原则一：注意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.   
尤其是“主键ID=？”这样的条件。

原则二： 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 。

简单地讲，语句执行的时间越短越好（尤其对于系统的终端用户来说）。而对于查询语句，由于全表扫描读取的数据多，尤其是对于大型表不仅查询速度慢，而且对磁盘IO造成大的压力，通常都要避免，而避免的方式通常是使用索引Index。

使用索引的优势与代价。  
优势：  
1）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.  
2） 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.  
代价： 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.。而且表越大，影响越严重。

使用索引需要注意的地方：

1、避免在索引列上使用NOT ，　   
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.

2、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;

3、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性能上将无法使用该索引．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;

4、注意通配符%的影响   
使用通配符的情况下Oracle可能会停用该索引。如 :

SELECT…FROM DEPARTMENT WHERE DEPT\_CODE like ‘%123456%'（无效）。

SELECT…FROM DEPARTMENT WHERE DEPT\_CODE = ‘123456'（有效）

5、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   
  
6、索引的一些“脾气”   
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!   
  
除了使用索引，我们还有其他能减少资源消耗的方法：  
  
1、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:

(低效): SELECT DISTINCT DEPT\_NO,DEPT\_NAME FROM DEPT D , EMP E

WHERE D.DEPT\_NO = E.DEPT\_NO

And E.sex =man

(高效): SELECT DEPT\_NO,DEPT\_NAME FROM DEPT D

WHERE EXISTS

( SELECT ‘X' FROM EMP E WHERE E.DEPT\_NO = D.DEPT\_NO

And E.sex =man

);

2、用(UNION)UNION ALL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.   
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.

高效: SELECT LOC\_ID , LOC\_DESC , REGION FROM LOCATION WHERE LOC\_ID = 10 UNION ALL

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”

3、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.   
4、Order By语句加在索引列，最好是主键PK上。

SELECT DEPT\_CODE FROM DEPT ORDER BY DEPT\_TYPE（低效）

SELECT DEPT\_CODE FROM DEPT ORDER BY DEPT\_CODE （高效）

5、避免使用耗费资源的操作:   
带 有DISTINCT,UNION,MINUS,INTERSECT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   
  
6、使用Where替代Having（如果可以的话）   
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'AND AVG(SAL)>XXX

高效:

SELECT JOB , AVG(SAL)

FROM EMP

WHERE JOB = ‘PRESIDENT'

OR JOB = ‘MANAGER' GROUP JOB Having AND AVG(SAL)>XXX

7、通常来说，如果语句能够避免子查询的 使用，就尽量不用子查询。因为子查询的开销是相当昂贵的。具体的例子在后面的案例“一条SQL的优化过程”中。

1:索引，我们最先想到的就是创建索引，创建索引可以成倍的提升查询的效率，节省时间。但是如果数据量太过于巨大的时候，这个时候单纯的创建索引是无济于事的，我们知道假如特别是在大数据量中统计查询，就拿1000W数据来说吧，如果使用count函数的话，最少要50-100秒以上，当然如果你的服务器配置够高，处理够快，或许会少很多但是一样会超过10秒。

　　单纯的建立索引是无济于事的。我们可以在创建索引的时候给索引加个属性，compress,这个属性可以将所创建的索引进行一个良好的归类，这样的话，查询速度会提升5-10倍，或者更高。但是唯一的缺点是，压缩索引只能手动创建，对于那些KEY是无法进行压缩的，因为KEY（主键）是自动创建的索引，compress必选的属性，一般默认是不创建。所以在创建压缩索引的时候，可以找其他的关键字段进行压缩，比如工单表里面的流水号

　　2:尽量少的使用那些函数，比如 IS NUll;IS NOT NULL,IN;NOT IN等这样的匹配函数，可以使用符号程序进行操作

　　3:尽量少使用子查询，如果你写个类，里面模仿子查询的效果，你就会发现，简直在要命，我们可以使用联合查询，或者是外连接查询，这样速度会比子查询快很多。

　　4:在使用索引的时候，注意如下：

　　Where子句中有"!="将使索引失效

　　select account\_name from test where amount != 0  （不使用）

　　select account\_name from test where amount > 0  （使用）

　　Where条件中对字段增加处理函数将不使用该列的索引

　　select \* from emp where to\_char（hire\_date,'yyyymmdd'）='20080411' （不使用）

　　select \* from emp where hire\_date = to\_char（'20080411','yyyymmdd'） （使用）

　　避免在索引列上使用IS NULL和 IS NOT NULL

　　select \* from emp where dept\_code is not null  （不使用）

　　select \* from emp where dept\_code > 0  （使用）

　　通配符% 的使用

　　select \* from emp where name like '%A'  （不使用索引）

　　select \* from emp where name like 'A%'  （使用索引）

|  |  |  |
| --- | --- | --- |
| 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。 |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | select id from t where num=10 or num=20 |

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | 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。 |