Overview of Query Evaluation

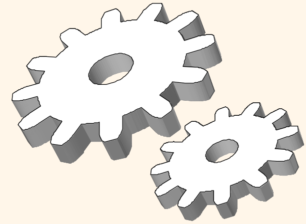
Chapter 12

Overview of Query Evaluation

- *Query Plan:*
  - *Tree of relational algebra operators*
  - *with choice of algorithm for each operator.*
- Example: What are the names of sailors who have reserved boat 103
  - What are the operators

```
SELECT  S.name
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

Overview of Query Evaluation

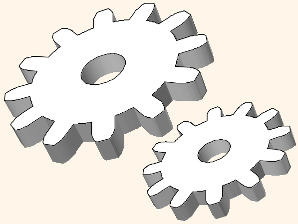


- Two main issues in query optimization:
  - For a given query, what plans are considered?
    - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?

- Ideally: Want to find best plan.
  - Practically: Avoid worst plans!

- Each operator is typically implemented using a \`pull' interface: when an operator is \`pulled' for the next output tuples, it \`pulls' on its inputs and computes them.

Relational Operations



- We will consider how to implement:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Join* ($\bowtie$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
  - *Aggregation* (SUM, MIN, etc.) and GROUP BY
- Since each op returns a relation, ops can be *composed*! After we cover the operations, we will discuss how to *optimize* queries formed by composing them.

Some Common Techniques

- Algorithms for evaluating relational operators use some simple ideas extensively:
  - Indexing: Can use WHERE conditions to retrieve small set of tuples (selections, joins)
  - Iteration: Sometimes, faster to scan all tuples even if there is an index. (And sometimes, we can scan the data entries in an index instead of the table itself.)
  - Partitioning: By using sorting or hashing, we can partition the input tuples and replace an expensive operation by similar operations on smaller inputs.

Statistics and Catalogs

- Need information about the relations and indexes involved. ***Catalogs*** typically contain at least:
  - # tuples (NTuples) and # pages (NPages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.

Statistics and Catalogs

- Catalogs are updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.
- More detailed information (e.g., histograms of the values in some field) are sometimes stored.

A Note on Complex Selections

(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3

- Selection conditions are first converted to *conjunctive normal form* (CNF):

  (day<8/9/94 OR bid=5 OR sid=3) AND
  (rname='Paul' OR bid=5 OR sid=3)

- We only discuss case with no ORs; see text if you are curious about the general case.

Access Paths

- An access path is a method of retrieving tuples:
  - File scan, or index that *matches* a selection (in the query)
- A tree index *matches* (a conjunction of) terms that involve only attributes in a *prefix* of the search key.
  - E.g., Tree index on <*a, b, c*> matches the selection *a*=5 AND *b*=3, and *a*=5 AND *b*>6, but not *b*=3.
- A hash index *matches* (a conjunction of) terms that has a term *attribute = value* for every attribute in the search key of the index.
  - E.g., Hash index on <*a, b, c*> matches *a*=5 AND *b*=3 AND *c*=5; but it does not match *b*=3, or *a*=5 AND *b*=3, or *a*>5 AND *b*=3 AND *c*=5.

The Selection Operator

- *Most selective access path:* An index or file scan that we estimate will require the fewest page I/Os.
- Find the *most selective access path*, retrieve tuples using it, and apply any remaining terms that don't match the index:
  - Terms that match this index reduce the number of tuples *retrieved*; other terms are used to discard some retrieved tuples, but do not affect number of tuples/pages fetched.
  - Consider *day<8/9/94 AND bid=5 AND sid=3*. A B+ tree index on *day* can be used; then, *bid=5* and *sid=3* must be checked for each retrieved tuple. Similarly, a hash index on <*bid, sid*> could be used; *day<8/9/94* must then be checked.

General Selections (CNF Form)

- First approach: Find the *most selective access path*, retrieve tuples using it, and apply any remaining terms that don't match the index:
  - Consider *day<8/9/94 AND bid=5 AND sid=3*. A B+ tree index on *day* can be used; then, *bid=5* and *sid=3* must be checked for each retrieved tuple. Similarly, a hash index on <*bid, sid*> could be used; *day<8/9/94* must then be checked.

- Second approach Get sets of rids of data records using each matching index.
  - Then *intersect* these sets of rids
  - Retrieve the records and apply any remaining terms.
  - Consider *day<8/9/94 AND bid=5 AND sid=3*. If we have a B+ tree index on *day* and an index on *sid*, we can retrieve rids of records satisfying *day<8/9/94* using the first, rids of recs satisfying *sid=3* using the second, intersect, retrieve records and check *bid=5*.

The Selection Operator: Reduction factor

- *Reduction factor.* The fraction of tuples in a table that satisfy a given conjunct
  - When there are several primary conjuncts, the total reduction factor is the product of all reduction factors (approximately)
- If there is no available information about the reduction factor, we can assume either uniform distribution, or simply reduction factor is set to a default value (0.1)
  - More sophisticated techniques use histograms
- Based on the reduction factor, we may decide upon several index choices

Using an Index for Selections

- Cost depends on #qualifying tuples, and clustering.
  - Cost of finding qualifying data entries (typically small) plus cost of retrieving records (could be large w/o clustering).
  - In example, assuming uniform distribution of names, about 10% of tuples qualify (100 pages, 10000 tuples). With a clustered index, cost is little more than 100 I/Os; if unclustered, up to 10000 I/Os!

```
SELECT  *
FROM    Reserves R
WHERE   R.rname < ‘C%’
```

The Selection Operation

```
SELECT  R.rating
FROM    Reserves R
WHERE   R.attr op value
```

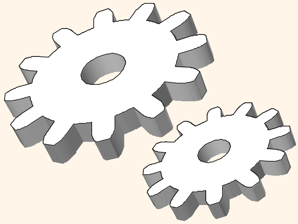
- **No Index, Unsorted Data**
  - Most selective access path is *"file scan"*. Cost is O(M) where M is the file size in pages
- **No Index, Sorted Data**
  - Most selective access path is *"binary search"*. Cost is $O(\log_2 M)$ + number of pages that contains qualifying tuples
- **Clustered B+-tree**
  - Using the clustered index would be best in case of range search. Cost is 2-3 I/Os to identify the start record + number of pages that contain qualifying tuples
  - Good for equality search in case hash index is not available. Cost is 2 -3 I/Os

The Selection Operation

- Unclustered B+-tree
  - Works for equality search for keys in case hash index is not available. Cost is 2 -3 I/Os A worst case scenario is that every single qualified tuple results in one page I/O
  - A refinement for the unclustered index

    1. Find qualifying data entries.
    2. Sort the rid's of the data entire based on the page identifiers.
    3. Fetch rids in order.
- Clustered Hash Index
  - Best for equality search. Cost is 1-2 I/Os + Number of pages with qualifying tuples
- Unclustered Hash Index
  - Used for equality search. Cost is 1-2 I/Os + Number of qualifying tuples

Projection



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

- Projection is: (1) Dropping unwanted columns, and (2) Removing duplicates
- The expensive part is removing duplicates.
  - SQL systems don't remove duplicates unless the keyword DISTINCT is specified in a query.
- If no duplicate elimination is needed, an iteration is performed either on the table or an index whose key contains all the projection fields

Projection with duplicate elimination

- *Sorting Approach:* Sort on <sid, bid> and remove duplicates. (Can optimize this by dropping unwanted information while sorting.)
- *Hashing Approach:* Hash on <sid, bid> to create partitions. Load partitions into memory one at a time, build in-memory hash structure, and eliminate duplicates.
- If there is an index with both R.sid and R.bid in the search key, may be cheaper to sort data entries!

Discussion of Projection

- Sort-based approach is the standard; better handling of skew and result is sorted.
- If an index on the relation contains all wanted attributes in its search key, can do *index-only* scan.
  - Apply projection techniques to data entries (much smaller!)
- If an ordered (i.e., tree) index contains all wanted attributes as *prefix* of search key, can do even better:
  - Retrieve data entries in order (index-only scan), discard unwanted fields, compare adjacent tuples to check for duplicates.