CSE 444: Database Internals

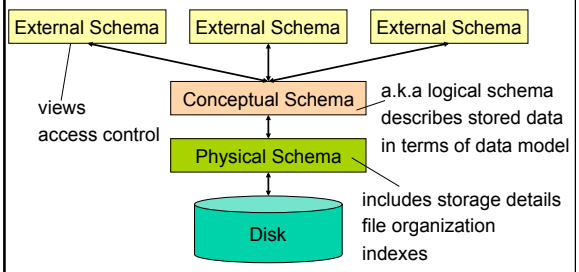
Lectures 13
Database Tuning

Database Tuning Overview

- The database tuning problem
- Index selection (discuss in detail)
- Horizontal/vertical partitioning (discuss in detail)
- Denormalization (discuss briefly)

This material is partially based on the book: "Database Management Systems" by *Ramakrishnan and Gehrke*, **Ch. 20**

Levels of Abstraction in a DBMS



The Database Tuning Problem

- We are given a workload description
  - List of queries and their frequencies
  - List of updates and their frequencies
  - Performance goals for each type of query
- Perform *physical database design*
  - Choice of indexes and materialized views
  - Tuning the conceptual schema
    - Denormalization, vertical and horizontal partition
  - Query and transaction tuning

The Index Selection Problem

- Given a database schema (tables, attributes)
- Given a "query workload":
  - Workload = a set of (query, frequency) pairs
  - The queries may be both SELECT and updates
  - Frequency = either a count, or a percentage
- Select a set of indexes that optimizes the workload

In general this is a very hard problem

Index Selection: Which Search Key

- Make some attribute K a search key if the WHERE clause contains:
  - An exact match on K
  - A range predicate on K
  - A join on K

The Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

What indexes ?

The Index Selection Problem 1

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

A: V(N) and V(P) (hash tables or B-trees)

The Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

What indexes ?

The Index Selection Problem 2

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100 queries:

```
SELECT *
FROM V
WHERE P=?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

A: definitely V(N) (must B-tree); unsure about V(P)

The Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1000000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

What indexes ?

The Index Selection Problem 3

V(M, N, P);

Your workload is this

100000 queries:

```
SELECT *
FROM V
WHERE N=?
```

1000000 queries:

```
SELECT *
FROM V
WHERE N=? and P>?
```

100000 queries:

```
INSERT INTO V
VALUES (?, ?, ?)
```

A: V(N, P)

The Index Selection Problem 4

V(M, N, P);

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

What indexes ?

The Index Selection Problem 4

V(M, N, P);

Your workload is this

1000 queries:

```
SELECT *
FROM V
WHERE N>? and N<?
```

100000 queries:

```
SELECT *
FROM V
WHERE P>? and P<?
```

A: V(N) secondary, V(P) primary index

The Index Selection Problem

- SQL Server 2000 Index Tuning Wizard
  - Based on *AutoAdmin* research project
  - Much acclaimed research project from mid 90's
- Similar ideas adopted by other major vendors
  - E.g., DB2 Index Advisor
- How they work:
  - They walk through a large number of configurations
  - Compute their costs
  - And choose the configuration with minimum cost

Basic Index Selection Guidelines

- Consider queries in workload in order of importance
- Consider relations accessed by query
  - No point indexing other relations
- Look at WHERE clause for possible search key
- Try to choose indexes that speed-up multiple queries
- And then consider the following…

Index Selection: Multi-attribute Keys

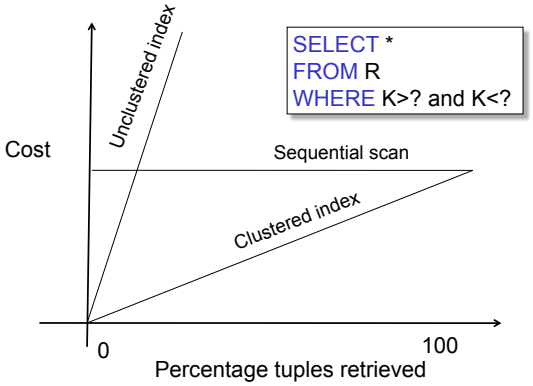
Consider creating a multi-attribute key on K1, K2, … if

- WHERE clause has matches on K1, K2, …
  - But also consider separate indexes
- SELECT clause contains only K1, K2, ..
  - A *covering index* is one that can be used exclusively to answer a query, e.g. index R(K1,K2) covers the query:

```
SELECT K2 FROM R WHERE K1=55
```


To Cluster or Not

- Range queries benefit mostly from clustering
- Covering indexes do *not* need to be clustered: they work equally well unclustered



Hash Table v.s. B+ tree

- Rule 1: always use a B+ tree ☺
- Rule 2: use a Hash table on K when:
  - There is a very important selection query on equality (WHERE K=?), and no range queries
  - You know that the optimizer uses a nested loop join where K is the join attribute of the inner relation

Balance Queries v.s. Updates

- Indexes speed up queries
  - SELECT FROM WHERE
- But they usually slow down updates:
  - INSERT, DELETE, UPDATE
  - However some updates benefit from indexes

```
UPDATE R
  SET A = 7
  WHERE K=55
```


Tuning a Single Query

- Often, a problem arises with a single query
- Step 1: Take a look at the optimizer query plan
  - EXPLAIN
- Step 2: Study the plan
  - Is optimizer using indexes? What join algorithm? Etc.
- Step 3: Add indexes as needed
- Step 4: Inspect query plan again

Tuning the Conceptual Schema

- Index selection
- Horizontal/vertical partitioning
- Denormalization

Vertical Partitioning

Resumes

SSN	Name	Address	Resume	Picture
234234	Mary	Huston	Clob1…	Blob1…
345345	Sue	Seattle	Clob2…	Blob2…
345343	Joan	Seattle	Clob3…	Blob3…
234234	Ann	Portland	Clob4…	Blob4…

T1

SSN	Name	Address
234234	Mary	Huston
345345	Sue	Seattle
. . .		

T2

SSN	Resume
234234	Clob1…
345345	Clob2…

T3

SSN	Picture
234234	Blob1…
345345	Blob2…

Vertical Partitioning

```
CREATE VIEW Resumes AS
    SELECT T1.ssn, T1.name, T1.address,
           T2.resume, T3.picture
    FROM   T1,T2,T3
    WHERE  T1.ssn=T2.ssn and T2.ssn=T3.ssn
```


Vertical Partitioning

```
SELECT address
FROM   Resumes
WHERE  name = 'Sue'
```

Which of the tables T1, T2, T3 will be queried by the system ?

When do we use vertical partitioning ?

Vertical Partitioning Applications

1. Can improve performance of some queries
   - When queries touch small fraction of columns
   - Only need to read desired columns from disk
   - Can produce big I/O savings for wide tables
   - Potential benefit in data warehousing applications

- But
  - Repeated key columns add a lot of overhead
  - Need expensive joins to reconstruct tuples

Vertical Partitioning Applications

2. When some fields are large + rarely accessed
   - E.g. Picture
3. In distributed databases
   - Customer personal info at one site, profile at another
4. In data integration
   - T1 comes from one source
   - T2 comes from a different source

Horizontal Partitioning

Customers

SSN	Name	City	Country
234234	Mary	Houston	USA
345345	Sue	Seattle	USA
345343	Joan	Seattle	USA
234234	Ann	Portland	USA
--	Frank	Calgary	Canada
--	Jean	Montreal	Canada

CustomersInHouston

SSN	Name	City	Country
234234	Mary	Houston	USA

CustomersInSeattle

SSN	Name	City	Country
345345	Sue	Seattle	USA
345343	Joan	Seattle	USA

CustomersInCanada

SSN	Name	City	Country
--	Frank	Calgary	Canada
--	Jean	Montreal	Canada

Horizontal Partitioning

```
CREATE VIEW Customers AS
    CustomersInHouston
      UNION ALL
    CustomersInSeattle
      UNION ALL
    . . .
```

Horizontal Partitioning

```
SELECT name
FROM   Customers
WHERE  city = 'Seattle'
```

Which tables are inspected by the system ?

WHY ???

Horizontal Partitioning

Better:

```
CREATE VIEW Customers AS
   (SELECT * FROM CustomersInHouston
    WHERE city = 'Houston')
      UNION ALL
   (SELECT * FROM CustomersInSeattle
    WHERE city = 'Seattle')
      UNION ALL
   . . .
```

Other techniques exist: read DBMS documentation

Horizontal Partitioning

```
SELECT name
FROM   Customers
WHERE  city = 'Seattle'
```

⇩

```
SELECT name
FROM   CustomersInSeattle
```


Horizontal Partitioning Applications

- Performance optimization
  - Especially for data warehousing
  - E.g. one partition per month
  - E.g. archived applications and active applications
- Distributed and parallel databases
- Data integration

Tuning the Conceptual Schema

- Index selection
- Horizontal/vertical partitioning (see lecture 4)
- Denormalization

Denormalization

Product(**pid**, pname, price, cid)
Company(**cid**, cname, city)

A very frequent query:

```
SELECT x.pid, x.pname
FROM Product x, Company y
WHERE x.cid = y.cid and x.price < ? and y.city = ?
```

How can we speed up this query workload ?

Denormalization

Product(**pid**, pname, price, cid)
Company(**cid**, cname, city)

Denormalize:

ProductCompany(**pid**, pname, price, cname, city)

```
INSERT INTO ProductCompany
  SELECT x.pid, x.pname, x.price, y.cname, y.city
  FROM Product x, Company y
  WHERE x.cid = y.cid
```


Denormalization

Next, replace the query

```
SELECT x.pid, x.pname
FROM Product x, Company y
WHERE x.cid = y.cid and x.price < ? and y.city = ?
```

```
SELECT pid, pname
FROM ProductCompany
WHERE price < ? and city = ?
```


Issues with Denormalization

- It is no longer in BCNF
  - We have the hidden FD: cid → cname, city
- When Product or Company are updated, we need to propagate updates to ProductCompany
  - Use RULE in PostgreSQL (see PostgreSQL doc.)
  - Or use a trigger on a different RDBMS
- Sometimes cannot modify the query
  - What do we do then ?

Denormalization Using Views

```
INSERT INTO ProductCompany
  SELECT x.pid, x.pname,.price, y.cid, y.cname, y.city
  FROM Product x, Company y
  WHERE x.cid = y.cid;

DROP Product; DROP Company;

CREATE VIEW Product AS
  SELECT pid, pname, price, cid FROM ProductCompany

CREATE VIEW Company AS
  SELECT DISTINCT cid, cname, city FROM ProductCompany
```