[基于 oracle 的 sql 优化]

基于 oracle 的 sql 优化

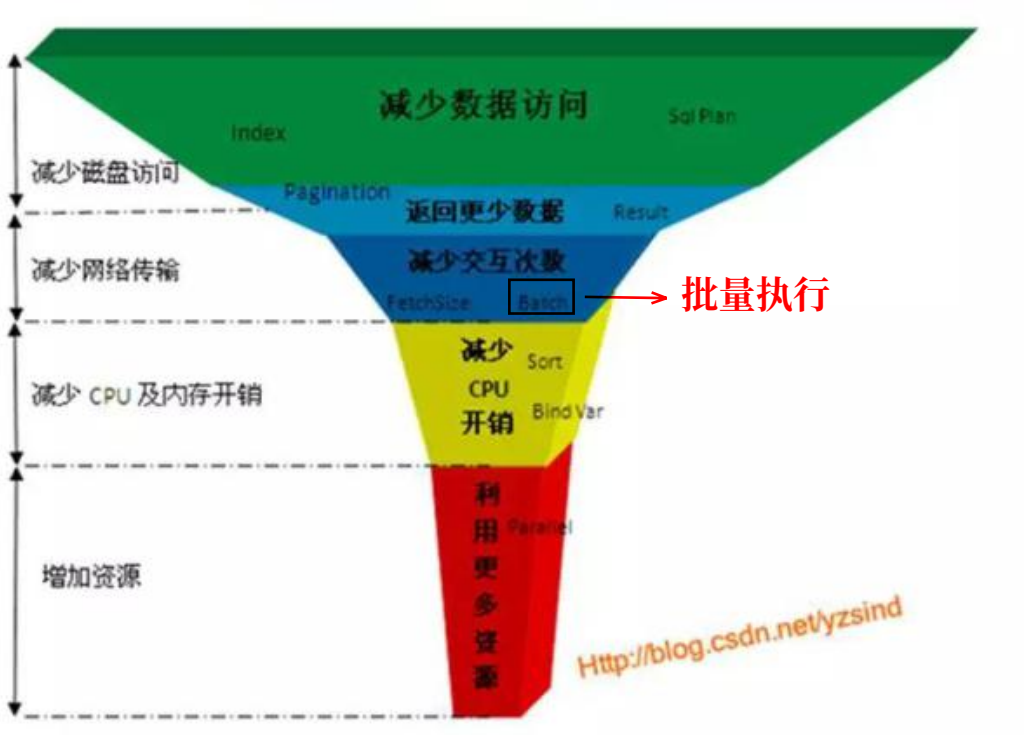
一. 编写初衷描述

在应有系统开发初期，由于数据库数据较少，对于 sql 语句各种写法的编写体现不出 sql 的性能优劣，随着数据的不断增加，出现海量数据，劣质 sql 与优质 sql 在执行效率甚至存在百倍差距，可见 sql 优化的重要性

二. Sql 语句性能优化

2.1 认识 Oracle 的执行过程

2.2 Oracle 优化法则---漏斗法则



2.3 Oracle 执行计划

2.3.1 什么是 Oracle 执行计划

执行计划是一条查询语句在 Oracle 中执行过程或者访问路径的描述.

2.3.2 查看 Oracle 执行计划

1.执行计划常用的列字段解释

基数：返回的结果集行数

字节：执行该步骤后返回的字节数

耗费(cust),CPU 耗费：Oracle 估计的该步骤的执行成本，用于说明 SQL 执行的代价，理论上越小越好.

2.3.3 看懂 Oracle 执行计划

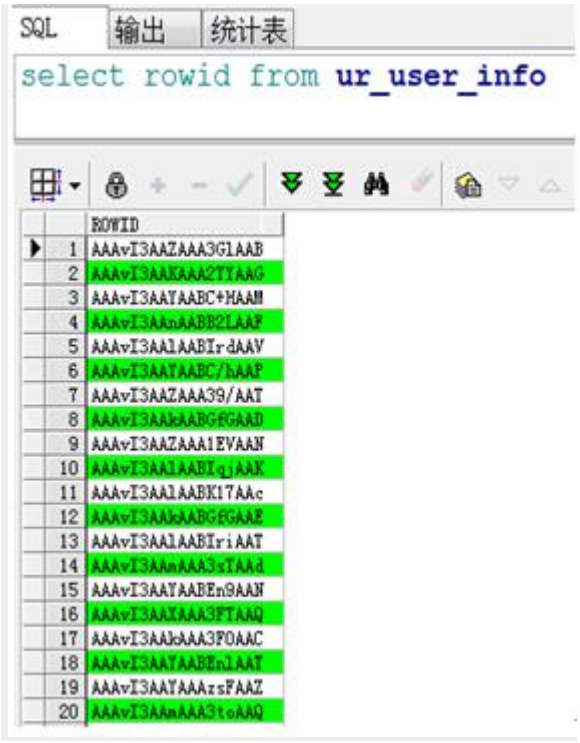
Description	对象...	耗费	基数	字节	CPU 耗费
SELECT STATEMENT, GOAL = ALL_ROWS		186050	17044918	1806761308	15253011532
TABLE ACCESS BY INDEX ROWID	DBACCADM	1	1	14	8341
INDEX UNIQUE SCAN	DBACCADM	0	1		1050
TABLE ACCESS BY INDEX ROWID	DBACCADM	1	1	16	8381
INDEX UNIQUE SCAN	DBACCADM	0	1		1050
PARTITION RANGE ALL		185959	17044918	1806761308	15252249421
PARTITION RANGE ALL		185959	17044918	1806761308	15252249421
TABLE ACCESS FULL	DBACCADM	185959	17044918	1806761308	15252249421

2.3.3.1 执行顺序

根据缩进来判断，缩进最多的最先执行(缩进相同时，最上面的最先执行)

2.4 表的访问方式

- TABLE ACCESS FULL(全表扫描)
- TABLE ACCESS BY ROWID(通过 rowid 的表存取)
- TABLE ACCESS BY INDEX SCAN(索引扫描)



2.4.1 ABLE ACCESS FULL(全表扫描)

Oracle 会读取表中的所有行，并检查是否满足 where 语句中条件；

使用建议：数据量太大的表不建议全表扫描

2.4.2 TABLE ACCESS BY ROWID(通过 ROWID 的表存取)

ROWID 的解释：oracle 会自动加在表的每一行的最后一列伪列，表中并不会物理存储 ROWID 的值，一旦一行数据插入后，则其对应的 ROWID 在该行的生命周期内是唯一的，即使发生行迁移，该行的 ROWID 值也不变。

2.4.3 TABLE ACCESS BY INDEX SCAN(索引扫描)

在索引块中即存储每个索引的键值，也存储具有该键值所对的 ROWID.

索引的扫描分两步：首先是找到索引所对的 ROWID，其次通过 ROWID 读取改行数据

索引扫描又分五种：

- INDEX UNIQUE SCAN（索引唯一扫描）
- INDEX RANGE SCAN（索引范围扫描）
- INDEX FULL SCAN（索引全扫描）
- INDEX FAST FULL SCAN（索引快速扫描）
- INDEX SKIP SCAN（索引跳跃扫描）

(a).INDEX UNIQUE SCAN（索引唯一扫描）：

针对唯一性索引（UNIQUE INDEX）的扫描，每次至多只返回一条记录,主要针对该字段为主键或者唯一；

(b). INDEX RANGE SCAN（索引范围扫描）

使用一个索引存取多行数据；

发生索引范围扫描的三种情况：

- 在唯一索引列上使用了范围操作符（如：> < <> >= <= between）
- 在组合索引上，只使用部分列进行查询（查询时必须包含前导列，否则会走全表扫描）
- 对非唯一索引列上进行的任何查询

(c). INDEX FULL SCAN（索引全扫描）

- 进行全索引扫描时，查询出的数据都必须从索引中可以直接得到

(d). INDEX FAST FULL SCAN（索引快速扫描）

- 扫描索引中的所有的数据块，与 INDEX FULL SCAN 类似，但是一个显著的区别是它不对查询出的数据进行排序（即数据不是以排序顺序被返回）

(e). INDEX SKIP SCAN（索引跳跃扫描）：

Oracle 9i 后提供，有时候复合索引的前导列（索引包含的第一列）没有在查询语句中出现，oralce 也会使用该复合索引，这时候就使用的 INDEX SKIP SCAN;

当 Oracle 发现前导列的唯一值个数很少时，会将每个唯一值都作为常规扫描的入口，在此基础上做一次查找，最后合并这些查询；

例如：

假设表 emp 有 ename（雇员名称）、job（职位名）、sex（性别）三个字段，并且建立了如 create index idx_emp on emp (sex, ename, job) 的复合索引；

因为性别只有 '男' 和 '女' 两个值，所以为了提高索引的利用率，Oracle 可将这个复合索引拆成 ('男', ename, job), ('女', ename, job) 这两个复合索引；

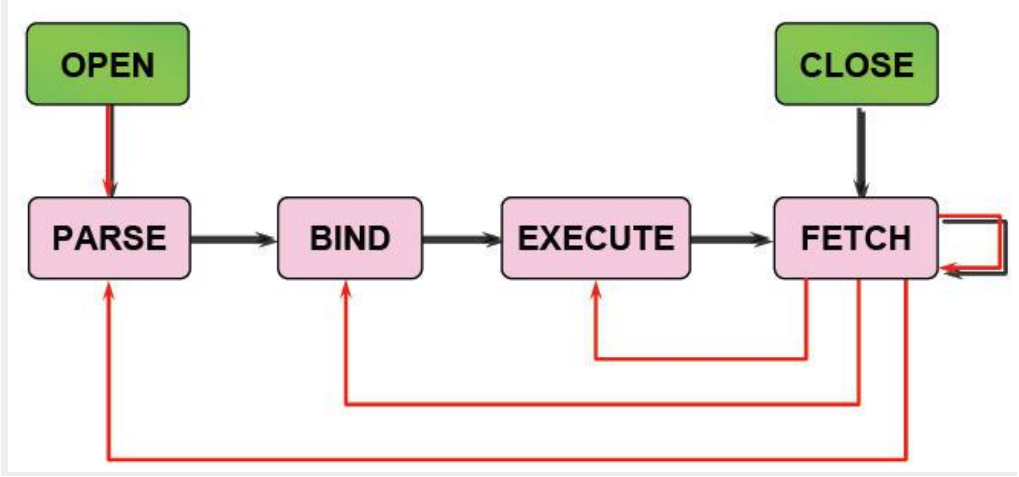
当查询 select * from emp where job = 'Programmer' 时，该查询发出后：

Oracle 先进入 sex 为'男'的入口，这时候使用到了 ('男', ename, job) 这条复合索引，查找 job = 'Programmer' 的条目；

再进入 sex 为'女'的入口，这时候使用到了 ('女', ename, job) 这条复合索引，查找 job = 'Programmer' 的条目；

最后合并查询到的来自两个入口的结果集。

2.5 Sql 语句的处理过程



1.在共享池中查找 SQL 语句

2.检查语法

3.检查语义和相关的权限

4.合并(MERGE)视图定义和子查询

5.确定执行计划

绑定(BIND)：

1.在语句中查找绑定变量

2.赋值(或重新赋值

执行(EXECUTE)：

1.应用执行计划

2.执行必要的 I/O 和排序操作

提取(FETCH)：

1.从查询结果中返回记录

2.必要时进行排序

3.使用 ARRAY FETCH 机制

共享游标：好处

1.减少解析

2.动态内存调整

3.提高内存使用率

2.5.1 Sql 共享原理

Oracle 将执行过程中的 sql 语句放在内存的共享池中，可以被所有的数据库用户共享到，当执行一条 sql 语句时，如果它和之前的 sql 执行语句完全相同时，oracle 会快速获取被解析的语句以及最好的执行路劲。

这块系统属于全局的区域，但是 oracle 只对简单的表提供高速缓存，如果是多表的连接查询，数据库管理员必须在启动参数文件中为该区域设置合适的参数，增加共享的可能性。

2.5.2 Sql 共享的条件(注意事项)

1.执行语句必须与共享池语句完全一样，包括(大小写，空格，换行等).

2.两条语句所指的对象必须完全相同。

3.两个 SQL 语句绑定变量的名字必须相同。

例子：字符级的比较

```
SELECT * FROM UR_USER_INFO
```

```
Select * from ur_user_info
```

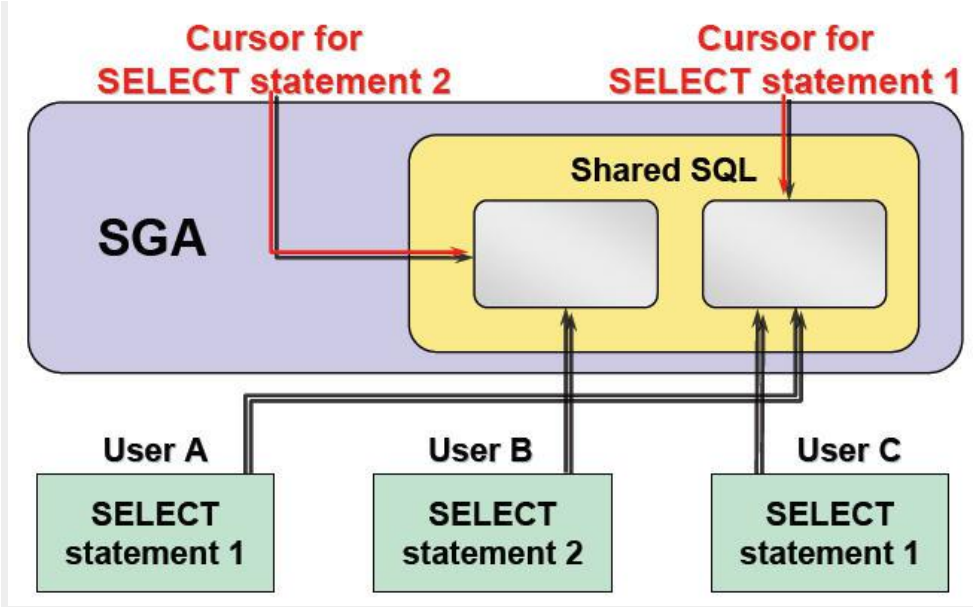
例子：相同的绑定变量名

```
select pay_fee,pay_method from bal_payment_info where pay_sn= : pay_sn;
```

```
select pay_fee,pay_method from bal_payment_info where pay_sn= : pay_no;
```

绑定变量不一样，不能共享。

2.5.3 共享 sql 区域



2.5.4 Sql 解析与共享 sql 语句

当一个 Oracle 实例接收一条 sql 后

1、Create a Cursor 创建游标

2、Parse the Statement 分析语句

3、Describe Results of a Query 描述查询的结果集

4、Define Output of a Query 定义查询的输出数据

5、Bind Any Variables 绑定变量

6、Parallelize the Statement 并行执行语句

7、Run the Statement 运行语句

8、Fetch Rows of a Query 取查询出来的行

9、Close the Cursor 关闭游标

2.6 绑定变量

2.6.1 重编译问题

例如：

```
select  *from ur_user_info where contract_no = 32013484095139
```

下面这个语句每执行一次就需要在 SHARE POOL 硬解析一

次，一百万用户就是一百万次，消耗 CPU 和内存，如果业务

量大，很可能导致宕库……

如果绑定变量，则只需要硬解析一次，重复调用即可

2.6.2 绑定变量解决重编译问题

例如：

```
select  *from ur_user_info where contract_no = 32013484095139

select  *from ur_user_info where contract_no = 12013481213149
```

使用绑定变量

```
select  *from ur_user_info where contract_no =：contract_no
```

2.6.3 绑定变量注意事项

a、不要使用数据库级的变量绑定参数 cursor_sharing 来强

制绑定，无论其值为 force 还是 similar

b、有些带> < 的语句绑定变量后可能导致优化器无法正确

使用索引

2.5 SQL 优化遵循的原则及注意事项

- 目标：

(1).SQL 优化的一般性原则设计方面：

- 设计方面：

(1).尽量依赖 oracle 的优化器，并为其提供条件；

(2).合适的索引，索引的双重效应，列的选择性；

- 编码方面：

(1).利用索引，避免大表 FULL TABLE SCAN；

(2).合理使用临时表；

(3).避免写过于复杂的 sql，不一定非要一个 sql 解决问题；

(4).在不影响业务的前提下减小事务的粒度；

2.5.1 IS NULL 与 IS NOT NULL

任何 sql 语句只要在 where 语句后面添加 is null 或者 is not null，那么 oracl 优化器将不再使用索引。exists

2.5.2 使用带通配符（%）的语句

列举两个例子说明该问题：

查询 ur_user_info 表中 phone_no 带 10 的服务号码

例子 1：Select *from ur_user_info where phone_no like '%10%';

例子 2：Select *from ur_user_info where phone_no like '10%';

由于例 1 中通配符（%）在搜寻词首出现，所以 oracle 系统不使用 phone_no 的索引，通配符会降低查询的效率，但当通配符不再首出现，又能使用索引，如例 2 所示。

三．ORACLE 语句优化规则

3.1 选择最有效的表名顺序

例如：TAB1 1000 条记录， TAB2 1 条记录

选择记录最少的作为基表

Select count(*) from tab1,tab2;

如果有 3 个或者 3 个以上的表则选择交叉表作为基表

3.2 where 字句中的连接顺序

oracle 的解析按照从上而下解析，因此表之间的连接必须写在 where 条件之前：

从左到右

例如：

低效率：

select .. from

```
        emp e

        where sal > 50000 and job = 'manager'

        and 25 < (select count(*) from emp where mgr=e.empno);
```

高效率：

```
select .. from

emp e

where 25 < (select count(*) from emp where mgr=e.empno)

        and sal > 50000

        and job = 'manager';
```

3.3 通配符’*’的使用

Sql 在执行带通配符的语句时，如果‘%’在首位，那么在字段上建立的主键或者索引将会失效!

应该避免类似语句的出现

```
Select name from user_info where name='%A';
```

3.4 使用 truncate 代替 delete

当删除表时，使用 delete 执行操作，回滚端用来存放可恢复的信息，当没有提交事务的时候，执行回滚事务，数据会恢复到执行 delete 操作之前，而当用 truncate 是，回滚端则不会存放可恢复的信息，减少资源的调用。

3.5 用 where 字句替换 HAVING 字句

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.

3.6 减少对表的查询

低效：

Select tab_name from tables where tab_name = (select

tab_name from tab_columns where version = 604) and db_ver=

(select db_ver from tab_columns where version = 604)

高效：

select tab_name from tables where (tab_name,db_ver) =

3.7 用 in 代替 or

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);

3.8 删除重复数据

最高效的删除重复记录的方法

```
Deleet from ur_user_info a
    Where a.rowid>(select min(b.rowid)
        From ur_user_info b
    Where b. uid=a. uid);
```

3.9 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.

例如,一个UNION查询,其中每个查询都带有GROUP BY子句, GROUP BY会触发嵌入排序(NESTED SORT) ; 这样, 每个查询需要执行一次排序, 然后在执行UNION时, 又一个唯一排序(SORT UNIQUE)操作被执行而且它只能在前面的嵌入排序结束后才能开始执行.嵌入的排序的深度会大大影响查询的效率.

3.10 自动选择索引

如果表中有两个以上（包括两个）索引，其中有一个唯一性索引，而其他是非唯一性．在这种情况下，ORACLE 将使用唯一性索引而完全忽略非唯一性索引．

举例：

select ename from emp where empno = 2326 and deptno = 20 ;这里，只有 empno 上的索引是唯一性的，所以 empno 索引将用来检索记录．

table access by rowid on emp index unique scan on emp_no_idx；

3.11 至少要包含组合索引的第一列

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．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。

3.12 避免在索引列上使用函数

低效：

```
select ..

from dept

where sal * 12 > 25000;
```

高效:

```
select ..

from dept
```

where sal > 25000/12;

3.13 避免出现索引列自动转换

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 EMP_TYPE 是一个字符类型的索引列.

select user_no,user_name,address

from user_files

where user_no = 109204421

这个语句被 ORACLE 转换为:

select user_no,user_name,address

from user_files

where to_number(user_no) = 109204421 因为内部发生的类型转换，这个索引将不会被用到!

3.14 避免出现索引列自动转换

如用 :

where a.order_no = b.order_no

不用 ：

```
where to_number (substr(a.order_no, instr(b.order_no, '.') - 1)

= to_number (substr(a.order_no, instr(b.order_no, '.') - 1)
```

3.15 使用 DECODE 来减少处理时间

例如:

```
select count(*) sum(sal)

  from emp

 where dept_no = 0020

and ename like 'smith%';

select count(*) sum(sal)

  from emp

 where dept_no = 0030

   and ename like 'smith%';
```

你可以用 DECODE 函数高效地得到相同结果

```
select count(decode(dept_no, 0020, 'x', null)) d0020_count,
       count(decode(dept_no, 0030, 'x', null)) d0030_count,
       sum(decode(dept_no, 0020, sal, null)) d0020_sal,
       sum(decode(dept_no, 0030, sal, null)) d0030_sal
  from emp
 where ename like 'smith%';
```

3.16 减少对表的查询

低效

```
        select tab_name
         from tables
         where tab_name = ( select tab_name
                            from tab_columns
                            where version = 604)
```

```
      and db_ver= ( select db_ver

                  from tab_columns

                  where version = 604)
```

高效

```
   select tab_name

      from tables

      where  (tab_name,db_ver)

     = ( select tab_name,db_ver)

            from tab_columns

            where version = 604)
```

3.17 Order by 语句

(a).ORDER BY 语句决定了 Oracle 如何将返回的查询结果排序。Order by 语句对要排序的列没有什么特别的限制，也可以将函数加入列中（象联接或者附加等）。任何在 Order by 语句的非索引项或者有计算表达式都将降低查询速度。

(b). order by 语句以找出非索引项或者表达式，它们会降低性能。解决这个问题的办法就是重写 order by 语句以使用索引，也可以为所使用的列建立另外一个索引，同时应绝对避免在 order by 子句中使用表达式。

3.18 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。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。

3.19 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.

低效:

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

高效:

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

3.20 用>= 替代 >

如果 DEPTNO 上有一个索引。

高效:

```
SELECT *
```

```
FROM EMP

WHERE DEPTNO >=4

```

低效:

```
SELECT *

FROM EMP

WHERE DEPTNO >3
```

3.21 通过使用>=、<=等，避免使用 NOT 命令

例子：

```
select * from employee where salary <> 3000;
```

对这个查询，可以改写为不使用 NOT：

```
select * from employee where salary<3000 or salary>3000;
```

虽然这两种查询的结果一样，但是第二种查询方案会比第一种查询方案更快些。第二种查询允许 Oracle 对 salary 列使用索引，而第一种查询则不能使用索引。

3.22 字符型字段的引号

比如有的表 PHONE_NO 字段是 CHAR 型,而且创建有索引，

但在 WHERE 条件中忘记了加引号，就不会用到索引。

WHERE PHONE_NO='13920202022'

WHERE PHONE_NO=13920202022

四．优化总结

a.创建表的时候。应尽量建立主键，尽量根据实际需要调整数据表的 PCTFREE 和 PCTUSED 参数；大数据表删除，用 truncate table 代替 delete。

OLTP面向事务OLAP面向分析

b．合理使用索引，在 OLTP 应用中一张表的索引不要太多。数据重复量大的列不要建立二叉树索引，可以采用位图索引；组合索引的列顺序尽量与查询条件列顺序保持一致；对于数据操作频繁的表，索引需要定期重建，以减少失效的索引和碎片。

c.查询尽量用确定的列名，少用*号。

select count(key)from tab where key> 0 性能优于 select count(*)from tab；

d．尽量少嵌套子查询，这种查询会消耗大量的 CPU 资源；对于有比较多 or 运算的查询，建议分成多个查询，用 union all 联结起来；多表查询的查询语句中，选择最有效率的表名顺序。Oracle 解析器对表解析从右到左，所以记录少的表放在右边。

e.尽量多用 commit 语句提交事务，可以及时释放资源、解锁、释放日志空间、减少管理花费；在频繁的、性能要求比较高的数据操作中，尽量避免远程访问，如数据库链等，访问频繁的表可以常驻内存：alter table．．．cache；

f.在 Oracle 中动态执行 SQL，尽量用 execute 方式，不用 dbms_sql 包。

-