**《mysql设计规范》**

1. **数据结构设计**：逻辑设计 –> 物理设计
2. **实际工作**中：逻辑设计 + 物理设计
3. **物理设计**：表名，字段名，字段类型
4. 磁盘IO和操作系统类型，对mysql的性能是非常大的

**一. 数据库命名规范**

1. 所有的数据库对象名称必须**使用小写字母并用下划线**表示，因为默认情况下，mysql对大小写敏感，mysql数据库本质上是linux系统下的一个文件，而linux系统是大小写敏感的
2. 所有数据库对象名称**禁止使用mysql保留关键字**
3. 数据库对象的命名要能做到见名知意，并且最好不要超过32个字符。太长不方便使用，并且会在传输时增加网络开销
4. **临时表**必须以tmp\_为前缀并以日期为后缀
5. **备份表**必须以bak\_为前缀并以日期为后缀
6. 所有存储相同数据的列名和列类型必须一致，比如user表中的id和order表中的user\_id

**二. 数据库基本设计规范**

1. **所有表必须使用Innodb存储引擎**   
   * 极少数特殊业务需求除外
   * Innodb引擎是5.6之后的默认存储引擎；mysql5.5之前使用Myisam(默认存储引擎)
   * Innodb优点：支持事务，行级锁，更好的恢复性，高并发下性能更好
2. **数据库和表的字符集统一使用UTF-8**   
   * 如果要存储一些如表情符号的，还需使用UTF-8的拓展字符集
   * 数据库，表，字段字符集一定要统一，统一字符集可以避免由于字符集转换产生的乱码
   * 在mysql中UTF-8字符集，汉字占3字节，ASCII码占1字节
3. **所有表和字段都需要添加注释**
4. **从一开始就进行数据字典的维护**   
   * 即数据库说明文档
5. **尽量控制单表数据量大小**，   
   * 建议控制在500万以内，虽然500万并不是mysql的数据库限制，但是会给修改表结构，备份，恢复带来很大困难。
   * 单表可存储数据量大小取决于存储设置和文件系统
   * 想减少单表数据量：历史数据归档(常见于日志表)，分库分表(常见于业务表)，分区表
   * 建议不要使用mysql分区表，因为分区表在物理上表现为多个文件，在逻辑上表现为一个表。如果一定要分区，请谨慎选择分区键，跨分区查询效率比查询大数据量的单表查询效率更低
   * 建议采物理分表的方式管理大数据，但是对应用程序的开发要求和复杂度更高
6. 尽量做到**冷热数据分离，减少表的宽度(字段数)**   
   * 减少磁盘IO，保证热数据的内存缓存命中率，更有效的利用缓存，避免读入无用的冷数据
   * 这样的话，就要对表的列进行拆分，将经常使用的列放到一个表中，可以避免过多的关联操作，也可以提高查询性能
7. **禁止在表中建立预留字段**   
   * 预留字段很难做到见名知义，预留字段无法确定存储的数据类型，后期如果修改字段类型，会对全表锁定，严重影响数据库的并发性
   * 对目前mysql来说，修改一个字段的成本要远远大于增加一个字段的成本
8. **禁止在数据库中存储图片，文件等二级制数据**   
   * 这类数据如果要存，就得使用blog或者text这样的大字段加以存储，会影响数据库的性能
   * 文件这种通常所占数据容量很大，会在短时间内造成数据库文件的快速增长，而数据库在读取数据时，会进行大量的随机IO操作，如果数据文件过大，IO操作会非常耗时，从而影响数据库性能
   * 正确做法是将这类数据存储在文件服务器中，而数据库只村存储地址信息
9. **禁止在线上做数据库压力测试**   
   * 会对正常业务造成影响，也会产生很多垃圾数据
   * 建议建立专门的压力测试数据库，进行测试，然后对比测试服务器和线上服务器的硬件环境，评估线上数据库的性能
10. **禁止从开发环境，测试环境直连生产环境数据库**

**三. 索引设计规范(Innodb中主键实质上是一个索引)**

1. **限制每张表上索引数量**，建议单表不超过5个索引。索引并不是越多越好，可以提高查询效率，但是会降低插入和更新的效率。甚至在一些情况下，还会降低查询效率，因为mysql优化器在选择如何优化查询时，会根据统计信息，对每一个可用索引来进行评估，以生成一个最好的执行计划，如果同时有很多索引都可以用于查询，就会增加mysql查询优化器生成查询计划的时间。
2. **每个Innodb表都必须有一个主键**。Innodb是一种索引索引组织表，是指数据存储的逻辑顺序和索引的顺序是相同，Innodb是按照主键索引的顺序来组织表的，因此，每个Innodb表都必须要有一个主键，如果我们没有指定主键，那么Innodb会优先选择表中第一个非空唯一索引来作为主键，如果没有这个索引，那么Innodb会自动生成一个占6字节的主键，而这个主键的性能并不是最好。
3. **不使用更新频繁的列作为主键，不使用多列联合主键**。因为Innodb是一种索引索引组织表，如果主键上的值频繁更新，就意味着数据存储的逻辑顺序频繁变动，必然会带来大量的IO操作，降低数据库性能。
4. **不要使用uuid，md5，hash，字符串列作为主键**。因为这种主键不能保证主键的值是顺序增长的，如果后来的主键值在已有主键值的中间段，那么这个主键插入的时候，会将所有主键值大于它的列都向后移。
5. **最好选择能保证值的顺序为顺序增长的列为主键**。并且数据不能重复，建议用mysql自增id建立主键
6. 面试问题1: **要在哪些列上建立索引?**   
   * 在select，delete，update的where从句中的列
   * 包含在order by，group by，distinct字段中的列
   * 多表join的关联列：mysql对关联操作的处理方式只有一种，那就是嵌套循环的关联方式，所以这种操作的性能对关联列上的索引的依赖性很大
7. 面试问题2: **复合索引，如何选择索引列的顺序?**   
   * 从左到右的顺序来使用的
   * 区分度(列中group by的数目和此列总行数的比值趋近于1)最高的列放在联合索引的最左侧
   * 在区分度差不多的情况下，尽量吧字段长度小的放在联合索引的最左侧，因为同样的行数，字段小的文件也小，读取时IO性能更优
   * 使用最频繁的列放在联合索引的左侧，这样的话，可以较少地建立索引就能满足需求
8. **避免建立冗余索引和重复索引**
9. **对于频繁的查询优先使用覆盖索引**   
   * 就是包含了所有查询字段的索引，这样可以避免Innodb表进行索引的二次查找，并可以把随机IO变为顺序IO提高查询效率
10. **尽量避免使用外键**   
    * mysql和别的数据库不同，会自动在外键上建立索引，会降低数据库的写性能
    * 建议不使用外键约束，但是一定要在表与表之间的关联键上建立索引，虽然外键是为了保证数据的完整性，但是最好在代码中去保证。

**四. 字段设计规范**

1. **优先选择符合存储需要的最小的数据类型**   
   * 尽量将字符串转化为数字类型存储：如将ip存储为数字：inet\_aton(‘255.255.255.255’) = 4294967295 ,反之， inet\_ntoa(4294967295) = ‘255.255.255.255’
   * 对于非负整型数据，优先使用无符号整型来存储，如：id,age,无符号相对于有符号，可以多出一倍的存储空间
   * mysql中，varchar(n)中n表示字符数而不是字节数
2. **避免使用text，blog来存储**字段，这种类型只能使用前缀索引，如果非要使用，建议将这种数据分离到单独的拓展表中
3. **避免使用enum类型**。枚举本身是一个字符串类型，但是内部确是用正数类型来存储的，所以最多可存储65535种不同的值，修改的话必须使用alter语句，直接修改元数据，有操作风险；order by效率低，必须转换并无法使用索引，禁止使用数值作为enum值，因为enum本身是索引顺序存储的，会造成逻辑混淆
4. **尽可能把所有列定义为not null**。   
   * 索引null列需要额外的空间来保存，占更多空间
   * 进行比较和计算时，对null值作特别的处理，可能造成索引失效
5. **禁止使用字符串来存储日期型数据**。   
   * 无法使用日期函数计算比较
   * 字符串存储要占更多的内存空间，datetime(8字节)和timestamp(本身是以int存储，占4字节,范围:1970-01-01 00:00:01到2038-01-19 03:14:07)
6. **财务相关数据，使用decimal类型** (精准浮点类型，在计算时不丢失精度)。

**五. SQL开发规范**

1. **建议使用预编译语句(prepareStatment)进行数据库操作**   
   * 可以同步执行预编译计划，减少预编译时间
   * 可以有效避免动态sql带来的SQL注入的问题
   * 只传参数，一次解析，多次使用，比传递sql语句更高效
2. **避免数据类型的隐式转换**   
   * 一般出现在where从句中，会导致索引失效，如：select id,name from user where id = ‘12’;
3. **充分利用已存在的索引**   
   * 避免使用双%的查询条件，不走索引
   * 一个SQL只能利用到复合索引中的一列进行范围查询
   * 使用left join或not exists来优化not in操作
4. **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
5. **禁止使用select \* 来查询，必须用字段名**   
   * 可能会消耗更多的cpu和IO以及网络资源
   * 无法使用覆盖索引
   * 可以减少表结构变更对已有程序的影响
6. **禁止使用不含字段列表的insert语句**。   
   * 可以减少表结构变更对已有程序的影响
7. **禁止使用子查询**   
   * 虽然可使sql可读性好，但是缺点远远大于优点
   * 子查询返回的结果集无法使用索引，结果集会被存储到一个临时表中，结果集越大性能越低
   * 把子查询优化为join操作，但是并不是所有的都可以优化为join，一般情况下，只有当子查询是在in字句中，并且子查询是一个简单的sql(不包含union，group by，order by，limit)才能转换为关联查询
8. **避免join过多的表**   
   * 每join一个表会占一部分内存(join\_buffer\_size)
   * 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
9. **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，提高处理效率
10. **使用in代替or**   
    * in的值不要超过500个
    * in 操作可以有效利用索引
11. **禁止使用order by rand()进行随机排序**   
    * 会把表中所有符合条件的数据装载到内存中进行排序
    * 会消耗大量的cpu和io及内存资源
    * 推荐在程序中获取随机值
12. **禁止在where从句中对列进行函数转换和计算**   
    * 导致无法使用相关列上的索引
    * where date(create\_time)=’20170901’ **写成** where create\_time >= ‘20170901’ and create\_time < ‘20170902’
13. **在明显不会有重复值时使用union all而不是union**   
    * union 会把所有数据放在临时表中后再进行去重操作，会多消耗内存，IO，网络资源
    * union all 不会再对结果集进行去重操作
14. **拆分复杂的大sql为多个小sql**   
    * 目前mysql中一个sql只能使用一个cpu计算，不支持多cpu并行计算
    * sql拆分后可以通过并行执行来提高处理效率

**六. 数据库操作行为规范**

* **主要面向手动操作数据库的行为**   
  1. **超过100万的批量写操作，要分批多次进行操作**   
     + 主从复制中：大批量操作可能会造成严重的主从延迟，因为当主库执行完成后，才会在从库执行
     + binlog日志为row格式时会产生大量的日志
     + 避免产生大量事务，产生阻塞，占满可用连接
  2. **对大表数据结构的修改一定要谨慎**   
     + 可能会造成严重的锁表操作，尤其是生产环境，是不能忍受的
     + 对于大表使用pt-online-schema-change修改表结构：
     + 首先会建立一个与原表结构相同的新表
     + 然后在新表上进行表结构的修改
     + 然后把原表中的数据复制到新表中，并且增加一些触发器，以便把原表中即时新增的数据也复制到新表中
     + 在行的所有数据复制完成之后，会在原表上增加一个很准的时间锁，同时把新表命名为原表，把原表删掉
     + [实际上是把一个原子的DDL操作分解成多批次进行]
     + [避免大表修改产生的主从延迟问题]
     + [避免在对表字段进行修改时进行锁表]
  3. **禁止为程序使用的账号赋予super权限**   
     + 当数据库连接数达到最大限制时，允许1个有super权限的用户连接
     + super权限只能留给DBA处理问题的账号使用
  4. **对于程序连接数据库账号，遵循权限最小原则**   
     + 程序使用的数据库账号只能在一个DB下使用，不准跨库
     + 程序使用的账号原则上不准有drop权限