Midterm Review

March 27, 2017
Overview

• Relational Algebra & Query Evaluation
• Relational Algebra Rewrites
• Index Design / Selection
• Physical Layouts
Relational Algebra & Query Evaluation
# Relational Algebra

<table>
<thead>
<tr>
<th>Operation</th>
<th>Sym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>$\sigma$</td>
<td>Select a subset of the input rows</td>
</tr>
<tr>
<td>Projection</td>
<td>$\pi$</td>
<td>Delete unwanted columns</td>
</tr>
<tr>
<td>Cross-product</td>
<td>$\times$</td>
<td>Combine two relations</td>
</tr>
<tr>
<td>Set-difference</td>
<td>$-$</td>
<td>Tuples in Rel 1, but not Rel 2</td>
</tr>
<tr>
<td>Union</td>
<td>$U$</td>
<td>Tuples either in Rel 1 or in Rel 2</td>
</tr>
</tbody>
</table>

**Also:** Intersection, **Join**, Division, Renaming (Not essential, but very useful)
SELECT [DISTINCT]
    target
FROM source
WHERE cond1
GROUP BY ...
HAVING cond2
ORDER BY order
LIMIT lim
UNION nextselect

U

lim

nextselect

distinct

order by

target (\pi)

cond2 (\sigma)

agg

cond1 (\sigma)

source (x, \bowtie)
GetNext()

Relation
Read One Line from File
Split Line into Fields
Parse Field Types
Return Tuple

What is the Working Set Size?
GetNext()

Projection ($\pi$)

Read One Tuple

Compute Projected Attributes

Return Tuple

What is the Working Set Size?
GetNext()

Selection ($\sigma$)

Read One Tuple

Test Condition

Reject Tuple  Return Tuple

What is the Working Set Size?
GetNext()

Union (U)

Read One Tuple from R

R Empty?

Read One Tuple from S → Return Tuple

What is the Working Set Size?
GetNext()

*Nested Loop Join/Cross (⨉)*

Read (and save) One Tuple from R

Read One Tuple from S

S Empty?

Construct Joint Tuple From S and last read from R

Reset S (Close(), Open())

What is the Working Set Size?

but…

Is there a saved tuple?

N

Y
Memory Conscious Algorithms

- Join
  - NLJ has a small working set (but is slow)
- GB Aggregate
  - Working Set $\sim$ # of Groups
- Sort
  - Working Set $\sim$ Size of Relation
Implementing: Joins

Solution 1 (Nested-Loop)

For Each (a in A) { For Each (b in B) { emit (a, b); } 

A  

B
Implementing: Joins

Solution 2 (Block-Nested-Loop)

1) Partition into Blocks  2) NLJ on each pair of blocks
Implementing: Joins

Solution 3 (Index-Nested-Loop)

Like nested-loop, but use an index to make the inner loop much faster!
Implementing: Joins

Solution 4 (Sort-Merge Join)

Keep iterating on the set with the lowest value. When you hit two that match, emit, then iterate both

A
1
2
3
5

B
1
4
5
6

Keep iterating on the set with the lowest value. When you hit two that match, emit, then iterate both.
Implementing: Joins

Solution 5 (2-Pass Hash)

1) Build a hash table on both relations

2) In-Memory Nested-Loop Join on each hash bucket

A

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

B

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

Hash

Nested-Loop
Implementing: Joins

Solution 6 (1-Pass Hash)

Keep the hash table in memory

(essentially a more efficient nested loop join)
Relational Algebra Rewrites
RA Equivalencies

Selection

\[ \sigma_{c_1 \land c_2}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(R)) \]  
(Decomposable)

\[ \sigma_{c_1 \lor c_2}(R) \equiv \delta(\sigma_{c_1}(R) \cup \sigma_{c_2}(R)) \]  
(Decomposable)

\[ \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \]  
(Commutative)

Projection

\[ \pi_a(R) \equiv \pi_a(\pi_{a \cup b}(R)) \]  
(Idempotent)

Cross Product (and Join)

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

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

Try It: Show that \( R \times (S \times T) \equiv T \times (R \times S) \)
Selection and Projection

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

Selection commutes with Projection
(but only if attribute set \(a\) and condition \(c\) are \emph{compatible})

\(a\) must include all columns referenced by \(c\)

Show that

\[ \pi_a(\sigma_c(R)) \equiv \pi_a(\sigma_c(\pi_{a \cup \text{cols}(c)}(R))) \]

When is this rewrite a good idea?
Join

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

Selection combines with Cross Product to form a Join as per the definition of Join.
(Note: This only helps if we have a join algorithm for conditions like \( c \))

Show that

\[ \sigma_{(R.B=S.B) \land (R.A>3)}(R \times S) \equiv \sigma_{(R.A>3)}(R \bowtie_{(R.B=S.B)} S) \]

When is this rewrite a good idea?
Selection and Cross Product

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

Selection commutes with Cross Product (but only if condition \(c\) references attributes of \(R\) exclusively)

Show that

\[
\sigma_{(R.B=S.B) \land (R.A>3)}(R \times S) \equiv \sigma_{(R.A>3)}(R) \Join_{(R.B=S.B)} S
\]

When is this rewrite a good idea?
Projection and Cross Product

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

Projection commutes (distributes) over Cross Product (where \(a_1\) and \(a_2\) are the attributes in \(a\) from \(R\) and \(S\) respectively)

Show that

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

(under what condition)

How can we work around this limitation?

\[ \pi_a((\pi_{a_1 \cup (cols(c) \cap cols(R))}(R)) \bowtie_c (\pi_{a_2 \cup (cols(c) \cap cols(S))(S)})) \]

When is this rewrite a good idea?
RA Equivalencies

Union and Intersections are **Commutative** and **Associative**

Selection and Projection both commute with both Union and Intersection

When is this rewrite a good idea?
Index Design / Selection
Indexes

Clustered Index

Unclustered Index (Secondary Index)
Indexes

How the Data is Organized

- ISAM
- B+Tree
- Other Tree-Based
- Hash Table
- Other Hash-Based
- Other…

How the Data is Laid Out

- In the Index
  - Clustered
- Outside of the Index
  - Sorted
  - Heap
Access Paths

\[ \sigma_{C_1 \text{ AND } C_2} \]

Can we “simplify” this condition
Access Paths

\[ \sigma_{C_1 \text{ AND } C_2} \]

\[ R \]

= 

\[ \bbox[red]{\sigma_{C_1}} \]

\[ \bbox[red]{\sigma_{C_2}} \]

\[ R \]

Index (C_1)
Access Paths

\[ \sigma_{A > 1 \text{ AND } A < 10} \]

Index

\[ A \in (1,10) \]
Access Paths

How could we compute this if we had an index on S.B?

Foreach r in R
  Foreach s in IndexLookup(S, B=r.b)
  Emit(r \times s)

What are the Working Set Size & IO Cost?
The ISAM Datastructure

Leaf Pages contain \(<K, RID>\) or \(<K, Record>\) pairs

What is the...
• Lookup Cost?
• Insert Cost?
• Delete Cost?
B+ Trees

Search proceeds as in ISAM via key comparisons

Find 5.     Find 15.     Find [24, ∞)

What is the…
• Lookup Cost?
• Insert Cost?
• Delete Cost?
Static Hashing

Primary Bucket Pages
(Contiguous)

Overflow Pages
(Linked List)

What is the…
• Lookup Cost?
• Insert Cost?
• Delete Cost?
Dynamic Hashing

- **Situation:** A bucket becomes full
  - **Solution:** Double the number of buckets!
  - **Expensive!** (N reads, 2N writes)
- **Idea:** Add one level of indirection
  - A directory of pointers to (noncontiguous) bucket pages.
  - Doubling just the directory is much cheaper.
  - Can we double only the directory?
Dynamic Hashing

Dir entries not being split point to the same bucket

What is the…
• Lookup Cost?
• Insert Cost?
• Delete Cost?

gd = 3

32,16
A (ld=3)

1,5,21,13
B (ld=2)

10
C (ld=2)

15,7,19
D (ld=2)

4,12,20
A2 (ld=3)
2.4.6 Forecasting Revenue Change

This query quantifies the amount of revenue increase that would have resulted from eliminating certain company-wide discounts in a given percentage range in a given year. Asking this type of "what if" query can be used to look for ways to increase revenues.

2.4.6.1 Business Question

The Forecasting Revenue Change Query considers all the lineitems shipped in a given year with discounts between DISCOUNT - 0.01 and DISCOUNT + 0.01. The query lists the amount by which the total revenue would have increased if these discounts had been eliminated for lineitems with l_quantity less than quantity. Note that the potential revenue increase is equal to the sum of \( l_{\text{extendedprice}} \times l_{\text{discount}} \) for all lineitems with discounts and quantities in the qualifying range.

2.4.6.2 Functional Query Definition

```sql
select
  sum(l_{\text{extendedprice}} \times l_{\text{discount}}) as revenue
from
  lineitem
where
  l_{\text{shipdate}} \geq \text{date '}[\text{DATE}]' \\
  \text{and } l_{\text{shipdate}} < \text{date '}[\text{DATE}]' + \text{interval '1' year} \\
  \text{and } l_{\text{discount}} \text{ between '[DISCOUNT] - 0.01 and [DISCOUNT] + 0.01} \\
  \text{and } l_{\text{quantity}} < [\text{QUANTITY}];
```

2.4.6.3 Substitution Parameters

Values for the following substitution parameters must be generated and used to build the executable query text:
1. DATE is the first of January of a randomly selected year within [1993 .. 1997];
2. DISCOUNT is randomly selected within [0.02 .. 0.09];
3. QUANTITY is randomly selected within [24 .. 25].

2.4.6.4 Query Validation

For validation against the qualification database the query must be executed using the following values for substitution parameters and must produce the following output data:

Values for substitution parameters:
1. DATE = 1994-01-01;
2. DISCOUNT = 0.06;
3. QUANTITY = 24.

2.4.6.5 Sample Output

<table>
<thead>
<tr>
<th>REVENUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>123141078.23</td>
</tr>
</tbody>
</table>

What features are interesting?
Which Index/Layout?

select  
    l_orderkey,  
    sum(l_extendedprice*(1-l_discount)) as revenue,  
    o_orderdate,  
    o_shippriority  
from  
    customer,  
    orders,  
    lineitem  
where  
    c_mktsegment = '[SEGMENT]'  
    and c_custkey = o_custkey  
    and l_orderkey = o_orderkey  
    and o_orderdate < date '[DATE]'  
    and l_shipdate > date '[DATE]'  
    group by  
    l_orderkey,  
    o_orderdate,  
    o_shippriority  
    order by  
    revenue desc,  
    o_orderdate;

What features are interesting?
Physical Layouts
Data Organization

• How do we store data?
  • How are records represented on-disk? (Serialization)
  • How are records stored within a page?
  • How are pages organized in a file?
  • What other metadata do we need?
• Our solutions must also be persisted to disk.
Record (Tuple) Formats

- Fixed Length Records

Base Address (B)  Address B + L1 + L2

Record information stored in System Catalog

What are some advantages/disadvantages of storing records this way?
Record (Tuple) Formats

- Delimited Records

Number of Fields

Delimiters

What are some advantages/disadvantages of storing records this way?
Record (Tuple) Formats

- Self-Describing Records

Array of Field Offsets

What are some advantages/disadvantages of storing records this way?
What are advantages/disadvantages of these formats?
Page Formats

Variable Size Records

What are advantages/disadvantages of this format?
Files of Records

IO is done at the Page/Block level

... but queries are done at the Record level

File: A collection of pages of records that must support:

Insert/Delete/Update a record
Read a record (using record ID)
Scan all records (possibly with some condition)
Unordered (Heap) Files

Store records in no particular order

Disk pages are allocated/freed as file grows and shrinks

Support for record level operations by:
  Keeping track of pages in the file
  Keeping track of free space in each page
  Keeping track of records on each page

This data must be stored somewhere!
Unordered (Heap) Files

Each page contains 2 pointers plus data
Directories are a collection of pages (e.g., a linked list)
Directories point to all data pages
(Entries can include # of free pages)
What are the advantages and disadvantages of each?