Lecture 17: Database Tuning
Database Tuning Overview

- The database tuning problem
- Index selection (discuss in detail)
- Horizontal/vertical partitioning (see lecture 4)
- 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

- **External Schema**: a.k.a logical schema. Describes stored data in terms of data model.
- **Conceptual Schema**
- **Physical Schema**: Includes storage details, file organization, indexes.
- **Disk**: Views, access control.
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
  - 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 decisions

- To index or not to index?
- Which key?
- Multiple keys?
- Clustered or unclustered?
- Hash or trees?
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

100,000 queries:

\[
\text{SELECT * FROM } V \text{ WHERE } N=? 
\]

100 queries:

\[
\text{SELECT * FROM } V \text{ WHERE } P=? 
\]

What indexes?
The Index Selection Problem 1

V(M, N, P)

Your workload is this

100,000 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

100,000 queries:
SELECT *
FROM V
WHERE N>? and N<?

100 queries:
SELECT *
FROM V
WHERE P=?

100,000 queries:
INSERT INTO V
VALUES (?, ?, ?)

What indexes?
The Index Selection Problem 2

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

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

**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

100,000 queries:
SELECT *
FROM V
WHERE N=?

1,000,000 queries:
SELECT *
FROM V
WHERE N=? and P>?

100,000 queries:
INSERT INTO V
VALUES (?, ?, ?)

What indexes?
The Index Selection Problem 3

V(M, N, P)

Your workload is this

100,000 queries:
SELECT * FROM V WHERE N=?

1,000,000 queries:
SELECT * FROM V WHERE N=? and P>?

100,000 queries:
INSERT INTO V VALUES (?, ?, ?)

A: V(N, P)
The Index Selection Problem 2

V(M, N, P)

Your workload is this

1,000 queries:
SELECT *
FROM V
WHERE N>? and N<?

100,000 queries:
SELECT *
FROM V
WHERE P>? And P<?

What indexes?
The Index Selection Problem 2

$$V(M, N, P)$$

Your workload is this

1,000 queries:

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

100,000 queries:

```sql
SELECT *
FROM V
WHERE P>? And P<?
```

A: V(N) secondary (unclustered); V(P) primary (clustered)
The Index Selection Problem

- **SQL Server**
  - Automatically, thanks to *AutoAdmin* project
  - Much acclaimed successful research project from mid 90’s, similar ideas adopted by the other major vendors

- **PostgreSQL**
  - You will do it manually, part of project 3
  - But tuning wizards also exist
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:

```sql
SELECT K2
FROM R
WHERE K1=55
```
Can be answered with an index-only plan
To Cluster or Not to Cluster?

- Range queries benefit mostly from clustering
- Covering indexes do not need to be clustered  Why?
Percentage of tuples retrieved

Cost

SELECT *
FROM R
WHERE K>? and K<?

Unclustered index

Sequential scan

Clustered index

Percentage of tuples retrieved

0

100
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 (you will understand that in a few lectures)
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
```
Tools for Index Selection

- SQL Server 2000 Index Tuning Wizard
- DB2 Index Advisor

How they work:
- They walk through a large number of configurations, compute their costs, and choose the configuration with minimum cost.
Horizontal/Vertical Partitioning

- When would we want to do this?

Contracts(cid, supplierID, projectId, deptID, partID, qty, value)

(in BCNF)

Q1: Find the contracts held by supplier S
Q2: Find the contracts held by department D
Tuning the Conceptual Schema

- Index selection
- Horizontal/vertical partitioning
- Denormalization
Denormalization

Product(\textbf{pid}, \textit{pname}, \textit{price}, \textit{cid})
Company(\textbf{cid}, \textit{cname}, \textit{city})

A very frequent query:

\begin{verbatim}
SELECT x.pid, x.pname
FROM Product x, Company y
WHERE x.cid = y.cid \textbf{and} x.price < \textit{?} \textbf{and} y.city = \textit{?}
\end{verbatim}

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

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

```sql
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, x.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
```