总原则：做最少的事；该做的事一件都不少，但让更多的资源来做。

001、CPU占用率高，大量sql语句的parse操作。

alter system set cursor\_sharing = ‘SIMILAR’

原因在于应用程序大量拼接sql，应尽量使用变量绑定，提高sql共享。

002、SQL\*Net more data from client，数据库吞吐量低。

01：网络瓶颈；02：应用服务器资源瓶颈，无法及时向数据库发送服务请求。

003、常见问题和观点。

系统参数一定要调，还要合理地调，但调好了并不一定能解决所有问题。

应用系统的问题无法交给DBA处理。

开发过程中就应该考虑到性能问题。

优化sql语句的性能，并不都是改语句。

多表连接性能太差是个错误的观点。

CPU占率用越低越好是错误的，可能是因为人为造成的堵塞造成的。

内存操作不一定最快，CPU占用无法忽视。

业务规则优化、数据库体系结构优化、数据库逻辑结构设计、数据库物理结构设计、应用系统设计等。合理的索引策略，合理地访问路径，优化SQL语句的分析数，充分使用PL/SQL，减少锁冲突等。



先拿最耗资源的SQL和模块动手。

IT系统一般分为OLTP(联机事务处理)和DSS(Decision Support System，决策分析系统，即数据仓库)。前者优化应当以响应速度为主，后者则应当以整体吞吐量为主。随着报表的增多，现在已逐渐都是混合型的系统，白天OLTP，晚上DSS。

004、sql执行计划分析：

1、表的访问方式；2、表的索引类型和策略；3、表的连接类型和过程；4、排序过程；

5、汇总过程；6、并行处理过程

量化指标：

1、消耗时间：包括Elapsed Time、CPU Time等；

2、内存消耗：包括Buffer Gets、Consistant Gets等；

3、I/O消耗：包括Physical Reads、Physical Writes等；

4、语句分析次数：包括Parses、Hard Parses、Soft Parses等

005、常用工具

explain，这个不解释了。

sql\*trace+TKPROF，前者生成的文件交由后者解析成可读文件。设置方法：

alter session set sql\_trace = true;

execute dbms\_session.set\_sql\_trace(true);

execute dbms\_system.set\_sql\_trace\_in\_session(session\_id,serial\_id,true);session\_id和serial\_id通过其他视图查询得到。

AutoTrace：

打开：set autotrace on; autotrace的执行计划是真正执行后分析的，而explain并没有执行sql。

统计信息重点看consistent gets和physical reads，分别代表内存消耗和磁盘I/O消耗，单位是db\_block\_size大小。

以上都在命令行模式下进行。

006、基本索引

基本概念：(1) rowid，索引的门牌号；(2) 索引就是目录

B\*树索引单字段设计：

1、分析sql语句中的约束条件

2、如果约束条件不固定，则建议创建针对单字段的B\*树索引

3、选择可选性最高的字段建立索引(即不同值最多的字段)

4、如果是多表连接的sql语句，注意被驱动表的连接字段是否需要创建索引

5、通过分析工具分析执行计划，量化执行效果

驱动表：在RBO(基于规则的优化器)中，from后的最后一张表是驱动表；CBO中，输出量最少的是驱动表。

索引被抑制：尽可能不对查询字段进行函数、表达式处理，自定义函数也是函数！函数索引应尽量少用，原因有二，一是影响DML效率，每次执行时都会重新计算；二是增加了索引存储空间。

复合索引(作用在多字段上):

1、前缀性(Prefixing)。比如索引建立在(empNo,empName)上，则查询条件必须包含empNo才能使用索引。这是因为索引是经过排序的，并非智能，此处就使用了empNo+empName排序。

关于skip scan index。这种是因为对索引进行了拆分。如(gender,ename,job)被拆分为(‘男’,ename,job)和(‘女’,ename,job)。这种索引违反了索引的可选性规则。

2、可选性(Selectivity)。索引字段的排列顺序，按值的多少排列。原因：值越多越容易定位。

设计原则：

如果查询条件涉及的字段不固定，组合十分灵活，则分别建立单字段索引；反之，则考虑建立复合索引，要注意前缀性和可选性。

索引监控：

alter index <indexname> monitoring usage; --打开

alter index <indexname> nomonitoring usage; --关闭

通过查询v$Object\_Usage视图判断索引是否使用。

索引碎片：

视图index\_stat中包含索引信息。索引重建、索引压缩，建议使用索引重建。

alter index <indexname> rebuild;

007、语句共享(变量绑定)

sql语句的主要执行过程：

1、Parse(解析)。在share pool(共享池)中查找该sql，如找不到，则进行硬解析(hard parse)，否则进行软解析(soft parse)，要尽可能地让oracle进行软解析。解析结果包括执行计划等。

2、Bind(绑定)。如果该语句中有变量，则通过赋值或传参进行变量绑定。

3、Execute(执行)。根据步骤1执行语句。

4、Fetch(抓取)。只针对select语句，进行查询记录的读取和排序。

OLTP和OLAP：

OLTP优化parse阶段，OLAP优化Execute和Fetch阶段，这跟它们的应用特性有关。

parse优化：(70%以上)

参见001.但是这项工作更应该交给开发人员，在应用程序中进行。如果数据畸形分布，001的解决方式不会最优。

查找可共享的sql语句：

select sql\_text from v$sqlarea where executions = 1;

008、排序、表连接

排序：在PGA(属于内存)中排序、能不排序就不排序。

除了显式的order by，distinct、union都隐含了排序。对于group by，10G之后，由于采用了新的算法，不再保证结果集排序。

执行计划里，sort \*\*\*表示进行了排序；另外就是利用索引，如果查询条件全在索引里，则会使用索引的排序。

Top-N：select \* from (select \* from emp order by salary desc) where rownum <11;直排了前10个。

表连接：(不管有多少表，一次都只连接两个表，因此掌握两表连接再扩展即可)

Nested Loop(嵌套循环)：

select e.\*,d.\* from emp e,dept d where e.deptId = d.id

执行：先循环dept表(假设它被作为驱动表，外表)，再按dept每条记录去查询emp表(被驱动表、内表)。

Sort Merge(排序合并)：

两表先按连接字段进行排序，再将排序结果进行顺序匹配，将合并结果返回。

Hash Join：

使用全表扫描，将两表按连接条件划分成足够多的区。每一对匹配区都在内存中建立一个hash表。其他区用于探测这些hash表。

其中，sort merge和hash适合用于大表，且hash效率更高(同样高于nested loop)，但hash比较耗内存。

优化思路：

OLTP应用：(1)把限制性最强的表作为驱动表，限制性条件字段上要有索引。(2)每次连接之后保持记录数最小，传递给下一次连接。(3)每次连接基本使用嵌套循环。(4)尽量通过被驱动表上的索引访问被驱动表。(5)如果被驱动表上还有其他限制性条件，考虑复合索引。(6)全表扫描也许是合理的。比如一些小表和代码表。(7)依次类推。

OLAP：hash、sort merge

子查询：能不用，就不用，因为parse后还是表连接，白白增加parse时间，而且当嵌套比较多时，可能得不到合适的执行路径。

in和exist：(切忌一概而论)

in的原理：先查子查询，再查主查询。

exist原理：先查主查询，再到子查询中匹配。

限制性强的条件在子查询，用in；在主查询，用exists

009、常见问题

1、相同含义的字段在不同表中数据类型不同，导致索引不可用，如外键。

2、不合理的全表扫描。v$session\_longops视图记录运行时长超过6s的sql语句。

3、存储部署不合理，ASM(自动存储管理)改进。

4、过量的排序操作。

5、优化器和统计信息问题。由于没有即时统计，生产环境选择了错误的执行路径。

6、大量递归的SQL语句。

7、Redo Log设计不合理。

010、**分区**

**思想：分而治之。**

oracle分区表简单来说就是将一张大表按一定规则划分为物理上的很多小表，逻辑上仍然是一张大表。

删除某分区：alter table <tableName> truncate <分区名>，几乎瞬间清除，无碎片，很少的日志。

概述：

range(范围)分区，按一个或多个字段的值范围进行分区。

举例：

create table sales

(acct\_no number(5),

week\_no number(2),

…)

partition by range (week\_no)

(partition p1 values less than (4) tablespace data0, -- 指定单独表空间

partition p2 values less than (8) tablespace data1,

…

partition p13 values less than (53) tablespace data13);

范围分区适合按时间周期进行数据存储的表。缺点：数据可能分部不均匀；分区跟字段值联系太紧，如果字段值分部太广，将无法均匀分区；如果字段值经常变化，则数据会在不同分区之间频繁转移。

hash分区，按字段值的hash值分区。

举例：

create table customer(….)

partition by hash(customer\_no) partitions 8 store in (data01,data02)

意思是将该表分为8个区，轮询分配在data01和data02表空间上，如1和3分区在data01上，2和4分区在data02上。建议分区数为2的幂。

hash分区适合于那些不需要进行历史数据迁移的表，因为数据分布在哪个区是oracle决定的。大批量导入导出也不适合hash分区。

列表(list)分区，对分区字段的离散值进行分区，即枚举分区。

举例：

create table address(…)

partition by list (city\_name)

(partition p1 values (‘beijing’) tablespace data01,

…

partition pN values (‘guangzhou’) tablespace dataN);

列表分区不排序，分区之间没有关联。只支持单个字段。优缺点和范围分区一样。

组合分区

范围-Hash，范围-列表：语法举例：

create table (…)

partition by range (field1)

subpartition by hash (field2)

subpartition template

(分区条件)(分区条件)。

11g新增：间隔分区(interval)、更多的组合分区(范范、范列、范哈、列列、列范、间列、间范、间哈)、基于虚拟列的分区、引用分区(主从表)、系统分区

问题：count怎么优化，本地前缀分区索引，分区表的表连接

175