Introduction to Data Management

Lecture #22
(Physical DB Design)

Instructor: Mike Carey
mjcarey@ics.uci.edu

It’s the penultimate episode of....

Friday Nights with Databases...!

Brought to you by...
Announcements

- No class Monday! (Awwwwww.....)
- Two HW assignments remain:
  - HW #7: Due next Thursday, May 30\(^{th}\) (5 PM)
    - Physical DB design (for MySQL and beyond)
  - HW #8: Due on Thursday, June 6\(^{th}\) (5 PM)!
    - NoSQL (and NoLateDay due to Endterm timing)
- Today’s plan :
  - Today: Physical DB design (esp. indexing)
  - Next up: NoSQL & Big Data (a la AsterixDB)
    - Not in the textbook, so... See the wiki for readings!

Overview

- After ER design, schema refinement, and the definition of views, we have the **conceptual** and **external** schemas for our database.
- Next step is to choose **indexes**, make clustering **decisions**, and refine the conceptual and external **schemas** (if needed) to meet **performance goals**.
- Start by understanding the **workload**:
  - Most important queries and how often they arise.
  - Most important updates and how often they arise.
  - Desired performance goals for those queries/updates?
Decisions to Be Made Include...

- What indexes should we create?
  - Which relations should have indexes? What field(s) should be their search keys? Should we build several indexes?
- For each index, what kind of an index should it be?
  - B+ tree? Hashed? Clustered? Unclustered?
- Should we make changes to the conceptual schema?
  - Consider alternative normalized schemas? (There are multiple choices when decomposing into BCNF, etc.)
  - Should we ``undo'' some decomposition steps and settle for a lower normal form? ("Denormalization.")
  - Horizontal partitioning, materialized views, replication, ...

Understanding the Workload

- For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes appear in selection/join conditions? (And how selective are those conditions expected to be?)
- For each update in the workload:
  - Which attributes are involved in selection/join conditions? (And how selective are those conditions likely to be?)
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.
Index Classification (Review)

- **Primary vs. secondary:** If index search key contains the primary key, this is called the primary index.
  - Unique index: Search key contains a candidate key.

- **Clustered vs. unclustered:** If the order of data records is the same as, or `close to`, the order of stored data records, we have a clustered index.
  - A table can be clustered on at most one search key.
  - Cost of retrieving data records via an index varies greatly based on whether index is clustered or not!

Clustered vs. Unclustered Indexes (Reminder)

(Read each page once.) (Read more pages – and repeatedly!)
Choice of Indexes (Cont’d.)

- **One approach:** Consider the most important queries in turn. Consider the best query plan using the current indexes, and see if a better plan is possible with an additional index. If so, create it.
  - This means we must understand and see how a DBMS evaluates its queries. *(Query execution plans.)*
  - Let’s start by discussing simple 1-table queries!

- Before creating an index, must also consider its impact on updates in the workload.
  - **Trade-off:** Indexes can make queries go faster, but updates will become slower. *(Indexes require disk space, too.)*

Index Selection Guidelines

- **Attributes in WHERE clause are candidates for index keys.**
  - Exact match condition \( \rightarrow \) hashed index (or B+ tree if not).
  - Range query \( \rightarrow \) B+ tree index.
    - Clustering especially useful for range queries, but can also help with equality queries with duplicate values (non-key field index).

- **Multi-attribute** search keys should be considered when a WHERE clause contains several conditions.
  - Order of attributes matters for range queries.
  - Such indexes can sometimes enable **index-only** strategies for important queries (e.g., aggregates / grouped aggregates).
    - **Note:** For index-only strategies, clustering isn’t important!

- Choose indexes that benefit **as many queries** as possible.
  - Only one index can be clustered per relation, so choose it based on important queries that can benefit the most from clustering.
**Examples of Clustered Indexes**

- B+ tree index on `E.age` can be used to get qualifying tuples.
  - How selective is the condition?
  - Should the index be clustered?

- Consider the `GROUP BY` query.
  - If most tuples have `E.age > 10`, using `E.age` index and sorting the retrieved tuples may be costly.
  - Clustered `E.dno` index may win!

- Equality queries & duplicates:
  - Clustering on `E.hobby` helps!

---

**Indexes with Composite Search Keys**

- **Composite Search Keys**: Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. E.g., wrt `<sal,age>` index:
    - `(age=20 AND sal=75)`
  - Range query: Some field value is a range, not a constant. E.g., again wrt `<sal,age>` index:
    - `age=20; or (age=20 AND sal > 10)`

- Data entries in index sorted by search key to support such range queries.
  - **Lexicographic order**

---

Database Management Systems 3ed, R. Ramakrishnan and J. Gehrke
Composite Search Keys

- To retrieve Emp records with \( \text{age}=30 \ \text{AND} \ \text{sal}=4000 \), an index on \(<\text{age},\text{sal}>\) would be better than an index only on \(\text{age}\) or an index only on \(\text{sal}\).
  - Note: Choice of index key is orthogonal to clustering.

- If condition is: \(20<\text{age}<30 \ \text{AND} \ 3000<\text{sal}<5000\):
  - Clustered B+ tree index on \(<\text{age},\text{sal}>\) or \(<\text{sal},\text{age}>\) is best.

- If condition is: \(\text{age}=30 \ \text{AND} \ 3000<\text{sal}<5000\):
  - Clustered \(<\text{age},\text{sal}>\) index much better than \(<\text{sal},\text{age}>\) index! (Think about why! Picture the index...)

- Composite indexes are larger; updated more often.

Index-Only Query Plans

- Some queries can be answered without retrieving any tuples from one or more of the relations involved if a suitable index is available.
  - (Sometimes called a “covering index” for the given query.)

- \(\text{SELECT} \ \text{E.dno}, \ \text{COUNT}(*) \ \text{FROM} \ \text{Emp} \ \text{E} \ \text{GROUP} \ \text{BY} \ \text{E.dno}\)
- \(\text{SELECT} \ \text{E.dno}, \ \text{MIN(\text{E.sal})} \ \text{FROM} \ \text{Emp} \ \text{E} \ \text{GROUP} \ \text{BY} \ \text{E.dno}\)
- \(\text{SELECT} \ \text{AVG(\text{E.sal})} \ \text{FROM} \ \text{Emp} \ \text{E} \ \text{WHERE} \ \text{E.age}=25 \ \text{AND} \ \text{E.sal} \ \text{BETWEEN} \ 3000 \ \text{AND} \ 5000\)
Some Illustrated Index-Only Plans

Note: The index files are each much smaller than the main file!