数据库的优化方法有很多种，在应用层来说，主要是基于索引的优化。

建立必要的索引  
这次传授的降龙十八掌，总纲只有一句话：建立必要的索引，这就是后面降龙十八掌的内功基础。

1、表的主键、外键必须有索引；

2、数据量超过300的表应该有索引；

3、经常与其他表进行连接的表，在连接字段上应该建立索引；

4、经常出现在Where子句中的字段，特别是大表的字段，应该建立索引；

5、索引应该建在选择性高的字段上；

6、索引应该建在小字段上，对于大的文本字段甚至超长字段，不要建索引；

7、复合索引的建立需要进行仔细分析；尽量考虑用单字段索引代替：

   A、正确选择复合索引中的主列字段，一般是选择性较好的字段；

   B、复合索引的几个字段是否经常同时以AND方式出现在Where子句中？单字段查询是否极少甚至没有？如果是，则可以建立复合索引；否则考虑单字段索引；

   C、如果复合索引中包含的字段经常单独出现在Where子句中，则分解为多个单字段索引；

   D、如果复合索引所包含的字段超过3个，那么仔细考虑其必要性，考虑减少复合的字段；

   E、如果既有单字段索引，又有这几个字段上的复合索引，一般可以删除复合索引；

8、频繁进行数据操作的表，不要建立太多的索引；

9、删除无用的索引，避免对执行计划造成负面影响；

 索引的建立必须慎重，对每个索引的必要性都应该经过仔细分析，要有建立的依据。因为太多的索引与不充分、不正确的索引对性能都毫无益处：在表上建立的每个索引都会增加存储开销，索引对于插入、删除、更新操作也会增加处理上的开销。另外，过多的复合索引，在有单字段索引的情况下，一般都是没有存在价值的；相反，还会降低数据增加删除时的性能，特别是对频繁更新的表来说，负面影响更大。

第一掌 避免对列的操作

任何对列的操作都可能导致全表扫描，这里所谓的操作包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等式的右边，甚至去掉函数。

例1：下列SQL条件语句中的列都建有恰当的索引，但30万行数据情况下执行速度却非常慢：

select \* from record where substrb(CardNo,1,4)='5378'(13秒)

select \* from record where amount/30< 1000（11秒）

select \* from record where to\_char(ActionTime,'yyyymmdd')='19991201'（10秒）

由于where子句中对列的任何操作结果都是在SQL运行时逐行计算得到的，因此它不得不进行表扫描，而没有使用该列上面的索引；如果这些结果在查询编译时就能得到，那么就可以被SQL优化器优化，使用索引，避免表扫描，因此将SQL重写如下：

select \* from record where CardNo like '5378%'（< 1秒）

select \* from record where amount < 1000\*30（< 1秒）

select \* from record where ActionTime= to\_date ('19991201' ,'yyyymmdd')（< 1秒）

差别是很明显的！

第二掌 避免不必要的类型转换

需要注意的是，尽量避免潜在的数据类型转换。如将字符型数据与数值型数据比较，ORACLE会自动将字符型用to\_number()函数进行转换，从而导致全表扫描。

例2：表tab1中的列col1是字符型（char)，则以下语句存在类型转换：  
select col1,col2 from tab1 where col1>10，

应该写为： select col1,col2 from tab1 where col1>'10'。

第三掌 增加查询的范围限制

增加查询的范围限制，避免全范围的搜索。

例3：以下查询表record 中时间ActionTime小于2001年3月1日的数据：  
select \* from record where ActionTime < to\_date ('20010301' ,'yyyymm')  
查询计划表明，上面的查询对表进行全表扫描，如果我们知道表中的最早的数据为2001年1月1日，那么，可以增加一个最小时间，使查询在一个完整的范围之内。修改如下：

select \* from record where ActionTime < to\_date ('20010301' ,'yyyymm') and   ActionTime > to\_date ('20010101' ,'yyyymm')

后一种SQL语句将利用上ActionTime字段上的索引，从而提高查询效率。把'20010301'换成一个变量，根据取值的机率，可以有一半以上的 机会提高效率。同理，对于大于某个值的查询，如果知道当前可能的最大值，也可以在Where子句中加上 “AND 列名<MAX(最大值)”。

第四掌 尽量去掉"IN"、"OR"

含有"IN"、"OR"的Where子句常会使用工作表，使索引失效；如果不产生大量重复值，可以考虑把子句拆开；拆开的子句中应该包含索引。

例4：select count(\*) from stuff where id\_no in('0','1')（23秒）

可以考虑将or子句分开：  
select count(\*) from stuff where id\_no='0'  
select count(\*) from stuff where id\_no='1'  
然后再做一个简单的加法，与原来的SQL语句相比，查询速度更快。

第五掌 尽量去掉 "<>"（不等于）

尽量去掉 "<>"，避免全表扫描，如果数据是枚举值，且取值范围固定，则修改为"OR"方式。

例5：

UPDATE SERVICEINFO SET STATE=0 WHERE STATE<>0;

以 上语句由于其中包含了"<>"，执行计划中用了全表扫描（TABLE ACCESSFULL），没有用到state字段上的索引。实际应用中，由于业务逻辑的限制，字段state为枚举值，只能等于0，1或2，而且，值等于=1，2的很少，因此可以去掉"<>"，利用索引来提高效率。

修改为：

UPDATE SERVICEINFO SET STATE=0 WHERE STATE = 1 OR STATE = 2 。进一步的修改可以参考第4种方法。

第六掌 去掉Where子句中的IS NULL和IS NOT NULL

Where字句中的IS NULL和IS NOT NULL将不会使用索引而是进行全表搜索，因此需要通过改变查询方式，分情况讨论等方法，去掉Where子句中的IS NULL和IS NOT NULL。

第七掌 索引提高数据分布不均匀时查询效率

索引的选择性低，但数据的值分布差异很大时，仍然可以利用索引提高效率。

A、数据分布不均匀的特殊情况下，选择性不高的索引也要创建。

表 ServiceInfo中数据量很大，假设有一百万行，其中有一个字段DisposalCourseFlag，取值范围为枚举值：[0，1，2，3，4，5，6，7]。按照前面说的索引建立的规则，“选择性不高的字段不应该建立索引，该字段只有8种取值，索引值的重复率很高，索引选择性明显很低，因此不建索引。然而，由于该字段上数据值的分布情况非常特殊，具体如下表：

取值范围                  1~5        6        7

占总数据量的百分比        1%        98%        1%

而且，常用的查询中，查询DisposalCourseFlag<6 的情况既多又频繁。毫无疑问，如果能够建立索引，并且被应用，那么将大大提高这种情况的查询效率。因此，我们需要在该字段上建立索引。

第八掌 利用HINT强制指定索引

在ORACLE优化器无法用上合理索引的情况下，利用HINT强制指定索引。

第九掌 屏蔽无用索引

继续上面8的例子，由于实际查询中，还有涉及到DisposalCourseFlag=6的查询，而此时如果用上该字段上的索引，将是非常不明智的，效率也极低。因此这种情况下，我们需要用特殊的方法屏蔽该索引，以便ORACLE选择其他字段上的索引。比如，如果字段为数值型的就在表达式的字段名后，添加“+ 0”，为字符型的就并上空串：“||""”

//屏蔽无用索引

如： select \* from serviceinfo where DisposalCourseFlag+ 0 = 6 and workNo = '36' 。

不过，不要把该用的索引屏蔽掉了，否则同样会产生低效率的全表扫描。

第十掌 分解复杂查询，用常量代替变量

对于复杂的Where条件组合，Where中含有多个带索引的字段，考虑用IF语句分情况进行讨论；同时，去掉不必要的外来参数条件，减低复杂度，以便在不同情况下用不同字段上的索引。

继续上面9的例子，对于包含

Where (DisposalCourseFlag < v\_DisPosalCourseFlag) or(v\_DisPosalCourseFlag is null) and....的查询，(这里v\_DisPosalCourseFlag为一个输入变量，取值范围可能为[NULL，0，1，2，3，4，5，6，7])，可以考虑分情况用IF语句进行讨论，类似：

IF v\_DisPosalCourseFlag =1 THEN

Where DisposalCourseFlag = 1 and ....

ELS IF v\_DisPosalCourseFlag =2 THEN

Where DisposalCourseFlag = 2 and ....

。。。。。。

第十一掌 like子句尽量前端匹配

因为like参数使用的非常频繁，因此如果能够对like子句使用索引，将很高的提高查询的效率。

例6：select \* from city where name like ‘%S%’

以上查询的执行计划用了全表扫描（TABLE ACCESS FULL），如果能够修改为：

select \* from city where name like ‘S%’

那么查询的执行计划将会变成（INDEX RANGE SCAN），成功的利用了name字段的索引。这意味着OracleSQL优化器会识别出用于索引的like子句，只要该查询的匹配端是具体值。因此我们在做like查询时，应该尽量使查询的匹配端是具体值，即使用like ‘S%’。

第十二掌 用Case语句合并多重扫描

我们常常必须基于多组数据表计算不同的聚集。例如下例通过三个独立查询：

例8：1）select count(\*) from emp where sal<1000;

     2）select count(\*) from emp where sal between 1000 and 5000;

     3）select count(\*) from emp where sal>5000;

这样我们需要进行三次全表查询，但是如果我们使用case语句：

select

count (sale when sal <1000 then 1 else null end) count\_poor,

count (sale when between 1000 and 5000 then 1 else null end) count\_blue,

count (sale when sal >5000 then 1 else null end) count\_poor

from emp;

这样查询的结果一样，但是执行计划只进行了一次全表查询。

第十三掌 使用nls\_date\_format

例9：

select \* from record where to\_char(ActionTime,'mm')='12'

这个查询的执行计划将是全表查询，如果我们改变nls\_date\_format，

SQL> alert session set nls\_date\_formate=’MM’;

现在重新修改上面的查询：

select \* from record where ActionTime='12'

这样就能使用actiontime上的索引了，它的执行计划将是（INDEX RANGE SCAN）。

第十四掌 使用基于函数的索引

前面谈到任何对列的操作都可能导致全表扫描，例如：

select \* from emp where substr(ename,1,2)=’SM’;

但是这种查询在客服系统又经常使用，我们可以创建一个带有substr函数的基于函数的索引，

create index emp\_ename\_substr on eemp ( substr(ename,1,2) );

这样在执行上面的查询语句时，这个基于函数的索引将排上用场，执行计划将是（INDEX RANGE SCAN）。Index range scan

第十五掌 基于函数的索引要求等式匹配

上面的例子中，我们创建了基于函数的索引，但是如果执行下面的查询：

select \* from emp where substr(ename,1,1)=’S’

得 到的执行计划将还是（TABLE ACCESSFULL），因为只有当数据列能够等式匹配时，基于函数的索引才能生效，这样对于这种索引的计划和维护的要求都很高。请注意，向表中添加索引是非常危险的操作，因为这将导致许多查询执行计划的变更。然而，如果我们使用基于函数的索引就不会产生这样的问题，因为Oracle只有在查询使用了匹配的内置函数时才会使用这种类型的索引。

第十六掌 使用分区索引

在用分析命令对分区索引进行分析时，每一个分区的数据值的范围信息会放入Oracle的数据字典中。Oracle可以利用这个信息来提取出那些只与SQL查询相关的数据分区。

例如，假设你已经定义了一个分区索引，并且某个SQL语句需要在一个索引分区中进行一次索引扫描。Oracle会仅仅访问这个索引分区，而且会在这个分区上调用一个此索引范围的快速全扫描。因为不需要访问整个索引，所以提高了查询的速度。

第十七掌 使用位图索引

位图索引可以从本质上提高使用了小于1000个唯一数据值的数据列的查询速度，因为在位图索引中进行的检索是在RAM中完成的，而且也总是比传统的B树索引的速度要快。对于那些少于1000个唯一数据值的数据列建立位图索引，可以使执行效率更快。

第十八掌 决定使用全表扫描还是使用索引

和所有的秘笈一样，最后一招都会又回到起点，最后我们来讨论一下是否需要建立索引，也许进行全表扫描更快。在大多数情况下，全表扫描可能会导致更多的物理磁盘输入输出，但是全表扫描有时又可能会因为高度并行化的存在而执行的更快。如果查询的表完全没有顺序，那么一个要返回记录数小于10％的查询可能会读取表中大部分的数据块，这样使用索引会使查询效率提高很多。但是如果表非常有顺序，那么如果查询的记录数大于40％时，可能使用全表扫描更快。因此，有一个索引范围扫描的总体原则是：

1）对于原始排序的表 仅读取少于表记录数40％的查询应该使用索引范围扫描。反之，读取记录数目多于表记录数的40％的查询应该使用全表扫描。

2）对于未排序的表    仅读取少于表记录数7％的查询应该使用索引范围扫描。反之，读取记录数目多于表记录数的7％的查询应该使用全表扫描。

总结

以上的招式，是完全可以相互结合同时运用的。而且各种方法之间相互影响，紧密联系。这种联系既存在一致性，也可能带来冲突，当冲突发生时，需要根据实际情况进行选择，没有固定的模式。最后决定SQL优化功力的因素就是对ORACLE内功的掌握程度了。

另外，值得注意的是：随着时间的推移和数据的累计与变化，ORACLE对SQL语句的执行计划也会改变，比如：基于代价的优化方法，随着数据量的增大，优化器可能错误的不选择索引而采用全表扫描。这种情况可能是因为统计信息已经过时，在数据量变化很大后没有及时分析表；但如果对表进行分析之后，仍然没有用上合理的索引，那么就有必要对SQL语句用HINT提示，强制用合理的索引。但这种HINT提示也不能滥用，因为这种方法过于复杂，缺乏通用性和应变能力，同时也增加了维护上的代价；相对来说，基于函数右移、去掉“IN ，OR ，<> ，IS NOT NULL”、分解复杂的SQL语句等等方法，却是“放之四海皆准”的，可以放心大胆的使用。

同时，优化也不是“一劳永逸”的，必须随着情况的改变进行相应的调整。当数据库设计发生变化，包括更改表结构：字段和索引的增加、删除或改名等；业务逻辑发生变化：如查询方式、取值范围发生改变等等。在这种情况下，也必须对原有的优化进行调整，以适应效率上的需求。