#### 1、建议使用预编译语句进行数据库操作

* 预编译语句可以重复使用这些计划，减少SQL编译所需要的时间，还可以解决动态SQL所带来的SQL注入的问题。
* 只传参数，比传递SQL语句更高效。
* 相同语句可以一次解析，多次使用，提高处理效率。

#### 2、避免数据类型的隐式转换

隐式转换会导致索引失效如:

select name,phone from customer where id = '111';

#### 3、充分利用表上已经存在的索引

避免使用双%号的查询条件。如：a like '%123%'，（如果无前置%,只有后置%，是可以用到列上的索引的）

一个SQL只能利用到复合索引中的一列进行范围查询。如：有 a,b,c 列的联合索引，在查询条件中有a列的范围查询，则在b，c列上的索引将不会被用到。

在定义联合索引时，如果a列要用到范围查找的话，就要把a列放到联合索引的右侧，使用left join或not exists来优化not in操作，因为not in也通常会使用索引失效。

#### 4、数据库设计时，应该要对以后扩展进行考虑

#### 5、程序连接不同的数据库使用不同的账号，禁止跨库查询

* 为数据库迁移和分库分表留出余地
* 降低业务耦合度
* 避免权限过大而产生的安全风险

#### 6、禁止使用 SELECT \* 必须使用 SELECT <字段列表> 查询

原因：

* 消耗更多的CPU和IO以网络带宽资源
* 无法使用覆盖索引
* 可减少表结构变更带来的影响

#### 7、禁止使用不含字段列表的 INSERT 语句

如：

insert into values ('a','b','c');

应使用：

insert into t(c1,c2,c3) values ('a','b','c');

#### 8、避免使用子查询，可以把子查询优化为join操作

通常子查询在in子句中，且子查询中为简单SQL(不包含 union、group by、order by、limit从句)时，才可以把子查询转化为关联查询进行优化。

子查询性能差的原因：

子查询的结果集无法使用索引，通常子查询的结果集会被存储到临时表中，不论是内存临时表还是磁盘临时表都不会存在索引，所以查询性能会受到一定的影响。特别是对于返回结果集比较大的子查询，其对查询性能的影响也就越大。

由于子查询会产生大量的临时表也没有索引，所以会消耗过多的CPU和IO资源，产生大量的慢查询。

#### 9、避免使用JOIN关联太多的表

对于MySQL来说，是存在关联缓存的，缓存的大小可以由join\_buffer\_size参数进行设置。

在MySQL中，对于同一个SQL多关联（join）一个表，就会多分配一个关联缓存，如果在一个SQL中关联的表越多，所占用的内存也就越大。

如果程序中大量的使用了多表关联的操作，同时join\_buffer\_size设置的也不合理的情况下，就容易造成服务器内存溢出的情况，就会影响到服务器数据库性能的稳定性。

同时对于关联操作来说，会产生临时表操作，影响查询效率，MySQL最多允许关联61个表，建议不超过5个。

#### 10、减少同数据库的交互次数

数据库更适合处理批量操作，合并多个相同的操作到一起，可以提高处理效率。

#### 11、对应同一列进行or判断时，使用in代替or

in的值不要超过500个，in操作可以更有效的利用索引，or大多数情况下很少能利用到索引。

#### 12、禁止使用order by rand()进行随机排序

order by rand()会把表中所有符合条件的数据装载到内存中，然后在内存中对所有数据根据随机生成的值进行排序，并且可能会对每一行都生成一个随机值，如果满足条件的数据集非常大，就会消耗大量的CPU和IO及内存资源。

推荐在程序中获取一个随机值，然后从数据库中获取数据的方式。

#### 13、WHERE从句中禁止对列进行函数转换和计算

对列进行函数转换或计算时会导致无法使用索引

不推荐：

where date(create\_time)='20190101'

推荐：

where create\_time >= '20190101' and create\_time < '20190102'

#### 14、在明显不会有重复值时使用UNION ALL而不是UNION

* UNION会把两个结果集的所有数据放到临时表中后再进行去重操作
* UNION ALL不会再对结果集进行去重操作

#### 15、拆分复杂的大SQL为多个小SQL

* 大SQL逻辑上比较复杂，需要占用大量CPU进行计算的SQL
* MySQL中，一个SQL只能使用一个CPU进行计算
* SQL拆分后可以通过并行执行来提高处理效率