Lecture 15:
Query Processing & Indexes

Monday, March 23, 2015

Where we are

- Annotated slides on concurrency control
- HW 3 is over! Now focus on class project
- Today: Query processing and indexes

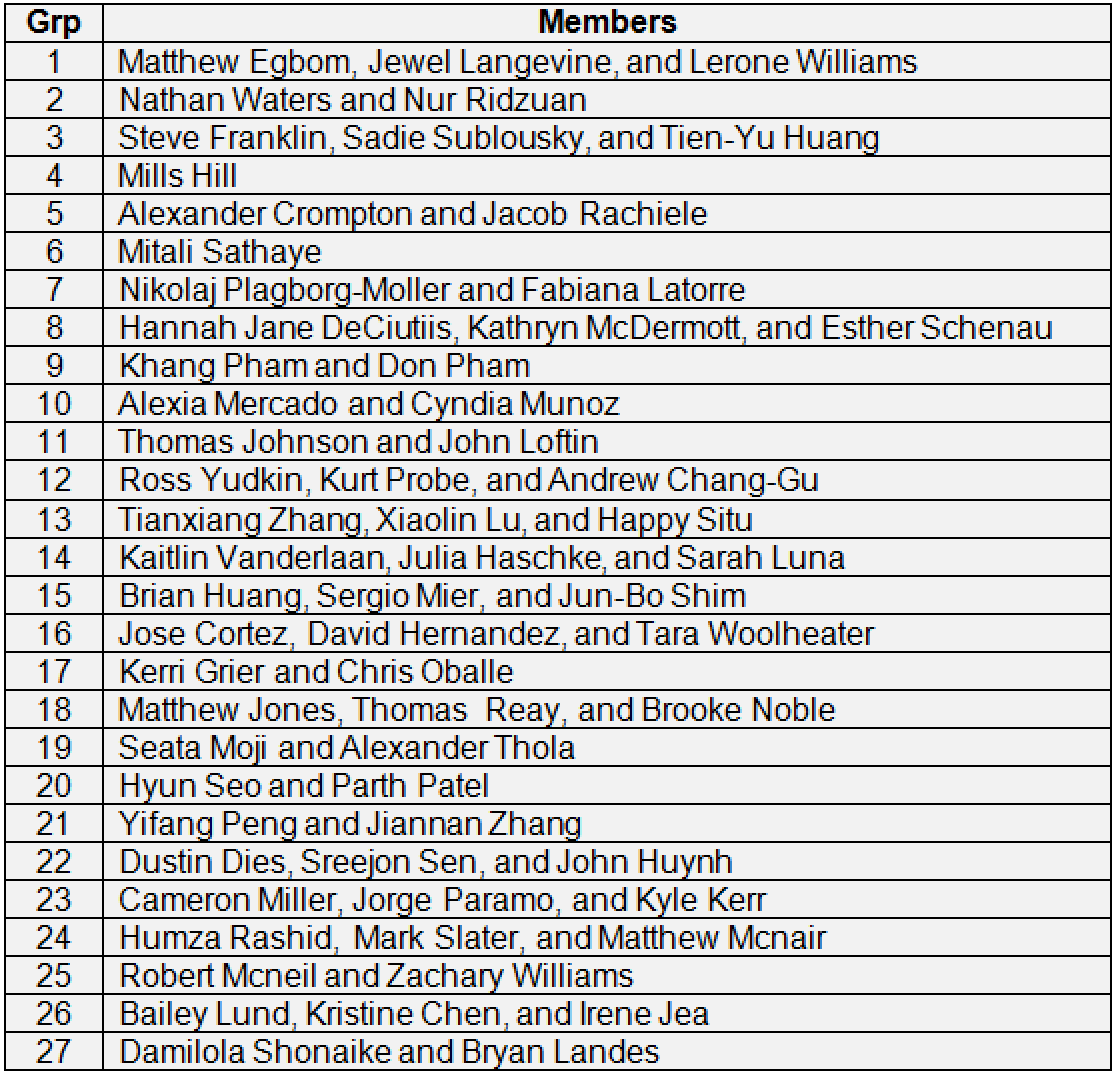
Class Project Schedule

- **Project ERD and SQL feedback**
  - Replied to your emails with my comments

- **Support sessions (only this week):**
  - SQL*Loader tutorial
  - cx_Oracle catch-up (outstanding issues with HW #3)

- **Upcoming schedule:**
  - Class presentations on 03/30, 04/01, and 04/06
    - Groups 1 – 9 on 03/30
    - Groups 10 – 18 on 04/01
    - Groups 19 – 27 on 04/06
  - Final submissions due on 04/06

Project Groups



Grp	Members
1	Matthew Egbom, Jewel Langevine, and Lerone Williams
2	Nathan Waters and Nur Ridzuan
3	Steve Franklin, Sadie Sublousky, and Tien-Yu Huang
4	Mills Hill
5	Alexander Crompton and Jacob Rachiele
6	Mitali Sathaye
7	Nikolaj Plagborg-Moller and Fabiana Latorre
8	Hannah Jane DeCiutiis, Kathryn McDermott, and Esther Schenau
9	Khang Pham and Don Pham
10	Alexia Mercado and Cyndia Munoz
11	Thomas Johnson and John Loftin
12	Ross Yudkin, Kurt Probe, and Andrew Chang-Gu
13	Tianxiang Zhang, Xiaolin Lu, and Happy Situ
14	Kaitlin Vanderlaan, Julia Haschke, and Sarah Luna
15	Brian Huang, Sergio Mier, and Jun-Bo Shim
16	Jose Cortez, David Hernandez, and Tara Woolheater
17	Kerri Grier and Chris Oballe
18	Matthew Jones, Thomas Reay, and Brooke Noble
19	Seata Moji and Alexander Thola
20	Hyun Seo and Parth Patel
21	Yifang Peng and Jiannan Zhang
22	Dustin Dies, Sreejon Sen, and John Huynh
23	Cameron Miller, Jorge Paramo, and Kyle Kerr
24	Humza Rashid, Mark Slater, and Matthew Mcnair
25	Robert Mcneil and Zachary Williams
26	Bailey Lund, Kristine Chen, and Irene Jea
27	Damilola Shonaike and Bryan Landes

Project Presentation

- **10 minutes** per project: 7 minutes presentation plus 3 minutes for questions.
- Suggested content:

  -describe the problem

  -describe your approach

  -give short demo

  -discuss unexpected issues or problems

  -discuss possible extensions

Final Project Submission

- A one page report on how the project was implemented and how it works internally.
- End-user documentation (instructions and examples on how somebody can use this project)
- Submit all code including dataset and test cases
- Submission deadline is **04/06 at 11:59pm**

Query Processing without Indexes

Customers (id, first_name, last_name, address, city)

```
SELECT  *
FROM    Customers
WHERE   city = ‘Austin’
```

Question: How do we evaluate this query?

Query Processing without Indexes

Customers (id, first_name, last_name, address, city)

```
SELECT  *
FROM    Customers
WHERE   city = ‘Austin’
```

Question: How do we evaluate this query?

Problem: it takes too long to scan the entire Customers table

Query Processing without Indexes

Customers (id, first_name, last_name, address, city)

```
SELECT  *
FROM    Customers
WHERE   city = ‘Austin’
```

Question: How do we evaluate this query?
Problem: it takes too long to scan the entire Customers table

Orders (id, order_date, ship_date, customer_id)

```
SELECT  *
FROM Customers c, Orders o
WHERE c.id = o.customer_id
AND c.city = ‘Austin’
AND o.order_date BETWEEN ‘01-FEB-2015’ AND ‘28-FEB-2015’
```

Questions: How do we evaluate this query? How can we speed this up?

Indexes

- **Critical** to database systems
- At least one index per table
- They work “behind the scenes”
- DBA looks at the workload and decides which indexes to create (no easy answers)
- Creating indexes can be an expensive operation
- Query optimizer decides which indexes to use during query execution
- Primary keys are automatically indexed
- Indexes are updated during a transaction

Creating Indexes

Customers (id, first_name, last_name, address, city)

```
SELECT  *
FROM     Customers
WHERE   city = 'Austin'
```

Problem: it takes too long to scan the entire Customers table

Solution: create an index on the city column

```
CREATE INDEX  cust_city_indx ON Customers(city)
```

Now the above query runs much faster

Creating Indexes

Indexes can be created on more than one attribute:

Example:

```
CREATE INDEX cust_city_indx ON Customers (city, last_name)
```

Helps with:

```
SELECT *

FROM Customers

WHERE city = 'Austin' AND last_name = 'Johnson'
```

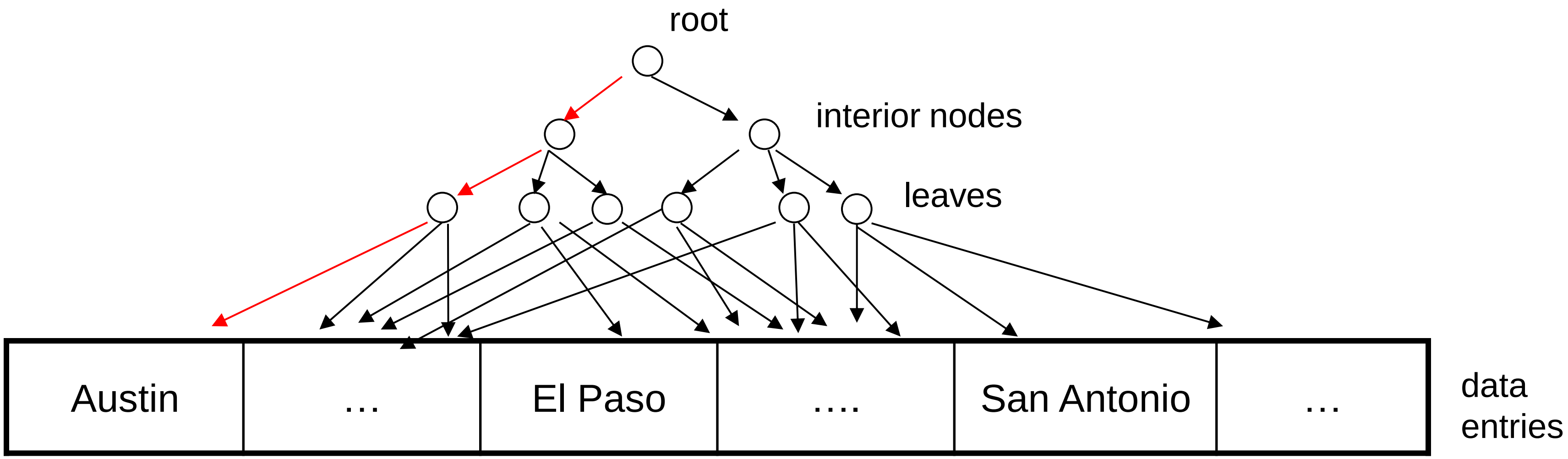
Even helps with:

```
SELECT *

FROM Customers

WHERE city = 'Austin'
```

B+ Tree



root

interior nodes

leaves

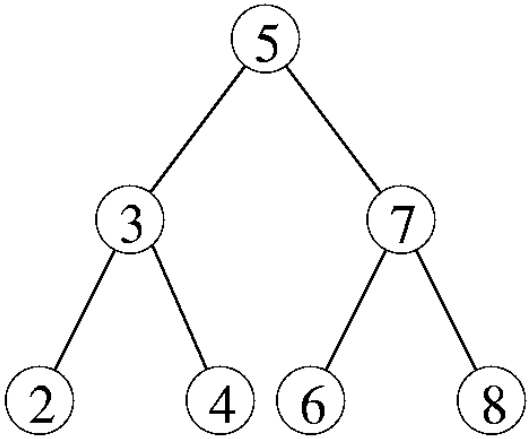
Austin | … | El Paso | …. | San Antonio | …

data entries

- B+ Tree = Balanced search tree
- The index is a separate file that is essentially organized as a table: Index(search_key, *record(s))
- Given a search_key, the index returns pointers to the records
- Search_key can be an attribute, collection of attributes of even an expression

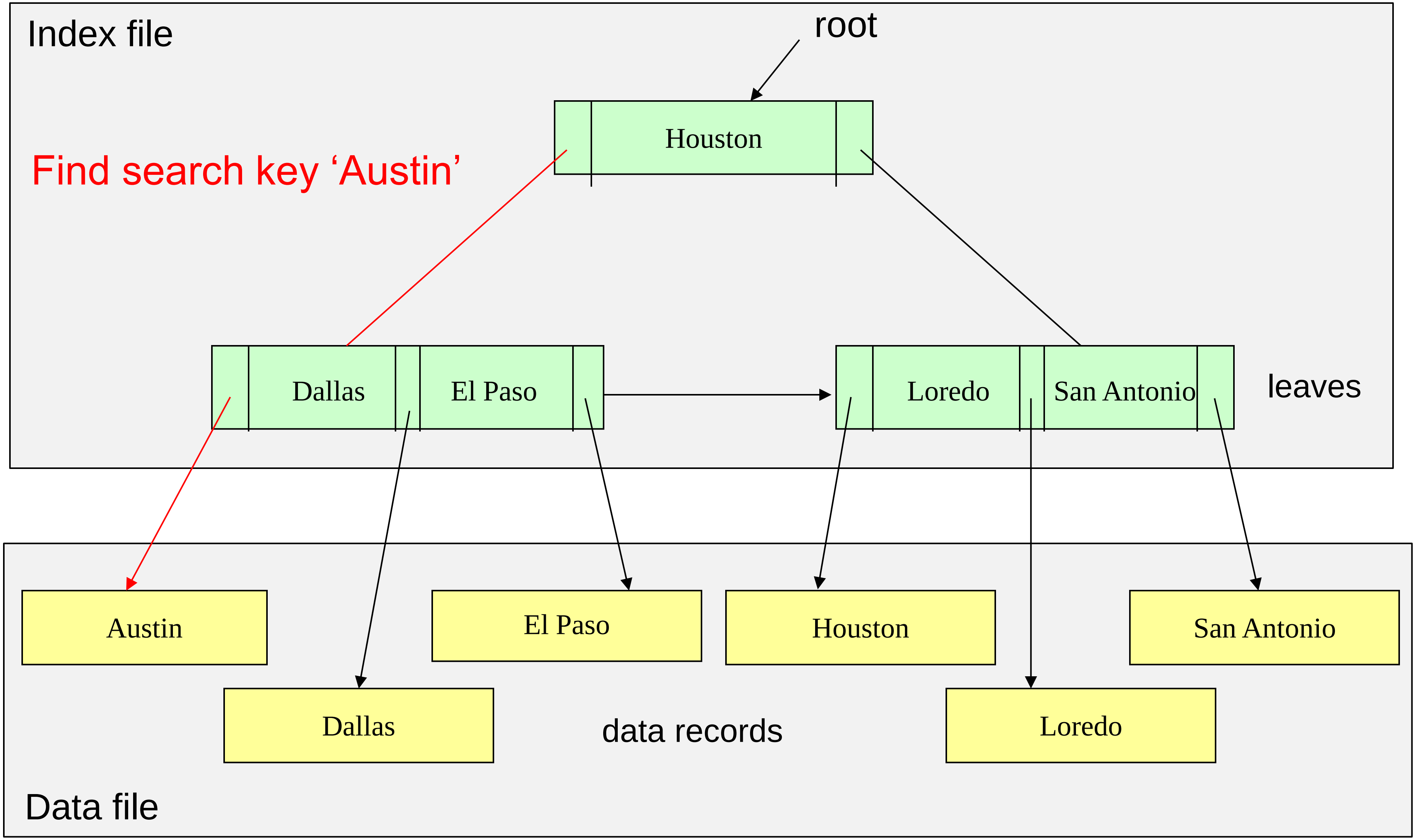
Note that the search key is not the same as the key of a table

Why not use Binary Search Trees?



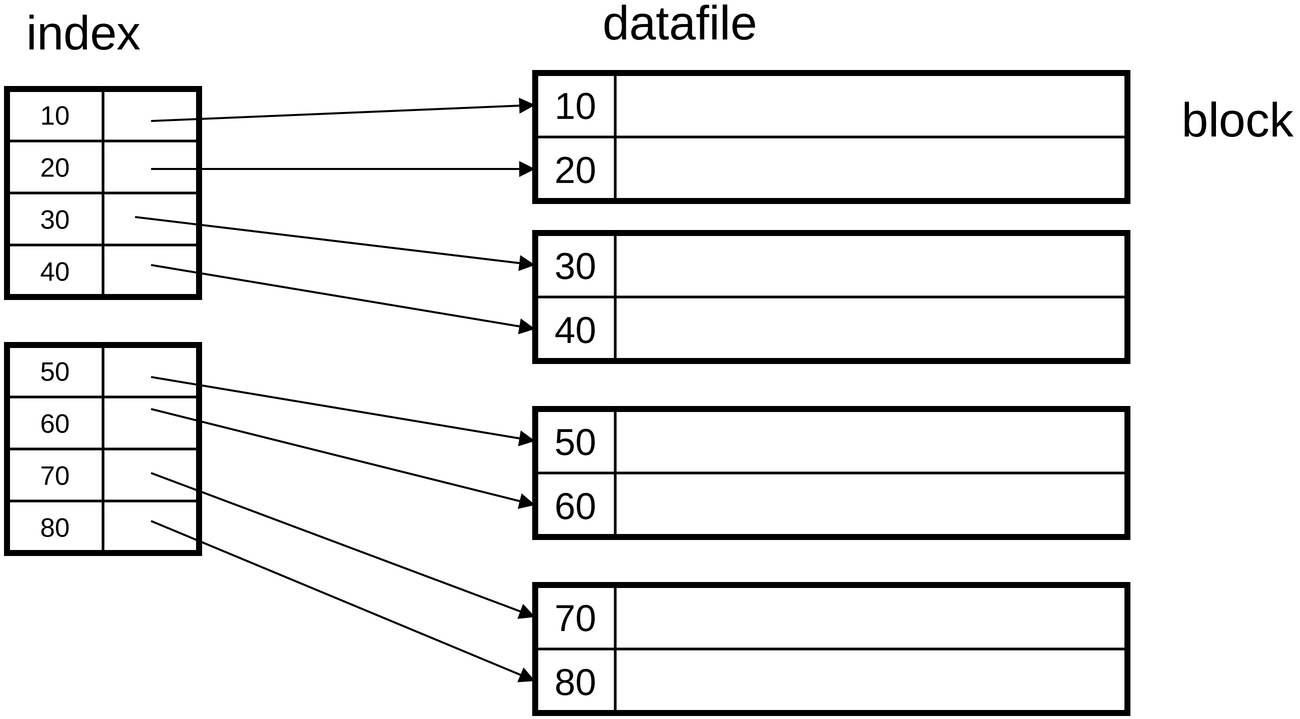
- Nodes in a binary tree only have a single key = too small for databases
- In databases, index tree assumed to be on disk (not main memory)
- Want each node in the index to be as wide as a block
- Due to the cost of reading from disk, want to use the information stored in a block as aggressively as possible

B+ Tree Example



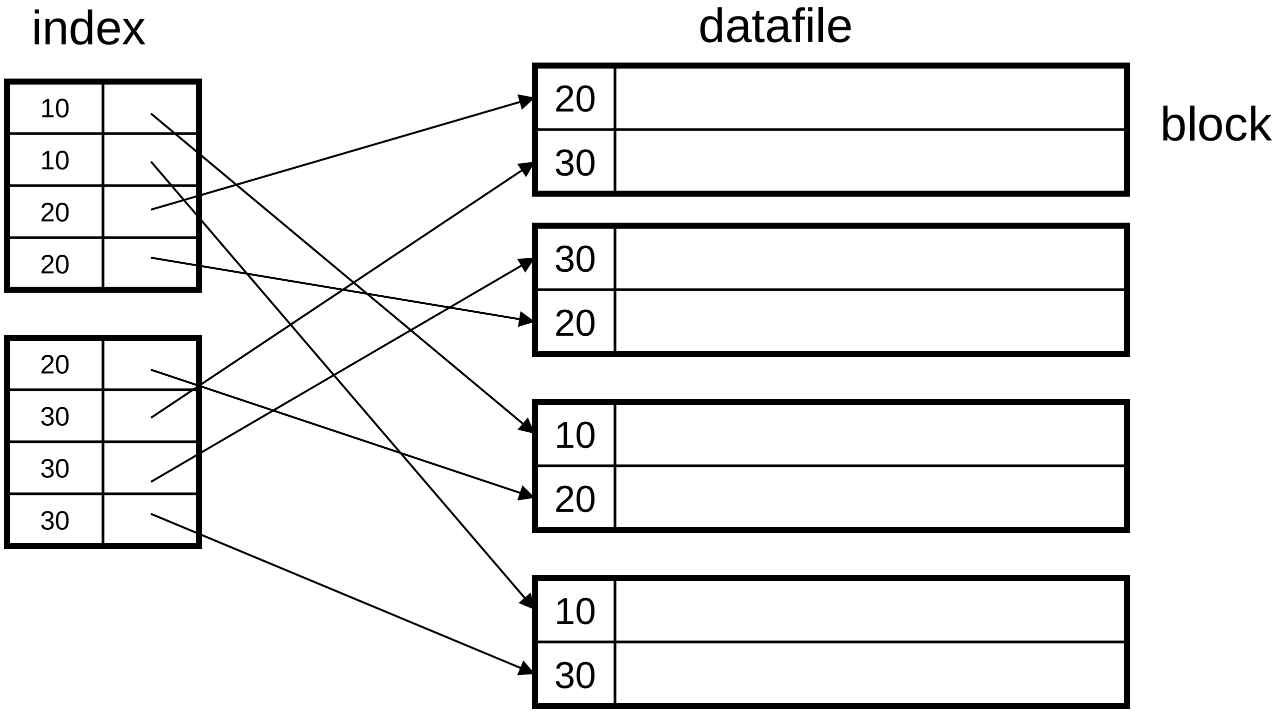
Clustered Indexes

- Datafile is sorted on the index attribute
- Only one clustered index per table
- Known as Index Organized Table (IOT) in Oracle



Unclustered Index

- Can have multiple unclustered indexes per table
- Separate index and data files



Question: when does it make sense to ignore an unclustered index?

Query Processing with B+ Trees

Customers (id, first_name, last_name, address, city)

```
CREATE INDEX  cust_city_indx ON Customers(city)
```

```
SELECT  last_name
FROM     Customers
WHERE   city = ‘Austin’
```

Question: How do we use the index to answer this query?

Query Processing with B+ Trees

Customers (id, first_name, last_name, address, city)

```
CREATE INDEX  cust_city_indx ON Customers(city)
```

```
SELECT  last_name
FROM    Customers
WHERE   city = 'Austin'
```

Question: How do we use the index to answer this query?

Answer:

- Start at the root of the B+ tree
- Search the index for the key ‘Austin’
- Once we find the key ‘Austin’, follow pointers to all data records

Question: Why do we have multiple pointers?

Query Processing with B+ Trees

Customers (id, first_name, last_name, address, city)

```
CREATE INDEX  cust_last_name_indx ON Customers(last_name)
```

```
SELECT   *
FROM     Customers
WHERE    last_name
BETWEEN ‘Johnson’ AND ‘Jones’
```

Question: How can we use the index to answer this range query?

Query Processing with B+ Trees

Customers (id, first_name, last_name, address, city)

```
CREATE INDEX  cust_last_name_indx ON Customers(last_name)
```

```
SELECT  *
FROM     Customers
WHERE   last_name
BETWEEN ‘Johnson’ AND ‘Jones’
```

Question: How can we use the index to answer this range query?

Answer:

- Start at the root of the B+ tree
- Search for the key ‘Johnson’, the lower bound of the range
- Once we’ve reached the key for ‘Johnson’, follow the pointers to the right, examining their search keys until we’ve passed ‘Jones’, the upper bound of the range

Query Processing with B+ Trees

Customers (id, first_name, last_name, address, city)

```
CREATE INDEX  cust_last_name_indx ON Customers(last_name)
```

```
SELECT  DISTINCT last_name
FROM     Customers
```

Question: How can we use the index to answer this query?

Query Processing with B+ Trees

Customers (id, first_name, last_name, address, city)

```
CREATE INDEX  cust_last_name_indx ON Customers(last_name)
```

```
SELECT  DISTINCT last_name
FROM     Customers
```

Question: How can we evaluate this query?

Answer:

- Scan the index for all the last name values.

Note: we don’t need to access the table to answer this query

Query Processing with B+ Trees

Customers (id, first_name, last_name, address, city)

```
CREATE INDEX  cust_city_last_name_indx
ON Customers(city, last_name)
```

A composite index that is sorted first by city and second by last_name.

```
SELECT  *  FROM  Customers
WHERE  city = ‘Austin’ AND last_name = ‘Johnson’
```

```
SELECT  * FROM  Customers WHERE city = ‘Austin’
```

```
SELECT  * FROM  Customers WHERE last_name = ‘Johnson’
```

We can use the index to answer to first two queries. We can’t use it to answer the last query though because the last_name values are scattered across the index.

Optional References

- Douglas Comer. “The Ubiquitous B-Tree”. ACM Computing Survey. 11(2): 121-137 (1979).

- R. Ramakrishnan and J. Gehrke. Database Management Systems (3rd edition). McGraw-Hill 2003.

Next class

- Views
- Quiz #5