1. 选择最有效率的表名顺序(只在基于规则的优化器中有效)：

数据库的解析器按照从右到左的顺序处理FROM子句中的表名，FROM子句中写在最后的表(基础表 driving table)将被最先处理，在FROM子句中包含多个表的情况下,你必须选择记录条数最少的表作为基础表。如果有3个以上的表连接查询, 那就需要选择交叉表(intersection table)作为基础表, 交叉表是指那个被其他表所引用的表。

sql语句的执行过程是：from-->where-->group by -->having --- >order by --> select;

1. WHERE子句中的连接顺序：

数据库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。

1. SELECT子句中避免使用‘\*’：

数据库在解析的过程中, 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。

1. 减少访问数据库的次数：

数据库在内部执行了许多工作: 解析SQL语句, 估算索引的利用率, 绑定变量 , 读数据块等。

1. 整合简单,无关联的数据库访问：

如果你有几个简单的数据库查询语句,你可以把它们整合到一个查询中(即使它们之间没有关系)。

1. 删除重复记录：

最高效的删除重复记录方法 ( 因为使用了ROWID)例子：

oracle：DELETE FROM EMP E WHERE E.ROWID > (SELECT MIN(X.ROWID) FROM EMP X WHERE X.EMP\_NO = E.EMP\_NO);

Db2：delete from (select id,NAME, row\_number() over(partition by ID order by ID) as row\_num from emp) where row\_num>1；

1. 用TRUNCATE替代DELETE：

当删除表中的记录时,在通常情况下, 回滚段(rollback segments ) 用来存放可以被恢复的信息. 如果你没有COMMIT事务,数据库会将数据恢复到删除之前的状态(准确地说是恢复到执行删除命令之前的状况) 而当运用TRUNCATE时, 回滚段不再存放任何可被恢复的信息。当命令运行后,数据不能被恢复.因此很少的资源被调用,执行时间也会很短。（TRUNCATE只在删除全表适 用,TRUNCATE是DDL不是DML）。

ALTER TABLE < 表名 > ACTIVATE NOT LOGGED INITIALLY WITH EMPTY TABLE

1. 尽量多使用COMMIT：只要有可能，在程序中尽量多使用COMMIT, 这样程序的性能得到提高,需求也会因为COMMIT所释放的资源而减少，COMMIT所释放的资源:  
     
   a. 回滚段上用于恢复数据的信息。  
     
   b. 被程序语句获得的锁。  
     
   c. redo log buffer 中的空间。  
     
   d. 数据库为管理上述3种资源中的内部花费。
2. 减少对表的查询：

在含有子查询的SQL语句中,要特别注意减少对表的查询。例子：   
SELECT TAB\_NAME FROM TABLES WHERE (TAB\_NAME,DB\_VER) = ( SELECTTAB\_NAME,DB\_VER FROM TAB\_COLUMNS WHERE VERSION = 604)

1. 用表的别名(Alias)：

当在SQL语句中连接多个表时, 请使用表的别名并把别名前缀于每个Column上。这样一来,就可以减少解析的时间并减少那些由Column歧义引起的语法错误。

1. 用EXISTS替代IN、用NOT EXISTS替代NOT IN：

在许多基于基础表的查询中，为了满足一个条件,往往需要对另一个表进行联接。在这种情况下，使用EXISTS(或NOT EXISTS)通常将提高查询的效率。在子查询中，NOT IN子句将执行一个内部的排序和合并。无论在哪种情况下，NOT IN都是最低效的 (因为它对子查询中的表执行了一个全表遍历)。为了避免使用NOT IN ，我们可以把它改写成外连接(Outer Joins)或NOT EXISTS。

例子：

（高效）SELECT \* FROM EMP (基础表) WHERE EMPNO > 0 AND EXISTS (SELECT ‘X' FROM DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO AND LOC = ‘MELB')

(低效)SELECT \* FROM EMP (基础表) WHERE EMPNO > 0 AND DEPTNO IN(SELECT DEPTNO FROM DEPT WHERE LOC = ‘MELB')

1. 识别‘低效执行’的SQL语句：

虽然目前各种关于SQL优化的图形化工具层出不穷,但是写出自己的SQL工具来解决问题始终是一个最好的方法：

索引是表的一个概念部分，用来提高检索数据的效率，数据库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 数据库找出执行查询和Update语句的最佳路径时, 数据库优化器将使用索引。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。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。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：ALTER INDEX <INDEXNAME> REBUILD <TABLESPACENAME>

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

(高效): SELECT DEPT\_NO,DEPT\_NAME FROM DEPT D WHERE EXISTS ( SELECT ‘X' FROM EMP E WHERE E.DEPT\_NO = D.DEPT\_NO);

1. SQL语句用大写,因为数据库（ORACLE）总是先解析SQL语句，把小写的字母转换成大写的再执行。
2. 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在索引列上使用会产生和函数相同的影响。当数据库“遇到”NOT,他就会停止使用索引转而执行全表扫描。

1. 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;

1. 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

1. 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’

1. 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，在数据库8i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);

1. 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，数据库将无法使用该索引。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)， 数据库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，数据库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数据库停用该索引。

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;

1. 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。

1. 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'

1. 避免改变索引列的类型:

当比较不同数据类型的数据时， 数据库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,经过数据库类型转换, 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 。

这个语句被数据库转换为: 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! 为了避免数据库对你的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，数据库会优先转换数值类型到字符类 型。

1. 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：

(a)‘!=' 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。

(b)‘||'是字符连接函数。就象其他函数那样, 停用了索引。

(c)‘+'是数学函数。就象其他数学函数那样, 停用了索引。

(d)相同的索引列不能互相比较,这将会启用全表扫描。

1. 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。

1. 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