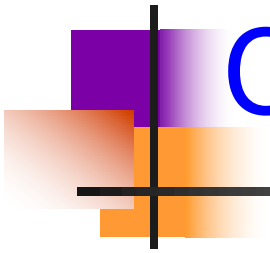
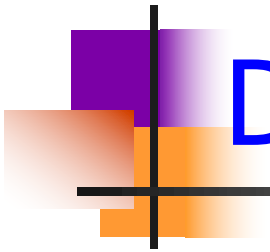
Performance Tuning

Chapter 20



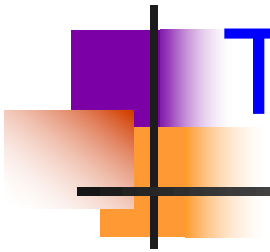
Overview

- 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.



Decisions to Make

- What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- For each index, what kind of an index should it be?
  - Clustered? Hash/tree?
- Should we make changes to the conceptual schema?
  - Consider alternative normalized schemas? (Remember, there are many choices in decomposing into BCNF, etc.)
  - Should we ``undo'' some decomposition steps and settle for a lower normal form? (*Denormalization.*)
  - Horizontal partitioning, replication, views ...



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 *vertical decompositions*.
  - 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*.

Example Schemas

Contracts (Cid, Sid, Jid, Did, Pid, Qty, Val)
Depts (Did, Budget, Report)
Suppliers (Sid, Address)
Parts (Pid, Cost)
Projects (Jid, Mgr)

- We will concentrate on Contracts, denoted as CSJDPQV. The following ICs are given to hold:
  JP → C, SD → P, C is the primary key.
  - What are the candidate keys for CSJDPQV?
  - What normal form is this relation schema in?

Question?

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 is the primary key.

- What are the candidate keys for CSJDPQV?

A. C, JP
B. C, JP, SD
C. C, JP, JSD
D. None of the above

Question?

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 is the primary key.

- What normal form is this relation schema in?

A. BCNF
B. 3NF
C. 1NF
D. None of the above

Settling for 3NF vs BCNF

- CSJDPQV can be decomposed into SDP and CSJDQV, and both relations are in BCNF. (Which FD suggests that we do this?)
  - 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.

QPC should in same table

⇓

don't decompose is better

Denormalization

- Suppose that the following query is important:
  - *Is the value of a contract less than the budget of the department?*
- To speed up this query, we might add a field *budget* B to Contracts.
  - This introduces the FD $D \rightarrow B$ wrt Contracts.
  - Thus, Contracts is no longer in 3NF.
- We might choose to modify Contracts thus if the query is sufficiently important, and we cannot obtain adequate performance otherwise (i.e., by adding indexes or by choosing an alternative 3NF schema.)

Decomposition of a BCNF Relation

- Suppose that we choose { SDP, CSJDQV }. This is in BCNF, and there is no reason to decompose further (assuming that all known ICs are FDs).
- However, suppose that these queries are important:
  - *Find the contracts held by supplier S.*
  - *Find the contracts that department D is involved in.*
- Decomposing CSJDQV further into CS, CD and CJQV could speed up these queries. (Why?)
- On the other hand, the following query is slower:
  - *Find the total value of all contracts held by supplier S.*

Horizontal Decompositions

Scaling reason / performance

- Our definition of 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.

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 (CSJDPQV).
  - Performs like index on such queries, but no index overhead.
  - Can build clustered indexes on other attributes, in addition!

Masking Conceptual Schema Changes

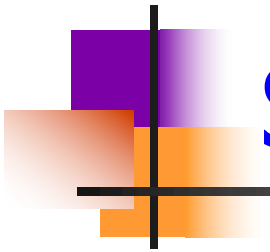
```
CREATE VIEW  Contracts(cid, sid, jid, did, pid, qty, val)
        AS  SELECT  *
        FROM  LargeContracts
        UNION
        SELECT  *
        FROM  SmallContracts
```

- 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.

Tuning Queries and Views

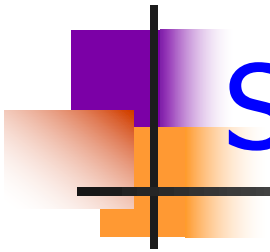
use to estimate cost

- 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:
  1. Selections involving null values.
  2. Selections involving arithmetic or string expressions.
  3. 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.



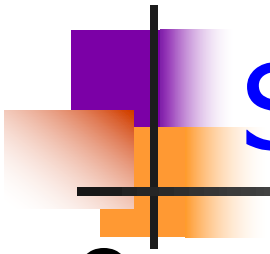
Summary

- 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

- 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.
  - May decompose a BCNF relation further!
  - May choose a *horizontal decomposition* of a relation.
  - Importance of dependency-preservation based upon the dependency to be preserved, and the cost of the IC check. (Needs additional tables, and non-BCNF).



Summary (Contd.)

- 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:
  - 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.

Organizing Data

- Creates value!!
- An address list comprises thousands of addresses.
- A "complete" address list is more valuable than a random list of the same size.
- Joining information also adds value – e.g. address list is immensely more valuable if joined with information on hobbies.

Observation vs …

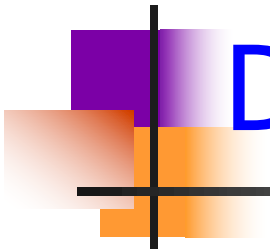
- No expectation of privacy on a public street.
  - I am free to watch you walk down the street.
  - Also, (largely) free to photograph you, and publish the photo.
- Usually not a concern.

… vs Surveillance

- What if automated cameras record you constantly every time you are everywhere on a public street?
- There is a complete record of all your (public) movements.
- Which can determine exactly where you were at any point in time.

Winners and Losers

- Good for law enforcement
- May be good for some big businesses
- Bad for opposition/protest
- Bad for adherents of minority practices that are not widely accepted in society
- Bad if "unauthorized" entities can gain access



Data Matter

- As a database expert, you have a great deal of power in an increasingly data-driven world.
- Wield it wisely.