

CSE 541: Database Systems I

Query Processing – Basic Operations

Relational Algebra Operators

- Selection (σ)
 - Select a subset of rows from a table (relation)
- Projection (π)
 - Pick a subset of columns
- Set difference
 - $S1 - S2$: return tuples that appear in $S1$ but not in $S2$
- Union
 - $S1 \cup S2$: return tuples in $S1$ and $S2$
- Aggregation
 - Operations such as SUM, MIN and GROUP BY
- Join
 - Combine two relations based on some conditions

Relational Algebra Operators

- Selection (σ)
 - Select a subset of rows from a table (relation)
- Projection (π)
 - Pick a subset of columns
- Set difference
 - $S1 - S2$: return tuples that appear in $S1$ but not in $S2$
- Union
 - $S1 \cup S2$: return tuples in $S1$ and $S2$
- Aggregation
 - Operations such as SUM, MIN and GROUP BY
- Join
 - Combine two relations based on some conditions

Simple Selections

- $\sigma_{R.\text{attribute op value}} R$

- Queries like:

```
SELECT *  
FROM Reserves R  
WHERE R.bid > 100
```

- Q: How to evaluate this query?
- A: Depends on index availability and file organization
 - Index type (tree vs. hash table, clustered vs. unclustered)
 - Whether the table is sorted on selection attribute

Simple Selection ($\sigma_{R.attribute \text{ op value}}$ R)

Case 1 – no index, unsorted relation:

- Scan the whole table
- Cost = M (number of pages in the relation)

Case 2 – no index, relation sorted on selection attribute:

- Use binary search to find the first qualifying tuple
- Then scan until the selection condition no longer holds
- Cost = $O(\log_2 M)$ + retrieval cost (0 to M)

Case 3 – index exists on the selection attribute:

- Use index to find the qualifying tuples
- Then retrieve the actual data records

Using an Index for Selection

Cost depends on the number of qualifying tuples and if the index is clustered

Cost includes:

- Find qualifying records
- Retrieve records (could be large with unclustered index)
- Example: 10000 tuples (occupying 100 pages) qualify
 - Clustered index: cost is a little higher than 100 I/Os
 - For fetching all pages
 - Unclustered index: up to more than 10000 I/Os – 1 I/O per tuple

Refinement under Unclustered Index

1. Find qualifying data entries (Alternatives 2/3)
2. Sort the RIDs of the data records to be retrieved
 - Sort by the page ID field in each RID
3. Fetch RIDs in order

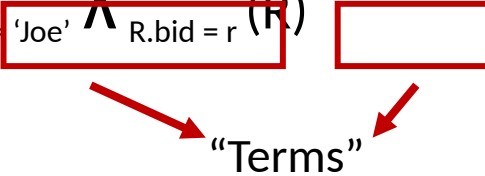
➔ Each data page is looked at just once

General Selections

A general selection condition is a Boolean combination

- Expression using logical connectives \wedge and \vee
- Example

• $R.rname = 'Joe' \text{ AND } r.bid = r \rightarrow \sigma_{R.rname = 'Joe' \wedge R.bid = r} (R)$



- Conjunctive normal form (CNF): a collection of conjuncts connected through the use of the \wedge operator
- Conjunct: one or more terms connected using \vee
 - E.g., $bid = 5 \vee sid = 3$
 - With \vee : “disjunctive” or “contain disjunction”

Selection without Disjunction

Approach 1:

- Retrieve tuples using file scan or a single index that matches some conjuncts
 - Typically the most selective access path, i.e., an index or file scan that we estimate will require the fewest page I/Os
 - Terms that match this index reduce the number of tuples retrieved
- Then apply other conjuncts in the selection to each retrieved tuple
- Example: $\text{day} < 8/9/94 \text{ AND } \text{bid} = 5 \text{ AND } \text{sid} = 3$
 - Use a B+-tree index on day to retrieve tuples
 - Check $\text{bid} = 5$ and $\text{sid} = 3$ for each tuple retrieved

Selection without Disjunction

Approach 2: utilize several indexes

- Assumption: several indexes that contain RIDs in data entries are available (Alternatives 2 and 3)
- 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.
- Example: $\text{day} < 8/9/94$ AND $\text{bid} = 5$ AND $\text{sid} = 3$, and we have index on day and bid
 - Use index on day to retrieve tuples ($S1$), use index on bid to retrieve tuples ($S2$)
 - $S = S1 \cap S2$, and check for $\text{sid} = 3$ for each tuple in S

Selection with Disjunction

Conjuncts with multiple terms connected using \vee

- Example: $\text{day} < 9/9/2015 \vee \text{rname} = \text{'Joe'}$
- If every term has a matching index
 - Use the index for each term
 - Then union the results
- May have to use file scan if one of the terms requires a file scan (matching index not available on all terms)
 - Could still benefit from index
 - Example: $(\text{day} < 9/9/2015 \vee \text{rname} = \text{'Joe'}) \wedge \text{sid} = 3$
 - Use index on sid to find qualifying tuples first, then apply $\text{day} < 9/9/2015 \vee \text{rname} = \text{'Joe'}$ to the qualifying tuples only

Relational Algebra Operators

- Selection (σ)
 - Select a subset of rows from a table (relation)
- Projection (π)
 - Pick a subset of columns
- Set difference
 - $S1 - S2$: return tuples that appear in $S1$ but not in $S2$
- Union
 - $S1 \cup S2$: return tuples in $S1$ and $S2$
- Aggregation
 - Operations such as SUM, MIN and GROUP BY
- Join
 - Combine two relations based on some conditions

Projection

Example query:

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

Relational algebra expression: $\pi_{sid, bid}(R)$ Reserves

Need to do two things:

- Remove unwanted attributes (easy)
- Eliminate duplicate tuples produced (difficult)
- Two basic algorithms: sorting based and hashing based

Projection based on Sorting

Basic algorithm:

1. Scan the relation and produce tuples that contain only the desired attributes
2. Sort the result of step 1
 - With the combination of all its attributes as the key for sorting
3. Scan the result of step 2
 - Compare adjacent tuple and discard duplicates

Projection based on Sorting

An improved algorithm:

- Modify the external sorting algorithm
 1. Modify Pass 0 to eliminate unwanted fields
 - Each runs has about 2B pages
 - Tuples in runs can be smaller than in the input tuple (depends on the number of wanted attributes)
 2. Modify merging passes to eliminate duplicates.
 - The number of result tuples can be smaller than the input table (depends on the number of duplicates)

Projection based on Hashing

Two phases: partitioning and duplicate elimination

Partitioning phase:

- Read input table R using one buffer, using the remaining B-1 buffers for output
 - For each tuple, remove unwanted fields
 - Apply hash function h1 to distribute each tuple into one of the B-1 buffers
- ➔ Two tuples are in the same partition if they are duplicates
- ➔ Two tuples are not duplicates if they are in different partitions

Projection based on Hashing

Two phases: partitioning and duplicate elimination

Duplicate elimination phase:

- Read in each partition and build an in-memory hash table using a different hash function h_2 ($h_2 \neq h_1$)
 - Duplicates are detected and removed during this process
 - Hash to the same value → check keys to make sure
- Write out the tuples in the hash table after processing the entire partition, clear hash table for the next partition

Discussion on Projection

- Sort-based approach is the standard
 - Better handling of skew
 - Result is sorted
- If an index on the relation contains all the wanted attributes in its search key
 - Do index-only scan
 - Apply projection techniques to data entries
- If an ordered index on the relation contains all the wanted attributes as prefix of its search key
 - E.g., $\langle a \rangle$, $\langle a, b \rangle$ are prefixes of $\langle a, b, c \rangle$
 - Do index-only scan, discard unwanted fields
 - Compare adjacent tuples to remove duplicates

Relational Algebra Operators

- Selection (σ)
 - Select a subset of rows from a table (relation)
- Projection (π)
 - Pick a subset of columns
- Set difference
 - $S1 - S2$: return tuples that appear in S1 but not in S2
- Union
 - $S1 \cup S2$: return tuples in S1 and S2
- Aggregation
 - Operations such as SUM, MIN and GROUP BY
- Join
 - Combine two relations based on some conditions

Set Operations

Intersection: special cases of join

- Intersection: equality on all fields as the join condition

Union ($R \cup S$):

- Sorting based: sort both relations, then scan-merge them
- Hashing based
 - Partition R and S using hash function h_1
 - For each S partition
 - Build in-memory hash table using $h_2 \neq h_1$
 - Scan the corresponding R partition: for each R tuple, probe it in the hash table; discard if it is in the hash table, otherwise add it to the table
 - Write out and clear the hash table for the next partition

Aggregate Operation (AVG, MIN, etc)

Case 1 – without grouping:

- Often need to scan the entire relation
- Can do index-only scan if index is available

Case 2 – with grouping:

- Sort on group-by attributes
- Scan relation and compute aggregate for each group
- Similar approach based on hashing on group-by attributes
- If tree index is available, can do index-only scan
- If the group-by attributes form prefix of search key, can retrieve data entries/tuples in group-by order