

Introduction to Database Systems

CSE 444

Lecture 16: Database Tuning

About the Midterm

- Open book and open notes
 - But you won't have time to read during midterm!
 - No laptops, no mobile devices
- Three topics:
 1. SQL
 2. ER Diagrams
 3. Transactions

More About the Midterm

- Review Lectures 1 through 14
 - Read the lecture notes carefully
 - Read the book for extra details, extra explanations
 - Look at the Franklin paper on transactions, ARIES
- Review Project 1 (Project 2 not on any exam)
- Review HW1 and HW2
- Take a look at sample midterms

Where We Are?

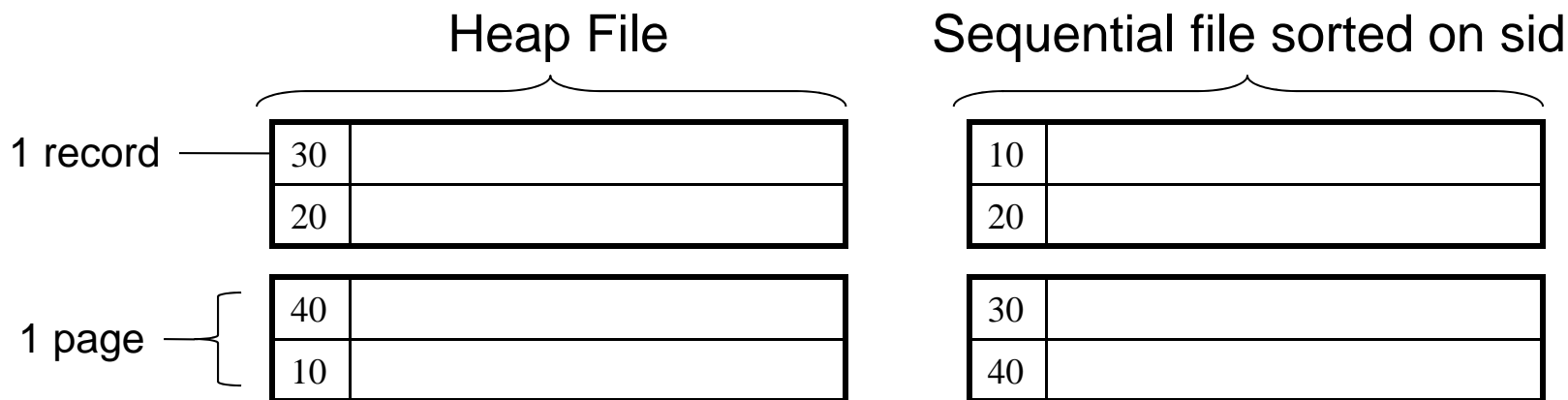
- We just started to learn how a DBMS executes a query...
- ... we started with data storage and indexing

Data Storage & Indexing: Review

How does a DBMS store data?

- Typically one relation = one file
- **Heap file**: tuples inside file are not sorted
- **Sequential file**: tuples sorted on a key

`Student(sid: int, age: int, ...)`



Indexes: Motivation

- **Index**: data structure to speed-up selections on **search key fields** for the index
- An index contains a collection of **data entries**, and supports **efficient retrieval** of all data entries with a given search key value **k**

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**

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

Indexes in PostgreSQL


```
CREATE TABLE V(M int, N varchar(20), P int);
```

```
CREATE INDEX V1_N ON V(N)
```

```
CREATE INDEX V2 ON V(P, M)
```

```
CREATE INDEX VVV ON V(M, N)
```

```
CLUSTER V USING V2
```



Makes V2 clustered

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

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=?
```

What indexes ?

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 3

V(M, N, P);

Your workload is this

100,000 queries: 1,000,000 queries: 100,000 queries:

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

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

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

What indexes ?

The Index Selection Problem 4

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

- **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 (you won't use these on project 3!)

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
- To cluster or not?
 - Range queries benefit mostly from clustering

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)

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 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:

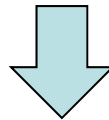
ProductCompany(pid, pname, price, cname, city)

```
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: $cid \rightarrow cname, city$
- 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
```