

Database Systems CSE 414

Lecture 10:
Basics of Data Storage and Indexes

CSE 414 - Spring 2016 1

Reminder

- HW3 is due next Tuesday

CSE 414 - Spring 2016 2

Motivation

- My database application is too slow... why?
- One of the queries is very slow... why?
- ...
- To understand performance, need to understand how a DBMS works

CSE 414 - Spring 2016 3

Data Storage

- DBMSs store data in **files**
- Most common organization is **row-wise storage**
- On disk, a file is split into **blocks**
- Each block contains a set of tuples

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

10	Tom	Hanks
20	Amy	Hanks
50
200
220		
240		
420		
800		

In the example, we have **4 blocks** with 2 tuples each

CSE 414 - Spring 2016 4

Data File Types

The data file can be one of:

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

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

CSE 414 - Spring 2016 5

Data File Types

The data file can be one of:

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

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

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 running on our database.

CSE 414 - Spring 2016 6

Index

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

CSE 414 - Spring 2016 7

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

CSE 414 - Spring 2016 8

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

CSE 414 - Spring 2016 9

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 - Spring 2016

Example 1: Index on ID

Index **Student_ID** on **Student.ID**

10
20
50
200
220
240
420
800
...
560
...

Data File **Student**

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...
220
240
...
420
800

CSE 414 - Spring 2016 11

Example 2: Index on fName

Index **Student_fName** on **Student.fName**

Amy
Ann
Bob
Cho
...
...
...
...
...
Tom

Data File **Student**

ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...
50
200
220
240
...
420
800

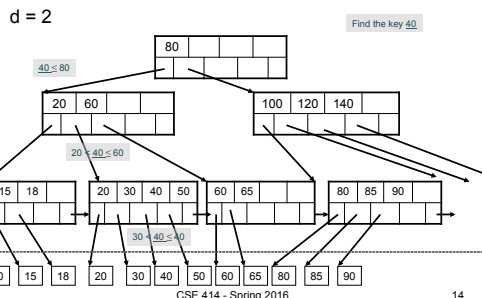
CSE 414 - Spring 2016 12

Index Organization

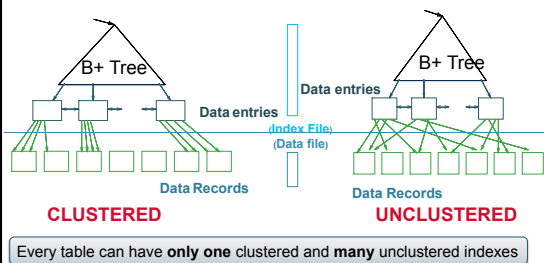
Several index organizations:

- Hash table
- B+ trees – most popular
 - They are search trees, but they are not binary instead have higher fanout
 - Will discuss them briefly next
- Specialized indexes: bit maps, R-trees, inverted index

B+ Tree Index by Example



Clustered vs Unclustered



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

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

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


- 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, ...



CSE 414 - Spring 2016 19

Scanning a Data File

- 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)
- Solid state (SSD): \$\$\$ expensive; put indexes, other "hot" data there, not enough room for everything



CSE 414 - Spring 2016 20

Example

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

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

Assume the database has indexes on these attributes:

- **index_takes_courseID** = index on Takes.courseID
- **index_student_ID** = index on Student.ID

for y1 in index_Takes_courseID where y1.courseID = 300
for y in y1.Takes
for x1 in index_student_ID where x1.ID = y.studentID
for x in x1.Student
output x.*,y.*

CSE 414 - Spring 2016 21

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)
```

CSE 414 - Spring 2016 22

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)
```

What does this mean?

```
CREATE INDEX V3 ON V(M, N)
```

```
CREATE UNIQUE INDEX V4 ON V(N)
```

```
CREATE CLUSTERED INDEX V5 ON V(N)
```

CSE 414 - Spring 2016 23

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)
```

What does this mean?

```
CREATE INDEX V3 ON V(M, N)
```

```
CREATE UNIQUE INDEX V4 ON V(N)
```

```
CREATE CLUSTERED INDEX V5 ON V(N)
```

Not supported in SQLite

CSE 414 - Spring 2016 24

Which Indexes?

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- How many indexes **could** we create?
- Which indexes **should** we create?

25

Which Indexes?

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- How many indexes **could** we create?

18, namely: (ID), (fName), (lName), (ID,fName),(fName,ID),...

- Which indexes **should** we create?

26

Which Indexes?

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- How many indexes **could** we create?

18, namely: (ID), (fName), (lName), (ID,fName),(fName,ID),...

- Which indexes **should** we create?

Few! Each new index slows down updates to Student

27

Which Indexes?

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- How many indexes **could** we create?

18, 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

28

Which Indexes?

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- 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:
 - The database administrator DBA
 - Semi-automatically, using a database administration tool

29

Which Indexes?

Student		
ID	fName	lName
10	Tom	Hanks
20	Amy	Hanks
...		

- 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:
 - The database administrator DBA
 - Semi-automatically, using a database administration tool

30

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

CSE 414 - Spring 2016 31

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=?
```

CSE 414 - Spring 2016 32

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 ?

CSE 414 - Spring 2016 33

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)

CSE 414 - Spring 2016 34

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 ?

CSE 414 - Spring 2016 35

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)

CSE 414 - Spring 2016 36

The Index Selection Problem 3

V(M, N, P);

Your workload is this
 100000 queries: 1000000 queries: 100000 queries:

SELECT *
FROM V
WHERE N=?

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

INSERT INTO V
VALUES (?, ?, ?)

What indexes ?

CSE 414 - Spring 2016 37

The Index Selection Problem 3

V(M, N, P);

Your workload is this
 100000 queries: 1000000 queries: 100000 queries:

SELECT *
FROM V
WHERE N=?

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

INSERT INTO V
VALUES (?, ?, ?)

A: V(N, P)

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

CSE 414 38

The Index Selection Problem 4

V(M, N, P);

Your workload is this
 1000 queries: 100000 queries:

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

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

What indexes ?

CSE 414 - Spring 2016 39

The Index Selection Problem 4

V(M, N, P);

Your workload is this
 1000 queries: 100000 queries:

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

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

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

CSE 414 - Spring 2016 40

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

CSE 414 - Spring 2016 41

To Cluster or Not

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

CSE 414 - Spring 2016 42

