ORM可以防止SQL Injection：如果直接用SQL语句进行操作，有可能SQL语句包含恶意代码，导致数据库被破坏。

直接写SQL，如果误写了代码，其实很难发现，而ORM则很容易发现。

ORM会自动在Model中创建一个自增主键id，除非指明不创建。

例：一个典型的Model



模型Tweet对应的表的实际名称为tweets\_tweet，其中tweets为app的名称，自动把大写变小写，如果不希望自动生成表单名，可以在class Meta中进行指定(不建议这么做)。

Tweet类中的属性名称和表中的字段名称相对应，但外键的名称有所不同。Django中的数据库中保存ForeignKey时，会自动在Model定义的对象名后加上\_id，例如上述的模型在数据库存储的就是user\_id。表示只存储User的id属性，也就是User的主键。

ForeignKey：另一个表的主键，如果表中有外键，则用外键作为主键的表和该表构成主从关系。外键的主要作用是保持数据的一致性和完整性，例如在tweet中插入一条数据时，user\_id为外键，如果插入的user\_id不存在，则会报错，这样保证了数据的有效性，不会出现一条user\_id无效的数据，这就是外键约束。

设置外键后，会引入一些问题，例如刚开始user\_id=N的用户时存在的，但后来该用户注销了，那么Tweet表中引用了该user\_id的数据就会面临删除的问题。on\_delete默认的是models.CASCADE，是所谓的级联删除，这个最好不要用，因为如果使用级联删除，会产生无法预料的后果，例如某个大明星，他的user\_id可能会存在于数据库的1亿条记录中，一旦他注销了账户，则有可能导致短时间内1亿条删除操作，并且级联删除时会把多层引用相关的都删除掉，后果太大不可控。

models.SET\_NULL，会把删除掉user\_id设置为null，但同样会有问题，例如如果进行数据分析，无法判断tweet是谁发的。

针对ForeignKey，现在流行不再使用ForeignKey，取而代之的是使用IntegerField，这样当某个user\_id被注销了，包含了这个user\_id的记录也不会被影响到，不会产生级联删除。另外不使用ForeignKey，可以进行解耦合。如果使用ForeignKey，则两张表必须在同一个数据库中，也必须在同一台实体机上。



Django中的PrimaryKey(主键)和ForeignKey(外键)都会自动帮你创建一个INDEX。

如果一个表单10个字段分别建立INDEX，10个INDEX都会生效，但会有损耗，效率会变慢，会生成多个隐式的表单(其实不是真正的表单，而是一个B+树，但可以看成表单)。

数据库存储DateTimeField等日期时间戳时，都存储UTC时间，即0时区，不要存储别的时区，这样可以在展示时很容易变换成其它时区。

Model中创建属性的一些常用的关键字参数：

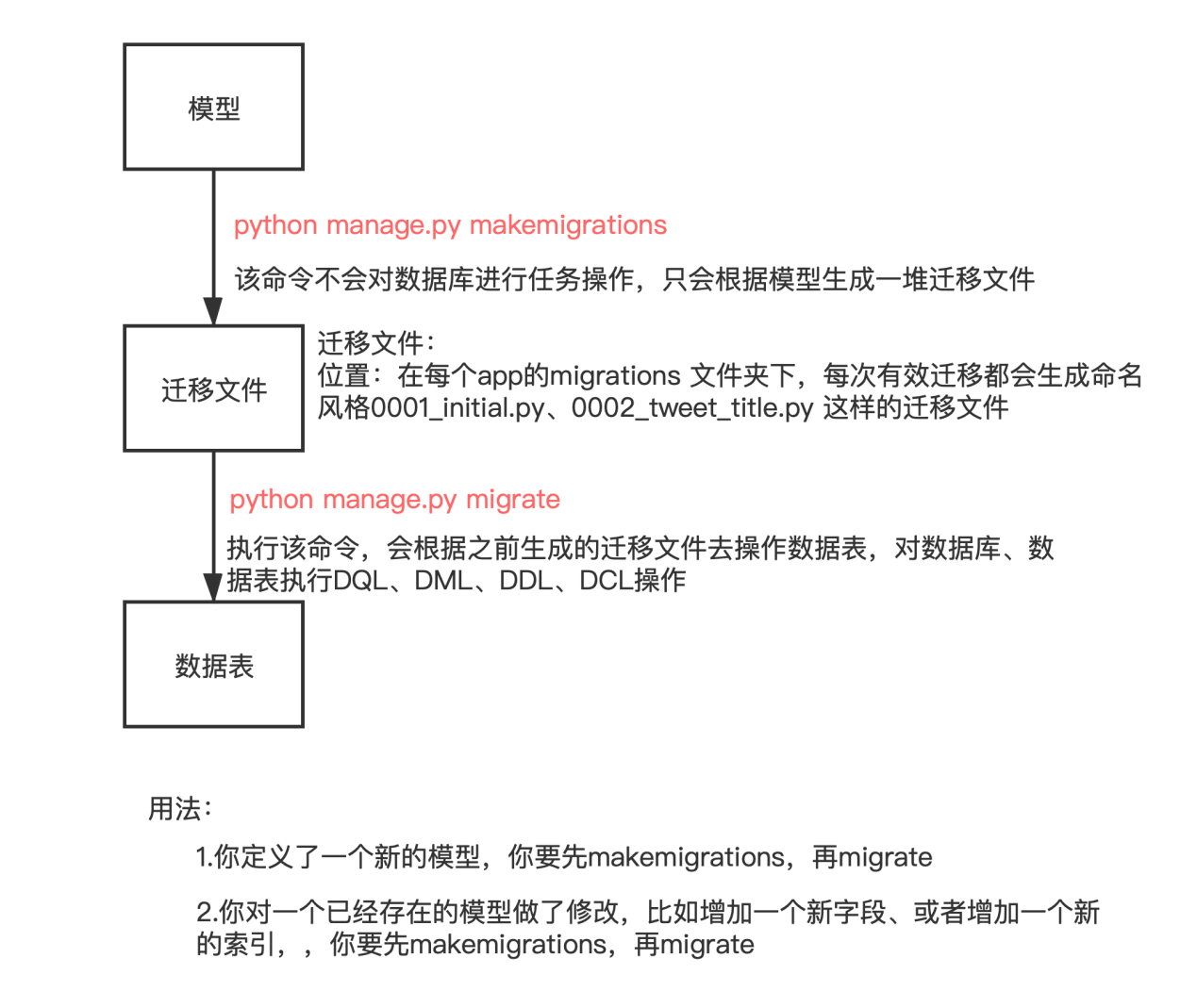
null=True，表示这个字段在数据库中允许出现空字段

blank=True，表示允许为空字符串，和上面的空字段不同

default=0，表示的是默认值。

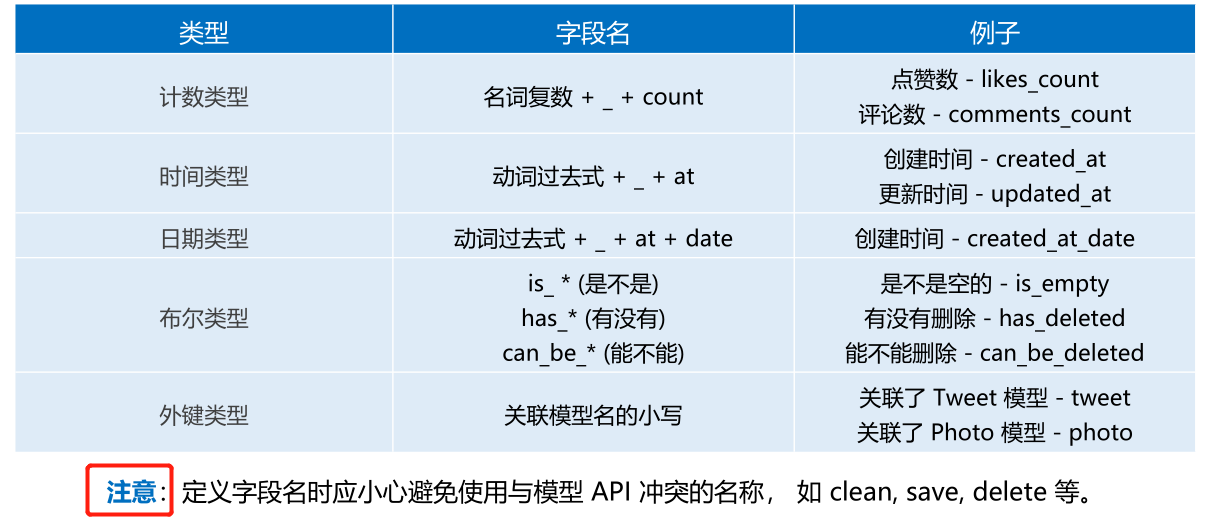
数据迁移：Migrations

Migrations：django将模型的更改(添加或删除字段，删除模型等)应用到数据库中的一种方式。

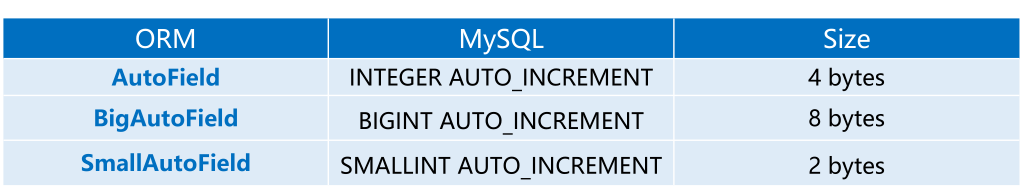


之所以先makemigrations，然后再进行migrate，类似于git add，之后进行git commit，makemigrations可以生成记录文件，知道数据库修改的中间过程，这也是比较重要的，相当于version control，从1.0到2.0到3.0，而不是直接到3.0。

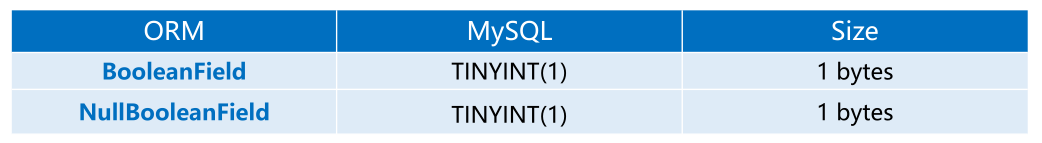
字段命名规范：



自增字段：



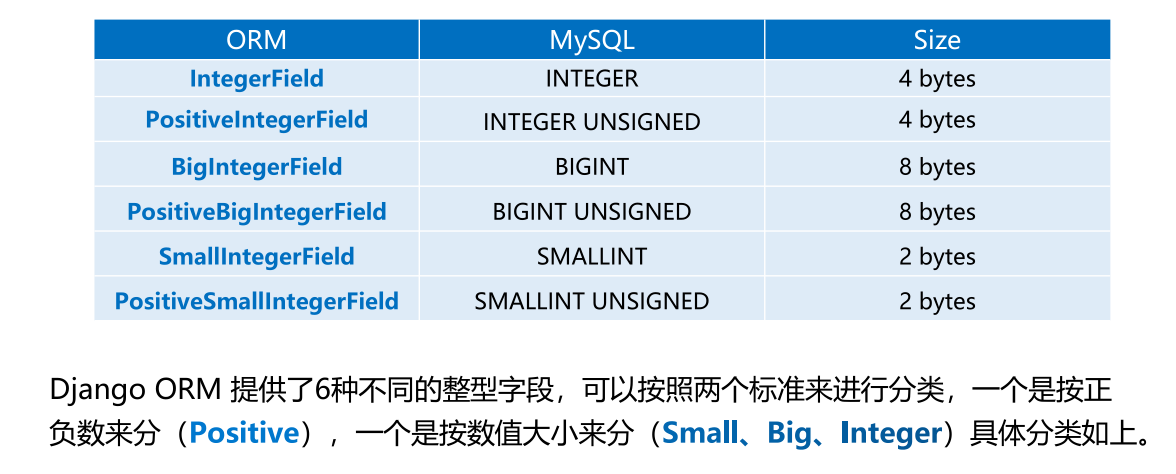
布尔字段：只能存True或False



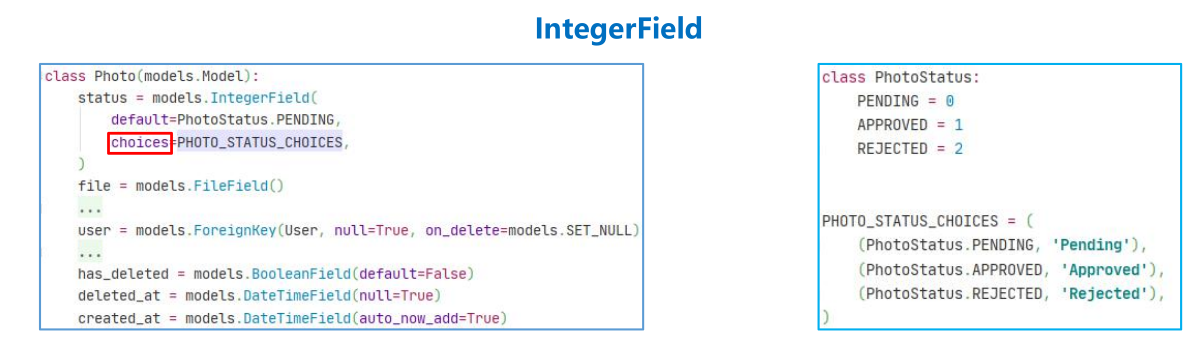
BooleanField不能为空，NullBooleanField可以为空，一般就使用NullBooleanField。

布尔字段有个很重要的作用就是存储“软删除”标记，例如当一条tweet被删除时，通常是在数据库中标记它已被删除，在一定时间后才会被真正删除掉。如果立即删除，执行真删除会耗费一定时间，影响效率，而软删除可以建立一个异步任务慢慢去删除。

整型字段：

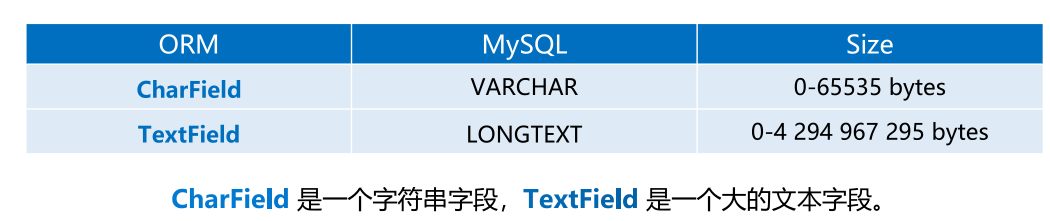


整型字段可以用来存储status状态，通常不要字符串存储status状态，尽管字符串可能看起来更清晰一些。这是因为在工程开发中，status对应的状态是会变的，例如之前叫’Rejected’，后来可能拆分成’Rejected\_by\_admin’,’Rejected\_by\_host’,’Rejected\_by\_user’，如果直接存储字符串，数据库修改起来很麻烦，而如果数据库不修改，代码中’Rejected’也要标注为’Rejected\_by\_admin’，如果不同的人来写代码，很容易就会出错。

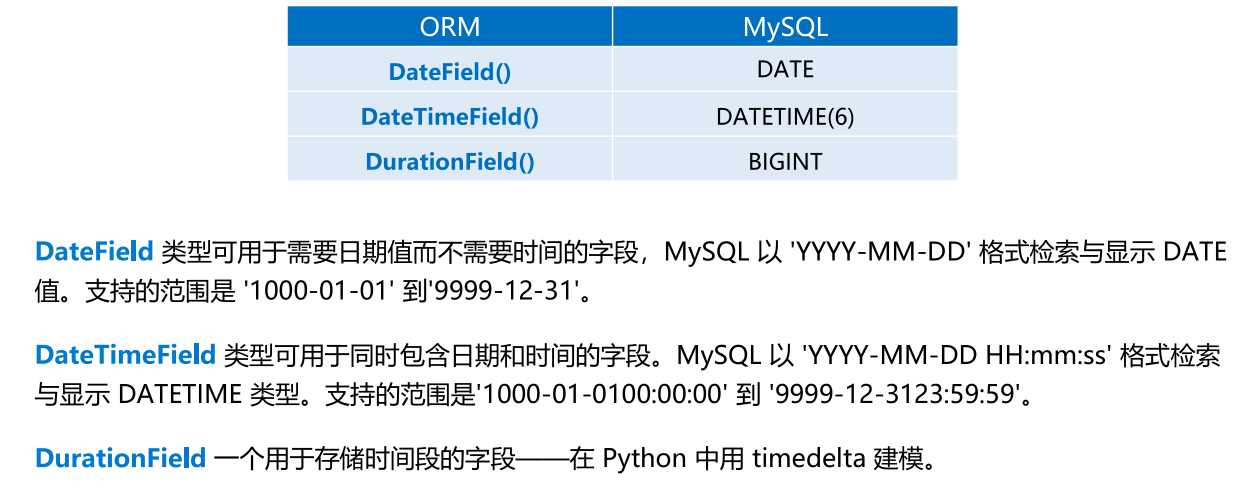


choices参数：choices的值是一个可迭代的序列，序列的元素是一个二元组。每个元组中的第一个元素是模型上设置的实际值，第二个元素是可读的名称(html页面上针对choices会有一个下拉框，里面的值就是第二个元素)

字符串字段：



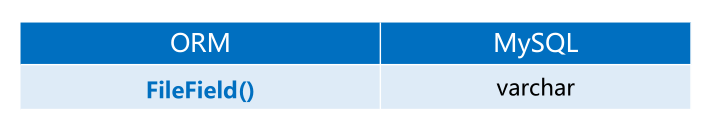
时间字段：



created\_at = DateTimeField(auto\_now\_add=True)：auto\_now\_add在创建时更新，之后created\_at就不会更新了

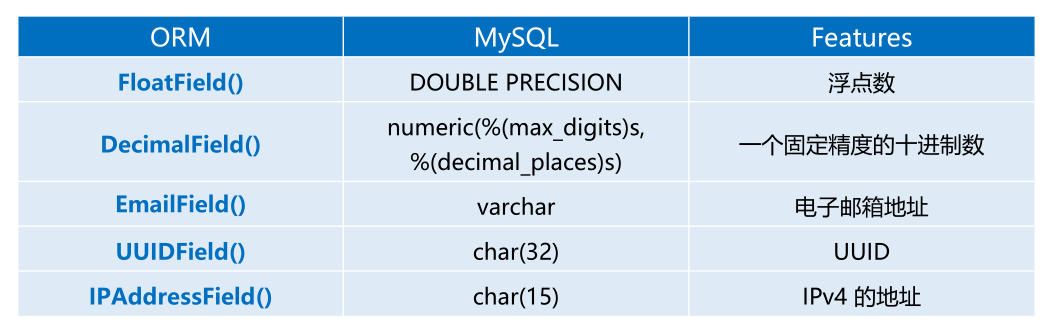
updated\_at = DateTimeField(auto\_now=True)：每次修改最后保存时updated\_at会自动更新，用当前时间戳覆盖之前的值。

文件字段：



在数据库设计阶段就应该明确主键PrimaryKey是自增int型还是UUID，如果非要进行更改，用UUID替换自增主键，可以在一段时间内自增主键和UUID并存，利用脚本将数据库的所有记录重新生成一遍，并且在程序中进行相应的处理，最后删除掉自增主键

其它类型：



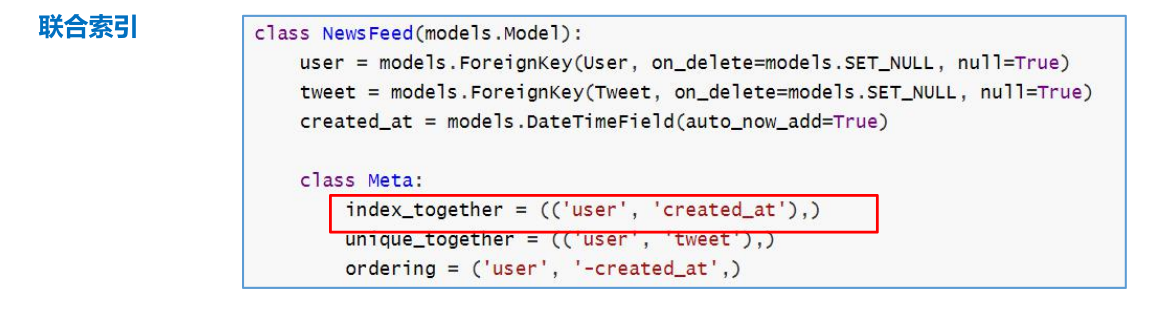
DecimalField()通常用于存储金额，金额不能用FloatField()。DecimalField本质存储了一个字符串。

索引：

设置单个索引，可以在字段创建的函数里设置db\_index=True

根据数据库某个字段y建立索引时，相当于建立一个新的表单(实际是一个B+树，但可以看成是一个表单)，这个表单是以y为关键字进行排序的，只有y和原始表的主键id。根据索引进行查询时，例如查找y最大的前10条记录，会先在索引中查找y的前10条记录(索引是按y排序的)，根据这些记录的id去原始表中查询最终的数据。

联合索引的工作原理和单个索引相同，只是生成的隐式表中会有联合索引的多个字段和主键。

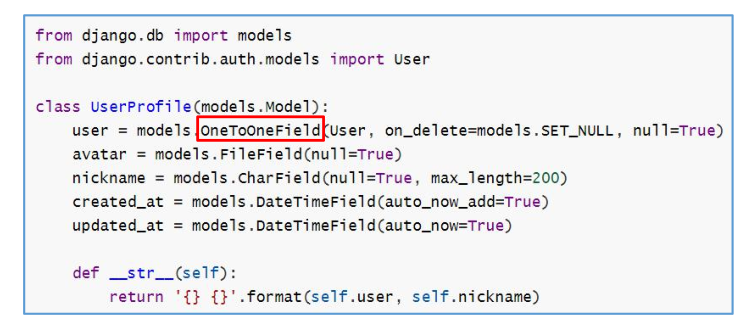


联合索引查询时，前面的字段只能指定特定的值，只有最后一项可以指定范围，例如user\_id=1 and created\_at> ‘xxxx’。也可以只指定user\_id在某个范围内，这样只使用部分索引。

表与表之间的关系：

1. 一对一的关系

一对一的关系OneToOneField类似于ForeignKey+Unique。





1. 一对多的关系，就是ForeignKey。