Database Systems
CSE 414

Lecture 15-16:
Basics of Data Storage and Indexes
(Ch. 8.3-4, 14.1-1.7, & skim 14.2-3)
Announcements

• Midterm on Monday, November 6th, in class
  – Allow 1 page of notes (both sides, 8+pt font)

• WQ4 is due Friday 11pm

• Prof. Gang Luo will be out of town Nov. 3-8
  – No office hour on Nov. 8
  – TA will handle the midterm in class
  – Prof. Magdalena Balazinska will teach the lecture this Friday (Nov. 3)
  – Prof. Dan Suciu will teach the lecture next Wednesday (Nov. 8)
Midterm

• Content
  – Lectures 1 through 17 (today - Friday)
  – HW 1–5, WQ 1–4

• Closed book. No computers, phones, watches, etc.!

• Can bring one letter-sized piece of paper with notes, but…
  – test will not be about memorization
  – formulas provided for join algorithms & selectivity

• Similar in format & content to CSE 414 17sp midterm
  – CSE 344 tests include some things we did not cover
Motivation

- To understand performance, need to understand a bit about how a DBMS works
  - my database application is too slow… why?
  - one of the queries is very slow… why?

- Understanding query optimization
  - we have seen SQL query $\Rightarrow$ logical plan (RA), but not much about RA $\Rightarrow$ physical plan

- Choice of indexes is often up to you
Data Storage

- DBMSs store data in **files**
- Most common organization is row-wise storage:
  - File is split into **blocks**
  - Each block contains a set of tuples
- DBMS reads entire block

In the example, we have 4 blocks with 2 tuples each
Data File Types

The data file can be one of:

- **Heap file**
  - Unsorted

- **Sequential file**
  - Sorted according to some attribute(s) called *key*

Note: *key* here means something different from primary key: it just means that we order the file according to that attribute. In our example, we ordered by **ID**. Might as well order by **fName**, if that seems a better idea for the applications using our DB.

<table>
<thead>
<tr>
<th>ID</th>
<th>fName</th>
<th>lName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Index

• An additional file, that allows fast access to records in the data file given a search key

• The index contains (key, value) pairs:
  – The key = an attribute value (e.g., student ID or name)
  – The value = a pointer to the record

• Could have many indexes for one table

Key = means here search key
This Is Not A Key

Different keys:

• **Primary key** – uniquely identifies a tuple
• **Key of the sequential file** – how the data file is sorted, if at all
• **Index key** – how the index is organized

CSE 414 - Fall 2017
Example 1: Index on ID

Index on **Student.ID**

Data File **Student**

<table>
<thead>
<tr>
<th>ID</th>
<th>fName</th>
<th>lName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
</tbody>
</table>

...
Example 2: Index on fName

Index on Student.fName

Data File Student

<table>
<thead>
<tr>
<th>ID</th>
<th>fName</th>
<th>lName</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Tom</td>
<td>Hanks</td>
</tr>
<tr>
<td>20</td>
<td>Amy</td>
<td>Hanks</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```
Student ID fName lName
10  Tom  Hanks
20  Amy  Hanks
...
```
Index Organization

Several index organizations:

• B+ trees – most popular
  – They are search trees, but they are not binary instead have higher fan-out

• Hash table

• Specialized indexes: bit maps, R-trees, inverted index
Recap: B+ Tree

Level 1 How to find IDs in data file
Level 2 How to find IDs in Level 1
Level 3 How to find IDs in Level 2

(Each level is a fraction of the size of the one below)
A (naïve) hash function:

\[ h(x) = x \mod 10 \]

= disk block

Cost per lookup:
- one access in array
- one access in list

No range queries!
Clustered vs. Unclustered

Every table can have **only one** clustered and **many** unclustered indexes.

SQL Server defaults to cluster by **primary key**.
Index Classification

• **Clustered/unclustered**
  – Clustered = records close in index are close in data
    • Option 1: Data inside data file is sorted on disk
    • Option 2: Store data directly inside the index (no separate files)
  – Unclustered = records close in index may be far in data

• **Primary/secondary**
  – Meaning 1:
    • Primary = is over attributes that include the primary key
    • Secondary = otherwise
  – Meaning 2: means the same as clustered/unclustered

• **Organization**: B+ tree or Hash table
Scanning a Data File

- Hard disks are mechanical devices!
  - Technology from the 60s; density much higher now
- We read only at the rotation speed!
- Consequence: sequential scan is MUCH FASTER than random reads
  - **Good**: read blocks 1, 2, 3, 4, 5,…
  - **Bad**: read blocks 2342, 11, 321, 9, …
- **Rule of thumb**:
  - Random reading 1-2% of the file ≈ sequential scanning the entire file
  - this is decreasing over time (because of increased density of disks)
HDD ~> SSD

- Solid state (SSD): used to be too expensive... not any more
  - entirely different performance characteristics!
Example

```sql
for y in Takes
  if courseID = 300 then
    for x in Student
      if x.ID = y.studentID
    output *

SELECT name
FROM Student x, Takes y
WHERE x.ID = y.studentID AND y.courseID = 300
```

Assume the database has indexes on these attributes:
- `index_takes_course` = index on Takes.courseID
- `index_studentID` = index on Student.ID

```sql
for y1 in index_takes_course where y1.courseID = 300
  for y in y1.Takes
    for x1 in index_studentID where x1.ID = y.studentID
      for x in x1.Student
        output x.*, y.*
```

Takes(studentID, courseID)
Student(studentID, name, ...)

Index selection
Index join
Getting Practical: Creating Indexes in SQL

CREATE TABLE V(M int, N varchar(20), P int);
CREATE INDEX V1 ON V(N);
CREATE INDEX V2 ON V(P, M);
CREATE INDEX V3 ON V(M, N);
CREATE UNIQUE INDEX V4 ON V(N);
CREATE CLUSTERED INDEX V5 ON V(N);

What does this mean?

Not supported in SQLite
Which Indexes?

- How many indexes **could** we create?
  
  15, namely: (ID), (fName), (lName), (ID,fName), (fName,ID), …

- Which indexes **should** we create?
  
  Few! Each new index slows down updates to Student

Index selection is a hard problem
Which Indexes?

• The *index selection problem*
  – given a table, and a “workload” (big Java application with lots of SQL queries), decide which indexes to create (and which ones NOT to create!)

• Who does index selection:
  – database administrator DBA
  – semi-automatically, using a database administration tool
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
Index Selection Problem

Suppose the database has the index I1 below. Discuss physical query plans for these queries.

\[
V(M, N, P);
\]

**SELECT * FROM V WHERE V.M = 33**

**INDEX I1 on V(M)**

Scan V
For each record:
if M=33 then output

Lookup key 33 in I1
For each record: output

**SELECT * FROM V WHERE V.M = 33 and V.P = 55**

Scan V
For each record:
if M=33 and P=55 then output

Lookup key 33 in I1
For each record
if P=55 then output
Index Selection Problem 1

V(M, N, P);

Your workload is this (and nothing else)

100000 queries:

SELECT *
FROM V
WHERE N=?

100 queries:

SELECT *
FROM V
WHERE P=?

Which indexes?
Index Selection Problem 1

V(M, N, P);

Your workload is this (and nothing else)

100000 queries: 100 queries:

SELECT *
FROM V
WHERE N=?

SELECT *
FROM V
WHERE P=?

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

Which indexes?
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)
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 (?, ?, ?)

Which indexes?
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) (B-tree)

How does this index differ from:
1. Two indexes V(N) and V(P)?
2. An index V(P, N)?
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<?

Which indexes?
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 (both B-tree)
V(M, N, P);

Suppose the database has these indexes. Which ones can the optimizer use?

INDEX I1 on V(M)
INDEX I2 on V(M, P)
INDEX I3 on V(P, M)

SELECT *
FROM V
WHERE V.M = 33

SELECT *
FROM V
WHERE V.M = 33 and V.P = 55
Recap – Indexes

V(M, N, P);

Suppose the database has these indexes. Which ones can the optimizer use?

INDEX I1 on V(M)
INDEX I2 on V(M, P)
INDEX I3 on V(P, M)

SELECT *
FROM V
WHERE V.M = 33

SELECT *
FROM V
WHERE V.M = 33 and V.P = 55

Yes
Suppose the database has these indexes. Which ones can the optimizer use?

- INDEX I1 on V(M)
- INDEX I2 on V(M, P)
- INDEX I3 on V(P, M)

Yes

SELECT *
FROM V
WHERE V.M = 33

Yes (why?)

SELECT *
FROM V
WHERE V.M = 33 and V.P = 55

Yes
Recap – Indexes

V(M, N, P);

SELECT *
FROM V
WHERE V.M = 33

Suppose the database has these indexes. Which ones can the optimizer use?

INDEX I1 on V(M)
INDEX I2 on V(M, P)
INDEX I3 on V(P, M)

No! (why?)

SELECT *
FROM V
WHERE V.M = 33 and V.P = 55

Yes
Recap – Indexes

<table>
<thead>
<tr>
<th>Movie(mid, title, year)</th>
<th>CLUSTERED INDEX I on Movie(id)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>INDEX J on Movie(year)</td>
</tr>
</tbody>
</table>

SELECT *
FROM Movie
WHERE year = 2010

SELECT *
FROM Movie
WHERE year = 1910

The system uses the index J for one of the queries, but not for the other.

Which and why?
Basic Index Selection Guidelines

- Consider queries in workload in order of importance
  - ignore infrequent queries if you also have many writes

- 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

• Covering indexes do *not* need to be clustered: they work equally well unclustered
  – (a covering index for a query is one where every attribute mentioned in the query is part of the index’s search key)
  – in that case, index has all the info you need anyway
The query returns only a few records

The query returns almost all records in R
SELECT * FROM R WHERE K>? and K<?

CSE 414 - Fall 2017
Midterm Concept Review I

• relational data model
  – set semantics vs. bag semantics
  – primary & secondary keys
  – foreign keys
  – schemas

• SQL
  – CREATE TABLE
  – SELECT-FROM-WHERE (SFW)
  – joins: inner vs. outer, natural
  – group by & aggregation
  – ordering
  – CREATE INDEX
Midterm Concept Review II

• relational queries
  – languages for writing them:
    • standard relational algebra
    • Datalog (even without recursion)
    • SQL (even without grouping / aggregation)
  – monotone queries are a proper subset
  – SFW queries (i.e., w/out subqueries) are monotone
Midterm Concept Review III

• types of indexes
  – B+ tree vs. hash
    • hash indexes use at most 2 disk accesses
    • B+ tree can be used for < predicates
    • B+ tree index on (X, Y) also allows searching for X=a matches
  – clustered vs non-clustered
    • selectivity above 1-2% => not helped by non-clustered indexes

• cost-based query optimization
  – consider choices over logical and physical query plans
    • most important choice in latter is choice of join algorithm
    • those include nested loop, sorted merge, hash, and indexed joins
  – primary goal of the optimizer is to avoid really bad plans