Oracle SQL 优化

参考 ：https://www.cnblogs.com/rootq/archive/2008/11/17/1334727.html

1. 选择有效的表名顺序
2. WHERE子句的连接顺序
3. SELECT子句避免使用*
4. 减少访问数据库的次数
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函数减少处理时间
7. 整合简单，无关联的数据库访问
8. 删除重复记录
9. 用TRUNCATE代替DELETE
10. 尽量多使用COMMIT
11. 用WHERE子句代替HAVING子句
12. 减少对表的查询
13. 通过内部函数提高SQL效率
14. 使用表的别名
15. 用EXISTS代替IN, NOT EXISTS代替NOT IN
16. 识别低效执行的SQL语句
17. 用索引提高效率
18. 用EXISTS代替DISTINCT

eg ：

(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
```

参考 ： https://www.cnblogs.com/eric_ibm/archive/2012/01/09/oracle_skill.html

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num is null
```

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

```
select id from t where num=0
```

3.应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

4.应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num=10 or num=20
```

可以这样查询：

```
select id from t where num=10
union all
select id from t where num=20
```

5.in 和 not in 也要慎用，否则会导致全表扫描，如：

```
select id from t where num in(1,2,3)
```

对于连续的数值，能用 between 就不要用 in 了：

```
select id from t where num between 1 and 3
```

6.下面的查询也将导致全表扫描：

select id from t where name like '%abc%'

若要提高效率，可以考虑全文检索。

7.如果在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：

select id from t where num=@num

可以改为强制查询使用索引：

select id from t with(index(索引名)) where num=@num

8.应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：

select id from t where num/2=100

应改为:

select id from t where num=100*2

9.应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：

select id from t where substring(name,1,3)='abc' // oracle总有的是substr函数。

select id from t where datediff(day,createdate,'2005-11-30')=0 //查过了确实没有datediff函数。

应改为:

select id from t where name like 'abc%'

select id from t where createdate>='2005-11-30' and createdate<'2005-12-1' //

oracle 中时间应该把char 转换成 date 如： createdate >= to_date('2005-11-30','yyyy-mm-dd')

10.不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

11.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

12.不要写一些没有意义的查询，如需要生成一个空表结构：

select col1,col2 into #t from t where 1=0

这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：

create table #t(...)

13.很多时候用 exists 代替 in 是一个好的选择：

select num from a where num in(select num from b)

用下面的语句替换：

select num from a where exists(select 1 from b where num=a.num)

14.并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

15.索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

16.应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

17.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

18.尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

19.任何地方都不要使用 select * from t ，用具体的字段列表代替“*”，不要返回用不到的任何字段。

20.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

21.避免频繁创建和删除临时表，以减少系统表资源的消耗。

22.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

23.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

24.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

25.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

26.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

27.与临时表一样，游标并不是不可使用。对小型数据集使用 FAST_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

28.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE_IN_PROC 消息。

29.尽量避免大事务操作，提高系统并发能力。

30.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

参考 “ https://www.cnblogs.com/netjxz/archive/2009/09/21/1570991.html

一、执行顺序及优化细则

1.表名顺序优化

(1) 基础表放下面,当两表进行关联时数据量少的表的表名放右边

表或视图:

Student_info (30000条数据)

Description_info (30条数据)

```
select *
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
```

与

```
select *
  from student_info     si--学生信息表
      ,description_info di
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
```

以student_info作为基础表,你会发现运行的速度会有很大的差距。

(2) 当出现多个表时,关联表被称之为交叉表,交叉表作为基础表

```
select *
  from description_info di
    ,description_info di2
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and si.school_id = di.lookup_code(+)
   and di.lookup_type(+) = 'SCHOOL_ID'
```

与

```
select *
  from student_info     si--学生信息表
      ,description_info di
      ,description_info di2
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and si.school_id = di.lookup_code(+)
   and di.lookup_type(+) = 'SCHOOL_ID'
```

以student_info作为基础表,你会发现运行的速度会有很大的差距,

当基础表放在后面,这样的执行速度会明显快很多。

2.where执行顺序

where执行会从至下往上执行

```
select *
from student_info si --学生信息表
where si.school_id=10 --学院ID
and  si.system_id=100--系ID
```

摆放where子句时,把能过滤大量数据的条件放在最下边

3. is null 和is not null

当要过滤列为空数据或不为空的数据时使用

```
select *
from student_info si --学生信息表
where si.school_id is null(当前列中的null为少数时用is not null,否则is null)
```

4.使用表别名

当查询时出现多个表时,查询时加上别名,

避免出现减少解析的时间字段歧义引起的语法错误。

5. where执行速度比having快

尽可能的使用where代替having

```
select  from student_info si
group by si.student_id
having si.system_id!=100
  and si.school_id!=10
(select  from student_info si
wehre si.system_id!=100
and si.school_id!=10
group by si.student_id)
```

6. * 号引起的执行效率

尽量减少使用select * 来进行查询,当你查询使用*,

数据库会进行解析并将*转换为全部列。

二、替代优化

1、用>=替代>

```
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id>=10
  与
  select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id>9
  执行时>=会比>执行得要快
```

2、用UNION替换OR (适用于索引列)

```
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id=10
  union
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id=2
   上面语句可有效避免全表查询
   select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id=10
  or ui.student_id=2
  如果坚持要用OR, 可以把返回记录最少的索引列写在最前面
```

3、用in 代替or

```
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id=10
  or ui.student_id=20
  or ui.student_id=30
  改成
  select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id in (10,20,30)
  执行会更有效率
```

4、 Union All 与Union

Union All重复输出两个结果集合中相同记录

如果两个并集中数据都不一样.那么使用Union All 与Union是没有区别的,

```
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id=10
  union All
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id=2
  与
  select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id=10
  union
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id=2
```

但Union All会比Union要执行得快

5、分离表和索引

总是将你的表和索引建立在另外的表空间内

决不要将这些对象存放到SYSTEM表空间里

三、一些优化技巧

1、计算表的记录数时

```
select count(si.student_id)
from Student_info si(student_id为索引)
```

与

```
select count(*) from Student_info si
```

执行时.上面的语句明显会比下面没有用索引统计的语句要快

2.使用函数提高SQL执行速度

当出现复杂的查询sql语名,可以考虑使用函数来提高速度

查询学生信息并查询学生(李明)个人信息与的数学成绩排名

如

```
select di.description student_name
      ,(select res.order_num--排名
         from result res
        where res.student_id = di.student_id
        order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and di.description = '李明'
```

而且我们将上面order_num排名写成一个fuction时

```
create or replace package body order_num_pkg is
function order_num(p_student_id number) return_number is
  v_return_number number;
begin
  select res.order_num --排名
    into v_return_number
    from result res
   where res.student_id = di.student_id
   order by result_math;
  return v_return_number;
exception
  when others then
    null;
    return null;
```

```
end;
end order_num_pkg;
```
执行
```
select di.description student_name
      ,order_num_pkg.order_num(di.student_id) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and di.description = '李明'
```
执行查询时的速度也会有所提高

3.减少访问数据库的次数

执行次数的减少(当要查询出student_id=100的学生和student_id=20的学生信息时)
```
select address_id
from student_info si --学生信息表
where si.student_id=100
```
与
```
select address_id
from student_info si --学生信息表
where si.student_id=20
```
都进行查询.这样的效率是很低的

而进行
```
(
select si.address_id,si2.address_id
from student_info si --学生信息表
,student_info si2
where si.student_id=100
and si2.student_id=20
```
与
```
select decode(si.student_id,100,address_id)
   ,decode(si.student_id,20,address_id)
from student_info si
)
```
执行速度是提高了,但可读性反而差了..

所以这种写法个人并不太推荐

4、用Exists(Not Exists)代替In(Not In)

在执行当中使用Exists或者Not Exists可以高效的进行查询

5、Exists取代Distinct取唯一值的

取出关联表部门对员工时,这时取出员工部门时,出现多条..
```
select distinct di.dept_name
  from departments_info di --部门表
      ,user_info        ui --员工信息表
where ui.dept_no = di.dept_no
```
可以修改成
```
  select di.dept_name
    from departments_info di --部门表
   where  exists (select 'X'
            from user_info ui --员工信息表
           where di.dept_no = ui.dept_no)
```
6、用表连接代替Exists

通过表的关联来代替exists会使执行更有效率
```
select ui.user_name
  from user_info ui--员工信息表
where exists (select 'x '
          from departments_info di--部门表
         where di.dept_no = ui.dept_no
           and ui.dept_cat = 'IT');
```

执行是比较快,但还可以使用表的连接取得更快的查询效率

```
   select ui.user_name
    from departments_info di
        ,user_info        ui --员工信息表
   where ui.dept_no = di.dept_no
     and ui.department_type_code = 'IT'
```

代码是经测试并进行优化所写,

以上只例子,具体使用还是要针对各个不同的具体的业务使用用Exists(Not Exists)代替In(Not In)

四、索引篇

1、运算导致的索引失效

```
select di.description student_name
      ,(select res.order_num--排名
         from result res
        where res.student_id = di.student_id
        order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and si.student_id+0=100/*student_id索引将失效*/
```

2、类型转换导致的索引失效

```
select di.description student_name
      ,(select res.order_num--排名
         from result res
        where res.student_id = di.student_id
        order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and di.student_id='100'
```

student_id为number类型的索引,当执行下列语句,

oracle会自动转换成

```
select di.description student_name
      ,(select res.order_num--排名
         from result res
        where res.student_id = di.student_id
        order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and di.student_id=to_number('100')
```

所幸,只是解析并转换类型,并没有导到失效,

但要是写成下面,将会使用其失效

```
select di.description student_name
      ,(select res.order_num--排名
         from result res
        where res.student_id = di.student_id
        order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and to_char(di.student_id)='100'
```

3、在索引列上进行计算引起的问题

```
select di.description student_name
      ,(select res.order_num--排名
          from result res
         where res.student_id = di.student_id
         order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and di.student_id-2=10
```

在索引列中进行运算,将会不使用索引而使用全表扫描

而将

```
select di.description student_name
      ,(select res.order_num--排名
          from result res
         where res.student_id = di.student_id
         order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and di.student_id=10+2
```

将会得到高效的运行速度

4、 Is not null引起的问题(student_id为索引)

不要把存在空值的列做为索引,否则无法使用索引

```
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id is not null--索引失效

select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id>=-1--索引有效
```

5、Order by导致索引失效(student_id为索引)

```
select ui.user_name
  from user_info ui--员工信息表
  group by ui.student_id
```

而使用

```
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id>=-1
```

将使其有效,

在order by中只存在两种条件下可以使用索引

(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.)

6、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.

7、 !=导致索引失效

```
select ui.user_name
  from user_info ui--员工信息表
  where ui.student_id!=0
```

在Where中使用!=将会把索引失效

8、%导致的索引失效

```
select di.description student_name
```

```
      ,(select res.order_num--排名
         from result res
        where res.student_id = di.student_id
        order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and di.look_code Like '%12'/*look_code为索引,索引将失效*/
```

而

```
select di.description student_name
      ,(select res.order_num--排名
         from result res
        where res.student_id = di.student_id
        order by result_math) order_num
  from description_info di
      ,student_info     si --学生信息表
where si.student_id = di.lookup_code(+)
   and di.lookup_type(+) = 'STUDENT_ID'
   and di.look_code Like '12%'/*索引有效*/
```

以上只例子,具体还是要针对各个不同的具体的业务使用

五、oracle 中的not Exists与Not in的性能巨大差异

Not Exists与Not in的作用同样是排除数据,在oracle 中使用not in并不象mysql中的执行那么快,如(

```
select jt1.doc_num --单据号码
      ,oalc.description school_name --学校名称
      ,oalc2.description system_name --系名称
      ,oalc.description class_name --班级名称
  from java_table1            jt1
      ,java_table_description oalc
      ,java_table_description oalc2
      ,java_table_description oalc3
where oalc.lookup_type(+) = 'JAVA_SCHOOL_NAME'
   and jt1.school_id = oalc.lookup_code(+)
   and oalc2.lookup_type(+) = 'JAVA_SYSTEM_NAME'
   and jt1.system_id = oalc2.lookup_code(+)
   and oalc3.lookup_type(+) = 'JAVA_CLASS_NAME'
   and jt1.class_id = oalc3.lookup_code(+)
   and not exists
(select jt2.header_id
         from java_table2 jt2 jt1.header_id = jt2.header_id))
```

与

```
select jt1.doc_num --单据号码
      ,oalc.description school_name --学校名称
      ,oalc2.description system_name --系名称
      ,oalc.description class_name --班级名称
  from java_table1            jt1
      ,java_table_description oalc
      ,java_table_description oalc2
      ,java_table_description oalc3
where oalc.lookup_type(+) = 'JAVA_SCHOOL_NAME'
   and jt1.school_id = oalc.lookup_code(+)
   and oalc2.lookup_type(+) = 'JAVA_SYSTEM_NAME'
   and jt1.system_id = oalc2.lookup_code(+)
   and oalc3.lookup_type(+) = 'JAVA_CLASS_NAME'
   and jt1.class_id = oalc3.lookup_code(+)
   and jt1.header_id not in (select jt2.header_id from java_table2 jt2)
```

当jt2表中的数据比较大时,就会出现巨大的差异,以上只能是我的个人理解与测试结果(java_table1 视图测试数据量为36749,java_table2 为300条),如有其它可相互讨论