**1.检查索引**

在SQL语句的WHERE和JOIN部分中用到的所有字段上，都应该加上索引。

**2.限制工作数据集的大小**

检查那些SELECT语句中用到的表，看看你是否可以应用WHERE子句进行过滤。一个典型的例子是，当表中只有几千行记录时，一个查询能够很好地执行。但随着应用程序的成长，查询慢了下来。解决方案或许非常简单，限制查询来查看当前月的数据即可。当你的查询语句带有子查询时，注意在子查询的内部语句上使用过滤，而不是在外部语句上。

**3.只选择你需要的字段**

额外的字段通常会增加返回数据的纹理，从而导致更多的数据被返回到SQL客户端。另外：

•使用带有报告和分析功能的应用程序时，有时报告性能低是因为报告工具必须对收到的、带有详细形式的数据做聚合操作。  
•偶尔查询也可能运行地足够快，但你的问题可能是一个网络相关的问题，因为大量的详细数据通过网络发送到报告服务器。  
•当使用一个面向列的DBMS时，只有你选择的列会从磁盘读取。在你的查询中包含的列越少，IO开销就越小。

**4.移除不必要的表**

移除不必要的表的原因，和移除查询语句中不需要的字段的原因一致。

编写SQL语句是一个过程，通常需要大量编写和测试SQL语句的迭代过程。在开发过程中，你可能将表添加到查询中，而这对于SQL代码返回的数据可 能不会有任何影响。一旦SQL运行正确，我发现许多人不会回顾他们的脚本，不会删除那些对最终的返回数据没有任何影响和作用的表。通过移除与那些不必要表 的JOINS操作，你减少了大量数据库必须执行的流程。有时，就像移除列一样，你会发现你减少的数据又通过数据库返回来了。

**5.移除外部连接查询**

这说起来容易做起来难，它取决于改变表的内容有多大的影响。一个解决办法是通过在两个表的行中放置占位符来删除OUTER JOINS操作。假设你有以下的表，它们通过定义OUTER JOINS来确保返回所有的数据:

| **customer\_id** | **customer\_name** |
| --- | --- |
| 1 | John Doe |
| 2 | Mary Jane |
| 3 | Peter Pan |
| 4 | Joe Soap |

| **customer\_id** | **sales\_person** |
| --- | --- |
| NULL | Newbee Smith |
| 2 | Oldie Jones |
| 1 | Another Oldie |
| NULL | Greenhorn |

解决办法是在customer表的行中增加一个占位符，并更新sales表中的所有NULL值到占位符。

| **customer\_id** | **customer\_name** |
| --- | --- |
| 0 | NO CUSTOMER |
| 1 | John Doe |
| 2 | Mary Jane |
| 3 | Peter Pan |
| 4 | Joe Soap |

| **customer\_id** | **sales\_person** |
| --- | --- |
| 0 | Newbee Smith |
| 2 | Oldie Jones |
| 1 | Another Oldie |
| 0 | Greenhorn |

你不只是删除了对OUTER JOIN操作的依赖，同时标准化了没有客户的销售人员如何表示。其他开发人员不必编写额外语句，例如ISNULL(customer\_id, “No customer yet”)。

**6.删除JOIN和WHERE子句中的计算字段**

这是另外一个有时可能说起来容易做起来难的技巧，它取决于你更改表模式的权限大小。可以将连接语句中用到的计算字段作为一个新字段在表中创建。给出以下SQL语句：

FROM sales a

JOIN budget b ON    ((YEAR(a.sale\_date)\* 100) + MONTH(a.sale\_date)) = b.budget\_year\_month

在sales表中利用年和月增加一列，可以提高性能。更新后的SQL语句将如下:

SELECT \* FROM PRODUCTSFROM sales a

JOIN budget b ON    a.sale\_year\_month = b.budget\_year\_month