Carnegie Mellon Univ.  
Dept. of Computer Science  
15-415/615 - DB Applications  

Lecture #18: Physical Database Design (R&G ch. 20)  

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

FD - summary:  
• ‘Good’ schemas: BCNF (or 3NF)  
  – Everything should depend on the key, the full key, and nothing but the key  

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

Introduction

• How to decide the above?

We must begin by understanding the *workload*:
  – The most important queries and how often they arise.
  – ………………………. updates ………………………
  – 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 ...

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
```

EMP | name | dno
---|------|-----
| toy |

DEPT | dno | dname | mgr
---|-----|-------|------
| toy |

- 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.

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

EMP | dno | DEPT | dno | dname | mgr | INL | outer
---|-----|------|-----|-------|-----|-----|-----
| toy |

- 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

- Emp should be the outer relation – indices for Dept?
  - hash index on D.dno.
- What index for Emp?

- B+ tree on E.sal, OR an index on E.hobby. Only one is probably enough – whichever has better selectivity.
- As a rule of thumb, equality selections more selective than range selections.

Notice: in both examples, the choice of indexes depends on the plan(s) we expect from the optimizer. *Have to understand optimizers!*
Clustering and Joins

- What plan? what clustering?

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

```
EMP | ename  | dno |
----|-------|-----|
dept |       |     |

DEPT | dno  | dname | mgr |
-----|------|-------|-----|
toy  |      |       |     |
```

- Index on Emp.dno – clustered or not?

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

```
EMP | dno |
----|-----|
dep |     |

DEPT | dno |
-----|-----|
toy  |    |
```

Clustering and Joins

- Index on Emp.dno – clustered or not?
- A: clustered will be much faster

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

```
EMP | dno |
----|-----|
dep |     |

DEPT | dno |
-----|-----|
toy  |    |
```

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*.

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

Q: How to mask these changes from applications?

A: *Views*!

\[
\text{create view student as select ssn, name from new_student}
\]
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*.

Example?

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

  • A: Student-Teacher-subj ect (STJ)
    S J -> T
    T -> J
    and queries ask for all three attributes (select * )
Decomposition of a BCNF Relation

• Q: Scenario?
  – e.g., STUDENT(ssn, name, address, ph#, ...)

• A: 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?
  – E.g.,
    STUDENT (ssn, name)
    TAKES (ssn, cid, grade)
    COURSE (cid, cname)
De-normalization

• Q: Scenario?
  – E.g.,
    STUDENT (ssn, name)
    TAKES (ssn, cid, grade)
    COURSE (cid, cname)
• A: and many queries like: ‘class roster for db-apps’
• Q: resulting table(s) and 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 horizontal decompositions.

Horizontal Decompositions

Sometimes, might want to replace relation by a collection of relations that are selections. E.g., STUDENT (ssn, name, status)
decomposed to
    CurrentStudent (ssn, name, status)
    Alumni (ssn, name, status)
Q: under what scenario would this help performance?

A: most queries on ‘current’, & too large ‘alumni’ table
Horizontal Decompositions

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

\[ \text{STUDENT} (\text{ssn, name, status}) \]

decomposed to

\[ \text{CurrentStudent} (\text{ssn, name, status}) \]
\[ \text{Alumni} (\text{ssn, name, status}) \]

Q’: How to mask the change?

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

---

Masking Conceptual Schema Changes

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

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

---

Tuning Queries and Views

• If a query runs slower than expected, what to check?

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

---

Tuning Queries and Views

• If a query runs slower than expected, check – the plan that is used! (and adjust indices/query/views)
Tuning Queries and Views

• If a query runs slower than expected, check
  – the plan that is used! (and adjust indices/query/views)
  – whether statistics are too old

Tuning Queries and Views

• Sometimes, the DBMS may not be executing the plan you had in mind. Common areas of weakness:
  
  ```sql
  select *
  from employee
  where name like 'smith%'
  or salary > 10
  ```

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.

```
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 - when?
More Guidelines for Query Tuning

- Minimize the use of DISTINCT - when?
- A1: when duplicates are acceptable, or
- A2: if answer contains a key.

```sql
SELECT DISTINCT ssn
FROM Student;
```

More Guidelines for Query Tuning

- 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:

```sql
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:

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

```sql
SELECT MIN (E.age)
FROM Employee E
WHERE E.dno=102
```
Guidelines for Query Tuning (Contd.)

• Avoid using intermediate relations:

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

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

vs.

```sql
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!
Overview

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

Concurrency

• Reduce lock durations
• Reduce hot spots

• Reduce lock durations
  – make transactions faster
  – break long transactions in shorter ones (but...)
  – build a data 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)

• 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)

Know the workload
Know the Q-opt internals
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

So, may have to rewrite the query/view: Avoid
• nested queries,
• temporary relations,
• complex conditions, and
• operations like DISTINCT and GROUP BY.

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. > 3* salary
  – Selections involving arithmetic or string expressions.
  – Selections involving OR conditions. like “%main%”
  – Lack of evaluation features like index-only strategies or certain join methods or poor size estimation.