


Physical Database Design and Tuning


1



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 must begin by understanding the *workload*:
 - The most important queries and how often they arise.
 - The most important updates and how often they arise.
 - The desired performance for these queries and updates.


2



Choice of Indexes

- ❖ One approach: consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index. If so, create it.
- ❖ Before creating an index, must also consider the impact on updates in the workload!
 - Trade-off: indexes can make queries go faster, updates slower. Require disk space, too.


3



Issues to Consider in Index Selection

- ❖ Attributes mentioned in a WHERE clause are candidates for index search keys.
 - Exact match condition suggests hash index.
 - Range query suggests tree index.
 - ◆ Clustering is especially useful for range queries, although it can help on equality queries as well in the presence of duplicates.
- ❖ Try to choose indexes that benefit as many queries as possible. Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.


4



Issues in Index Selection (Contd.)

- ❖ Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
 - If range selections are involved, order of attributes should be carefully chosen to match the range ordering.
- ❖ When considering a join condition:
 - Hash index on inner is very good for Index Nested Loops.
 - ◆ Should be clustered if join column is not key for inner, and inner tuples need to be retrieved.
 - *Clustered* B+ tree on join column(s) good for Sort-Merge.

5



Examples of Clustering

- ❖ B+ tree index on E.age can be used to get qualifying tuples.
 - How selective is the condition?
 - Is the index clustered?
- ❖ Consider the GROUP BY query.
 - If many tuples have *E.age* > 10, using *E.age* index and sorting the retrieved tuples may be costly.
 - Clustered *E.dno* index may be better!
- ❖ Equality queries and duplicates
 - Clustering on *E.hobby* helps!

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

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

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

6

Clustering and Joins

```
SELECT E.ename, D.mgr
FROM Emp E, Dept D
WHERE D.dname='Toy' AND E.dno=D.dno
```

- ❖ Clustering is especially important when accessing inner tuples in INL.
 - Should make index on *E.dno* clustered.
- ❖ Suppose that the WHERE clause is instead:
WHERE E.hobby='Stamps' AND E.dno=D.dno
 - If many employees collect stamps, Sort-Merge join may be worth considering. A *clustered* index on D.dno would help.
- ❖ *Bottom line*: Clustering is useful whenever many tuples are to be retrieved.

7

Co-Clustering Relations

- ❖ SELECT E.name, D.dname
FROM Emp E, Dept D
WHERE E.dno = D.dnum
- ❖ Fast if employees are stored "with" their departments (co-clustered).
- ❖ Co-clustering slower for file scans of each file.
- ❖ Difficult implementation.
- ❖ Provided in Oracle, other systems.

8

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 may further decompose a BCNF schema!
 - We might *denormalize* (i.e., undo a decomposition step), or we might add fields to a relation.
 - We might consider *horizontal decompositions*.
- ❖ If such changes are made after a database is in use, called *schema evolution*; might want to mask some of these changes from applications by defining *views*.

9

Horizontal Decompositions

- ❖ Vertical decomposition: Relation is replaced by a collection of relations that are *projections*. Most important case.
- ❖ Sometimes, might want to replace relation by a collection of relations that are *selections*.
 - Each new relation has same schema as the original, but a subset of the rows.
 - Collectively, new relations contain all rows of the original. Typically, the new relations are disjoint.

10

Horizontal Decompositions (Contd.)

- ❖ Suppose that contracts with value > 10000 are subject to different rules. This means that queries on Contracts will often contain the condition *val>10000*.
- ❖ One way to deal with this is to build a clustered B+ tree index on the *val* field of Contracts.
- ❖ A second approach is to replace contracts by two new relations: LargeContracts and SmallContracts, with the same attributes.
 - Performs like index on such queries, but no index overhead.
 - Can build clustered indexes on other attributes, in addition!

11

Masking Conceptual Schema Changes

```
CREATE VIEW Contracts(cid, sid, jid, did, pid, qty, val)
AS SELECT *
FROM LargeContracts
UNION
SELECT *
FROM SmallContracts
```

- ❖ The replacement of Contracts by LargeContracts and SmallContracts can be masked by the view.
- ❖ However, queries with the condition *val>10000* must be asked wrt LargeContracts for efficient execution: so users concerned with performance have to be aware of the change (*if* DBMS can't figure it out!).

12

Tuning Queries and Views

- ❖ If a query runs slower than expected, check if an index needs to be re-built, or if statistics are too old.
- ❖ Sometimes, the DBMS may not be executing the plan you had in mind. Common areas of weakness:
 - Selections involving null values.
 - Selections involving arithmetic or string expressions.
 - Selections involving OR conditions.
 - Lack of evaluation features like index-only strategies or certain join methods or poor size estimation.
- ❖ Check the plan that is being used! Then adjust the choice of indexes or rewrite the query/view.

13

More Guidelines for Query Tuning

- ❖ Unnest queries when possible (single-block opt.)
- ❖ Minimize the use of DISTINCT: don't need it if duplicates are acceptable, or if answer contains a key.
- ❖ 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
```

- ❖ Consider DBMS use of index when writing arithmetic expressions: $E.age=2*D.age$ will benefit from index on $E.age$, but might not benefit from index on $D.age$!

14

Guidelines for Query Tuning (Contd.)

- ❖ Avoid using intermediate relations:

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

vs.

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

and

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

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

15

The Notorious COUNT "Bug"

```
SELECT dname FROM Department D
WHERE D.num_emps >
(SELECT COUNT(*) FROM Employee E
WHERE D.building = E.building)
```

```
CREATE VIEW Temp (empcount, building) AS
SELECT COUNT(*), E.building
FROM Employee E
GROUP BY E.building
```

```
SELECT dname
FROM Department D, Temp T
WHERE D.building = T.building
AND D.num_emps > T.empcount;
```

16

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?

17

Summary (Contd.)

- ❖ Indexes must be chosen to speed up important queries (and perhaps some updates!).
 - Clustering and Co-clustering are important decisions
 - Performance trade-offs
- ❖ The conceptual schema should be refined by considering performance criteria and workload:
 - May *denormalize*, or undo some decompositions to speed up joins
 - May decompose (vertically or horizontally) speed up selections, projections
 - Redundancy vs. join-inducing IC check

18

Summary (Contd.)

- ❖ Statistics have to be periodically updated.
- ❖ Over time, indexes have to be fine-tuned (dropped, created, re-built, ...) for performance.
 - Should determine the plan used by the system, and adjust the choice of indexes appropriately.
- ❖ System may still not find a good plan:
 - Only left-deep plans considered (usually)!
 - Null values, arithmetic conditions, string expressions, the use of ORs, etc. can 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.

19

State of the Art, physical design

- ❖ Clustering (not indexes) for applications
- ❖ Simulations for performance evaluation
- ❖ Benchmarks for performance evaluation
- ❖ Bitmap indexes
- ❖ Compression techniques (stay tuned...)
- ❖ DBMS implementations must take advantage of modern CPUs better (e.g. overlapped execution)
- ❖ Data distribution and replication

20