ECS 165B: Database System Implementation
Lecture 20

UC Davis
May 12, 2010

Portions based on slides due to Zack Ives

Class Agenda

- Last time:
  - Views and Relational Encodings of XML (1)
- Today:
  - Views and Relational Encodings of XML (2)
  - Cookbook Session: Query Evaluation Plans
- Reading:
  - none

Views and Relational Encodings of XML (2)

The Simplest Way to Encode a Tree: Tree Tables

- Suppose we had:
-

```
<tree id="0">
  <content id="1">
    <sub-
content>XYZ
    </sub-content>
    <i-content>14
    </i-content>
  </content>
</tree>
```

- Where we have no IDs, invent values

Tree

key	label	type	value	parent
0	tree	ref	-	-
1	content	ref	-	0
2	sub-content	ref	-	1
3	i-content	ref	-	1
4	-	str	XYZ	2
5	-	int	14	3

What are the shortcomings of this approach?

Florescu/Kossmann Improved Tree Approach

- Consider order, typing; separate the values

Tree

parent	ord	label	flag	target
-	1	tree	ref	0
0	1	content	ref	1
1	1	sub-content	str	v2
1	1	i-content	int	v3

Vint

vid	value
v3	14

Vstring

vid	value
v2	XYZ

A Relation that Mirrors the XML Hierarchy

```
<tree id="0">
    <content id="1">
        <sub-content>XYZ</sub-content>
        <i-content>14</i-content>
    </content>
</tree>
```

- Output relation, encoding this tree, would look like:

rLabel	rid	rOrd	clabel	cid	cOrd	sLabel	sid	sOrd	str	int
tree	0	1	-	-	-	-	-	-	-	-
-	0	1	content	1	1	-	-	-	-	-
-	0	1	-	1	1	sub-content	2	1	-	-
-	0	1	-	1	1	-	2	1	XYZ	-
-	0	1	-	1	2	i-content	3	1	-	-
-	0	1	-	1	2	-	3	1	-	14

“Inlining” Techniques

- Folks at Wisconsin noted we can exploit the “structured” aspects of semi-structured XML
  - If we’re given a DTD, often the DTD has a lot of required (and often singleton) child elements
    - Book(title, author*, publisher)
  - Recall how normalization works in a traditional DBMS:
    - Decompose until we have everything in a relation determined by the keys
    - … But don’t decompose any further than that
  - Shanmugasundaram et al. try not to decompose XML beyond the point of singleton children

XML -> Relations; XQuery -> SQL?

- Once we've encoded the XML in relational form, would like to query it! (Using XQuery, of course)
- For limited fragments of XQuery, this is possible, via translation to SQL
- Details of translation depend heavily on the encoding scheme
- We'll look at 2 examples: one for XPERANTO-style encoding, and one for an inlined encoding

Running Example for XQuery -> SQL Translation

XQuery:

```
for $X in document(“mydoc”)/tree/content
where $X/sub-content = “XYZ”
return $X
```

Source document:

```
<tree id=“0”>
    <content id=“1”>
        <sub-content>XYZ</sub-content>
        <i-content>14</i-content>
    </content>
</tree>
```

Query output:

```
<content id=“1”>
    <sub-content>XYZ</sub-content>
    <i-content>14</i-content>
</content>
```

EXPERANTO-Style Encoding: Input

Source document:

```
<tree id="0">
    <content id="1">
        <sub-content>XYZ</sub-content>
        <i-content>14</i-content>
    </content>
</tree>
```

rLabel	rid	rOrd	clabel	cid	cOrd	sLabel	sid	sOrd	str	int
tree	0	1	-	-	-	-	-	-	-	-
-	0	1	content	1	1	-	-	-	-	-
-	0	1	-	1	1	sub-content	2	1	-	-
-	0	1	-	1	1	-	2	1	XYZ	-
-	0	1	-	1	2	i-content	3	1	-	-
-	0	1	-	1	2	-	3	1	-	14

EXPERANTO-Style Encoding: Output

Query output:

```
<content id="1">
    <sub-content>XYZ</sub-content>
    <i-content>14</i-content>
</content>
```

rlabel	rid	rOrd	cLabel	cid	cOrd	str	int
content	1	1	-	-	-	-	-
-	1	1	sub-content	2	1	-	-
-	1	1	-	2	1	XYZ	-
-	1	2	i-content	3	1	-	-
-	1	2	-	3	1	-	14

EXPERANTO-Style Encoding: Query

XQuery:

```
for $X in document("mydoc")/tree/content
where $X/sub-content = "XYZ"
return $X
```

SQL:

```
select T5.clabel as rlabel, T5.cid as rid, T5.cOrd as rOrd,
    T5.sLabel as cLabel, T5.sid as cid, T5.sOrd as cOrd,
    T5.str  as str, T5.int as int
from Tree T1, Tree T2, Tree T3, Tree T4, Tree T5
where T1.rLabel = "tree" and
    T2.rid = T1.rid and T2.rOrd = T1.rOrd and T2.clabel = "content" and
    T3.rid = T2.rid and T3.rOrd = T2.rOrd and T3.cid = T2.cid and
        T3.cOrd = T2.cOrd and T3.label = "sub-content" and
    T4.rid = T3.rid and ... T4.sid = T3.sid and  T4.str = "XYZ" and
    T5.rid = T2.rid and ... and T5.cOrd = T2.cOrd
```

EXPERANTO-Style Encoding: Query (2)

```
select T5.clabel as rlabel, ..., T5.int as int
from Tree T1, Tree T2, Tree T3, Tree T4, Tree T5
where T1.rLabel = "tree" and
    T2.rid = T1.rid and ... and T2.clabel = "content" and
    T3.rid = T2.rid and ... and T3.label = "sub-content" and
    T4.rid = T3.rid and ... and T4.str = "XYZ" and
    T5.rid = T1.rid and ... and T5.cid = T2.cid
```

rLabel	rid	rOrd	clabel	cid	cOrd	sLabel	sid	sOrd	str	int
tree	0	1	–	–	–	–	–	–	–	–
–	0	1	content	1	1	–	–	–	–	–
–	0	1	–	1	1	sub-content	2	1	–	–
–	0	1	–	1	1	–	2	1	XYZ	–
–	0	1	–	1	2	i-content	3	1	–	–
–	0	1	–	1	2	–	3	1	–	14

Inlined Encoding: Input Tables

Source document:

```
<tree id="0">
    <content id="1">
        <sub-content>XYZ</sub-content>
        <i-content>14</i-content>
    </content>
</tree>
```

- Suppose we know from DTD or XML-Schema that:
  - root node is "tree";
  - "content" node only occurs under "tree";
  - "content" node contains ID attribute "id" and exactly two children: "sub-content", with text data, and "i-content", with integer-valued data
- Can encode the tree as follows:

Tree

id
0

Content

parentID	id	ord	sub-content	i-content
0	1	0	XYZ	14

Inlined Encoding: Query

XQuery:

```
for $X in document("mydoc")/tree/content
where $X/sub-content = "XYZ"
return $X
```

SQL:

```
select Content.*
from Tree, Content
where Tree.id = Content.parentID and
    Content.sub-content = "XYZ"
```

Output is another inline-encoded relation:

parentID	id	ord	sub-content	i-content
0	1	0	XYZ	14

2-way join, instead of a 5-way join!

Summary: Relational "Views" of XML

- We've seen that views are useful things
- Allow us to store and refer to the results of a query
- We've seen a examples of relational "views" of XML, and SQL translations of XQuery over such views
  - Current versions of the major DBMSs support XML and (fragments of) XQuery this way
  - Challenge: limiting the number of joins required in the translated queries

And Now For Something Completely Different

Query Engine Sneak Preview and

Cookbook Session

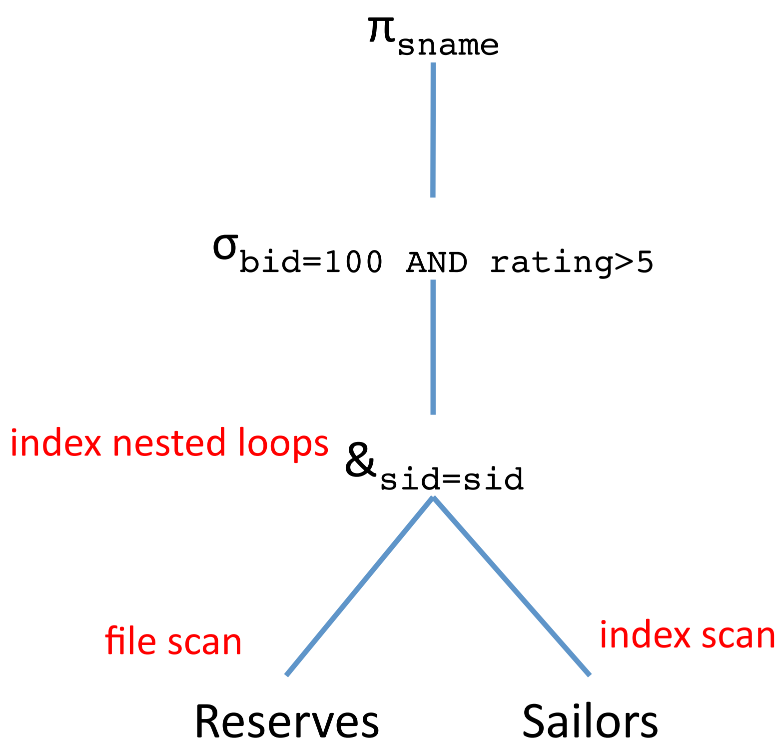
DavisDB, Part 4: Query Engine

- Culmination of the project: you'll implement a query engine for a fragment of SQL
  - Queries: `select-from-where`
  - Updates: `insert into`, `delete from`, `update`
- As in Part 3, `SystemParser` handles the front-end, you implement the back-end: a class called `QueryEngine`
- Relatively low bar for getting full credit
  - Optimization not required
  - Starter code and architectural template provided
- Opportunities for extra credit
  - e.g., Query optimizer

Query Engine API

- (cf. the Doxygen docs...)

How to Implement an Execution Engine?



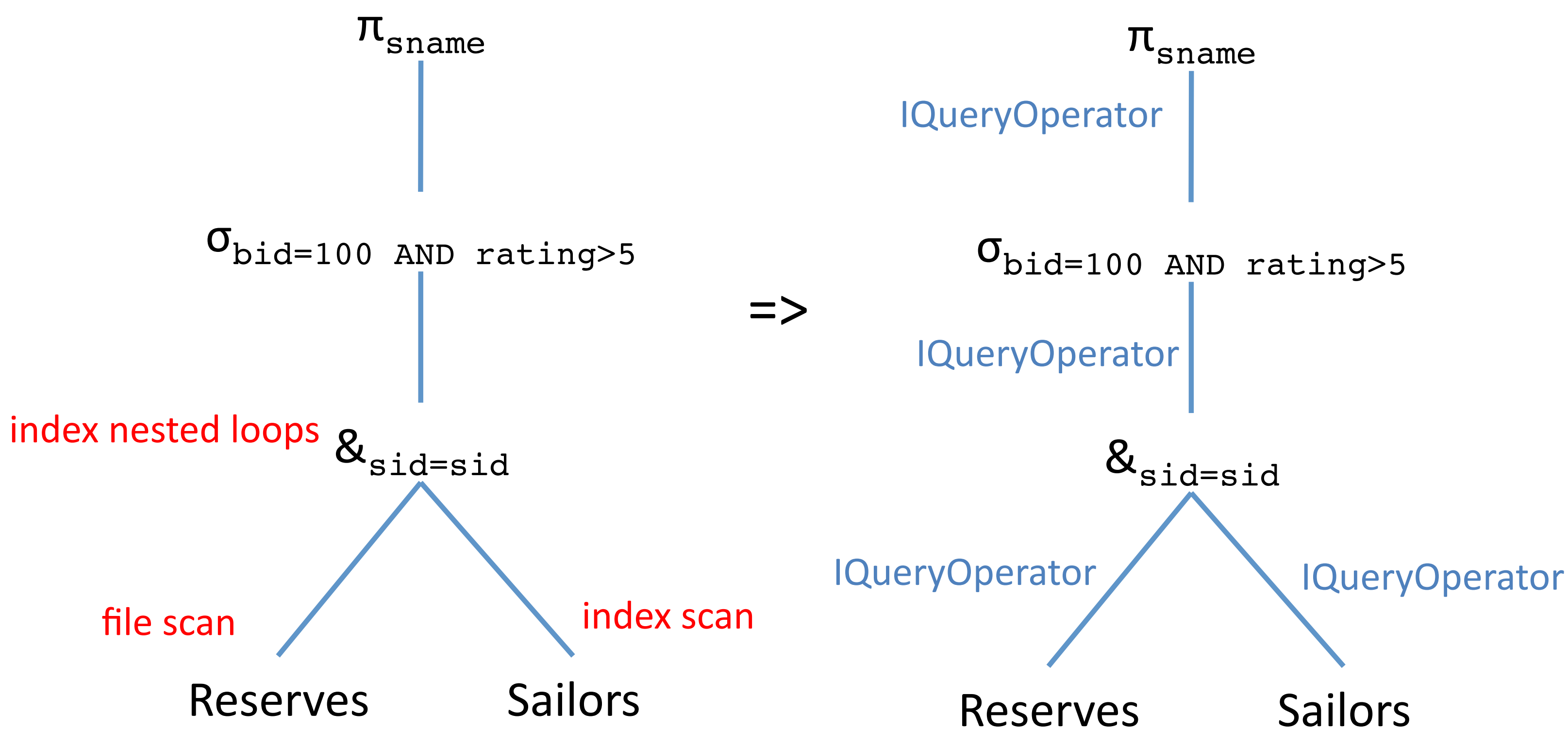
=>

???

Physical query plan

C++ implementation

How to Implement an Execution Engine?



Physical query plan

C++ implementation

IQueryOperator: an Abstract Interface

- What is an "abstract interface" in C++?
- A base class with only **abstract virtual** methods

```
virtual ReturnCode getNextRecord(char* data) = 0;
```

- Other classes inherit from this base class ("implement" the interface) and fill in the method implementations
- One technical exception: virtual destructor must have implementation, but can be empty

```
virtual ~IQueryOperator() {};
```

What's a Virtual Method?

- C++ versus Java: in Java, **all** methods are virtual!

```
class A {
    void foo() { printf("A foo"); }
    virtual void bar() { printf("A bar"); }
}
```

```
class B : A {
    void foo() { printf("B foo"); }
    virtual void bar() { printf("B bar"); }
}
```

```
void biz(A* a, B* b) {
    A* a, B* b;  A* c = (A*)b;
    a->foo();
    a->bar();
    b->foo();
    b->bar();
    c->foo();
    c->bar();
}
```

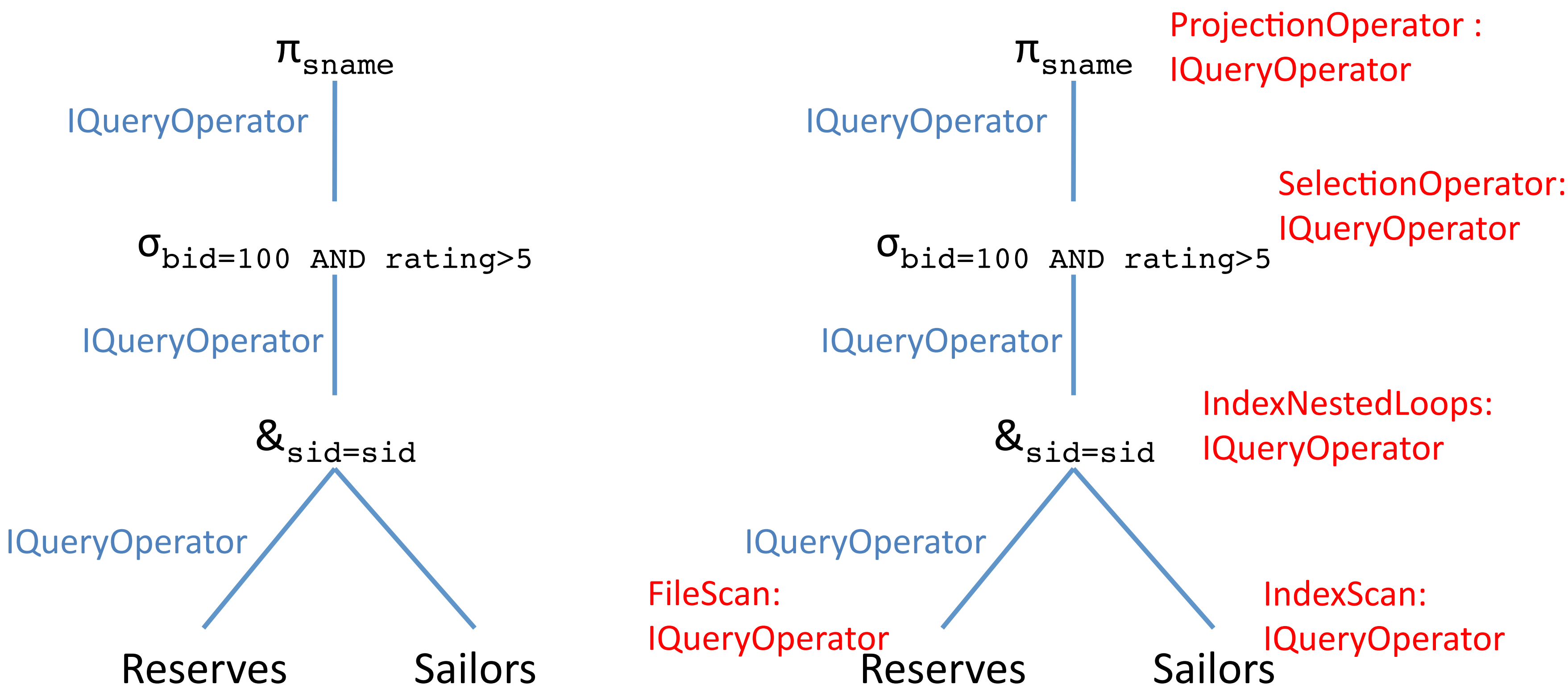
OUTPUT of call to biz(a,b):

A foo
A bar
B foo
B bar
A foo
B bar

QUESTION: why declare destructors virrtual?

Why Use Interfaces in the Execution Engine?

- An operator shouldn't have to know about all the different physical operators that might be below it in the tree!



IQueryOperator

- (cf. code in Xcode...)

"Canonical" Execution Plans

- Given a select-from-where query, will only be required to build a "canonical" execution plan
  - A plan fully determined by the order of relations in the "from" clause and availability of indices
- Extra credit: build an optimizer to explore other plans
  - Based on heuristics or statistics

Putting it All Together

- Let's go to the videotape!