数据库命名规范

1. 数据库对象名称必须小写并使用下划线进行连接；
2. 命名禁止使用MySQL预留关键字；
3. 数据库对象命名要见名知意，最长不要超过32字符；
4. 临时表必须以tmp_为前缀，日期为后缀进行命名，备份表以bak_为前缀日期为后缀；
5. 所有存储相同数据的列名和列类型必须一致；

数据库基本设计规范

1. **所有表必须使用InnoDB为引擎，**（有特殊要求InnoDB无法满足的除外）；
2. **数据库和表的字符集统一使用UTF8；**统一字符集避免字符集转换乱码，字符集有需要存储emoji表情需要使用utf8mb4；
3. **所有表和字段都需要注释；**
4. **尽量控制单表数据量大小，建议在500万以内；**过大造成备份，恢复有很大的问题；
5. **谨慎使用MySQL分区表；**物理上表现多个表，逻辑上一个表，谨慎选择分区键，跨区查询效率低，建议物理分表管理数据；
6. **冷热数据尽量分离，减少表的宽度；**减少IO，保证热数据内存缓存命中率，更有效的使用缓存，避免读入无用的冷数据，经常一起使用的放到一个表中，避免关联操作；
7. **禁止在表中建立预留字段**：
8. **禁止在数据库中存储图片，文件等大的二进制数据**；
9. **禁止在线上做数据库压力测试**；
10. **禁止从开发环境，测试环境直接连接生成环境数据库**；

数据库字段设计规范

1. **优先选择符合存储需要的最小的数据类型**；列的字段越大，建立索引所需的空间就越大，这样一页所能存储的节点就会越少，遍历时所需要的的IO次数也就越多，索引的性能越差； 字符串转换为数字类型存储，IP-》整数； 非负性数据优先存储无符号
2. **避免使用TEXT，BLOB类型数据，最常见的TEXT类型可以存储64k的数据；** 建议把BLOB或者TEXT列分离到单独扩展表中，内存临时表放不下，必须放到磁盘临时表，对于这种数据MySQL还需要进行二次查询，会使sql性能变很差，也不是一定不能用，但是在不需要这些数据的时候，选取需要是列数据输出，不要全部输出；
3. **避免使用ENUM类型；** 修改ENUM需要使用ALTER语句，ENUM类型的ORDEY BY效率低需要额外操作禁止使用数值作为ENUM的枚举值；
4. **尽可能把所有的列定义为NOT NULL**
5. **使用TIMESTAMP（4字节）或者DATETIME（8字节）存储时间；** 使用String存储时间是不对的，无法确定日期函数进行计算和存储，在字符串存储日期要占用更多的空间；
6. **同财务相关的金额类型数据必须用decimal类型**：

索引设计规范

1. **限制每一个表上的索引值，建议单张表索引不超过5个；**索引不是越多越好，多了会影响插入修改更新的效率；
2. **禁止给表中每一列建立单独的索引**：
3. **每个InnoDB必须有一个主键**；innoDB是一种索引组织表，数据的存储逻辑和索引的顺序是相同的。每个表都可以有多个索引，但是存储顺序只有一个就是主键的顺序；不要使用更新频繁的列作为主键，不使用与多列主键；主键建议使用自增ID值，不要使用UUID、hash等字符串作为主键；
4. **常见索引列建议**： 出现在SELECT、UPDATE、DELETE语句的WHERE从句中的列， 包含在ORDER BY 、GROUP BY、DISTINCT中的字段， 并不要将符合1和2中的字段都建立一个索引，通常将1，2中的字段建立联合索引好； 多表join关联列；
5. **如何选择索引列的顺序**：建立索引的目的是，希望通过索引进行数据查找，减少随机IO，增加查询性能，索引过滤出越少是数据，则从磁盘中读入的数据也就越少。 区分度最高的放在联合索引的最左列；尽量吧字段长度小的列放在最左边 使用最频繁的列放在左侧；
6. **避免建立冗余索引和重复索引**：
7. **对于频繁的查询优先考虑使用覆盖索引**： 覆盖索引：包含了所有查询字段的索引；
8. **索引SET规范**；

数据库SQL开发规范

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所带来的的SQL注入问题； 只传参数，比传递SQL语句更高效； 相同的语句一次解析，多次使用，提高效率；

2. **避免数据类型的隐式转换**： 隐式转换会导致索引失效，select name,phone from customer where id = '111';

3. **充分利用表上已经存在的索引**： 避免使用通配符的查询条件，如 a like '%20%', 一个SQL只能利用符合索引的一列进行范围查询； 在定义联合索引时，如果a列要用到范围查询的话，就把a列放到索引的右侧，使用 left join或者not exists来优化 not in操作；

4. **数据库设计时，应该要对之后扩展考虑**；

5. **程序连接不同的数据库使用不同的账号，进制跨库查询** 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安全风险；

6. **禁止使用SELECT * 必须使用SELECT <字段>查询**；

7. **禁止使用不含字段列表的INSERT语句**；

8. **避免使用子查询，可以把子查询优化成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中才可以吧子查询转化为关联查询进行优化； **子查询性能差的原因** 子查询的结果集存储在临时表中，不论是内存还是磁盘都不会存在索引，所以查询性能会受到影响；

9. **避免使用JOIN关联太多的表**： 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。大量使用多表关联操作的话，容易造成服务器内存溢出的情况，关联操作建议不超过5个表；

10. **减少通数据库的交互次数** 数据库更适合批量工作；

11. **对应同一列进行Or判断的时候，使用in代替Or** in操作更有效的使用索引，or大多数情况很少利用到索引；
12. **禁止使用order by rand()进行随机排序**
13. **WHERE从句中禁止对列进行函数转换和计算**
14. **在明显不会有重复值的时候使用UNION ALL而不是UNION**
15. **拆分复杂的大SQL为多个小SQL**

数据库操作行为规范

1. **超100万行的批量写要分批多次进行操作** 大批量操作可能会造成严重的主从延迟；主从环境中，大批量操作会造成严重的主从延迟，大批量的写操作需要执行一定长的时间，而只有主库上执行完成之后，才会在其他从库上执行；

   binlog日志为row格式时会产生大量的日志，大批量的写操作产生大量的日志，特别是对于row格式二进制的数据而言，由于在 row 格式中会记录每一行数据的修改，我们一次修改的数据越多，产生的日志量也就会越多，日志的传输和恢复所需要的时间也就越长，这也是造成主从延迟的一个原因

   避免产生大事务操作，大批量的修改数据，一定是在一个事务中进行的，这就会造成表中大量数据进行锁定，从而导致大量的阻塞，阻塞会对MySQL的性能大大影响，特别是阻塞会占满所有的可用连接，这会使生产环境中其他应用无法连接到数据库；

2. **对于大表使用pt-online-schema-change修改表结构** 避免大表修改产生的主从延迟，避免对表字段的修改时进行的锁表； 对大表数据结构的修改一定要谨慎，会造成严重的锁表操作，尤其是生产环境，是不能容忍的；

   pt-online-schema-change 它会首先建立一个与原表结构相同的新表，并且在新表上进行表结构的修改，然后再把原表中的数据复制到新表中，并在原表中增加一些触发器。把原表中新增的数据也复制到新表中，在行所有数据复制完成之后，把新表命名成原表，并把原来的表删除掉。把原来一个 DDL 操作，分解成多个小的批次进行。 (hashmap扩容思想)；

3. **禁止为程序使用账号赋予super权限**
4. **对于程序连接数据库账号，遵循权限最小原则**

阿里Java开发手册数据库部分最佳实践总结

模糊查询

强制：页面搜索严禁左模糊或者全模糊，实在要走搜索引擎来解决；索引文件具有B-Tree的最左前缀匹配特性，如果左边的值没有确定，那么无法确定使用此索引；

外键和级联

强制： 不得使用外键和级联，一切外键概念必须在应用层解决； 说明： 以学生和成绩的关系为例，学生表中的student_id是主键，那么成绩表中的student_id则是外键。如果更新学生表中student_id，成绩表中的

student_id也更新，即为级联更新。外键和级联更新适用于单机低并发，不适合分布式，高并发集群，级联更新更是强阻塞，存在数据更新风暴的风险；

为什么不要用外键呢？

1. 增加了复杂性：每次做更新删除操作的时候要考虑外键约束，开发痛苦，外键的主从关系是定的，如果需求变化，数据库中不需要这个字段和其他表有关联，会增加很多的麻烦；

2. 额外增加了工作： 数据库需要增加维护外键的工作，比如当我们做一些设计外键字段的增删改操作之后，需要触发相关操作去检查，保证数据的一致性和正确性；

3. 外键会需要请求其他表内部加锁容易出现死锁

4. 对分库分表不友好：分库分表下外键是无法生效的；