Evaluating Relational Operators

Linda Wu

(CMPT 354 • 2004-2)

Topics

- Selection operation
- Projection operation
- Set operations
- Aggregate operations
- Join operation
- Impact of buffering

Relational Operations

- We will consider how to implement:
  1. Selection (σ): select a subset of rows from relation
  2. Projection (π): delete unwanted columns from relation
  3. Union ($\cup$): tuples in relation 1 or 2
  4. Set-difference (–): tuples in relation 1 but not in relation 2
  5. Aggregation (SUM, MIN, etc) and GROUP BY
  6. Join ($\triangleright\triangleleft$): combine two relations
     - Intersection($\cap$) and cross-product ($\times$) are implemented as special cases of join

Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: date, *rname*: string)

- Similar to old schema; *rname* added for variations
- Reserves
  - Each tuple is 40 bytes long; 100 tuples per page; 1000 pages
- Sailors
  - Each tuple is 50 bytes long; 80 tuples per page; 500 pages

Selection σ

$\sigma_{R.attr\ \mathbf{op}\ \text{value}}(R)$

- No index on *R.attr*, *R* is not sorted on *attr*
  - Scan the entire relation *R*
  - Add tuples to result if the condition is satisfied
- No index on *R.attr*, *R* is sorted on *attr*
  - Binary search to find the first qualifying tuple
  - From there, scan *R* till the condition is no longer satisfied
- B+ tree index on *R.attr*
  - Search the tree to find the first index pointing to a qualifying tuple of R
  - Scan the leaf pages to retrieve all qualifying data entries, and, the corresponding tuples if Alternative (2), (3)
- Hash index on *R.attr*, op is equality
  - Retrieve the correct bucket page in the index
  - Retrieve qualifying tuples from *R*

Selection σ (Cont.)

- If using an index for selections, the I/O cost depends on # of qualifying tuples and clustering
  - Cost of finding qualifying data entries (typically small) + cost of retrieving tuples (could be large without clustering)
  - Example

    SELECT * FROM Reserves R
    WHERE R.rname < 'C%'

    Assuming uniform distribution of names, about 10% of tuples qualify (100 pages; 10,000 tuples)

    If a clustered B+ tree index on *rname*, cost ≈ 100 I/Os; if an unclustered index, up to 10,000 I/Os (even worst than entire file scan!)

Selection σ (Cont.)

- Important refinement for unclustered indexes
  1. Find qualifying data entries
  2. Sort the rid's of the data records to be retrieved by their page-id component
  3. Fetch rids in order. This ensures that each data page is looked at just once (though # of such pages likely to be higher than with clustering)

  Cost of retrieving tuples = # of pages containing qualifying tuples

Selection σ (Cont.)

- General selection conditions
  - A Boolean combination ($\wedge$,$\vee$) of terms
  - Terms have the form:

    attr op *constant*, or, *attr1* op *attr2*
  - Conditions expressed in conjunctive normal form (CNF)
    - A condition is a collection of conjuncts that are connected by $\wedge$
    - A conjunct contains consist of one or more terms connected by $\vee$
    - A conjunct containing $\vee$ is said to be disjunctive (contain disjunction)

Selection σ (Cont.)

- Selections without disjunction

  Approach 1: find the most selective access path, retrieve tuples using it, and apply any remaining terms that don't match the access path

  - Most selective access path: an index or file scan that is estimated to require the fewest page I/Os
  - Terms that match the access path reduce # of tuples retrieved; other terms used to discard some retrieved tuples
  - *day<8/9/03 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<8/9/03* must then be checked

Selection σ (Cont.)

Approach 2 (if we have 2 or more matching indexes that use Alternatives (2) or (3) for data entries)

- Retrieve the 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 terms
- *day<8/9/03 AND bid=5 AND sid=3*

  If we have a B+ tree index on *day* and an index on *sid*, both using Alternative (2), we can retrieve rids of records satisfying *day<8/9/03* using the first, rids of records satisfying *sid=3* using the second, intersect, retrieve records and check bid=5

* We don't discuss selections with disjunction

Projection π

$$\pi_{\,attr1,\ attr2,\ \ldots\ attrn}\ (R)$$

- Implementing projection
  - Remove unwanted attributes
  - Eliminate duplicates (based on sorting / hashing)

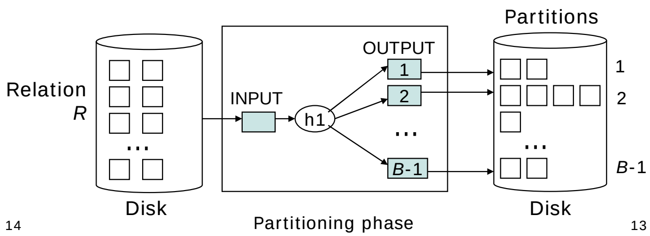
```
SELECT  DISTINCT R.sid, R.bid
FROM    Reserves R
```


Projection π (Cont.)

- Projection based on sorting
  - Modify pass 0 of external merge sort to eliminate unwanted fields
    - Runs of about 2B pages are produced, but tuples in runs are smaller than input tuples (size ratio depends on the total size of dropped fields)
  - Modify merging passes to eliminate duplicates
    - Less result tuples than input (difference depends on # of duplicates)
  - I/O cost
    - In Pass 0, read the original relation (*M* pages), write out the same number of smaller tuples
    - In merging passes, fewer pages are written out in each pass

Projection π (Cont.)

- Projection based on hashing
  - Used when # of buffer pages (*B*) is much larger than # of pages in relation *R*
  - Cost: read *R* for partitioning, write out each tuple with fewer fields (therefore fewer pages); they are read in the next phase, in-memory hash table is written out for each partition



Projection π (Cont.)

- *Phase 1: partitioning*

  Read *R* using one input buffer; for each tuple, discard unwanted fields, apply hash function *h1* to choose one of *B*-1 output buffers
  - Result is B-1 partitions (with no unwanted fields)
  - 2 tuples from different partitions must be distinct
- *Phase 2: duplicate elimination in each partition*

  For each partition: read it in, one page at a time; hash its tuples using *h2* ($\neq$ *h1*), then insert tuples into an in-memory hash table, discard duplicates; write hash table to the result file
  - If a tuple hashes to the same value as an existing tuple, compare the two to check whether duplicates

Projection π (Cont.)

- Sort-based approach is the standard
  - Better handling of non-uniformly distributed values
  - 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

- R ∩ S, R × S, R ∪ S, R – S
- Intersection (∩) and cross-product (×) are implemented as special cases of join
  - ∩: equality on all fields as the join condition
  - × : no join condition
- Implementation of union (∪) and set-difference (–) are similar