oracle 数据库的性能优化

2013-07-20 0 个评论 来源：tianfeng1208 的专栏

oracle 数据库的性能优化

对于 ORACLE 数据库的数据存取，主要有四个不同的调整级别，第一级调整是操作系统级包括硬件平台,第二级调整是ORACLE RDBMS 级的调整,第三级是数据库设计级的调整,最后一个调整级是 SQL 级。通常依此四级调整级别对数据库进行调整、优化，数据库的整体性能会得到很大的改善。下面从九个不同方面介绍 ORACLE 数据库优化设计方案。

一、数据库优化自由结构 OFA(Optimal flexible Architecture)

数据库的逻辑配置对数据库性能有很大的影响,优化自由结构 OFA,简单地讲就是在数据库中可以高效自由地分布逻辑数据对象,因此首先要对数据库中的逻辑对象根据他们的使用方式和物理结构对数据库的影响来进行分类,这种分类包括将系统数据和用户数据分开、一般数据和索引数据分开、低活动表和高活动表分开等等。

数据库逻辑设计的结果应当符合下面的准则：（1）把以同样方式使用的段类型存储在一起；（2）按照标准使用来设计系统；（3）存在用于例外的分离区域；（4）最小化表空间冲突；（5）将数据字典分离。

二、充分利用系统全局区域 SGA（SYSTEM GLOBAL AREA）

SGA 是 oracle 数据库的心脏。用户的进程对这个内存区发送事务，并且以这里作为高速缓存读取命中的数据，以实现加速的目的。正确的 SGA 大小对数据库的性能至关重要。SGA 包括以下几个部分：

1、数据块缓冲区（data block buffer cache）是 SGA 中的一块高速缓存，占整个数据库大小的 1%-2%,用来存储从数据库重读取的数据块(表、索引、簇等),因此采用 least recently used (LRU,最近最少使用)的方法进行空间管理。

2、字典缓冲区。该缓冲区内的信息包括用户账号数据、数据文件名、段名、盘区位置、表说明和权限，它也采用 LRU 方式管理。

3、重做日志缓冲区。该缓冲区保存为数据库恢复过程中用于前滚操作。

4、SQL 共享池。保存执行计划和运行数据库的 SQL 语句的语法分析树。也采用 LRU 算法管理。如果设置过小，语句将被连续不断地再装入到库缓存，影响系统性能。

三、规范与反规范设计数据库

1.规范化

所谓规范化实质上就是概念的单一化。数据库中数据规范化的优点是减少了数据冗余，节约了存储空间，相应逻辑和物理的 I/O 次数减少，同时加快了增、删、改的速度。

2.反规范化

在数据库的设计过程中有时故意保留非规范化约束，或者规范化以后又反规范，这样做通常是为了改进数据库的查询性能，加快数据库系统的响应速度。

3.数据库设计中的优化策略

数据应当按两种类别进行组织：频繁访问的数据和频繁修改的数据。比较复杂的方法是将规范化的表作为逻辑数据库设计的基础，然后再根据整个应用系统的需要，物理地非规范化数据。规范与反规范都是建立在实际的操作基础之上的约束，脱离了实际两者都没有意义。只有把两者合理地结合在一起，才能相互补充，发挥各自的优点。

四、合理设计和管理表

1、利用表分区

分区将数据在物理上分隔开，不同分区的数据可以制定保存在处于不同磁盘上的数据文件里。这样，当对这个表进行查询时，只需要在表分区中进行扫描，而不必进行全表扫描，明显缩短了查询时间，另外处于不同磁盘的分区也将对这个表的数据传输分散在不同的磁盘I/O，一个精心设置的分区可以将数据传输对磁盘I/O竞争均匀地分散开。

2、避免出现行连接和行迁移

在建立表时，由于参数pctfree和pctused不正确的设置，数据块中的数据会出现行链接和行迁移，也就是同一行的数据不保存在同一的数据块中。因此,在创建表时，就应该充分估计到将来可能出现的数据变化，正确地设置这两个参数，尽量减少数据库中出现行链接和行迁移。

3、别名的使用

别名是大型数据库的应用技巧，就是表名、列名在查询中以一个字母为别名，查询速度要比建连接表快1.5倍。

五、索引Index的优化设计

1、管理组织索引

索引可以大大加快数据库的查询速度，索引把表中的逻辑值映射到安全的RowID，因此索引能进行快速定位数据的物理地址。对一个建有索引的大型表的查询时，索引数据可能会用完所有的数据块缓存空间，ORACLE不得不频繁地进行磁盘读写来获取数据，因此在对一个大型表进行分区之后，可以根据相应的分区建立分区索引。

2、聚簇的使用

Oracle提供了另一种方法来提高查询速度，就是聚簇（Cluster）。聚簇根据共同码值将多个表的数据存储在同一个Oracle块中，这时检索一组Oracle块就同时得到两个表的数据，这样就可以减少需要存储的Oracle块，从而提高应用程序的性能。

3、优化设置的索引，就必须充分利用才能加快数据库访问速度。

ORACLE 要使用一个索引，有一些最基本的条件：1）where 子名中的这个字段，必须是复合索引的第一个字段；2）where 子名中的这个字段，不应该参与任何形式的计算。

六、多 CPU 和并行查询 PQO(Parallel Query Option)方式的利用

1.尽量利用多个 CPU 处理器来执行事务处理和查询

CPU 的快速发展使得 ORACLE 越来越重视对多 CPU 的并行技术的应用，一个数据库的访问工作可以用多个 CPU 相互配合来完成，加上分布式计算已经相当普遍，只要可能，应该将数据库服务器和应用程序的 CPU 请求分开，或将 CPU 请求从一个服务器移到另一个服务器。对于多 CPU 系统尽量采用 Parallel Query Option(PQO,并行查询选项)方式进行数据库操作。

2.使用 Parallel Query

Option(PQO,并行查询选择)方式进行数据查询.使用 PQO 方式不仅可以在多个 CPU 间分配 SQL 语句的请求处理，当所查询的数据处于不同的磁盘时，一个个独立的进程可以同时进行数据读取。

3.使用 SQL*Loader Direct Path 选项进行大量数据装载

使用该方法进行数据装载时，程序创建格式化数据块直接写入数据文件中，不要求数据库内核的其他 I/O。

七、实施系统资源管理分配计划

ORACLE 提供了 Database Resource Manager（DRM,数据库资源管理器）来控制用户的资源分配，DBA 可以用它分配用户类和作业类的系统资源百分比。在一个 OLDP 系统中，可给联机用户分配 75%的 CPU 资源，剩下的 25%留给批用户。另外，还可以进行 CPU 的多级分配。除了进行 CPU 资源分配外，DRM 还可以对资源用户组执行并行操作的限制。

八、使用最优的数据库连接和 SQL 优化方案

1.使用直接的 OLE DB 数据库连接方式。

通过 ADO 可以使用两种方式连接数据库，一种是传统的 ODBC 方式，一种是 OLE DB 方式。ADO 是建立在 OLE DB 技术上的，为了支持 ODBC，必须建立相应的 OLE DB 到 ODBC 的调用转换，而使用直接的 OLE DB 方式则不需转换，从而提高处理速度。

2.使用 Connection Pool 机制

在数据库处理中，资源花销最大的是建立数据库连接，而且用户还会有一个较长的连接等待时间。解决的办法就是复用现有的 Connection，也就是使用 Connection Pool 对象机制。Connection Pool 的原理是：IIS+ASP 体系中维持了一个连接缓冲池，这样，当下一个用户访问时，直接在连接缓冲池中取得一个数据库连接，而不需重新连接数据库，因此可以大大地提高系统的响应速度。

3.高效地进行 SQL 语句设计

通常情况下，可以采用下面的方法优化 SQL 对数据操作的表现：

（1）减少对数据库的查询次数，即减少对系统资源的请求，使用快照和显形图等分布式数据库对象可以减少对数据库的查询次数。

（2）尽量使用相同的或非常类似的 SQL 语句进行查询，这样不仅充分利用 SQL 共享池中的已经分析的语法树，要查询的数据在 SGA 中命中的可能性也会大大增加。

（3）避免不带任何条件的 SQL 语句的执行。没有任何条件的 SQL 语句在执行时，通常要进行 FTS，数据库先定位一个数据块，然后按顺序依次查找其它数据，对于大型表这将是一个漫长的过程。

（4）如果对有些表中的数据有约束，最好在建表的 SQL 语句用描述完整性来实现，而不是用 SQL 程序中实现。

九、充分利用数据的后台处理方案减少网络流量

1.数据库打包技术的充分利用

利用数据库描述语言编写数据库的过程或函数,然后把过程或函数打成包在数据库后台统一运行包即可。

2.数据复制、快照、视图，远程过程调用技术的运用

数据复制，即将数据一次复制到本地，这样以后的查询就使用本地数据，但是只适合那些变化不大的数据。使用快照也可以在分布式数据库之间动态复制数据，定义快照的自动刷新时间或手工刷新，以保证数据的引用参照完整性。调用远程过程也会大大减少因频繁的 SQL 语句调用而带来的网络拥挤。总之,对所有的性能问题,没有一个统一的解决方法,但 ORACLE 提供了丰富的选择环境，可以从 ORACLE 数据库的体系结构、软件结构、模式对象以及具体的业务和技术实现出发，进行统筹考虑。

ORACLE 提供了丰富的选择环境，可以从 ORACLE 数据库的体系结构、软件结构、模式对象以及具体的业务和技术实现出发，进行统筹考虑。提高系统性能需要一种系统的整体的方法，在对数据库进行优化时，应对应用程序、I/O 子系统和操作系统（OS）进行相应的优化。优化是有目的地更改系统的一个或多个组件，使其满足一个或多个目标的过程。对 Oracle 来说，优化是进行有目的的调整组件级以改善性能，即增加吞吐量，减少响应时间。如果 DBA 能从上述九个方面综合考虑优化方案，相信多数 ORACLE 应用可以做到按最优的方式来存取数据。

我们要做到不但会写 SQL,还要做到写出性能优良的 SQL,以下为笔者学习、摘录、并汇总部分资料与大家分享！

（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.

```
?
1decode (expression, search_1, result_1)
2decode (expression, search_1, result_1, search_2, result_2)
3decode (expression, search_1, result_1, search_2, result_2, ....,
4search_n, result_n)
5
6decode (expression, search_1, result_1, default)
7decode (expression, search_1, result_1, search_2, result_2, default)
 decode (expression, search_1, result_1, search_2, result_2, ....,
 search_n, result_n, default)
```

decode 函数比较表达式和搜索字，如果匹配，返回结果；如果不匹配，返回 default 值；如果未定义 default 值，则返回空值。

以下是一个简单测试，用于说明 Decode 函数的用法:

```
?
 1SQL> create table t as select username,default_tablespace,lock_date from
 2dba_users;
 3
```

```
Table created.

SQL> select * from t;
USERNAME                       DEFAULT_TABLESPACE             LOCK_DATE
------------------------------ ------------------------------ ---------
SYS                             SYSTEM
SYSTEM                         SYSTEM
OUTLN                           SYSTEM
CSMIG                           SYSTEM
SCOTT                           SYSTEM
EYGLE                           USERS
DBSNMP                         SYSTEM
WMSYS                           SYSTEM                          20-OCT-04

8 rows selected.


SQL> select username,decode(lock_date,null,'unlocked','locked') status
from t;

USERNAME                       STATUS
------------------------------ --------
SYS                             unlocked
SYSTEM                         unlocked
OUTLN                           unlocked
CSMIG                           unlocked
SCOTT                           unlocked
EYGLE                           unlocked
DBSNMP                         unlocked
WMSYS                           locked

8 rows selected.

SQL> select username,decode(lock_date,null,'unlocked') status from t;

USERNAME                       STATUS
------------------------------ --------
SYS                             unlocked
SYSTEM                         unlocked
OUTLN                           unlocked
CSMIG                           unlocked
SCOTT                           unlocked
EYGLE                           unlocked
DBSNMP                         unlocked
```

```
WMSYS
8 rows selected.
```

（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：

```
?
1DELETE  FROM  EMP E  WHERE  E.ROWID > (SELECT MIN(X.ROWID)
2FROM  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：

```
?
1                SELECT  TAB_NAME FROM TABLES WHERE (TAB_NAME,DB_VER) =
2( 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：

```
?
1（高效）SELECT * FROM  EMP (基础表)  WHERE  EMPNO > 0  AND  EXISTS (SELECT
21  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：

```
?
1SELECT  EXECUTIONS , DISK_READS, BUFFER_GETS,
2ROUND((BUFFER_GETS-DISK_READS)/BUFFER_GETS,2) Hit_radio,
3ROUND(DISK_READS/EXECUTIONS,2) Reads_per_run,
4SQL_TEXT
5FROM  V$SQLAREA
6WHERE  EXECUTIONS>0
7AND  BUFFER_GETS > 0
8AND  (BUFFER_GETS-DISK_READS)/BUFFER_GETS < 0.8
9ORDER 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.：在“系统维护清理”里有个“垃圾文件清理”

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：

```
?
1      (低效):
2SELECT  DISTINCT  DEPT_NO,DEPT_NAME  FROM  DEPT D , EMP E
3WHERE  D.DEPT_NO = E.DEPT_NO
4(高效):
5SELECT  DEPT_NO,DEPT_NAME  FROM  DEPT D  WHERE  EXISTS ( SELECT ‘X'
```

```
6FROM  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:

```
?
1低效:
2SELECT … FROM  DEPT  WHERE SAL * 12 > 25000;
3高效:
4SELECT … FROM DEPT WHERE SAL > 25000/12;
```

（23） 用>=替代>

```
?
1高效:
2SELECT * FROM  EMP  WHERE  DEPTNO >=4
3低效:
4SELECT * 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.

```
?
 1高效:
 2SELECT LOC_ID , LOC_DESC , REGION
 3FROM LOCATION
 4WHERE LOC_ID = 10
 5UNION
 6SELECT 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

```
?
1低效:
2SELECT  ACCT_NUM, BALANCE_AMT
3FROM  DEBIT_TRANSACTIONS
4WHERE TRAN_DATE = '31-DEC-95'
5UNION
6SELECT ACCT_NUM, BALANCE_AMT
7FROM DEBIT_TRANSACTIONS
8WHERE TRAN_DATE = '31-DEC-95'
9高效:
10SELECT ACCT_NUM, BALANCE_AMT
11FROM DEBIT_TRANSACTIONS
12WHERE TRAN_DATE = '31-DEC-95'
13UNION ALL
14SELECT ACCT_NUM, BALANCE_AMT
15FROM DEBIT_TRANSACTIONS
16WHERE 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:

```
?
1表DEPT包含以下列:
2DEPT_CODE PK NOT NULL
3DEPT_DESC NOT NULL
4DEPT_TYPE NULL
5低效: (索引不被使用)
6SELECT DEPT_CODE FROM  DEPT  ORDER BY  DEPT_TYPE
7高效: (使用索引)
8SELECT 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 WHERE TO_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.

```
?
1低效:
2SELECT JOB , AVG(SAL)
3FROM EMP
4GROUP JOB
5HAVING JOB = ‘PRESIDENT'
6OR JOB = ‘MANAGER'
7高效:
8SELECT JOB , AVG(SAL)
9FROM EMP
10WHERE JOB = ‘PRESIDENT'
11OR JOB = ‘MANAGER'
12GROUP JOB
```