Chapter3

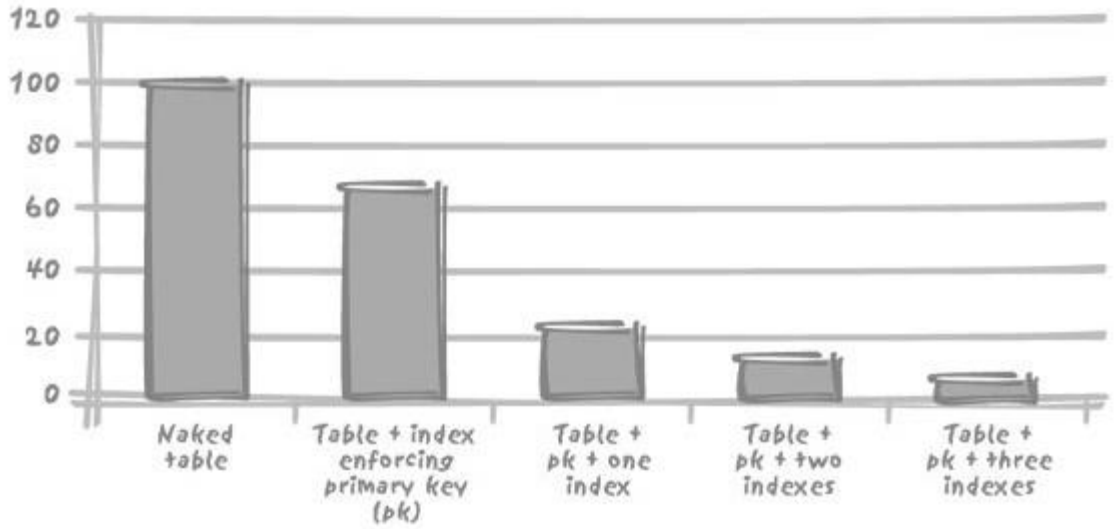
索引结构及使用

找到“切入点”

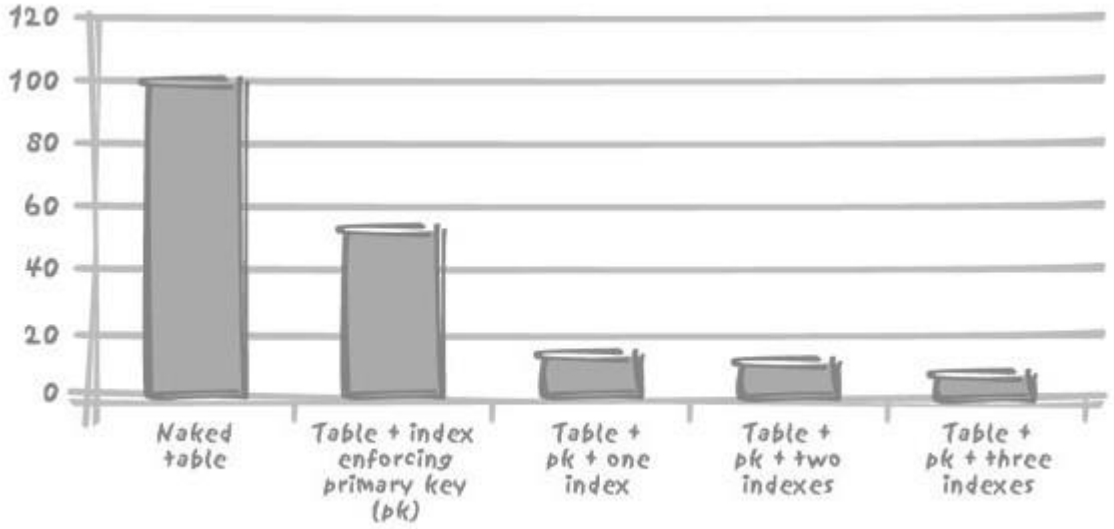
- 建立索引的基础
  - 检索条件、子集大小
- 索引不是万能的
  - 索引也有可能降低查询性能
- 索引的开销
  - 磁盘空间的开销、处理开销等

找到“切入点”（cont’）

- Oracle



- MySQL



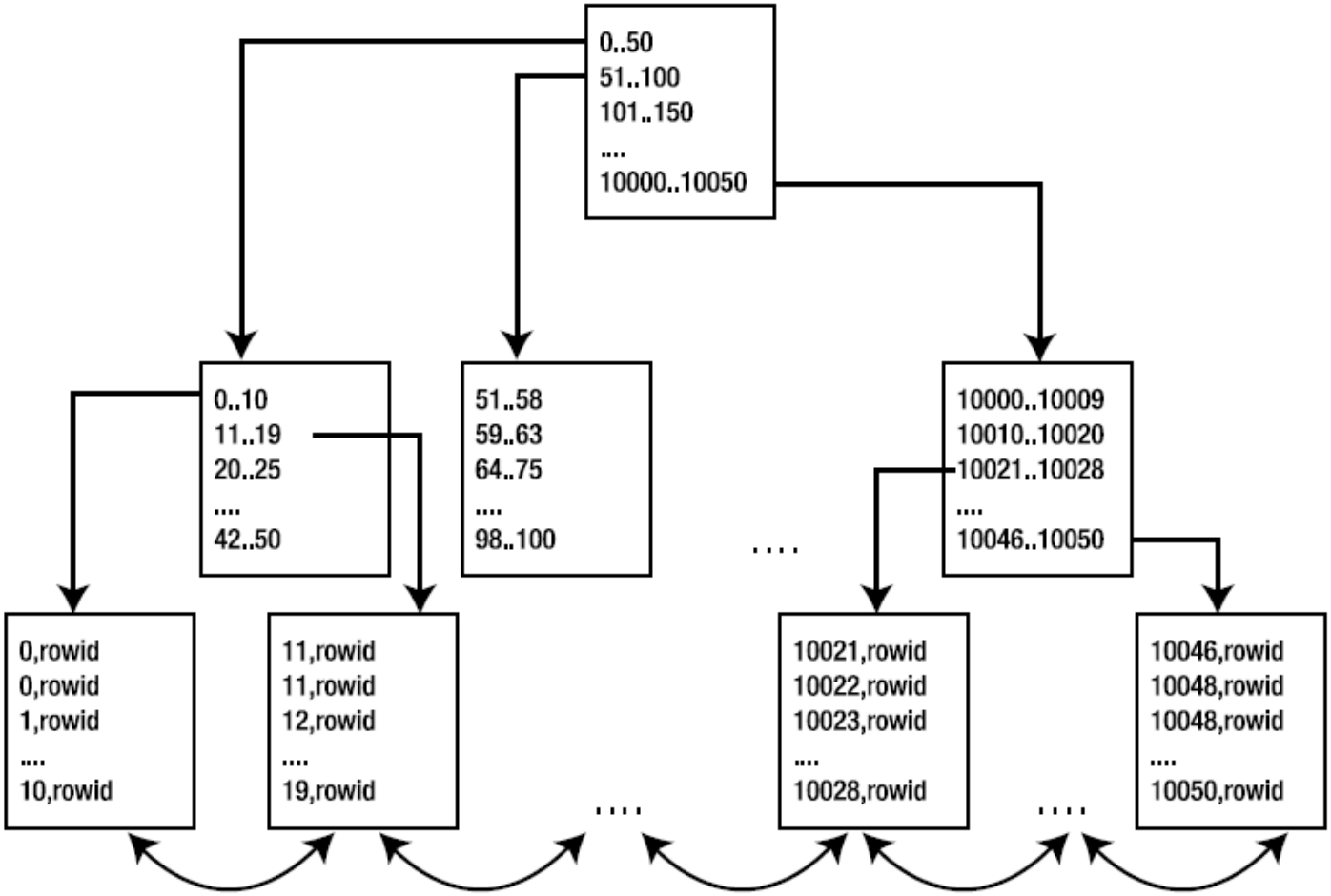
找到“切入点”（cont’）

- 索引始终是数据库中极重要的组成部分
  - 通用目的或事务处理型数据库系统
  - 决策支持系统
- 事务处理型数据库中 “太多索引≈设计不够稳定”

索引与目录

- 目录和索引是两种不同的机制
- 索引是一种以原子粒度访问数据的手段，而不是为了检索大量数据的

B-Tree（B+Tree）的结构



让索引发挥作用

- 索引的使用是否合理，首先取决于它是否有用。
- 长久以来，判断索引适用性的依据是检索比率（retrieval ratios）
- 什么时候应该使用B树索引
  - 仅当要通过索引访问表中很少一部分行
  - 如果要处理表中多行，而且可以使用索引而不用表

让索引发挥作用（cont’）

- 如何使用索引的因素错综复杂
  - 磁盘访问
  - 内存访问
  - 记录存储
  - ……

其他索引类型

- 哈希索引（Hash Index）：MySQL

- 位图索引（Bitmap Index）：Oracle
- 位图联结索引（Bitmap join index）:Oracle

- 函数索引（function-based index）
- 全文索引

函数和类型转换对索引的影响

- Where f (indexed_col) = ‘ some value‘
- 这种检索条件会使索引无法发挥作用
  - 日期函数
  - 隐式类型转换
- 基于函数的索引（function-based index）
  - Functional index
  - Index extension
  - Index on computed column

索引与外键

- 系统地对表的外键加上索引的做法非常普遍
  - *但是为什么呢?*
  - *有例外吗?*
- 建立索引必须有理由，无论是对外键，或是其他字段都是如此

同一字段，多个索引

- 如果系统为外键自动增加索引，常常会导致同一字段属于多个索引的情况
  - Orders—Order_Details—Articles
- 为每个外键建立索引，可能会导致多余索引

系统生成键

- 系统生产序列号，远好于
  - 寻找当前最大值并加1
  - 用一个专用表保存”下一个值“且加锁更新
- 但如果插入并发性过高，在主键索引的创建操作上会发生十分严重的资源竞争
- 解决方案
  - 反向键索引或叫逆向索引（reverse index）
  - 哈希索引（hash indexing）

为什么没有使用我的索引?

- 情况1：我们在使用B+树索引，而且谓词中没有使用索引的最前列
  - T，T(X,Y)上有索引，做SELECT * FROM T WHERE Y=5
- 跳跃式索引（仅CBO）

为什么没有使用我的索引？（cont’）

- 情况2：使用SELECT COUNT(*) FROM T，而且T上有索引，但是优化器仍然全表扫描
- 情况3：对于一个有索引的列作出函数查询
  - Select * from t where f(indexed_col) = value
- 情况4：隐形函数查询

为什么没有使用我的索引？（cont’）

- 情况5：此时如果用了索引，实际反而会更慢
- 情况6：没有正确的统计信息，造成CBO无法做出正确的选择
- 总结：归根到底，不使用索引的通常愿意就是“不能使用索引，使用索引会返回不正确的结果”，或者“不该使用索引，如果使用了索引就会变得更慢”

总结：索引访问的不同特点

- “查询使用了索引就万事大吉了”—误解啊～～
- 索引只是访问数据的一种方式
- “通过索引定位记录”只是查询工作的一部分
- 优化器有更多的选择权利
- 总结：索引不是万灵药。充分理解要处理的数据，做出合理的判断，才能获得高效方案

作业

- 作业1：文字描述多种索引各自的结构和各自适用的范围：B*Tree； Bitmap； Hash index；函数索引

- 实验1：B树索引的等值查询效率比较
- 实验2：B树索引对插入的影响
- 实验3：使用所用数据库特有的索引结构针对两种不同情况下和B树索引的效率对比

- 不是必须提交项目，只是对课程内容必要的练习
- 10月13日前提交TSS