数据库设计规范

[数据库设计规范 1](#_Toc1747013)

[一、数据库环境介绍 2](#_Toc1747014)

[二、 数据库规范 2](#_Toc1747015)

[（1） 三大范式 2](#_Toc1747016)

[（2） 数据库命名规范 3](#_Toc1747017)

[（2）表命名规范 3](#_Toc1747018)

[（3）字段命名规范 4](#_Toc1747019)

[（4）数据库表字段类型规范 5](#_Toc1747020)

[（5）数据库表索引规范 5](#_Toc1747021)

[（6）简单熟悉数据库范式 6](#_Toc1747022)

[三、主键设计 6](#_Toc1747023)

[（1）主键定义 6](#_Toc1747024)

[（2）主键设计原则 6](#_Toc1747025)

[（3）主键方案 7](#_Toc1747026)

一、数据库环境介绍

通常来讲，各个互联网公司的数据库分为5个数据库环境：

1. dev : 开发环境, 开发可读写,可修改表结构; 常用的163的数据库表; 开发人员可以修改表结构, 可以随意修改其中的数据; 但是需要保证不影响其他开发同事;

2. qa : 测试环境, 开发可读写, 开发人员可以通过工具修改表结构;

3. sim: 模拟环境, 开发可读写, 通过web平台;发起上线请求时，会先在这个环境上进行预执行， 这个环境也可供部署上线演练或压力测试使用 可以读写;

4. real: 生产数据库从库（准实时同步）,只读环境,不允许修改数据,不允许修改表结构; 供线上问题查找,数据查询等使用;

5. online: 线上环境;开发人员不允许直接在线上环境进行数据库操作,如果需要操作必须找DBA进行操作并进行相应记录;

这些环境的机器，一定要做到权限划分明确，读写帐号分离，并且有辨识度，能区分具体业务。例如用户名w\_wap, r\_wap 能看出来，读写帐号是wap应用的。

1. 数据库规范
2. 三大范式
3. 每一列都是不可分割的原子数据项，数据库所有字段都是不可分解的原子值；合理的遵循第一范式需要根据系统的实际需求来调整，如“地址”属性，通常将这个属性设置为一个字段，但是如果系统中经常需要访问地址这个属性中城市这个值，就可以考虑将“地址”这个字段分解成“省份”、“城市”，“详细地址”等三个字段。
4. 要求实体的属性完全依赖于主关键字，数据表中的每一列都和主键相关，而不能只与主键的某一部分相关（针对联合主键而言）；如商品订单业务，不要将以订单编号，商品编号作为联合主键而将商品信息和订单信息放在同一张表中，可以分为订单表，商品表，和订单项目表（订单编号，商品编号，订单数量）。
5. 任何非主属性不依赖于其他非主属性，每一列数据都和主键直接相关而不同通过其他非主属性间接相关。如不能将客户详细的信息字段（客户名称，联系方式，公司地址）加入到订单表中，而应该独立一张客户表，通过客户编号和订单表关联。
6. 数据库命名规范

1. 尽量简洁明义，能够一眼看出来这个数据库是用来做什么的；

2. 使用名词作为数据库名称，并且只用英文，不用中文拼音；

3. 使用英文字母，全部小写，控制在3-7个字母以内；

4. 如果有多个单词，则使用下划线隔开，不建义驼峰命名；

例如，每个公司都有crm业务，那就叫做xx\_crm, 字符集统一utf8。字符集踩过的坑很多，为了通用性统一utf8：

create database xx\_crm default character set=utf8;

（2）表命名规范

1. 具备统一前缀，对相关功能的表应当使用相同前缀，如acl\_xxx，house\_xxx,ppc\_xxx；其中前缀通常为这个表的模块或依赖主实体对象的名字，通常来讲表名为：业务\_动作\_类型，或是业务\_类型；

2. 表名使用英文小写单词，如果有多个单词则使用下划线隔开；

3.表名简介，使用常见单词，避免使用长单词和生僻词；

4. 表引擎取决于实际应用场景及当前数据库中的已经存在的存储引擎；日志及报表类表建议用myisam，与交易，审核，金额相关的表建议用innodb引擎。总体来讲数据库默认innodb；

5. 数据表必须有主键，且建议均使用auto\_increment的id作为主键（与业务无关）,和业务相关的要做为唯一索引；

6. 默认使用utf8字符集（由于数据库定义使用了默认，数据表可以不再定义，但为保险起见，建议都写上）；

7. 所有的表都必须有备注，写明白这个表中存放的数据内容；

8. 预估表数据量，如果数据量较大（超过500w）则需要考虑分表策略。可以等量均衡分表或根据业务规则分表均可。要分表的数据表必须与DBA商量分表策略；

9. 职责相近的表，命名规则应该相同；如合同申请，账户信息，交友相关等；

举个例子，一张在线冲值记录表：user\_bank\_deposit 这个就非常符合标准，如果叫做userBankDeposit或是user\_chongzhi，就非常不友好。

CREATE TABLE `house\_refresh\_log` (

  `id` int(11) NOT NULL AUTO\_INCREMENT COMMENT '自增ID',

  `fangid` int(11) NOT NULL COMMENT '房贴子ID',

  `refresh\_time` int(11) NOT NULL COMMENT '刷新时间',

  PRIMARY KEY (`id`),

  UNIQUE KEY `fangid` (`fangid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='房刷新记录表'

（3）字段命名规范

1. 数据库字段命名与表名命名类似：

2. 使用小写英文单词，如果有多个单词使用下划线隔开；

3. 使用简单单词，避免生僻词；

4. 字段应当有注释，描述该字段的用途及可能存储的内容，如枚举值则建议将该字段中使用的内容都定义出来；

5. 是别的表的外键均使用xxx\_id的方式来表明；

6. 表的主键一般都约定成为id，自增类型；

7. 时间字段，除特殊情况一律采用int来记录unix\_timestamp；

8. 网络IP字段，除特殊情况一律用bigint来记录inet\_aton值；

9. 所有字段，均为非空，最好显示指定默认值；

10. 有些驱动对tinyint支持不够好，通常建义按容量来选择字段；

11. text字段尽量少用，或是拆到冗余表中；

CREATE TABLE `wanted\_post` (

  `id` int(10) NOT NULL AUTO\_INCREMENT,

  `puid` int(10) unsigned NOT NULL,

  `user\_id` int(10) NOT NULL COMMENT '发贴用户的id',

  `username` varchar(50) NOT NULL COMMENT '发贴用户的用户名',

  `city` smallint(4) NOT NULL COMMENT '所在城市',

  `ip` bigint(14) NOT NULL COMMENT '发帖人的ip',

  `district\_id` tinyint(2) NOT NULL COMMENT '所在区域的id',

  `district\_name` varchar(20) NOT NULL COMMENT '行政区名字',

  `street\_id` tinyint(2) NOT NULL COMMENT '所在街道(地标)的id',

  `street\_name` varchar(20) NOT NULL COMMENT '小区名字',

  `title` varchar(255) NOT NULL COMMENT '帖子的标题',

  `description` text NOT NULL COMMENT '帖子详情描述',

  `post\_at` int(11) NOT NULL COMMENT '用户发帖时间,数据创建的时间,使用整型存储',

  `refresh\_at` int(11) NOT NULL COMMENT '帖子被修改的时间,整型存储',

  `show\_time` int(11) NOT NULL COMMENT '帖子显示时间',

  `age\_max` int(11) NOT NULL DEFAULT '0' COMMENT '招聘最小年龄',

  `age\_min` int(11) NOT NULL DEFAULT '0' COMMENT '招聘最大年龄',

  `post\_refresh\_at` int(11) NOT NULL COMMENT '刷新时间',

  PRIMARY KEY (`id`),

  UNIQUE KEY `idx\_puid` (`puid`),

  KEY `user\_id\_index` (`user\_id`),

  KEY `post\_at\_index` (`post\_at`),

  KEY `refresh\_at\_index` (`refresh\_at`),

  KEY `show\_time\_index` (`show\_time`)

) ENGINE=InnoDB AUTO\_INCREMENT=55295 DEFAULT CHARSET=utf8 COMMENT='招聘帖子表'

（4）数据库表字段类型规范

用尽量少的存储空间来存数一个字段的数据;

例如：能使用int就不要使用varchar、char,能用varchar(16)就不要使用varchar(256);

IP地址最好使用int类型;

固定长度的类型最好使用char,例如：邮编;

能使用tinyint就不要使用smallint,int;

最好给每个字段一个默认值,最好不能为null;

（5）数据库表索引规范

命名简洁明确,例如：user\_login表user\_name字段的索引应为user\_name\_index唯一索引;

为每个表创建一个主键索引;

为每个表创建合理的索引;

建立复合索引请慎重;

（6）简单熟悉数据库范式

第一范式(1NF)：字段值具有原子性,不能再分(所有关系型数据库系统都满足第一范式);

例如：姓名字段,其中姓和名是一个整体,如果区分姓和名那么必须设立两个独立字段;

第二范式(2NF)：一个表必须有主键,即每行数据都能被唯一的区分;

备注：必须先满足第一范式;

第三范式(3NF)：一个表中不能包涵其他相关表中非关键字段的信息,即数据表不能有沉余字段;

备注：必须先满足第二范式;

备注：往往我们在设计表中不能遵守第三范式,因为合理的沉余字段将会给我们减少join的查询;

 例如：相册表中会添加图片的点击数字段,在相册图片表中也会添加图片的点击数字段;

三、主键设计

（1）主键定义

表中经常有一个列或多列的组合，其值能唯一地标识表中的每一行。这样的一列或多列称为表的主键，通过它可强制表的实体完整性。

（2）主键设计原则

总原则：根据数据库表的具体使用范围来决定采用不同的表主键定义。

1.确保主键的无意义性:

在开发过程中，有意义的字段例如“用户登录信息表”将“登录名”（英文名）作为主键，“订单表”中将“订单编号”作为主键，如此设计主键一般都是没什么问题，因为将这些主键基本不具有“意义更改”的可能性。

但是，也有一些例外的情况，例如“订单表”需要支持需求“订单可以作废，并重新生成订单，而且订单号要保持原订单号一致”，那将“订单编号”作为主键就满足不了要求了。

因此在使用具有实际意义的字段作为主键时，需要考虑是否存在这种可能性。

要用代理主键，不要使用业务主键。任何一张表，强烈建议不要使用有业务含义的字段充当主键。我们通常都是在表中单独添加一个整型的编号充当主键字段。

2.采用整型主键：

主键通常都是整数，不建议使用字符串当主键。（如果主键是用于集群式服务，可以采用字符串类型）。

3. 减少主键的变动：

       主键的值通常都不允许修改，除非本记录被删除。

4. 避免重复使用主键：

       主键的值通常不重用，意味着记录被删除后，该主键值不再使用。

5. 主键字段定义区分：

主键不要直接定义成【id】，而要加上前缀，定义成【表名id】或者【表名\_id】。

（3）主键方案

1. 自增ID

优点：

数据库自动编号，速度快，而且是增量增长，聚集型主键按顺序存放，对于检索非常有利。

数字型，占用空间小，易排序，在程序中传递方便。

缺点：

当系统与其他系统集成时，需要数据导入时，很难保证原系统的ID不发生主键冲突。在多个数据库间进行数据的复制时（SQL Server的数据分发、订阅机制允许我们进行库间的数据复制操作），自动增长式字段可能造成数据合并时的主键冲突及表关联关系的丢失。

如果其他系统主键不是数字型，会导致修改主键数据类型，导致其他相关表的修改。

在数据缓冲模式下，很难预先填写主键与外键的值。

自增量的值都是需要在系统中维护一个全局的数据值，每次插入数据时即对此次值进行增量取值。当在产生唯一标识的并发环境中，每次的增量取值都必须为此全局值加锁解锁以保证增量的唯一性。造成并发瓶颈，降低查询性能。每创建一条记录都需要对表加一次锁，在高并发环境下开销较大。

3. UUID

UUID是指在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的。在UUID的算法中，可能会用到诸如网卡MAC地址，IP，主机名，进程ID等信息以保证其独立性。

优点：

全局唯一性、安全性、可移植性。

能够保证独立性，程序可以在不同的数据库间迁移，效果不受影响。

保证生成的ID不仅是表独立的，而且是库独立的，在你切分数据库的时候尤为重要。

缺点：

InnoDB为聚集主键类型的引擎，数据会按照主键进行排序，由于UUID的无序性，InnoDB会产生巨大的IO压力。InnoDB主键索引和数据存储位置相关（簇类索引），uuid 主键可能会引起数据位置频繁变动，严重影响性能。

作为主键，UUID长度过长，主键索引KeyLength长度过大，而影响能够基于内存的索引记录数量，进而影响基于内存的索引命中率，而基于硬盘进行索引查询性能很差。严重影响数据库服务器整体的性能表现。

4. ID物理主键+UUID逻辑主键

InnoDB不适合使用UUID做物理主键，可以把它作为逻辑主键，物理主键依然使用自增ID。

主键仍然用auto\_increment\_int来做，而另加一个uuid做唯一索引，表外键关联什么的，还用uuid来做，也就是说auto\_increment\_int只是一个形式上的主键，而uuid才是事实上的主键，这样，一方面int主键不会浪费太多空间，另一方面，还可以继续使用uuid。

优点：

InnoDB会对主键进行物理排序，这对auto\_increment\_int类型有好处，因为后一次插入的主键位置总是在最后。但是对uuid来说则有缺点，因为uuid是杂乱无章的，每次插入的主键位置是不确定的，可能在开头，也可能在中间，在进行主键物理排序的时候，势必会造成大量的 IO操作影响效率。

缺点：

同自增ID的缺点：全局值加锁解锁以保证增量的唯一性带来的性能问题。

5. 总结

 本文主要针对MySQL数据库中的InnoDB存储引擎的主键设计原则进行调研，挑选了几种主流的主键方案进行优缺点的分析和对比，并最终建议选择自增ID作为物理主键，同时使用UUID作为逻辑主键的方案。

如果一些特殊的表，比如说日志表，其不需要维护，可以采用数据库自动增长ID的方式。这种方式性能好，产生也很方便。但是维护很麻烦。