初级 SQL 优化

Oracle SQL 语句优化原理和技巧

ajksunkang

August 2019

0 前言

学习 SQL 优化，从本质上讲就是学习如何从优化器和执行计划的角度看待 SQL 语句的执行，看待 SQL 语句如何解析、优化器如何制定出最优的执行计划！

1 SQL 优化原理

SQL 本无需优化，SQL 语句的实现难度业并不大。但是随着物联网产业的涌现，移动互联网、5G 通信技术、云存储技术的发展，游离在互联网中的数据量在指数型膨胀，计算机系统的复杂度变高，用户的访问频率、访问量和并发度也都在增加，这些压力都加到了数据库模块上，而且任何 IT 系统中数据都是核心，所以 SQL 必须要优化。

简单地，SQL 优化所需要的基本知识包括：优化器、执行计划、Cursor、绑定变量、查询转换、统计信息、Hint 提示和并行。限于篇幅和笔者的水平，这里就简单介绍一些基本概念，为具体怎样做 SQL 优化做基础知识的储备。如果有疑问，可以参阅`Reference` 中的书籍资料。

1.1 优化器

优化器是数据库最核心的功能，也是最复杂的一部分。它负责将用户提交的 SQL 语句根据各种判断标准，从所有的执行路径中找到最优的执行路径，交由执行器最终执行。有基于规则和基于成本的优化器，目前 Oracle 采用的都是基于成本的优化器（`CBO, Cost Based Optimizer`）。

1.2 执行计划

执行目标 SQL 语句，**优化器最终选择的最优执行路径中所有步骤的顺序组合，叫做执行计划**。为了分析 SQL 语句，最必须的操作就是查看执行计划。但需要注意的是，Oracle 提供给我们的执行计划和真是执行可能完全不同，这取决于我们采用的统计信息收集方法。很多 DBA 到现在还不知道执行计划可能有假这个真相！所以有时候不用太迷恋 Oracle 呈现出来的执行计划，不过基本上沿用给定的执行计划来分析 SQL 就可。

1.3 绑定变量

1.3.1 绑定变量定义

绑定变量是数据库中的一种特殊类型的变量，又称占位符，用于替代 where 语句或者 value 子句的具体输入值。引入绑定变量可以大大减少 SQL 语句的硬解析次数，提高 SQL 语句重用度和系统性能。

```
select * from test where owner='aaa'
select * from test where owner='bbb'
select * from test where owner='ccc'
select * from test where owner='ddd'
select * from test where owner='eee'
```

语句中，'aaa' 等可以用:B1 替代，此时:B1 是绑定变量，可以传入任意值，这样以上 5 条语句只需要解析一次。

```
select * from test where owner=:B1;
```

1.3.2 绑定变量窥探

Oracle 9i 在第一次将绑定变量的真实值传入生成执行计划后，后续带有绑定变量的 SQL 语句会复用这个执行计划，也就是绑定变量只窥探 1 次。这种方案的问题是，一旦第一次的真实值恰好是比较特殊的值，那这将严重影响产生的执行计划和后续语句的执行效率.

为此 Oracle 11g 增加了新的特性——自适应游标共享(Adaptive Cursor Sharing), 对于一个同样绑定变量的 SQL 可以有多个执行计划，从而达到动态优化执行计划的作用。

1.4 查询转换

对子查询的等价改写，参考 3.2 子查询展开

1.5 统计信息

Oracle 的统计信息是存储在数据字典中的一组数据。它可以从多个维度描述存储在 Oracle 数据库中的对象或 Oracle 系统本身的详细信息。**基于成本的优化器正是利用了统计信息来分析生成执行计划的**。如果没有收集统计信息，或者统计信息过期了，那么优化器就会产生严重偏差，从而导致性能问题。因此要确保统计信息准确性。在进行 SQL 语句优化过程中，第一步的操作往往就是查看相关对象的统计信息是否完整、准确。

统计信息主要分为表的统计信息、列的统计信息、索引的统计信息、系统的统计信息、数据字典的统计信息以及动态性能试图基表的统计信息，本文章重点关注表的统计信息、列的统计信息和索引的统计信息。

1.6 Oracle 里的 Hint 提示

Hint 是 select 语句中特殊的注释，用于提示优化器对于执行计划的选择，但这种影响不是强制性的。如果优化器判断这个 Hint 给出的建议是合适的，可以应用，就会直接遵循这个 Hint 给出的

建议，并据此选择执行计划；反之，优化器会忽略该 Hint, 仍然采用原先的判断标准来选择执行计划。

例子：嵌套循环的 Hint `/* use_nl*/`

```
select /*use_nl(a,b)*/ a.dname,b.empno from dept a,emp b
where a.deptno = b.deptno;
```

1.7 并行

并行是 Oracle 支持的一种处理方式，目的是将一条语句映射到多个进程上执行，最后整合结果返回给用户。包括：语句级、会话级、对象级三种级别，笔者没做深入研究，学习者可以直接阅读参考文献中书籍。

1.8 表的扫描方式

表的扫描方式主要分为全表扫描和索引扫描。

1.9 表的连接方式

Oracle 的表连接有内连接和外连接两种，其中外连接又分为全外连接、左连接和右连接。

表连接的实现方式有三种：嵌套循环、排序合并、哈希连接。

从主查询和子查询的角度又有半连接和反连接两种连接方式，半连接和反连接都可以分别采用嵌套循环、排序合并、哈希连接的实现方式。

表 1: 三种表连接实现方式的比较

类别	嵌套循环	排序合并	哈希连接
优化器提示	USE_NL	USE_MERGE	USE_HASH
使用条件	任何连接	主要用于不等价连接，如 <、<=、>、>=, 但是不包括不等于 <>	仅用于等价连接 =
相关资源	CPU、磁盘 IO	内存、临时空间	内存、临时空间
特点	当有高选择性索引或者进行限制性搜索时效率比较高，能够快速返回第一次的搜索结果	当缺乏索引或者索引条件模糊时，排序合并比嵌套循环有效	当缺乏索引或者索引条件模糊时，哈希连接比嵌套循环有效。通常比排序合并连接快。在数据仓科环境下，如果表的记录数较多时，效率较高
缺点	当索引丢失或者查询条件限制不够时，效率很低；当表的记录数较多时，效率较低	所有的表都需要排序。它为最优化的吞吐量而设计，并且在结果没有全部找到前不返回数据	为建立哈希表需要大量内存。第一次的结果返回较慢

总的来说，当外部驱动表比较小，内部被驱动表包含索引时，嵌套循环的效率很高。

2 SQL 优化方法论

Oracle 里面的 SQL 优化本质是从优化器和执行计划的角度理解 SQL 语句的执行。SQL 优化的终极目标是降低目标 SQL 语句的执行时间，通常有如下的方式可以进行选择：

- 对目标 SQL 语句进行等价重写，在不更改业务逻辑的情况下
- 建立合适的索引避免不必要的全表扫描/排序导致的目标 SQL 性能问题
- 使用 Hint 提示
- 并行执行目标 SQL 语句
- 如果上述方式都失效，我们还可以在联系实际业务的基础上更改目标 SQL 的业务逻辑，甚至不执行 SQL，这是最彻底的 SQL 优化:-)

3 SQL 优化案例

假定优化器比较蠢

3.1 索引

通过添加索引来提高查询性能，是最为常见的一种优化手段。甚至很多非 DBA 人员认为，数据库优化就是加索引，虽然有失偏颇，但也说明了索引对于优化的重要意义。

3.1.1 索引的优化案例

通常索引的数据结构是 B 树索引，添加索引就是用为索引列建立并维护一个 B 树数据结构，用来存储该列的键值和对应的`ROWID`，搜索的过程就变成了 B 树搜索，所以索引扫描比全表扫描要快得多。因此, 添加索引会换来很多 select 的性能提升。

使用索引优化，简单地说，就是将 where 语句中的列建立索引。

优化案例 SQL1：

```
select * from t1 where object_id = 1;
```

在 where 后的列创建索引：

```
create index idx_t1 on t1(object_id) tableplace users;
select * from t1 where object_id = 1;
```

注意，where 后的索引键值不能用来判断 NULL 或者 NOT NULL，因为 NULL 不入 B 树索引，所以在执行以下 SQL 语句时即使`object_id` 列建立了索引也会走全表扫描。(解决办法是建立符合 B 树索引)

```
select * from t1 where object_id is NULL;
```

解决办法：复合 B 树索引，第二列是一个常数 0：

```
create index idx_t1 on t1(object_id,0) tablespace users;
```

3.1.2 索引的局限性

当然，添加索引并不是只有好处，建立索引也会有额外的内存开销。以下两种情况不适合建立索引：

- 1. 对于一个查询操作远远小于修改、插入、删除操作的数据库表，不建立索引。因为添加索引会增加插入、修改、删除等 DML 的开销，来维护 B 数索引数据结构。
- 2. 对于选择性不高的列，不建立索引。选择性表征的是非重复元素占所有元素的比例，选择性过低，该列的重复数据就会过高，索引扫描会退化到全表扫描。

3.2 子查询展开

子查询是在一个 select 语句中嵌套另一个 select 语句的情况，通常 where 条件中带有关键字EXIST、NOT EXIST、IN、NOT IN、ANY、ALL、=、<、>、<=、>=、<> 等。如果不做子查询展开，子查询通常会在目标 SQL 执行计划的最后一步才会被执行，并且会走FILTER 类型的执行计划，这就意味着对于外部查询所在结果集中的每一条记录，该子查询都会当成一个独立的执行单元来执行一次，外部查询所在的结果集有多少条记录，子查询执行单元就会被执行多少次，这种FILTER 类型的执行计划通常效率都不高。当子查询展开成两个表或两个表以上的表连接时，优化器会有更多更高效的执行路径可以选择。

子查询展开的核心思想是将嵌套子查询等价改写成表连接。主要分为两种情形：

- 情形一：将子查询拆开，将它自身与外部查询等价改写成表连接；
- 情形二：子查询无法拆开，将它自身转换成一个内嵌视图(VIEW)，再与外部查询进行表连接；

优化案例 SQL2：

```
select t1.cust_last_name,t1.cust_id from customers t1 where t1.cust_id
   in (select t2.cust_id from sales t2 where t2.amount_sold > 700);
```

改写为：

```
select t1.cust_last_name,t1.cust_id from customers t1, sales t2 where
   t1.cust_id = t2.cust_id and t2.amount_sold > 700;
```

优化案例 SQL3：

```
select t1.cust_last_name,t1.cust_id from customers t1 where t1.cust_id
   in (select t2.cust_id from sales t2,products t3 where t2.prod_id =
   t3.prod_id and t2.amount_sold > 700);
```

改写为：

```
select t1.cust_last_name,t1.cust_id from customers t, (select t2.
   cust_id from sales t2,products t3 where t2.prod_id = t3.prod_id and
   t2.amount_sold > 700) VW_NSO_1 where t1.cust_id = VM_NSO_1.cust_id;
```

3.3 OR 用 Union 替代

or 后接的子查询是无法展开的，会走 FILTER 这种低效的执行计划，用 UNION 进行改写，从而消除 FILTER。

优化案例 SQL4:

```
select * from t1 where owner = 'SCOTT' or object_id in (select
    object_id from t2)
```

改写为:

```
select * from t1 where owner = 'SCOTT' union select * from t1 where
    object_id in (select object_id from t2)
```

3.4 分页语句的优化

分页语句最能考察一个人究竟会不会 SQL 优化，因为分页语句几乎囊括了所有 SQL 优化必须具备的知识。形如

```
select * from (select t.*,rownum rn from(需要分页的SQL) t) where rn >=
    1 and rn <= 10;
```

都可以进行优化，基本的方案是用`COUNT STOPKEY`替换`COUNT`，笔者对分页语句一知半解，但是`MYSQL`对`LIMIT M, N`的优化有非常成熟的思路，建议读者查阅。

4 参考文献

[1] 韩峰.《SQL 优化最佳实践》. 机械工业出版社.2016

[2] 梁敬彬, 梁敬弘.《收获，不止 SQL 优化》. 中国工信出版社.2017

[3] 罗炳森.《SQL 优化核心思想》. 中国工信出版社.2018

[4] 崔华.《基于 Oracle 的 SQL 优化》. 电子工业出版社.2014

[5] 黄玮.《Oracle 高性能 SQL 引擎剖析》. 机械工业出版社.2013