

# Indexing & Storage Engines

November 12, 2019

## Chapter 8

# Join

What happens? Primary Key? Index?

Professors

<u>ProfID</u>	FName	LName	StartDate	StatusID
1	D	R	1999-08-01	3
2	S	K	2002-08-01	3
3	C	W	2006-08-01	2
4	C	L	1999-08-01	2

JobStatus

<u>StatusID</u>	Name	PayBonus	Tenure
1	Professor	10000	Yes
2	Associate	1000	Yes
3	Assistant	0	No

```
select * from Prof_Ex where ProfID=1;
```

```
select * from Prof_Ex where StatusID=3;
```

```
select * from JobStatus Where PayBonus > 100;
```

```
select * from Prof_Ex where StartDate > "2001-01-01";
```

```
select * from Prof_Ex, JobStatus where Prof_Ex.StatusID=JobStatus.StatusID;
```

# MySQL/MariaDB Explain

- Join Types

<https://dev.mysql.com/doc/refman/5.5/en/estimating-performance.html>

<https://dev.mysql.com/doc/refman/5.5/en/explain-output.html#explain-join-types>

# Hardware Basics

- Disk access time: ~10 msecs
- Memory access time: 60 nanoseconds
  - faster than disk access by ???
- We can run many instructions in 10 msecs!
- What does it cost to find a row?

# Storage Engine

- How is the data stored?
  - file format
  - indexes
  - transactions/concurrency
- MySQL/MariaDB ships with a number of storage engines
  - MyISAM / Aria
  - InnoDB / XtraDB
  - plug-ins can add support for others

<https://mariadb.com/kb/en/library/choosing-the-right-storage-engine/>

<https://dev.mysql.com/doc/refman/5.7/en/innodb-introduction.html>

# InnoDB Transactions

- **A**tomic - all changes are either committed as a group, or all are rolled back as a group
- **C**onsistent - transactions operate on a consistent view of the data, leaving the data in a consistent state (by transaction's end)
- **I**solated - each transaction “thinks” it is running by itself - effects of other transactions are invisible until it commits
- **D**urable - once committed, all changes persist, even if there are system failures

<http://www.innodb.com/wp/wp-content/uploads/2008/04/intro-to-innodb-at-the-2008-mysql-uc-final.pdf>

# Indexing

```
mysql> CREATE TABLE Actors
      (ActorID INT NOT NULL AUTO_INCREMENT,
      LastName VARCHAR(50),
      FirstName VARCHAR(50) NOT NULL,
      BirthDay DATE NOT NULL,
      PRIMARY KEY (ActorID),
      INDEX (BirthDay)
      ) ENGINE=InnoDB;
```

- Common access methods
  - Scan
  - Equality
  - Range

<http://www.innodb.com/products/innodb/info/>  
Intro to InnoDB at the 2008 MySQL User Conference

# Database Files

- Data File – data from one table
  - Collection of file pages
    - Each page contains a number of data records
    - InnoDB: 16KB page size
    - One disk access to retrieve each page
  - Data records
    - 1 record = 1 row in a table
    - Each data record has a record id (**rid**) <pageid, slotid>
    - Can be used to retrieve the record
    - Heap file:
- Index File
  - Auxiliary file that matches database indexes to rids

Assume each index is tied to exactly 1 column in the table



# Index Files

- Match index value to rid
- Three types:
  - 1 The data entry is the database row
    - No auxiliary file
    - Called an indexed file
  - 2 The data entry is a <db index, rid> pair
  - 3 The data entry is a <db index, rid-list> pair
- For any table, you can have one indexed file and many of 2 or 3
- Primary & Secondary indexes

# Clustered Indexes

- Data records stored in near sorted order
  - Records in a page are nearly ordered
- Generally, only option 1 is clustered
  - Expensive to keep a file sorted
  - often gaps are kept in the file to allow easy (sorted) insertion
- Why would this be useful?

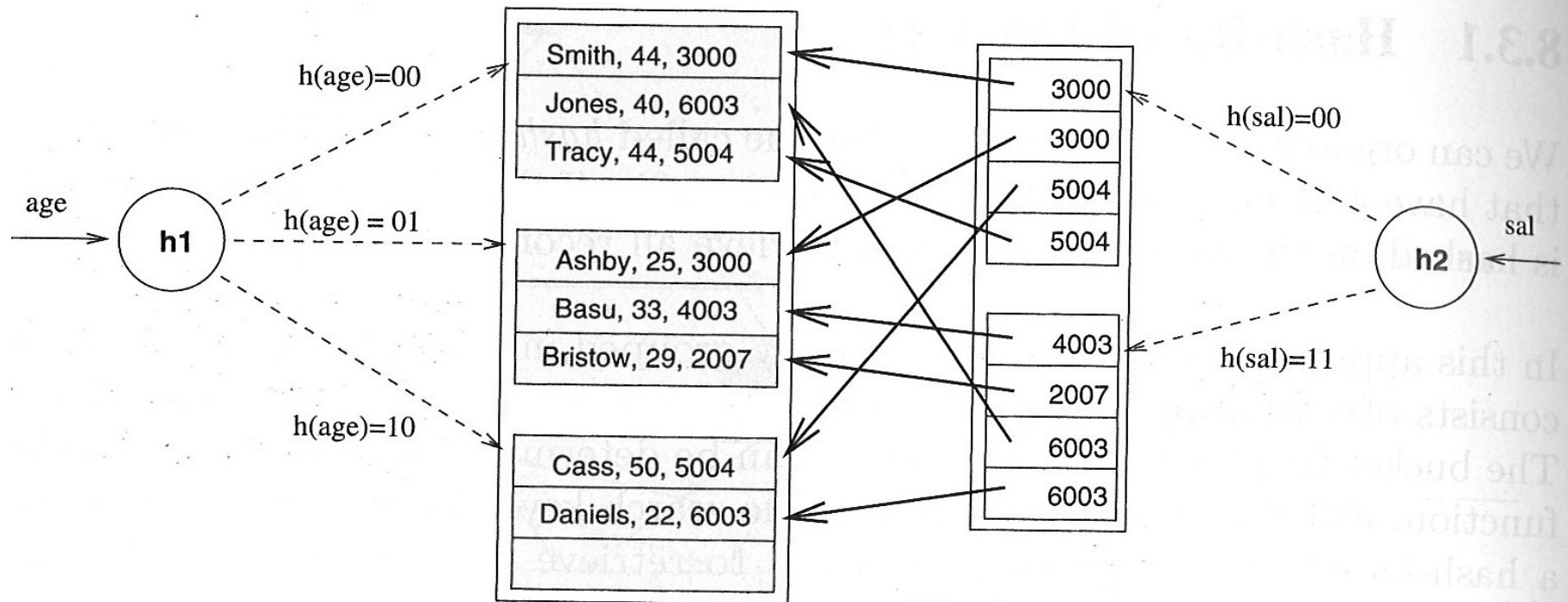
# Index Data Structures

- Hash table
  - Chapter 11
    - $\text{hash}(\text{ActorID}) = \text{PageID}$
  
- Trees
  - Chapter 10
  - B+ Trees

# Hashing

- What is the  $O()$  for the access time of a hash table?
- Example: Page 280, Figure 8.2

280



File of <sal, rid> pairs

Employees file hashed on age hashed on sal

Ramakrishnan, Gehrke, Database Management Systems, 3<sup>rd</sup> edition

Figure 8.2 Index-Organized File Hashed on *age*, with Auxiliary Index on *sal*

# Trees

- Let's review Binary Search Trees
  - fan-out?
  - $O()$  for finding a value in a BST?
  - Why?
  - What problems do BSTs have?

# B+ Tree

- B+ Tree
  - rebalancing tree!
    - all paths from the root to any leaf are the same length
  - B+ tree of order  $b$  has between  $(b/2)+1$  and  $b$  keys per node
    - except the root, between 2 and  $b$  keys
  - all data stored at the leaf nodes
    - (B trees can store data in any node)
- Example: page 281, Figure 8.3

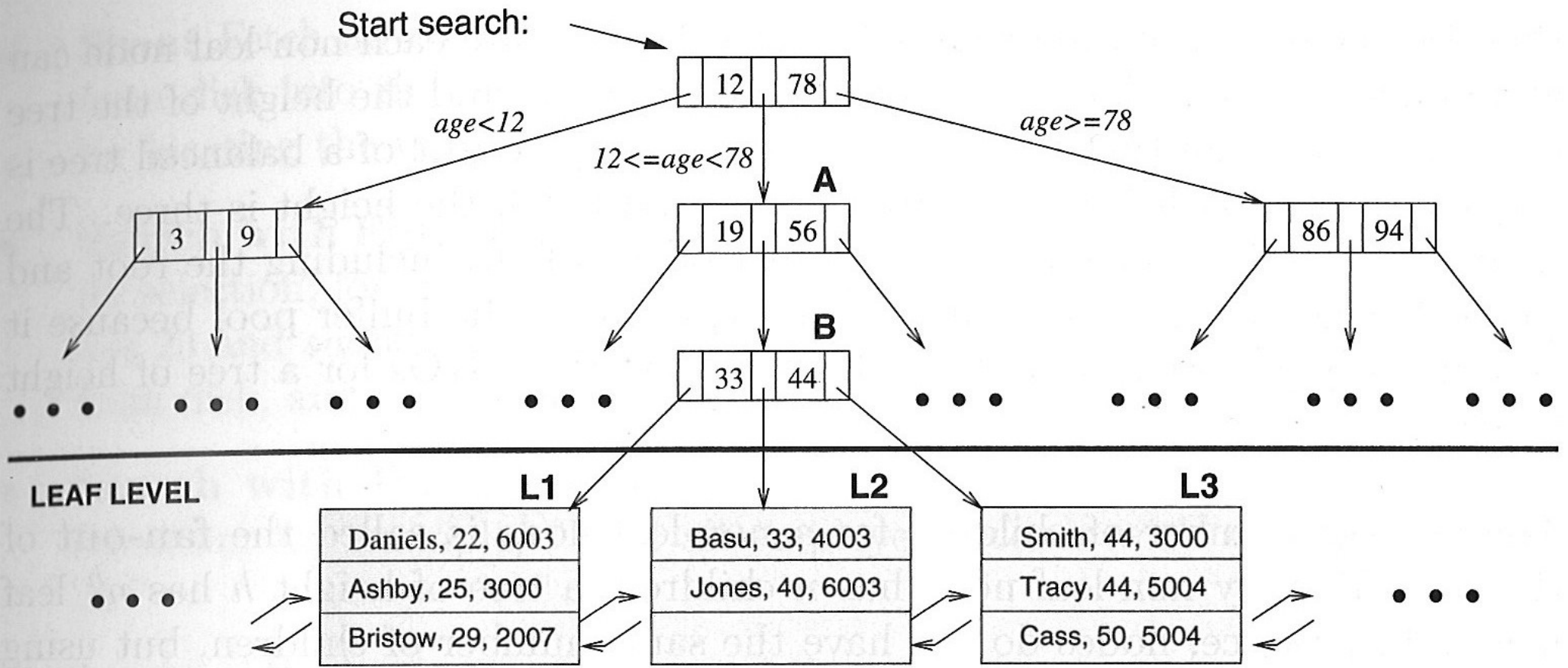


Figure 8.3 Tree-Structured Index

# B+ vs BST

- If we have 100,000,000 records
  - how long would it take to find a record with a BST?
  - with a B+ Tree with fan-out 100?
    - 100 is a typical fan-out for a B+ Tree in an index
  - Each step in the tree may be a disk read



# Comparison

- Cost Model
  - B: number of data pages
  - R: number of records per page
  - D: average time to read/write a page
  - C: average time to process a record
  - H: cost of the hash function
  - F: fan-out for trees

# Files to compare

- Heap file
- Sorted file
- Cluster B+ tree
- Heap file, B+ tree index
- Heap file, hash index

# Scan

- I/O costs

# Equality

# Range

# InnoDB

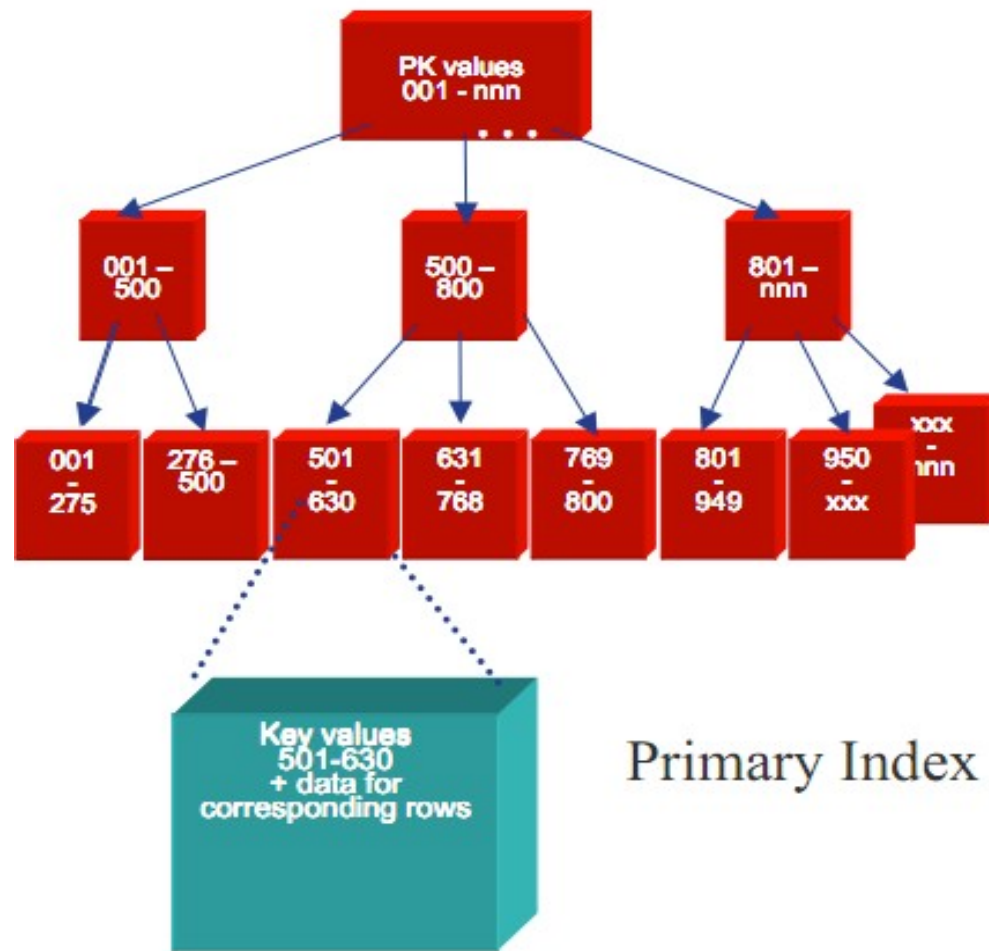
<https://dev.mysql.com/doc/refman/5.7/en/innodb-tables-indexes.html>

<https://dev.mysql.com/doc/refman/5.7/en/innodb-introduction.html>

<https://dev.mysql.com/doc/refman/5.7/en/innodb-indexes.html>

<https://dev.mysql.com/doc/refman/5.7/en/mysql-acid.html>

# InnoDB Indexes - Primary



- Data rows are stored in the B-tree leaf nodes of a clustered index
- B-tree is organized by primary key or non-null unique key of table, if defined; else, an internal column with 6-byte ROW\_ID is added.

<http://www.innodb.com/wp/wp-content/uploads/2009/05/innodb-file-formats-and-source-code-structure.pdf>

**INNOBASE**

<http://www.innodb.com/wp/wp-content/uploads/2007/04/innodb-overview-mysql-uc-2006-pdf.pdf>

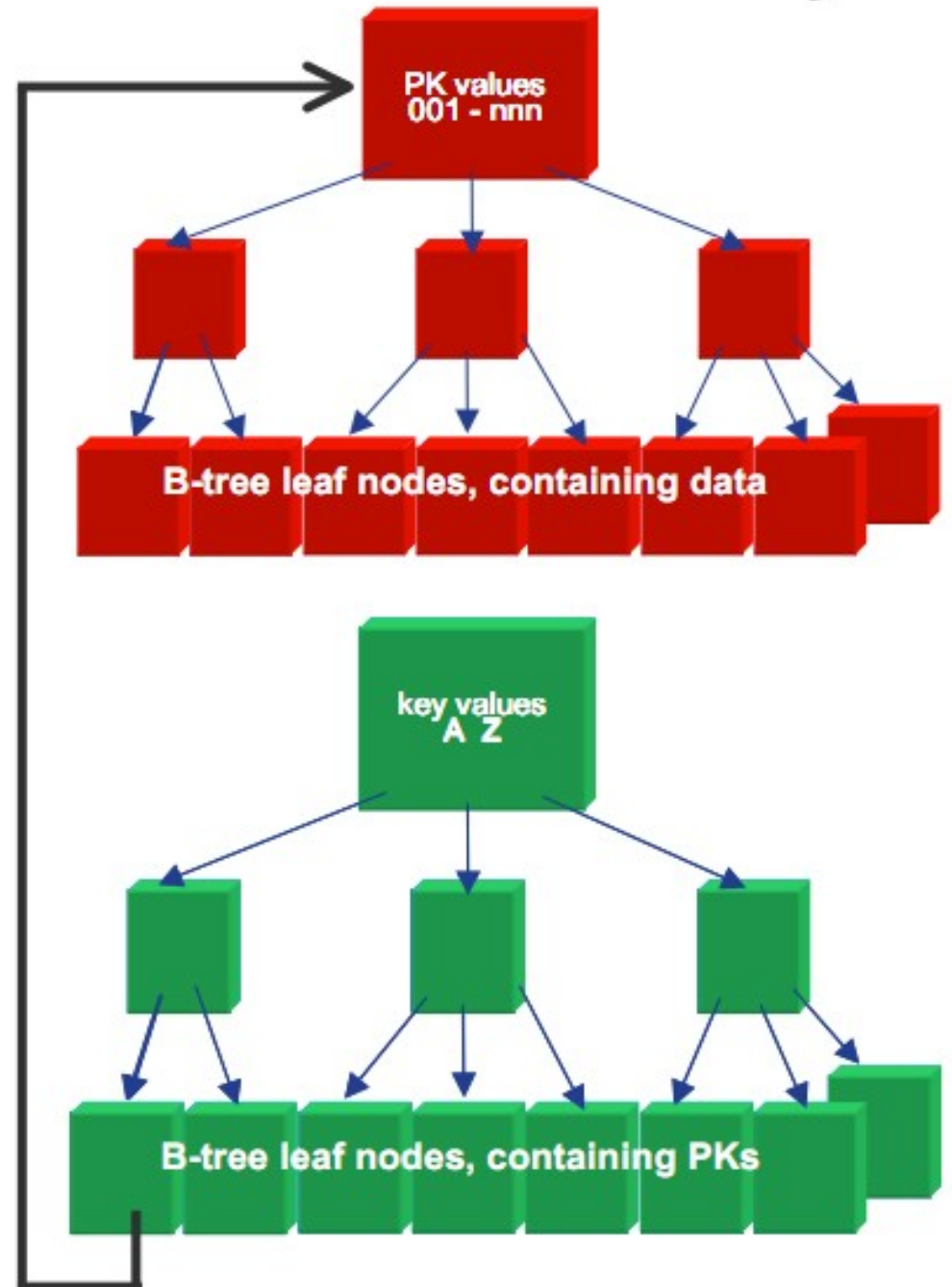
CS445

Pacific University

# InnoDB Indexes - Secondary

Secondary index B-tree leaf nodes contain, for each key value, the primary keys of the corresponding rows, used to access clustering index to obtain the data

Secondary Index





# Resources

- <http://en.oreilly.com/mysql2011/public/schedule/proceedings>
  - **A Beginner's Guide to MariaDB**
    - community version of MySQL
  - **InnoDB: Status, Architecture, and Latest Enhancements**
- <http://dev.mysql.com/doc/refman/5.7/en/innodb-index-types.html>
- <http://dev.mysql.com/doc/refman/5.7/en/innodb-introduction-features.html>