**Oracle Advanced Product Services**

建信人寿

Oracle现场服务报告

保单系统应用开发规范和SQL量化指标

Advanced Customer Support



文档

**文档控制**

此文档仅供建信人寿与 Oracle公司支持服务部审阅。

此文档仅供建信人寿与Oracle公司支持服务部审阅，不得向与此无关的个人或机构传阅或复制。

### 修改记录

| 日期 | 作者 | 版本 | 修改记录 |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### 分发者

| 姓名 | 职位 |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

### 审阅记录

| 姓名 | 职位 |
| --- | --- |
|  |  |
|  |  |
|  |  |

# 开发建议

## 应用设计开发总体规范

### 应用设计开发重要性

一般而言，大多数最终用户和系统开发人员在系统上线，大量业务开始运行，出现反应速度慢之后，才会意识到系统优化的重要性。此时，对优化而言，已不是最佳时期。如前所述，系统优化应是包括系统设计、开发、产品上线、平台优化等的全过程，不同阶段的优化工作对全系统所带来的效益是不同的。下图是Oracle公司总结的不同阶段进行优化工作所带来的效益和代价变化图：

可见，在设计和开发阶段就全面考虑系统优化问题，带来的效益越明显，代价也越小。反之，效益更低，代价更大。

其中设计阶段包括的与优化相关的工作如下：

* 业务规则的优化设计
* 数据库结构的优化设计
* 应用系统的优化设计

开发阶段与优化相关的工作主要包括：

* 数据库操作方式的优化和调整
* 访问路径的优化和调整

产品阶段能优化的工作主要与数据库系统和操作系统平台相关，包括：

* 内存和CPU的优化和调整
* 磁盘I/O和物理结构的设计和调整
* 系统资源竞争的监测和调整
* 操作系统平台的优化

本文档将重点在设计开发阶段的SQL编程规范方面提出建议。

### 20/80规则

如同在IT的众多领域所呈现的规律一样，在涉及数据库系统质量特别是性能问题上，也体现了一种20/80定律。根据Oracle公司经验，这种20/80定律体现在如下方面：

* 应用设计对系统性能的影响能占到80%，而数据库体系结构的设计、数据库系统参数调整、操作系统参数的调整等系统方面因素，只占到20%。
* 80%的性能问题是由20%的应用导致的。如少量大表的全表扫描导致的性能瓶颈。并不是应用一有问题，就一定要对现有数据库结构大卸八块，应用推倒重来。
* 80%的性能问题可以由20%的优化技术所解决。如简单的索引策略，执行路径分析等，能解决绝大部分性能问题。

### 应用开发指导思想

建议应用系统设计和开发人员在开发过程中，在开发指导思想上进行如下方面的加强：

* 不仅关注SQL语句功能，而且要关注性能。即用量化手段，进行SQL语句质量控制。
* 开发队伍能有层次性和专业分工。不仅按照业务模块分工，而且有专门的质量控制，尤其是SQL质量控制人员。
* 加强软件开发的规范管理
* 注重知识共享和传递，减少低级错误的重复性。
* 强调实际测试的重要性。切忌想当然的主观推断，一切以实际应用在尽可能真实数据和环境下的测试数据为准。

### 编程方式建议

建议开发人员在编程方式上采取如下策略：

* 尽量采用PL/SQL的存储过程、Package等方式实现主要计算逻辑。这样将充分发挥Oracle数据库服务器的处理能力，减少网络传输，同时也提高应用系统的封装性、可移植性。
* 充分发挥Oracle数据库的各种统计运算计算能力，避免在应用层进行复杂的统计运算。一方面减轻应用开发工作量，另一方面能充分利用Oracle的现成的统计运算功能，以及并行处理能力。

总之，建议不仅把Oracle当成存储和管理数据的服务器，而且也充分发挥Oracle数据库服务器内置的强大处理能力，

### 联机业务和批处理业务系统

联机业务和批处理业务无论在业务操作特征，还是在技术运用方面均有很大区别：

|  | 比较项目 | 联机业务 | 批处理业务 |
| --- | --- | --- | --- |
|  |  |  |  |
| 业务特征 | 操作特点 | 日常业务操作，尤其是包含大量前台操作 | 后台操作，例如统计报表、大批量数据加载。单次执行访问和返回记录多。 |
| 响应速度 | 优先级最高，要求反应速度非常高 | 要求速度高、吞吐量大 |
| 吞吐量 | 小 | 大 |
| 并发访问量 | 非常高 | 不高 |
| 单笔事务的资源消耗 | 小 | 大 |
| SQL语句类型 | 主要是单条或者少量插入和修改操作（DML） | 主要是大量查询操作或批量DML操作 |
| 技术运用 | 索引类型 | B\*索引 | 全表扫描，全分区扫描 |
| 索引量 | 适量 | 多 |
| 访问方式 | 按索引访问 | 全表扫描 |
| 连接方式 | Nested\_loop | Hash Join |
| BIND变量 | 使用或强制使用 | 不使用 |
| 并行技术 | 使用不多 | 大量使用 |
| 分区技术 | 使用，但目标不同 | 使用，但目标不同 |
| 物化视图使用 | 少量使用 | 大量使用 |

大多数系统是联机业务和批处理相结合的混合系统。例如白天主要是联机业务，夜间主要是批处理业务。因此，首先建议对业务系统进行联机业务和批处理业务的特征分析，并针对不同业务模块，提出针对性的技术运用策略。

下面将在应用设计开发的详细规范和策略上，对上述相关技术的应用进行详细描述。

## 应用开发详细规范和建议

### Oracle数据库中语句的执行过程



### 合理使用绑定变量和硬编码值

通常对于OLTP系统，语句执行非常频繁，但是每次执行耗时很短的语句，强烈建使用绑定变量，实现软解析，以减少语句的解析.

对于报表或者批量语句，语句执行不多，但是每次执行需要花费很多资源的语句，可以考虑使用硬编码值，消耗一定的解析时间，但是可能得到更好的执行计划，较少语句的执行时间。

### 使用拼接语句可能引入的SQL注入风险

对于使用硬编码，拼接后生成的语句，需要特别SQL注入的风险，应用程序应该在检查输入值的过程中，就检查用户数字值中是否包含SQL注入攻击的情况。

### 应用不要使用oracle关键字

应用对象，变量等不要使用ORACLE 的关键字和保留字

Oracle的关键字和保留字可以参见视图V$RESERVED\_WORDS。

### 大小写

对象大小写风格保持一致

### 缩进和对齐

建议缩进和对齐，以空格而不是tab键对齐，因为在不同地方，tab键的位置不一定完全一致。

### 选用适合的ORACLE优化器

在oracle 19c中，推荐使用CBO，rule已经是oracle不支持的优化器了

### 访问路径的选择

索引和全表扫描的选择，哪种方法更优，主要取决于返回记录和全表记录的比例，以及索引和表数据的对应关系。

通常的测试经验，对于未按照索引列的进行排序的表，返回结果集在10%以内，索引才优于全表扫描。

所以索引通常更加适用于返回结果占比很小的OLTP系统。

而全表扫描则更加适合需要访问大量数据的批量和报表系统。

### 尽可能不要在列上使用函数

在列上做进行函数计算，会导致无法使用索引

错误写法:

where to\_char(insert\_time, 'yyyymmdd') >= '20170508'

正确写法:

where insert\_time >= to\_date('20170508','yyyymmdd')

错误写法:

where to\_char(insert\_time, 'yyyymmdd') = '20170508'

正确写法:

where insert\_time >= to\_date('20170508','yyyymmdd')

and insert\_time < to\_date('20170509','yyyymmdd');

### 不要在列上进行运算

错误写法:

insert\_time +7 >= sysdate

正确写法:

insert\_time >= sysdate -7

### 带通配符（%）的like语句

%在硬编码值前面索引就不会使用。

不使用索引

Select \* from ac01 where aac002 like '%210104';

使用索引

Select \* from ac01 where aac002 like '210104%';

### 尽可能避免在索引列上使用NOT

避免在索引列上使用NOT, NOT不会使查询条件使用索引。对于!=这样的判断也是不能使用索引的，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

低效: (不使用索引)   
select \*

From ac02

Where not aab019='10';

高效: (这里,使用了索引)

select \*

From ac02

Where aab019 in ('20','30');

### 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.

### like替换substr

使用一些场景下，比如字符类型的查询列上，前几个字符的过滤性很强，like替换substr能够有效利用索引.

（低效)

substr(name,1,4)='1234'

（高效）

等价写法： name like '1234%'

### SELECT子句中避免使用\*，指定列名

避免select \*可能引发的问题：

* 将访问所有的列，增加了数据库服务器和应用服务器之间的通讯流量
* 如果日后出现数据库添加字段，老的应用程序将出错。
* 如果发生删除再添加字段，甚至可能导致数据库和应用程序的列的对应关系的错位。

### 在INSERT SQL语句中指定列名

使用

INSERT INTO t (col1, col2, ...) VALUES (:v1, :v2,... )

而不是使用

INSERT INTO t VALUES (:v1, :v2)

避免日后出现数据库添加字段的表结构发生变化，导致语句报错，或者插入的值和列的对应关系发生错位

### 多表连接查询,列名前加上表的别名

* 多表连接情况下，被引用的列，前面加上表的别名以明确列和表的对应关系
* 可以减少解析时间。

### 减少访问数据库的次数--一次取多列

如果某个业务逻辑，需要获取多个字段，建议一次性获取，而不是每次获取一个字段

### 减少访问数据库的次数—一次取多行

如果某个批量，需要访问大量记录，建议以批量方式获取数据，而不是通过cursor逐条获取

### 避免隐式转换

变量或值的数据类型要与列的数据类型一致，避免数据发生隐式转换，否则可能导致无法使用索引

更加严重的情况，可能出现返回结果和预期不一致问题，详细情况参见后续章节

### char和varchar2之间的比较规则

create table t

( char\_column char(20),

varchar2\_column varchar2(20)

)

/

insert into t values ( 'Hello World', 'Hello World' );

1 row created.

select \* from t;

CHAR\_COLUMN VARCHAR2\_COLUMN

-------------------- --------------------

Hello World Hello World

#对于硬编码值，尾部空格被忽略，认为相等

select \* from t where char\_column = 'Hello World';

CHAR\_COLUMN VARCHAR2\_COLUMN

-------------------- --------------------

Hello World Hello World

select \* from t where varchar2\_column = 'Hello World';

CHAR\_COLUMN VARCHAR2\_COLUMN

-------------------- --------------------

Hello World Hello World

#对于列比较，尾部空格个数不一样会认为不等

select \* from t where char\_column = varchar2\_column;

no rows selected

#对于列比较，通过列上做转换，认为转换后相等

select \* from t where rtrim(char\_column) = varchar2\_column;

CHAR\_COLUMN VARCHAR2\_COLUMN

-------------------- --------------------

Hello World Hello World

select \* from t where char\_column = rpad( varchar2\_column, 20 );

CHAR\_COLUMN VARCHAR2\_COLUMN

-------------------- --------------------

Hello World Hello World

#通过绑定变量赋值

variable varchar2\_bv varchar2(20)

exec :varchar2\_bv := 'Hello World';

PL/SQL procedure successfully completed.

#通过绑定变量赋值，变量数据类型和列数据类型一致，认为不相等

select \* from t where char\_column = :varchar2\_bv;

no rows selected

#通过绑定变量赋值，变量数据类型和列数据类型一致，认为相等

select \* from t where varchar2\_column = :varchar2\_bv;

CHAR\_COLUMN VARCHAR2\_COLUMN

-------------------- --------------------

Hello World Hello World

#通过绑定变量赋值

variable char\_bv char(20)

exec :char\_bv := 'Hello World';

PL/SQL procedure successfully completed.

#通过绑定变量赋值，变量数据类型和列数据类型一致，认为相等

select \* from t where char\_column = :char\_bv;

CHAR\_COLUMN VARCHAR2\_COLUMN

-------------------- --------------------

Hello World Hello World

#通过绑定变量赋值，变量数据类型和列数据类型一致，认为不相等

select \* from t where varchar2\_column = :char\_bv;

no rows selected

### 避免不必要的排序操作

创建索引，考虑应用的排序要求的时候，可以考虑结合应用的排序操作，合理设置索引列的顺序

查询结果集不要求去除重复值时，考虑union all代替union。

### 全表的所有数据的删除

* 从性能角度考虑，删除一个表中的所有数据时，建议使用truncate。
* 并且truncate会降低相关表的高水位表，实现空间的真正释放
* 但是truncate是DDL语句，不能回滚，并且如果在一个事务中，将隐式提交前面的DML操作。
* 务必区分生产和开发环境，避免误操作

### OLTP事务的加锁顺序和commit的频率

* 如果多个事务，需要对多张表进行DML操作，建议以相同的顺序对表进行操作，以减少和避免锁等待以及死锁的情况
* 如果是OLTP程序，及时commit，避免锁占用时间过长。
* 锁定记录的3条原则：

尽可能晚地获取锁资源

尽可能锁定少的记录

尽可能快地提交记录

* 在获得锁资源后，提交前尽可能减少和避免过长的查询，保存结果到外部等操作
* 公用的、重要的、稀缺的资源较一般资源而言，更加应该尽可能晚申请尽可能早释放

### DDL语句考虑online

一些操作，比如给表上创建索引操作，为了避免添加索引过程中，长时间锁表，请合理使用online。

### 避免在业务高峰期，进行DDL操作和对象编译

尽量避免在业务高峰期，进行DDL操作比如alter,drop,truncate,grant,revoke,debug,analyze，和对象编译，以及统计信息采集等会导致大量解析，应该避免这些操作导致正常业务受影响。

### 大批量的DML操作的commit的数据量

批量处理过程中commit数据量适中

–数据量太小，比如每条都提交，次数太多会降低效率

–数据量过大，积累过多的事务不commit，Rollback代价也高，也可能导致锁占用时间过长

比如可以考虑1万条到10万记录提交一次

### 单条语句影响行数的控制

通常情况下，控制对每个SQL都需要确认不会影响超过100万记录

### 禁止无过滤条件的update语句

除非业务逻辑是更新表上所有行记录，不得在大表（超过1千万条记录的表，下同）上执行无过滤条件的UPDATE语句。

### UPDATE应仅针对有修改的字段进行

UPDATE操作时，应该做到仅对有修改的字段UPDATE，避免对表中无修改的字段也一并更新，后者的做法将耗费更多的CPU、内存、REDO，如果有LOB字段，还会消耗更多的LOB字段，容易造成LOB相关的等待事件。

同时更新更多的字段也容易因为绑定变量长度的不同造成需要生成不同的子游标，增加了解析工作量。

### 控制in语句的硬编码和绑定变量的值个数

一些语句，因为是拼接出来的，导致in里面的值的列表达到几百甚至数千个，导致相关语句解析出现性能问题，并且占用很大的shared pool。in的值应该控制在50个以内，如果超过50个，应该考虑存放在表中，通过列关联的方式进行查询。

### 访问其它用户中的对象时，建议通过用同义词访问而不是用户名.对象名进行访问

访问其它用户中的对象时，建议通过用同义词访问而不是用户名.对象名进行访问

因为一旦数据库用户名改变了，需要调整所有涉及的代码，一方面工作量很大，另外也容易出现疏漏。

如果通过同义词访问，只需要修改同义词和所访问对象的对应关系。

### 关于并行

如果在批量，报表等访问大批量数据的时候，合理使用并行，启用多个进程，可以提升程序的响应速度，成本是占用较多的系统资源。

建议需要并行时，通过hint在SQL语句中，指定并行度，例如

/\*+ parallel(a 8) parallel(b 8) \*/

这里指定访问别名为a和b的表上，分别开8个并行进程。

并行可以用于全表扫描或者索引快速全扫描。

另外在采集统计信息，创建索引等场景。

创建索引的时候，如果指定指定并行，索引创建完成后，请立即修改索引并行度为1。

### 关于翻页查询

对于翻页查询语句，应该合理控制返回记录数，比如每页控制在20-50行记录

联机交易应用查询必须增加分页机制，对返回记录数进行控制，限制最大不超过500条。返回记录数过多，不但造成数据库负载高，极端情况下还可能造成中间件out of memory错误。

应用应该防止客户短时间内反复点击同样的查询，避免过度频繁的查询请求。

### 避免笛卡尔积

一般情况下，不应该笛卡尔积，如果出现笛卡尔积，请检查相关表是否存在关联条件出现疏漏的情况。

### 表关联个数

一般情况下，表关联数不要超过3个。以避免语句生成执行计划耗时过长，另外也避免因为表关联数过多，引起执行计划容易出现变化的情况。如果是报表或者批量，的确有需要多表关联的场景，必要情况下应用可以通过leading hint的指定表的访问顺序。

### exists和in的权衡

如果关联查询，外层表返回结果集少，子查询的表上的查询有高效索引，通常情况下exist的性能会更好

如果关联查询，子查询的表上的查询没有高效索引，并且外层查询满足条件记录多，子查询返回的的记录数较小的情况下，通常情况下in的效率会更好

### not in语句和列的nullalbe定义需要特别关注的地方

对于列的nullable，应该根据实际情况进行设计，并且需要考虑，如果定义为nullable的情况下，和not in查询，返回结果和期望不一致的情况。

另外如果not in查询返回0行记录，not in会返回外表自身带的所有过滤条件的记录！！！

使用not in需要特别谨慎！！！

create table t\_in\_exists1(id number,name varchar2(30));

create table t\_in\_exists2(id number,name varchar2(30));

insert into t\_in\_exists1 values (1,'1');

insert into t\_in\_exists1 values (2,'2');

insert into t\_in\_exists1 values (3,'3');

insert into t\_in\_exists1 values (4,'4');

insert into t\_in\_exists1 values (5,null);

commit;

insert into t\_in\_exists2 values (1,'1');

insert into t\_in\_exists2 values (2,'2');

insert into t\_in\_exists2 values (3,'3');

insert into t\_in\_exists2 values (4,null);

commit;

select \* from t\_in\_exists1;

select \* from t\_in\_exists2;

--想找出t1表中，name在t2表中，没有对应记录的行

期望返回：

ID NAME

---------- ------------------------------------------------------------

5

4 4

--用not exists，返回的结果和期望值一致

select \* from t\_in\_exists1 t1

where not exists(

select 1 from t\_in\_exists2 t2

where t1.name = t2.name)

;

ID NAME

---------- ------------------------------------------------------------

5

4 4

--用not in，返回的结果和期望值不一致

select \* from t\_in\_exists1 t1

where t1.name not in (

select name from t\_in\_exists2 t2 )

;

未选定行

### 关于TOP N查询

oracle 12c中 引入了 row\_limiting\_clause子句，用于加强select的查询功能，简化TOP N查询和分页查询的写法

--取薪水最高的5条记录

SELECT ENO,ENAME,SAL FROM EMP ORDER BY SAL DESC

FETCH FIRST 5 ROWS ONLY;

--取薪水最高的5%的记录

SELECT ENO,ENAME,SAL FROM EMP ORDER BY SAL DESC

FETCH FIRST 5 PERCENT ROWS ONLY;

### 关于对象长度的变化

从12c开始，oracle支持的对象，如用户名，表名，索引名，长度从30字符增加到128字符。

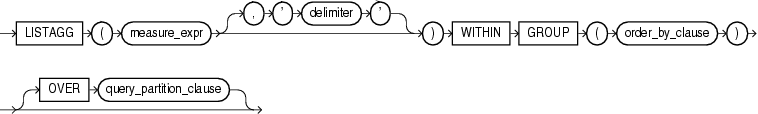
需要强调的是，如果还将相关对象导入到之前的版本，则必须控制在30字符以内

### 行列转换和拼接函数--禁止使用wm\_concat函数

wm\_concat函数是oracle的内部函数，效率差，11G版本后返回clob字段，在19g中已经没有这个函数了，应使用Listagg来代替。

从11.2开始，可以通过LISTAGG函数可以非常方便地实现行列转换和拼接功能。

语法如下

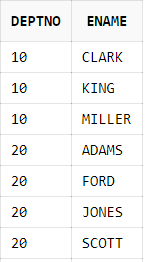


范例：

SELECT deptno, ename

FROM scott.emp

ORDER BY deptno,ename;

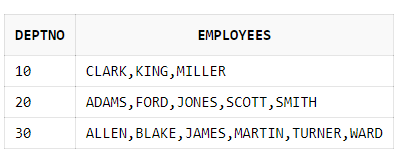


SELECT deptno, LISTAGG(ename, ',') WITHIN GROUP(ORDER BY ename) employees

FROM scott.emp

GROUP BY deptno;

ORDER BY deptno;



12.2以前版本，如果返回结果超过数据类型最大长度，语句会报一下异常ORA-01489: result of string concatenation is too long

从12.2开始，可以通过遇到上述情况，可以通过[ON OVERFLOW TRUNCATE [truncate\_literal]或者ON OVERFLOW ERROR，来指定截短结果还是维持老版本的报错

LISTAGG ( [ALL] <measure\_column> [,<delimiter>] [ON OVERFLOW TRUNCATE

[truncate\_literal] | ON OVERFLOW ERROR [WITH | WITHOUT COUNT]])

WITHIN GROUP (ORDER BY <oby\_expression\_list>)

遇到结果超长报异常ORA-01489，也可以考虑使用XMLAGG函数

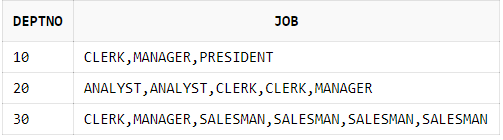
从19c开始，可以通过LISTAGG内的结果进行去除重复值。

SELECT deptno, LISTAGG(JOB, ',') WITHIN GROUP(ORDER BY JOB) JOB

FROM scott.emp

GROUP BY deptno

ORDER BY deptno;

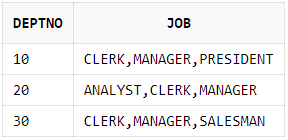


SELECT deptno, LISTAGG(DISTINCT JOB, ',') WITHIN GROUP(ORDER BY JOB) JOB

FROM scott.emp

GROUP BY deptno

ORDER BY deptno;



### 分析函数

Oracle提供了功能非常强大的分析函数，不仅可以简化SQL编写，还可以极大提升相关功能的效率。

### 批量或者报表语句考虑一次性处理大批记录而不是逐条处理

只要业务逻辑上支持，批量或者报表语句，应该尽可能一次性处理大批记录，而不是通过open cursor方式，逐条记录进行处理，可以极大提升批量和报表语句的性能。

### 批量或者报表语句考虑使用存储过程进行处理

通过存储过程处理大批量数据，可以减少数据库服务器和应用服务器的网络通讯次数，从而提升效率

### 游标（Cursor）的关闭

使用游标时，每次使用结束后都应该关闭游标，避免游标泄露。

不同的查询应使用不同的游标，避免定义同名游标在不同查询中使用。

### 避免过长SQL

过长的SQL文本会造成SQL编译耗时长、内存使用增加，严重时，还可能导致数据库解析异常，导致交易受影响。

通常建议SQL文本长度不超过2000字节。

SQL文本超长大多数情况下是由于拼接查询条件造成，开发人员可以通过使用中间表、限制单次选中记录数等方式避免。

### 标量子查询

标量子查询，即select 的字段列表中存在的一个子查询，且该子查询对于主查询一行记录只能返回一条记录。应确保标量子查询里与稍大表的关联都走索引。

### 关键资源更新

对关键资源UPDATE时，如果使用SELECT FOR UPDATE来获取行锁，根据业务特点可以加上wait 时间限制或SKIP LOCK，并在应用中对相应的出错返回进行捕获，设计合适的重做、告警措施，从而避免因为关键资源无法获取造成UPDATE任务不断堆积，影响不断扩大，尽量降低问题影响范围。

执行关键资源UPDATE应设置SQL执行超时时间，避免单个AP、单个连接的问题不能自动解决，持续影响整个系统。

### 不得使用数据字典表用于业务SQL

应用业务类相关SQL语句，不允许使用数据库数据字典表用于业务逻辑的执行或判断。

数据字典表记录的内容会因为环境、参数、对象数量、应用DDL操作等差异，造成生产环境执行计划或逻辑不符合预期，因此SQL语句中，不得使用数据字典表用于业务逻辑的执行或判断，数据字典表，仅为系统提供即时数据和运行信息的数据支撑。

### 秒杀超高并发场景必须避免数据库单条记录瓶颈

交易类尤其是秒杀类高并发业务，不得仅通过数据库单条记录进行数量控制控制。可以考虑总数分解成多行记录，不同入口更新不同记录以应对超高并发需求。

### 申请独占锁的操作需要错开执行

在进行独占锁方式锁定锁资源的语句执行过程中，如果在此期间如果对同一个对象的其他操作也申请独占锁，将会阻塞其他申请共享锁的会话，严重影响业务，所以要错开执行相关操作。

相关操作的操作包括：

* 各种DDL,包括alter,drop,truncate,grant,revoke,debug,analyze等操作。
* 显式指定Lock exclusive。
* 采集统计信息。
* 添加，删除和拆分分区等相关操作。
* 其它一些锁定整个对象的操作。

## 数据库连接规范

### 避免使用短连接，建议使用长连接

短连接导致每次执行语句都要新建连接，完成后再释放连接，连接风暴会对大量消耗CPU,并且可能造成监听器过载，导致正常的连接请求也受到影响。

另外在开启了数据库登录审计的数据库中，登录操作会在记录SYSTEM或者SYSAUX表空间或者文件系统中，频繁连接会占用SYSTEM或者SYSAUX大量空间,可能导致相关表空间过大的问题和数据字典访问的低效或者文件系统文件数过多，文件系统的空间问题。

### 合理控制总连接数

每个专有数据库连接都会占用内存等资源，并且过多的数据库连接数并不能提高业务性能。

数据库服务器的内存等资源是有限的，并且一些测试结果显示过多的连接数可能出现性能下降的情况。

请根据实际需求，合理设置应用服务器的连接数。

### 禁止使用第三方工具连接生产数据库

因为第三方工具存在版本兼容性、安全等方面的不确定风险，特别一些从非官方网站下载的工具或者破解程序，可能带有木马程序和勒索程序，可能对生产数据带来严重的影响。禁止使用第三方工具访问生产数据库。建议通过自带的sqlplus等工具，访问生产数据库。

### 通过合法途径连接到数据库

数据库应用用户应该通过合法途径登录到数据库。

应用用户应该通过JDBC等正规程序，连接到数据库。

数据库监控用户应该只有查询权限，并且严格控制访问对象，按照最小授权方式进行赋权。

## SQL量化指标

### 数据库系统级别负载情况指标

|  |  |  |
| --- | --- | --- |
| 指标 | 含义 | 说明 |
| DB Time(s) | 数据库整体响应时间 |  |
| DB CPU(s) | 数据库整体CPU |  |
| Background CPU(s) | 数据库后台进程消耗CPU |  |
| Redo size (bytes) | Redo大小 |  |
| Logical read (blocks) | 逻辑读 |  |
| Block changes | 块变化数量 |  |
| Physical read (blocks) | 物理读 |  |
| Physical write (blocks) | 物理写 |  |
| Read IO requests | 读IO请求数 |  |
| Write IO requests | 写IO请求 |  |
| Read IO (MB) | 读IO（MB） |  |
| Write IO (MB) | 写IO（MB） |  |
| IM scan rows | In Memory扫描行数 |  |
| Session Logical Read IM | 会话In Memory逻辑读 |  |
| Global Cache blocks received | 接受到的Global Cache块数 |  |
| Global Cache blocks served | 服务的Global Cache块数 |  |
| User calls | 用户发起调用数 |  |
| Parses (SQL) | 语句解析次数 |  |
| Hard parses (SQL) | 语句硬解析次数 |  |
| SQL Work Area (MB) | SQL工作区大小 |  |
| Logons | 登录次数 |  |
| Executes (SQL) | SQL执行次数 |  |
| Rollbacks | 回滚次数 |  |
| Transactions | 事务数 |  |

### SQL语句的需要关注的指标

后期通过实际数据的测试，对TOP SQL消耗各类资源记录并分析相关资源消耗

|  |  |  |
| --- | --- | --- |
| 指标 | 含义 | 说明 |
| Elapsed Time | 语句总耗时 |  |
| CPU Time | 语句消耗CPU |  |
| User I/O Wait Time | 语句的IO等待时间 |  |
| Gets | 逻辑读 |  |
| Reads | 物理读 |  |
| Executions | 执行次数 |  |
| Parse Calls | 解析调用次数 |  |
| Sharable Memory | 占用Shared pool占用空间 |  |
| Version Count | 语句不同版本数 | 可能因为一些原因，导致SQL无法共享 |
| Cluster Wait Time | 集群等待时间 |  |

### SQL量化分析和优化工具

建议在开发过程中，通过如下一些分析工具进行上述指标的量化分析和优化工作：

* Explain
* SQL\*Trace
* TKPROF
* Auto\*Trace
* AWR
* ADDM
* SQL\*Profiling
* SQL Access Advisor
* SQL Tuning Advisor
* ASH
* 其它工具

### SQL语句执行计划分析

一个优秀的数据库开发人员，一定不能把Oracle当成一个黑盒子，只关注结果的正确性，而应该深入理解SQL语句的执行机制，至少要对关键模块的SQL语句进行Oracle SQL语句的执行计划分析。分析内容至少包括：

* 表的访问方式
* 表的索引类型和策略
* 表的连接类型和过程
* 排序过程
* 汇总过程
* 并行处理过程

**Oracle Corporation is the world's largest enterprise software company. Oracle offers its database, tools and application products, along with related consulting, education, and support services.**

**Oracle Support Services provides technical and operational expertise for all Oracle technology and applications.**

**Oracle is a registered trademark of Oracle Corporation. Other names may be trademarks of their respective owners.**

**All Rights Reserved. No part of this publication may be reprinted or otherwise reproduced without permission from the publisher.**

**Copyright © 2020 Oracle Corporation.**

