

SQL优化

1. 应尽量避免全表扫描，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。例如：

```
SELECT id from t WHERE num IS NULL;
```

- 最好不要给数据库留 `NULL`，尽可能的使用 `NOT NULL` 填充数据库。
 - 备注、描述、评论之类的可以设置为 `NULL`，其他最好不要使用 `NULL`。
 - `NULL` 也会占用空间，如 `CHAR(100)`，不管是否插入值（`NULL` 也包含在内），都占用 100 个字符空间。但如果是 `VARCHAR` 这样的变长字段，`NULL` 就不占用空间。
3. 应尽量避免在 `where` 子句中使用 `!=` 或 `<>` 操作符，否则将使得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，如果一个字段有索引，一个字段没有索引，将导致引擎放弃使用索引而进行全表扫描，如：

```
SELCET id FROM t WHERE num=10 or name="admin"
```

可使用如下替代：

```
SELECT id FROM t WHERE num=10
UNION ALL
SELCET id FROM t WHERE name='admin';
```

5. 慎用 `in` 和 `not in`，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;
```

- 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);
```

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 "=" 左边 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';  
select id from t where datediff(day,createdate,'2005-11-30') = 0;
```

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';
```

10. 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。

11. 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：

```
create table #t(...);
```

12. Update 语句，如果只更改 1、2 个字段，**不要 Update 全部字段**，否则频繁调用会引起明显的性能消耗，同时带来大量日志。

13. 对于多张大数据量（这里几百条就算大了）的表 JOIN，要先分页再 JOIN，否则逻辑读会很高，性能很差。

14. `select count(*) from table;` 这样不带任何条件的 count 会引起全表扫描，并且没有任何业务意义，一定要杜绝。

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。

20. 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。

21. 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）。

22. 避免频繁创建和删除临时表，以减少系统表资源的消耗。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。