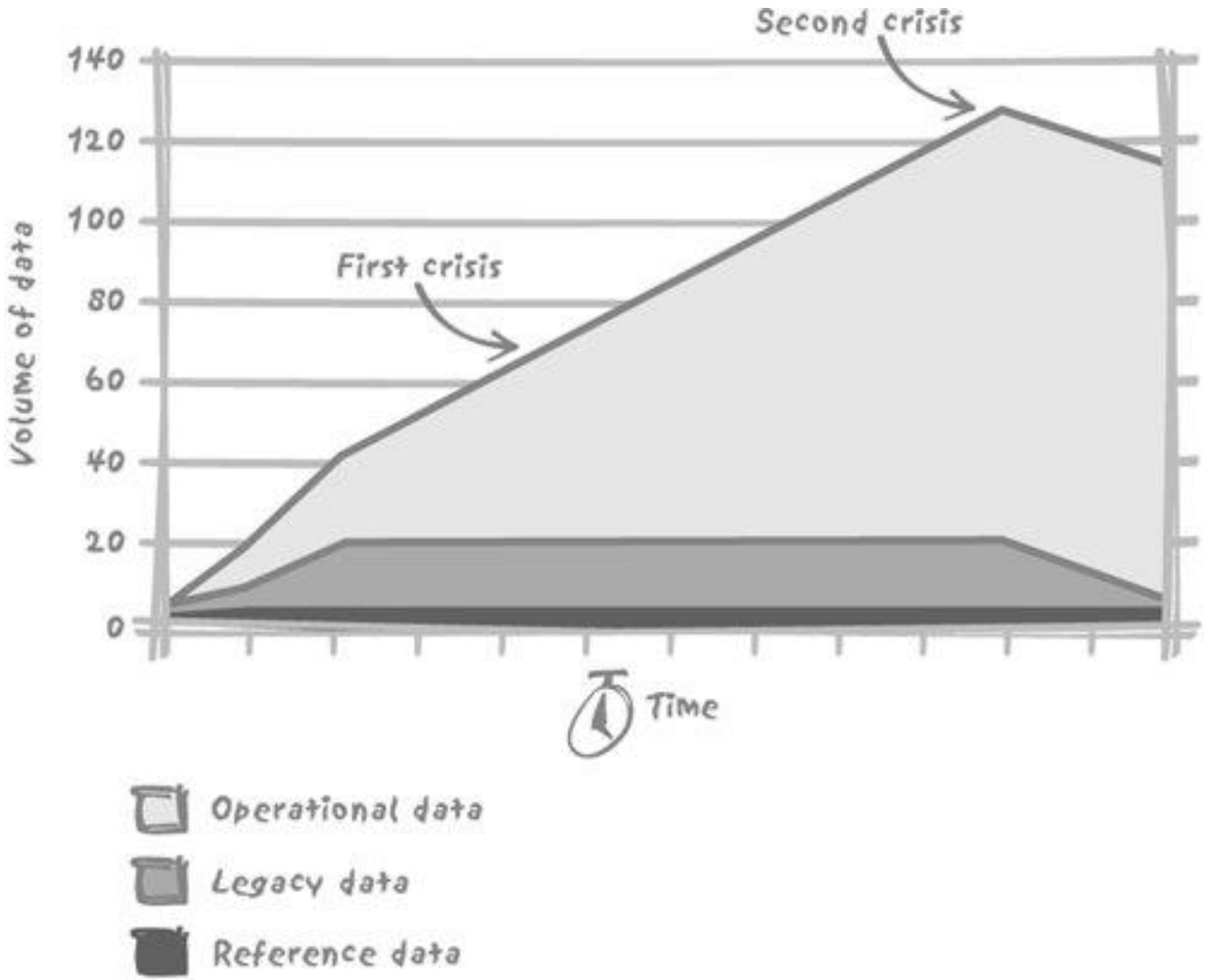
# Chapter 8-part 2

应付大数据量

Coping with Large Volumes of Data

# 增长的数据量


Second crisis
140
120
100
First crisis
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 /
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.27
```

```
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 /
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.63
```

# 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。



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

**1) primary key-based search:**

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

**2) sort:**

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

**3) grouping:**

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

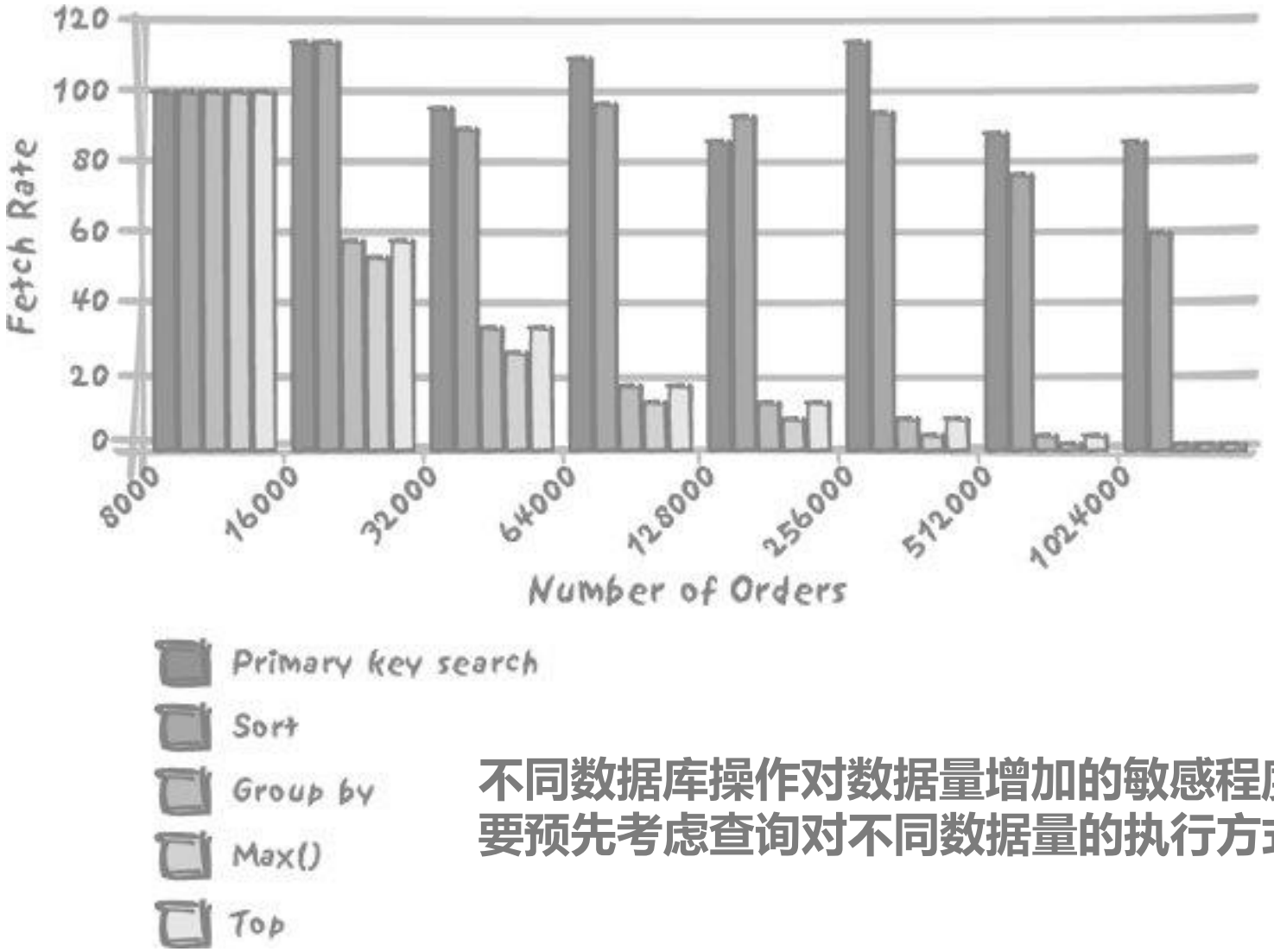
**4) maximum value in a nonindexed column:**

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

**5) "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
```

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。**

# End

那我们还有哪些办法处理数据量增加呢?