表设计的18条军规

前言

对于后端开发同学来说，访问数据库，是代码中必不可少的一个环节。

系统中收集到用户的核心数据，为了安全性，我们一般会存储到数据库，比如：mysql，oracle等。

后端开发的日常工作，需要不断的建库和建表，来满足业务需求。

通常情况下，建库的频率比建表要低很多，所以，我们这篇文章主要讨论建表相关的内容。

如果我们在建表的时候不注意细节，等后面系统上线之后，表的维护成本变得非常高，而且很容易踩坑。

今天就跟大家一起聊聊，数据库建表的18个小技巧。

文章中介绍的很多细节，我在工作中踩过坑，并且实践过的，非常有借鉴意义，希望对你会有所帮助。

1.名字

建表的时候，给`表`、`字段`和`索引`起个好名字，真的太重要了。

1.1 见名知意

名字就像`表`、`字段`和`索引`的一张脸，可以给人留下第一印象。

好的名字，言简意赅，见名知意，让人心情愉悦，能够提高沟通和维护成本。

坏的名字，模拟两可，不知所云。而且显得杂乱无章，看得让人抓狂。

反例：

```
用户名称字段定义成：yong_hu_ming、用户_name、name、user_name_123456789
```

你看了可能会一脸懵逼，这是什么骚操作?

正例：

```
用户名称字段定义成：user_name
```

> 温馨提醒一下，名字也不宜过长，尽量控制在`30`个字符以内。

1.2 大小写

名字尽量都用`小写字母`，因为从视觉上，小写字母更容易让人读懂。

反例：

```
字段名：PRODUCT_NAME、PRODUCT_name
```

全部大写，看起来有点不太直观。而一部分大写，一部分小写，让人看着更不爽。

正例：

```
字段名：product_name
```

名字还是使用全小写字母，看着更舒服。

1.3 分隔符

很多时候，名字为了让人好理解，有可能会包含多个单词。

那么，多个单词间的`分隔符`该用什么呢?

反例：

```
字段名：productname、productName、product name、product@name
```

单词间没有分隔，或者单词间用驼峰标识，或者单词间用空格分隔，或者单词间用@分隔，这几种方式都不太建议。

正例：

```
```

字段名：product_name

```

强烈建议大家在单词间用`_`分隔。

### 1.4 表名

对于表名，在言简意赅，见名知意的基础之上，建议带上`业务前缀`。

如果是订单相关的业务表，可以在表名前面加个前缀：`order_`。

例如：order\_pay、order\_pay\_detail等。

如果是商品相关的业务表，可以在表名前面加个前缀：`product_`。

例如：product\_spu，product\_sku等。

这样做的好处是为了方便归类，把相同业务的表，可以非常快速的聚集到一起。

另外，还有有个好处是，如果哪天有非订单的业务，比如：金融业务，也需要建一个名字叫做pay的表，可以取名：finance\_pay，就能非常轻松的区分。

这样就不会出现`同名表`的情况。

[8000页BAT大佬写的刷题笔记，让我offer拿到手软](http://cxyroad.com/”http://www.susan.net.cn/zt/8000.html”)

### 1.5 字段名称

`字段名称`是开发人员发挥空间最大，但也最容易发生混乱的地方。

比如有些表，使用flag表示状态，另外的表用status表示状态。
```

可以统一一下，使用status表示状态。

如果一个表使用了另一个表的主键，可以在另一张表的名后面，加`_id`或`_sys_no`，例如：

在product_sku表中有个字段，是product_spu表的主键，这时候可以取名：product_spu_id或product_spu_sys_no。

还有创建时间，可以统一成：create_time，修改时间统一成：update_time。

删除状态固定为：delete_status。

其实还有很多公共字段，在不同的表之间，可以使用全局统一的命名规则，定义成相同的名称，以便于大家好理解。

1.6 索引名

在数据库中，索引有很多种，包括：主键、普通索引、唯一索引、联合索引等。

每张表的主键只有一个，一般使用：`id`或者`sys_no`命名。

普通索引和联合索引，其实是一类。在建立该类索引时，可以加`ix_`前缀，比如：ix_product_status。

唯一索引，可以加`ux_`前缀，比如：ux_product_code。

[8000页BAT大佬写的刷题笔记，让我offer拿到手软](http://cxyroad.com/”http://www.susan.net.cn/zt/8000.html”)

2.字段类型

在设计表时，我们在选择`字段类型`时，可发挥空间很大。

时间格式的数据有：date、datetime和timestamp等等可以选择。

字符类型的数据有：varchar、char、text等可以选择。

数字类型的数据有：int、bigint、smallint、tinyint等可以选择。

说实话，选择很多，有时候是一件好事，也可能是一件坏事。

如何选择一个`合适`的字段类型，变成了我们不得不面对的问题。

如果字段类型选大了，比如：原本只有1-10之间的10个数字，结果选了`bigint`，它占`8`个字节。

其实，1-10之间的10个数字，每个数字`1`个字节就能保存，选择`tinyint`更为合适。

这样会白白浪费7个字节的空间。

如果字段类型择小了，比如：一个18位的id字段，选择了`int`类型，最终数据会保存失败。

所以选择一个合适的字段类型，还是非常重要的一件事情。

以下原则可以参考一下：

1. 尽可能选择占用存储空间小的字段类型，在满足正常业务需求的情况下，从小到大，往上选。
2. 如果字符串长度固定，或者差别不大，可以选择char类型。如果字符串长度差别较大，可以选择varchar类型。
3. 是否字段，可以选择bit类型。
4. 枚举字段，可以选择tinyint类型。
5. 主键字段，可以选择bigint类型。
6. 金额字段，可以选择decimal类型。
7. 时间字段，可以选择timestamp或datetime类型。

3.字段长度

前面我们已经定义好了`字段名称`，选择了合适的`字段类型`，接下来，需要重点的是`字段长度`了。

比如：varchar(20)，biginit(20)等。

那么问题来了，`varchar`代表的是`字节`长度，还是`字符`长度呢?

答：在mysql中除了`varchar`和`char`是代表`字符`长度之外，其余的类型都是代表`字节`长度。

biginit(n) 这个`n`表示什么意思呢?

假如我们定义的字段类型和长度是：bigint(4)，bigint实际长度是`8`个字节。

现在有个数据a=1，a显示4个字节，所以在不满4个字节时前面填充0（前提是该字段设置了zerofill属性），比如：0001。

当满了4个字节时，比如现在数据是a=123456，它会按照实际的长度显示，比如：123456。

但需要注意的是，有些mysql客户端即使满了4个字节，也可能只显示4个字节的内容，比如会显示成：1234。

所以bigint(4)，这里的4表示显示的长度为4个字节，实际长度还是占8个字节。

4.字段个数

我们在建表的时候，一定要对`字段个数`做一些限制。

我之前见过有人创建的表，有几十个，甚至上百个字段，表中保存的数据非常大，查询效率很低。

如果真有这种情况，可以将一张`大表`拆成多张`小表`，这几张表的主键相同。

建议每表的字段个数，不要超过`20`个。

5. 主键

在创建表时，一定要创建`主键`。

因为主键自带了主键索引，相比于其他索引，主键索引的查询效率最高，因为它不需要回表。

此外，主键还是天然的`唯一索引`，可以根据它来判重。

在`单个`数据库中，主键可以通过`AUTO_INCREMENT`，设置成`自动增长`的。

但在`分布式`数据库中，特别是做了分库分表的业务库中，主键最好由外部算法(比如：雪花算法）生成，它能够保证生成的id是全局唯一的。

除此之外，主键建议保存跟业务无关的值，减少业务耦合性，方便今后的扩展。

不过我也见过，有些一对一的表关系，比如：用户表和用户扩展表，在保存数据时是一对一的关系。

这样，用户扩展表的主键，可以直接保存用户表的主键。

6.存储引擎

在`mysql8`以前的版本，默认的存储引擎是`myisam`，而`mysql8`以后的版本

，默认的存储引擎变成了`innodb`。

之前我们还在创建表时，还一直纠结要选哪种存储引擎?

`myisam`的索引和数据分开存储，而有利于查询，但它不支持事务和外键等功能。

而`innodb`虽说查询性能，稍微弱一点，但它支持事务和外键等，功能更强大一些。

以前的建议是：读多写少的表，用myisam存储引擎。而写多读多的表，用innodb。

但虽说mysql对innodb存储引擎性能的不断优化，现在myisam和innodb查询性能相差已经越来越小。

所以，建议我们在使用`mysql8`以后的版本时，直接使用默认的`innodb`存储引擎即可，无需额外修改存储引擎。

[8000页BAT大佬写的刷题笔记，让我offer拿到手软](http://cxyroad.com/”http://www.susan.net.cn/zt/8000.html”)

7. NOT NULL

在创建字段时，需要选择该字段是否允许为`NULL`。

我们在定义字段时，应该尽可能明确该字段`NOT NULL`。

为什么呢?

我们主要以innodb存储引擎为例，myisam存储引擎没啥好说的。

主要有以下原因：

1. 在innodb中，需要额外的空间存储null值，需要占用更多的空间。
2. null值可能会导致索引失效。
3. null值只能用`is null`或者`is not null`判断，用`=号`判断永远返回false。

因此，建议我们在定义字段时，能定义成NOT NULL，就定义成NOT NULL。

但如果某个字段直接定义成NOT NULL，万一有些地方忘了给该字段写值，就会`insert`不了数据。

这也算合理的情况。

但有一种情况是，系统有新功能上线，新增了字段。上线时一般会先执行sql脚本，再部署代码。

由于老代码中，不会给新字段赋值，则insert数据时，也会报错。

由此，非常有必要给NOT NULL的字段设置默认值，特别是后面新增的字段。

例如：

```
alter table product_sku add column  brand_id int(10) not null default 0;
```

8.外键

在mysql中，是存在`外键`的。

外键存在的主要作用是：保证数据的`一致性`和`完整性`。

例如：

```
create table class (
  id int(10) primary key auto_increment,
  cname varchar(15)
);
```

有个班级表class。

然后有个student表：

```
create table student(
  id int(10) primary key auto_increment,
  name varchar(15) not null,
  gender varchar(10) not null,
  cid int,
  foreign key(cid) references class(id)
);
```

其中student表中的cid字段，保存的class表的id，这时通过`foreign key`增加了一个外键。

这时，如果你直接通过student表的id删除数据，会报异常：

```
a foreign key constraint fails
```

必须要先删除class表对于的cid那条数据，再删除student表的数据才行，这样能够保证数据的一致性和完整性。

> 顺便说一句：只有存储引擎是innodb时，才能使用外键。

如果只有两张表的关联还好，但如果有十几张表都建了外键关联，每删除一次主表，都需要同步删除十几张子表，很显然性能会非常差。

因此，互联网系统中，一般建议不使用外键。因为这类系统更多的是为了性能考虑，宁可牺牲一点数据一致性和完整性。

除了`外键`之外，`存储过程`和`触发器`也不太建议使用，他们都会影响性能。

[8000页BAT大佬写的刷题笔记，让我offer拿到手软](http://cxyroad.com/”http://www.susan.net.cn/zt/8000.html”)

9. 索引

在建表时，除了指定`主键索引`之外，还需要创建一些`普通索引`。

例如：

```
create table product_sku(
  id int(10) primary key auto_increment,
  spu_id int(10) not null,
  brand_id int(10) not null,
  name varchar(15) not null
);
```

在创建商品表时，使用spu_id（商品组表）和brand_id（品牌表）的id。

像这类保存其他表id的情况，可以增加普通索引：

```
create table product_sku (
  id int(10) primary key auto_increment,
  spu_id int(10) not null,
  brand_id int(10) not null,
```

```
  name varchar(15) not null,
  KEY `ix_spu_id` (`spu_id`) USING BTREE,
  KEY `ix_brand_id` (`brand_id`) USING BTREE
);

```

后面查表的时候，效率更高。

但索引字段也不能建的太多，可能会影响保存数据的效率，因为索引需要额外的存储空间。

建议单表的索引个数不要超过：`5`个。

如果在建表时，发现索引个数超过5个了，可以删除部分`普通索引`，改成`联合索引`。

顺便说一句：在创建联合索引的时候，需要使用注意`最左匹配原则`，不然，建的联合索引效率可能不高。

对于数据重复率非常高的字段，比如：状态，不建议单独创建普通索引。因为即使加了索引，如果mysql发现`全表扫描`效率更高，可能会导致索引失效。

如果你对索引失效问题比较感兴趣，可以看看我的另一篇文章《[聊聊索引失效的10种场景，太坑了](http://cxyroad.com/”https://mp.weixin.qq.com/s?__biz=MzkwNjMwMTgzMQ==&mid=2247491626&idx=1&sn=18fc949c06f04fe8f4c29b6fc5c66f9c&chksm=c0e838c2f79fb1d45c6f9b2ab188bb4663414690bab0718a7d46beb875e6b83e5e67ec27d2ff&token=660773166&lang=zh_CN&scene=21#wechat_redirect”)》，里面有非常详细的介绍。

10.时间字段

`时间字段`的类型，我们可以选择的范围还是比较多的，目前mysql支持：date、datetime、timestamp、varchar等。

`varchar`类型可能是为了跟接口保持一致，接口中的时间类型是String。

但如果哪天我们要通过时间范围查询数据，效率会非常低，因为这种情况没法走索引。

`date`类型主要是为了保存`日期`，比如：2020-08-20，不适合保存`日期和时间`，比如：2020-08-20 12:12:20。

而`datetime`和`timestamp`类型更适合我们保存`日期和时间`。

但它们有略微区别。

* `timestamp`：用4个字节来保存数据，它的取值范围为`1970-01-01 00:00:01` UTC ~ `2038-01-19 03:14:07`。此外，它还跟时区有关。
* `datetime`：用8个字节来保存数据，它的取值范围为`1000-01-01 00:00:00` ~ `9999-12-31 23:59:59`。它跟时区无关。

优先推荐使用`datetime`类型保存日期和时间，可以保存的时间范围更大一些。

> 温馨提醒一下，在给时间字段设置默认值是，建议不要设置成：`0000-00-00 00:00:00`，不然查询表时可能会因为转换不了，而直接报错。

11.金额字段
---

mysql中有多个字段可以表示浮点数：float、double、decimal等。

而`float`和`double`可能会丢失精度，因此推荐大家使用`decimal`类型保存金额。

一般我们是这样定义浮点数的：decimal(m,n)。

其中`n`是指`小数`的长度，而`m`是指`整数加小数`的总长度。

假如我们定义的金额类型是这样的：decimal(10,2)，则表示整数长度是8位，并且保留2位小数。

12. json字段

我们在设计表结构时，经常会遇到某个字段保存的数据值不固定的需求。

举个例子，比如：做异步excel导出功能时，需要在异步任务表中加一个字段，保存用户通过前端页面选择的查询条件，每个用户的查询条件可能都不一样。

这种业务场景，使用传统的数据库字段，不太好实现。

这时候就可以使用MySQL的json字段类型了，可以保存json格式的结构化数据。

保存和查询数据都是非常方便的。

MySQL还支持按字段名称或者字段值，查询json中的数据。

13.唯一索引

`唯一索引`在我们实际工作中，使用频率相当高。

你可以给单个字段，加唯一索引，比如：组织机构code。

也可以给多个字段，加一个联合的唯一索引，比如：分类编号、单位、规格等。

单个的唯一索引还好，但如果是联合的唯一索引，字段值出现null时，则唯一性约束可能会失效。

关于唯一索引失效的问题，感兴趣的小伙伴可以看看我的另一篇文章《[明明加了唯一索引，为什么还是产生重复数据？](http://cxyroad.com/”https://mp.weixin.qq.com/s?__biz=MzkwNjMwMTgzMQ==&mid=224749

7090&idx=1&sn=53b81535a9815853382c3a4bff8c844b&chksm=c0e82d6af79fa47ccfece23a8b0ad46ef6c647f1d2f961823eed181732c47e250ed0cbcfe6fd&token=1169141359&lang=zh_CN&scene=21#wechat_redirect”)》。

> 创建唯一索引时，相关字段一定不能包含null值，否则唯一性会失效。

14.字符集
---

mysql中支持的`字符集`有很多，常用的有：latin1、utf-8、utf8mb4、GBK等。

这4种字符集情况如下：![图片](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/aaa260a6587641a096bc8933f01e3cfd~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1080&h=247&s=14482&e=webp&b=fafafa)

`latin1`容易出现乱码问题，在实际项目中使用比较少。

而`GBK`支持中文，但不支持国际通用字符，在实际项目中使用也不多。

从目前来看，mysql的字符集使用最多的还是：`utf-8`和`utf8mb4`。

其中`utf-8`占用3个字节，比`utf8mb4`的4个字节，占用更小的存储空间。

但utf-8有个问题：即无法存储emoji表情，因为emoji表情一般需要4个字节。

由此，使用utf-8字符集，保存emoji表情时，数据库会直接报错。

所以，建议在建表时字符集设置成：`utf8mb4`，会省去很多不必要的麻烦。

15. 排序规则

不知道，你过没，在mysql中创建表时，有个`COLLATE`参数可以设置。

例如：

```
CREATE TABLE `order` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `code` varchar(20) COLLATE utf8mb4_bin NOT NULL,
  `name` varchar(30) COLLATE utf8mb4_bin NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `un_code` (`code`),
  KEY `un_code_name` (`code`,`name`) USING BTREE,
  KEY `idx_name` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 C
OLLATE=utf8mb4_bin
```

它是用来设置`排序规则`的。

字符排序规则跟字符集有关，比如：字符集如果是`utf8mb4`，则字符排序规则也是以：`utf8mb4_`开头的，常用的有：`utf8mb4_general_ci`、`utf8mb4_bin`等。

其中utf8mb4_general_ci排序规则，对字母的大小写不敏感。说得更直白一点，就是不区分大小写。

而utf8mb4_bin排序规则，对字符大小写敏感，也就是区分大小写。

说实话，这一点还是非常重要的。

假如order表中现在有一条记录，name的值是大写的YOYO，但我们用小写的yoyo去查，例如：

```
select * from order where name='yoyo';
```

```

如果字符排序规则是utf8mb4\_general\_ci，则可以查出大写的YOYO的那条数据。

如果字符排序规则是utf8mb4\_bin，则查不出来。

由此，字符排序规则一定要根据实际的业务场景选择，否则容易出现问题。

最近就业形式比较困难，为了感谢各位小伙伴对苏三一直以来的支持，我特地创建了一些工作内推群， 看看能不能帮助到大家。

你可以在群里发布招聘信息，也可以内推工作，也可以在群里投递简历找工作，也可以在群里交流面试或者工作的话题。



**进群方式**

添加苏三的**私人**：su\_san\_java，备注：**掘金+所在城市**，即可加入。

16.大字段
---

我们在创建表时，对一些特殊字段，要额外，比如：`大字段`，即占用较多存储空间的字段。

比如：用户的评论，这就属于一个大字段，但这个字段可长可短。

但一般会对评论的总长度做限制，比如：最多允许输入500个字符。

如果直接定义成`text`类型，可能会浪费存储空间，所以建议将这类字段定义成
```

`varchar`类型的存储效率更高。

当然，我还见过更大的字段，即该字段直接保存合同数据。

一个合同可能会占`几Mb`。

在mysql中保存这种数据，从系统设计的角度来说，本身就不太合理。

像合同这种非常大的数据，可以保存到`mongodb`中，然后在mysql的业务表中，保存mongodb表的id。

17.冗余字段

我们在设计表的时候，为了性能考虑，提升查询速度，有时可以冗余一些字段。

举个例子，比如：订单表中一般会有userId字段，用来记录用户的唯一标识。

但很多订单的查询页面，或者订单的明细页面，除了需要显示订单信息之外，还需要显示用户ID和用户名称。

如果订单表和用户表的数据量不多，我们可以直接用userId，将这两张表join起来，查询出用户名称。

但如果订单表和用户表的数据量都非常多，这样join是比较消耗查询性能的。

这时候我们可以通过冗余字段的方案，来解决性能问题。

我们可以在订单表中，可以再加一个userName字段，在系统创建订单时，将userId和userName同时写值。

当然订单表中历史数据的userName是空的，可以刷一下历史数据。

这样调整之后，后面只需要查询订单表，即可查询出我们所需要的数据。

不过冗余字段的方案，有利也有弊。

对查询性能有利。

但需要额外的存储空间，还可能会有数据不一致的情况，比如用户名称修改了。

我们在实际业务场景中，需要综合评估，冗余字段方案不适用于所有业务场景。

18.注释

我们在做表设计的时候，一定要把表和相关字段的注释加好。

例如下面这样的：

```
CREATE TABLE `sys_dept` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `name` varchar(30) NOT NULL COMMENT '名称',
  `pid` bigint NOT NULL COMMENT '上级部门',
  `valid_status` tinyint(1) NOT NULL DEFAULT 1 COMMENT '有效状态 1:有效 0:无效',
  `create_user_id` bigint NOT NULL COMMENT '创建人ID',
  `create_user_name` varchar(30) NOT NULL COMMENT '创建人名称',
  `create_time` datetime(3) DEFAULT NULL COMMENT '创建日期',
  `update_user_id` bigint DEFAULT NULL COMMENT '修改人ID',
  `update_user_name` varchar(30)  DEFAULT NULL COMMENT '修改人名称',
  `update_time` datetime(3) DEFAULT NULL COMMENT '修改时间',
  `is_del` tinyint(1) DEFAULT '0' COMMENT '是否删除 1：已删除 0：未删除',
  PRIMARY KEY (`id`) USING BTREE,
  KEY `index_pid` (`pid`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COMMENT='部门';
```

```

表和字段的注释，都列举的非常详细。

特别是有些状态类型的字段，比如：valid\_status字段，该字段表示有效状态， 1:有效 0:无效。

让人可以一目了然，表和字段是干什么用的，字段的值可能有哪些。

最怕的情况是，你在表中创建了很多status字段，每个字段都有1、2、3、4、5、6、7、8、9等多个值。

没有写什么注释。

谁都不知道1代表什么含义，2代表什么含义，3代表什么含义。

可能刚开始你还记得。

但系统上线使用一年半载之后，可能连你自己也忘记了这些status字段，每个值的具体含义了，埋下了一个巨坑。

由此，我们在做表设计时，一定要写好相关的注释，并且经常需要更新这些注释。

### 最后说一句(求，别白嫖我)

如果这篇文章对您有所帮助，或者有所启发的话，帮忙扫描下发二维码一下，您的支持是我坚持写作最大的动力。

求一键三连：点赞、转发、在看。

[8000页BAT大佬写的刷题笔记，让我offer拿到手软](http://cxyroad.com/”http://www.susan.net.cn/zt/8000.html”)
```

公众号：【苏三说技术】，在公众号中回复：面试、代码神器、开发手册、时间管理有超赞的粉丝福利，另外回复：加群，可以跟很多BAT大厂的前辈交流和学习。

原文链接: https://juejin.cn/post/7352789840352690185