RSR DB设计规范

为了规范公司开发流程，减少后期维护数据库时产生的不必要的耗时，提供开发人员的效率，故从以下几个方面提出相应的设计规范，望大家谨记：

**一：基础规范**

**1、数据库命名规范**

  采用公司名+‘\_’+26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成;

  命名简洁明确(长度不能超过30个字符);

   例如：rsr\_user, rsr\_stat, rsr\_log

除非是备份数据库可以加0-9的自然数：rsr\_user\_bf\_20151210;

**2、数据库表名命名规范**

  采用26个英文字母(区分大小写)和0-9的自然数(经常不需要)加上下划线'\_'组成，不超过32个字符，要见名知意;

  命名简洁明确,多个单词用下划线'\_'分隔;

   例如：user\_login, user\_profile, user\_detail, user\_role, user\_role\_relation, user\_role\_right, user\_role\_right\_relation表前缀'user\_'可以有效的把相同关系的表显示在一起;

**3、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

**4、使用UTF8字符集**

解读：万国码，无需转码，无乱码风险，节省空间（由于移动设备原因在需要时最好使用utf8mb4)

**5、数据表、数据字段必须加入中文注释**

解读：N年后谁知道这个r1,r2,r3字段或者a1,a2数据库是干嘛的

**6、禁止使用存储过程、视图、触发器、Event**

解读：高并发[大数据](http://lib.csdn.net/base/hadoop)的互联网业务，[架构](http://lib.csdn.net/base/architecture)设计思路是“解放[数据库](http://lib.csdn.net/base/mysql)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](http://lib.csdn.net/base/mysql)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

**7、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

**8、索引名规范 : 非唯一索引名 idx\_xxxx 唯一索引名 uniq\_xxxx**

**二：表设计规范**

1. **单实例（单库）表数目必须小于200**
2. **单表列数目必须小于30**
3. **表必须有主键，例如自增主键**

解读：

a）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

b）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

c） 无主键的表删除，在row模式的主从架构，会导致备库夯住

**4、不使用外键，如果有外键完整性约束，需要应用程序控制**

解读：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

**三、字段设计规范**

**1、尽可能将字段定义为NOT NULL并且提供默认值**

解读：

a）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

b）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

c）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

d）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 != ’wangkaiping’，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、尽量不使用TEXT、BLOB类型**

解读：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

**3、不使用小数存储货币**

解读：使用整数吧，小数容易导致钱对不上

**4、存储手机号使用varchar(20)**

解读：

a）涉及到区号或者国家代号，可能出现+-()

b）手机号会去做数学运算么？

c）varchar可以支持模糊查询，例如：like“138%”

**5、能使用tinyint存储的数据禁止使用int**

解读：

a）**tinyint：**从 0 到 255 的整型数据。存储大小为 1 字节。

**b）int:**从 -2^31 (-2,147,483,648) 到 2^31 – 1 (2,147,483,647) 的整型数据（所有数字）。存储大小为 4 个字节。

c）int(M) 在 integer 数据类型中，M 表示最大显示宽度。在 int(M) 中，M 的值跟 int(M) 所占多少存储空间并无任何关系。和数字位数也无关系 int(3)、int(4)、int(8) 在磁盘上都是占用 4 btyes 的存储空间

**6、对于使用数字表示的枚举类型，使用tinyint**

解读：在已知的状态\类型，使用tinyint(2)数据类型，虽然我们也可以使用char来存储，存储大小一致，但是在速度上tinyint优于char

**四：索引规范**

**1、单表索引建议控制在5个以内**

**2、单索引字段数不允许超过5个**

解读：字段超过5个时，实际已经起不到有效过滤数据的作用了

**3、禁止在更新十分频繁、区分度不高的属性上建立索引**

解读：

a）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

b）“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

**4、建立组合索引，必须把区分度高的字段放在前面**

**解读：**能够更加有效的过滤数据

**五：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**

解读：

a）读取不需要的列会增加CPU、IO、NET消耗

b）不能有效的利用覆盖索引

c）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**

解读：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**

解读：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、不使用负向查询，以及%开头的模糊查询**

解读：

a）负向查询条件：NOT、!=、<>、!<、!>、NOT IN、NOT LIKE等，会导致全表扫描

b）%开头的模糊查询，会导致全表扫描