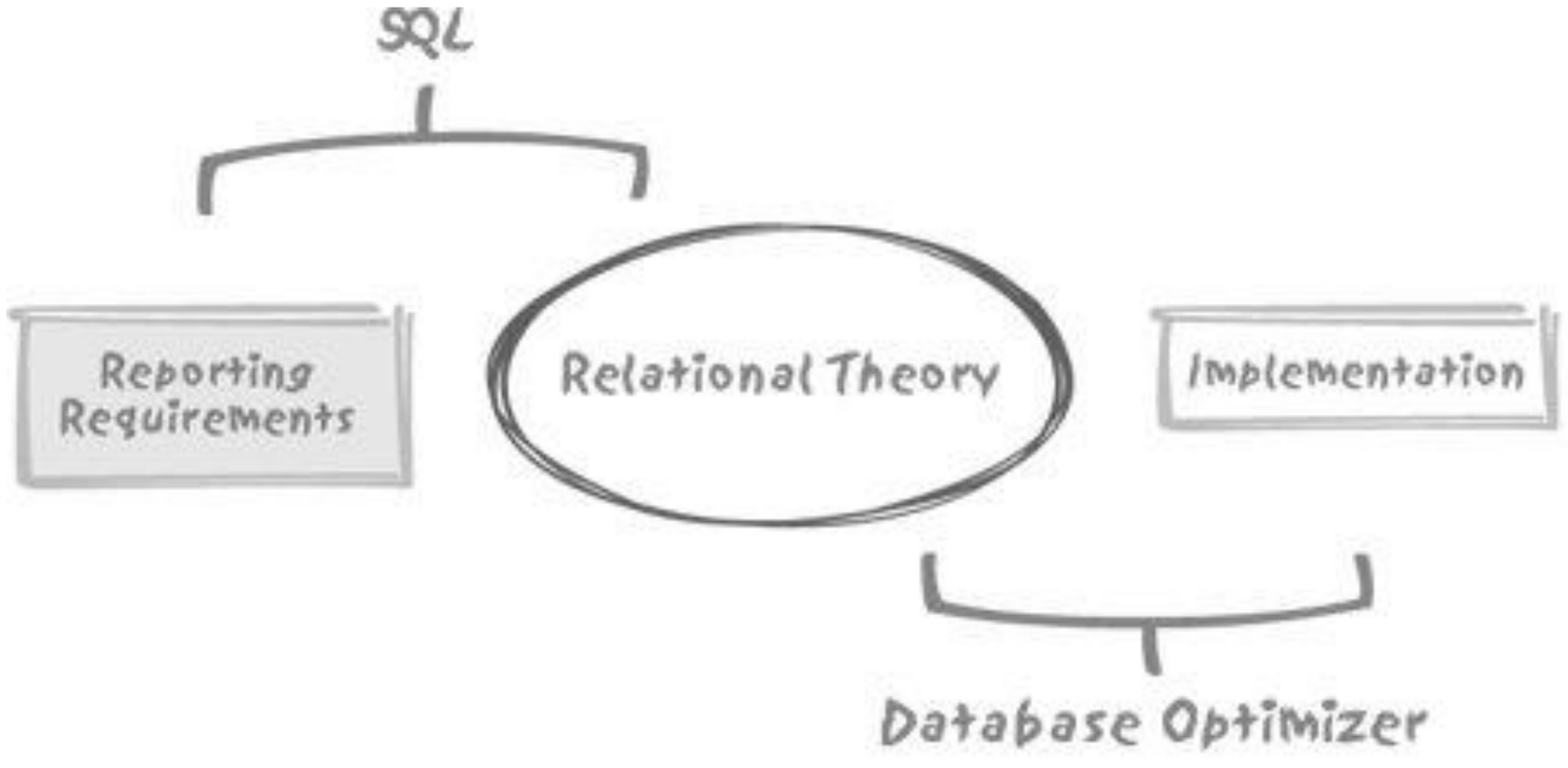
Chapter 4

查询优化器和SQL优化

--SQL的本质（The Nature of SQL）

SQL和数据库

- E.F.Codd关系理论之父
- SQL语言、数据库和关系模型



SQL和优化器

- 优化器借助关系理论提供的语义无误的原始查询进行有效的等价变换

- 优化时在数据处理真正被执行时发生的
  - 索引、数据的物理布局、可用内存大小、可用处理器个数、直接或间接涉及的表和索引的数据量

- 优化器根据数据库的实际实现情况对理论上等价的不同优化方案做出权衡，产生可能的最优查询执行方案

SQL执行顺序

- 语法
- 语义
- 解析
  - 硬解析
  - 软解析
- 执行计划
- 执行....（缓冲区寻找数据，或者磁盘读取）
  - 逻辑读
  - 物理读
  - 物理写

绑定变量的问题

- 性能问题

```
Create or replace procedure proc1
As
Begin
 for I in 1..10000
 loop
  execute immediate
  'insert into t values(:x)' using I;
 end loop
End;
```

```
Create or replace procedure proc1
As
Begin
 for I in 1..10000
 loop
  execute immediate
  'insert into t values('||i||')';
 end loop
End;
```

绑定变量的问题

- 安全问题
- String sql = "SELECT id,nick FROM user WHERE username='"+username+"' AND password='"+password+"'";
- 假设攻击者从客户端传的username为任意值（如test）password变量为 1’ or ‘1’=’1
- 实际传到DBMS的SQL为：
  - SELECT id,nick FROM user WHERE username='test' AND password='1' or '1'='1'

绑定变量的问题

- 解决安全问题

- String sql = "SELECT id,nick FROM user WHERE username=? AND password=?";
  preparedStatement.setString(1,username);
  preparedStatement.setString(2,password);

- Spring的SimpleJdbcTemplate

- String sql = "SELECT id,nick FROM user WHERE username=? AND password=?";
  jdbcTemplate.queryForList(sql,username,password);

SQL和优化器

- SQL“方言”会误导用户以为自己活在关系世界中
  - 例子：不是经理的员工当中，找出收入最高的5个员工
  - Oracle中有个rownum的虚拟字段为结果编号

```
select empname, salary
from employees
where status != 'EXECUTIVE'
        and rownum <= 5
order by salary desc
```

SQL和优化器

- 用oracle的都中过招

```
select *
from (select empname, salary
        from employees
       where status != 'EXECUTIVE'
       order by salary desc)
where rownum <= 5
```

SQL和优化器

- 一旦查询中的关系操作结束就再也回不去了
  - 把查询结果传给外部查询的关系操作
  - 无论优化器多么聪明，都不回合并两个查询，而只是顺序执行
  - 只要不是纯关系操作层，查询语句的编写对性能的影响重大，因为SQL引擎将严格执行它规定的执行路径

- 最稳妥的是：在关系操作层完成尽量多的工作，对于不完全的关系操作，加倍留意查询的编写。

优化器的有效范围

- 优化器需要借助数据库中找到的信息
- 能够进行数学意义上的等价变换
- 优化器考虑整体响应时间
- 优化器改善的是独立的查询
- 如果是若干个小查询，优化器会个个优化；如果是一个大的查询，优化器会将它作为一个整体优化

使用SQL需要考虑的因素

- 获得结果集所需访问的数据量
- 定义结果集所需的查询条件
- 结果集的大小
- 获得结果集所涉及的表的数量
- 同时修改这些数据用户的多少

数据总量

- SQL考虑最重要因素：必须访问的数据总量
- 没有确定目标容量之前，很难断定查询执行的效率

定义结果集的查询条件

- Where子句，特别在子查询或视图中可能有多个where子句
- 过滤条件的效率有高有低，受到其他因素的影响很大
- 影响因素：过滤条件、主要SQL语句、庞大的数据量对查询的影响

结果集的大小

- 查询所返回的数据量，重要而被忽略
  - 取决于表的大小和过滤条件的细节
  - 例外是若干个独立使用效率不高的条件结合起来效率非常高
- 从技术角度来看，查询结果集的大小并不重要，重要的是用户的感觉
- 熟练的开发者应该努力使响应时间与返回的记录数成比例

表的数量

- 表的数量会对性能有影响
- 连接
  - （太）多表连接该质疑设计的正确性了
  - 对于优化器，随着表数量的增加，复杂度将呈指数增长。
  - 编写（太）多表的复杂查询时，多种方式连接的选择失误的几率很高
- 复杂查询和复杂视图

并发用户数

- 设计的时候需要注意
  - 数据块访问争用（block-access contention）
  - 阻塞(locking)
  - 闩定(latching)
  - 保证读取一致性(read consistency)

- 一般而言，整体吞吐量>个体响应时间

过滤

- 如何限定结果集是最为关键的因素

- 是使用SQL各种技巧的判断因素

SQL查询语句的转换

- CBO发现常量或常量表达式会先行计算
  - CBO无法将等号（泛指，包括不等号大于号和小于号）一边的常量移动到另一边
  - 时间比较上特别容易出现类似问题

- 比较运算符的转换
  - IN
  - ANY/SOME
  - ALL
  - BETWEEN

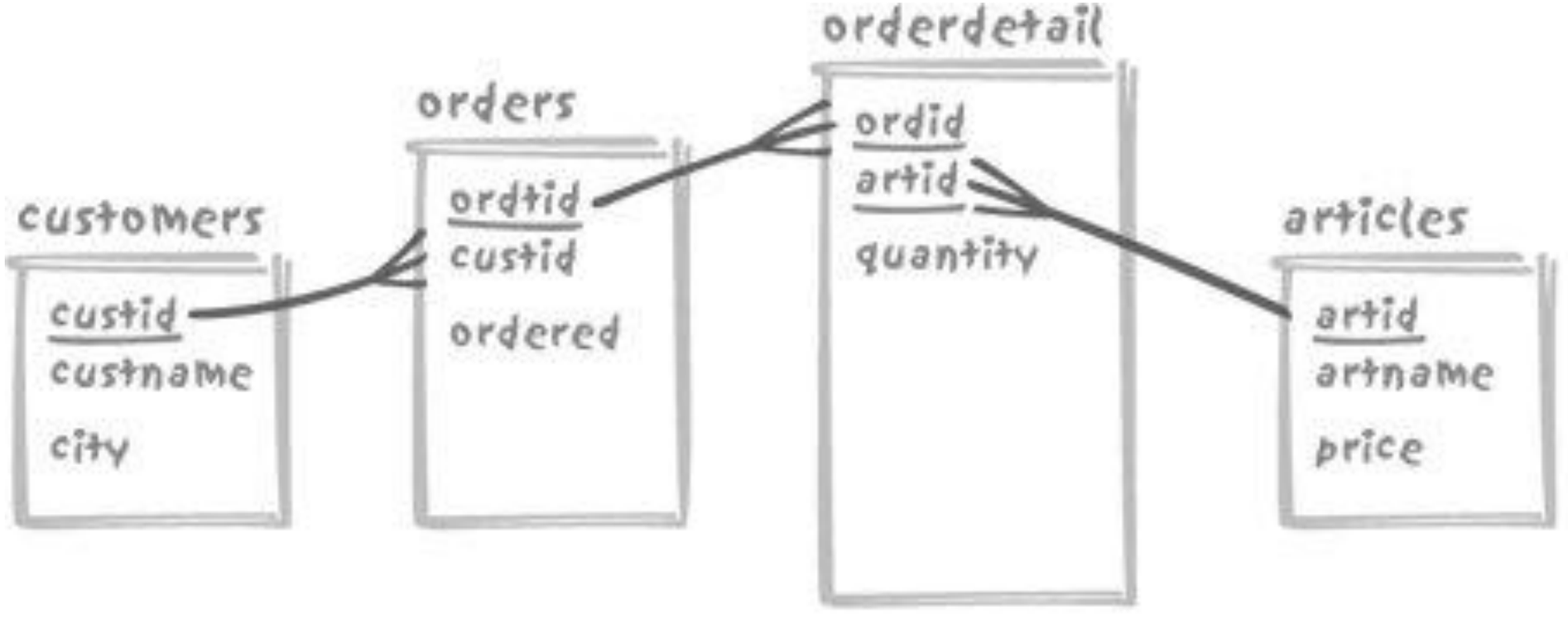
过滤条件的含义

- Where子句和having子句
  - Join过滤条件
  - Select过滤条件

过滤条件的好坏

- 最终需要的数据是什么，来自哪些表
- 哪些输入值会传递到DBMS引擎
- 能过滤掉不想要的数据的条件有哪些
- 高效过滤条件是查询的主要驱动力

来，去买BMW……


orderdetail
orders
ordid
artid
ordtid
quantity
custid
customers
custid
ordered
articles
custid
custname
city
artid
artname
price

找出最近6个月住在nanjing，购买了BMW的所有客户

```
select distinct  c.custname
    from customers c
        join orders o
          on o.custid = c.custid
       join orderdetail od
         on od.ordid = o.ordid
       join articles a
         on a.artid = od.artid
 where c.city = ‘Nanjing'
   and a.artname = ‘BMW'
   and o.ordered >= somefunc /*函数，返回六个月前的具体日期*/
```

古老的自然连接方式

```
select distinct c.custname
    from customers c,
        orders o,
        orderdetail od,
        articles a
where c.city = ‘Nanjing'
  and c.custid = o.custid
  and o.ordid = od.ordid
  and od.artid = a.artid
  and a.artname = ‘BMW'
  and o.ordered >= somefunc
```

进一步

- 避免在最高层distinct应该是一条基本规则
  - 发现重复数据容易，发现不准确的连接难
  - 发现结果不正确就更难了

摆脱distinct的方法

```
select c.custname
    from customers c
    where c.city = ‘Nanjing'
      and exists (select null
 from orders o,
          orderdetail od,
          articles a
where a.artname = ‘BMW'
     and a.artid = od.artid
     and od.ordid = o.ordid
     and o.custid = c.custid
     and o.ordered >= somefunc )
```

客户在Nanjing市，
而且满足Exists存在性测试
即在最近六个月买了BMW

Exists嵌套子查询和外层
select关系非常密切

非关联子查询

```
select custname
from customers
where city = ‘Nanjing'
    and custid in (select o.custid
                   from orders o,
                        orderdetail od,
                        articles a
                   where a.artname = ‘BMW'
                     and a.artid = od.artid
                     and od.ordid = o.ordid
                     and o.ordered >= somefunc)
```

关联子查询中，orders表中custid字段要有索引，而对非关联子查询则不需要，因为要用到的索引是customers的主键索引

内层查询不再依赖外层查询，只需要执行一次

还可以进一步嵌套

- 你们可以回去写写看吗？（作业）
- 子查询选择exists还是in的规则是一致的
  - 取决于日期与商品条件的有效性
  - 除非过去六个月生意清淡，否则商品名称是最有效的过滤条件，所以in比exists好
    - 因为找出BMW的订单再检查是不是在六个月内比反过来操作好，特别是orderdetail的artid字段有索引
  - 否则，exists比in好
- 还有更多种写法（比如from子句中的内嵌视图）

总结一下

- 找到分辨率最强的条件
- 解决方案不止一种，查询和数据隐含的假设密切相关
- 预先考虑优化器的工作，以确定它能找到所需要的数据

大数据量查询

- **越快剔除不需要的数据，查询的后续阶段必须处理的数据量就越少，查询效率就越高**

- Set operator是对该原则很好的应用
  - Union语句，但是不要cut-and-paste

- Group by & having子句也存在类似问题
  - 所有影响聚合函数结果的条件都应在Having子句
  - 任何无关聚合条件都应该放在where子句
  - 减少group by必须执行排序操作所处理的数据量

```
Select …
From A, B, C , D, E1
Where (condition on E1)
  and (join and other conditions)
Union
Select …
From A,B,C,D,E2
Where (condition on E2)
 and (join and other conditions)
```

这类就是典型的照搬式编程，把union语句降级为内嵌视图：

```
Select…
From A,B,C,D,
     (select …
       from E1
       where (condition on E1)
       union
      select…
       from E2
       where (condition on E2)
      )E
Where (joins and other conditions)
```

将子查询转换为JOIN

- 不包含聚合函数，不出现多种条件选择则不需要子查询

Jobs（employee，title） Ranks（title，rank）Salary（rank，payment）

```
Select payment from salary where rank=
  (select rank from ranks where title=
     (select title from jobs where employee = '…'))
```

```
Select payment from salary, ranks,jobs
 Where salary.rank = ranks.rank
   And ranks.title = jobs.title
   And jobs.employee = '…'
```

查询不存在的内容

- 是否存在某个等级当前没有分配职位
- 暴力方法

```
Select salary.ranks from salary
Where rank NOT IN (select rank from ranks)
```

- 外连接

```
Select salary.rank
From salary
        LEFT OUTER JOIN ON(salary.rank = ranks.rank)
Where ranks.rank IS NULL
```

将聚合子查询转换为JOIN

- Orders（custid，ordered，totalitems）
- 需要显示每一个客户购物件数最多的日期

```
Select custid, ordered, totalitems
From orders o1
Where o1.ordered = (
    select max(ordered)
    from orders o2
    where o1.custid = o2.custid )
```

将聚合子查询转换为JOIN(con’t)

```
Select o1.custid, o1.ordered, o1.totalitems

From orders o1
    JOIN orders o2 on (o1.custid = o2.custid)

Group by o1.custid, o1.ordered, o1.totalitems

Having o1.ordered = max(o2.ordered)
```

再回头看订单和客户的例子

- 在订单完成前有不同状态，记录在orderstatus（ordid,status,statusdate）中

- 需求是：列出所有尚未标记为完成状态的订单的下列字段：订单号，客户名，订单的最后状态，以及设置状态的时间

再回头看订单和客户的例子(con't)

```
select c.custname, o.ordid, os.status, os.statusdate
    from customers c,
            orders o,
            orderstatus os
 where o.ordid = os.ordid
   and not exists (select null
                          from orderstatus os2
                          where os2.status = 'COMPLETE'
                             and os2.ordid = o.ordid)
   and os.statusdate = (select max(statusdate)
                                 from orderstatus os3
                                 where os3.ordid = o.ordid)
```

非关联子查询变成内嵌视图

```
select c.custname, o.ordid, os.status, os.statusdate
    from customers c,
            orders o,
            orderstatus os,
           (select ordid, max(statusdate) laststatusdate
             from orderstatus
              group by ordid) x
where o.ordid = os.ordid
    and os.statusdate = x.laststatusdate
    and os.ordid = x.ordid
    and os.status != 'COMPLETE'
    and o.custid = c.custid
```