1. **提高数据库性能的原则？（数据库连接，集中，分散等方面）**
2. 保持数据库连接的稳定性，减少交互。
3. 保持数据集中化，避免对远程数据的透明引用访问，避免对不同数据源数据访问。
4. 考虑解决方案细节之前，应该把握大局，着眼于最终结果而不是眼前细节。
5. 充分利用每次数据库访问，在每次访问中完成尽量多的工作，提取多段信息。
6. 尽量把条件逻辑放到SQL语句中，而不是SQL宿主语言中。
7. 慎用自定义函数，防止语句无法被基于成本的查询优化器优化。
8. 合理建立索引，提高查询效率。
9. **SQL 优化（针对查询优化器的隐性 SQL 优化）  
   例子：BMW  
   关于DISTINCT，嵌套 SQL，EXISTS 和 IN 的差异**

优化器借助关系理论提供的语义无误的原始查询进行有效的等价变换，在数据处理真正被执行的时候发生。

优化器的有效范围：

优化器需要借助数据库中找到的信息

能够进行数学意义上的等价变换

优化器考虑整体响应时间

优化器改善的是独立的查询

（如果是若干小查询，会各个优化；如果是一个大查询，会整体优化）

select distinct c.custname

from customers c

join orders o

on o.custid= c.custid

join orderdetailod

on od.ordid= o.ordid

join articles a

on a.artid= od.artid

where c.city= ‘Nanjing'

and a.artname= ‘BMW'

and o.ordered>= somefunc/\*函数，返回六个月前的具体日期\*/

古老的自然连接方式：

select distinct c.custname

from customers c,

orders o,

orderdetailod,

articles a

where c.city= ‘Nanjing'

and c.custid= o.custid

and o.ordid= od.ordid

and od.artid= a.artid

and a.artname= ‘BMW'

and o.ordered>= somefunc

摆脱distinct 关联子查询：o.custid需要有索引

select c.custname

from customers c

where c.city= ‘Nanjing'

and exists(select null

from orders o,

orderdetail od,

articles a

where a.artname= ‘BMW'

and a.artid= od.artid

and od.ordid= o.ordid

**and o.custid= c.custid**

and o.ordered>= somefunc)

非关联子查询：内层查询不在依赖外层查询，只需要执行一次；o.custid不需要有索引

select custname

from customers

where city = ‘Nanjing'

and custid in (select o.custid

from orders o,

orderdetailod,

articles a

where a.artname= ‘BMW'

and a.artid= od.artid

and od.ordid= o.ordid

and o.ordered>= somefunc)

exits适合内小外大的查询，in适合内大外小的查询

SQL优化：

1. 避免在最高层使用distinct
2. 针对分辨率最强的条件使用exist或in
3. 尽快剔除不需要的数据，查询后续阶段必须处理的数据量就越少，查询效率就越高
4. 所有影响聚合函数结果的条件都应在having子句中
5. 任何无关聚合条件都应该放在where子句中
6. 减少group by必须执行排序操作处理的数据量
7. **索引。目的：提高查询效率。  
   结构：B 树  
   为什么不使用索引？六点（不要标题党）**

什么时候使用B树索引：

仅当要通过索引访问表中很少一部分行

如果要处理表中多行，而且可以使用索引而不用表

1. 我们在使用B+树索引，谓词中没有使用索引的最前列

T，T(X,Y)上有索引，做SELECT \* FROM T WHERE Y=5

1. 使用SELECT COUNT(\*) FROM T，而且T上有索引，但是优化器仍然全表扫描
2. 对于一个有索引的列作出**函数查询，函数和隐式类型转换会使索引无法发挥作用**

Select \* from t where f(indexed\_col) = value

1. 隐形函数查询
2. 此时如果用了索引，实际反而会更慢
3. 没有正确的统计信息，造成CBO无法做出正确的选择
4. **数据库的物理组织：  
   堆文件**

堆文件就是一般的表，获取表中的数据是按命中率来得到的。没有明确的先后之分，在进行全表扫描时，并不是先插入的数据就先获取。数据的存放是随机的，也可以根据可用空闲的空间来决定。

**IOT 索引组织表**

当索引中增加额外的字段（一个或多个，它们本身与实际搜索条件无关，但包含查询所需的数据），能提高某个频繁运行的查询的速度。

IOT允许在主键索引中存储表中所有数据，表就是索引

对IOT表插入的效率也许低于堆文件

IOT的用途：全索引表，代码查找表，高频度的一组关联数据查询

IOT最大优点：记录是排序的

**分区**

分区能够提高并发性和并行性

从而增强系统架构的可伸缩性

对分区表进行查询，当数据按分区键均匀分布时，收益最大。

**问题：它们分别是如何提高查询效率的（考试可能考其中的一个）**

1. **表结构设计。  
   例子：如何把一棵树存到二维表中？  
   一，<id, pid>方式；**

**二，物化路径；**

**三，嵌套集合**

**问题：这三种方式的性能区别**

自顶向下查询

效率：邻接模式>物化路径>嵌套集合

自底向上查询

效率：邻接模式>物化路径>嵌套集合