

Introduction to Database Systems CSE 414

Lecture 25: Basics of Data Storage and Indexes

CSE 414 - Spring 2018

1


Announcements

- HW8 and WQ7
 - Due on 5/30
- OH changes
 - Alvin will be away next Wed
 - Jonathan will give next Wed's lecture
- Final on Thurs 6/7
 - Final review on 6/3 afternoon

CSE 414 - Spring 2018

2

Recap: Transactions

- Protocols discussed:
 - Nothing 
 - 2PL → unrecoverable schedules
 - Strict 2PL → phantom problem
 - Predicate locking → expensive!
- Recall our execution model!

CSE 414 - Spring 2018

3

Isolation Levels in SQL

1. "Dirty reads"
`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED`
2. "Committed reads"
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED`
3. "Repeatable reads"
`SET TRANSACTION ISOLATION LEVEL REPEATABLE READ`
4. Serializable transactions
`SET TRANSACTION ISOLATION LEVEL SERIALIZABLE`

Try these in HW8!

4

Beware!

In commercial DBMSs:

- Default level is often NOT serializable
- Default level differs between DBMSs
- Some engines support subset of levels!
- Serializable may not be exactly ACID
 - Locking ensures isolation, not atomicity
- Also, some DBMSs do NOT use locking and different isolation levels can lead to different pbs
- **Bottom line: RTFM for your DBMS!**

CSE 414 - Spring 2018

5

Class Overview

- Unit 1: Intro
- Unit 2: Relational Data Models and Query Languages
- Unit 3: Non-relational data
- Unit 4: RDBMS internals and query optimization
- Unit 5: Parallel query processing
- Unit 6: DBMS usability, conceptual design
- Unit 7: Transactions
- Unit 8: Advanced topics: Query optimization

6

Query Performance

- My database application is too slow... why?
 - One of the queries is very slow... why?
 - To understand performance, we need to understand:
 - How is data organized on disk
 - How to estimate query costs
- In this course we will focus on **disk-based DBMSs**

CSE 414 - Spring 2018

7

Data Storage

Student

| ID | fName | lName |
|-----|-------|-------|
| 10 | Tom | Hanks |
| 20 | Amy | Hanks |
| ... | | |

- 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

| | | | |
|-----|-----|-------|---------|
| 10 | Tom | Hanks | block 1 |
| 20 | Amy | Hanks | |
| 50 | ... | ... | block 2 |
| 200 | ... | ... | |
| 220 | | | block 3 |
| 240 | | | |
| 420 | | | |
| 800 | | | |

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

CSE 414 - Spring 2018

8

Data File Types

Student

| ID | fName | lName |
|-----|-------|-------|
| 10 | Tom | Hanks |
| 20 | Amy | Hanks |
| ... | | |

The data file can be one of:

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

CSE 414 - Spring 2018

9

Data File Types

Student

| ID | fName | lName |
|-----|-------|-------|
| 10 | Tom | Hanks |
| 20 | Amy | Hanks |
| ... | | |

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 running on our database.

Index

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

CSE 414 - Spring 2018

11

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 2018

12

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 2018

13

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



This is not a pipe.

CSE 414 - Spring 2018

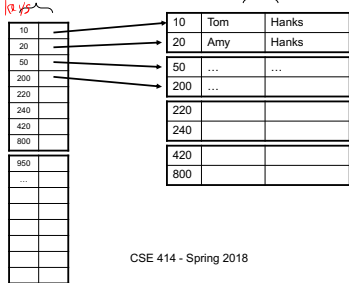


Example 1: Index on ID

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

Data File **Student**

| ID | fName | lName |
|-----|-------|-------|
| 10 | Tom | Hanks |
| 20 | Amy | Hanks |
| ... | ... | ... |



CSE 414 - Spring 2018

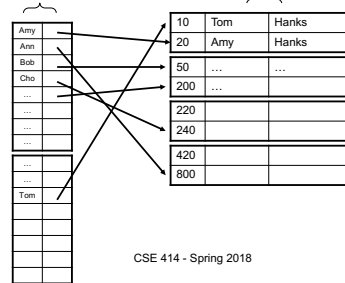
15

Example 2: Index on fName

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

Data File **Student**

| ID | fName | lName |
|-----|-------|-------|
| 10 | Tom | Hanks |
| 20 | Amy | Hanks |
| ... | ... | ... |



CSE 414 - Spring 2018

16

Index Organization

We need a way to represent indexes after loading into memory so that they can be used
Several ways to do this:

- 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

CSE 414 - Spring 2018

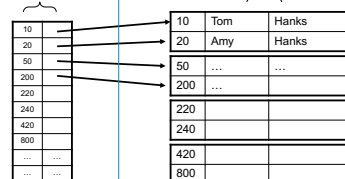
17

Hash table example

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

Data File **Student**

| ID | fName | lName |
|-----|-------|-------|
| 10 | Tom | Hanks |
| 20 | Amy | Hanks |
| ... | ... | ... |

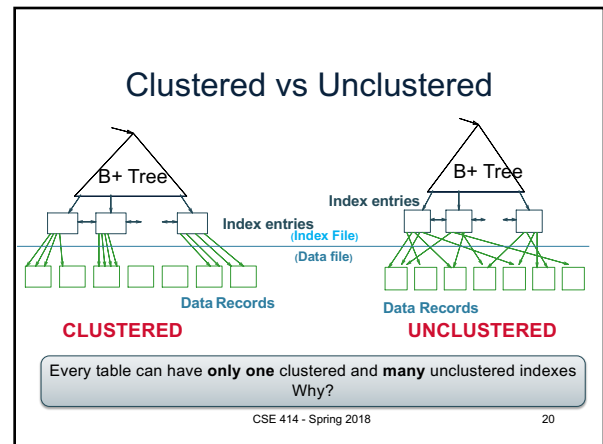
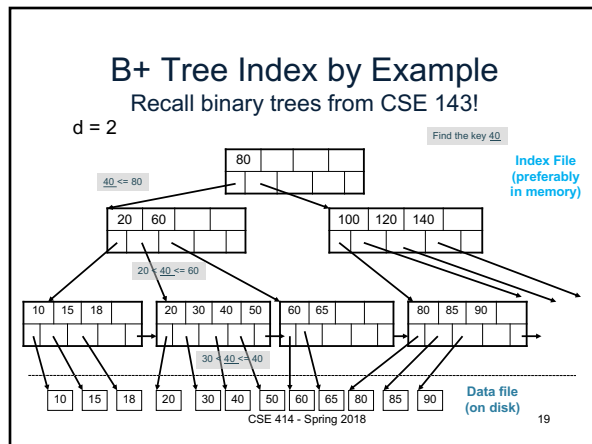


Index File
(preferably
in memory)

Data file
(on disk)

CSE 414 - Spring 2018

18



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

CSE 414 - Spring 2018

21

Scanning a Data File

- Disks are mechanical devices!
 - Technology from the 60s; density much higher now
- 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 \approx 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, still too expensive for everything

CSE 414 - Spring 2018

22

Example

Student(ID, fname, lname)
Takes(studentID, courseID)

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

CSE 414 - Spring 2018

23

Example

Student(ID, fname, lname)
Takes(studentID, courseID)

```
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 *
```

CSE 414 - Spring 2018

24

Student(ID, fname, lname)
Takes(studentID, courseID)

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

Example

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:

- Takes_courseID = index on Takes.courseID
- Student_ID = index on Student.ID

CSE 414 - Spring 2018 25

Student(ID, fname, lname)
Takes(studentID, courseID)

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

Example

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:

- Takes_courseID = index on Takes.courseID
- Student_ID = index on Student.ID

for y' in Takes_courseID where y'.courseID > 300

CSE 414 - Spring 2018 26

Student(ID, fname, lname)
Takes(studentID, courseID)

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

Example

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:

- Takes_courseID = index on Takes.courseID
- Student_ID = index on Student.ID

Index selection

for y' in Takes_courseID where y'.courseID > 300
y = fetch the Takes record pointed to by y'

CSE 414 - Spring 2018 27

Student(ID, fname, lname)
Takes(studentID, courseID)

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

Example

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:

- Takes_courseID = index on Takes.courseID
- Student_ID = index on Student.ID

Index selection

for y' in Takes_courseID where y'.courseID > 300
y = fetch the Takes record pointed to by y'
for x' in Student_ID where x'.ID = y.studentID
x = fetch the Student record pointed to by x'

Index join

CSE 414 - Spring 2018 28

Student(ID, fname, lname)
Takes(studentID, courseID)

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

Example

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:

- Takes_courseID = index on Takes.courseID
- Student_ID = index on Student.ID

Index selection

for y' in Takes_courseID where y'.courseID > 300
y = fetch the Takes record pointed to by y'
for x' in Student_ID where x'.ID = y.studentID
x = fetch the Student record pointed to by x'
output *

Index join

CSE 414 - Spring 2018 29

Student(ID, fname, lname)
Takes(studentID, courseID)

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

Example

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:

- Takes_courseID = index on Takes.courseID
- Student_ID = index on Student.ID

Index selection

for y' in Takes_courseID where y'.courseID > 300
y = fetch the Takes record pointed to by y'
for x' in Student_ID where x'.ID = y.studentID
x = fetch the Student record pointed to by x'
output *

Index join

CSE 414 - Spring 2018 30

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 2018

31

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 2018

32

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)
```

yes

select *
from V
where P=55 and M=77

What does this mean?

CSE 414 - Spring 2018

33

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)
```

yes

select *
from V
where P=55 and M=77

What does this mean?

select *
from V
where P=55

CSE 414 - Spring 2018

34

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)
```

yes

select *
from V
where P=55 and M=77

What does this mean?

select *
from V
where P=55

yes

CSE 414 - Spring 2018

35

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)
```

yes

select *
from V
where P=55 and M=77

What does this mean?

select *
from V
where P=55

yes

select *
from V
where M=77

CSE 414 - Spring 2018

36

Getting Practical: Creating Indexes in SQL

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

yes

```
CREATE INDEX V1 ON V(N)
```

select *
from V
where P=55 and M=77

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

What does this mean?

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

select *
from V
where P=55

yes

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

select *
from V
where M=77

no

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

CSE 414 - Spring 2018

37

Getting Practical: Creating Indexes in SQL

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

yes

```
CREATE INDEX V1 ON V(N)
```

select *
from V
where P=55 and M=77

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

What does this mean?

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

select *
from V
where P=55

yes

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

select *
from V
where M=77

no

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

Not supported
in SQLite

CSE 414 - Spring 2018

38