple currentTuple = null; coolean over = false; }
    public void init() {
        /* ** open the database of elem. identifiers on path */ }
    public boolean hasNext() {
        if (over) return false;
        try {
            currentTuple = db.get(..., Db.DB_NEXT, ...)
        } catch (Exception e) {
            over = true; return false; }
        return true; }
    public Tuple getNext() { return currentTuple; }
    public void close() { /* ** close the database */ }
}
```
Implementing StructuralJoin

- Problem: hash-based joins won't work (no join key)
- Solution: stack-based algo on sorted inputs

- Push successively all nodes from ancestor list
- Probe the stack with nodes from the descendent list
- Pop the stack when ancestors are "too old"
Implementing StructuralJoin

- Problem: hash-based joins won't work (no join key)
- Solution: stack-based algo on sorted inputs

- Push successively all nodes from ancestor list
- Probe the stack with nodes from the descendent list
- Pop the stack when ancestors are "too old"
Implementing StructuralJoin

- Problem: hash-based joins won't work (no join key)
- Solution: stack-based algo on sorted inputs

- Push successively all nodes from ancestor list
- Probe the stack with nodes from the descendent list
- Pop the stack when ancestors are "too old"
Implementing StructuralJoin

- Problem: hash-based joins won't work (no join key)
- Solution: stack-based algo on sorted inputs

- Push successively all nodes from ancestor list
- Probe the stack with nodes from the descendent list
- Pop the stack when ancestors are "too old"

Short demo

- Loading an XML document

- QEP for joining /site/regions/samerica/item Ids with all their descendent keywords

- QEP for the query
  
  for $i in site/regions/samerica/item, $k in $i//keyword
  return <artigo>
    <numero>$i/@id/value()</numero>
    <keyword> $k/text() </numero>
  </artigo>
Simple extensions: XML indexing

- Indexing `<items>` by their `@id`:
  - Storing index entries at parsing time
    - From `myParser.startElement(...)`
  - Implementing a new index access physical operator
    - Public class `indexAccess implements Iterator{... }`
- Any XML indexing technique can be implemented and tested in a similar way

Simple extensions: twig joins

- Twig joins: establish pattern relationships among several lists of node IDs in a single join step
- *Holistic twig joins*: improve the performance of twig joins by using B+ tree indices on ID lists
Complex extension: query optimizer

- First choose the language
  - XPath may be too little, or too much
  - Full XQuery may be a little complex
- Example:
  - for $b in document("auction.xml")/site/people/person[id="person0"]
    return $b/name/text()
- Nice exercise 😊

Conclusions from building our simple system

- A simple framework can be extended in many ways
- Unless your XML research is limited to <1 Mb documents, a persistent storage (even a simple one) is required in order to be plausible
  - Intelligent pruning goes a long way
Part IV: (A non-exhaustive) Resource Directory

Agenda

- Complete Public Source Systems
- Parser Generator Tools
- Storage Sub-systems
- Rapid Development Tools
- Generating or Obtaining Data
- Readings
Relational Systems

- PostgreSQL (www.postgresql.org)
- MySQL (www.mysql.org)
- Predator (www.cs.cornell.edu/database/predator)

XML Systems

- Galax (db.bell-labs.com/galax)
- XQEngine (www.fatdog.com)
- QizXOpen (www.xfra.net/qizxopen)
  - Storage is in files
- Rainbow (www.davis.wpi.edu)
Stream Systems

- TelegraphCQ
  (telegraph.cs.berkeley.edu)

Storage Sub-systems

- BerkeleyDB (www.sleepycat.com)
- Shore (www.cs.wisc.edu/shore/)
- For XML problems, commercial or free-source relational systems may be provide some solutions
Parsing Tools

- Flex
- Bison/Yacc
- LLgen
- JavaCC
- JJTree

Rapid Development

- SWIG and Perl/Python
- “Exotic” languages
  - ML
  - K
- GraphViz for tree displaying and debugging
  (www.research.att.com/tools/graphviz)
Data generators

- Data generator of the TPC benchmarks (www.tpc.org)
- The XMark XML generator (monetdb.cwi.nl/xml/index.html)
- The ToxGene XML generator (www.cs.toronto.edu/tox/toxgene/)

Data sources

- XML data sets at U. of Washington (www.cs.washington.edu/research/xmldatasets)
- Wharton Research Data Services (wrds.wharton.upenn.edu) finance data
- Click Stream Data (ask your web manager!)
Readings

- If I could read one article in … that would be …
- Books
- Tutorials

Query execution and optimization

- A. Silberschatz and H.Korth, "Principles of database systems", chapters on query execution and query optimization
  http://cs-www.cs.yale.edu/homes/avi/dbbook/
  – Do read the complete examples
  – Slides are freely available
- Transformation-based optimization framework
Adaptive (self-improving) query processing

- Entry point: survey


Distributed query processing

- Complete overview:
- Shorter, denser overview:
- Cost models for data integration frameworks
XML-on-relational

- D. DeHaan, D. Toman, M. Consens, T. Oszu: A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding, SIGMOD 2003

Conclusion

- Yes, we have made the task of building database systems easier!
- Several research groups or public-source efforts produced components that can be reused
- A prototype may be able to take advantage of existing code and focus on its specific problem
- Careful design and understanding of current DBMS architecture are important
- The more experiments (and code) you produce, the easier it is to understand how other people’s software works…