SQL 优化

1. 对查询进行优化，应尽量避免全表扫描，首先应考虑在 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，创建表时NULL是默认值，但大多数时候应该使用NOT NULL，或者使用一个特殊的值，如0，-1作为默认值。
3. 应尽量避免在where子句中使用!=或<>操作符，MySQL只有对以下操作符才使用索引：<，<=，=，>，>=，BETWEEN，IN，以及某些时候的LIKE。
4. 应尽量避免在where子句中使用or来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，可以使用UNION合并查询。

   ```
   select id from t where num=10 union all select id from t where num = 20
   ```

5. in和not in也要慎用，否则会导致全表扫描，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
   ```

6. 下面的查询也将导致全表扫描👇

   ```
   select id from t where name like '%abc%';

   select id from t where name like '%abc';
   ```

   若要提高效率，可以考虑全文检索。而模糊查询中 `%` 在后面才能用到索引。

   ```
   select id from t where name like 'abc%';
   ```

7. 如果在where子句中使用参数，也会导致全表扫描。
8. 应尽量避免在where子句中对字段进行表达式操作，应尽量避免在where子句中对字段进行函数操作。
9. 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);
   ```

10. 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。
11. 应尽可能的避免更新clustered索引（聚簇索引）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的顺序的调整，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。
12. 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。

13. 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。

14. 不要使用 `*` 返回所有：

    ```
    select * from table_name;
    ```

    用具体的字段列表代替 `*` ，不要返回用不到的任何字段。

15. 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。

16. 使用表的别名(Alias)：当在SQL语句中连接多个表时，请使用表的别名并把别名前缀于每个Column上。这样一来，就可以减少解析的时间并减少那些由Column歧义引起的语法错误。

17. 使用"临时表"暂存中间结果：

    简化SQL语句的重要方法就是采用临时表暂存中间结果，但是临时表的好处远远不止这些，将临时结果暂存在临时表，后面的查询就在tempdb中了，这可以避免程序中多次扫描主表，也大大减少了程序执行中"共享锁"阻塞"更新锁"，减少了阻塞，提高了并发性能。

18. 一些SQL查询语句应加上nolock，读、写是会相互阻塞的，为了提高并发性能，对于一些查询，可以加上nolock，这样读的时候可以允许写，但缺点是可能读到未提交的脏数据。

使用nolock有3条原则：

- 查询的结果用于"插、删、改"的不能加nolock；
- 查询的表属于频繁发生页分裂的，慎用nolock ；
- 能采用临时表提高并发性能的，不要用nolock。

19. 不要有超过5个以上的表连接（JOIN），考虑使用临时表或表变量存放中间结果。少用子查询，视图嵌套不要过深，一般视图嵌套不要超过2个为宜。

20. 将需要查询的结果预先计算好放在表中，查询的时候再Select。这在SQL7.0以前是最重要的手段，例如医院的住院费计算。

21. 用OR的字句可以分解成多个查询，并且通过UNION 连接多个查询。他们的速度只同是否使用索引有关，如果查询需要用到联合索引，用 UNION all 执行的效率更高。多个OR的字句没有用到索引，改写成UNION的形式再试图与索引匹配。一个关键的问题是否用到索引。

22. 在IN后面值的列表中，将出现最频繁的值放在最前面，出现得最少的放在最后面，减少判断的次数。

23. 尽量将数据的处理工作放在服务器上，减少网络的开销，如使用存储过程。

    存储过程是编译好、优化过、并且被组织到一个执行规划里、且存储在数据库中的SQL语句，是控制流语言的集合，速度当然快。反复执行的动态SQL，可以使用临时存储过程，该过程（临时表）被放在Tempdb中。

24. 当服务器的内存够多时，配制线程数量 = 最大连接数+5，这样能发挥最大的效率；否则使用 配制线程数量小于最大连接数启用SQL SERVER的线程池来解决，如果还是数量等于最大连接数+5，严重的损害服务器的性能。

25. 查询的关联同写的顺序：

    ```
    select a.personMemberID, * from chineseresume a,personmember b
    where personMemberID = b.referenceid and a.personMemberID = 'JCNPRH39681' （A = B ,B = '号码'）

    select a.personMemberID, * from chineseresume a,personmember b
    where a.personMemberID = b.referenceid
    and a.personMemberID = 'JCNPRH39681'
    and b.referenceid = 'JCNPRH39681' （A = B ,B = '号码'， A = '号码'）
    ```

```
select a.personMemberID, * from chineseresume a,personmember b
where b.referenceid = 'JCNPRH39681' and a.personMemberID = 'JCNPRH39681' （B = '号码'， A = '号码'）
```

26. 尽量使用 exists 代替 select count(1) 来判断是否存在记录，count 函数只有在统计表中所有行数时使用，而且 count(1) 比 count(*) 更有效率。

27. 尽量使用">="，不要使用">"。

28. 索引的使用规范：

    - 索引的创建要与应用结合考虑，建议大的OLTP表不要超过6个索引；
    - 尽可能的使用索引字段作为查询条件，尤其是聚簇索引，必要时可以通过index index_name来强制指定索引；
    - 避免对大表查询时进行table scan，必要时考虑新建索引；
    - 在使用索引字段作为条件时，如果该索引是联合索引，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；
    - 要注意索引的维护，周期性重建索引，重新编译存储过程。

29. 下列SQL条件语句中的列都建有恰当的索引，但执行速度却非常慢：

```
SELECT * FROM record WHERE substrINg(card_no,1,4)='5378' (13秒)

SELECT * FROM record WHERE amount/30< 1000 （11秒）

SELECT * FROM record WHERE convert(char(10),date,112)='19991201' （10秒）
```

    分析：

    WHERE子句中对列的任何操作结果都是在SQL运行时逐列计算得到的，因此它不得不进行表搜索，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，避免表搜索，因此将SQL重写成下面这样👇

```
SELECT * FROM record WHERE card_no like '5378%' （< 1秒）

SELECT * FROM record WHERE amount< 1000*30 （< 1秒）

SELECT * FROM record WHERE date= '1999/12/01' （< 1秒）
```

30. 当有一批处理的插入或更新时，用批量插入或批量更新，绝不会一条条记录的去更新。

31. 在所有的存储过程中，能够用SQL语句的，不要用循环去实现。

    例如：列出上个月的每一天，会用connect by去递归查询一下，绝不会去用循环从上个月第一天到最后一天。

32. 选择最有效率的表名顺序（只在基于规则的优化器中有效）：

    Oracle的解析器按照从右到左的顺序处理FROM子句中的表名，FROM子句中写在最后的表（基础表 driving table）将被最先处理，在FROM子句中包含多个表的情况下，你必须选择记录条数最少的表作为基础表。

```
如果有3个以上的表连接查询，那就需要选择交叉表（intersection table）作为基础表，交叉表是指那个被其他表所引用的表。
```

33. 提高GROUP BY语句的效率，可以通过将不需要的记录在GROUP BY之前过滤掉。下面两个查询返回相同结果，但第二个明显就快了许多。

```
```mysql
# 低效
```
```

```
SELECT JOB , AVG(SAL)
FROM EMP
GROUP BY JOB
HAVING JOB =’PRESIDENT’ OR JOB =’MANAGER’

# 高效
SELECT JOB , AVG(SAL)
FROM EMP
WHERE JOB =’PRESIDENT’ OR JOB =’MANAGER’
GROUP BY JOB
```

34. SQL语句用大写，因为Oracle总是先解析SQL语句，把小写的字母转换成大写的再执行。

35. 别名的使用，别名是大型数据库的应用技巧，就是表名、列名在查询中以一个字母为别名，查询速度要比建连接表快1.5倍。

36. 避免死锁，在你的存储过程和触发器中访问同一个表时总是以相同的顺序；事务应经可能地缩短，在一个事务中应尽可能减少涉及到的数据量；永远不要在事务中等待用户输入。

37. 避免使用临时表，除非却有需要，否则应尽量避免使用临时表，相反，可以使用表变量代替；大多数时候(99%)，表变量驻扎在内存中，因此速度比临时表更快，临时表驻扎在TempDb数据库中，因此临时表上的操作需要跨数据库通信，速度自然慢。

38. 最好不要使用触发器：

    - 触发一个触发器，执行一个触发器事件本身就是一个耗费资源的过程；
    - 如果能够使用约束实现的，尽量不要使用触发器；
    - 不要为不同的触发事件(Insert，Update和Delete)使用相同的触发器；
    - 不要在触发器中使用事务型代码。

39. 索引创建规则：

    - 表的主键、外键必须有索引；
    - 数据量超过300的表应该有索引；
    - 经常与其他表进行连接的表，在连接字段上应该建立索引；
    - 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；
    - 索引应该建在选择性高的字段上；
    - 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；
    - 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；
    - 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；
    - 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；
    - 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；
    - 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；
    - 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；
    - 频繁进行数据操作的表，不要建立太多的索引；
    - 删除无用的索引，避免对执行计划造成负面影响；
    - 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。
    - 尽量不要对数据库中某个含有大量重复的值的字段建立索引。

40. MySQL查询优化总结：

使用慢查询日志去发现慢查询，使用执行计划去判断查询是否正常运行，总是去测试你的查询看看是否他们运行在最佳状态下。

久而久之性能总会变化，避免在整个表上使用count(*)，它可能锁住整张表，使查询保持一致以便后续相似的查询可以使用查询缓存，在适当的情形下使用GROUP BY而不是DISTINCT，在WHERE、GROUP BY和ORDER BY子句中使用有索引的列，保持索引简单，不在多个索引中包含同一个列。

有时候MySQL会使用错误的索引，对于这种情况使用USE INDEX，检查使用SQL_MODE=STRICT的问题，对于记录数小于5的索引字段，在UNION的时候使用LIMIT不是是用OR。

为了避免在更新前SELECT，使用INSERT ON DUPLICATE KEY或者INSERT IGNORE，不要用UPDATE去实现，不要使用MAX，使用索引字段和ORDER BY子句，LIMIT M，N实际上可以减缓查询在某些情况下，有节制地使用，在WHERE子句中使用UNION代替子查询，在重新启动的MySQL，记得来温暖你的数据库，以确保数据在内存和查询速度快，考虑持久连接，而不是多个连接，以减少开销。

基准查询，包括使用服务器上的负载，有时一个简单的查询可以影响其他查询，当负载增加在服务器上，使用SHOW PROCESSLIST查看慢的和有问题的查询，在开发环境中产生的镜像数据中测试的所有可疑的查询。

41. MySQL 备份过程：
    - 从二级复制服务器上进行备份；
    - 在进行备份期间停止复制，以避免在数据依赖和外键约束上出现不一致；
    - 彻底停止MySQL，从数据库文件进行备份；
    - 如果使用MySQL dump进行备份，请同时备份二进制日志文件 – 确保复制没有中断；
    - 不要信任LVM快照，这很可能产生数据不一致，将来会给你带来麻烦；
    - 为了更容易进行单表恢复，以表为单位导出数据——如果数据是与其他表隔离的。
    - 当使用mysqldump时请使用–opt；
    - 在备份之前检查和优化表；
    - 为了更快的进行导入，在导入时临时禁用外键约束。；
    - 为了更快的进行导入，在导入时临时禁用唯一性检测；
    - 在每一次备份后计算数据库，表以及索引的尺寸，以便更够监控数据尺寸的增长；
    - 通过自动调度脚本监控复制实例的错误和延迟；
    - 定期执行备份。
42. 查询缓冲并不自动处理空格，因此，在写SQL语句时，应尽量减少空格的使用，尤其是在SQL首和尾的空格（因为查询缓冲并不自动截取首尾空格）。
43. member 用mid做标准进行分表方便查询么？一般的业务需求中基本上都是以username为查询依据，正常应当是username做hash取模来分表。

    而分表的话MySQL的partition功能就是干这个的，对代码是透明的；在代码层面去实现貌似是不合理的。
44. 我们应该为数据库里的每张表都设置一个ID做为其主键，而且最好的是一个INT型的（推荐使用UNSIGNED），并设置上自动增加的AUTO_INCREMENT标志。
45. 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。
46. MySQL查询可以启用高速查询缓存。这是提高数据库性能的有效MySQL优化方法之一。当同一个查询被执行多次时，从缓存中提取数据和直接从数据库中返回数据快很多。

47. EXPLAIN SELECT查询用来跟踪查看效果：

    使用EXPLAIN关键字可以让你知道MySQL是如何处理你的SQL语句的。这可以帮你分析你的查询语句或是表结构的性能瓶颈。EXPLAIN的查询结果还会告诉你你的索引主键被如何利用的，你的数据表是如何被搜索和排序的。

48. 当只要一行数据时使用LIMIT 1 ：

    当查询表的有些时候，已经知道结果只会有一条结果，但因为可能需要去fetch游标，或是也许会去检查返回的记录数。

    在这种情况下，加上LIMIT 1可以增加性能。这样一来，MySQL数据库引擎会在找到一条数据后停止搜索，而不是继续往后查少下一条符合记录的数据。

49. 选择表合适存储引擎：

    - myisam：应用时以读和插入操作为主，只有少量的更新和删除，并且对事务的完整性，并发性要求不是很高的。
    - InnoDB：事务处理，以及并发条件下要求数据的一致性。除了插入和查询外，包括很多的更新和删除。（InnoDB有效地降低删除和更新导致的锁定）。

      对于支持事务的InnoDB类型的表来说，影响速度的主要原因是AUTOCOMMIT默认设置是打开的，而且程序没有显式调用BEGIN 开始事务，导致每插入一条都自动提交，严重影响了速度。可以在执行SQL前调用begin，多条SQL形成一个事物（即使autocommit打开也可以），将大大提高性能。

50. 优化表的数据类型，选择合适的数据类型：

    原则：更小通常更好，简单就好，所有字段都得有默认值，尽量避免null。

    例如：数据库表设计时候更小的占磁盘空间尽可能使用更小的整数类型。(mediumint就比int更合适)

    比如时间字段：datetime和timestamp，datetime占用8个字节，而timestamp占用4个字节，只用了一半，而timestamp表示的范围是1970—2037适合做更新时间

    MySQL可以很好的支持大数据量的存取，但是一般说来，数据库中的表越小，在它上面执行的查询也就会越快。

    因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

    例如：在定义邮政编码这个字段时，如果将其设置为CHAR(255)，显然给数据库增加了不必要的空间。甚至使用VARCHAR这种类型也是多余的，因为CHAR(6)就可以很好的完成任务了。

    同样的，如果可以的话，我们应该使用MEDIUMINT而不是BIGIN来定义整型字段，应该尽量把字段设置为NOT NULL，这样在将来执行查询的时候，数据库不用去比较NULL值。

    对于某些文本字段，例如“省份”或者“性别”，我们可以将它们定义为ENUM类型。因为在MySQL中，ENUM类型被当作数值型数据来处理，而数值型数据被处理起来的速度要比文本类型快得多。这样，我们又可以提高数据库的性能。

51. 字符串数据类型：char，varchar，text选择区别。

    - char长度固定， 即每条数据占用等长字节空间；适合用在身份证号码、手机号码等定。
    - varchar可变长度，可以设置最大长度；适合用在长度可变的属性。
    - text不设置长度， 当不知道属性的最大长度时，适合用text。

52. 任何对列的操作都将导致表扫描，它包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等号右边。