Oracle SQL 性能优化

（1） 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，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。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.

（2） 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.

（3） SELECT 子句中避免使用 ‘ * ‘：

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

（4） 减少访问数据库的次数：

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；

（5） 在 SQL*Plus , SQL*Forms 和 Pro*C 中重新设置 ARRAYSIZE 参数，可以增加每次数据库访问的检索数据量 ,建议值为 200

（6） 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.

（7） 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)

（8） 删除重复记录：

最高效的删除重复记录方法 (因为使用了 ROWID)例子：

```
DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID)
FROM EMP X WHERE X.EMP_NO = E.EMP_NO);
```

（9） 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. (译者按: TRUNCATE 只在删除全表适用,TRUNCATE 是 DDL 不是 DML)

（10） 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:

a. 回滚段上用于恢复数据的信息.

b. 被程序语句获得的锁

c. redo log buffer 中的空间

d. ORACLE 为管理上述 3 种资源中的内部花费

（11） 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. (非 oracle 中)on、where、having 这三个都可以加条件的子句中，on 是最先执行，where 次之，having 最后，因为 on 是先把不符合条件的记录过滤后才进行统计，它就可以减少中间运算要处理的数据，按理说应该速度是最快的，where 也应该比 having 快点的，因为它过滤数据后才进行 sum，在两个表联接时才用 on 的，所以在一个表的时候，就剩下 where 跟 having 比较了。在这单表查询统计的情况下，如果要过滤的条件没有涉及到要计算字段，那它们的结果是一样的，只是 where 可以使用 rushmore 技术，而 having 就不能，在速度上后者要慢如果要涉及到计算的字段，就表示在没计算之前，这个字段的值是不确定的，根据上篇写的工作流程，where 的作用时间是在计算之前就完成的，而 having 就是在计算后才起作用的，所以在这种情况下，两者的结果会不同。在多表联接查询

时，on 比 where 更早起作用。系统首先根据各个表之间的联接条件，把多个表合成一个临时表后，再由 where 进行过滤，然后再计算，计算完后再由 having 进行过滤。由此可见，要想过滤条件起到正确的作用，首先要明白这个条件应该在什么时候起作用，然后再决定放在那里

（12） 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.例子：

SELECT TAB_NAME FROM TABLES WHERE (TAB_NAME,DB_VER) = (SELECT TAB_NAME,DB_VER FROM TAB_COLUMNS WHERE VERSION = 604)

（13） 通过内部函数提高 SQL 效率.：

复杂的 SQL 往往牺牲了执行效率. 能够掌握上面的运用函数解决问题的方法在实际工作中是非常有意义的

（14） 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.

（15） 用 EXISTS 替代 IN、用 NOT EXISTS 替代 NOT 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. 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.

例子：

（高效）SELECT * FROM EMP (基础表) WHERE EMPNO > 0 AND **EXISTS** (SELECT 'X' FROM DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO AND LOC = 'MELB')

(低效)SELECT * FROM EMP (基础表) WHERE EMPNO > 0 AND DEPTNO **IN**(SELECT DEPTNO FROM DEPT WHERE LOC = 'MELB')

（16） 识别'低效执行'的 SQL 语句：

虽然目前各种关于 SQL 优化的图形化工具层出不穷,但是写出自己的 SQL 工具来解决问题始终是一个最好的方法：

```
SELECT EXECUTIONS , DISK_READS, BUFFER_GETS,
ROUND((BUFFER_GETS-DISK_READS)/BUFFER_GETS,2) Hit_radio,
ROUND(DISK_READS/EXECUTIONS,2) Reads_per_run,
SQL_TEXT
FROM V$SQLAREA
WHERE EXECUTIONS>0
AND BUFFER_GETS > 0
AND (BUFFER_GETS-DISK_READS)/BUFFER_GETS < 0.8
ORDER BY 4 DESC;
```

（17） 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>
```

（18） 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, EXISTS 使查询更为迅速,因为 RDBMS 核心模块将在子查询的条件一旦满足后,立刻返回结果. 例子：

(低效):

```
SELECT DISTINCT DEPT_NO,DEPT_NAME FROM DEPT D , EMP E
WHERE D.DEPT_NO = E.DEPT_NO
```

(高效):

```
SELECT DEPT_NO,DEPT_NAME FROM DEPT D WHERE EXISTS ( SELECT 'X'
FROM EMP E WHERE E.DEPT_NO = D.DEPT_NO);
```

（19） 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

（20） 在 java 代码中尽量少用连接符“+”连接字符串！

（21） 避免在索引列上使用 NOT 通常，

我们要避免在索引列上使用 NOT, NOT 会产生在和在索引列上使用函数相同的影响. 当 ORACLE”遇到”NOT,他就会停止使用索引转而执行全表扫描.

（22） 避免在索引列上使用计算.

WHERE 子句中，如果索引列是函数的一部分. 优化器将不使用索引而使用全表扫描.

举例:

低效：

```
SELECT … FROM DEPT WHERE SAL * 12 > 25000;
```

高效:

```
SELECT … FROM DEPT WHERE SAL > 25000/12;
```

（23） 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.

（24） 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.

（25） 用 IN 来替换 OR

这是一条简单易记的规则，但是实际的执行效果还须检验，在 ORACLE8i 下，两者的执行路径似乎是相同的.

低效:

```
SELECT…. FROM LOCATION WHERE LOC_ID = 10 OR LOC_ID = 20 OR LOC_ID = 30
```

高效
SELECT… FROM LOCATION WHERE LOC_IN IN (10,20,30);
（26） 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;
（27） 总是使用索引的第一个列：
如果索引是建立在多个列上，只有在它的第一个列(leading column)被 where 子句引用时,优化器才会选择使用该索引．这也是一条简单而重要的规则，当仅引用索引的第二个列时,优化器使用了全表扫描而忽略了索引
（28） 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．下面的 SQL 可以用来查询排序的消耗量
低效：
SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
UNION
SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
高效:
SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
UNION ALL
SELECT ACCT_NUM, BALANCE_AMT
FROM DEBIT_TRANSACTIONS
WHERE TRAN_DATE = '31-DEC-95'
（29） 用 WHERE 替代 ORDER BY：
ORDER BY 子句只在两种严格的条件下使用索引．
ORDER BY 中所有的列必须包含在相同的索引中并保持在索引中的排列顺序．
ORDER BY 中所有的列必须定义为非空．
WHERE 子句使用的索引和 ORDER BY 子句中所使用的索引不能并列．
例如:
表 DEPT 包含以下列:
DEPT_CODE PK NOT NULL

DEPT_DESC NOT NULL
DEPT_TYPE NULL
低效: (索引不被使用)
SELECT DEPT_CODE FROM DEPT ORDER BY DEPT_TYPE
高效: (使用索引)
SELECT DEPT_CODE FROM DEPT WHERE DEPT_TYPE > 0
（30） 避免改变索引列的类型.:
当比较不同数据类型的数据时, ORACLE 自动对列进行简单的类型转换.
假设 EMPNO 是一个数值类型的索引列.
SELECT … FROM EMP WHERE EMPNO = '123'
实际上,经过 ORACLE 类型转换, 语句转化为:
SELECT … FROM EMP WHERE EMPNO = TO_NUMBER('123')
幸运的是,类型转换没有发生在索引列上,索引的用途没有被改变.
现在,假设 EMP_TYPE 是一个字符类型的索引列.
SELECT … FROM EMP WHERE EMP_TYPE = 123
这个语句被 ORACLE 转换为:
SELECT … FROM EMP WHERETO_NUMBER(EMP_TYPE)=123
因为内部发生的类型转换, 这个索引将不会被用到! 为了避免 ORACLE 对你的 SQL 进行隐式的类型转换, 最好把类型转换用显式表现出来. 注意当字符和数值比较时, ORACLE 会优先转换数值类型到字符类型
（31） 需要当心的 WHERE 子句:
某些 SELECT 语句中的 WHERE 子句不使用索引. 这里有一些例子.
在下面的例子里, (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.
（32） a. 如果检索数据量超过 30%的表中记录数.使用索引将没有显著的效率提高.
b. 在特定情况下, 使用索引也许会比全表扫描慢, 但这是同一个数量级上的区别. 而通常情况下,使用索引比全表扫描要块几倍乃至几千倍!
（33） 避免使用耗费资源的操作:
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
（34） 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.下面两个查询返回相同结果但第二个明显就快了许多.
低效:
SELECT JOB , AVG(SAL)
FROM EMP
GROUP JOB
HAVING JOB = 'PRESIDENT'
OR JOB = 'MANAGER'
高效:
SELECT JOB , AVG(SAL)
FROM EMP
WHERE JOB = 'PRESIDENT'
OR JOB = 'MANAGER'
GROUP JOB