CS122A: Introduction to Data Management

Lecture #15: Physical DB Design

Instructor: Chen Li

Overview

- After ER design, schema refinement, and the definition of views, we have the *conceptual* and *external* schemas for our database.
- The next step is to choose indexes, make clustering decisions, and to refine the conceptual and external schemas (if necessary) to meet performance goals.
- We should 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 key? Should we build several indexes?
- For each index, what kind of an index should it be?
  - B+ tree? Hashed? Clustered or 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*.)

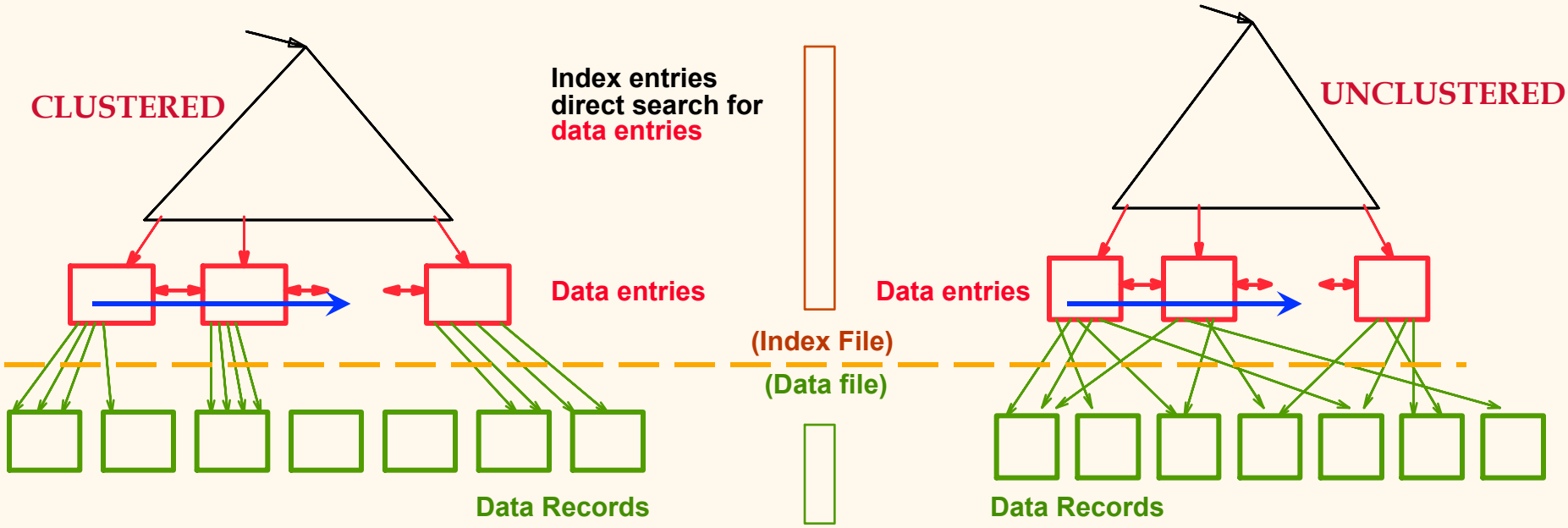
Understanding the Workload

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

*Index Classification (**Review**)*

- *Primary* vs. *secondary*: If search key contains the primary key, then called the primary index.
  - *Unique* index: Search key contains a *candidate* key.
- *Clustered* vs. *unclustered*: If order of data records is the same as, or `close to', the order of stored data records, then called 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 (Review)



(Read each page once.)

(Read more pages – and repeatedly!)

Choice of Indexes (Contd.)

- 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 implies that we must understand and see how a DBMS evaluates its queries (query evaluation 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 → hashed index (B+ tree if not available).
  - Range query → B+ tree index.
    - Clustering especially useful for range queries; also helps equality queries with duplicates (non-key field index).
- Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
  - Order of attributes important for range queries.
  - Such indexes can sometimes enable index-only strategies for important queries (e.g., aggregates / grouped aggregates).
    - 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

```
SELECT  E.dno
FROM  Emp E
WHERE  E.age > 40
```

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

```
SELECT  E.dno,
        COUNT (*)
FROM  Emp E
WHERE  E.age > 10
GROUP BY E.dno
```

- 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!

```
SELECT  E.dno
FROM  Emp E
WHERE E.hobby='Stamps'
```

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

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.

<*E.dno*>

```
SELECT  E.dno, COUNT(*)
FROM  Emp E
GROUP BY  E.dno
```

Tuning the Conceptual Schema

- The choice of conceptual schema should be guided by the workload, in addition to redundancy issues:
  - We may settle for a 3NF schema rather than BCNF.
  - Workload may influence the choice we make in decomposing a relation into 3NF or BCNF.
  - We might *denormalize* (i.e., undo a decomposition step), or we might add fields to a relation.
  - We might consider *vertical decompositions*.
- If such changes come after a database is in use, it's called *schema evolution*; might want to mask some of the changes from applications by defining *views*.

Example Schemas

Contracts (Cid, Sid, Jid, Did, Pid, Qty, Val)

- We will concentrate on Contracts, denoted as CSJDPQV. The following ICs are given to hold:
  JP → C, SD → P, C → CSJDPQV.
  - Find the candidate keys for CSJDPQV.
  - Currently this relation is in 3NF.

Settling for 3NF vs BCNF

- CSJDPQV can be decomposed into SDP and CSJDQV, and both relations are then in BCNF
  - Lossless decomposition, but not dependency-preserving.
  - Adding CJP makes it dependency-preserving as well.
- Suppose that this query is very important:
  - *Find the number of copies Q of part P ordered in contract C.*
  - Requires a join on the decomposed schema, but can be answered by a scan of the original relation CSJDPQV.
  - Could lead us to settle for the 3NF schema CSJDPQV!

More Guidelines for Query Tuning

- Minimize the use of DISTINCT: don't need/say it if duplicates are acceptable or answer contains a key.
- Perhaps minimize the use of GROUP BY and HAVING:

```
SELECT  MIN (E.age)
FROM  Employee E
GROUP BY  E.dno
HAVING  E.dno=102
```

```
SELECT  MIN (E.age)
FROM  Employee E
WHERE  E.dno=102
```

Guidelines for Query Tuning (Contd.)

- Avoid using intermediate relations:

```
SELECT  * INTO  Temp
FROM  Emp E, Dept D
WHERE  E.dno=D.dno
        AND  D.mgrname= 'Joe'
```

+

```
SELECT  T.dno, AVG(T.sal)
FROM  Temp T
GROUP BY  T.dno
```

vs.

```
SELECT  E.dno, AVG(E.sal)
FROM  Emp E, Dept D
WHERE  E.dno=D.dno
        AND  D.mgrname= 'Joe'
GROUP BY  E.dno
```

- Does not materialize the intermediate reln Temp.
- If there is a dense B+ tree index on <*dno, sal*>, an index-only plan can be used to avoid retrieving Emp tuples in the second query!

Summary

- Database design consists of several tasks: *requirements analysis, conceptual design, schema refinement, physical design* and *tuning*.
  - In general, have to go back and forth between these tasks to refine a database design, and decisions in one task can influence the choices in another task.
- Understanding the nature of the *workload* for the application, and the performance goals, is essential to developing a good design.
  - What are the important queries and updates? What attributes/relations are involved?

Summary (Cont'd.)

- The conceptual schema should be refined by considering performance criteria and workload:
  - May choose 3NF or lower normal form over BCNF.
  - May choose among alternative decompositions into BCNF (or 3NF) based upon the workload.
  - May *denormalize*, or undo, some decompositions.

Summary (Cont'd.)

- Over time, indexes may have to be fine-tuned (dropped, created, re-built, ...) for performance.
  - Should determine the query plan(s) used by the system, and adjust the choice of indexes appropriately.
- System may still not find a good plan:
  - Null values, arithmetic conditions, string expressions, the use of ORs, etc., can potentially confuse an optimizer.
- So, may have to rewrite the query/view:
  - Avoid nested queries, temporary relations, complex conditions, and operations like DISTINCT and GROUP BY.