ECS 165B: Database System Implementation
Lecture 13

UC Davis

April 26, 2010

Acknowledgements: portions based on slides by Raghu Ramakrishnan and Johannes Gehrke.

Class Agenda

- Last time:
  - Finish with external sorting
  - Physical join operators
- Today:
  - More physical operators: selection, projection, duplicate elimination, aggregates
  - Quiz review
- Reading
  - Chapter 14 of Ramakrishnan and Gehrke (or Chapter 13 of Silberschatz et al)

Announcements

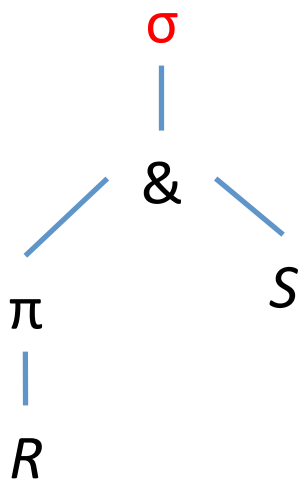
Grades for Part 1: out tonight

Quiz #1 in class on Wednesday

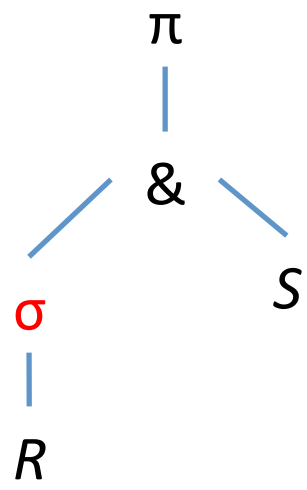
Evaluation of Relational Operations, cont.

Selections

- Selection above another operator: just evaluate the selection condition



- Selection above a source relation (common case): can use an index, if available



Using an Index for Selections

- Cost depends on # of qualifying tuples ("selectivity") and clustering
  - Cost of finding qualifying data entries (typically small) plus cost of retrieving records (could be large w/o clustering)
- Important refinement for unclustered indices:
  1. Find qualifying data entries
  2. Sort the record ids of the data records to be retrieved
  3. Fetch record ids in order. This ensures that each data page is looked at just once (though # of such pages likely to be higher than with clustering)

Complex Selections (1)

```
select …
from …
where day<8/9/94 and bid=5 and sid=3
```

conjunction of =,<,etc **terms**

- **First approach**: find the *most selective access path*, retrieve tuples using it, then apply any remaining selection conditions
  - *Most selective access path*: an index or file scan that we estimate will require the fewest page I/Os
  - Selection terms matching this index reduce the number of tuples *retrieved* from disk; remaining terms filter the retrieved tuples, but do not affect # of tuples fetched
  - E.g., a B+ tree index on `day` can be used; then, `bid=5` and `sid=3` must be checked for each retrieved tuple. Or, a hash index on <`bid`,`sid`> could be used; `day<8/9/94` must then be checked

Complex Selections (2)

```
select …
from …
where day<8/9/94 and bid=5 and sid=3
```

- **Second approach**: use *intersection of record ids* (if we have 2 or more matching, non-clustered indices)
  - Get sets of rids of data records using each matching index
  - Then *intersect* these sets of rids (we'll discuss intersection soon)
  - Retrieve the records and apply any remaining selection terms
  - e.g., if we have a B+ tree index on `day` and another on `sid`, both using Alternative (2), we can retrieve rids of records satisfying `day<8/9/94` using the first, rids of records satisfying `sid=3` using the second, intersect, retrieve records and check `bid=5`

Duplicate-Eliminating Projection (1)

```
select distinct R.sid, S.bid
from Reserves R
```

- **First approach**: use modified external merge-sort
  - Modify Pass 0 of external sort to eliminate unwanted fields, to save space. Thus, runs of about 2*B* pages are produced (heapsort), but tuples in runs are smaller than input tuples. Size ratio depends on # and size of fields that are dropped.
  - Modify merging passes to eliminate duplicates. Thus, # of result tuples smaller than input. Difference depends on # of duplicates.
  - Cost: in pass 0, read original relation (size *M*), write out same number of smaller tuples. In merging passes, fewer tuples written out in each pass. Using Reserves example, 1000 input pages reduced to 250 in Pass 0 if size ratio is 0.25.

Duplicate-Eliminating Projection (2)

```
select distinct R.sid, S.bid
from Reserves R
```

- **Second approach**: use modified external hash sort
  - *Partitioning phase*: read *R* using one input buffer. For each tuple, discard unwanted fields, apply hash function *h* on all fields to choose one of *B*-1 output buffers. Result is *B*-1 partitions (of tuples with no unwanted fields). Tuples from different partitions guaranteed to be distinct.
  - *Duplicate elimination phase*: for each partition, read and build in-memory hash table, using hash function *h*' (≠ *h*) on all fields, while discarding duplicates. If partition does not fit in memory, can apply hash-based projection algorithm recursively to this partition
  - Cost: for partition, read *R*, write out each tuple, but with fewer fields. This is read in next phase.

Discussion of Projection

- Merge-sort based approach is the standard; better handling of skew and (as a bonus) 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

Set Operations

- `intersect` and cross-product: special cases of join
- `union` (distinct) and `except` are similar; we'll do `union`
- Sort-based approach to `union`:
  - Sort both relations (on all attributes)
  - Scan sorted relations and merge them, discarding duplicates
  - *Alternative*: merge runs from Pass 0 for *both* relations
- Hash-based approach to `union`:
  - Partition *R* and *S* using hash function *h* (on all attributes)
  - For each *S*-partition, build in-memory hash table (using *h'*), scan corresponding *R*-partition and add tuples to table while discarding duplicates

Aggregate Operators (`avg`, `min`, etc)

```
select avg(S.age)
from Sailors S
where S.rating > 2
```

- Without grouping:
  - In general, requires scanning the relation
  - Given index whose search key includes all attributes in the `select` or `where` clauses, can do index-only scan

Aggregate Operators (2)

```
select rating, min(S.age)
from Sailors S
group by rating
```

- With grouping:
  - Sort on `group-by` attributes, then scan relation and compute aggregate for each group. (Can improve upon this by combining sorting and aggregate computation.)
  - Similar approach based on hashing on `group-by` attributes
  - Given tree index whose search key includes all attributes in `select`, `where`, and `group-by` clauses, can do index-only scan; if `group-by` attributes form prefix of search key, can retrieve data entries/tuples in `group-by` order

Impact of Buffering

- If several operations are executing concurrently, estimating the # of available buffer pool pages is guesswork
- *Repeated access patterns* interact with buffer replacement policy
  - e.g., inner relation is scanned repeatedly in Simple Nested Loops Join. With enough buffer pages to hold inner, replacement policy does not matter. Otherwise, MRU is best, LRU is worst (*sequential flooding*).
  - Does replacement policy matter for Block Nested Loops?
  - What about Index Nested Loops?

Summary

- A virtue of relational DBMSs: *queries are composed of a few basic operators*; the implementation of these operators can (and must!) be carefully tuned
- Many alternative implementation techniques for each operator; no universally superior technique for most operators
- Must consider available alternatives for each operation in a query and choose best one based on system statistics, etc. This is part of the broader task of optimizing a query composed of several operators.

Quiz #1 (Wednesday): Whirlwind Review

What Have We Covered So Far in this Class?

- Buffer and file management
- Indexing
- External sorting
- Query evaluation
  - But not query optimization
- Relational calculus, relational algebra, conjunctive queries
- DavisDB Parts 1 and 2; subversion

- *Quiz will cover all of these topics!*
- Study aids: (1) slides, (2) project documentation, (3) review questions in textbook: Ch 8, 9, 10, 12, 13, 14

Buffer and File Management

- Buffer pool
  - pinning
  - replacement policies
- Record formats
  - fixed-length versus variable-length
- Page formats
  - for fixed-length versus variable-length records
- Record files / unordered heap files
  - Schemes for keeping track of free space

DavisDB Part 1

What are the roles of…

- PageFileManager
- FileHandle
- RecordManager
- RecordFileHandle
- RecordFileScan

Indexing

- Clustered versus unclustered
- Alternatives for unclustered indices (alts. (2) vs (3))
- Hash-based indices
- Tree-structured indices
  - ISAM versus B+ tree
  - B+ trees: lookups, range searches, insertions and deletions

DavisDB Part 2

- What are the roles of…
  - IndexManager
  - IndexHandle
  - IndexScan

External Sorting

- External merge sort (2-way, n-way, …)
- Hash-based external sort
- Using B+ trees to sort

Query Evaluation

- Logical versus physical operators
- Join:
  - Simple Nested Loops Join
  - Index Nested Loops Join
  - Block Nested Loops Join
  - Sort-Merge Join
  - Hash Join
- Selection:
  - Index-based selection
- Projection, union, etc:
  - duplicate elimination

Relational calculus, relational algebra, conjunctive queries

- Relational calculus: logic-based query language
- Expressive equivalence of relational calculus and relational algebra
- What is a conjunctive query (SQL, relational calculus, relational algebra)

Miscellaneous

- Subversion!