关于数据库设计中的命名规范问题整理如下：

## 0、引言

在前面提到过，数据库设计过程中表、字段等的命名规范也算是设计规范的一部分，但因为牵涉的内容较多，所以将其单独列出。不过设计规范更多的是为了确保数据库设计的合理性、为了项目最终的协调稳定性，而命名规范则更多的是为了确保设计的正式和统一。公正的讲，数据库中表字段等等以什么样的方式命名、取具体什么名字，并不会直接影响到项目的稳定性，不是说叫黑猫项目就是正常的，叫白猫就运行异常了。

制定规范的直接目的是约束设计行为，最终目的是确保设计的合理统一。规范虽然是有丰富项目经验的人制定的，但维护的却不是某个人的意志，而是项目的意志，因为遵守此规范对项目是好的有利的，此规范才有意义。所以规范是为了项目利益最大化而在团队人员中形成的一种约定（貌似约定的英文单词Convention本身就有规范的意思），所有参与设计的人员都要遵守此约定，所有参与开发的人员都会依此约定解读设计。我们约定，所有的主键统一命名为id，结果有设计人员违反约定将一个非主键字段命名为id，约定被打破，共识也就被打破，设计人员之间、开发人员与设计人员之间的沟通就出现了隔阂。

设计规范更多的是为了合理，命名规范更多的是为了统一，团队协作中，统一在某种程度上比局部设计开发的好坏更重要。违反了约定，局部设计开发的再好，反而可能影响到项目整体的稳定协调。

约定优先于配置（Convention Over Configuration）。

在“设计规范”中提到过一些命名规范，也详细讲述了表、字段的类型、注释等属性的设置，为什么要求主键统一命名为id、统一为char(32)类型，为什么要求浮点型数值统一为decimal类型？我们希望团队中所有人看到设计成果，一眼就可以明白这个字段是做什么的、代表的含义是什么，可以但不止于见名知意。再者，当前的开发模式，前后端代码及数据库文档、程序文档、接口文档等等大都是由工具生成，而其最底层的依据就是数据库，表、字段的命名注释同时会影响到工具生成的文档、代码中的类属性方法甚至是前台页面的命名注释，数据库设计命名的规范关系到整个项目的规范。

命名规范会分三个大模块来介绍：基本规范、名大小写、具体规范，各大模块下面有的会有子模块特别说明。最后进行梳理总结。

## 1、基本规范

### A.可用字符

数据库、表、字段等所有名称的可用字符范围为：A-Z，a-z， 0-9 和\_下划线，除此外不允许使用其它字符作为名称。数据库及表名均不允许出现数字，字段名除非特殊情况不允许出现数字。

在前面介绍关系范式时曾提到过一个破坏范式的例子：平时的多图片上传功能，可能只设计一个字段存储图片名称，这样字段值中就会包含多个图片的名称，里面用|或其它符号分隔。像这种情况，其实也可以设计成三五个字段image\_name1、image\_name2、image\_name3……分别存储，然后限制可上传图片个数，这就是字段名中可出现数字的特殊情况——虽然也不建议这样设计或取名。

### B.命名方式

数据库、表、字段等所有名称使用英文单词或英文短语或相应缩写，禁止使用汉语拼音，且均使用单数名，例如：对存储客人信息的表命名为customer而不是customers。名称应该清晰明了，能够准确表达事物的含义，遵循见名知意的原则。

Oracle表、字段等名称统一使用大写，单词间用\_下划线分隔；SQLServer数据库、表等名称采用Pascal命名法，字段名称采用Camel命名法，大小写字母混排；MySQL数据库、表、字段等名称统一使用小写，单词间用\_下划线分隔。至于为何这样规定，下一个模块会有详细介绍。

Oracle相对特殊，通常的操作顺序是，先创建数据库实例，然后创建表空间，然后创建用户并设定此用户的默认表空间，最后在此用户下建表。多数情况下我们都是只建一个实例，然后在此实例下建不同的表空间、不同的用户，根据不同的用户来区分不同的库。关于实例、表空间及用户的命名方式并无限制，可以采用大小写混排，也可以只用大写或小写，但对于表和字段，我们要求统一为大写。

我们要求统一为大写或小写的名称，两个单词间用\_下划线分隔，SQLServer使用Pascal或Camel方式命名。这些不仅仅是为了数据库设计的可读性，也是为了最终生成代码的可读性。这里简单介绍下编程中常用的三种类、变量、函数等的命名方式：

a． 匈牙利命名法。由微软的一位匈牙利程序员Charles Simonyi 提出，相对复杂，首字母小写，基本原则是：变量名＝属性＋类型＋对象描述，其中每一对象的名称都要求有明确含义，可以取对象名字全称或名字的一部分。匈牙利命名法主要在C或C++这种面向过程的程序语言中使用，如果用在Java、C#这种面向过程的语言中就很别扭。

不过自己在写Web前端页面或脚本时，借用了这种命名方式，form表单中涉及的常用HTML标签不外乎如下几种：label、text、button、submit、password、textarea、radio、checkbox、select等，那我在给表单元素命名或者说是给id或name赋值时，就会将元素类型做前缀，例如用户名输入框为textName、性别单选按钮名为radioGender。这样做的好处是我在编写脚本时，根据id或name名称一眼就可以看出这个表单元素是什么类型。在修改页面中初始化表单数据时我可以直接遍历表单元素、根据元素名称判断出元素的类型进而采用适当的赋值动作，而不用逐个选择元素去赋值。

在[http://ASP.NET](https://link.zhihu.com/?target=http%3A//ASP.NET)编程中，如果使用微软的服务器控件，在命名时我会用控件类型做名称后缀，例如Name\_TextBox、Gender\_RadioButtonList等。之所以不再将类型做前缀，一来是VisualStudio本身默认的服务器控件命名方式即时如此，控件类型做后缀；二来是因为服务器控件的类型名称太长，而自己又不愿用缩写，因为没必要，VisualStudio的提示功能强大，后缀的长度不会影响到编程速度。

b． Camel命名法。即骆驼式命名法，首字母小写，采用该命名法的名称看起来就像骆驼的驼峰一样高低起伏。Camel命名法有两种形式：

第一种是混合使用大小写字母，例如englishName、fartherCode。在Java中，属性名和方法名一般都采用这种命名方式，在C#中只有属性名采用这种命名方式，我们在前面也规定，SQLServer中字段的命名也采用这种方式。第二种是单词之间加下划线，例如english\_name、farther\_code。我们在前面规定，Oracel和MySQL表、字段的命名都采用这种方式，不过我们要求Oracle全部使用大写字母，MySQL全部使用小写字母。再者，无论是在Java还是C#，甚至是在JavaScript中，所有的常量，都使用这种命名方式，但和Oracle表字段的命名方式一样要全部使用大写字母，比如前面的设计规范中介绍数据字典表时，字典类型、字典项的编码和文本信息需要即时获取，以往的习惯在程序中建立一个常量类，所有用到的字典数据在里面用常量标明，这时常量的命名方式即是如此。

c． Pascal命名法。即帕斯卡命名法，与Camel命名法类似，不过是首字母大写。在C#中，类名和方法名一般采用这种命名方式，在Java中类名一般采用这种方式。在前面也规定，SQLServer中数据库、表的命名也采用这种方式。

除数据库的设计外，不同编程语言、前端HTML标签、JavaScript脚本、样式等等部分都会涉及命名的问题，如果细细整理，项目开发中每个子模块的命名规范都够再出一份长篇文档的。这里只简单介绍下三种常用的命名方式，其它部分的命名方式只是一提，重点还是在数据库的命名规范上。前面说过多次，程序、文档甚至前端页面有大部分通过工具自动生成，只有数据库严格按要求来命名，才能根据不同的编程语言编写不同的代码模板，统一控制生成部分各处的命名方式。比如，我们要求在MySQL数据库中，表名都使用小写，单词间用下划线分隔，交易记录表名称为trade\_log，那可以设定生成规则，对应生成的实体类名就是TradeLog，对应生成的Dal层就是TradeLogDal，对应的Service名就是TradeLogService，等等。可如果设计没有规范、不统一，那文档生成规则、代码生成规则、程序编写规则等等也就无法统一制定了。

### C.长度限制

关于各种数据库管理系统（DBMS，Database Management System）本身对表、字段等名称的长度限制如下：

[https://pic4.zhimg.com/v2-cf4ad6d3c6ca2534d2e1b7b8b7193edf\_b.png](https://link.zhihu.com/?target=https%3A//pic4.zhimg.com/v2-cf4ad6d3c6ca2534d2e1b7b8b7193edf_b.png)

以上是从网络整理而来，Oracle、SQLServer及MySQL的限制长度亲自测试过。但也有说因为数据库和表的名字可能对应于目录和文件名，故而服务器运行的操作系统可能强加额外的限制。不过除了Oracle的限制长度过短外，其它的一般不会被超出。我们希望名称尽可能详细准确的表达事物含义，但如果过于冗长，就会给操作及后面的程序编写带来诸多不便。

### D.单词缩写

自己以往设计数据库时，经常头疼于表、字段的命名，一来找不到好的单词去表述，二来有时可能涉及多个单词，导致名称过长。字段名过长带来的不便有限，最终影响的不过是程序实体类中的一个属性，可如果表名也过长，就比较麻烦了，生成的程序各层间针对此表的类名、变量名都可能受到影响，给后期的编写带来很大不便。使用单词缩写又拿不准，找不到合适的缩写方式。这里建议当表名超过15个字符、字段名超过20个字符时就应该尝试用单词缩写重新命名，如果名称长度在此之内，原则上讲则尽可能不用缩写以使表述具体清晰，表、字段最终的名称长度要严格控制在30个字符以内。关于单词缩写规则如下：

a． 如果可以在字典里找到一个词的缩写，就用这个做为缩写，比如：Monday=Mon、December=Dec ，可在此网站下查找到一些英文单词的缩写：[http://shortof.com/；](https://link.zhihu.com/?target=http%3A//shortof.com/%25EF%25BC%259B)

b． 可以删除单词元音(词首字母除外)和每个单词的重复字母来缩写一个单词。比如：Current = Crnt、Address = Adr、Error = Err、Average = Avg；

c． 对于主从表，如果主表名称没有缩写而从表的名称需要缩写，则从表名称从第二个单词开始缩写，第一个名词尽可能和主表保持一致。比如企业基本信息表名称为enterprise，则企业诉讼表enterprise\_litigation可简写为enterprise\_ltg，企业证书表enterprise\_certificate可简写为enterprise\_crt。最终的数据库表及由数据库表生成的程序在集成开发环境（IDE，Integrated Development Environment ）中是按名称排列的，这样做是为了让相似功能的表、类文件排列在一起，方便开发者操作。

更详细的单词缩写规则介绍可以参考文档末尾的参考文献。

## 2、名大小写

理想情况下所有关系型数据库对于表名、字段名、字段内容等大小写的处理会有个大一统的方式，比如要求所有都是大小写敏感的，可实际的情况却是，不同的数据库及同一数据库在不同的操作系统下对大小写的处理都是不同的。以往笔记中记录的第一次遇到数据库处理大小写的问题是，做的一个登录页功能，测试人员发现输入用户名MengXianzhi或mengxianzhi均可以正常登录，但数据库用户表里只有一条用户名为MengXianzhi的记录，当时用的是SQLServer数据库。

### A.编码和字符序的介绍

在前面介绍数据库编码时曾经提到，如果使用MySQL Workbench创建新的数据库，会要求选择Collation。Collation的字面意思是字符序，用于指定数据集如何排序、及字符串间的比对规则。可字符本来是不分大小的，这样对字符的>、=、< 操作就需要有个字符序的规则。Collation做的就是这个事情，你可以对表进行字符序的设置，也可以单独对某个字段进行字符序的设置，优先级从高到底可分为四种：服务器层、数据库层、表层、字段层，真正决定性因素是在字段层，如果没有指定则默认从上一层继承过来：字段层继承表层，表继承数据库层，数据库层继承服务器层，服务器层则需要设置，如果不设置默认为latin1\_general\_ci。

平时我们说的设置MySQL编码为gbk、gb2312、utf8或lantin等指的是字符编码，也就是Character Set。当表的Character Set 是lantin1时，若字段类型为nvarchar，则字段的字符集自动变为utf8。

可见数据库、表、字段的Character Set可逐级覆盖，这有点像上面说的四种字符序设置方式间的优先级关系。本规范中建议数据库统一设置编码为utf8，不仅仅是为了应付数据库间导入导出过程中、因编码格式不统一而导致的恼人的乱码问题，也是因为utf8是一种万国码（Unicode）。软件的国际化是大趋势 ， 而Unicode 是国际化最佳的选择。在MySQL中有两个支持 Unicode的Character Set：第一个是UCS2，使用 16 bits 来表示一个 Unicode字符；第二个是utf8，使用 1~3 bytes 来表示一个 Unicode字符。

那字符编码（Character Set）和字符序（Collation）之间的关系是什么呢？

每个Character Set 会对应一定数量的 Collation，在MySQL命令窗口中输入Show Collation;命令可以查看到所有字符序及其所属的字符编码列表：

[https://pic4.zhimg.com/v2-65271c7142d84f3c4270804b67394ae3\_b.png](https://link.zhihu.com/?target=https%3A//pic4.zhimg.com/v2-65271c7142d84f3c4270804b67394ae3_b.png)

同一个Character Set的不同 Collation 的区别在于排序、字符集对比的准确度以及性能，这里的准确度是指相同两个字符在不同国家语言中的排序规则可能是不同的 ，性能是指排序以及比对速度。例如：utf8\_general\_ci 在排序的准确度上要逊于 utf8\_unicode\_ci， 当然，对于英语用户应该没有什么区别，但性能上要略优于 utf8\_unicode\_ci，例如前者没有对德语中ß = ss的支持。而 utf8\_danish\_ci 相比 utf8\_unicode\_ci 增加了对丹麦语的特殊排序支持。

Collation名字的规则可以归纳为两类：<Character Set>\_<Language/Other>\_<CI/CS>以及<Character Set>\_Bin。CI 是Case Insensitive的缩写， CS是Case Sensitive的缩写，即指定数据库对大小写是否敏感。MySQL中Character Set对应的Collation多是CI的，CS这种校验字符已经逐渐被淘汰，gbk、gb2312、utf8等编码的所有Collation没有一个是CS的。Bin表示用二进制存储数据，用编码值进行比较，区分大小写。

在上面的截图中也可以看到，gb2312编码默认的Collation是gb2312\_chinese\_ci、gbk编码默认的Collation是gbk\_chinese\_ci、utf8编码默认的Collation是utf8\_general\_ci。按本文档中的规范，建议所有编码统一设置为utf8，如果不单独设置Collation，则按默认的utf8\_general\_ci，字段值是不区分大小写的。

那在字符序为CI的情况下，如何在执行SQL查询时区分字段值的大小写呢？假设用户表user中有两个用户：MengXianzhi和mengxianzhi，当我们执行如下查询时会得到两条记录：

select \* from user where user\_name = 'MengXianzhi';

如果要区分大小写，有下面两种方式可以精确查询：

select \* from user where binary user\_name = 'MengXianzhi';

select \* from user where user\_name = binary 'MengXianzhi';

推荐使用第二种查询方式，这样可以保证当前字段的索引依然有效，而第一种会使索引失效。其实个人更倾向于建议统一设置数据库默认的Collation为utf8\_bin，也就是对大小写敏感。程序中针对数据库字段内容的比对查询处处都是，英文内容存储也处处都有，如果所有相关查询语句都加binary关键字，太过麻烦，不如在数据库中统一设置，这样也不会出现在本章节开头所描述的问题了。

如果不想在数据库中统一设置，也可以只针对表、字段单独设置，但非常不建议如此，因为这会导致局部配置和全局配置相悖 。一直坚持规范、约定、配置等尽可能采用大一统的方式，除非不得以。开放局部配置会导致配置的多样性，不利于统一管理维护，不过下面还是会简单介绍下局部配置的方法。

### B.编码和字符序的设置

这部分会分别对比MySQL、SQLServer、Oracle三种关系型数据库的字符编码和字符序配置，先从MySQL开始。

在MySQL中，自己没有找到从服务器层面直接配置Collation的方法，但是数据库、表及具体字段设置Collation的方法都有。再就是在PD中未能找到全局设置Collation的方法，只找到了具体到某一字段设置的地方。截图如下，最后一张是PD中对某一具体字段进行配置的方法：

[https://pic4.zhimg.com/v2-fc3ffc761b80903b2edf63a29e3dbb3b\_b.pnghttps://pic2.zhimg.com/v2-6e55b0cdc41740f8af78c14d454fc6dd\_b.pnghttps://pic1.zhimg.com/v2-50e80eb394d31a0e61c4ea191134127c\_b.pnghttps://pic4.zhimg.com/v2-408cea09eefee5e5f0490d196ebebb27\_b.png](https://link.zhihu.com/?target=https%3A//pic4.zhimg.com/v2-fc3ffc761b80903b2edf63a29e3dbb3b_b.pnghttps%3A//pic2.zhimg.com/v2-6e55b0cdc41740f8af78c14d454fc6dd_b.pnghttps%3A//pic1.zhimg.com/v2-50e80eb394d31a0e61c4ea191134127c_b.pnghttps%3A//pic4.zhimg.com/v2-408cea09eefee5e5f0490d196ebebb27_b.png)

如果要直接更改某一个数据库的Character Set或Collation可以在MySQL Workbench中做如下设置：

[https://pic2.zhimg.com/v2-6d876bdf6ce4825cc17ba3fbb6d54d0d\_b.pnghttps://pic4.zhimg.com/v2-f4b3f32aac9b6218814ea05c761d0e5b\_b.png](https://link.zhihu.com/?target=https%3A//pic2.zhimg.com/v2-6d876bdf6ce4825cc17ba3fbb6d54d0d_b.pnghttps%3A//pic4.zhimg.com/v2-f4b3f32aac9b6218814ea05c761d0e5b_b.png)

其实Character Set和Collation本就是一体的，所以其实都是在这一个地方设置，两种选项对应的SQL语句就是：

ALTER SCHEMA `bsctelmed` DEFAULT CHARACTER SET utf8 ;

ALTER SCHEMA `bsctelmed` DEFAULT CHARACTER SET utf8 DEFAULT COLLATE utf8\_bin ;

在SQLServer 2008中，只找到了从数据库层及具体字段层面直接配置Collation的方法，但是从表层面中的配置却没有。和MySQL一样，在PD中未能找到全局设置Collation的方法，只找到了具体到某一字段设置的地方。截图如下，最后一张是PD中对某一具体字段进行配置的方法：

[https://pic2.zhimg.com/v2-cfb3c140419ac6f2f47a64b2275e13b9\_b.pnghttps://pic4.zhimg.com/v2-b079e10d3aa4b6acf56856dd3cb1bc5b\_b.pnghttps://pic2.zhimg.com/v2-1d8e21cff0712599547171021015ec49\_b.png](https://link.zhihu.com/?target=https%3A//pic2.zhimg.com/v2-cfb3c140419ac6f2f47a64b2275e13b9_b.pnghttps%3A//pic4.zhimg.com/v2-b079e10d3aa4b6acf56856dd3cb1bc5b_b.pnghttps%3A//pic2.zhimg.com/v2-1d8e21cff0712599547171021015ec49_b.png)

如果想查看SQLServer的版本、字符序等相关信息也可以用如下SQL语句：

SELECT

SERVERPROPERTY(N'Edition') AS Edition,

SERVERPROPERTY(N'ServerName') AS ServerName,

SERVERPROPERTY('ProductVersion ') AS ProductVersion,

SERVERPROPERTY('ProductLevel') AS ProductLevel,

SERVERPROPERTY('ResourceVersion') AS ResourceVersion,

SERVERPROPERTY('ResourceLastUpdateDateTime') AS ResourceLastUpdateDateTime,

SERVERPROPERTY('Collation') AS Collation;

直接在SQL Server Management Studio 图形化界面中更改SQLServer的字符集可能会出现问题：

[https://pic2.zhimg.com/v2-6e2f49813d7f86fb52efa1a8f8b6cb85\_b.png](https://link.zhihu.com/?target=https%3A//pic2.zhimg.com/v2-6e2f49813d7f86fb52efa1a8f8b6cb85_b.png)

这时可通过如下更改语句进行更改：

[https://pic1.zhimg.com/v2-2675ff53fb17c0a54f7ea0970beb1148\_b.png](https://link.zhihu.com/?target=https%3A//pic1.zhimg.com/v2-2675ff53fb17c0a54f7ea0970beb1148_b.png)

ALTER DATABASE HealthManagement COLLATE Chinese\_PRC\_CI\_AS

在Oracle中貌似没有Collation的概念，又或者是换了另外一个概念来进行类似的设置？在PD中具体到表字段的Oracle选项卡中也没有字符集相关的配置：

[https://pic2.zhimg.com/v2-129fc7359049f2f420f9029f0d3d6071\_b.png](https://link.zhihu.com/?target=https%3A//pic2.zhimg.com/v2-129fc7359049f2f420f9029f0d3d6071_b.png)

Oracle中默认是严格区分字段内容大小的，如果不想对大小写进行区分可以使用Lower()或Upper()函数来达到目的，也可以使用NLSSORT（）函数，觉得这个函数就和MySQL中的Collation设置所达到的效果相似。如下三个SQL语句所达到的效果是一样的：

select \* from user where user\_name = 'MengXianzhi' COLLATE Latin1\_General\_CI\_AI;

select \* from user where Upper(user\_name) = Upper('MengXianzhi');

select \* from user where NLSSORT(user\_name,'NLS\_SORT = Latin\_CI') = NLSSORT('MengXianzhi','NLS\_SORT = Latin\_CI');

但是不清楚上面第三种写法，如果不是精确查询，而是模糊查询，用like关键字，查询语句如何下，尝试下面的写法，好像不起作用：

select \* from user where NLSSORT(user\_name,'NLS\_SORT = Latin\_CI') like NLSSORT('Meng','NLS\_SORT = Latin\_CI')+"%";

Oracle9i之前，中文是按照二进制编码进行排序的，而在Oracle9i中新增了按照拼音、部首、笔画排序功能，使用方法如下：按汉字拼音排序：

SELECT \* FROM USER ORDER BY NLSSORT(user\_name ,'NLS\_SORT = SCHINESE\_PINYIN\_M')

按汉字笔划排序：

SELECT \* FROM USER ORDER BY NLSSORT(user\_name ,'NLS\_SORT = SCHINESE\_STROKE\_M')

按汉字部首排序：

SELECT \* FROM USER ORDER BY NLSSORT(user\_name ,'NLS\_SORT = SCHINESE\_RADICAL\_M')

注意，我虽然严格测试过MySQL、SQLServer和Oracle三种不同关系型数据库针对CharacterSet和Collation设置的区别，但对于同一数据库的不同版本间的区别却未深究。各种关系型数据库总是在不停升级，某些升级可能会导致新旧版本间差异巨大 ，而本文档中所述细节又甚多，所以具体到实际情况，某些地方出现不同也很正常。

### C.由此引出的乱码问题

Character Set和Collation并不仅仅影响到数据库存储内容的大小写敏感问题，还会影响到数据库操作中常见的乱码问题。这里既然提到了，所以简单讲下。

以往负责的项目较为杂乱，所以对各种常见关系型数据库多有接触，就自己的经验，乱码问题出现最多的是MySQL数据库，尤其是早期版本的MySQL，其次是Oracle，SQLServer当然也有，但相对少。乱码问题可以分为以下几种：

a． 不同类型的关系型数据库间、数据互相导入导出，导致的中文数据乱码。

比如将MySQL中的数据导入到SQLServer，将SQLServer中的数据导入到Oracle。这种情况其实相对少见，因为一般数据操作都是在同一类型的数据库间进行。遇到这种情况时，数据间的导入导出一般都有中间过程，比如先从源数据库中将数据导出成CSV文件，然后再将CSV文件导入到目标库。又或者是，借助目标数据库的管理工具，直接连接源数据库进行导入。也有将源数据库中的数据导出成SQL文件，然后对SQL文件进行一定更改后在目标数据库中执行的。

b． 类型相同、版本不同的关系型数据间的数据导入导出，导致的中文数据乱码。

c． 类型相同、版本相同的关系型数据间的数据导入导出，导致的中文数据乱码。

d． 针对Oracle，客户端版本和服务端版本不同所致。客户端的版本比较新、而服务端比较旧，或者是客户端为32位的而服务端是64位等箸。

e． 主要也是针对Oracle，客户端和客户端所在操作系统不协调、服务端和服务端所在操作系统不协调。比如操作系统为32位，但下载的客户端却是64位的。

f． 针对程序，官方管理工具操作数据库查询没有问题，但是程序访问数据库查询出的中文却是乱码。

中文乱码问题在各种数据库的操作中、在种程序语言各种项目的开发中时常出现，针对以上种种我们建议：

a． 安装及操作数据库时，编码相关的默认设置，除非有把握，否则不要随意更改；

b． 项目开发环境、测试环境、模拟环境、真实环境、线上环境的操作系统及数据库等尽可能统一版本统一配置，选择和操作系统相匹配的数据库版本；

c． 针对Oracle数据库，客户端和服务端尽可能统一版本，尽可能选择和操作系统相匹配的客户端及服务端版本；

d． 在数据库日常操作过程中均使用官方的管理工具，或直接在命令行中操作；SQLServer不用说，MySQL我们建议使用MySQL Workbench，Oracle我们建议使用SQL Developer；

e． 如果现实情况不方便或不允许达到以上要求，或者虽然按以上要求操作配置数据库后依旧出现乱码问题，那就根据实际情况网络搜索寻求相应解决方案。

遇到具体问题时刻记得Google是第一位的，仅这一项就可以帮我们解决99%的问题。我们的分析讨论建议更多的是为了全面了解问题本身，但遇到具体问题如何解决，仍旧要靠自己思考、靠Google的智能搜索。下面的截图来自于以往笔记，和某一乱码问题的交锋：

[https://pic1.zhimg.com/v2-6282fee2ad7a88971a5c945f0886524c\_b.png](https://link.zhihu.com/?target=https%3A//pic1.zhimg.com/v2-6282fee2ad7a88971a5c945f0886524c_b.png)

### D.表名字段名等大小写

上面讲字符序的大小写敏感，针对的都是数据库表字段值或者说是字段对容，而对于数据库名、表名、字段名、变量名、执行目录名等（在执行SQL查询时）的大小写敏感呢？

在Linux下MySQL的数据库名、执行目录名、 表名、表的别名、变量名默认是严格区分大小写的，数据库名大小写敏感不可改，执行目录名大小写敏感可参数调配（lower\_case\_file\_system），表名大小写敏感也可参数（lower\_case\_table\_names）调配，但不确定这个参数是否影响表别名及变量名的大小写敏感。列名与列的别名在所有的情况下均是忽略大小写的，也不清楚可否参数调配。

MySQL在Windows下数据库名、执行目录名、表名、表的别名、变量名、列名、列别名等默认都不区分大小写。

用root登录服务器修改 /etc/my.cnf配置文件，在[mysqld]节点下，加入一行： lower\_case\_table\_names=1 可以另其不再区分表名大小写。而在 Windows系统下， lower\_case\_table\_names参数缺省设置即为1，即不区分表名大小写。

在SQLServer中自己测试的结果是，数据库名、用户名、表名、表别名、列名、列别名默认在执行SQL查询时均不区分大小写。SQLServer版本为2008 R2。

[https://pic4.zhimg.com/v2-edfb67dc409b837b8b41cfec8f0fc037\_b.png](https://link.zhihu.com/?target=https%3A//pic4.zhimg.com/v2-edfb67dc409b837b8b41cfec8f0fc037_b.png)

在Oracle中自己测试的结果是，实例名、表空间名、用户名、表名、表别名、列名、列别名默认均不区分大小写。Oracle为Linux版本，11.2.0.4。

[https://pic3.zhimg.com/v2-f5b04cbab90c2e0e4e7bd7084ebec9ee\_b.png](https://link.zhihu.com/?target=https%3A//pic3.zhimg.com/v2-f5b04cbab90c2e0e4e7bd7084ebec9ee_b.png)

这样整理之后，如下表格：

[https://pic3.zhimg.com/v2-fe90b57872fe5b1c606f2a127187e25a\_b.png](https://link.zhihu.com/?target=https%3A//pic3.zhimg.com/v2-fe90b57872fe5b1c606f2a127187e25a_b.png)

内容中的是否表示默认情况下是否大小写敏感，括号中的（可）表示可参数调配。空白部分表示不确定，或者没有这一项。

SQLServer和Oracle 、MySQL（Windows系统下）虽然同样默认对表名、字段名等不区分大小写，但不同的是Oracle及MySQL处理的更严谨。通过SQL\*Plus、PL/SQL Developer或SQL Devloper在Oracle中建表，默认会自动将表名转换成大写后再写入数据库。 在Windows系统中，默认情况下，建表时MySQL会强制要求所有表名和列名均为小写。SQLServer虽然在执行SQL查询时不区分表名、列名大小写，但在命名及在可视化管理工具中显示时却又区分大小写。也有另外一种可能，目前我测试用的Oracle及MySQL版本比较新，则SQLServer则较旧，最新版的SQLServer或许已经没有这种问题。

[https://pic3.zhimg.com/v2-d742875c144bb15782fa4efcfff8b4a6\_b.png](https://link.zhihu.com/?target=https%3A//pic3.zhimg.com/v2-d742875c144bb15782fa4efcfff8b4a6_b.png)

前面说通过SQL\*Plus、PL/SQL Developer或SQL Devloper在Oracle中建表，默认会自动将表名转换成大写后再写入数据库。但实际上Oracle是可以支持大小写混排的命名方式的，但前提是要在表名外面加双引号。

仔细查看过，使用PD设计针对Oracle的PDM，如果你的表名全部大写，那PD在生成SQL建表语句时不会在表名外面加双引号，可如果你的表名是大小写混排的，那PD在生成SQL建表语句时会自动在表名外加双引号，保留这种大小写混排的命名方式。其实不光是创建表，在Oracle中创建触发器、序列时也是如此，名字不加引号就不会区分大小写，加上引号就会区分。

不建议在Oracle中使用大小写混排的命名方式，原因有很多：

a． 当你使用Oracle SQL Developer工具查看表时，点选“详细资料”选项卡，可能会报错：执行请求的操作时遇到错误，ORA-00904:"STATUS":invalid identifier。网上搜到ORA-00904错误原因和Oracle建表时表名大小写有关，但不清楚和Oracle版本有没有关系。

b． 如果表列名都区分大小写，那在建立查询时表名和列名都应该带有双引号，会给后面程序的编写带来麻烦。如果使用Hibernate框架，那其生成的查询是不会带有双引号的，会出现无法找到表或视图的错误。

c． 使用PL/SQL Developer工具可视化地进行表的删除等操作时，后台采用的是不带双引号的表名，也会出现无法找到表或视图的错误。这时只能采用类似 drop table "tableName" 的语法，在SQL\*Plus或PL/SQL Developer手工删除或修改表。

我们在基本规范中为什么要求MySQL的数据库名、表名、列名等统一为小写，Oracle中的表名、字段名等统一为大写，正是基于以上原因。我们希望藉此规定，将命名大小写规则统一，尽可能的让数据库设计不要在名称大小写这个问题上多出不必要的麻烦。

这里顺便一提，在PD中可以将PDM中的表名或列名统一转换成大写或小写，菜单Tools——Model Options——Naming——Convertion——Table或Column中进行设置。

### E.针对大小写合理建议

个人认为Oracle数据库对表名、字段名、字段内容等大小写敏感的默认处理是最合适的，在执行SQL查询时不区分表名、表别名、列名、列别名的大小写，但严格区分字段内容的大小写。也正因此，我们在基本规范中建议在Oracle数据库的设计过程中表、字段等的名称统一使用大写，单词间用\_下划线分隔。

我们在基本规范中建议，MySQL数据库、表、字段等名称统一使用小写，单词间用\_下划线分隔。同时，我们建议在MySQL数据库中将Character Set设置为utf8、将Collation设置为utf8\_bin，并在数据库配置文件中设置lower\_case\_table\_names=1，当然，Windows系统中默认就是此种设置，无需再做更改。

我们建议在SQLServer中将排序规则设置为Chinese\_PRC\_CS\_AS，其默认为Chinese\_PRC\_CI\_AS，因为SQLServer数据库不用考虑部署在不同系统的问题，所以不建议更改除此外的其它编码、字符序相关的默认设置。我们上面也说过SQLServer虽然在执行SQL查询时不区分表名、列名大小写，但在命名及在可视化管理工具中显示时却又区分大小写，为了查看方便所以我们在“基本规范”中要求SQLServer用Pascal的命名方式。

在“名大小写”这个章节，更多的不是制定规范，而是在讲解前面的“数据编码”、“基本规范”等模块中列出的一些规范制定的原因。在这里详细讲解了MySQL、SQLServer、Oracle三种数据库的编码、字符序相关的配置说明以及表名、字段名、字段内容等大小写敏感的控制处理等。

## 3、具体规范

### A.关于数据库的命名

对于数据库的命名不做特别要求，简单明了即可，这里主要注意在一个大环境中相似项目的数据库命名，最好有明显区分。

这里顺带一提，互联网公司的数据库一般分为五个环境：

a． 开发环境（Development Environment）。开发可读写，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。

b． 测试环境（Test Environment）。开发可读写，部署的测试系统访问此库，代测试人员使用。

c． 模拟环境（Simulation Environment ）。 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。

d． 线上从库（Real Environment）。 只读，会实时从线上数据库同步，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。

e． 线上环境（Online Environment）。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数据库主负责人，并做相应记录。

在这些环境中，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 分别表示对wap数据库进行读、写的帐号。

做企业内部应用系统，要求不是特别严格的话，没有模拟环境和线上从库。而且通常情况下，线上环境的库在客户那边，开发测试的环境在公司这边，两边还不能互通，有时不得不驻场开发直接连接线上环境。但是对于线上环境的直接操作是非常危险的，且容易导致线上环境和开发测试环境表结构的不同步，这个一定注意。客户那边应该用权限严格限制对生产环境访问的人员，开发人员自己这边要时刻做好数据备份工作，并提前准备好数据出现意外更改或丢失情况的应对措施。同时，在现场开发，针对线上环境的更改要实施同步到公司的开发环境中。线上线下的所有更改，都要经过数据库主设计师的审核同意。

我们建议，如果可以控制的话，则在不同的数据库环境中统一表空间名、数据库名等，甚至是数据库访问的账号名、权限也可以统一，这样在部署项目时，配置文件则无需再做过多更改，不同数据库环境间有表结构或数据的移植时也可避免出现不必要的问题。在对这些环境的数据库进行备份时，建议在备份文件名中加上前缀和备份时间，以防混淆，比如备份开发环境的数据库可命名为：DevelopmentEnvironment201703271149。这些都是非常细节的地方，有点吹毛求疵，不做强制要求。

### B.数据库功能块概述

在前面“设计规范”——“基本原则”——“高级功能”中提到过，现有的开发模式，数据库只用来做数据存储。一直坚持业务相关的部分都由程序处理，不到不得以的情况下不要在数据库中建存储过程、触发器、函数、序列甚至是视图等，尽管如此，这里还是会简单介绍下这些高级功能使用时的命名方式。下面的表格列出了数据库所涵盖常见功能元素的英文名称及缩写：

[https://pic1.zhimg.com/v2-10682d031e96351714599360d8371674\_b.png](https://link.zhihu.com/?target=https%3A//pic1.zhimg.com/v2-10682d031e96351714599360d8371674_b.png)

有建议，除表和表字段外，其它功能块在命名时均要加英文缩写前缀。但就个人意见，除视图外，其它部分加不加前缀不太重要，视图加前缀是为了在执行查询时和表区分开，而存储过程、函数、约束等，我们一眼即可看出它是什么，更何况在可视化管理工具中，这些功能块本来就是各自独立展示的。所以本规范中不强制要求在这些功能上加前缀，但如果要统一加的话，建议使用上图表格中的英文缩写。

### C.关于数据表的命名

关于表的命名，TB这种前缀是毫无意义的，本来就是一个表，为什么还要说明？这也是我上面不建议在其它功能块中加前缀的原因。如果表格数量较少，后期项目扩展升级的可能性不大，也没有必要加其它前缀。但有时规模相对庞大、业务逻辑相对复杂的项目，表格数量多到一定程度，在可视化管理工具中查阅浏览不太方便，这时，根据业务或功能对表格进行分类，加前缀也就有必要了。个人感觉是50张表内的数据库，加前缀意义不大，超过100张，则很有必要加前缀。而且我们要求，为了不给后期代码生成造成非必要麻烦，如果要给表加前缀，则所有表均要有前缀，不要出现有些表有、有些没有的情况。

表前缀主要是为了区分不同功能的表，而非解释表的功能，表的功能由表名来解释。前面要求表名的长度要控制在30个字符以内，在此前提下，为了尽可能不影响表的命名，表前缀应该越短越好。我们建议表前缀控制在两个以内。具体表前缀添加规则建议如下，括号内的单个大写字母表示要添加的前缀。这里以Oracle数据库为例，具体表名、前缀的大小写根据实际数据库参照“命名规范”——“名大小写”章节的说明：

a． 系统表（S\_）：System，系统配置相关的基本信息表。系统用户表（S\_USER）、系统角色表（S\_ROLE）、系统菜单（S\_LINK\_MENU）、操作日志（S\_OPERATION\_LOG）、登录日志（S\_LOGIN\_LOG）、系统字典（S\_DICTIONARY）、系统字典类型（S\_DICTIONARY\_TYPE）等。

b． 字典表（D\_）：Dictionary，非系统字典外的字典表。在“设计规范”——“相关注释”——“字典字段”中提到过字典表的定义，除了数据库中的通用字典表，还有一些常见表，比如地区表（D\_REGION）、ICD编码（D\_ICD）等，也是一种字典表，这里的D\_前缀即加在这类字典表名前面。

c． 中间表（R\_）：Relationship，多对多关系中间表。具体命名方式建议为：R\_主表名\_从表名，在多对多关系中其实不分主从表，这里我们规定核心表为主表，另外一个为从表。比如用户角色关系中，用户表（S\_USER）为主、角色（S\_ROLE）表为从，那中间表就命名为R\_USER\_ROLE。当中间表名超长时，则根据实际情况缩写主从表名，建议优先缩写从表表名。

d． 业务表（B\_）：Business，核心业务涉及的基本信息表。这里的业务是非系统配置业务相关的，比如登录、注册、权限这些业务涉及的表都是和系统配置相关的，前缀应该是S\_，而非B\_。比如在线商城的项目中订单业务涉及的表即是核心业务表，会诊系统中会诊单业务涉及的表即是核心业务表，如果项目庞大，涉及业务较多，可以在B后面继续加单字母区分不同的业务，BA\_、BB\_、BC\_……，没必要非得和某个英文对应，只是个代号，和项目组的人员说明即可。

表名前缀的说明如上，已经足够明确，除此外还应该避免无谓的表格后缀。比如存储客户信息的表直接命名为Guest而非GuestInfo，存储航班信息的表直接命名为Flight而非FlightList。还有命名表时，一律使用单数形式。例如，使用 Employee，而不是 Employees，总之，表的命名应该简单明了。

### D.关于表字段的命名

a． 所有表中的主键统一命名为id，主键统一使用UUID，类型统一为char(32)。 不建议使用复合主键，即便是在多对多关系的中间表中，个人还是建议用单独的字段做主键，复合字段加惟一约束。

b． 所有的表字段中，除外键，其它字段名都无需刻意加前后缀，也不要在字段名前出现表名。这里的外键是广义上的外键，不仅包括从表引用主表主键的外键字段，还包括存放主表相应关键信息的扩展字段。

比如病人表（Patient），主键就是id而不是pateint\_id，名称就是name而不是patient\_name。但对于外键，比如其它表引用Patient表的主键那就是patient\_id，对应Patient表的name字段那就是patient\_name。如果一个表中有多个外键（字段）同时引用（对应）一张表的同一个字段，那再用其它标识，比如在“设计规范”——“基本原则”——“主键外键”中提到的会诊单申请表中会诊发起医院（sender\_hopital\_id）和会诊接收医院（receiver\_hospital\_id）。

在前面的“设计规范”——“基本原则”——“主键外键”和“设计规范”——“约束控制”中有提到主键字段和外键字段的命名 ，这里再次做以上说明。另，PD中在由CDM转换成PDM时，会自动根据引用关系在从表中添加外键字段，可以自定义外键名称的命名规则：

[https://pic1.zhimg.com/v2-5506f2fd741969bc5109166d53025a70\_b.png](https://link.zhihu.com/?target=https%3A//pic1.zhimg.com/v2-5506f2fd741969bc5109166d53025a70_b.png)

c． 在前面的“设计规范”——“基本原则”——“连接查询”和“设计规范”——“相关注释”——“字典字段”有关于字典字段的详细介绍，这里再次说明其命名方式：对于字典字段，编码字段后面跟Code后缀，文本字段跟Text后缀，比如gender\_code、gender\_text。

d． 本规范中要求所有表示日期时间的字段，都要有后缀，如果只精确到天则以Date为后缀，如果要精确到时分秒那就用Time作后缀。在“设计规范”——“字段设置”——“通用字段处理”中有关于日期时间类型设置的说明，要求日期时间类型的字段，尽可能精确到时分秒，即便是像生日（birth\_date）这种字段，一般只存储到年月日，但在选择字段类型时建议还是为datetime而非date。所以这里的后缀并不是和具体字段类型对应，而是根据实际业务情况，这个字段存储的数据多是精确到年月日还是时分秒，则后缀相应的为Date或Time。

网上有建议说，日期时间不要用Time做后缀，因为Time还有一个很常用的意思，就是次数。比如登录日志表中有用户最后一次登录时间字段login\_time，不去看表的内容，很容易将login\_time理解成登录的次数。这里我们不予考虑，只要内部统一规范，这就不会是个问题。

e． 本规范中建议是否注销、是否成功等类似的布尔型字段，名称前统一加is前缀，比如是否成功（is\_success）、是否注销（is\_active）、是否显示（is\_display）等。

f． 关于一些通用字段的命名方式建议如下，仅作参考：

[https://pic1.zhimg.com/v2-ac7295f07397e0e523daa4afff848838\_b.png](https://link.zhihu.com/?target=https%3A//pic1.zhimg.com/v2-ac7295f07397e0e523daa4afff848838_b.png)

### E.关于约束控制命名

在“设计规范”——“约束控制”中介绍过五种约束类型：唯一性和主键约束、外键约束、检查约束、空值约束、默认值约束， 本规范中仅对外键约束的命名做要求，因外键约束标明着表与表之间的关系。我们建议外键约束以fk做前缀，后跟从表名称和主表名称：fk\_从表名\_主表名。这种定义方式，约束名称很容易超长，比如在Oracle中，约束名称的长度限制和表名一样，不能超过30个字符。如果超长，我们建议从后向前自动截取多出部分。前面提到过，CDM转换成PDM时会自动根据引用关系在从表中添加外键字段，外键名称的命名规则可以自定义。外键约束名称没必要手动添加，在PD的PDM图中选择：Database——Edit current DBMS——General选项卡——右侧树形菜单 Script\Objects\Reference\ConstName ，在里面可以编辑ConstraintName的命名方式，交由PD自动统一处理，比如可设置为：FK\_%.U30:CHILD%\_%.U30:PARENT%。此设置在PD 15中起作用，16版本中的设置没找到。

其它四种约束的命名，本规范中不做要求，窃以为这些约束怎样命名也不太重要，如果需要统一命名规范， 有些也可借助PD工具进行统一设置。

### F.其它功能块的命名

前面说过，因为自己所主张的开发模式，以往的项目中很少在数据库中建存储过程、触发器、函数、序列、事件甚至是视图等，这里只根据经验，给出少量建议。

视图的命名和表的命名有很多相似点，但认为视图的名称最好可直接反应出其查询的主表，或者可明确反应出视图功能。存储过程、触发器、函数、索引的名称则直接反应其功能为好，其命名方式类似于在编程语言中给某一方法命名。序列只在Oracle中有，一般用来填充主键和计数。在早期的数据库设计中，喜欢用自增主键，比如要让用户表（USER）的主键ID自增，则创建名为SQ\_USER\_ID的序列和名为TR\_SET\_USER\_ID的触发器。序列名直接反应出自己要计数的表的列，触发器名直接反应出自己的功能，这种命名方式或可借鉴。

不过后期项目的数据库设计，自己不再用自增主键，原因在“设计原则”——“基本规范”——“主键外键”中有描述。如果项目庞大，数据库设计的模式有变动，要大量使用存储过程、触发器、函数、序列等，对于这些部分的命名还是有必要规范化的。

## 4、梳理总结

“命名规范”牵涉的细节太多，在介绍过程中也一直妄求事无巨细，反而导致有些地方比较散乱，这里把关键部分梳理总结如下：

a． 建议在SQLServer中将排序规则设置为Chinese\_PRC\_CS\_AS，在MySQL数据库中将Character Set设置为utf8、将Collation设置为utf8\_bin，并在数据库配置文件中设置lower\_case\_table\_names=1，

b． 数据库、表、字段等所有名称的可用字符范围为：A-Z，a-z， 0-9 和\_下划线，长度要严格控制在30个字符以内。

c． 数据库、表、字段等所有名称均使用英文单词或英文短语或相应缩写，均使用单数名，禁止使用汉语拼音。

d． Oracle表、字段等名称的统一使用大写，单词间用\_下划线分隔；SQLServer数据库、表等名称采用Pascal命名法，字段名称采用Camel命名法；MySQL数据库、表、字段等名称统一使用小写，单词间用\_下划线分隔。

e． 表主键统一命名为id，主键统一使用UUID，类型统一为char(32)。

f． 表（广义）外键建议命名为：主表名\_字段名，类型和主表中字段类型一样。如果一个表中有多个外键（字段）同时引用（对应）一张表的同一个字段，再根据实际情况加前后缀区分

g． 对于字典字段，编码字段后面跟Code后缀，文本字段跟Text后缀，

h． 表示日期时间的字段，都要有后缀，如果只精确到天则以Date为后缀，如果要精确到时分秒那就用Time作后缀。

i． 建议是否注销、是否成功等类似的布尔型字段，名称前统一加is前缀，比如是否成功（is\_success）、是否注销（is\_active）、是否显示（is\_display）等。

j． 建议外键约束以fk做前缀，后跟从表名称和主表名称：fk\_从表名\_主表名。