# SQL 语句性能调优

## 初级篇 —— 简单查询语句的调优

李明慧
2010 年 2 月 04 日发布

经常听到有做应用的朋友抱怨数据库的性能问题，比如非常低的并发，令人崩溃的响应时间，
死锁，等等。在解决这些问题的过程中，DBA 经常发现应用开发人员对数据库的“误用”。包括
和不适当加锁，对隔离级别的误用和对存储过程的误用等等。但是，面对浩如烟海的数据库知
员来说也确实枯燥艰深 . 因此，笔者特别提炼对应用开发人员有帮助的 SQL 书写部分，以期望

*“根据我们的经验（由很多业界专家证明），在 SQL Server 上取得的性能提高有 80% 来自对 S
于配置或系统性能的调整。”*
*—凯文 克莱恩等，Transact-SQL Programming 作者*

*“经验表明 80%-90% 的性能调优是在应用级做的，而不是在数据库级”*
*—托马斯 白特，Expert One on One: Oracle 作者*

本文将主要讨论基于语法的优化以及简简单的查询条件。基于语法的优化指的是为不考虑任何
小和存储等），仅考虑在 SQL 语句中对于词语的选择以及书写的顺序。

# 一般规则

这一部分，将看一下一些在书写简单查询语时需要注意的通用的规则。

## 根据权值来优化查询条件

最好的查询语句是将简单的比较操作作用于最少的行上。以下两张表，表 1 和表 2 以由好到差并赋与权值。

表 1. 查询条件中操作符的权值

| 操作符 | 权值 |
| --- | --- |
| = | 10 |
| > | 5 |
| >= | 5 |
| < | 5 |
| <= | 5 |
| LIKE | 3 |
| <> | 0 |

表 2. 查询条件中操作数的权值

| 操作数 | 权值 |
| --- | --- |
| 仅常量字符 | 10 |
| 仅有列名 | 5 |
| 仅有参数 | 5 |
| 多操作数表达式 | 3 |
| 精确数值类型 | 2 |
| 其它数值类型 | 1 |

| 操作数 | 权值 |
| --- | --- |
| 时间数据类型 | 1 |
| 字符数据类型 | 0 |
| NULL | 0 |

根据表 1 和表 2 中分配的权值，可以看出最好的查询条件应该是像下面这样的：

```
1 … WHERE smallint_column = 789
```

这个例子得到 27 分，计算如下：

- 左侧只有列名（smallint_column）得 5 分
- 操作数为精确数据类型（smallint_column）得 2 分
- 等号（=）操作符得 10 分
- 右侧是文字字符（789）得 10 分

下面是另外一个例子

```
1 … WHERE char_column >= varchar_column || ‘ x ’
```

这种类型的查询权值得分就很低，只有 13 分

- 左侧只有列名（char_column）得 5 分
- CHAR 类型的操作数得 0 分
- 大于等于操作符得 5 分
- 左侧是多操作数表达示得 3 分
- VARCHAR 类型的操作数得 0 分

上面表格中的权值数可能在不同类型的数据库系统中会有所不同，所以记住这些具体数值是没有
即可。用时越少的比较条件，得分也就越高，这样的比较条件通常是那些操作的行数少或者易于

## 传递法则

传递法则是这样定义的：

```
IF

(A <comparison operator> B) IS TRUE
AND (B <comparison operator> C) IS TRUE

THEN

(A <comparison operator> C) IS TRUE
AND NOT (A <comparison operator> C) IS FALSE
```

比较运算符包括：=、>、>=、<、<、+, 但不包括：<>、LIKE。

通过传递法则，我们可以看出，可以用 C 来替换 B，而不使表达式的意思发生变化。

下面的两个例子表达了同样的含义，但是第二个表达示要比第一个表达式执行的快。

表达式一：

```
... WHERE column1 < column2

AND column2 = column3

AND column1 = 5
```

表达式二：

```
... WHERE 5 < column2

AND column2 = column3

AND column1 = 5
```

大多数的数据库管理系统都会自动的做这样的调整，但是当表达式中含有括号时，它们就不会
SELECT 语句：

```
SELECT * FROM Table1

WHERE column1 = 5 AND

NOT (column3 = 7 OR column1 = column2)
```

如果进行转化的话，会得到如下的语句：

```
SELECT * FROM Table1

```

```
WHERE column1 = 5

AND column3 <> 7

AND column2 <> 5
```

进行这样变化后的语句会比第一个执行的更快。

## 1. Sargability

理想的 SQL 表达式应该采用下面这种通用的格式：

```
<column> <comparison operator> <literal>
```

早些时候，IBM 研究人员将这种查询条件语名命名为”sargable predicates”，因为 SARG 是 Se

根据这一规则，查询条件的左侧应该是一个列名；右侧应该是一个很容易进行查找的值。

遵循这一规则，所有的数据库系统都会将如下的表达式：

```
5 = column1
```

转换成：

```
column1 = 5
```

但是当查询条件中包含算术表达式时，只有部分的数据库系统进行转换。

例如：

```
... WHERE column1 - 3 = -column2
```

转换成：

```
... WHERE column1 = -column2 + 3
```

还是可以带来查询性能的优化的。

# 针对专门操作符的调优

前面，讲的是关于查询条件的一般规则，在这一节中，将讨论如何使用专门的操作符来改进 SQ

## 与 (AND)

数据库系统按着从左到右的顺序来解析一个系列由 AND 连接的表达式，但是 Oracle 却是个例外
可以利用数据库系统的这一特性，来将概率小的表达示放在前面，或者是如果两个表达式可能
达式放在前面。这样做的话，如果第一个表达式为假的话，那么数据库系统就不必再费力去解
样转换：

```
... WHERE column1 = 'A' AND column2 = 'B'
```

转换成：

```
... WHERE column2 = 'B' AND column1 = 'A'
```

这里假设 column2 = 'B'的概率较低，如果是 Oracle 数据库的话，只需将规则反过来用即可。

## 或 (OR)

和与 (AND) 操作符相反，在用或 (OR) 操作符写 SQL 语句时，就应该将概率大的表达示放在左
话，OR 操作符意味着需要进行下一个表达示的解析。

## 与 + 或

按照集合的展开法则，

```
A AND （B OR C） 与 （A AND B） OR （A AND C） 是等价表达示。
```

假设有如表 3 所示的一张表，要执行一个 AND 操作符在前的表达示

```
SELECT * FROM Table1

WHERE (column1 = 1 AND column2 = 'A')

OR (column1 = 1 AND column2 = 'B')
```

表 3. AND+OR 查询

| Row# | Colmun1 | Column2 |
|---|---|---|
| 1 | 3 | A |
| 2 | 2 | B |
| 3 | 1 | C |

当数据库系统按照查询语进行搜索时，它按照下面的步骤执行：

- 索引查找 column1 = 1, 结果集 = {row 3}
- 索引查找 column2 = ‘ A ’ , 结果集 = {row1}
- AND 合并结果集，结果集 = {}
- 索引查找 column 1 = 1, 结果集 = {row 3}
- 索引查找 column 2 = ‘ B ’ , 结果集 = {row2}
- AND 合并结果集，结果集 = {}
- OR 合并结集，结果集 = {}

现在根据集合的展开法则，对上面的语句进行转换：

```
1  SELECT * FROM Table1
2
3  WHERE column1 = 1
4
5  AND (column2 = 'A' OR column2 = 'B')
```

按照新的顺序进行查搜索时，它按照下面的步骤执行：

- 索引查找 column2 = ‘ A ’ , 结果集 = {row1}
- 索引查找 column 2 = ‘ B ’ , 结果集 = {row2}
- OR 合并结集，结果集 = {}
- 索引查找 column1 = 1, 结果集 = {row 3}
- AND 合并结果集，结果集 = {}

由此可见搜索次数少了一次。虽然一些数据库操作系统会自动的进行这样的转换，但是对于简
好处的。

## 非 (NOT)

让非 (NOT) 表达示转换成更易读的形式。简单的条件能通过将比较操作符进行反转来达到转换

```
1 ... WHERE NOT (column1 > 5)
```

转换成：

```
1 ... WHERE column1 <= 5
```

比较复杂的情况，根据集合的摩根定理：

```
1 NOT (A AND B) = (NOT A) OR (NOT B) 和 NOT (A OR B) = (NOT A) AND (NOT B)
```

根据这一定理，可以看出它可以至少二次的搜索有可能减少为一次。如下的查询条件：

```
1 ... WHERE NOT (column1 > 5 OR column2 = 7)
```

可以转换成：

```
1 ... WHERE column1 <= 5
2
3 AND column2 <> 7
```

但是，当转换成后的表达示中有不等操作符 <>，那么性能就会下降，毕竟，在一个值平均分布
大于相等的值的个数，正因为如此，一些数据库系统不会对非比较进行索引搜索，但是他们会
可以将下面的查询进行如下转换：

```
1 ... WHERE NOT (column1 = 0)
```

转换成：

```
1 ... WHERE column <0
2
3 OR column > 0
```

## IN

很多人认为如下的两个查询条件没有什么差别，因为它们返回的结果集是相同的：

条件 1：

```
... WHERE column1 = 5

OR column1 = 6
```

条件 2：

```
... WHERE column1 IN (5, 6)
```

这样的想法并不完全正确，对于大多数的数据库操作系统来说，IN 要比 OR 执行的快。所以如

当 IN 操作符，是一系列密集的整型数字时，最好是查找哪些值不符合条件，而不是查找哪些值
件就应该进行如下的转换：

```
... WHERE column1 IN (1, 3, 4, 5)
```

转换成：

```
... WHERE column1 BETWEEN 1 AND 5
AND column1 <> 2
```

当一系列的离散的值转换成算数表达示时，也可获得同样的性能提高。

## UNION

在 SQL 中，两个表的 UNION 就是两个表中不重复的值的集合，即 UNION 操作符返返回的两个
合。这是一个很好的合并数据的方法，但是这并不是最好的方法。

查询 1：

```
SELECT * FROM Table1

WHERE column1 = 5

```

```
UNION

SELECT * FROM Table1

WHERE column2 = 5
```

查询 2：

```
SELECT DISTINCT * FROM Table1

WHERE column1 = 5

OR column2 = 5
```

在上面的例子中，column1 和 column2 都没有索引。如果查询 2 总是比查询 1 执行的快的话，换成查询 2，但是有一种情况，这样做在一些数据库系统中可能会带来性能变差，这是由于两个

第一个优化缺陷就是很多优化器只优化一个 SELECT 语句中一个 WHERE 语句，所以查询 1 的两优化器根据查询条件 column1 = 5 为真来查找所有符合条件的所有行，然后据查询条件 colum所有行，即两次表扫描，因此，如果 column1 = 5 没有索引的话，查询 1 将需要 2 倍于查询 2 索引的话，仍然需要二次扫描，但是只有在某些数据库系统存在一个不常见的优化缺陷却将第化器发现查询中存在 OR 操作符时，就不使用索引查询，所以在这种情况下，并且只有在这种高。这种情况很少见，所以仍然建议大家当待查询的列没有索引时使用 OR 来代替 UNION。

# 总结

以上是作者对如何提高 SQL 性能的一些总结，这些规则并一定在所有的数据库系统上都能带来数据库的性能带来下降，所以掌握并使用这些规则可以对数据库应用程序的开发有所帮助。本较初级的方面，SQL 调优还包括 Order by，Group by 以及 Index 等等，作者将来后续的文章中

---

## 相关主题

- 通过 developerWorks Information Management 专区,获得关于 DB2、Informix、InfoSphe术资料和最佳实践。
- 访问 DB2 9 技术资源中心, 了解 DB2 产品家族的更多产品信息和相关技术。
- 访问 Optim 产品专题，这里汇聚了关于 Optim 集成数据管理和数据库应用开发相关的技术
- 访问 Optim Development Studio and pureQuery Runtime 集成开发产品专题，这里汇聚了关产品的技术特性和相关学习资源。
- 随时关注 developerWorks 技术活动 和 网络广播。