**Oracle性能优化**

**一、Oracle的优化器共有3种**

**1、RULE (基于规则) 2、COST (基于成本) 3、CHOOSE (选择性)**

设置缺省的优化器，可以通过对init.ora文件中OPTIMIZER\_MODE参数的各种声明，如RULE，COST，CHOOSE，ALL\_ROWS，FIRST\_ROWS 。 你当然也在SQL句级或是会话(session)级对其进行覆盖。

为了使用基于成本的优化器(CBO， Cost-Based Optimizer) ， 你必须经常运行analyze 命令，以增加数据库中的对象统计信息(object statistics)的准确性。

如果数据库的优化器模式设置为选择性(CHOOSE)，那么实际的优化器模式将和是否运行过analyze命令有关。 如果table已经被analyze过， 优化器模式将自动成为CBO ， 反之，数据库将采用RULE形式的优化器。

在缺省情况下，Oracle采用CHOOSE优化器， 为了避免那些不必要的全表扫描(full table scan) ， 你必须尽量避免使用CHOOSE优化器，而直接采用基于规则或者基于成本的优化器。   
**二、访问Table的方式**     Oracle 采用两种访问表中记录的方式：

**1、 全表扫描**

          全表扫描就是顺序地访问表中每条记录。Oracle采用一次读入多个数据块(database block)的方式优化全表扫描。

**2、 通过ROWID访问表**

          你可以采用基于ROWID的访问方式情况，提高访问表的效率， ROWID包含了表中记录的物理位置信息。Oracle采用索引(INDEX)实现了数据和存放数据的物理位置(ROWID)之间的联系。通常索引提供了快速访问ROWID的方法，因此那些基于索引列的查询就可以得到性能上的提高。   
**三、共享SQL语句** 为了不重复解析相同的SQL语句，在第一次解析之后，Oracle将SQL语句存放在内存中。这块位于系统全局区域SGA(system global area)的共享池(shared buffer pool)中的内存可以被所有的数据库用户共享。 因此，当你执行一个SQL语句(有时被称为一个游标)时，如果它和之前的执行过的语句完全相同， Oracle就能很快获得已经被解析的语句以及最好的执行路径。Oracle的这个功能大大地提高了SQL的执行性能并节省了内存的使用。

可惜的是Oracle只对简单的表提供高速缓冲(cache buffering)，这个功能并不适用于多表连接查询。

数据库管理员必须在init.ora中为这个区域设置合适的参数，当这个内存区域越大，就可以保留更多的语句，当然被共享的可能性也就越大了。

当你向Oracle提交一个SQL语句，Oracle会首先在这块内存中查找相同的语句。这里需要注明的是，Oracle对两者采取的是一种严格匹配，要达成共享，SQL语句必须完全相同(包括空格，换行等)。

数据库管理员必须在init.ora中为这个区域设置合适的参数，当这个内存区域越大，就可以保留更多的语句，当然被共享的可能性也就越大了。

共享的语句必须满足三个条件：

     1、 字符级的比较： 当前被执行的语句和共享池中的语句必须完全相同。

     2、 两个语句所指的对象必须完全相同：

     3、 两个SQL语句中必须使用相同的名字的绑定变量(bind variables)。   
**四、选择最有效率的表名顺序(只在基于规则的优化器中有效)** Oracle的解析器按照从右到左的顺序处理FROM子句中的表名，因此FROM子句中写在最后的表(基础表 driving table)将被最先处理。在FROM子句中包含多个表的情况下，你必须选择记录条数最少的表作为基础表。当Oracle处理多个表时， 会运用排序及合并的方式连接它们。首先，扫描第一个表(FROM子句中最后的那个表)并对记录进行派序，然后扫描第二个表(FROM子句中最后第二个表)，最后将所有从第二个表中检索出的记录与第一个表中合适记录进行合并。

如果有3个以上的表连接查询， 那就需要选择交叉表(intersection table)作为基础表， 交叉表是指那个被其他表所引用的表。   
**五、WHERE子句中的连接顺序** Oracle采用自下而上的顺序解析WHERE子句，根据这个原理，表之间的连接必须写在其他WHERE条件之前，那些可以过滤掉最大数量记录的条件必须写在WHERE子句的末尾。   
**六、SELECT子句中避免使用 ' \* '** 当你想在SELECT子句中列出所有的COLUMN时，使用动态SQL列引用 '\*' 是一个方便的方法。不幸的是，这是一个非常低效的方法。实际上，Oracle在解析的过程中， 会将'\*' 依次转换成所有的列名， 这个工作是通过查询数据字典完成的， 这意味着将耗费更多的时间。   
**七、减少访问数据库的次数** 当执行每条SQL语句时，Oracle在内部执行了许多工作：解析SQL语句，估算索引的利用率，绑定变量，读数据块等等。由此可见，减少访问数据库的次数，就能实际上减少Oracle的工作量。   
**八、使用DECODE函数来减少处理时间** 使用DECODE函数可以避免重复扫描相同记录或重复连接相同的表。   
**九、整合简单，无关联的数据库访问** 如果你有几个简单的数据库查询语句，你可以把它们整合到一个查询中(即使它们之间没有关系)   
**十、删除重复记录**

**十一、用TRUNCATE替代DELETE**     当删除表中的记录时，在通常情况下， 回滚段(rollback segments ) 用来存放可以被恢复的信息。 如果你没有COMMIT事务，Oracle会将数据恢复到删除之前的状态(准确地说是恢复到执行删除命令之前的状况)。

     而当运用TRUNCATE时， 回滚段不再存放任何可被恢复的信息。当命令运行后，数据不能被恢复。因此很少的资源被调用，执行时间也会很短。   
**十二、尽量多使用COMMIT**     只要有可能，在程序中尽量多使用COMMIT，这样程序的性能得到提高，需求也会因为COMMIT所释放的资源而减少

     COMMIT所释放的资源：

     1、 回滚段上用于恢复数据的信息。

     2、被程序语句获得的锁。

     3、 redo log buffer 中的空间。

     4、Oracle为管理上述3种资源中的内部花费。   
**十三、计算记录条数**     和一般的观点相反，count(\*) 比count(1)稍快，当然如果可以通过索引检索，对索引列的计数仍旧是最快的。例如 COUNT(EMPNO)   
**十四、用Where子句替换HAVING子句**     避免使用HAVING子句，HAVING 只会在检索出所有记录之后才对结果集进行过滤。 这个处理需要排序，总计等操作。如果能通过WHERE子句限制记录的数目，那就能减少这方面的开销。   
**十五、减少对表的查询**     在含有子查询的SQL语句中，要特别注意减少对表的查询。  
**十六、通过内部函数提高SQL效率**

**十七、使用表的别名(Alias)**     当在SQL语句中连接多个表时， 请使用表的别名并把别名前缀于每个Column上。这样一来，就可以减少解析的时间并减少那些由Column歧义引起的语法错误。   
**十八、用EXISTS替代IN**     在许多基于基础表的查询中，为了满足一个条件，往往需要对另一个表进行联接。在这种情况下，使用EXISTS(或NOT EXISTS)通常将提高查询的效率。   
**十九、用NOT EXISTS替代NOT IN**     在子查询中，NOT IN子句将执行一个内部的排序和合并。无论在哪种情况下，NOT IN都是最低效的 (因为它对子查询中的表执行了一个全表遍历)。为了避免使用NOT IN ，我们可以把它改写成外连接(Outer Joins)或NOT EXISTS。   
**二十、用表连接替换EXISTS**     通常来说 ， 采用表连接的方式比EXISTS更有效率   
**二十一、用EXISTS替换DISTINCT**     当提交一个包含一对多表信息(比如部门表和雇员表)的查询时，避免在SELECT子句中使用DISTINCT。 一般可以考虑用EXIST替换

**二十二、用索引提高效率：**

索引是表的一个概念部分,用来提高检索数据的效率，Oracle使用了一个复杂的自平衡B-tree结构. 通常,通过索引查询数据比全表扫描要快. 当Oracle找出执行查询和Update语句的最佳路径时, Oracle优化器将使用索引. 同样在联结多个表时使用索引也可以提高效率. 另一个使用索引的好处是,它提供了主键(primary key)的唯一性验证.。那些LONG或LONG RAW数据类型, 你可以索引几乎所有的列. 通常, 在大型表中使用索引特别有效. 当然,你也会发现, 在扫描小表时,使用索引同样能提高效率. 虽然使用索引能得到查询效率的提高,但是我们也必须注意到它的代价. 索引需要空间来存储,也需要定期维护, 每当有记录在表中增减或索引列被修改时, 索引本身也会被修改. 这意味着每条记录的INSERT , DELETE , UPDATE将为此多付出4 , 5 次的磁盘I/O . 因为索引需要额外的存储空间和处理,那些不必要的索引反而会使查询反应时间变慢.。定期的重构索引是有必要的.：   
 ALTER INDEX <INDEXNAME> REBUILD <TABLESPACENAME>

**二十三、sql语句用大写的**

因为Oracle总是先解析sql语句，把小写的字母转换成大写的再执行

**二十四、在java代码中尽量少用连接符“＋”连接字符串！**

**二十五、用>=替代>**高效: SELECT \* FROM EMP WHERE DEPTNO >=4

低效: SELECT \* FROM EMP WHERE DEPTNO >3

两者的区别在于, 前者DBMS将直接跳到第一个DEPT等于4的记录而后者将首先定位到DEPTNO=3的记录并且向前扫描到第一个DEPT大于3的记录.

**二十六、用UNION替换OR (适用于索引列)**

通常情况下, 用UNION替换WHERE子句中的OR将会起到较好的效果. 对索引列使用OR将造成全表扫描. 注意, 以上规则只针对多个索引列有效. 如果有column没有被索引, 查询效率可能会因为你没有选择OR而降低. 在下面的例子中, LOC\_ID 和REGION上都建有索引.

高效: SELECT LOC\_ID , LOC\_DESC , REGION FROM LOCATION WHERE LOC\_ID = 10 UNION SELECT LOC\_ID , LOC\_DESC , REGION FROM LOCATION WHERE REGION = “MELBOURNE”

低效: SELECT LOC\_ID , LOC\_DESC , REGION FROM LOCATION WHERE LOC\_ID = 10 OR REGION = “MELBOURNE”

如果你坚持要用OR, 那就需要返回记录最少的索引列写在最前面.

**二十七、避免在索引列上使用IS NULL和IS NOT NULL**

避免在索引中使用任何可以为空的列，Oracle将无法使用该索引．对于单列索引，如果列包含空值，索引中将不存在此记录. 对于复合索引，如果每个列都为空，索引中同样不存在此记录.　如果至少有一个列不为空，则记录存在于索引中．举例: 如果唯一性索引建立在表的A列和B列上, 并且表中存在一条记录的A,B值为(123,null) , Oracle将不接受下一条具有相同A,B值（123,null）的记录(插入). 然而如果所有的索引列都为空，Oracle将认为整个键值为空而空不等于空. 因此你可以插入1000 条具有相同键值的记录,当然它们都是空! 因为空值不存在于索引列中,所以WHERE子句中对索引列进行空值比较将使Oracle停用该索引.   
 低效: (索引失效) SELECT … FROM DEPARTMENT WHERE DEPT\_CODE IS NOT NULL;

高效: (索引有效) SELECT … FROM DEPARTMENT WHERE DEPT\_CODE >=0;

**二十八、用UNION-ALL 替换UNION ( 如果有可能的话)：**    当SQL语句需要UNION两个查询结果集合时,这两个结果集合会以UNION-ALL的方式被合并, 然后在输出最终结果前进行排序. 如果用UNION ALL替代UNION, 这样排序就不是必要了. 效率就会因此得到提高. 需要注意的是，UNION ALL 将重复输出两个结果集合中相同记录. 因此各位还是要从业务需求分析使用UNION ALL的可行性. UNION 将对结果集合排序,这个操作会使用到SORT\_AREA\_SIZE这块内存. 对于这块内存的优化也是相当重要的.

**二十九、需要当心的WHERE子句:**    某些SELECT 语句中的WHERE子句不使用索引.

(1)‘!=' 将不使用索引. 记住, 索引只能告诉你什么存在于表中, 而不能告诉你什么不存在于表中.

(2) ‘||'是字符连接函数. 就象其他函数那样, 停用了索引.

(3) ‘+'是数学函数. 就象其他数学函数那样, 停用了索引.

(4)相同的索引列不能互相比较,这将会启用全表扫描.

**三十、带有DISTINCT,UNION,MINUS,INTERSECT,ORDER BY的SQL语句会启动SQL引擎**

**执**行耗费资源的排序(SORT)功能. DISTINCT需要一次排序操作, 而其他的至少需要执行两次排序. 通常, 带有UNION, MINUS , INTERSECT的SQL语句都可以用其他方式重写. 如果你的数据库的SORT\_AREA\_SIZE调配得好, 使用UNION , MINUS, INTERSECT也是可以考虑的, 毕竟它们的可读性很强

**三十一、优化GROUP BY:**提高GROUP BY 语句的效率, 可以通过将不需要的记录在GROUP BY 之前过滤掉.