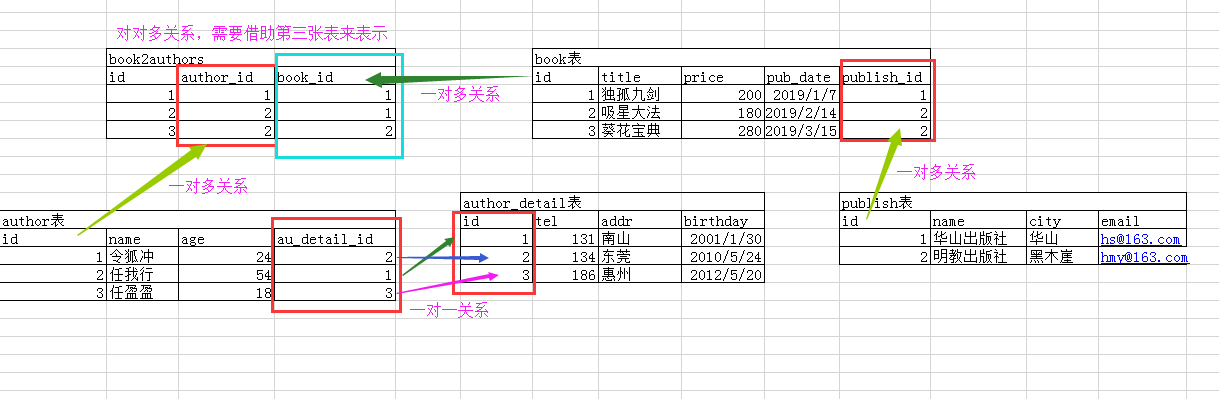
**Работа с несколькими таблицами Django ORM (дополнительно)**

**1, создать модель**

Затем мы разрабатываем соответствующие отношения между каждой таблицей через систему управления книгами.



С помощью приведенного выше отношения мы можем определить класс нашей модели.

|  |
| --- |
| **from** django.db **import** models  **class** Book(models.Model):  title = models.CharField(max\_length=32)  price = models.DecimalField(max\_digits=5, decimal\_places=2)  pub\_date = models.DateField()  publish = models.ForeignKey(**"Publish"**, on\_delete=models.CASCADE)  authors = models.ManyToManyField(**"Author"**)   **class** Publish(models.Model):  name = models.CharField(max\_length=32)  city = models.CharField(max\_length=64)  email = models.EmailField()   **class** Author(models.Model):  name = models.CharField(max\_length=32)  age = models.SmallIntegerField()  au\_detail = models.OneToOneField(**"AuthorDetail"**, on\_delete=models.CASCADE)   **class** AuthorDetail(models.Model):  gender\_choices = (  (0, **"female"**),  (1, **"male"**),  (2, **"secrecy"**),  )  gender = models.SmallIntegerField(choices=gender\_choices)  tel = models.CharField(max\_length=32)  addr = models.CharField(max\_length=64)  birthday = models.DateField() |

Обратите внимание, что не забывайте шаги django с использованием базы данных MySQL. Только после поэтапной реализации можно управлять базой данных через ORM.

2, запись в таблице операций

1. Добавьте несколько простых данных

1. опубликовать таблицу:

|  |
| --- |
| insert into multdb\_publish(name,city,email)  values  ("Huashan Publishing House", "Huashan Mountain", "hs@163.com"),  ("Mingjiao press", "Blackwood cliff", "mj@163.com") |

2. авторская таблица:

|  |
| --- |
| insert into multdb\_author(name,age,au\_detail\_id)  values  ("linghu chong",25,1),  ("ren woxing",58,2),  ("Ren yingying",23,3) |

3. Таблица данных авторов:

|  |
| --- |
| insert into multdb\_authordetail(gender,tel,addr,birthday)  values  (1,13432335433,"Huashan Mountain","1994-5-23"),  (1,13943454554,"Blackwood cliff","1961-8-13"),  (0,13878934322,"Blackwood cliff","1996-5-20") |

2. Один ко многим

|  |
| --- |
| **from** django.shortcuts **import** render **from** multdb.models **import** Book,Publish,Author **from** django.db **import** models  *#The form of object transmitted by mode I* pub\_obj = models.Publish.objects.get(pk=1) book = models.Book.objects.create(title=**"Dugu nine Swords"**, price=180, pub\_date=**"2018-10-23"**, publish=pub\_obj) *# Mode 2 transfer object id Form of* book = Book.objects.create(title=**"Dugu nine Swords"**, price=180, pub\_date=**"2018-10-23"**, publish\_id=1) |

Ядро: понять разницу между book.publish и book.publish "ID?

**3. Многие ко многим**

|  |
| --- |
| # The form of object transmitted by mode I  book = models.Book.objects.filter(title="Dugu nine Swords").first()  ling = models.Author.objects.filter(name="linghu chong").first()  ying = models.Author.objects.filter(name="Ren yingying").first()  book.authors.add(ling, ying)  # Mode 2 transfer object id Form of  book = models.Book.objects.filter(title="Dugu nine Swords").first()  ling = models.Author.objects.filter(name='linghu chong').first()  ying = models.Author.objects.filter(name='Ren yingying').first()  book.authors.add(ling.pk, ying.pk) |

Ядро: что такое book.authors.all()?

Многие другие распространенные API:

|  |
| --- |
| book\_obj.authors.remove() # Remove a specific object from the associated object collection. ====== book\_obj.authors.remove(\*[])  book\_obj.authors.clear() #Empty the collection of associated objects  book\_obj.authors.set() #Clear before setting |

**4. Кросс-табличный запрос на основе объекта**

1. Запрос один ко многим (публикация и бронирование)

Переслать запрос по полю публикации

|  |
| --- |
| # Query the city where the publishing house of the book with primary key 1 is located  book\_obj=Book.objects.filter(pk=1).first()  # book\_obj.publish is the publishing object associated with the book object with the primary key of 1  print(book\_obj.publish.city) |

Обратный запрос по таблице книг

|  |
| --- |
| # Inquire the book name published by Mingjiao Publishing House  publish=Publish.objects.get(name="Mingjiao press")  #publish.book\_set.all(): collection of all book objects associated with Mingjiao Publishing House  book\_list=publish.book\_set.all()  for book\_obj in book\_list:  print(book\_obj.title) |

2. Индивидуальный запрос (Author и AuthorDetail)

Запрос вперед (по полю: Деталь Au):

|  |
| --- |
| # Inquire Linghu Chong's phone number ling=Author.objects.filter(name="linghu chong").first()  print(ling.au\_detail.tel) |

Обратный запрос (по имени таблицы: автор):

|  |
| --- |
| # Find out the names of all the authors whose addresses are in heimuya    authorDetail\_list=AuthorDetail.objects.filter(addr="Blackwood cliff")  for obj in authorDetail\_list:  print(obj.author.name) |

3. Запрос многие ко многим (автор и книга)

Переслать запрос (по полям: авторы):

|  |
| --- |
| # All the authors' names and mobile phone numbers of Dugu Jiujian  book\_obj = Book.objects.filter(title="Dugu nine Swords").first()  authors = book\_obj.authors.all()  for author\_obj in authors:  print(author\_obj.name, author\_obj.au\_detail.tel) |

Обратный запрос (по имени таблицы: Набор книг):

|  |
| --- |
| # Query the names of all the books that Linghu rushed out  author\_obj=Author.objects.get(name="linghu chong")  book\_list=author\_obj.book\_set.all() #All books related to the author of Ling Huchong  for book\_obj in book\_list:  print(book\_obj.title) |

**Примечание**: Вы можете переопределить имя набора foo'u, установив значение related'name в определениях ForeignKey() и ManyToManyField. Например, если вы сделаете следующие изменения в модели Book:

|  |
| --- |
| publish = models.ForeignKey("Publish", on\_delete=models.CASCADE, related\_name="book\_list") |

Тогда мы увидим следующее:

|  |
| --- |
| # Query all books published by Mingjiao Publishing House  publish=Publish.objects.get(name="Mingjiao press")  book\_list=publish.book\_list.all() # Collection of all book objects associated with Mingjiao Publishing House |

**5. Кросс-табличный запрос на основе двойного подчеркивания**

Django также предоставляет интуитивно понятный и эффективный способ выражения отношения ассоциации в запросе (поиски), который может автоматически подтверждать соединение SQL JOIN. Чтобы выполнить запросы перекрестных отношений, используйте два символа подчеркивания, чтобы связать имена связанных полей между моделями, пока вы, наконец, не свяжете с нужной моделью.

|  |
| --- |
| '''  Forward query by field, reverse query by table name lowercase to tell ORM engine which table to join  ''' |

1. Запрос один ко многим

|  |
| --- |
| # Practice: Check the names and prices of all books published by Mingjiao press(One to many)  # Forward query by field:publish  queryResult=Book.objects.filter(publish\_\_name="Ming religionpress").values\_list("title","price")  # Reverse query by table name:book  queryResult=Publish.objects.filter(name="Ming religionpress").values\_list("book\_\_title","book\_\_price") |

2. Запрос многие ко многим

|  |
| --- |
| # Practice: Query the names of all the books that Linghu rushed out(Many to many)  # Forward query by field:authors:  queryResult=Book.objects.filter(authors\_\_name="linghu chong").values\_list("title")  # Reverse query by table name:book  queryResult=Author.objects.filter(name="linghu chong").values\_list("book\_\_title","book\_\_price") |

3. Индивидуальный запрос

|  |
| --- |
| # Query Ling Hu Chong's mobile number    # Forward query  ret=Author.objects.filter(name="linghu chong").values("au\_detail\_\_tel")   　　# inverse query  　　ret=AuthorDetail.objects.filter(author\_\_name="linghu chong").values("tel") |

4. Продвинутая практика (непрерывная таблица пролетов)

|  |
| --- |
| # Practice: Check the names of all the books published by Mingjiao publishing house and the names of the authors  # Forward query  queryResult=Book.objects.filter(publish\_\_name="Ming religionpress").values\_list("title","authors\_\_name")  # inverse query  queryResult=Publish.objects.filter(name="Ming religionpress").values\_list("book\_\_title","book\_\_authors\_\_age","book\_\_authors\_\_name")  # Practice: Names of all books published by authors with mobile numbers starting with 132 and the name of the publishing house  # Mode 1:  queryResult=Book.objects.filter(authors\_\_au\_detail\_\_tel\_\_startswith="132").values\_list("title","publish\_\_name")   　　 # Mode 2:  　　 ret=Author.objects.filter(au\_detail\_\_tel\_\_startswith="132").values("book\_\_title","book\_\_publish\_\_name") |

**6. Совокупный запрос и групповой запрос(**Aggregate query and group query)

**1. совокупный запрос (**aggregate query)

Давайте почувствуем это на примере.

|  |
| --- |
| # Calculate the average price of all books  books = models.Book.objects.aggregate(Avg("price"))  books = models.Book.objects.aggregate(avg\_price=Avg("price")) # Specify the key of the dictionary as avg\_ |

aggregate () является предложением завершения QuerySet, что означает, что он возвращает словарь, содержащий несколько пар ключ-значение. Имя ключа — это идентификатор агрегированного значения, а значение — вычисленное агрегированное значение. Имя ключа автоматически генерируется по имени поля и агрегатной функции. Если вы хотите указать имя для агрегированного значения, вы можете указать его в агрегатном предложении (как в приведенном выше примере).

Если вы хотите сгенерировать более одного агрегата, вы можете добавить еще один параметр в предложение агрегата(). Итак, если вы хотите узнать максимальные и минимальные цены на все книги, вы можете сделать запрос следующим образом:

|  |
| --- |
| from django.db.models import Avg, Max, Min  # Calculate the average, most expensive, and least expensive prices of all books  books = models.Book.objects.aggregate(Avg("price"), Max("price"), Min("price")) |

**2. Аннотировать групповой запрос**

Прежде чем мы поговорим об этом, давайте вспомним, как запрашивать операторы SQL, которые мы изучили ранее. Давайте сравним код ORM, чтобы углубить наше понимание.

Давайте сначала попрактикуемся в одиночных часах:

Структура таблицы такая:

|  |
| --- |
| class Emp(models.Model):  name = models.CharField(max\_length=32)  age = models.IntegerField()  salary = models.DecimalField(max\_digits=8, decimal\_places=2)  dep = models.CharField(max\_length=32)  province = models.CharField(max\_length=32) |

Подготовьте данные:

|  |
| --- |
| INSERT INTO `bkm`.`app01\_emp` (`id`, `name`, `age`, `salary`, `dep`, `province`) VALUES ('1', 'linghu chong', '24', '6000.00', 'Sales Department', 'Henan');  INSERT INTO `bkm`.`app01\_emp` (`id`, `name`, `age`, `salary`, `dep`, `province`) VALUES ('2', 'Ren yingying', '18', '8000.00', 'Guangong Department', 'Guangdong');  INSERT INTO `bkm`.`app01\_emp` (`id`, `name`, `age`, `salary`, `dep`, `province`) VALUES ('3', 'ren woxing', '56', '10000.00', 'Sales Department', 'Guangdong');  INSERT INTO `bkm`.`app01\_emp` (`id`, `name`, `age`, `salary`, `dep`, `province`) VALUES ('4', 'Yue Lingshan', '19', '6000.00', 'Guangong Department', 'Henan');  INSERT INTO `bkm`.`app01\_emp` (`id`, `name`, `age`, `salary`, `dep`, `province`) VALUES ('5', 'little dragon maiden', '20', '8000.00', 'Guangong Department', 'Hebei'); |

Операция запроса:

|  |
| --- |
| # Query the name of each department and the corresponding number of employees  SQL:  select dep, count(1) from emp group by dep;  ORM:  models.Emp.objects.values('dep').annotate(c=Count('id'))  # Query the name of each department and the average salary of the corresponding employee  SQL:  select dep, avg(salary) from app01\_emp GROUP BY dep;  ORM:  models.Emp.objects.values('dep').annotate(a=Avg('salary')) title = models.CharField(max\_length=32) |

Хорошо, дальше. Давайте поиграем в групповой запрос с несколькими таблицами.

Структура таблицы такая:

|  |
| --- |
| class Emps(models.Model):  name = models.CharField(max\_length=32)  age = models.IntegerField()  salary = models.DecimalField(max\_digits=8, decimal\_places=2)  dep = models.ForeignKey("Dep", on\_delete=models.CASCADE)  province = models.CharField(max\_length=32)  class Dep(models.Model):  title = models.CharField(max\_length=32) |

Подготовьте данные:

1. Деп стол:

|  |
| --- |
| INSERT INTO `bkm`.`app01\_dep` (`id`, `title`) VALUES ('1', 'Sales Department');  INSERT INTO `bkm`.`app01\_dep` (`id`, `title`) VALUES ('2', 'Guangong Department'); |

2. Таблица импульсов:

|  |
| --- |
| INSERT INTO `bkm`.`app01\_emps` (`id`, `name`, `age`, `salary`, `province`, `dep\_id`) VALUES ('2', 'linghu chong', '24', '8000.00', 'Henan', '1');  INSERT INTO `bkm`.`app01\_emps` (`id`, `name`, `age`, `salary`, `province`, `dep\_id`) VALUES ('3', 'Ren yingying', '18', '9000.00', 'Guangdong', '2');  INSERT INTO `bkm`.`app01\_emps` (`id`, `name`, `age`, `salary`, `province`, `dep\_id`) VALUES ('4', 'ren woxing', '57', '10000.00', 'Guangdong', '1');  INSERT INTO `bkm`.`app01\_emps` (`id`, `name`, `age`, `salary`, `province`, `dep\_id`) VALUES ('5', 'Yue Lingshan', '19', '6000.00', 'Henan', '2');  INSERT INTO `bkm`.`app01\_emps` (`id`, `name`, `age`, `salary`, `province`, `dep\_id`) VALUES ('6', 'little dragon maiden', '20', '8000.00', 'Hebei', '2'); |

Операция запроса:

|  |
| --- |
| # Query the name of each department and the corresponding number of employees  SQL:  SELECT app01\_dep.title,count(app01\_emps.id) FROM app01\_emps LEFT JOIN app01\_dep ON app01\_emps.dep\_id = app01\_dep.id GROUP BY app01\_emps.dep\_id;  ORM:  models.Emps.objects.values("dep\_\_title").annotate(c=Count("id"))  # Query the name of each department and the average salary of the corresponding employee  SQL:  SELECT app01\_dep.title,avg(app01\_emps.salary) FROM app01\_emps LEFT JOIN app01\_dep ON app01\_emps.dep\_id = app01\_dep.id GROUP BY app01\_emps.dep\_id;  ORM:  models.Emps.objects.values("dep\_\_title").annotate(a=Avg("salary")) |

**Резюме**: суть группового запроса между таблицами заключается в объединении связанных таблиц в таблицу, а затем в групповом запросе в соответствии с идеей единой таблицы.

3. Опросное упражнение

1. Практика: посчитайте цену самой дешевой книги каждого издательства

|  |
| --- |
| # Exercise: count the price of the cheapest book in each publishing house  SQL:  select name, min(app01\_book.price)  from app01\_publish LEFT  JOIN  app01\_book  on  app01\_book.publish\_id = app01\_publish.id  GROUP  BY  app01\_publish.name;  ORM:  models.Publish.objects.values("name").annotate(min\_price=Min("book\_\_price")) |

2. Упражнение: посчитайте количество авторов каждой книги

|  |
| --- |
| Exercise: count the number of authors of each book  SQL:  SELECT  title,  count(NAME)  FROM  app01\_book  LEFT JOIN app01\_book\_authors ON app01\_book.id = app01\_book\_authors.book\_id  LEFT JOIN app01\_author ON app01\_book\_authors.author\_id = app01\_author.id  GROUP BY  app01\_book.id;  ORM:  models.Book.objects.annotate(num=Count('authors\_\_name')).values("title", "num") |

3. Упражнение: подсчитайте количество авторов каждой книги, начиная с «Девять».

|  |
| --- |
| # Exercise: count the number of authors of each book beginning with "Nine"  SQL:  SELECT  title,  count(NAME)  FROM  app01\_book  LEFT JOIN app01\_book\_authors ON app01\_book.id = app01\_book\_authors.book\_id  LEFT JOIN app01\_author ON app01\_book\_authors.author\_id = app01\_author.id  WHERE  app01\_book.title LIKE 'Nine%'  GROUP BY  app01\_book.id;  ORM:  models.Book.objects.filter(title\_\_startswith="Nine").annotate(num=Count('authors\_\_name')).values("title", "num") |

4. Упражнение: посчитайте названия книг более чем одного автора.

|  |
| --- |
| # Exercise: count books by more than one author  SQL:  SELECT  title,  count(NAME) AS num  FROM  app01\_book  LEFT JOIN app01\_book\_authors ON app01\_book.id = app01\_book\_authors.book\_id  LEFT JOIN app01\_author ON app01\_author.id = app01\_book\_authors.author\_id  GROUP BY  app01\_book.id  HAVING  num > 1;  ORM:  models.Book.objects.annotate(num\_author=Count("authors\_\_name")).filter(num\_author\_\_gt=1).values("title") |

5. Упражнение: отсортируйте набор запросов по количеству авторов Книги.

|  |
| --- |
| # Exercise: pair query sets according to the number of authors in a Book QuerySet Sort  SQL:  SELECT  title,  count(author\_id) AS num  FROM  app01\_book  LEFT JOIN app01\_book\_authors ON app01\_book.id = app01\_book\_authors.book\_id  LEFT JOIN app01\_author ON app01\_author.id = app01\_book\_authors.author\_id  GROUP BY  app01\_book.id  ORDER BY  num;  ORM:  models.Book.objects.annotate(num\_author=Count("authors\_\_name")).order\_by("num\_author").values("title", "num\_author") |

6. Упражнение: запросите общую стоимость книг, опубликованных каждым автором.

|  |
| --- |
| Exercise: find out the total price of each author's book  SQL:  SELECT  NAME,  sum(price)  FROM  app01\_author  LEFT JOIN app01\_book\_authors ON app01\_author.id = app01\_book\_authors.author\_id  LEFT JOIN app01\_book ON app01\_book.id = app01\_book\_authors.book\_id  GROUP BY  app01\_author.id;  ORM:  models.Author.objects.annotate(total=Sum("book\_\_price")).values('name', 'total') |

7. F-запрос и Q-запрос

**1. F-запрос**

Во всех приведенных выше примерах созданный нами фильтр просто сравнивает значение поля с константой. Что нам делать, если мы хотим сравнить значения двух полей?

Django предоставляет F() для такого сравнения. Экземпляр F() может ссылаться на поля в запросе для сравнения значений двух разных полей в одном и том же экземпляре модели.

|  |
| --- |
| # Query the person whose salary is greater than age  from django.db.models import F, Q  models.Emp.objects.filter(salary\_\_gt=F('age')) |

Django поддерживает операции сложения, вычитания, умножения, деления и по модулю между объектами F(), а также между объектами F() и константами.

|  |
| --- |
| # Query people whose salary is less than twice the age value  models.Emp.objects.filter(salary\_\_lt=F("age") \* 2) |

Функцию F также можно использовать для модификации, например, для увеличения цены каждой книги на 100 юаней.

|  |
| --- |
| models.Book.objects.update(price=F('price') + 100) |

**2. Q-запрос**

Запросы параметров ключевого слова в filter() И другие методы являются "И" вместе. Если вам нужно выполнить более сложные запросы (например, операторы OR), вы можете использовать объект Q.

|  |
| --- |
| from django.db.models import Q  Q(title\_\_startswith="Nine") |

Объекты Q можно комбинировать с &, | и ￣ (и или нет) операторы. Когда оператор используется для двух объектов Q, он создает новый объект Q.

|  |
| --- |
| # Search for books whose price is more than 300 or whose name begins with nine    models.Book.objects.filter(Q(price\_\_gt=300)|Q(title\_\_startswith="Nine")) |

Эквивалентно следующему предложению SQL where:

|  |
| --- |
| WHERE (`app01\_book`.`price` > 300 OR `app01\_book`.`title` LIKE BINARY 'Nine%') |

Для комплексного использования см. пример:

|  |
| --- |
| # Search for books with a price greater than 300 or not march 2019  models.Book.objects.filter(Q(price\_\_gt=300)|~Q(Q(pub\_date\_\_year=2019)&Q(pub\_date\_\_month=3))) |