Chapter 10

处理并发
Tackling Concurrency

索引的优点

- 一个有三个字段的表，前两个字段为整数（1-50000）第一个字段是FK，第二个字段没有索引。第三个名为label字段是字符型，长度30-50的随机字符串

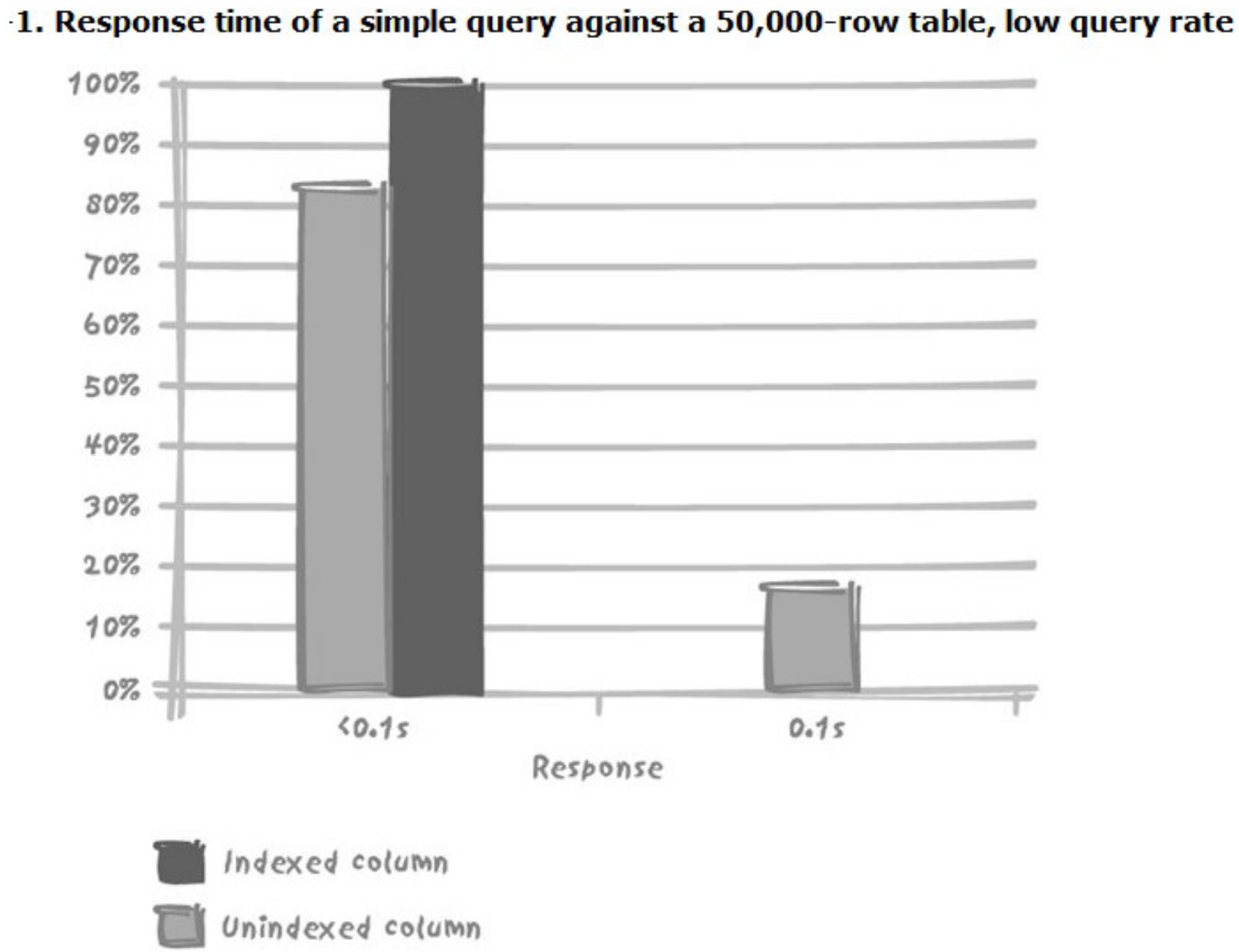
```
select label
from test_table
where indexed_column = random value

select label
from test_table
where unindexed_column = random value
```

- 响应时间不到一秒，仍然可能隐藏着重大的性能问题，不要相信单独某次测试。

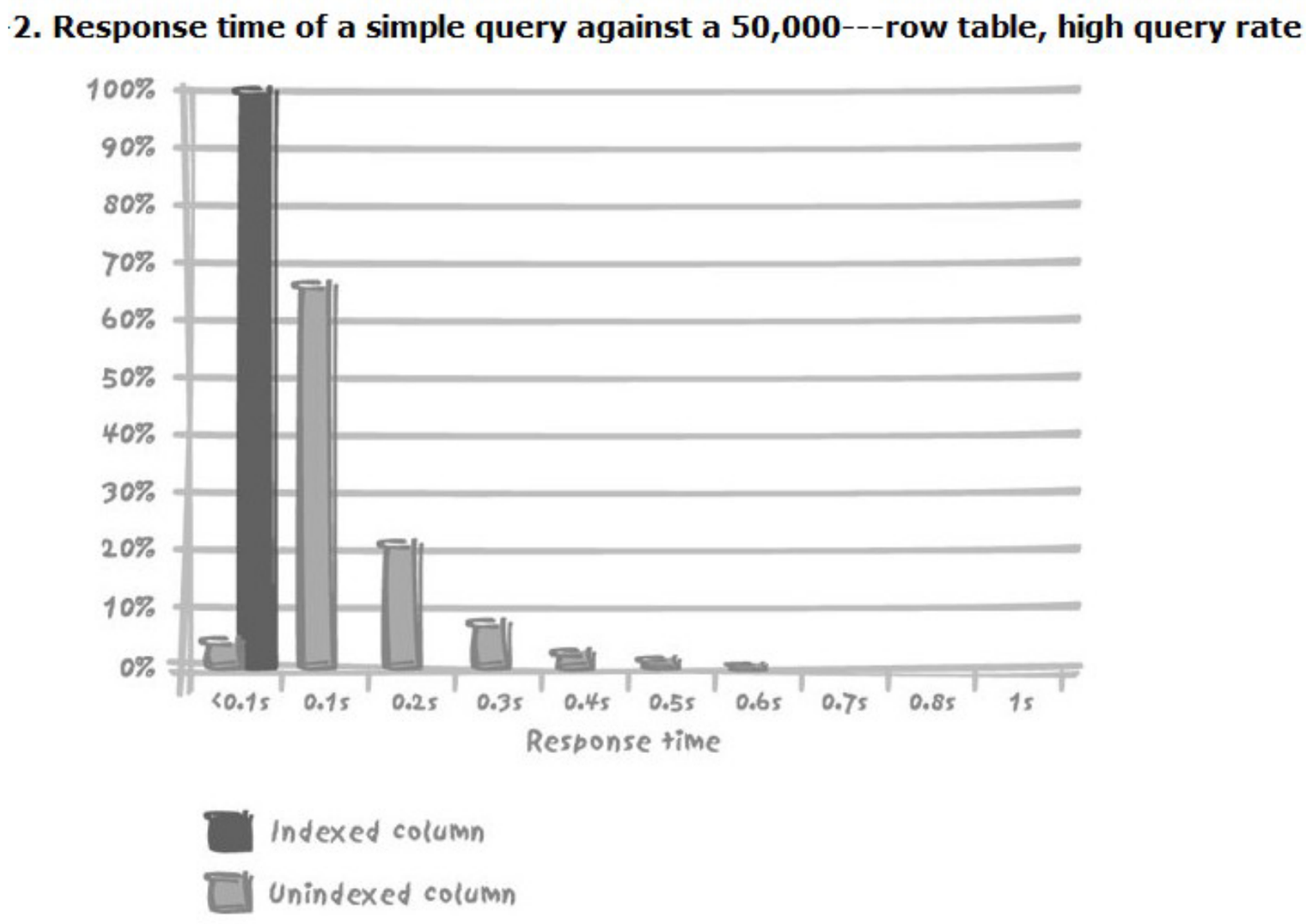
索引的优点

- 低频率查询（500次/分钟）



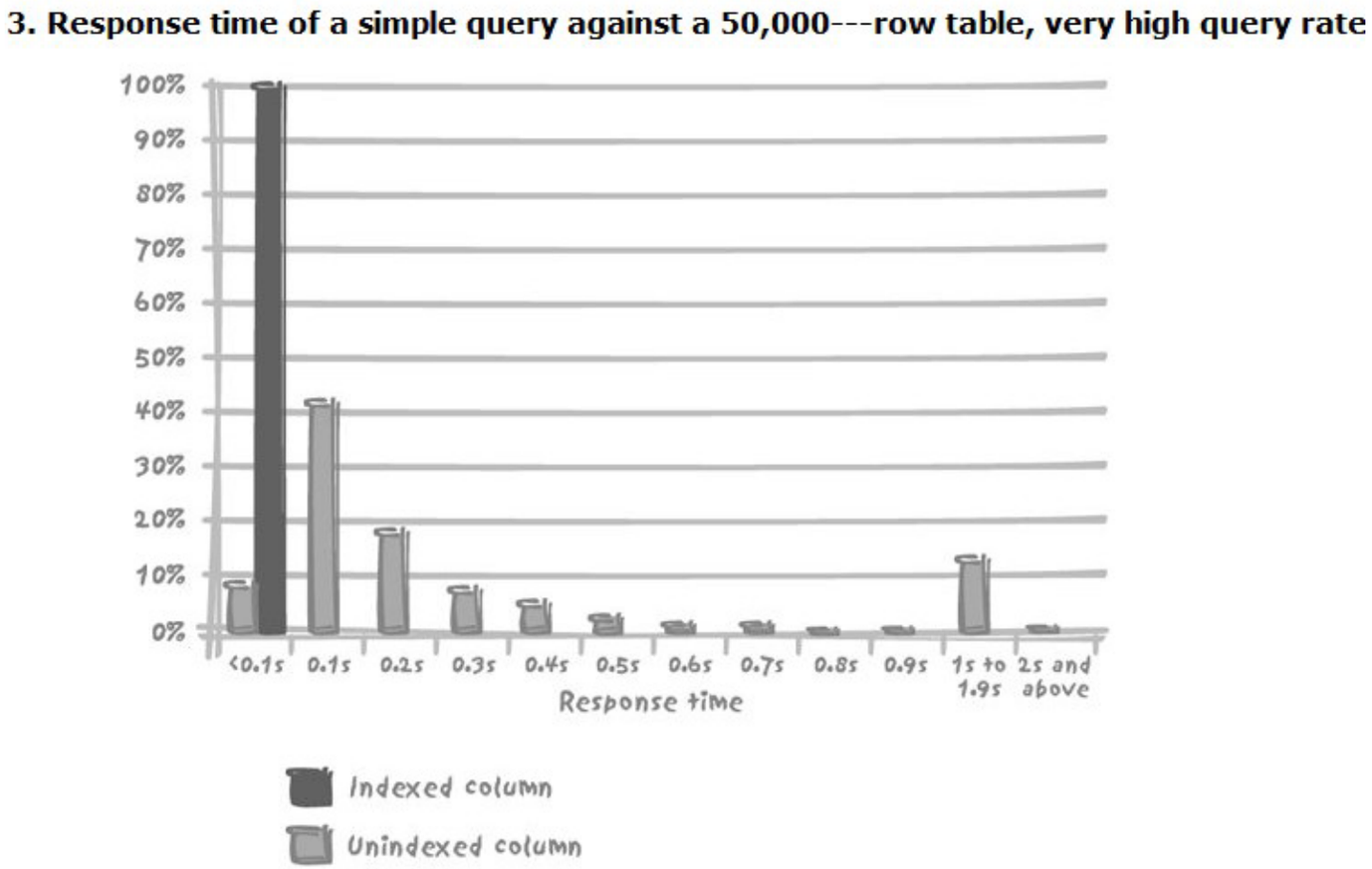
索引的优点

- 高频率查询（5000次/分钟）



索引的优点

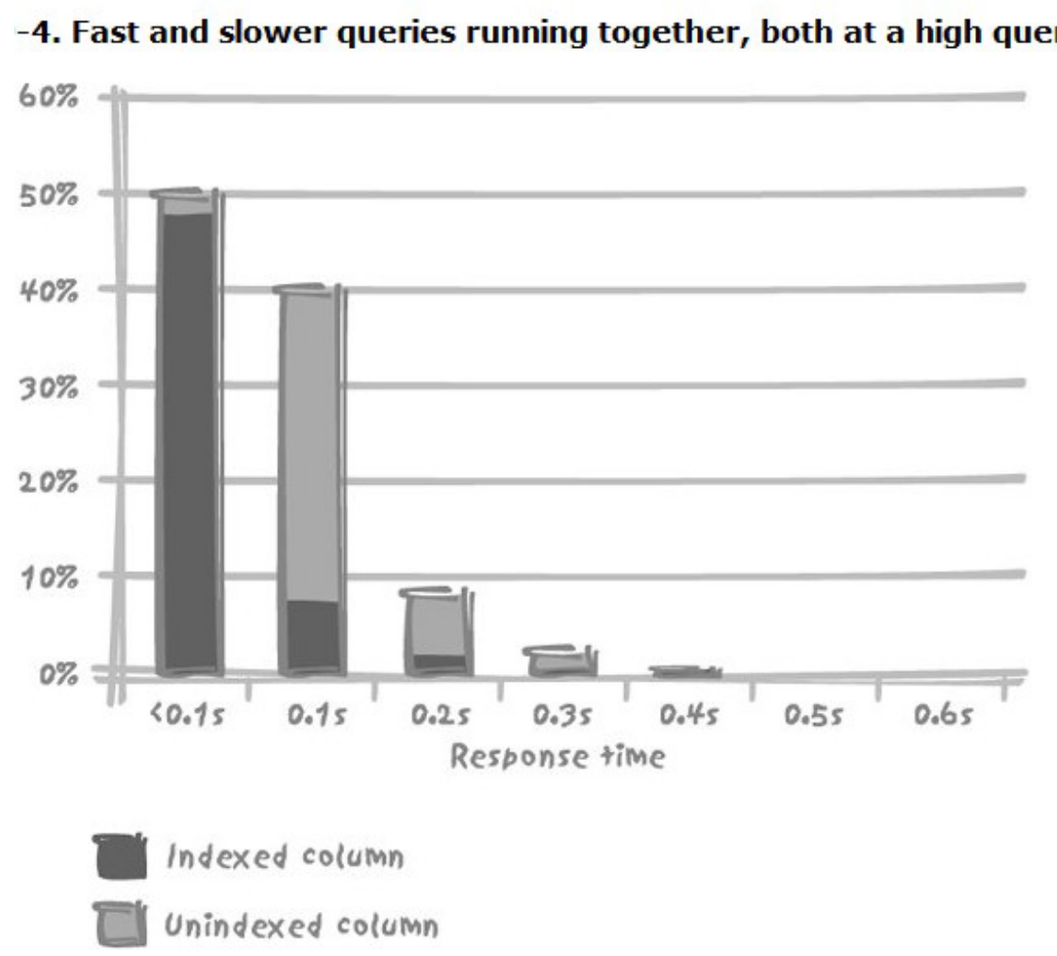
- 超高频率查询（10000次/分钟）



- 负载增加未必是造成性能问题的原因，它只不过使性能问题暴露出来了而已

排队

- 数据库引擎是否能快速服务
  - 数据库引擎性能（引擎、硬件、I/O系统效率...）
  - 数据服务的请求复杂度

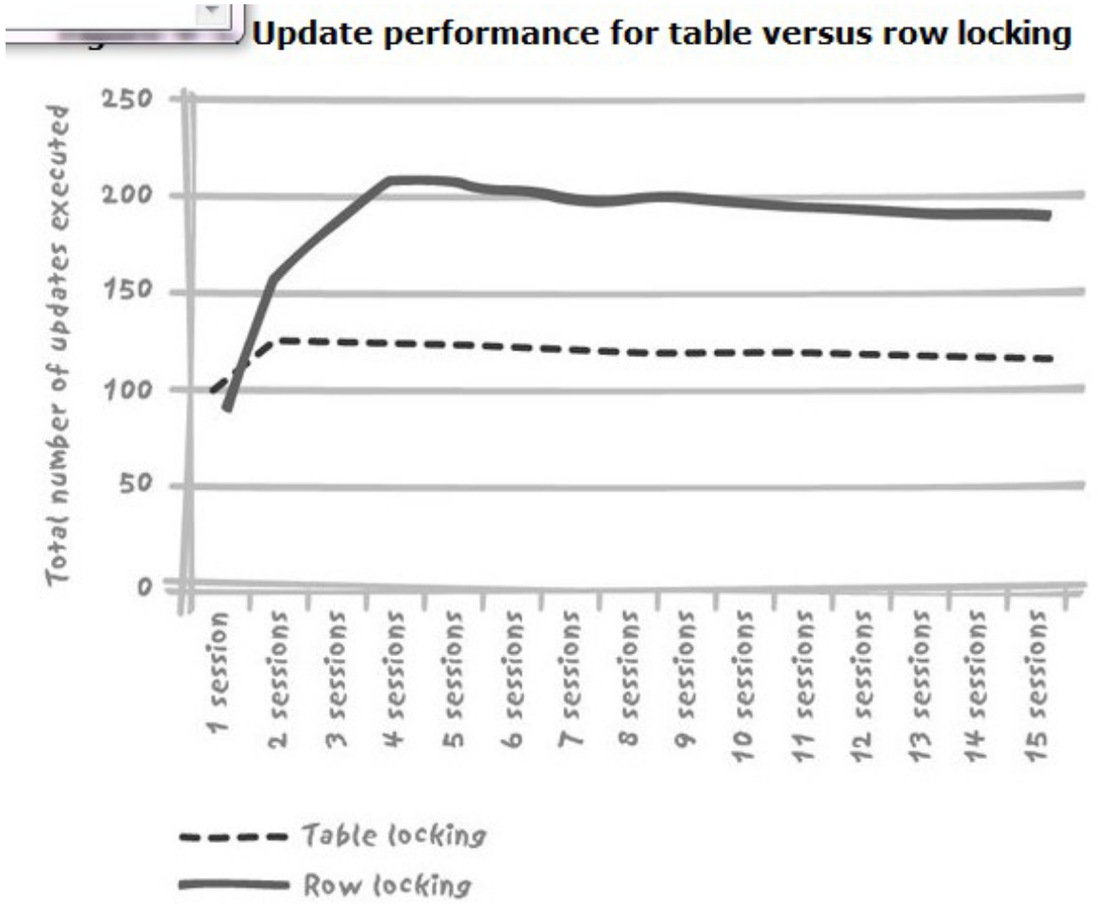


并发修改数据

- 修改数据操作越频繁，维持良好性能的难度就越大
- 加锁机制和资源争用会使得情况恶化

加锁

- 锁的粒度
  - 整个数据库、存储被修改的表的那部分物理单元、要修改的表、包含目标数据的的块和页、包含受影响数据的记录、记录中的字段

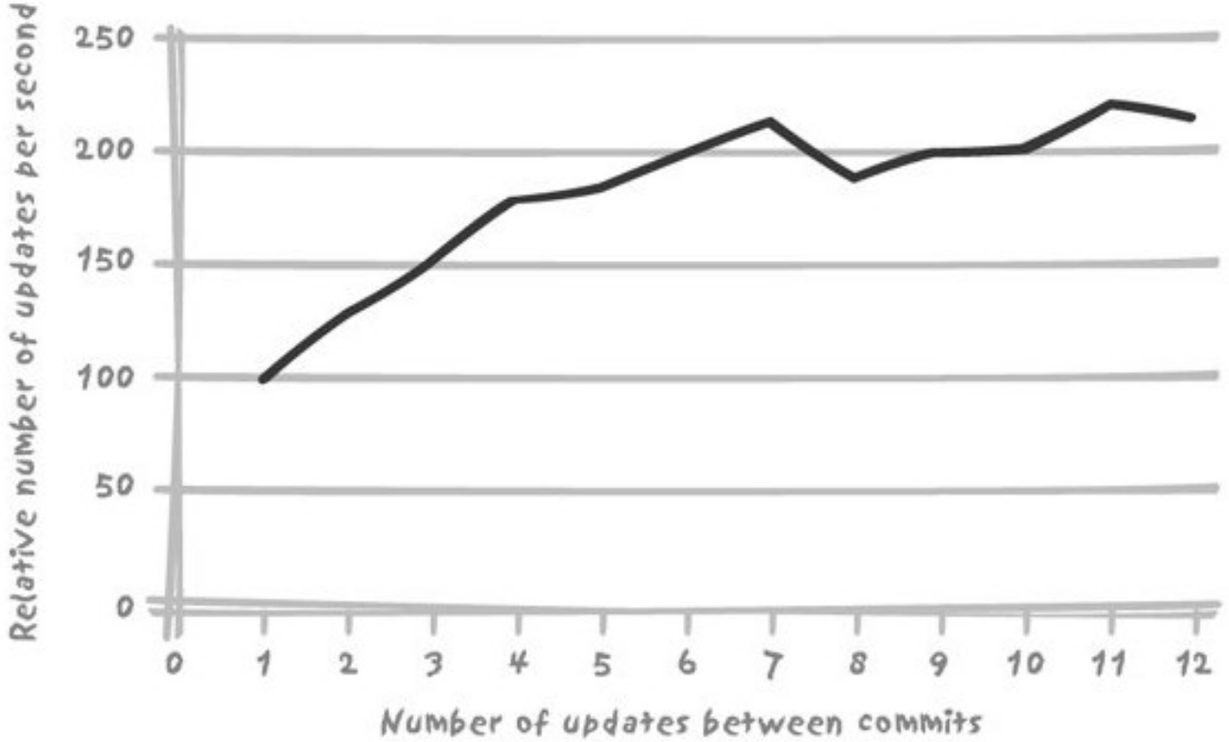


加锁处理

- 不要随便使用表级锁
- 尽量缩短加锁时间
  - Delete没有where，用truncate
- 索引也需要维护
- 语句性能高，未必程序性能高
  - 尽可能避免SQL语句上的循环处理
  - 尽量减少程序和数据库之间的交互次数
  - 充分利用DBMS提供的机制，使跨机器交互的次数降至最少
  - 把所有不重要不必须的SQL语句放在逻辑工作单元之外

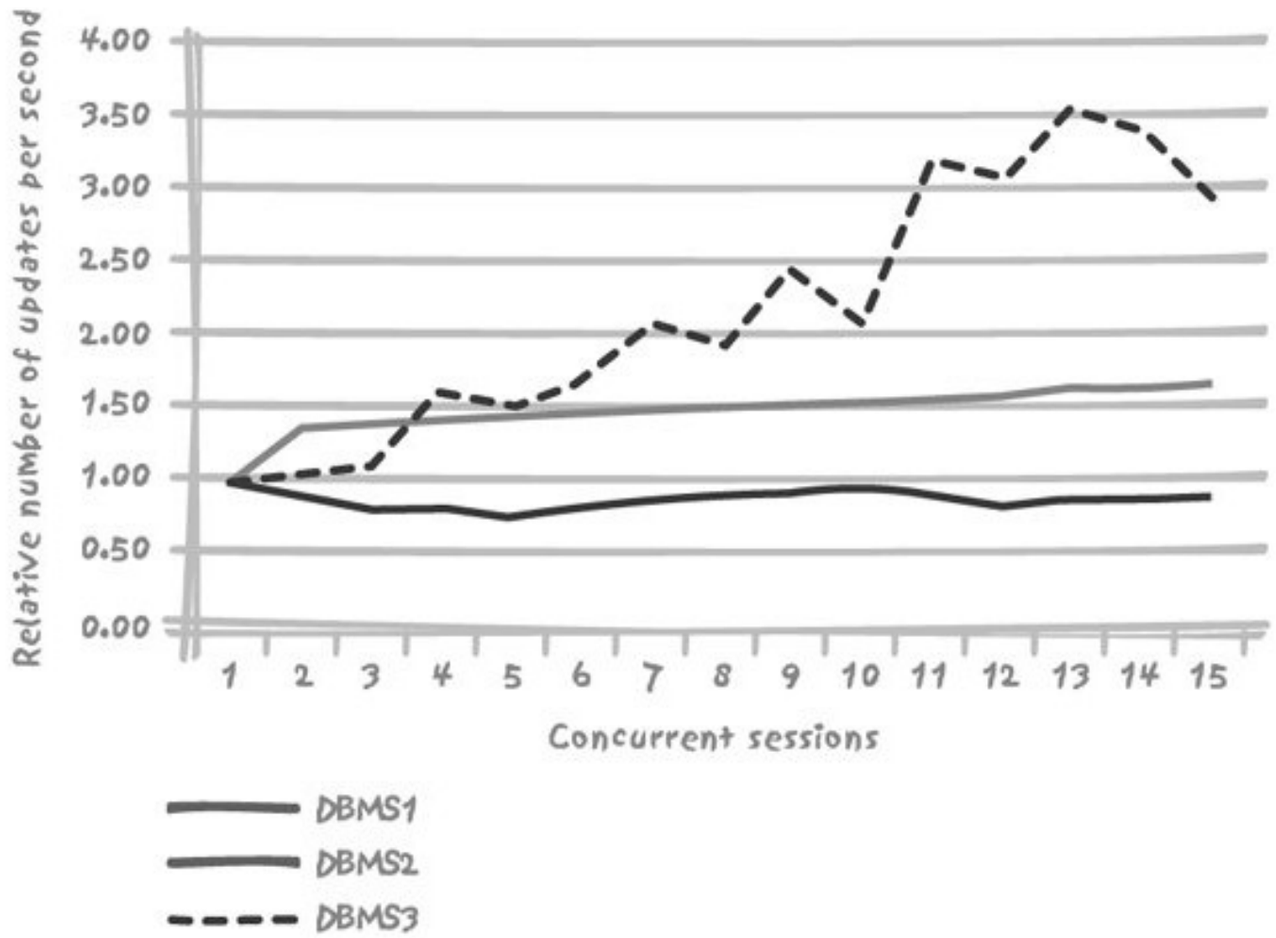
加锁与提交

- 想要使加锁时间最短，必须频繁的提交
- 但如果每个逻辑单元完成后都提交会增加大量开销



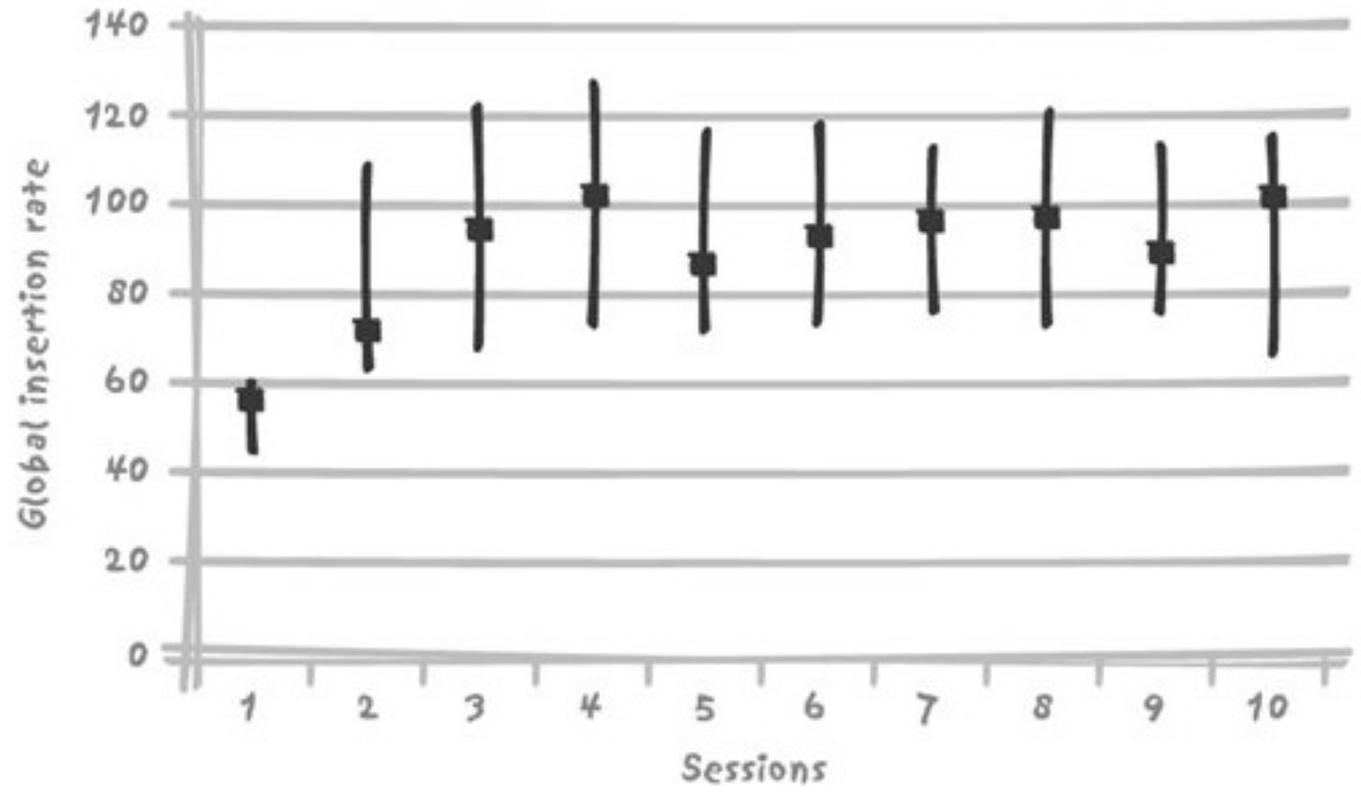
加锁与可伸缩性

- 与表级锁相比，行级锁能产生更佳的吞吐量
  - 行级锁大都性能曲线很快达到极限（不同的数据库产品不同）



资源竞争

- 插入与竞争
  - Table 是有14个字段、两个唯一性索引的表
  - 主键为系统产生的编号，而真正的键是由短字符串和日期值组成的复合键，必须满足唯一性约束

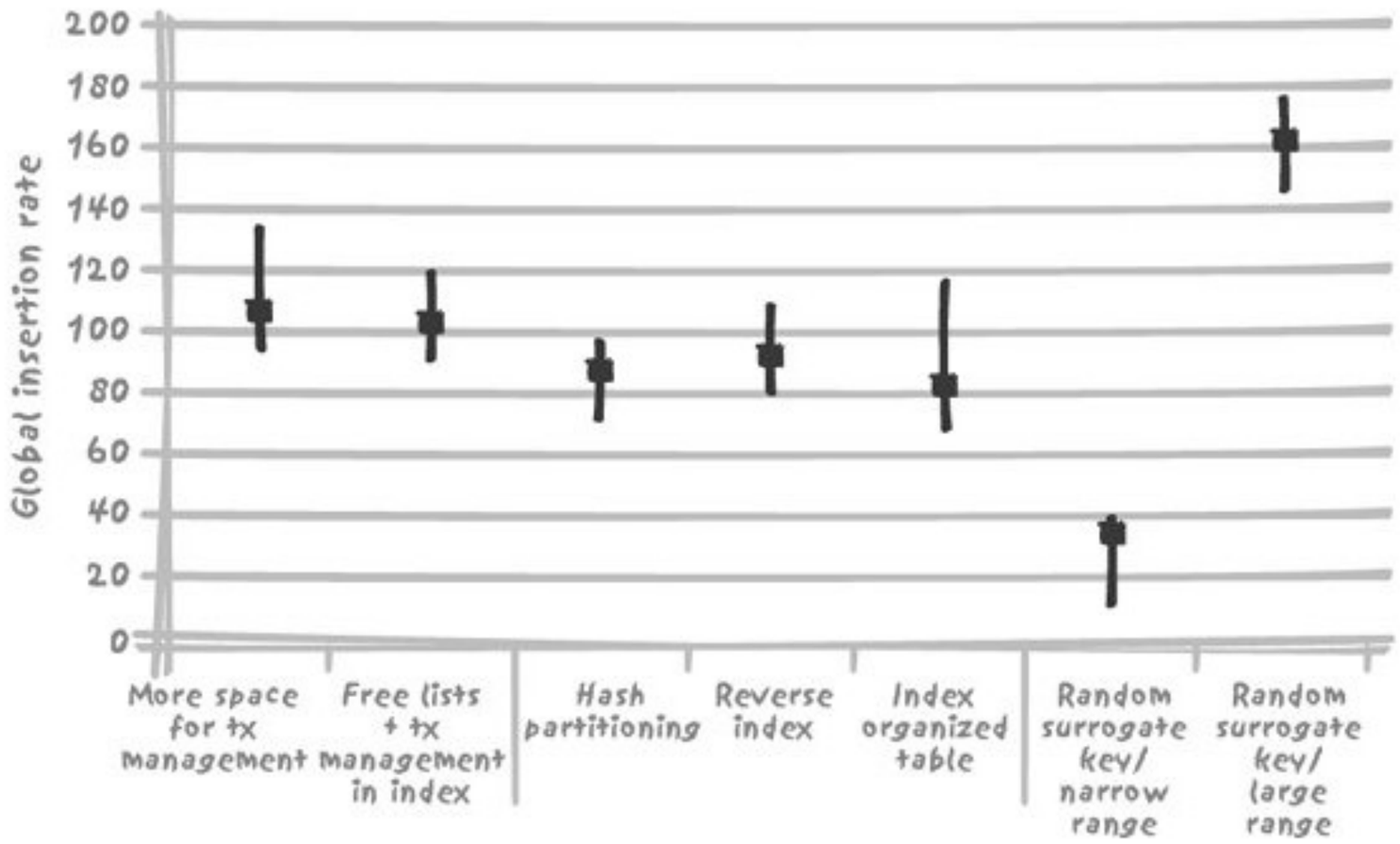


资源竞争

- DBA解决方案
  - 事务空间（Transaction space）
  - 可用列表（Free list）
- 架构解决方案
  - 分区（Partitioning）
  - 逆序索引（Reverse index）
  - 索引组织表（Index organized table）
- 开发解决方案
  - 调节并发数
  - 不适用系统产生值

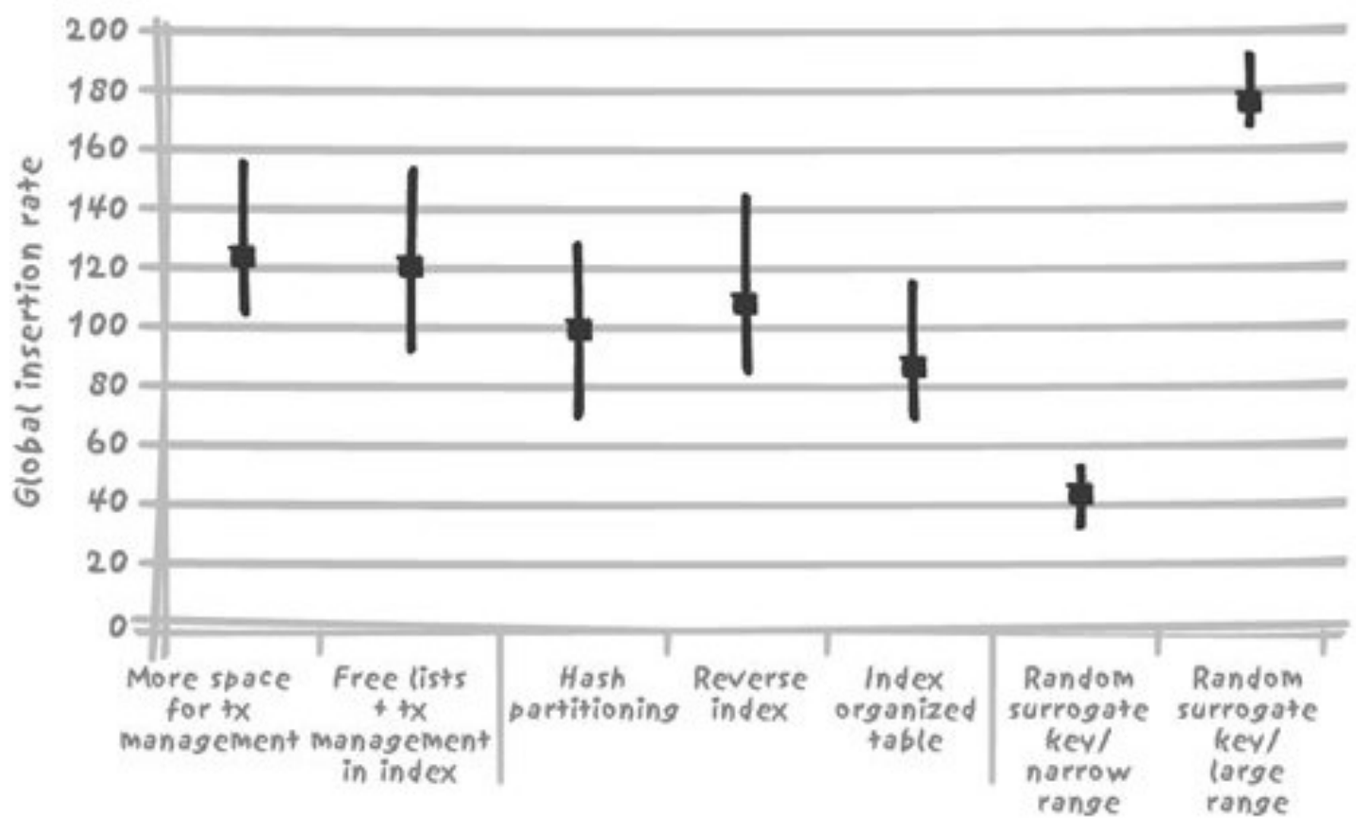
资源竞争

- 限制insert操作之间竞争的技术



资源竞争

- Session数较少时竞争限制技术的表现



资源竞争

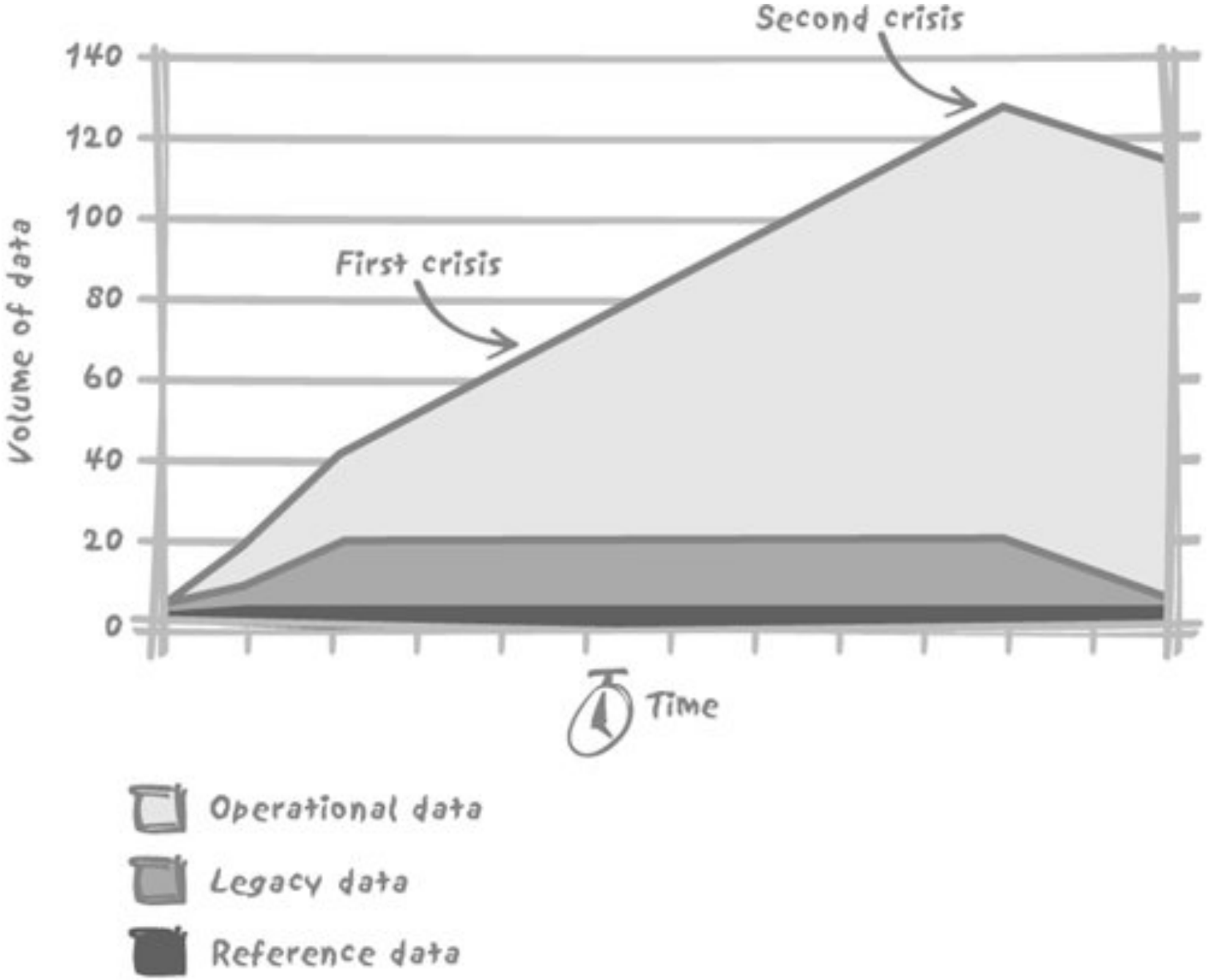
- 上述案例的瓶颈是主键索引
- Session的差异说明，有些技术需要已处于饱和状态的CPU提供资源，所以不能带来性能上的改善
- 上述案例避免竞争的方法是避免使用顺序产生的代理键……
- 总结：与加锁不同，数据库竞争是可以改善的。架构师、开发者和DBA都可以从各自的角度改善竞争。

Chapter 11

应付大数据量

Coping with Large Volumes of Data

增长的数据量


Second crisis
First crisis
140
120
100
80
60
40
20
0
Volume of data
Time
Operational data
Legacy data
Reference data

操作对数据量增加的敏感程度

- 受数据量的增加，影响不大
- 受数据量的增加，线性影响
- 受数据量的增加，非线性影响

影响不大

- 主键检索等值单一查询

```
SQL> declare
 2  n_id          number;
 3 cursor c is select customer_id
 4 from orders
 5 where order_id between 10000 and 20000;
 6 begin
 7 open c;
 8 loop
 9 fetch c into n_id;
10 exit when c%notfound;
11 end loop;
12 close c;
13 end;
14 /
```

```
SQL> declare
 2  n_id      number;
 3 begin
 4 for i in 10000 .. 20000
 5 loop
 6 select customer_id
 7 into n_id
 8 from orders
 9 where order_id = i;
10  end loop;
11 end;
12 /
```

- 第一个，用cursor，进行了显性范围扫描（explicit range scan）速度比迭代处理的单笔交易快两倍

- 第一个例子只要向下搜索索引。而第二个，每次搜索order_id字段时都要向下访问B树。

线性影响

- 返回记录数量和查询毫无关系
- SQL操作的数据和最后返回的结果无关（聚合函数）
- 可选的唯一方法：引入其他条件（例如时间范围）
  - 设定上限
  - 不是单纯的技术问题
  - 还依赖于业务需求

非线性影响

- 排序性能影响非线性

- 排序性能减低间歇性
  - 因为较小型的排序全部在内存中执行，而较大型的排序（涉及多个有序子集的合并）则需要将有序子集临时存储到硬盘中。所以通过调整分配给排序的内存数量来改善排序密集型操作的性能是常见且有效地调优技巧。

```
order_id          bigint(20) (primary key )
customer_id       bigint(20)
order_date        datetime
order_shipping    char(1)
order_comment     varchar(50)
```

The queries are first a simple primary key-based search:

```
select order_date
from orders
where order_id = ?
```

then a simple sort:

```
select customer_id
from orders
order by order_date
```

then a grouping:

```
select customer_id, count(*)
from orders
group by customer_id
having count(*) > 3
```

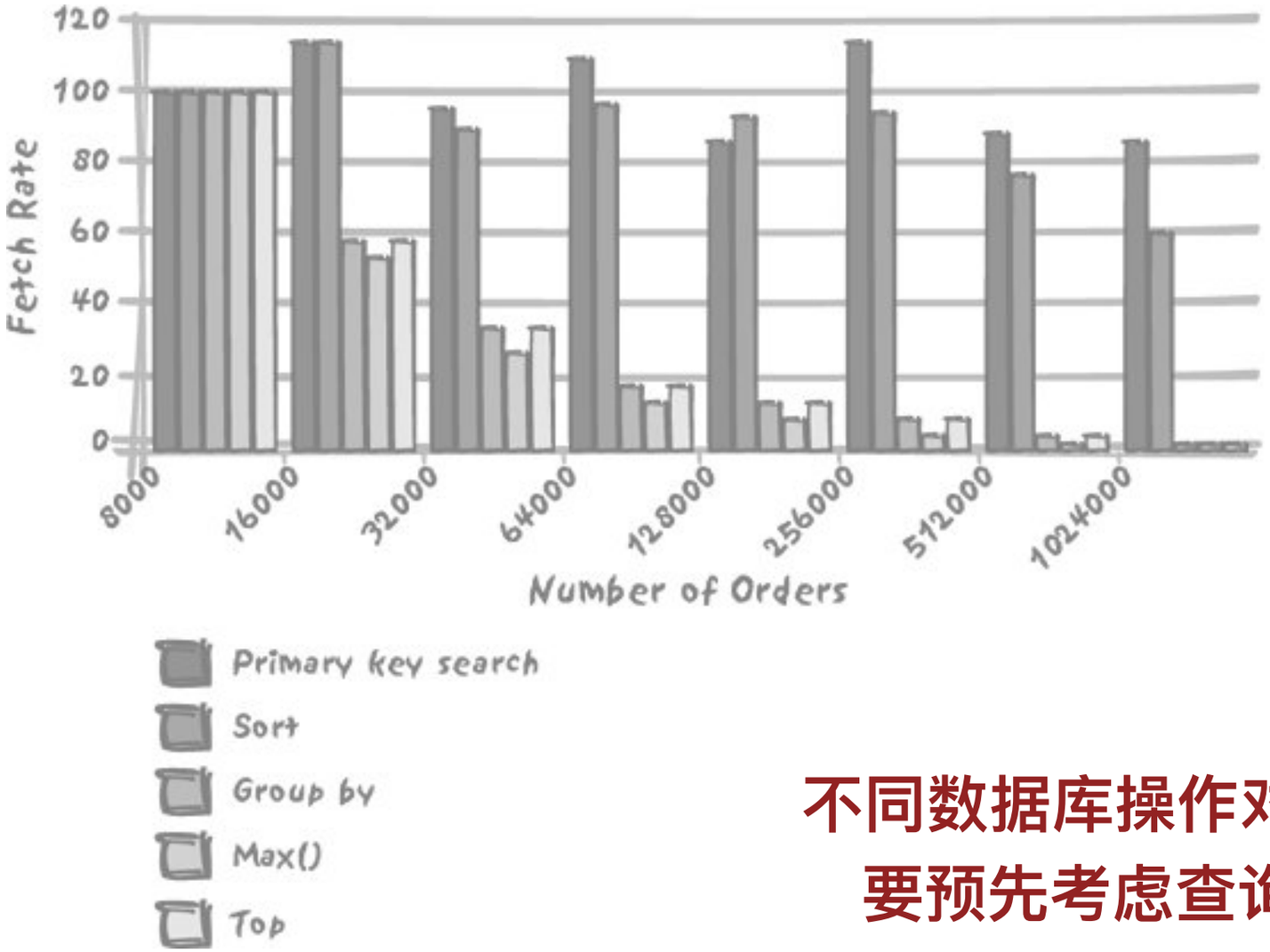
then the selection of the maximum value in a nonindexed column:

```
select max(order_date)
from orders
```

and finally, the selection of the "top 5" customers by number of orders:

```
select customer_id
from (select customer_id, count(*)
      from orders
      group by customer_id
      order by 2 desc) as sorted_customers
limit 5
```

非线性影响



记录数大概从8000-1000000之间
不同的cid大概3000个

不同数据库操作对数据量增加的敏感程度不同。
要预先考虑查询对不同数据量的执行方式。

综合的考量

- 数据量增加对性能的预估
  - 隐藏在查询背后对数据量的高敏感性
  - 比如max（）对高数据量的敏感，而直接引起子查询性能缓慢降低，必须使用非关联子查询。
- 排序的影响
  - 字节数量而不是记录数量
  - 也就是被排序的总数据量
  - Join应该延后到查询的最后阶段

Join延迟到查询的最后阶段

- 查询一年内的10大客户的名称和地址

```
select *
from (select c.customer_name,
             c.customer_address,
             c.customer_postal_code,
             c.customer_state,
             c.customer_country
             sum(d.amount)
      from customers c,
           orders_o,
           order_detail d
      where c.customer_id = o.customer_id
        and o.order_date >= some date expression
        and o.order_id = d.order_id
      group by c.customer_name,
               c.customer_address,
               c.customer_postal_code,
               c.customer_state,
               c.customer_country
       order by 6 desc) as A
limit 10
```

Join延迟到查询的最后阶段

- 为了避免连接修改内层子查询产生的记录的顺序

```
select c.customer_name,
       c.customer_address,
       c.customer_postal_code,
       c.customer_state,
       c.customer_country
       b.amount
from (select a.customer_id,
             a.amount
      from (select o.customer_id,
                   sum(d.amount) as amount
            from orders_o,
                 order_detail d
            where o.order_date >= some date expression
              and o.order_id = d.order_id
            group by o.customer_id
            order by 2 desc) as a
      limit 10) as b,
      customers c
where c.customer_id = b.customer_id
order by b.amount desc
```

还有几个需要注意的问题

- 谨慎使用关联嵌套子查询
- 通过分区提升性能是有瓶颈的
- 数据清除的风险（truncate）
- 关注不同的数据操作（insert、update、delete）的整体成本的高低

数据仓库

- Ralph Kimball 《The Data Warehouse Toolkit》
- Bill Inmon 《Building the Data Warehouse》
- 《数据仓库与知识发现》课程
- 数据仓库一定是非规范化的
- 操作型数据的存储和决策支持系统

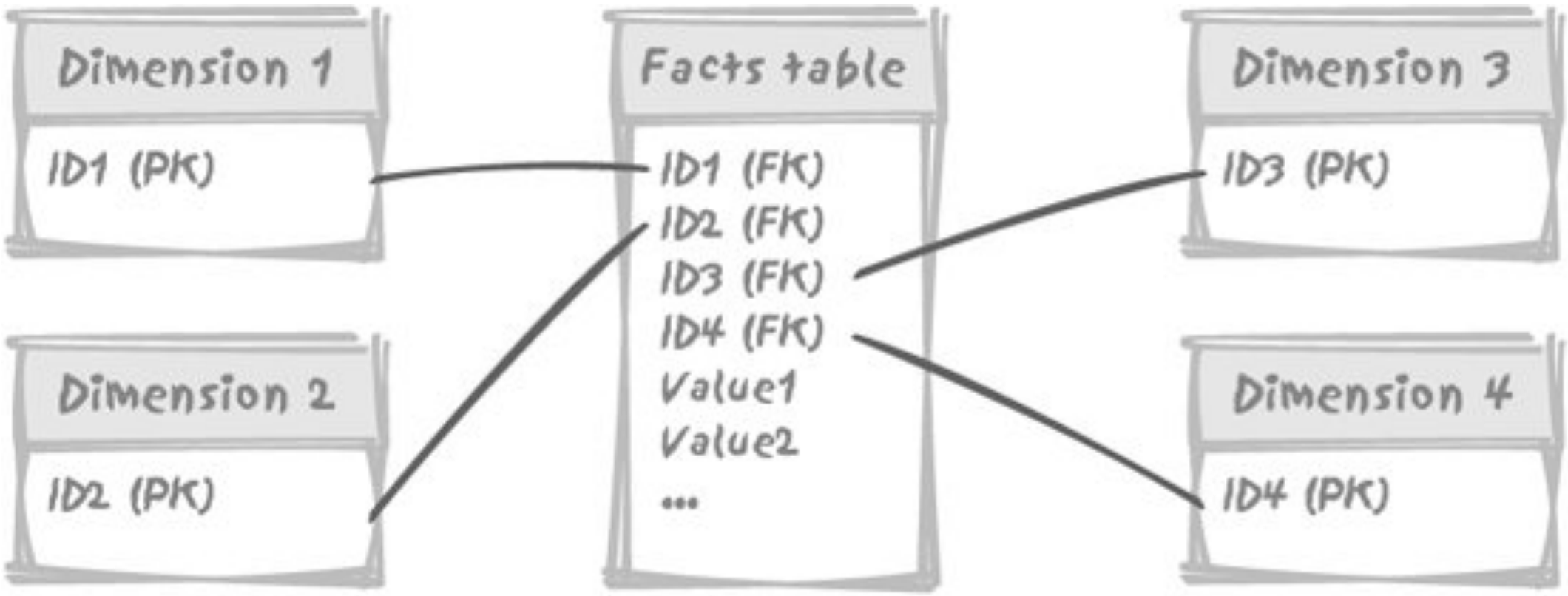
事实表与维度表：星型Schema

- 维度模型的原则是保存度量值（measurement value）
- 无论它们是数值量、统计量或任何其他需要放入大型事实表（fact table）中的信息
- 参考数据保存在维度表（dimension table）

date_key	date_value	date_description
12345	01/01/1970	January 1, 1970

day	month	year	quarter	holiday
Thursday	January	1970	Q1 1970	Holiday

事实表与维度表：星型Schema



查询工具

```
...
FROM (SELECT ((((((((((t2."FOREIGN_CURRENCY"
                        || CASE
                             WHEN 'tfp' = 'div' THEN t2."CODDIV"
                             WHEN 'tfp' = 'ac' THEN t2."CODACT"
                             WHEN 'tfp' = 'gsd' THEN t2."GSD_MNE"
                             WHEN 'tfp' = 'tfp' THEN t2."TFP_MNE"
                             ELSE NULL
                           END
                           )
                        || CASE
                             WHEN 'Y' = 'Y' THEN TO_CHAR (
                                    TRUNC (
                                           t2."ACC_PCI"
                                          )
                                    )
                             ELSE NULL
                           END
                          )
                   || CASE
                        WHEN 'N' = 'Y' THEN t2."ACC_E2K"
                        ELSE NULL
                      END
                     )
                  || CASE
                       WHEN 'N' = 'Y' THEN t2."ACC_EXT"
                       ELSE NULL
                     END
                    )
                 || CASE ...
```

数据抽取、数据转换、数据加载

- 数据抽取：一般不使用SQL语句，而是专用工具。

- 数据转换
  - SQL技术
  - 数据来源
  - 对生产环境的影响程度
  - 转换程度的大小

- 数据加载
  - 以系统生成KEY来跟踪各种逻辑上相同，技术上不同的数据项
  - 完整性约束与索引

查询维度表与事实表：即席报表

- 迷你维度（mini-dimension）
- 外部触发器（outrigger）
- 桥接表（bridge table）