1.合理使用索引

索引是数据库中重要的数据结构，它的根本目的就是为了提高查询效率。现在大多数的数据库产品都采用IBM最先提出的ISAM索引结构。

索引的使用要恰到好处，其使用原则如下：

在经常进行连接，但是没有指定为外键的列上建立索引，而不经常连接的字段则由优化器自动生成索引。

在频繁进行排序或分组（即进行group by或order by操作）的列上建立索引。

在条件表达式中经常用到的不同值较多的列上建立检索，在不同值少的列上不要建立索引。比如在雇员表的“性别”列上只有“男”与“女”两个不同值，因此就无必要建立索引。如果建立索引不但不会提高查询效率，反而会严重降低更新速度。

如果待排序的列有多个，可以在这些列上建立复合索引（compound index）。

使用系统工具。如Informix数据库有一个tbcheck工具，可以在可疑的索引上进行检查。在一些数据库服务器上，索引可能失效或者因为频繁操作而 使得读取效率降低，如果一个使用索引的查询不明不白地慢下来，可以试着用tbcheck工具检查索引的完整性，必要时进行修复。另外，当数据库表更新大量 数据后，删除并重建索引可以提高查询速度。

(1)在下面两条select语句中:

SELECT \* FROM table1 WHERE field1<=10000 AND field1>=0;

SELECT \* FROM table1 WHERE field1>=0 AND field1<=10000;

如果数据表中的数据field1都>=0,则第一条select语句要比第二条select语句效率高的多，因为第二条select语句的第一个条件耗费了大量的系统资源。

第一个原则：在where子句中应把最具限制性的条件放在最前面。

(2)在下面的select语句中:

SELECT \* FROM tab WHERE a=… AND b=… AND c=…;

若有索引index(a,b,c)，则where子句中字段的顺序应和索引中字段顺序一致。

第二个原则：where子句中字段的顺序应和索引中字段顺序一致。

以下假设在field1上有唯一索引I1，在field2上有非唯一索引I2。

(3) SELECT field3,field4 FROM tb WHERE field1='sdf' 快

SELECT \* FROM tb WHERE field1='sdf' 慢[/cci]

因为后者在索引扫描后要多一步ROWID表访问。

(4) SELECT field3,field4 FROM tb WHERE field1>='sdf' 快

SELECT field3,field4 FROM tb WHERE field1>'sdf' 慢

因为前者可以迅速定位索引。

(5) SELECT field3,field4 FROM tb WHERE field2 LIKE 'R%' 快

SELECT field3,field4 FROM tb WHERE field2 LIKE '%R' 慢，

因为后者不使用索引。

(6) 使用函数如：

SELECT field3,field4 FROM tb WHERE upper(field2)='RMN'不使用索引。

如果一个表有两万条记录，建议不使用函数；如果一个表有五万条以上记录，严格禁止使用函数！两万条记录以下没有限制。

(7) 空值不在索引中存储，所以   
SELECT field3,field4 FROM tb WHERE field2 IS[NOT] NULL不使用索引。

(8) 不等式如

SELECT field3,field4 FROM tb WHERE field2!='TOM'不使用索引。

相似地，

SELECT field3,field4 FROM tb WHERE field2 NOT IN('M','P')不使用索引。

(9) 多列索引，只有当查询中索引首列被用于条件时，索引才能被使用。

(10) MAX，MIN等函数，使用索引。

SELECT max(field2) FROM tb 所以，如果需要对字段取max，min，sum等，应该加索引。

一次只使用一个聚集函数，如：

SELECT “min”=min(field1), “max”=max(field1) FROM tb

不如：SELECT “min”=(SELECT min(field1) FROM tb) , “max”=(SELECT max(field1) FROM tb)

(11) 重复值过多的索引不会被查询优化器使用。而且因为建了索引，修改该字段值时还要修改索引，所以更新该字段的操作比没有索引更慢。

(12) 索引值过大（如在一个char(40)的字段上建索引），会造成大量的I/O开销（甚至会超过表扫描的I/O开销）。因此，尽量使用整数索引。 Sp\_estspace可以计算表和索引的开销。

(13) 对于多列索引，ORDER BY的顺序必须和索引的字段顺序一致。

(14) 在sybase中，如果ORDER BY的字段组成一个簇索引，那么无须做ORDER BY。记录的排列顺序是与簇索引一致的。

(15) 多表联结（具体查询方案需要通过测试得到）

where子句中限定条件尽量使用相关联的字段，且尽量把相关联的字段放在前面。

SELECT a.field1,b.field2 FROM a,b WHERE a.field3=b.field3  
field3上没有索引的情况下:

对a作全表扫描，结果排序

对b作全表扫描，结果排序

结果合并。

对于很小的表或巨大的表比较合适。

field3上有索引

按照表联结的次序，b为驱动表，a为被驱动表

对b作全表扫描

对a作索引范围扫描

如果匹配，通过a的rowid访问  
(16) 避免一对多的join。如：

SELECT tb1.field3,tb1.field4,tb2.field2 FROM tb1,tb2 WHERE tb1.field2=tb2.field2 AND tb1.field2=‘BU1032’ AND tb2.field2= ‘aaa’

不如：

declare @a varchar(80)

SELECT @a=field2 FROM tb2 WHERE field2=‘aaa’

SELECT tb1.field3,tb1.field4,@a FROM tb1 WHERE field2= ‘aaa’

(16) 子查询

用exists/not exists代替in/not in操作

比较：

SELECT a.field1 FROM a WHERE a.field2 IN(SELECT b.field1 FROM b WHERE b.field2=100)

SELECT a.field1 FROM a WHERE EXISTS( SELECT 1 FROM b WHERE a.field2=b.field1 AND b.field2=100)

SELECT field1 FROM a WHERE field1 NOT IN( SELECT field2 FROM b)   
SELECT field1 FROM a WHERE NOT EXISTS( SELECT 1 FROM b WHERE b.field2=a.field1)

(17) 主、外键主要用于数据约束，sybase中创建主键时会自动创建索引，外键与索引无关，提高性能必须再建索引。

(18) char类型的字段不建索引比int类型的字段不建索引更糟糕。建索引后性能只稍差一点。

(19) 使用count(\*)而不要使用count(column\_name)，避免使用count(DISTINCT column\_name)。

(20) 等号右边尽量不要使用字段名，如：

SELECT \* FROM tb WHERE field1 = field3

(21) 避免使用or条件，因为or不使用索引。

2.避免使用order by和group by字句。

因为使用这两个子句会占用大量的临时空间(tempspace),如果一定要使用，可用视图、人工生成临时表的方法来代替。

如果必须使用，先检查memory、tempdb的大小。

测试证明，特别要避免一个查询里既使用join又使用group by，速度会非常慢！

3.尽量少用子查询，特别是相关子查询。因为这样会导致效率下降。

一个列的标签同时在主查询和where子句中的查询中出现，那么很可能当主查询中的列值改变之后，子查询必须重新查询一次。查询嵌套层次越多，效率越低，因此应当尽量避免子查询。如果子查询不可避免，那么要在子查询中过滤掉尽可能多的行。

4．消除对大型表行数据的顺序存取

在 嵌套查询中，对表的顺序存取对查询效率可能产生致命的影响。

比如采用顺序存取策略，一个嵌套3层的查询，如果每层都查询1000行，那么这个查询就要查询 10亿行数据。

避免这种情况的主要方法就是对连接的列进行索引。

例如，两个表：学生表（学号、姓名、年龄……）和选课表（学号、课程号、成绩）。如果两个 表要做连接，就要在“学号”这个连接字段上建立索引。

还可以使用并集来避免顺序存取。尽管在所有的检查列上都有索引，但某些形式的where子句强迫优化器使用顺序存取。

下面的查询将强迫对orders表执行顺序操作：

SELECT ＊ FROM orders WHERE (customer\_num=104 AND order\_num>1001) OR order\_num=1008

虽然在customer\_num和order\_num上建有索引，但是在上面的语句中优化器还是使用顺序存取路径扫描整个表。因为这个语句要检索的是分离的行的集合，所以应该改为如下语句：

SELECT ＊ FROM orders WHERE customer\_num=104 AND order\_num>1001

UNION

SELECT ＊ FROM orders WHERE order\_num=1008

这样就能利用索引路径处理查询。

5．避免困难的正规表达式

MATCHES和LIKE关键字支持通配符匹配，技术上叫正规表达式。但这种匹配特别耗费时间。例如：SELECT ＊ FROM customer WHERE zipcode LIKE “98\_ \_ \_”

即使在zipcode字段上建立了索引，在这种情况下也还是采用顺序扫描的方式。如果把语句改为SELECT ＊ FROM customer WHERE zipcode >“98000”，在执行查询时就会利用索引来查询，显然会大大提高速度。

另外，还要避免非开始的子串。例如语句：SELECT ＊ FROM customer WHERE zipcode[2，3] >“80”，在where子句中采用了非开始子串，因而这个语句也不会使用索引。

6．使用临时表加速查询

把表的一个子集进行排序并创建临时表，有时能加速查询。它有助于避免多重排序操作，而且在其他方面还能简化优化器的工作。例如：

SELECT cust.name，rcvbles.balance，……other COLUMNS

FROM cust，rcvbles

WHERE cust.customer\_id = rcvlbes.customer\_id

AND rcvblls.balance>0

AND cust.postcode>“98000”

ORDER BY cust.name

如果这个查询要被执行多次而不止一次，可以把所有未付款的客户找出来放在一个临时文件中，并按客户的名字进行排序：

SELECT cust.name，rcvbles.balance，……other COLUMNS

FROM cust，rcvbles

WHERE cust.customer\_id = rcvlbes.customer\_id

AND rcvblls.balance>;0

ORDER BY cust.name

INTO TEMP cust\_with\_balance

然后以下面的方式在临时表中查询：

SELECT ＊ FROM cust\_with\_balance

WHERE postcode>“98000”

临时表中的行要比主表中的行少，而且物理顺序就是所要求的顺序，减少了磁盘I/O，所以查询工作量可以得到大幅减少。

注意：临时表创建后不会反映主表的修改。在主表中数据频繁修改的情况下，注意不要丢失数据。

7．用排序来取代非顺序存取

非顺序磁盘存取是最慢的操作，表现在磁盘存取臂的来回移动。SQL语句隐藏了这一情况，使得我们在写应用程序时很容易写出要求存取大量非顺序页的查询。