数据库设计规范

基础规范

1. 必须使用InnoDB存储引擎
   解读：支持事务、行级锁、并发性能更好、CPU及内存缓存页优化使得资源利用率更高
2. 必须使用UTF8字符集,支持 emoji 表情需要(UTF8MB4字符集)
   解读：万国码，无需转码，无乱码风险，节省空间
3. 数据表、数据字段必须加入中文注释
   解读：N年后谁tm知道这个r1,r2,r3字段是干嘛的
4. 禁止使用存储过程、视图、触发器、Event
   解读：高并发大数据的互联网业务，架构设计思路是“解放数据库CPU，将计算转移到服务层”，并发量大的情况下，这些功能很可能将数据库拖死，业务逻辑放到服务层具备更好的扩展性，能够轻易实现“增机器就加性能”。数据库擅长存储与索引，CPU计算还是上移吧
5. 禁止存储大文件或者大照片
   解读：为何要让数据库做它不擅长的事情？大文件和照片存储在文件系统，数据库里存URI多好

命名规范

1. 只允许使用内网域名，而不是ip连接数据库
2. 线上环境、开发环境、测试环境数据库内网域名遵循命名规范
   - 业务名称：db_user_
   - 从库在名称后加-s标识，备库在名称后加-ss标识
   - 线上从库：db_user_s
   - 线上备库：db_user_ss

3. 命名
   - 库名 -> 小写，下划线风格
   - 表名 -> 小写，下划线风格
   - 字段名-> 驼峰风格
   - 索引 -> idx_xxx(非唯一), uniq_xxx(唯一)
   - 所有命名不超过32个字符，必须见名知意，禁止拼音英文混用

表设计规范

1. 单实例表数目必须小于500
2. 单表列数目必须小于30
3. 表必须有主键，例如自增主键
   - 主键递增，数据行写入可以提高插入性能，可以避免page分裂，减少表碎片提升空间和内存的使用
   - 主键要选择较短的数据类型， Innodb引擎普通索引都会保存主键的值，较短的数据类型可以有效的减少索引的磁盘空间，提高索引的缓存效率
   - 无主键的表删除，在row模式的主从架构，会导致备库夯住

4. 禁止使用外键，如果有外键完整性约束，需要应用程序控制
    - 外键会导致表与表之间耦合，update与delete操作都会涉及相关联的表，十分影响sql的性能，甚至会造成死锁。高并发情况下容易造成数据库性能，大数据高并发业务场景数据库使用以性能优先

字段设计规范

1. 必须把字段定义为NOT NULL并且提供默认值
   null的列使索引/索引统计/值比较都更加复杂，对MySQL来说更难优化
   null 这种类型MySQL内部需要进行特殊处理，增加数据库处理记录的复杂性；同等条件下，表中有较多空字段的时候，数据库的处理性能会降低很多
   null值需要更多的存储空，无论是表还是索引中每行中的null的列都需要额外的空间来标识
   对null 的处理时候，只能采用is null或is not null，而不能采用=、in、<、<>、!=、not in这些操作符号。如：where name!=’shenjian’，如果存在name为null值的记录，查询结果就不会包含name为null值的记录
2. 设置时间字段
    - dateline 更新时间例如2018-06-04 10:30:30, 格式设置为datetime 默认值CURRENT_TIMESTAMP
    - 时间格式为20180604，格式设置为int(11)

3. 谨慎使用TEXT、BLOB类型
   会浪费更多的磁盘和内存空间，非必要的大量的大字段查询会淘汰掉热数据，导致内存命中率急剧降低，影响数据库性能
4. 禁止使用小数存储货币
   使用整数吧，小数容易导致钱对不上
5. 必须使用varchar(20)存储手机号
   涉及到区号或者国家代号，可能出现+-()
   手机号会去做数学运算么？
   varchar可以支持模糊查询，例如：like“138%”
6. 禁止使用ENUM，可使用TINYINT代替
   增加新的ENUM值要做DDL操作
   ENUM的内部实际存储就是整数，你以为自己定义的是字符串？
   索引设计规范
7. 单表索引建议控制在5个以内
8. 单索引字段数不允许超过5个
    - 字段超过5个时，实际已经起不到有效过滤数据的作用了

9. 禁止在更新十分频繁、区分度不高的属性上建立索引
    - 更新会变更B+树，更新频繁的字段建立索引会大大降低数据库性能
    - “性别”这种区分度不大的属性，建立索引是没有什么意义的，不能有效过滤数据，性能与全表扫描类似

10. 建立组合索引，必须把区分度高的字段放在前面
    能够更加有效的过滤数据
11. 如果是日志表, dateline 字段必须设置索引

SQL使用规范

1. 禁止使用SELECT *，只获取必要的字段，需要显示说明列属性
   读取不需要的列会增加CPU、IO、NET消耗
   不能有效的利用覆盖索引
   使用SELECT *容易在增加或者删除字段后出现程序BUG
2. 禁止使用INSERT INTO t_xxx VALUES(xxx)，必须显示指定插入的列属性
   容易在增加或者删除字段后出现程序BUG
3. 禁止使用属性隐式转换
   SELECT uid FROM t_user WHERE phone=13812345678 会导致全表扫描，而不能命中phone索引，猜猜为什么？（这个线上问题不止出现过一次）
4. 禁止在WHERE条件的属性上使用函数或者表达式
5. SELECT uid FROM t_user WHERE from_unixtime(day)>='2017-02-15' 会导致全表扫描, 正确的写法是：SELECT uid FROM t_user WHERE day>= unix_timestamp('2017-02-15 00:00:00')
6. 禁止负向查询，以及%开头的模糊查询
   负向查询条件：NOT、!=、<>、!<、!>、NOT IN、NOT LIKE等，会导致全表扫描
   %开头的模糊查询，会导致全表扫描
7. 禁止大表使用JOIN查询，禁止大表使用子查询
   会产生临时表，消耗较多内存与CPU，极大影响数据库性能
8. 禁止使用OR条件，必须改为IN查询
   旧版本Mysql的OR查询是不能命中索引的，即使能命中索引，为何要让数据库耗费更多的CPU帮助实施查询优化呢？
9. 应用程序必须捕获SQL异常，并有相应处理

参考一