Outline

- Sorting
- Evaluation of joins
- Evaluation of other operations
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).
  - Consider a selection of the form $gpa > 3.0$ and assume 10% of tuples qualify (100 pages, 10,000 tuples). With a clustered index, cost is little more than 100 I/Os; if unclustered, upto 10,000 I/Os!

- Important refinement for unclustered indexes:
  1. Find qualifying data entries.
  2. Sort the rid’s of the data records to be retrieved.
  3. Fetch rids in order.
Two Approaches to General Selections

- **First approach:** (1) Find the most selective access path, retrieve tuples using it, and (2) apply any remaining terms that don’t match the index on the fly.
  - *Most selective access path:* 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; other terms are used to discard some retrieved tuples, but do not affect number of tuples/pages fetched.
  - Consider \(\text{day}<8/9/94 \ \text{AND bid}=5 \ \text{AND sid}=3\).
    - A B+ tree index on \textit{day} can be used; then, \(\text{bid}=5\) and \(\text{sid}=3\) must be checked for each retrieved tuple.
    - A hash index on \(<\text{bid}, \text{sid}>\) could be used; \(\text{day}<8/9/94\) must then be checked on the fly.
Intersection of Rids

- **Second approach** (if we have 2 or more matching indexes that use Alternatives (2) or (3) for data entries):
  - 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 $\text{day} < 8/9/94 \text{ AND bid}=5 \text{ AND sid}=3$. If we have a B+ tree index on $\text{day}$ and an index on $\text{sid}$, both using Alternative (2), we can:
    - retrieve rids of records satisfying $\text{day} < 8/9/94$ using the first, rids of records satisfying $\text{sid}=3$ using the second,
    - intersect these rids,
    - retrieve records and check $\text{bid}=5$. 
The Projection Operation

- Projection consists of two steps:
  - Remove unwanted attributes (i.e., those not specified in the projection).
  - Eliminate any duplicate tuples that are produced.

- Algorithms: single relation sorting and hashing based on all remaining attributes.

```sql
SELECT DISTINCT R.sid, R.bid
FROM Reserves R
```
Set Operations

- Intersection and cross-product special cases of join.
  - Intersection: equality on all fields.

- Union (Distinct) and Except similar; we’ll do union.

- Sorting based approach to union:
  - Sort both relations (on combination of all attributes).
  - Scan sorted relations and merge them, removing duplicates.

- Hash based approach to union:
  - Partition R and S using hash function $h$.
  - For each S-partition, build in-memory hash table (using $h2$). Scan R-partition. For each tuple, probe the hash table. If the tuple is in the hash table, discard it; o.w. add it to the hash table.
Aggregate Operations (AVG, MIN, etc.)

- **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.

- **With grouping (GROUP BY):**
  - 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.
Summary

- A virtue of relational DBMSs: *queries are composed of a few basic operators*; the implementation of these operators can be carefully tuned.

- 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.
Summary (contd)

- Many 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 state (e.g., memory) and
  - statistics (table size, # tuples matching value k).

- This is part of the broader task of optimizing a query composed of several ops.