Database Applications (15-415)

DBMS Internals- Part IX
Lecture 20, April 5, 2015

Mohammad Hammoud
Today...

- Last Session:
  - DBMS Internals- Part VIII
    - Algorithms for Relational Operations (Cont’d)

- Today’s Session:
  - DBMS Internals- Part IX
    - Query Optimization

- Announcements:
  - PS4 is now posted. It is due on Sunday, April 12 by midnight
  - Quiz II is on Thursday, April 9th (all concepts covered after the midterm are included)
DBMS Layers

Queries

Query Optimization and Execution

Relational Operators

Files and Access Methods

Buffer Management

Disk Space Management

Transaction Manager

Lock Manager

Recovery Manager

DB
Outline

- A Brief Primer on Query Optimization
- Evaluating Query Plans
- Relational Algebra Equivalences
- Estimating Plan Costs
- Enumerating Plans
Cost-Based Query Sub-System

Queries

Select *  
From Blah B  
Where B.blah = blah

Usually there is a heuristics-based rewriting step before the cost-based steps.
Query Optimization Steps

- **Step 1**: Queries are parsed into internal forms (e.g., parse trees)
- **Step 2**: Internal forms are transformed into ‘canonical forms’ (syntactic query optimization)
- **Step 3**: A *subset* of alternative plans are enumerated
- **Step 4**: Costs for alternative plans are estimated
- **Step 5**: The query evaluation plan with the *least estimated cost* is picked
Required Information to Evaluate Queries

- To estimate the costs of query plans, the query optimizer examines the system catalog and retrieves:
  - Information about the types and lengths of fields
  - Statistics about the referenced relations
  - Access paths (indexes) available for relations

- In particular, the *Schema* and *Statistics* components in the Catalog Manager are inspected to find a good enough query evaluation plan
Cost-Based Query Sub-System

Queries

Select *  
From Blah B  
Where B.blah = blah

Usually there is a heuristics-based rewriting step before the cost-based steps.

Catalog Manager

Query Parser

Query Optimizer

Plan Generator

Plan Cost Estimator

Query Plan Evaluator

Schema

Statistics
Catalog Manager: The Schema

- What kind of information do we store at the Schema?
  - Information about **tables** (e.g., table names and integrity constraints) and **attributes** (e.g., attribute names and types)
  - Information about **indices** (e.g., index structures)
  - Information about **users**

- Where do we store such information?
  - In tables, hence, can be queried like any other tables
  - For example: Attribute_Cat (attr_name: **string**, rel_name: **string**; type: **string**; position: **integer**)

Catalog Manager: Statistics

- **What would you store at the Statistics component?**
  - NTuples(R): # records for table R
  - NPages(R): # pages for R
  - NKeys(I): # distinct key values for index I
  - INPages(I): # pages for index I
  - IHeight(I): # levels for I
  - ILow(I), IHigh(I): range of values for I
  - ...

- Such statistics are important for estimating plan costs and result sizes (*to be discussed shortly!*)
SQL Blocks

- SQL queries are optimized by *decomposing* them into a collection of smaller units, called **blocks**.

- A block is an SQL query with:
  - No nesting
  - Exactly 1 SELECT and 1 FROM clauses
  - At most 1 WHERE, 1 GROUP BY and 1 HAVING clauses

- A typical relational query optimizer concentrates on optimizing a single block at a time.
Translating SQL Queries Into Relational Algebra Trees

An SQL block can be thought of as an algebra expression containing:
- A cross-product of all relations in the FROM clause
- Selections in the WHERE clause
- Projections in the SELECT clause

Remaining operators can be carried out on the result of such SQL block
Translating SQL Queries Into Relational Algebra Trees *(Cont’d)*

Still the same result!

How can this be guaranteed?
Translating SQL Queries Into Relational Algebra Trees (Cont’d)

OBSERVATION: try to perform selections and projections early!
How to evaluate a query plan (as opposed to evaluating an operator)?
Outline

- A Brief Primer on Query Optimization
- Evaluating Query Plans
- Relational Algebra Equivalences
- Estimating Plan Costs
- Enumerating Plans
A query evaluation plan (or simply a plan) consists of an extended relational algebra tree (or simply a tree).

A plan tree consists of annotations at each node indicating:
- The access methods to use for each relation
- The implementation method to use for each operator

Consider the following SQL query $Q$:

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND R.bid=100 AND S.rating>5
```

What is the corresponding RA of $Q$?
Query Evaluation Plans (Cont’d)

- $Q$ can be expressed in relational algebra as follows:

$$\pi_{\text{ssname}}(\sigma_{\text{bid}=100 \land \text{rating} > 5}(\text{Reserves} \bowtie_{\text{sid} = \text{sid}} \text{Sailors}))$$

A RA Tree:

```
\set\text{ssname} \set\text{bid}=100 \land \text{rating} > 5
\bowtie\text{sid} = \text{sid}
```

An Extended RA Tree:

```
\set\text{ssname} \set\text{bid}=100 \land \text{rating} > 5
\bowtie\text{sid} = \text{sid}
```

(Simple Nested Loops)

(On-the-fly)

(File Scan) Reserves

Sailors (File Scan)
Pipelining vs. Materializing

- When a query is composed of several operators, the result of one operator can sometimes be *pipelined* to another operator.

*Pipeline* the output of the join into the selection and projection that follow.

**Diagram:**
- **Reserves** (File Scan)
- **Sailors** (File Scan)
  - *sid=sid*
  - *bid=100*
  - *rating > 5*
  - (*Simple Nested Loops*)
  - Applied on-the-fly
  - (On-the-fly)
Pipelining vs. Materializing

- When a query is composed of several operators, the result of one operator can sometimes be pipelined to another operator.

Pipeline the output of the join into the selection and projection that follow.

In contrast, a temporary table can be materialized to hold the intermediate result of the join and read back by the selection operation!

Pipelining can significantly save I/O cost!
The I/O Cost of the Q Plan

What is the I/O cost of the following evaluation plan?

\[
\begin{align*}
\text{(File Scan) Reserves} & \quad \text{Simple Nested Loops} \quad \text{Sailors (File Scan)} \\
\text{(On-the-fly)} & \quad \text{On-the-fly} \\
\text{sid=sid} & \\
\sigma_{\text{bid=}100 \land \text{rating > 5}} & \\
\prod_{\text{sname}} &
\end{align*}
\]

✓ The cost of the join is \(1000 + 1000 \times 500 = 501,000\) I/Os (assuming page-oriented Simple NL join)
✓ The selection and projection are done on-the-fly; hence, do not incur additional I/Os
Pushing Selections

- How can we reduce the cost of a join?
  - By reducing the sizes of the input relations!

Involves \( bid \) in Reserves; hence, “push” ahead of the join!
Involves \( rating \) in Sailors; hence, “push” ahead of the join!
Pushing Selections

- How can we reduce the cost of a join?
  - By reducing the sizes of the input relations!
The I/O Cost of the New Q Plan

- What is the I/O cost of the following evaluation plan?

\[
\begin{align*}
\text{Reserves} & \quad \text{sname} \quad \text{(On-the-fly)} \quad \text{sid=sid} \\
\text{(Scan; write to temp T1)} & \quad \text{bid=100} \\
\text{Sailors} & \quad \text{rating > 5} \quad \text{(Scan; write to temp T2)}
\end{align*}
\]

Cost of Scanning Reserves = 1000 I/Os  
Cost of Writing T1 = 10* I/Os (later)  
Cost of Scanning Sailors = 500 I/Os  
Cost of Writing T2 = 250* I/Os (later)

*Assuming 100 boats and uniform distribution of reservations across boats.  
*Assuming 10 ratings and uniform distribution over ratings.
The I/O Cost of the New Q Plan

- What is the I/O cost of the following evaluation plan?

**Merge Cost** = 10 + 250 = 260 I/Os (assuming B = 5)

**Cost** = 2×2×10 = 40 I/Os (assuming B = 5)

**Scan**; **write to temp T1**

**Cost** = 2×4×250 = 2000 I/Os (assuming B = 5)

**Scan**; **write to temp T2**

**Sort-Merge Join**

**Reserves**

**Sailors**

**On-the-fly**

(sid=sid)

(σ

rating > 5)

(σ

bid=100)
The I/O Cost of the *New* Q Plan

- What is the I/O cost of the following evaluation plan?

```
\[ \begin{array}{c}
\text{sid} = \text{sid} \\
\text{scan; write to temp T1} \\
\text{(Sort-Merge Join)} \\
\text{sid} = \text{sid} \\
\text{rating > 5} \\
\text{scan; write to temp T2} \\
\text{Reserves} \\
\text{Sailors} \\
\text{on-the-fly, thus, do not incur additional I/Os} \\
\end{array} \]
```
The I/O Cost of the New Q Plan

- What is the I/O cost of the following evaluation plan?

**Merge Cost** = 10 + 250 = 260 I/Os

**Cost** = 2×2×10 = 40 I/Os (assuming B = 5)

**Cost of Scanning Reserves** = 1000 I/Os

**Cost of Writing T1** = 10 I/Os (later)

**Cost of Scanning Sailors** = 500 I/Os

**Cost of Writing T2** = 250 I/Os (later)

**Merge Cost** = 10 + 250 = 260 I/Os

Done on-the-fly, thus, do not incur additional I/Os

**Total Cost** = 1000 + 10 + 500 + 250 + 40 + 2000 + 260 = 4060 I/Os
The I/O Costs of the Two Q Plans

Total Cost = 501,000 I/Os

Total Cost = 4060 I/Os
Pushing Projections

- How can we reduce the cost of a join?
  - By reducing the sizes of the input relations!

- Consider (again) the following plan:

```
Reserves \rightarrow
\mathrm{Scan;} \mathrm{write} \rightarrow T1
\sigma_{\text{bid}=100} \rightarrow \text{Reserves}
\pi_{\text{sid} = \text{sid}} \rightarrow \text{Reserves}
\sigma_{\text{rating} > 5} \rightarrow \text{Sailors}
\pi_{\text{sid}, \text{sname}} \rightarrow \text{Sailors}
```

- What are the attributes required from T1 and T2?
  - $\text{Sid}$ from T1
  - $\text{Sid}$ and $\text{sname}$ from T2

Hence, as we scan Reserves and Sailors we can also remove unwanted columns (i.e., “Push” the projections ahead of the join)!
Pushing Projections

- How can we reduce the cost of a join?
  - By reducing the sizes of the input relations!

- Consider (again) the following plan:

```
(Scan; write to temp T1) Reserves
  (σ_{bid=100} (Reserves))
  (σ_{rating > 5} (Sailors))
  (σ_{sid=sid} (Sailors))
  (σ_{sname} (Sailors))

"Push" ahead the join
```

The cost after applying this heuristic can become 2000 I/Os (as opposed to 4060 I/Os with only pushing the selection)!
Using Indexes

- What if indexes are available on Reserves and Sailors?

- With clustered index on `bid` of Reserves, we get 100,000/100 = 1000 tuples (assuming 100 boats and uniform distribution of reservations across boats)

- Since the index is clustered, the 1000 tuples appear consecutively within the same bucket; thus # of pages = 1000/100 = 10 pages
Using Indexes

- What if indexes are available on Reserves and Sailors?

- For each selected Reserves tuple, we can retrieve matching Sailors tuples using the hash index on the sid field.
- Selected Reserves tuples need not be materialized and the join result can be pipelined!
- For each tuple in the join result, we apply rating > 5 and the projection of sname on-the-fly.
Using Indexes

- What if indexes are available on Reserves and Sailors?

Is it necessary to project out unwanted columns?

**NO**, since selection results are NOT materialized

(Clustered hash index on bid)  
(Hash index on sid)  
(Use hash index; do not write result to temp)  
(On-the-fly)

- Reserves

- Sailors

- \( \text{rating} > 5 \) (On-the-fly)

- \( \text{sid} = \text{sid} \) (Index Nested Loops, with pipelining)
Using Indexes

- What if indexes are available on Reserves and Sailors?

```
(On-the-fly)

\[
\begin{align*}
&T \text{name} \\
&\sigma \text{rating > 5} \\
&\bowtie \text{sid=sid} \\
&\sigma \text{bid=100} \\
&\text{Sailors} \\
&\text{Reserves}
\end{align*}
\]

(Clustered hash index on bid)

(Hash index on sid)

Does the hash index on sid need to be clustered?

NO, since there is at most 1 matching Sailors tuple per a Reserves tuple! Why?
Using Indexes

- What if indexes are available on Reserves and Sailors?

```
\[ \begin{align*}
\text{Reserves} & \quad \text{Sailors} \\
\text{sid} & = \text{sid} \\
\text{rating} & > 5 \\
\text{sid} & = \text{sid} \\
\text{bid} & = 100
\end{align*} \]
```

(Use hash index; do not write result to temp)

(Clustered hash index on bid)

(Hash index on sid)

Cost = 1.2 I/Os (if A(1)) or 2.2 (if A(2))
Using Indexes

- What if indexes are available on Reserves and Sailors?

![Diagram showing query execution plan with indexes and joins]

Why not **pushing** this selection ahead of the join?

It would require a scan on Sailors!
The I/O Cost of the New $Q$ Plan

What is the I/O cost of the following evaluation plan?

- (Clustered hash index on bid) Reserves
- (Use hash index; do not write result to temp) Sailors
- (Index Nested Loops, with pipelining) sid=sid
- (On-the-fly) rating > 5
- (On-the-fly) sname

10 I/Os

Cost = 1.2 I/Os for 1000 Reserves tuples; hence, 1200 I/Os

Total Cost = 10 + 1200 = 1210 I/Os
Comparing I/O Costs: Recap

Total Cost = 501,000 I/Os

Total Cost = 4060 I/Os

Total Cost = 1210 I/Os
But, How Can we Ensure Correctness?

Canonical form

Still the same result!

How can this be guaranteed?
Outline

- A Brief Primer on Query Optimization
- Evaluating Query Plans
- Relational Algebra Equivalences
- Estimating Plan Costs
- Enumerating Plans
A relational query optimizer uses *relational algebra equivalences* to identify many *equivalent* expressions for a given query.

Two relational algebra expressions over the same set of input relations are said to be *equivalent* if they produce the same result on all relations’ instances.

Relational algebra equivalences allow us to:

- Push selections and projections ahead of joins
- Combine selections and cross-products into joins
- Choose different join orders
RA Equivalences: Selections

- Two important equivalences involve selections:
  1. Cascading of Selections:
     \[ \sigma_{c_1 \wedge \ldots \wedge c_n}(R) \equiv \sigma_{c_1}(\ldots \sigma_{c_n}(R)) \]
     Allows us to combine several selections into one selection
  2. Commutation of Selections:
     \[ \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \]
     Allows us to test selection conditions in either order
RA Equivalences: Projections

- One important equivalence involves projections:
  - Cascading of Projections:

\[
\pi_{a_1}(R) \equiv \pi_{a_1}(\ldots(\pi_{a_n}(R)))
\]

This says that successively eliminating columns from a relation is equivalent to simply eliminating all but the columns retained by the final projection!
RA Equivalences: Cross-Products and Joins

- Two important equivalences involve cross-products and joins:

  1. Commutative Operations:

     \[(R \times S) \equiv (S \times R)\]

     \[(R \bowtie S) \equiv (S \bowtie R)\]

     This allows us to choose which relation to be the inner and which to be the outer!
RA Equivalences: Cross-Products and Joins

- Two important equivalences involve cross-products and joins:

2. Associative Operations:

\[ R \times (S \times T) \equiv (R \times S) \times T \]
\[ R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T \]

It follows: \[ R \bowtie (S \bowtie T) \equiv (T \bowtie R) \bowtie S \]

This says that regardless of the order in which the relations are considered, the final result is the same!

This *order-independence* is fundamental to how a query optimizer generates alternative query evaluation plans.
RA Equivalences: Selections, Projections, Cross Products and Joins

- **Selections with Projections:**

\[ \pi_a (\sigma_c (R)) \equiv \sigma_c (\pi_a (R)) \]

This says we can commute a selection with a projection if the selection involves only attributes retained by the projection!

- **Selections with Cross-Products:**

\[ R \bowtie_c T \equiv \sigma_c (R \times S) \]

This says we can combine a selection with a cross-product to form a join (as per the definition of a join)!
RA Equivalences: Selections, Projections, Cross Products and Joins

- Selections with Cross-Products and with Joins:

\[ \sigma_c(R \times S) \equiv \sigma_c(R) \times S \]

\[ \sigma_c(R \bowtie S) \equiv \sigma_c(R) \bowtie S \]

**Caveat:** The attributes mentioned in \( c \) must appear only in \( R \) and *NOT* in \( S \)

This says we can commute a selection with a cross-product or a join if the selection condition involves only attributes of one of the arguments to the cross-product or join!
RA Equivalences: Selections, Projections, Cross Products and Joins

- Selections with Cross-Products and with Joins (Cont’d):

\[ \sigma_c(R \times S) \equiv \sigma_{c_1 \land c_2 \land c_3}(R \times S) \]

\[ \equiv \sigma_{c_1} (\sigma_{c_2}(\sigma_{c_3}(R \times S))) \]

\[ \equiv \sigma_{c_1} (\sigma_{c_2}(R) \times \sigma_{c_3}(S)) \]

This says we can push part of the selection condition \( c \) ahead of the cross-product!

This applies to joins as well!
RA Equivalences: Selections, Projections, Cross Products and Joins

- **Projections with Cross-Products and with Joins:**

  $\pi_a(R \times S) \equiv \pi_{a_1}(R) \times \pi_{a_2}(S)$

  $\pi_a(R \bowtie_c S) \equiv \pi_{a_1}(R) \bowtie_c \pi_{a_2}(S)$

  $\pi_a(R \bowtie_c S) \equiv \pi_a(\pi_{a_1}(R) \bowtie_c \pi_{a_2}(S))$

Intuitively, we need to retain only those attributes of $R$ and $S$ that are either mentioned in the join condition $c$ or included in the set of attributes $a$ retained by the projection.
How to Estimate the Cost of Plans?

- Now that correctness is ensured, how can the DBMS estimate the costs of various plans?

Canonical form

\[ \text{Reserves} \quad \text{Sailors} \]
Next Class

Queries

Query Optimization and Execution

Relational Operators

Files and Access Methods

Buffer Management

Disk Space Management

Transaction Manager

Lock Manager

Recovery Manager

DB

Continue…