



Physical Database Design and Database Tuning



Summary of Relational DBMS

- Logical level design
 - Schemas
 - Query languages – SQL
 - Design of schemas
- Physical Level design issues
 - File storage
 - Index structures
 - Query processing methods
 - ✓ How to “tune” the DB application to meet your needs?
 - Recovery
 - Concurrency



Database Tuning: Overview

- After schema design, refinement/normalization, 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.



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?
How selective are these conditions likely to be?
- For each update in the workload:
 - Which attributes are involved in selection/join conditions?
How selective are these conditions likely to be?
 - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

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? Dynamic/static? Dense/sparse?
 - Oracle uses tree index; allows sorting
- 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 ...

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.
 - How do you define “most important” ?
 - ✓ Frequency, Priority (transactions vs lookup)
- 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.

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*.

Example Schemas

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

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

Denormalization: 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.

Horizontal Decompositions

- 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

- 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**.

Rewriting SQL Queries

- Complicated by interaction of:
 - NULLS, duplicates, aggregation, subqueries.
- **Guideline: Use only one "query block", if possible.**

```
SELECT DISTINCT *           SELECT DISTINCT S.*
FROM Sailors S              FROM Sailors S,
WHERE S.sname IN            YoungSailors Y
                           (SELECT Y.sname
                             FROM YoungSailors Y)
                           WHERE S.sname = Y.sname
```

- v **Not always possible ...**

A real application: other factors ?

- How does the system look in a real application ?
- Banner ?
- Multi-tier architecture!

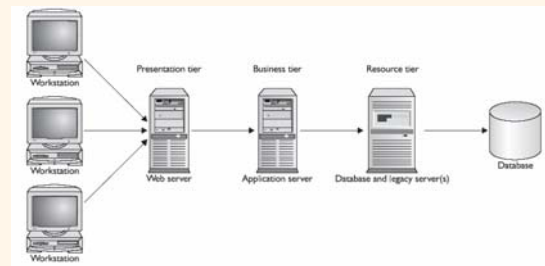
Why multi-tier architecture?

- **Decomposing** (breaking a large object into smaller component parts) software systems address two main concerns
 - Most systems are too complex to comprehend in their entirety.
 - Different audiences require different perspectives of a system.

Components of a multi-tier architecture

- Front-end: Provides portable presentation logic
- Back-end: Provides access to dedicated services, such as a database server
- Middle-tier: Allows users to share and control business logic by isolating it from the actual application

A Typical three tier architecture



Advantages of multi-tier architecture

- Changes to the user interface or to the application logic are largely independent from one another.
- Network bottlenecks are minimized as application layer does not transmit extra data to the client
- When business logic changes are required, only the server has to be updated.
- Database connections can be “pooled,” reducing the cost associated with per-user licensing.

What about performance of multi-tier architecture ?

- Different components have different characteristics and performance!
- Typically defined as Throughput of the system
 - Number of units of work (jobs) completed per unit time
 - ✓ Page downloads/sec
 - ✓ Jobs/sec
 - ✓ Disk IO/sec
 - ✓ Queries/sec
- Availability is also an issue
 - Ex: E-Bay does not want its server going down!
 - Build redundancy and fault tolerance into system

Response Time Breakdown

Browser Time		Network Time			E-commerce Server Time		
Processing	I/O	Browser to ISP Time	Internet Time	ISP to Server Time	Processing	I/O	Networking

..... CONGESTION

- Service time (does not depend on the load)
- Congestion (load-dependent)

Where do you think the bottleneck is?

- Client time is something the overall system designer has little control over
 - Depends on client machine
- Network time depends on network traffic
- DB time is query processing time
 - Send small packets across network or large packets ?

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 (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.

Next....

- Transaction Processing: Recovery
 - Concept of transaction
 - How does system recover from failures
 - ✓ Loss of main memory data
- Transaction processing: Concurrency
 - How do we interleave different queries into same table ?
- Exam 2
- Other topics:
 - Security & Privacy in Databases
 - ✓ Guest lectures
 - Information Analysis
 - ✓ Data Mining- guest lectures?
 - ✓ Search Engines and Information Retrieval
 - Multimedia and Spatial databases



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.

Example 1

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

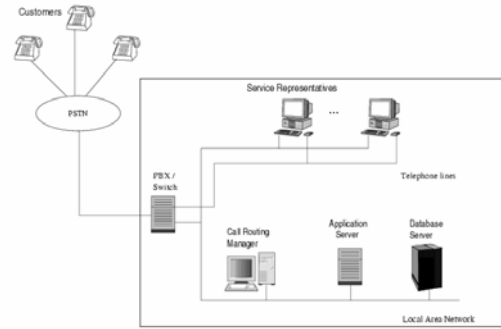
- Hash index on *D.dname* supports 'Toy' selection.
 - Given this, index on *D.dno* is not needed.
- Hash index on *E.dno* allows us to get matching (inner) Emp tuples for each selected (outer) Dept tuple.
- What if WHERE included: `` ... AND E.age=25'' ?
 - Could retrieve Emp tuples using index on *E.age*, then join with Dept tuples satisfying *dname* selection. Comparable to strategy that used *E.dno* index.
 - So, if *E.age* index is already created, this query provides much less motivation for adding an *E.dno* index.

Example 2

```
SELECT E.ename, D.mgr
FROM Emp E, Dept D
WHERE E.sal BETWEEN 10000 AND 20000
AND E.hobby='Stamps' AND E.dno=D.dno
```

- Clearly, Emp should be the outer relation.
 - Suggests that we build a hash index on *D.dno*.
- What index should we build on Emp?
 - B+ tree on *E.sal* could be used, OR an index on *E.hobby* could be used. Only one of these is needed, and which is better depends upon the selectivity of the conditions.
 - ✓ As a rule of thumb, equality selections more selective than range selections.
- As both examples indicate, our choice of indexes is guided by the plan(s) that we expect an optimizer to consider for a query. **Have to understand optimizers!**

Motivating Example: a Call Center



Call Center

- Goals:
 - Foster better relationship with customers, creating customer loyalty and ensuring quality service
 - Improve efficiency and service performance
 - Identify and explore new sales opportunities
- Main functions:
 - Order status inquiry
 - Shipment tracking
 - Problem resolution status inquiry
- Requirements: sub-second response time and 24 X 7 operation

Requirements Analysis Phase

- Workload definition
 - Call center's view: Arrival rate of phone calls
 - IT system's view: functions received from the representatives
 - DB Server view: SQL requests from the application server
 - LAN view: packet size distribution and interpacket arrival time

System Design Phase

- What should the system throughput be to meet sub-second response times ?
 - 200 customer service representatives and 80% are working during peak hour
 - Average think time of 30 seconds
- What is the capacity of the DB server so that the performance goals are met

At the operations phase

- Assume DB server is a problem: response times exceed sub-second goal!
 - Measurements during peak hour:
 - ✓ Queries per hour
 - ✓ Average query response time
- How to solve this ?
 - Database Tuning!