

Database Systems (資料庫系統)

December 3, 2008

Lecture #10

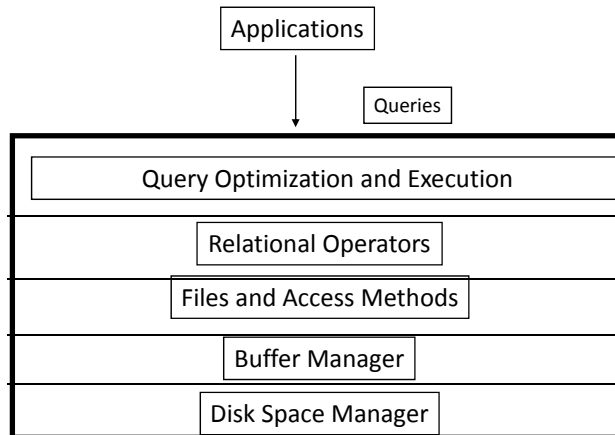
1

Overview of Query Evaluation

Chapter 12

2

Review



3

Outline

- Query evaluation (Overview)
- Relational Operator Evaluation Algorithms (Overview)
- Statistics and Catalogs
- Query Optimization (Overview)
- Example

4

Tables

Sailors(sid, sname, rating, age)

Reserves(sid, bid, day, rname)

5

Overview of Query Evaluation (1)

- What is Query Evaluation?
- What is a good Query Evaluation?
 - Find an efficient plan (minimal disk I/Os)
- How do you find a good plan?

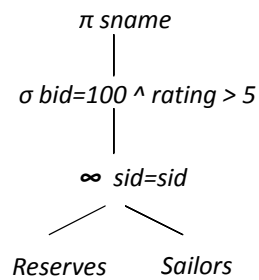
```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid ^ R.bid=100 ^ S.rating>5
```

6

Overview of Query Evaluation (II)

- Step 1: a query is translated into a relational algebra tree
 - σ (selection), π (projection), and \bowtie (join)

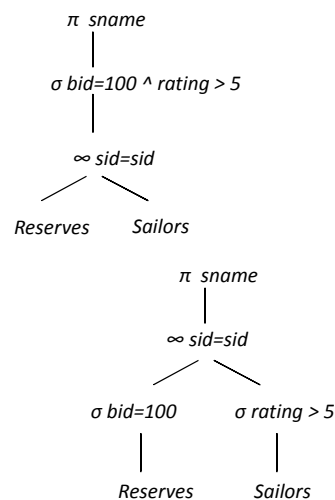
```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid ^ R.bid=100 ^ S.rating>5
```



7

Overview of Query Evaluation (III)

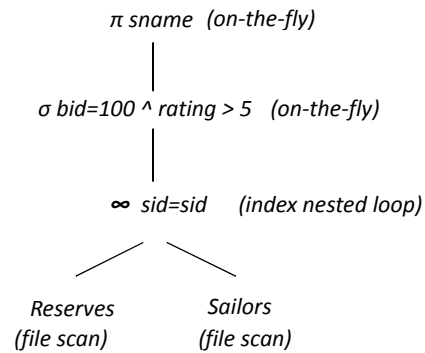
- Step 2: Find a good evaluation plan (Query Optimization)
 - Estimate costs for several alternative equivalent evaluation plans.
 - How to estimate disk I/O cost?
 - access method & evaluation algorithm
 - statistics on table sizes, etc.
 - Different order of evaluations gives different cost.
 - How does it affect cost?
 - Push selection (bid=100) before join)



8

Overview of Query Evaluation (IV)

- (continue step 2)
 - An evaluation plan is consisted of choosing access method & evaluation algorithm.
 - Selecting an access method to retrieve records for each table on the tree (e.g., file scan, index search, etc.)
 - Selecting an evaluation algorithm for each relational operator on the tree (e.g., index nested loop join, sort-merge join, etc.)



9

Overview of Query Evaluation (V)

- Two main issues in query optimization:
 - For a given query, what plans are considered?
 - Consider a few plans (considering all is too many & expensive), and find the one with the cheapest (estimated) cost.
 - How is the cost of a plan estimated?
 - Consider:
 - Reserves \bowtie reserves.sid = sailors.sid Sailors (assume simple nested loop join)

```

Foreach tuple r in reserves
  Foreach tuple s in sailors
    If (r.sid = s.sid) then add <r,s> to the results
          
```

 - Sailors \bowtie ($\sigma_{bid=10}$ Reserves)
 - Sailors \bowtie ($\sigma_{bid>5}$ Reserves)

10

Statistics and Catalogs

- Examine catalog table that has data schemas and statistics.
- Need information about the relations and indexes involved.
Catalogs typically contain at least:
 - # tuples (NTuples) and # pages (NPages) for each table.
 - # distinct key values (NKeys) and NPages for each index.
 - Index height, low/high key values (Low/High) for each tree index.
- There are system-wide factors that can also affect cost, such as size of buffer pool, buffer replacement algorithm.

11

Statistics and Catalogs

- Catalogs are updated periodically.
 - Updating whenever lots of data changes; lots of approximation anyway, so slight inconsistency is ok.
- More detailed information (e.g., histograms of the values in some fields) are sometimes stored.
 - They can be used to estimate # tuples matching certain conditions (bid > 5)

12

Relational Operator Evaluation

- There are several alternative algorithms for implementing each relational operator (selection, projection, join, etc.).
- No algorithm is always better (disk I/O costs) than the others. It depends on the following factors:
 - Sizes of tables involved
 - Existing indexes and sort orders
 - Size of buffer pool (Buffer replacement policy)
- Describe (1) common techniques for relational operator algorithms, (2) access path, and (3) details of these algorithms.

13

Some Common Techniques

- Algorithms for evaluating relational operators use some simple ideas repeatedly:
 - Consider: $\sigma_{bid = 10}$ Reserves
 - Indexing: If a selection or join condition is specified, use an index (<bid>) to retrieve the tuples that satisfy the condition.
 - Consider: $\sigma_{bid \neq 10}$ Reserves, $bid = 1 \dots 1000$
 - Iteration: Sometimes, faster to scan all tuples even if there is an index. Sometimes, scan the data entries in an index instead of the table itself. (π_{bid} Reserves).
 - Consider: $\pi_{sid, rname}$ Reserves
 - Partitioning: By using sorting or hashing, partition the input tuples and replace an expensive operation by similar operations on smaller inputs

14

Access Paths

- An access path is a method of retrieving tuples.
 - Note that every relational operator takes one or two tables as its input.
 - There are two possible methods: (1) file scan, or (2) index that matches a selection condition.
- Can we use an index for a selection condition? How does an index match a selection condition?
 - Selection condition can be rewritten into Conjunctive Normal Form (CNF), or a set of terms (conjuncts) connected by \wedge (and).
 - Example: $(\text{rname}='Joe') \wedge (\text{bid} = 5) \wedge (\text{sid} = 3)$
 - Intuitively, an index matching conjuncts means that it can be used to retrieve (just) tuples that satisfy the conjunct.

15

Access Paths for Tree Index

- A tree index matches conjuncts that involve only attributes in a prefix of its index search key.
 - E.g., Tree index on $\langle a, b, c \rangle$ matches the selection condition $(a=5 \wedge b=3)$, $(a=5 \wedge b>6)$, but not $(b=0)$.
 - Tree index on $\langle a, b, c \rangle$: $\langle a_0, b_0, c_0 \rangle$, $\langle a_0, b_0, c_1 \rangle$, $\langle a_0, b_0, c_2 \rangle$, ..., $\langle a_0, b_1, c_0 \rangle$, $\langle a_0, b_1, c_1 \rangle$, ..., $\langle a_1, b_0, c_0 \rangle$, ...
 - Can match range condition $(a=5 \wedge b>3)$.
- How about $(a=5 \wedge b=3 \wedge c=2 \wedge d=1)$?
 - $(a=5 \wedge b=3 \wedge c=2)$ is called primary conjuncts. Use index to get tuples satisfying primary conjuncts, then check the remaining condition $(d=1)$ for each retrieved tuple.
- How about two indexes $\langle a,b \rangle$ & $\langle c,d \rangle$?
 - Many access paths: (1) use $\langle a,b \rangle$ index, (2) use $\langle c,d \rangle$ index, ...
 - Pick the access path with the best selectivity (fewest page I/Os)

16

Access Paths for Hash Index

- A hash index matches a conjunct that has a term attribute = value for every attribute in the search key of the index.
 - E.g., Hash index on $\langle a, b, c \rangle$ matches $(a=5 \wedge b=3 \wedge c=5)$, but it does not match $(b=3)$, $(a=5 \wedge b=3)$, or $(a>5 \wedge b=3 \wedge c=5)$.
- Compare to Tree Index:
 - Cannot match range condition.
 - Partial matching on primary conjuncts is ok.

17

A Note on Complex Selections

$(\text{day} < 8/9/94 \text{ AND } \text{rname} = \text{'Paul'}) \text{ OR } \text{bid} = 5 \text{ OR } \text{sid} = 3$

- Selection conditions are first converted to conjunctive normal form (CNF):
 $(\text{day} < 8/9/94 \text{ OR } \text{bid} = 5 \text{ OR } \text{sid} = 3) \text{ AND } (\text{rname} = \text{'Paul'} \text{ OR } \text{bid} = 5 \text{ OR } \text{sid} = 3)$
- We only discuss case with no ORs; see text if you are curious about the general case.

18

Selectivity of Access Paths

- Selectivity of an access path is the number of page I/Os needed to retrieve the tuples satisfying the desired condition.
- Why do we want to use the most selective access path (with the fewest page I/Os)?
- What are the possible access paths for selections?
 - Use an index that matches the selection condition.
 - Scan the file records.
 - Scan the index (e.g., π_{bid} Reserves, index on bid)
- Access path using index may not be the most selective!
 - Why not?
 - Example: ($\sigma_{bid \neq 10}$ Reserves, unclustered index on bid).

19

Selection

```
SELECT (*)
FROM Reserves R
WHERE day < 8/9/94 ^ bid=5 ^ sid=3
```

- 1.
- 2.
- 3.

- Consider $\sigma_{day < 8/9/94 \wedge bid=5 \wedge sid=3}$
 - A B+ tree index on day can be used; then, ($bid=5 \wedge sid=3$) must be checked for each retrieved tuple.
 - A hash index on $\langle bid, sid \rangle$ could be used; $day < 8/9/94$ must then be checked.

20

Example

- Use the following example to estimate page I/O cost of different algorithms.
- Sailors(sid:integer, sname:string, rating:integer, age:real)
 - Each Sailor tuple is 50 bytes long
 - A page is 4KB. It can hold 80 sailor tuples.
 - We have 500 pages of Sailors (total 40,000 sailor tuples).
- Reserves(sid:integer, bid:integer, day:dates, rname:string)
 - Each reserve tuple is 40B long
 - A page is 4KB. It can hold 100 reserve tuples.
 - We have 1000 pages of Reserves (total 100,000 reserve tuples).

21

Reduction Factor & Catalog Stats

- Reduction factor is the fraction of tuples in the table that satisfy a given conjunct.
- Example #1:
 - Index H on Sailors with search key <bid>
 - Selection condition (bid=5)
 - Stats from Catalog: NKeys(H) = # of distinct key values = 10
 - Reduction factor =
- Examples #2:
 - Index on Reserves <bid, sid> (not a key, no stats on them)
 - Selection condition (bid=5 ^ sid=3)
 - Typically, can use default fraction of 1/10 for each conjunct.
 - Reduction factor =

22

More on Reduction Factor

- Examples 3:
 - Range condition as (day > 9/1/2002)
 - Index Key T on day
 - Stats from Catalog: High(T) = highest day value, Low(T) = lowest day value.
 - Reduction factor = ()
 - Say: High(T) = 12/31/2002, Low(T) = 1/1/2002
 - Reduction factor = 1/3

23

Using an Index for Selections

- Cost depends on #matched tuples and clustering.
 - Cost of finding qualifying data entries (typically small) plus cost of retrieving records (could be large w/o clustering)
 - Why large? Each matched entry could be on a different page.
 - Assume uniform distribution, 10% of tuples matched (100 pages, 10,000 tuples).
 - With a clustered index on <rid>, cost is little more than 100 I/Os;
 - If unclustered, worse case is 10,000 I/Os.
 - Faster to do file scan => 1,000 I/Os

```
SELECT *  
FROM Reserves R  
WHERE R.rid mod 10 = 1
```

24

Projection

```
SELECT DISTINCT R.sid, R.bid
FROM Reserves R
```

- Projection drops columns not in the select attribute list.
- The expensive part is removing duplicates.
- If no duplicate removal,
 -
 - Given index <sid, bid>, scan the index entries.
- If duplicate removal

25

More on Projection

- Hash-based projection:
 - Hash on <sid, bid> (#buckets = #buffer pages).
 - Load buckets into memory one at a time and eliminate duplicates.

26

Join: Index Nested Loops

```
foreach tuple r in R do
    foreach tuple s in S where r.sid = s.sid do
        add <r, s> to result
```

Reserves (R) ⋈ Sailors (S)

- There exists an index <sid> for Sailors.
- Index Nested Loops: Scan R, for each tuple in R, then use index to find matching tuple in S.
- Say we have unclustered static hash index <sid> in Sailors.

What is the cost of join operation?

- Scan R = 1000 I/Os
- R has 100,000 tuples. For each R tuple, retrieve index page (1.2 I/Os on average for hashing) and data page (1 I/O).
- Total cost = 1,000 + 100,000 * (1.2 + 1) = 221,000 I/Os.

27

Join: Sort-Merge

- It does not use any index.
- Sort R and S on the join column
- Scan sorted lists to find matches, like a merge on join column
- Output resulting tuples.
- More details about Sort-Merge in Chapter 14.

28

Example of Sort-Merge Join

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>

<i>sid</i>	<i>bid</i>	<i>day</i>	<i>rname</i>

- The cost of a merge sort is $2 * M \log_{(B-1)} M$, whereas $M = \#pages$, $B = \text{size of buffer}$.
 - $B-1$ is quite large, so $\log_{(B-1)} M$ is generally just 2.
- Total cost = cost of sorting R & S + Cost of merging = $2 * 2 * (1000 + 500) + (1000 + 500) = 7500$ I/Os. (a lot less than index nested loops join!)

29

Index Nested Loop Join vs. Sort-Merge Join

Reserves (R) ⋈ Sailors (S)

- Sort-merge join does not require a pre-existing index, and ...
 - Performs better than index-nested loop join.
 - Resulting tuples sorted on *sid*.
- Index-nested loop join has a nice property: incremental.
 - The cost proportional to the number of Reserves tuples.
 - Cost = $\#tuples(R) * \text{Cost}(\text{accessing index+record})$
 - Say we have very selective condition on Reserves tuples.
 - Additional selection: $R.bid=101$
 - Cost small for index nested loop, but large for sort-merge join (sort Sailors & merging)
 - Example of considering the cost of a query (including the selection) as a whole, rather than just the join operation → query optimization.

30

Other Relational Operators

- Discussed simple evaluation algorithms for
 - Projection, Selection, and Join
- How about other relational operators?
 - Set-union, set-intersection, set-difference, etc.
 - The expensive part is duplicate elimination (same as in projection).
 - How to do $R1 \cup R2$?
- SQL aggregation (group-by, max, min, etc.)
 - Group-by is typically implemented through sorting (without search index on group-by attribute).
 - Aggregation operators are implemented using counters as tuples are retrieved.

31

Query Optimization

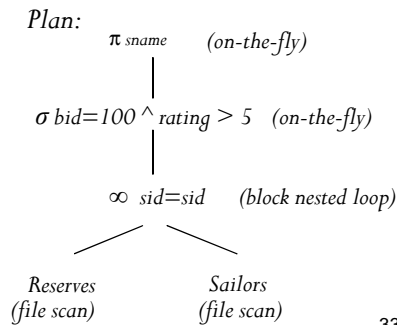
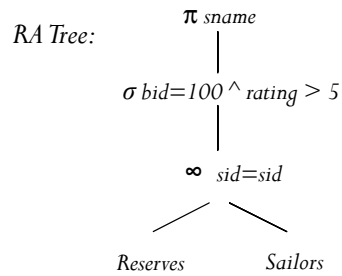
- Find a good plan for an entire query consisting of many relational operators.
 - So far, we have seen evaluation algorithms for each relational operator.
- Query optimization has two basic steps:
 - Enumerate alternative plans for evaluating the query
 - Estimating the cost of each enumerated plan & choose the one with lowest estimated cost.
 - Consult the catalog for estimation.
 - Estimate size of result for each operation in tree (reduction factor).
- More details on query optimization in Chapter 15.

32

Motivating Example

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

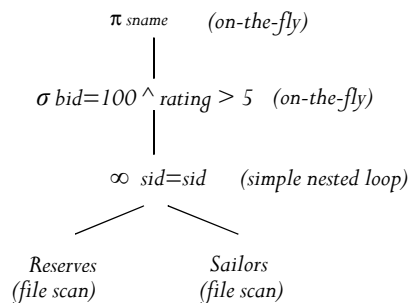
- Cost: 1000+1000*500 I/Os
- Misses several opportunities: selections could have been 'pushed' earlier, no use is made of any available indexes, etc.
- *Goal of optimization:* To find more efficient plans that compute the same answer.



33

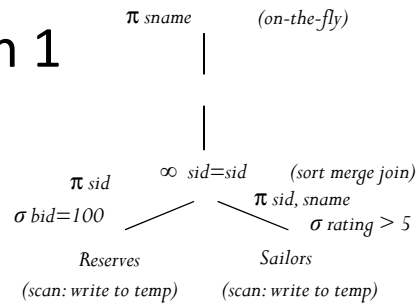
Pipeline Evaluation

- How to avoid the cost of physically writing out intermediate results between operators?
 - Example: between join and selection
 - Use pipelining, each join tuple (produced by join) is (1) checked with selection condition, and (2) projected out on the fly, before being physically written out (materialized).



34

Alternative Plan 1 (No Indexes)

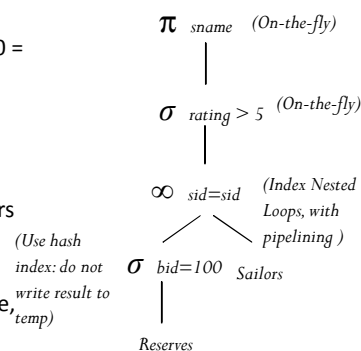


- Main difference: push selects.
- With 5 buffers, cost of plan:
 - Scan Reserves (1000 pages) for selection + write temp T1 (10 pages, if we have 100 boats, uniform distribution).
 - Scan Sailors (500 pages) for selection + write temp T2 (250 pages, if we have 10 ratings, uniform distribution).
 - Sort T1 ($2 \times 2 \times 10$), sort T2 ($2 \times 4 \times 250$), merge (10+250)
 - Total: $(1000 + 500 + 10 + 250) + (40 + 2000 + 260) = 4060$ page I/Os.
- If we used BNL join, join cost = $10 + 4 \times 250$, total cost = 2770.
- If we 'push' projection, T1 has only sid, T2 only sid and sname:
 - T1 fits in 3 pages, cost of BNL drops to under 250 pages, total < 2000.

35

Alternative Plan 2 With Indexes

- With clustered index on bid of Reserves, we get $100,000/100 = 1000$ tuples on $1000/100 = 10$ pages.
- Use pipelining (the join tuples are not materialized).
- Join column *sid* is a key for Sailors.
- Projecting out unnecessary fields from Sailors may not help.
 - Why not? Need Sailors.sid for join.
- Not to push *rating*>5 before the join because,
 - Want to use *sid* index on Sailors.
- Cost: Selection of Reserves tuples (10 I/Os); for each,
 - Must get matching Sailors tuple (1000×1.2); total 1210 I/Os.



36

Next 2 Lectures

- External sorting
- Evaluation of relational operators
- (Relational query optimizer)

37