Overview

• Introduction
• Index selection and clustering
• Database tuning (de-normalization etc)
• Impact of concurrency

Introduction

• After ER design, schema refinement, and the definition of views, we have the conceptual and external schemas for our database.
• Next step?
Introduction

• After ER design, schema refinement, and the definition of views, we have the conceptual and external schemas for our database.
• Next step?
  • choose indexes, make clustering decisions, and to refine the conceptual and external schemas (if necessary) to meet performance goals.
• How to decide the above?

Paraphrasing [Sun Tzu / Sun Wu / Sunzi]
Know [the] other,
know [the] self,
hundred battles without danger

Paraphrasing [Sun Tzu / Sun Wu / Sunzi]
Know [the] workload
know [the] Q-opt internals
Introduction

• 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?

• For each index, what kind of an index should it be?

• Should we make changes to the conceptual schema?
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 ...

Overview

- Introduction
- Index selection and clustering
- Database tuning (de-normalization etc)
- Impact of concurrency

Example 1

- which index, if any, would you build?

```
SELECT E.ename, D.mgr
FROM Emp E, Dept D
WHERE D.dname='Toy' AND E.dno=D.dno
```
Example 1

- Hash index on \textit{D.dname} supports ‘Toy’ selection.
  - Given this, index on D.dno is not needed.
- Hash index on \textit{E.dno} allows us to get matching (inner) Emp tuples for each selected (outer) Dept tuple.

\begin{align*}
\text{SELECT} & \quad \text{E.ename}, \quad \text{D.mgr} \\
\text{FROM} & \quad \text{Emp} \ E, \ \text{Dept} \ D \\
\text{WHERE} & \quad \text{D.dname}='\text{Toy}' \ \text{AND} \ \text{E.dno}=\text{D.dno}
\end{align*}

Example 1

- What if WHERE included: `... AND \textit{E.age}=25` ?
  - Could retrieve Emp tuples using index on \textit{E.age}, then join with Dept tuples satisfying \textit{dname} selection.
  - Comparable to strategy that used \textit{E.dno} index.
  - So, if \textit{E.age} index is already created, this query provides much less motivation for adding an \textit{E.dno} index.

\begin{align*}
\text{SELECT} & \quad \text{E.ename}, \quad \text{D.mgr} \\
\text{FROM} & \quad \text{Emp} \ E, \ \text{Dept} \ D \\
\text{WHERE} & \quad \text{D.dname}='\text{Toy}' \ \text{AND} \ \text{E.dno}=\text{D.dno}
\end{align*}
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!

Clustering and Joins

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

• What plan? what clustering?
Clustering and Joins

- Clustering is especially important when accessing inner tuples in INL.
  - Should make index on $E.dno$ clustered.

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

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.

**Summary:** Clustering is useful whenever many tuples are to be retrieved.
Overview

- Introduction
- Index selection and clustering
- Database tuning (de-normalization etc)
- Impact of concurrency

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.

Tuning the Conceptual Schema

- If such changes are made after a database is in use: called schema evolution
- Q: How to mask these changes from applications?
Tuning the Conceptual Schema

• If such changes are made after a database is in use: called **schema evolution**

• Q: How to mask these changes from applications?

• A: Views!

```
create view student as
select ssn, name
from new_student
```

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*. 
Example?

• Q: When would we choose 3NF instead of BCNF?

• A: Student-Teacher-subJect (STJ)
  \[ S \rightarrow T \]
  \[ T \rightarrow J \]
  and queries ask for all three attributes (select *)

Example?

• Q: When would we choose 3NF instead of BCNF?

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.
Decomposition of a BCNF Relation

• Q: Scenario?

A: eg., STUDENT(ssn, name, address, ph#, ...)

• with many queries like
  select ssn, name
  from student

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.
De-normalization

• Q: Scenario?

• A: E.g.,
  
  STUDENT (ssn, name)
  TAKES (ssn, cid, grade)
  COURSE (cid, cname)

  – and many queries like: ‘class roster for db-apps’

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.*
Horizontal Decompositions

Sometimes, might want to replace relation by a collection of relations that are selections. Eg.,

STUDENT (ssn, name, status)
decomposed to
CurrentStudent (ssn, name, status)
Alumni (ssn, name, status)

Q: under what scenario would this help performance?

------------------

Masking Conceptual Schema Changes

CREATE VIEW STUDENT(ssn, name, status)
AS SELECT *
FROM CurrentStudent
UNION
SELECT *
FROM Alumni

• Masks change
• But performance-minded users should query the right table

------------------

Tuning Queries and Views

• If a query runs slower than expected, what to check?
Tuning Queries and Views

• If a query runs slower than expected, check
  – whether an index needs to be re-built, or
  – whether statistics are too old or
  – the plan that is used! (and adjust indices/query/views)

Tuning Queries and Views

• 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.
Tuning Queries and Views

- 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.

Rewriting SQL Queries

- Complicated by interaction of:
  - NULLs, duplicates, aggregation, subqueries
- Guideline: Use only one “query block”, if possible.

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

== ??
```
Rewriting SQL Queries

• Complicated by interaction of:
  – NULLs, duplicates, aggregation, subqueries
• Guideline: Use only one “query block”, if possible.

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

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

More Guidelines for Query Tuning

• Minimize the use of DISTINCT: don’t need it if duplicates are acceptable, or if answer contains a key.

```
E.age = 2*D.age
```

• 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 $D.age$!
More Guidelines for Query Tuning

• Minimize the use of GROUP BY and HAVING:

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

More Guidelines for Query Tuning

• 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'

and

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

```
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
```

- 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!
Concurrency

• Reduce lock durations
• Reduce hot spots

• Reduce lock durations
  – make transactions faster
  – break long transactions in shorter ones (but...)
  – build a warehouse
  – consider lower isolation level
Concurrency

- Reduce hot spots
  - delay operations on hot spots
  - optimize access patterns
  - partition (batch) operations on hot spots
  - choice of index (root of B-tree -> hot spot)

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.

Also see the paper by Roussopoulos + Yeh
(on the course web site)
Summary (cont’d)

• Understanding the nature of the workload is vital:
  – What are the important queries and updates? What attributes/relations are involved?
• then:
  – refine conceptual schema and views
  – tune queries (indices, clustering, re-writing)

Summary - schema refinement

• May choose 3NF or lower normal form over BCNF.
• 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 (see text)

Summary - Tuning
Tuning: on slow queries, check the chosen plan!
Q: what are possible culprits?
Summary - Tuning

Tuning: on slow queries, check the chosen plan!

- Over time, indexes have to be fine-tuned (dropped, created, re-built, ...) for performance.
- System may still not find a good plan:
  - Only left-deep plans considered!
  - 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.