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

Alberto Lerner
New York University
USA

Ioana Manolescu
INRIA Futurs
France

1

Part 1: Goal and methodology

2

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?

3

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

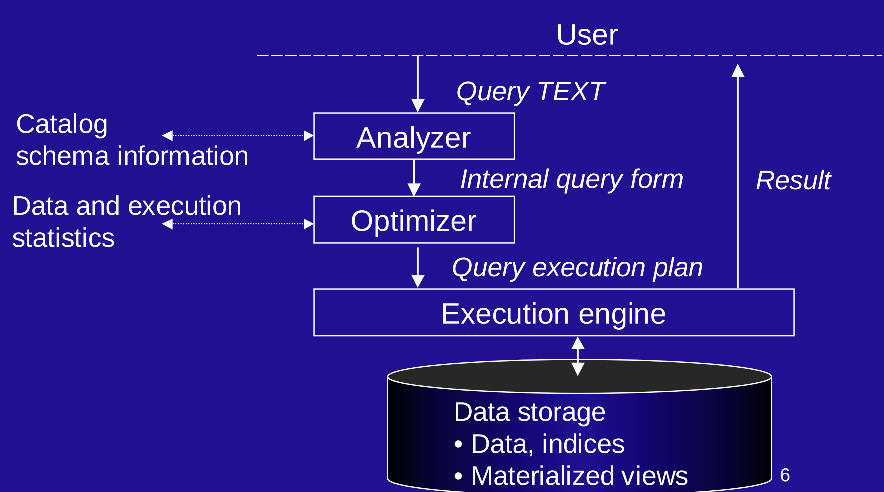
4

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

5

DBMS architecture



6

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 ?

7

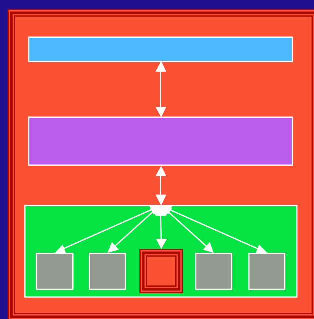
Identifying your target

- 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

8

Identifying your target

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



9

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…

10

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

11

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 ?)

12

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

13

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)

14

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

15

Part 2: A look into DBMS internals

16

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

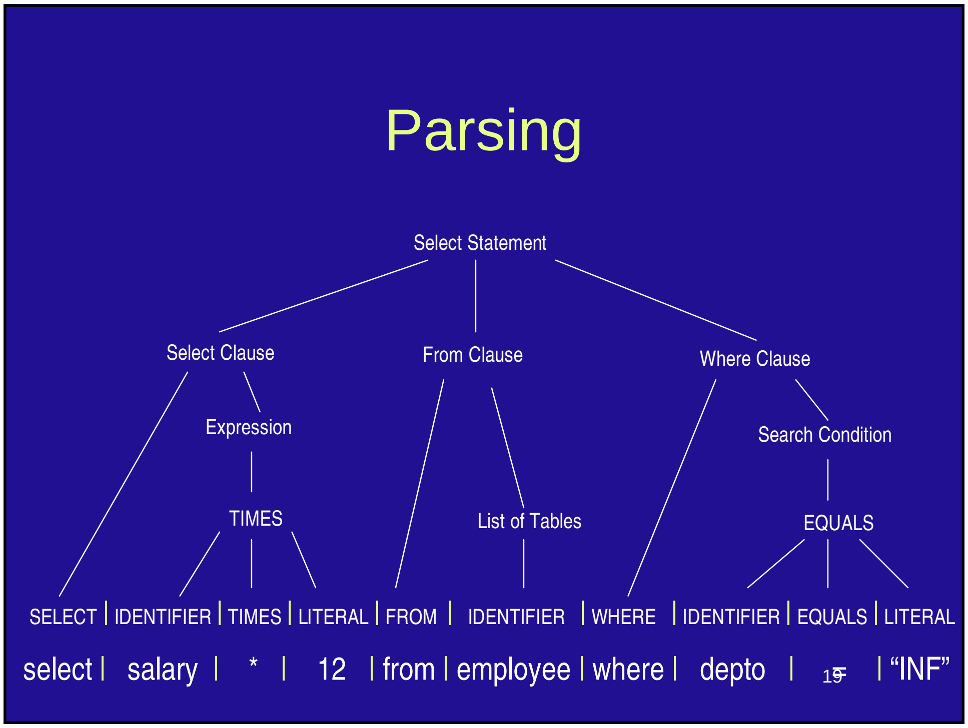
17

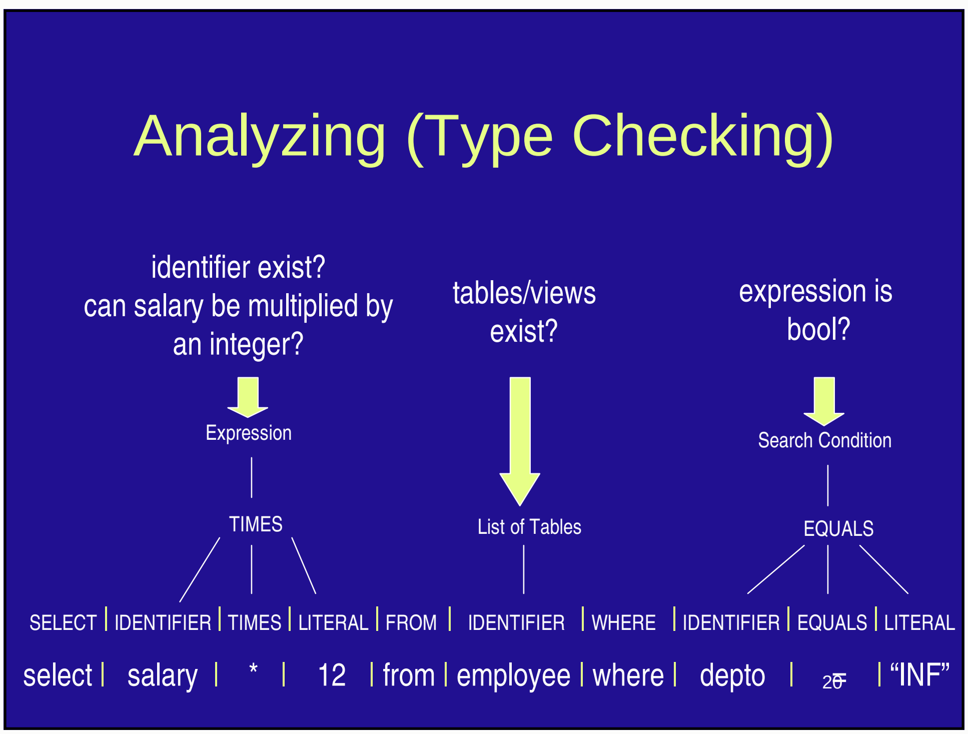
Tokenizing

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

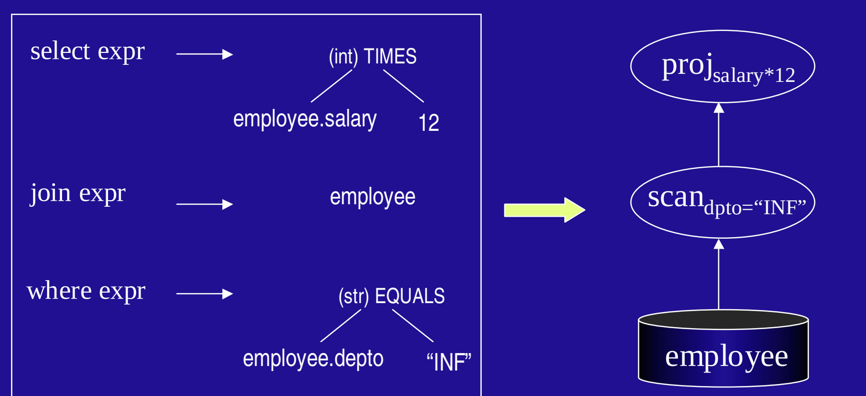
SELECT	IDENTIFIER	TIMES	LITERAL	FROM	IDENTIFIER	WHERE	IDENTIFIER	EQUALS	LITERAL
select	salary	*	12	from	employee	where	depto	=	"INF"

18


Parsing
Select Statement
Select Clause
From Clause
Where Clause
Expression
Search Condition
TIMES
List of Tables
EQUALS
SELECT | IDENTIFIER | TIMES | LITERAL | FROM | IDENTIFIER | WHERE | IDENTIFIER | EQUALS | LITERAL
select | salary | * | 12 | from | employee | where | depto | = | “INF”
19


Analyzing (Type Checking)
identifier exist?
can salary be multiplied by
an integer?
tables/views
exist?
expression is
bool?
Expression
Search Condition
TIMES
List of Tables
EQUALS
SELECT | IDENTIFIER | TIMES | LITERAL | FROM | IDENTIFIER | WHERE | IDENTIFIER | EQUALS | LITERAL
select | salary | * | 12 | from | employee | where | depto | = | “INF”
20

Intermediate Representation



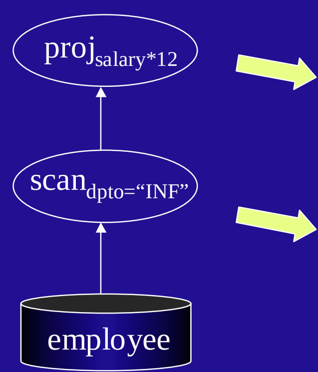
Annotated Syntax Tree

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

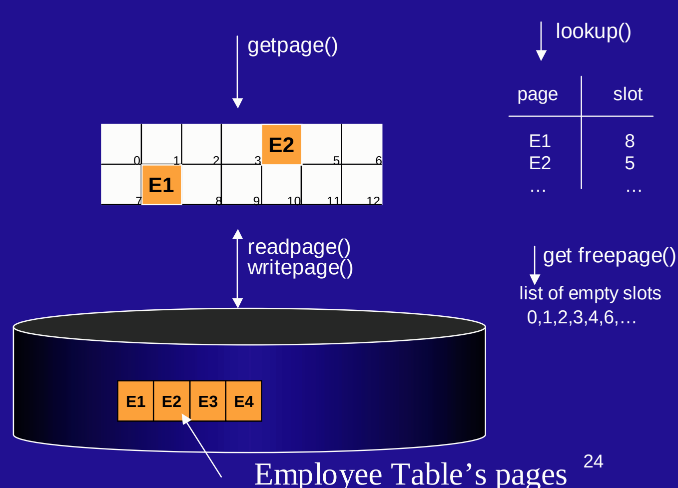



```
proj::proj(Operator op, List exprs)

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 */
}
```

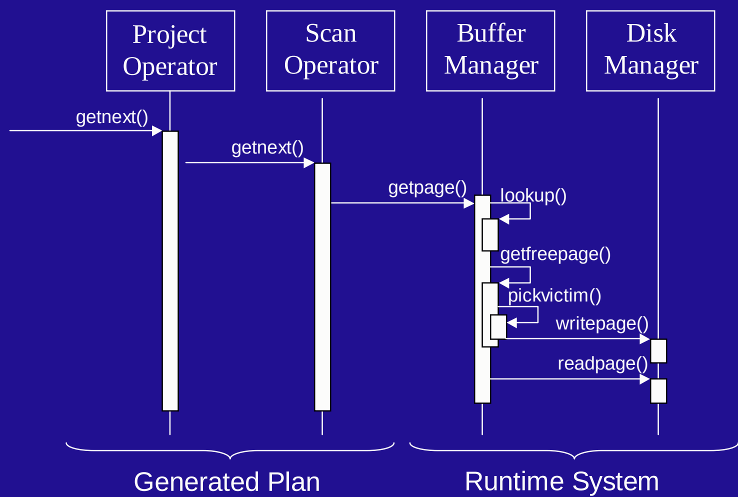
23

Buffer and Storage Manager



24

Plan Execution



25

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

26

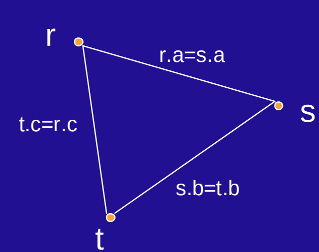
Query Optimizing

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

27

Query Graphs

```
select   …
from     r,s,t
where    r.a = s.a
         and s.b=t.b
         and t.c=r.c
```



28

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

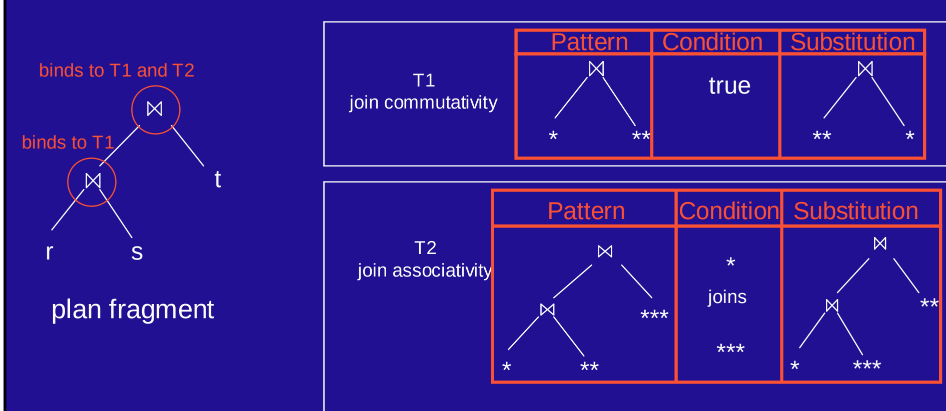
Best way to join with third relation	$r \bowtie s \bowtie t$	$r \bowtie t \bowtie s$	$s \bowtie t \bowtie r$
Best way to join two relations	$r \bowtie s$	$r \bowtie t$	$s \bowtie t$
Best access path to a single relation	r	s	t

Transformation-Based Enumeration

- A transformation is a triple <pattern, condition, substitution>
- Pattern is a sub-tree with wildcards to be matched to the plan
- Once a pattern is bound to the plan, the condition is tested
- If the condition passed the substitution is done
- Repeat the cycle

31

Transformation-Based Enumeration



32

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

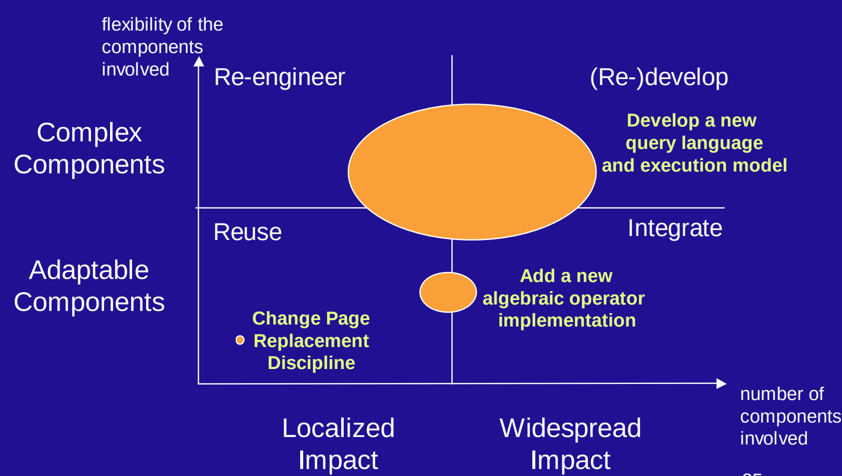
33

Part III: Case Studies

(Please, do try this at home!)

34

What if I wanted to…

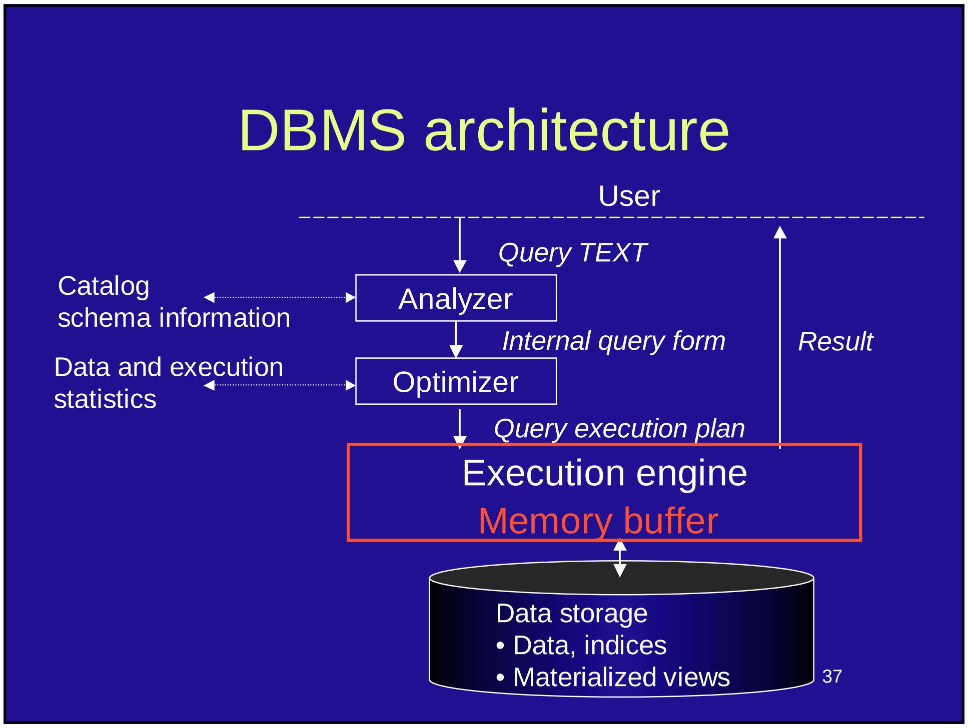


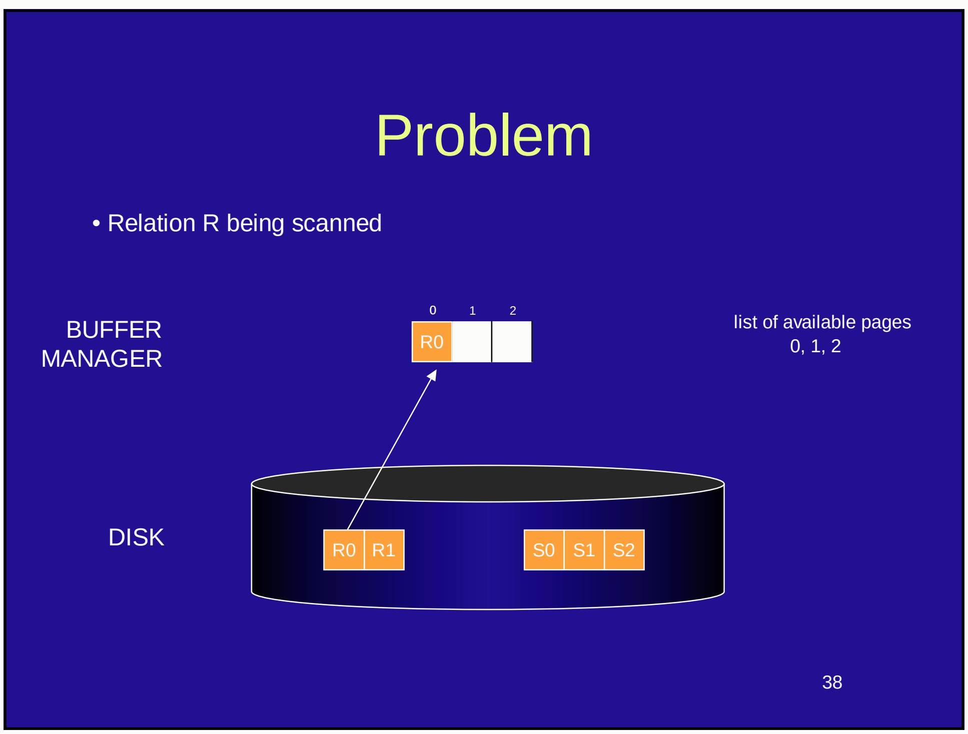
35

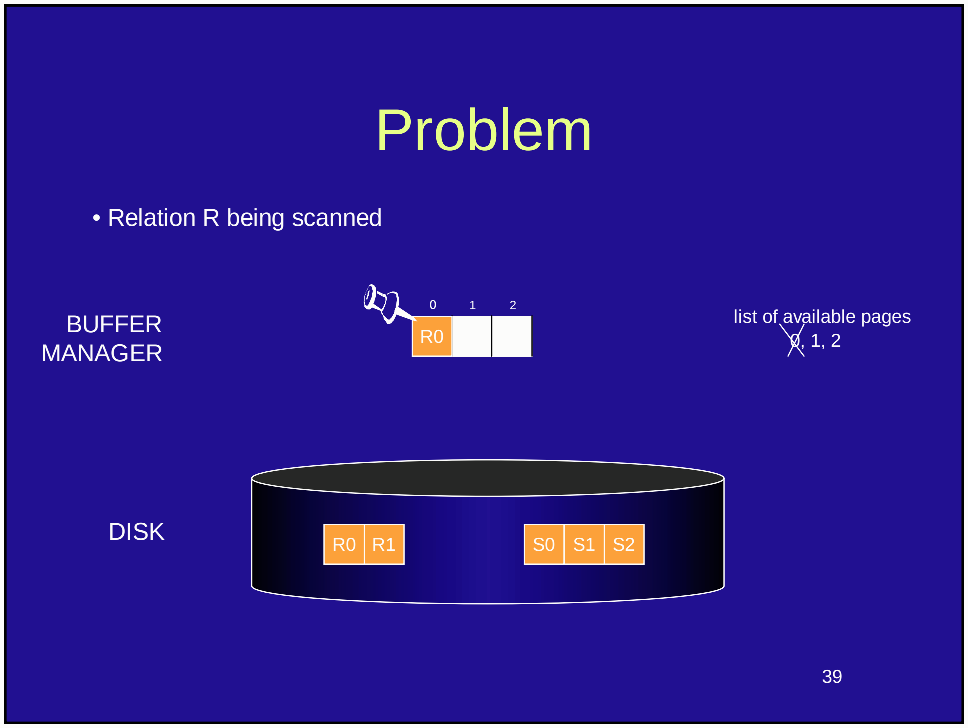
Case 1

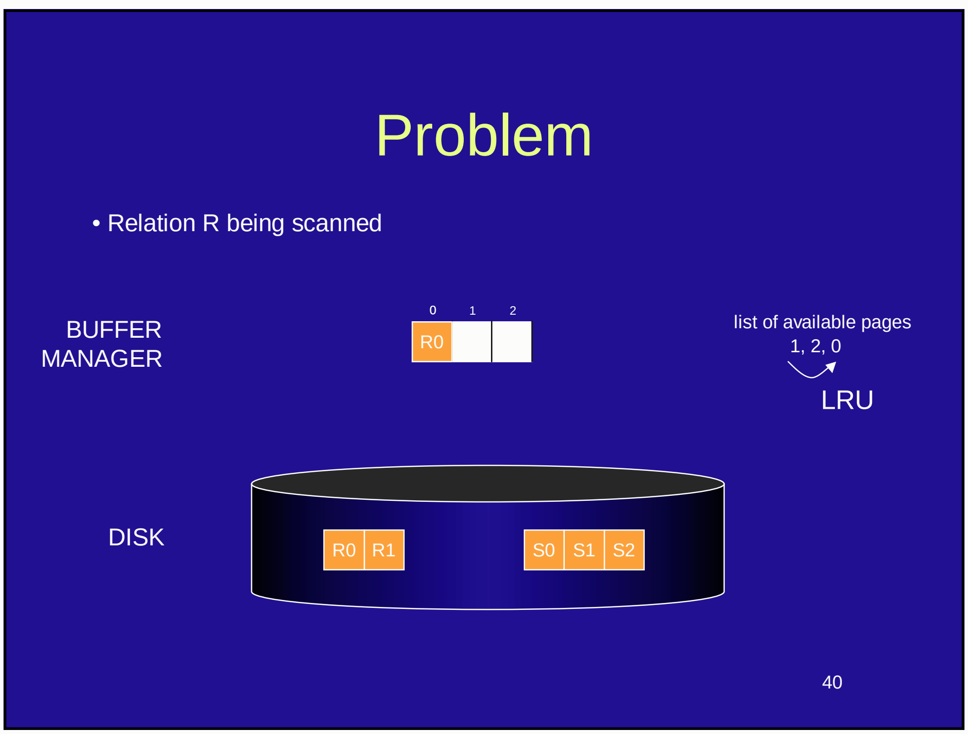
- Problem - LRU discipline may give poor cache hits
- Solution - Replace LRU discipline by MRU
- Implementation - Localized change in existing component of PostgreSQL

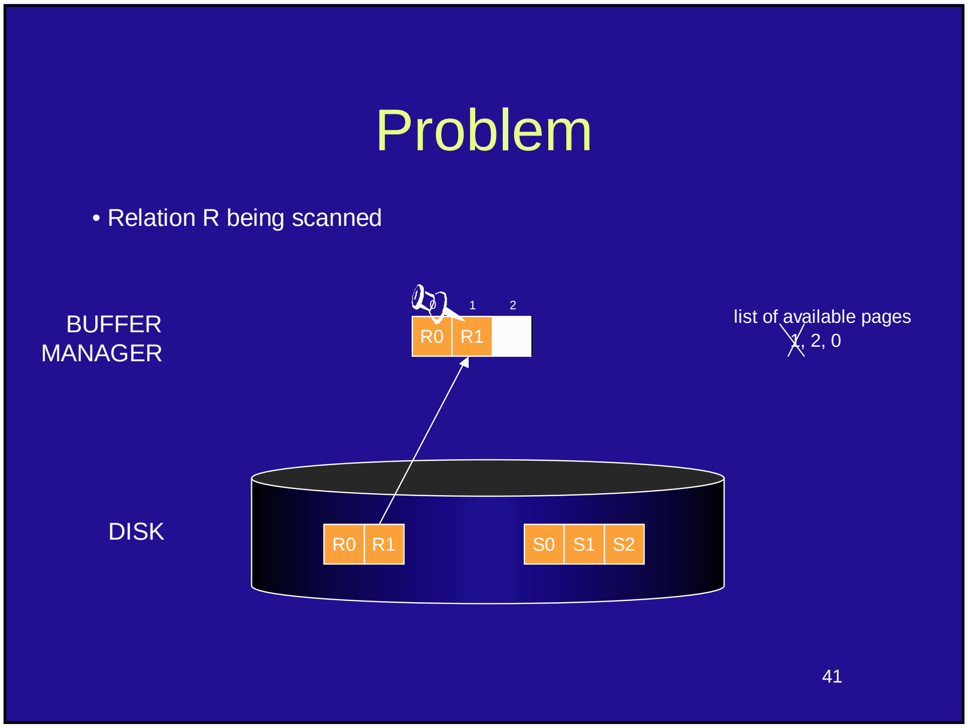
36

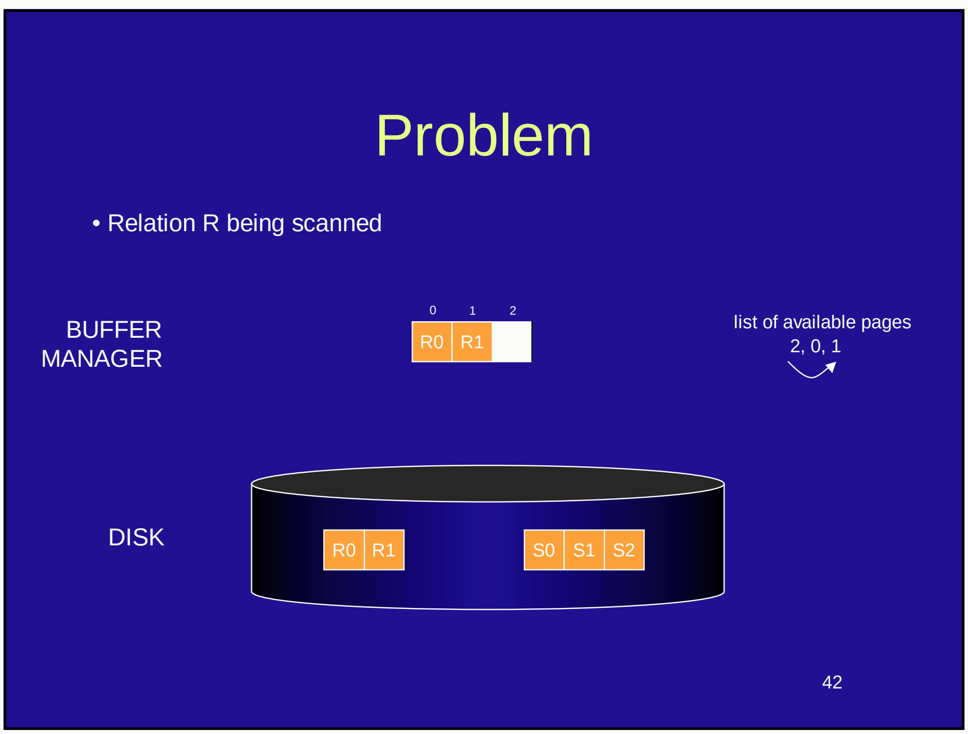

DBMS architecture
User
Query TEXT
Catalog
schema information
Analyzer
Internal query form
Result
Data and execution
statistics
Optimizer
Query execution plan
Execution engine
Memory buffer
Data storage
• Data, indices
• Materialized views
37

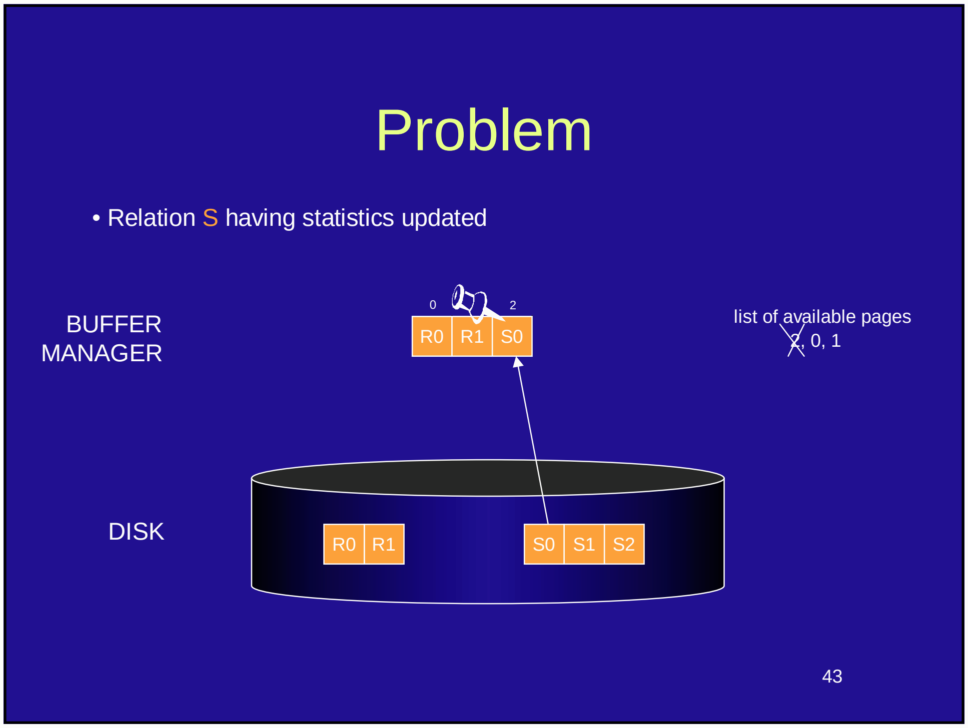

Problem
• Relation R being scanned
BUFFER
MANAGER
0
1
2
R0
list of available pages
0, 1, 2
DISK
R0
R1
S0
S1
S2
38

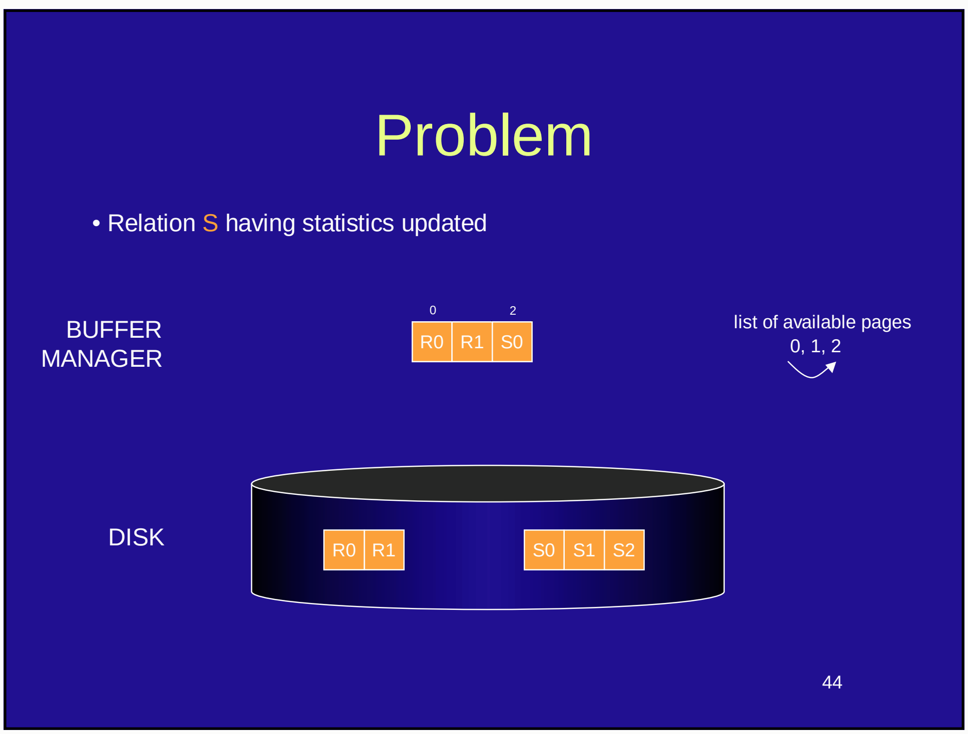

Problem
• Relation R being scanned
BUFFER MANAGER
0 1 2
R0
list of available pages
0, 1, 2
DISK
R0 R1
S0 S1 S2
39


Problem
• Relation R being scanned
BUFFER MANAGER
0 1 2
R0
list of available pages
1, 2, 0
LRU
DISK
R0 R1
S0 S1 S2
40


Problem
• Relation R being scanned
BUFFER MANAGER
0 1 2
R0 R1
list of available pages
1, 2, 0
DISK
R0 R1
S0 S1 S2
41

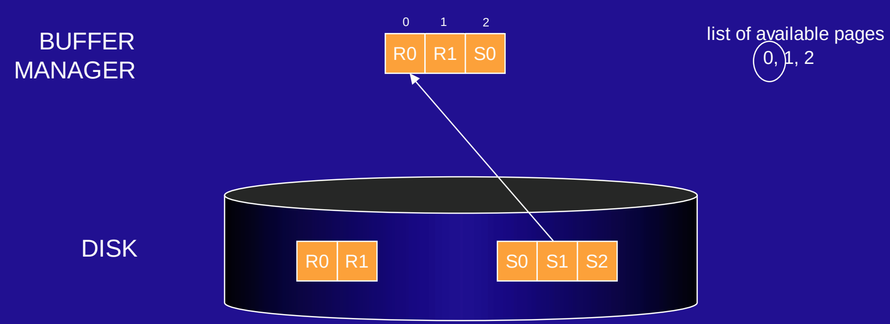

Problem
• Relation R being scanned
BUFFER MANAGER
0 1 2
R0 R1
list of available pages
2, 0, 1
DISK
R0 R1
S0 S1 S2
42


Problem
• Relation S having statistics updated
BUFFER MANAGER
0
2
R0
R1
S0
list of available pages
2, 0, 1
DISK
R0
R1
S0
S1
S2
43


Problem
• Relation S having statistics updated
BUFFER MANAGER
0
2
R0
R1
S0
list of available pages
0, 1, 2
DISK
R0
R1
S0
S1
S2
44

Problem

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

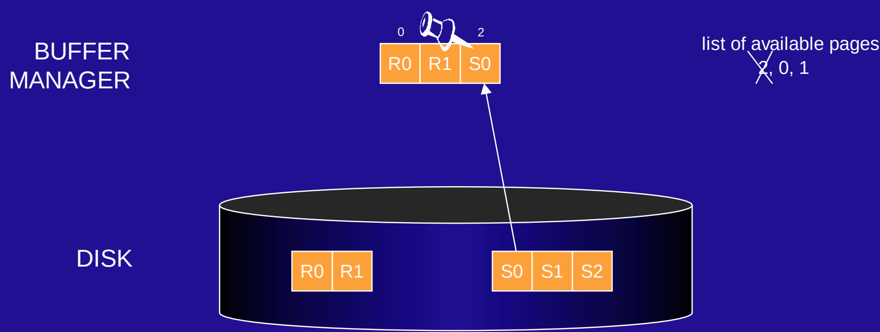


Most Recently Used Page Replacement

- 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

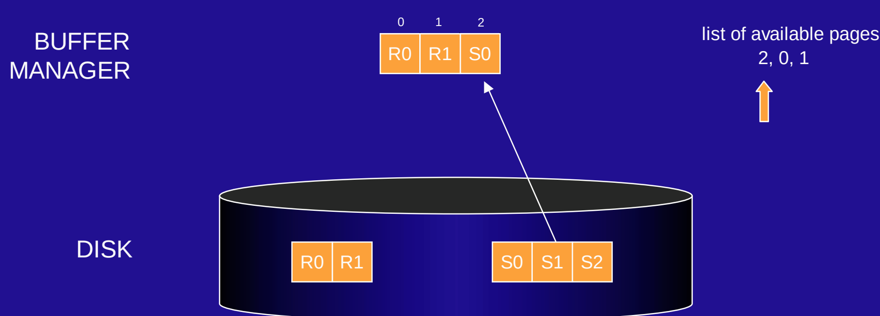
• When page S0 gets unpinned, the buffer #2 will be put in front of the available list



47

Statistics using MRU

• When page S0 gets unpinned, the buffer #2 will be put in front of the available list



48

Implementation Ingredients

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

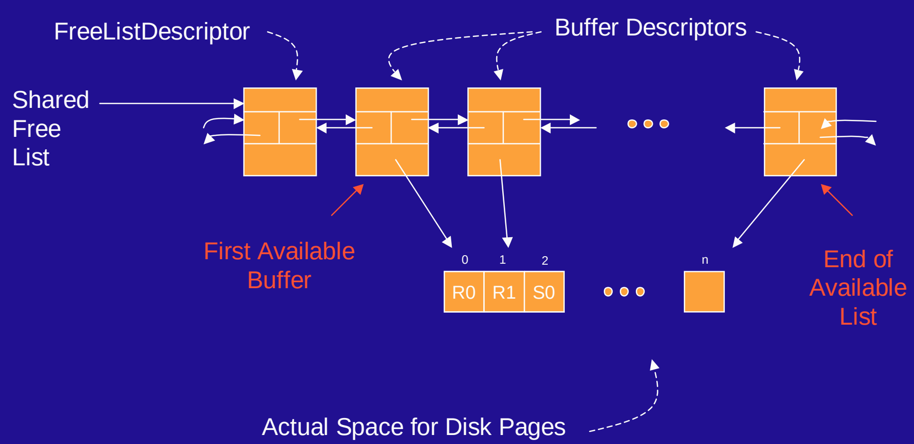
49

PostgreSQL Resources

- Download Sources
  - www.postgresql.org
- Development Documentation
  - developer.postgresql.org
- Discussion List for Developers
  - pgsql-hackers

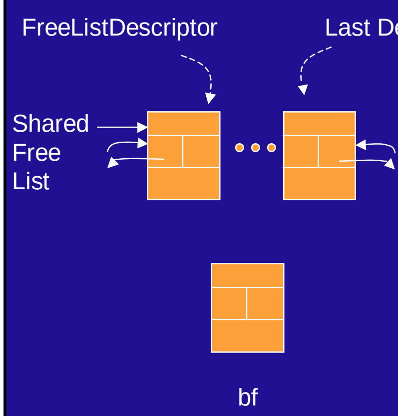
50

PostgreSQL Buffer Manager



51

Adding a Buffer

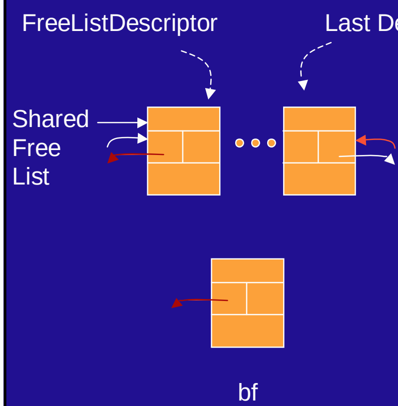



```
static void AddBufferToFreelist(BufferDesc *bf)
{
  /* change bf so it points to inFrontOfNew
     and its successor */
  bf->freePrev = SharedFreeList->freePrev;
  bf->freeNext = Free_List_Descriptor;

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

/src/backend/storage/buffer/freelist.c

52

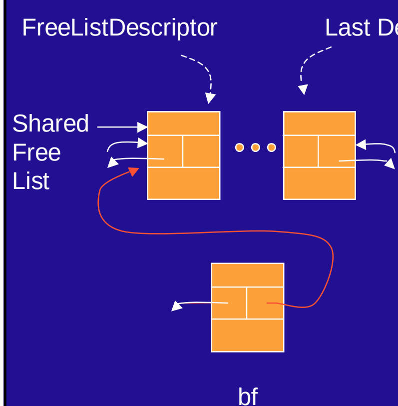
Adding a Buffer




```
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_
}
```

53

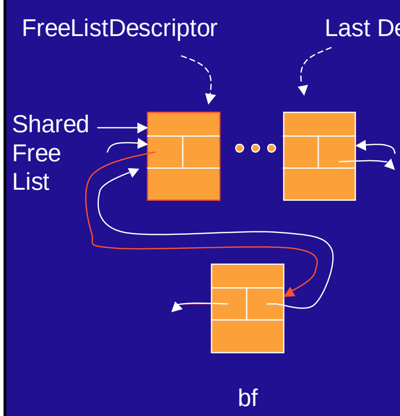
Adding a Buffer




```
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_
}
```

54

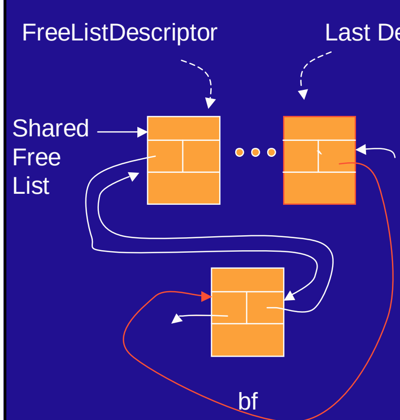
Adding a Buffer




```
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_
}
```

55

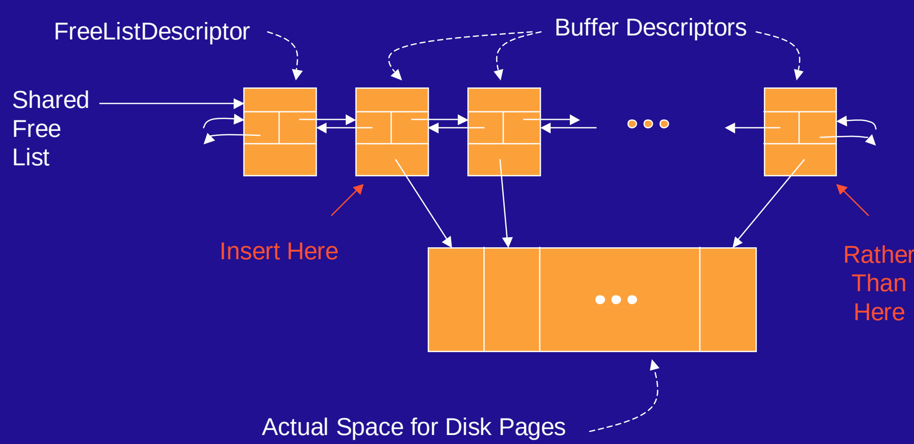
Adding a Buffer




```
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_
}
```

56

Implementing MRU



57

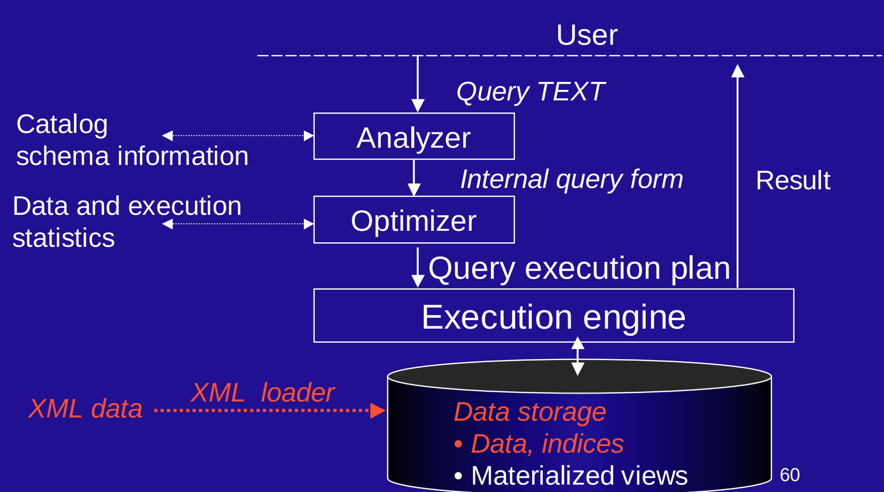
DEMO

58

Case 2

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

Simple XML storage system



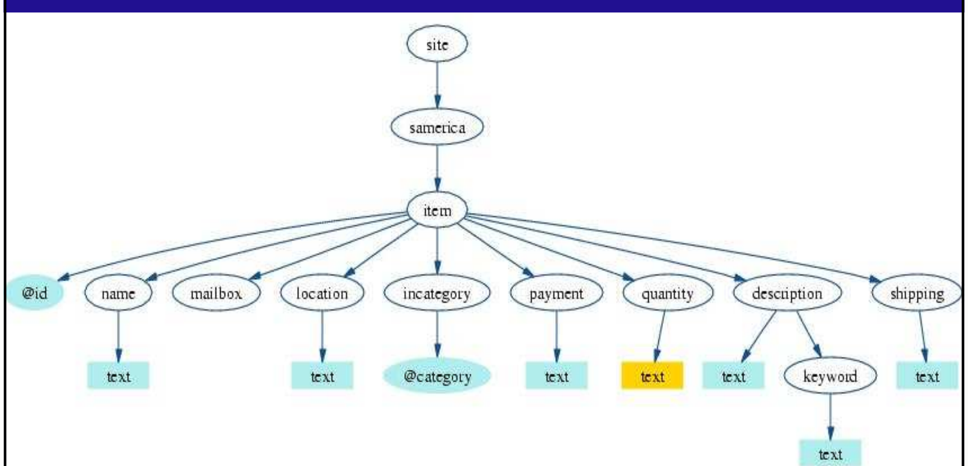
XML documents are data trees

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**<keyword>**wooden**</keyword>
window</description>
<shipping>**See description for charges**</shipping>
<incategory category="**category0**" />
<incategory category="**category1**" />
<mailbox />
</item>
</samerica>
</site>

61

XML documents are data trees



62

Simple data and storage model for XML

1. Add unique identifiers to all XML nodes

```
1 <site>
  2 <samerica> 4
    3 <item id="item6">
      5 <location>UnitedStates</location> 6<quantity>1</quantity>
      7 <name>chair</name>
      8 <payment>Creditcard, Cash</payment>
      9 <description>nice  10<keyword>wooden</keyword>
                    chair</description>
     11 <shipping>See description for charges</shipping>
     12 <incategory category="category0" /> 13
     14 <incategory category="category1" /> 15
     16 <mailbox />
    </item>
  </samerica>
</site>
```


Simple data and storage model for XML

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

Edge(pID, tag, cID, attr)

pID	tag	cID	attr
-	site	1	
1	samerica	2	
2	item	3	
3	id	4	X
2	name	5	
…………………………			
2	description	9	
9	keyword	10	
…………………………..			

Value(id, val)

id	val
3	"item6"
5	"United States"
6	"1"
………..	
9	"nice" "chair"
10	"wooden"

Simple data and storage model for XML

3. Many variants exist

Simple variant: partition by tag

Edge(pID, tag, cID, attr)

pID	tag	cID	attr
-	site	1	
1	samerica	2	
2	item	3	
3	id	4	X
2	name	5	
…………………………			
2	description	9	
9	keyword	10	
…………………………			

Value(id, val)

id	val
3	"item6"
5	"United States"
6	"1"
………..	
9	"nice" "chair"
10	"wooden"

65

Simple data and storage model for XML

A small step forward: indexing elements by their content

Edge(pID, tag, cID, attr)

pID	tag	cID	attr
-	site	1	
1	samerica	2	
2	item	3	
3	id	4	X
2	name	5	
…………………………			

Value(id, val)

id	val
3	"item6"
5	"United States"
6	"1"
………..	

Index <item> elements by their @id value

66

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)

67

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

68

Parsing and loading XML documents

```
import org.apache.xerces.parsers.*;
public class myParser extends DefaultHandler implements
   Parser {
   ..........................................................
   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 (unique ID), push on the stack
         // record the attributes
         // record the edge connecting to its father
   }
```

69

Parsing and loading XML documents

```
import org.apache.xerces.parsers.*;
public class myParser extends DefaultHandler
   implements Parser {
   ..........................................................
   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 lName,
   String qName) {
         // pop the stack
   }
```

70

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)

71

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

72

Loading element identifiers into BerkeleyDB

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

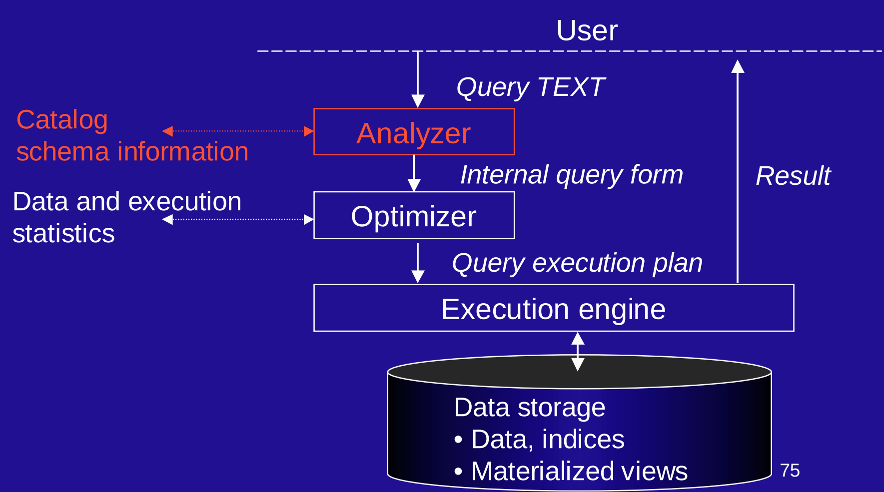
73

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

74

DBMS architecture

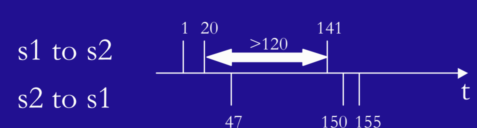


75

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

Packets	src	dst	len	time
	s1	s2	250	1
	s1	s2	270	20
	s1	s2	235	141
	s2	s1	330	47
	s2	s1	280	150
	s2	s1	305	155



76

AQuery in Pictures

Packets	src	dst	len	time
	s1	s2	250	1
	s1	s2	270	20
	s2	s1	330	47
	s1	s2	235	141
	s2	s1	280	150
	s2	s1	305	155

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

77

AQuery in Pictures

Packets	src	dst	len	time
	s1	s2	250	1
	s1	s2	270	20
	s2	s1	330	47
	s1	s2	235	141
	s2	s1	280	150
	s2	s1	305	155

Packets	src	dst	len	time
	s1	s2	250	1
	s1	s2	270	20
	s1	s2	235	141
	s2	s1	330	47
	s2	s1	280	150
	s2	s1	305	155

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

78

AQuery in Pictures

Packets	src	dst	len	time	c1
	s1	s2	250	1	F
	s1	s2	270	20	F
	s1	s2	235	141	T
	s2	s1	330	47	F
	s2	s1	280	150	F
	s2	s1	305	155	F

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

79

AQuery in Pictures

Packets	src	dst	len	time	c1	c2
	s1	s2	250	1	F	0
	s1	s2	270	20	F	0
	s1	s2	235	141	T	1
	s2	s1	330	47	F	1
	s2	s1	280	150	F	1
	s2	s1	305	155	F	1

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

80

AQuery in Pictures

Packets	src	dst	len	time	c1	c2
	s1	s2	250	1	F	0
	s1	s2	270	20	F	0
	s1	s2	235	141	T	1
	s2	s1	330	47	F	1
	s2	s1	280	150	F	1
	s2	s1	305	155	F	1

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

81

AQuery in Pictures

Packets	src	dst	len	time
	s1	s2	250	1
	s1	s2	270	20
	s1	s2	235	141
	s2	s1	330	47
	s2	s1	280	150
	s2	s1	305	155

src	dst	len	time
s1	s2	[250,270]	[1,20]
s1	s2	[235]	[141]
s2	s1	[330,280,305]	[47,150,155]

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

82

AQuery in Pictures

Packets	src	dst	len	time
	s1	s2	250	1
	s1	s2	270	20
	s1	s2	235	141
	s2	s1	330	47
	s2	s1	280	150
	s2	s1	305	155

src	dst	len	time
s1	s2	[250,270]	[1,20]
s1	s2	[235]	[141]
s2	s1	[330,280,305]	[47,150,155]

src	dst	avg(len)	count(*)
s1	s2	260	2
s1	s2	235	1
s2	s1	305	3

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

83

Other Order Built-in Functions

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

```
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

84

Type-checking Problem

```
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?

85

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, ant 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

86

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/~ceriel/LLgen.html)
    - Given a LL(1) grammar generates a recursive descent parser
    - Could have used yacc or bison (LALR parser generators) instead

87

More Ingredients

- K (www.kx.com)
  - APL descendent 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

88

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; }
…
```

89

AQuery Parser Definitions

```
query_specification    : { setctx(initqblk()); }
                         AQ_SELECT
                         { setsctx(SCTXSELECT); }
                         [ AQ_DISTINCT ]?
                         select_list
                         table_expression
                           [ AQ_EOC ]?
                       ;
select_list            : star_or_derived_column
                         select_list_tail
                       ;
…
derived_column         : value_expression
                         {appnexpr( POP, getctx(), getisctx());}
                       ;
```

90

D e m o

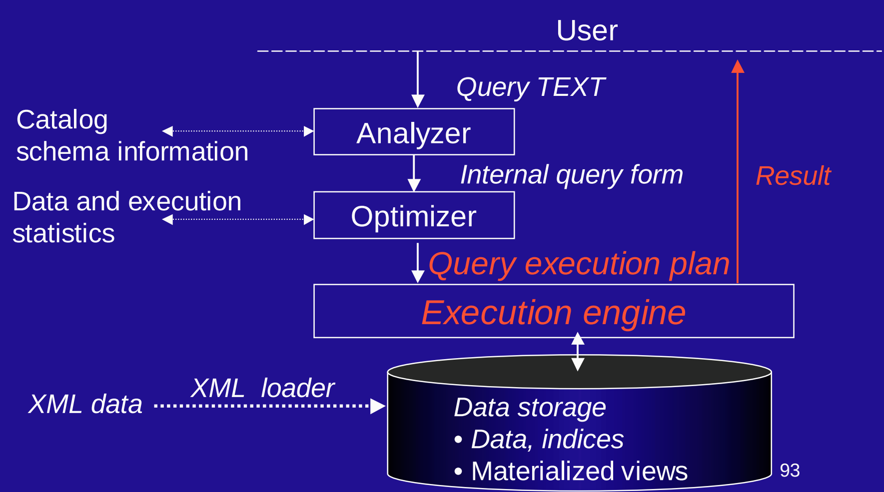
91

Case 4

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

92

Simple XML querying system



93

Simple XML querying system

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

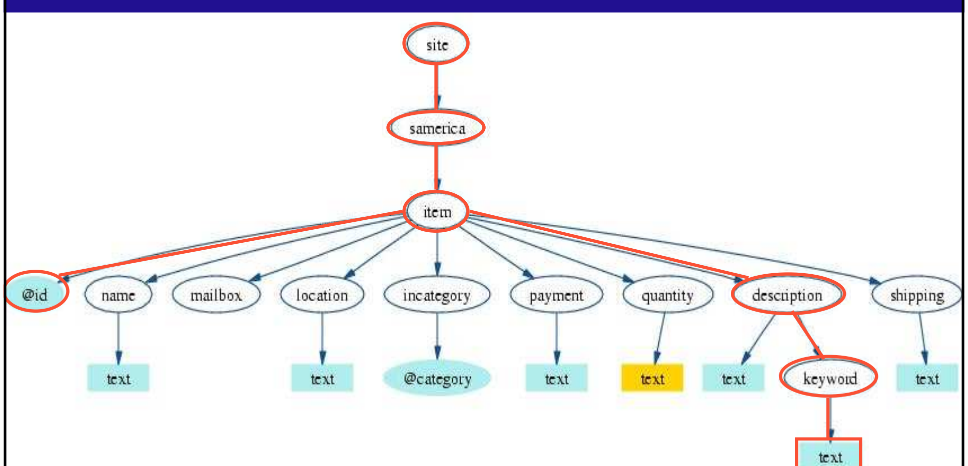
94

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

95

XML querying requires tree traversal



96

Tree traversal requires joins

- We want:
  all elements named "item", ($Edge_{item}$)
  whose parents have the tag "samerica", ($Edge_{samerica}$)
  whose parents have the tag "site" ($Edge_{site}$)
- Select e3.cID
  from $Edge_{site}$ e1, $Edge_{samerica}$ e2, $Edge_{item}$ e3
  where e1.cID=e2.pID and e2.cID=e3.cID

97

Implementing tree traversals

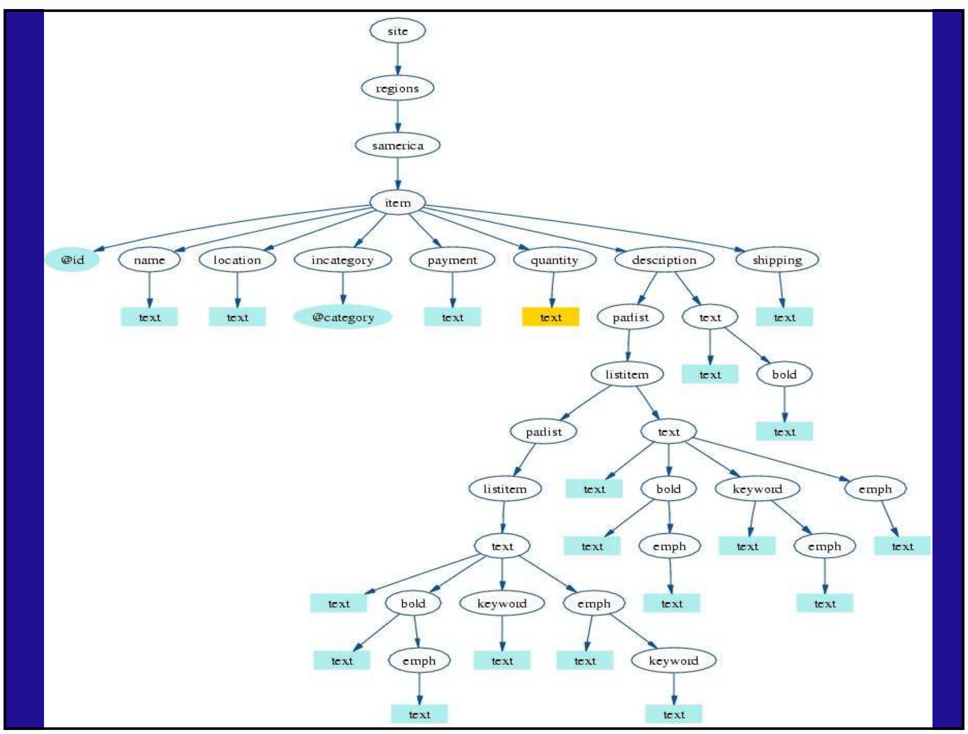
- site/samerica/item:
- $Edge_{site} \bowtie Edge_{samerica} \bowtie Edge_{item}$
  – 3-way join…
- site/samerica/item/keyword/description
- $Edge_{site} \bowtie Edge_{samerica} \bowtie Edge_{item} \bowtie \bowtie Edge_{description} \bowtie Edge_{keyword}$
  – 5-way join… …

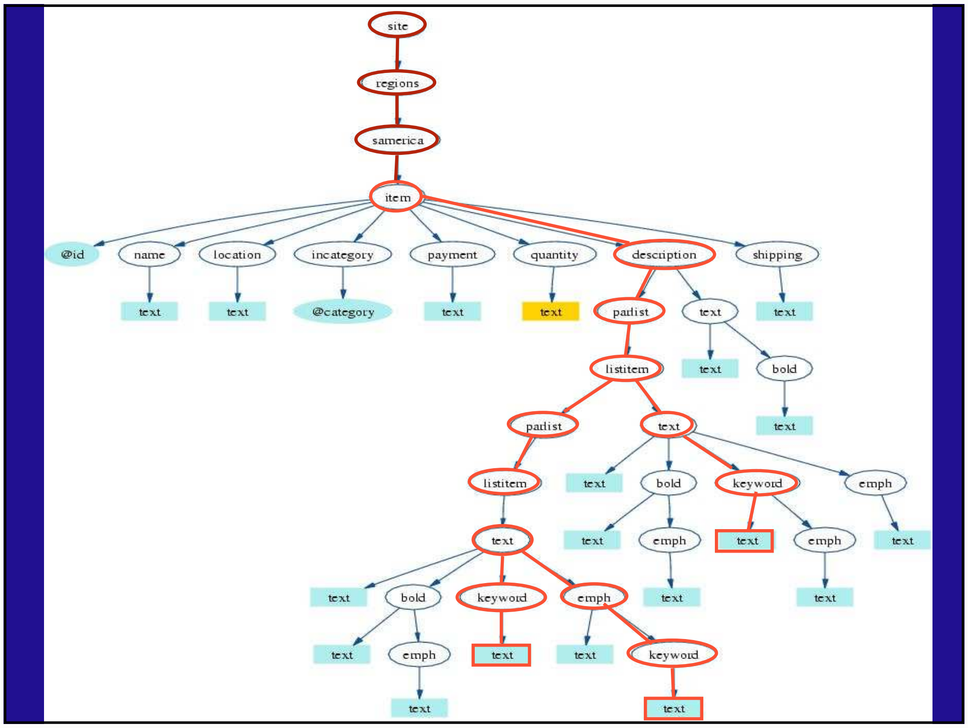
98

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, NAmerica, Africa
  - /site/people/person…
  - /site/catgraph/…
  - /site/open_auctions, /site/closed_auctions…

99





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.

102

Solution: better unique identifiers for XML nodes

Structural identifiers (beginTagNo, endTagNo, depth)

[1 1 13]<site>
[2 2 12]<samerica>
[3 3 11]<item id="**item6**">
[4 4 1]<location>**UnitedStates**</location>[5 5 2]<quantity>**1**</quantity>
[6 4 3]<name>chair</name>
[7 4 4]<payment>**Creditcard, Cash**</payment>
[8 4 6]<description>**nice** [9 5 5] <keyword>**wooden**</keyword>
chair</description>
[10 4 7]<shipping>**See description for charges**</shipping>
[11 4 8]<incategory category="**category0**" />
[12 4 9]<incategory category="**category1**" />
[13 4 10]<mailbox />
</item>
</samerica>
</site>

103

Structural identifiers and structural relationships

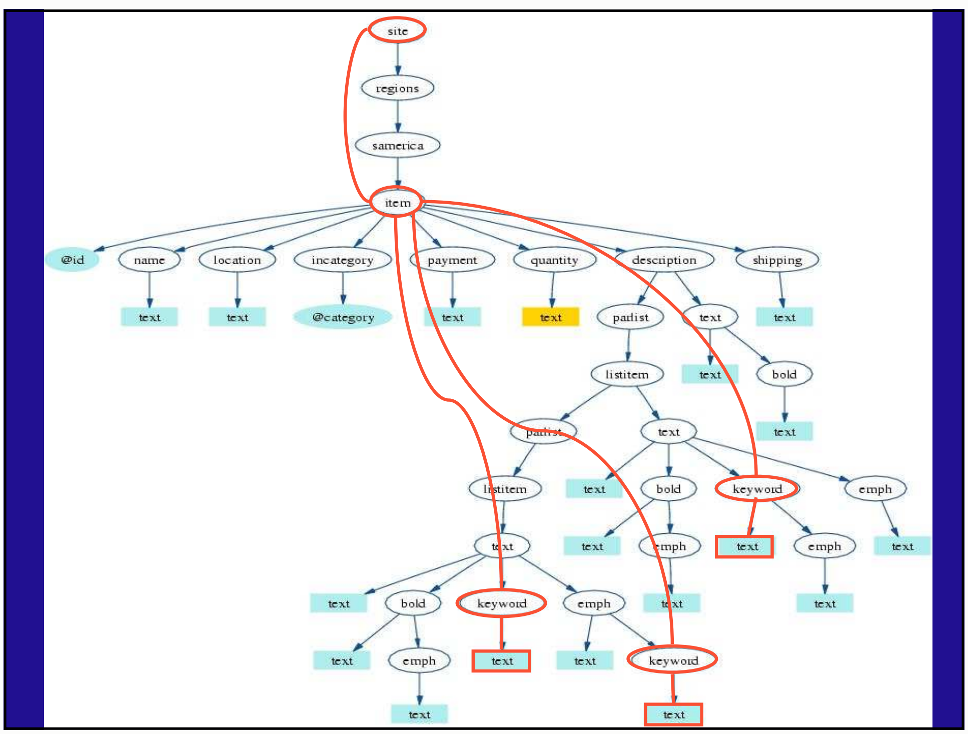
- Element x is an ancestor of element y iff:
  - *x.begin < y.begin and x.end > y.end*
- Example:
  - Item [3 3 11] is an ancestor of location [4 4 1]
  - Location [4 4 1] is not an ancestor of keyword [9 5 5]
- Element x is the parent of element y iff:
  - *x.begin < y.begin and x.end > y.end and x.depth + 1 = y.depth*
- Example:
  - Item [3 3 11] is the parent of location [4 4 1]

104

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:
  $ID_{site} \bowtie ID_{site/regions/samerica/item}$

105



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: $ID_{site} \bowtie ID_{site/regions/samerica/item}$
- site/regions/samerica/item//keyword: $ID_{site/regions.samerica/item} \bowtie (ID_{pathKeyword1} \cup ID_{pathkeyword2} \cup ID_{pathKeyword3})$

107

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

108

Parsing and loading, revisited with structural identifiers

```
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
```

109

Parsing and loading, revisited with structural identifiers

```
import org.apache.xerces.parsers.*;
public class myParser extends DefaultHandler
    implements Parser {
    ................................................................
    public void characters(char[] buffer, int start, int end){
          // same as before
    }
    public void endElement(String nsURI, String lName,
    String qName) {
          // same as before; update depth
    }
```

110

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

111

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…

112

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 Iterator1 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

113

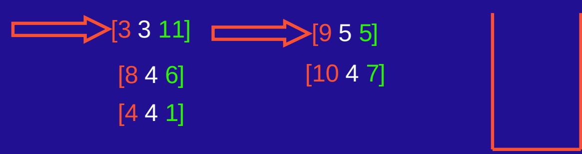
Implementing PathAccess

```
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 */ }
```

114

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 ancestores are "too old"

115

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"

116

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"

117

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"

118

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"

119

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"

120

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"

121

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>

122

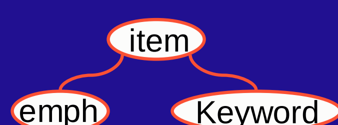
Simple extensions: XML indexing

- 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

123

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

124

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 ☺**

125

Conclusions from building our simple system

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

126

Part IV: (A non-exhaustive) Resource Directory

127

Agenda

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

128

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)

131

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

132

Parsing Tools

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

133

Rapid Development

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

134

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/)

135

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!)

136

Readings

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

137

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/dbbookl
  – Do read the complete examples
  – Slides are freely available
- Transformation-based optimization framework
  – Selinger, Astrahan, Lorie and Price. Access Path Selection in Relational Database Systems. SIGMOD 1979
  – G.Graefe, W.McKenna: The Volcano Optimizer Generator: Extensibility and Efficient Search, ICDE 1998

138

Adaptive (self-improving) query processing

- Entry point: survey

M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. Shah. *Adaptive query processing: Technology in evolution*. IEEE Data Engineering Bulletin, 23(2):7--18, 2000

139

Distributed query processing

- Complete overview:
  - P.Valduriez, T.Ozsu: "Principles of Distributed Database Systems", Prentice Halls, 1999
- Shorter, denser overview:
  - D. Kossmann: The State of the art in distributed query processing. ACM Computing Surveys 32(4): 422-469 (2000)
- Cost models for data integration frameworks
  - M.Roth, F. Ozcan, L.Haas: Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. VLDB 1999: 599-610

140

XML-on-relational

- J. Shanmugasundaram, E.Shekita, M.Carey, B.Lindsay, H.Pirahesh, B.Reinwald: Efficiently publishing relational data as XML documents. VLDB Journal 10(2-3): 133-154 (2001)
- J.Shanmugasundaram, J.Kiernan, E.Shekita, C.Fan, J.Funderburk: Querying XML Views of Relational Data. VLDB 2001: 261-270
- D. DeHaan, D. Toman, M. Consens, T.Ozsu: A Comprehensive XQuery to SQL **Translation** using Dynamic Interval Encoding, SIGMOD 2003

141

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…

142