

## **Лекция 5. Модели. Часть 2.**

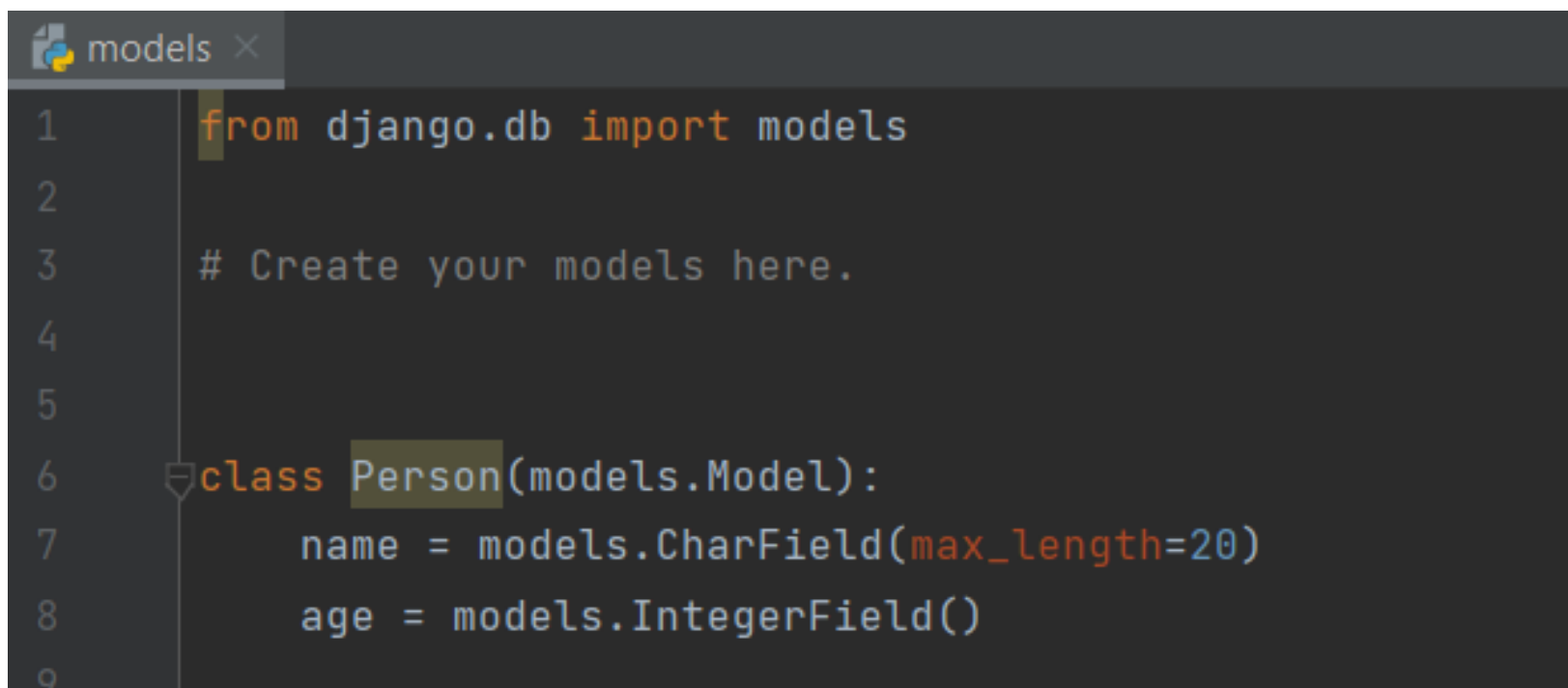


# Операции с моделями (CRUD) в интерактивной консоли фреймворка Django

Операции манипулирования данными: Create, Read, Update, Delete – CRUD.

При создании модели Django наследуют поведение от класса `django.db.models.Model`, который предоставляет ряд базовых операций манипулирования данными.

Рассмотрим эти операции в интерактивной консоли фреймворка Django на примере модели `Person` (файл `models.py`):



```
models x
1  from django.db import models
2
3  # Create your models here.
4
5
6  class Person(models.Model):
7      name = models.CharField(max_length=20)
8      age = models.IntegerField()
9
```



Наберем в терминале команду: **Python manage.py shell**  
и перейдем в интерактивную консоль фреймворка Django.

```
PS C:\Users\user\PycharmProjects\Kino\film> python manage.py shell
Python 3.9.0 (tags/v3.9.0:9cf6752, Oct 5 2020, 15:34:40) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> 
```

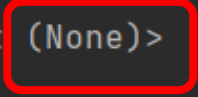
Импортируем модель: **from artist.models import Person**

```
>>> from artist.models import Person
>>> 
```



Попробуем добавить запись в таблицу Person:

```
>>> Person(name="Джанго Акробат",age=18)
<Person: Person object (None)>
>>>
```



Но запись в таблице еще не будет создана. **Значение id – None**. Создание экземпляра класса еще не означает добавления в таблицу. Модели во фреймворке Django по умолчанию являются «ленивыми».

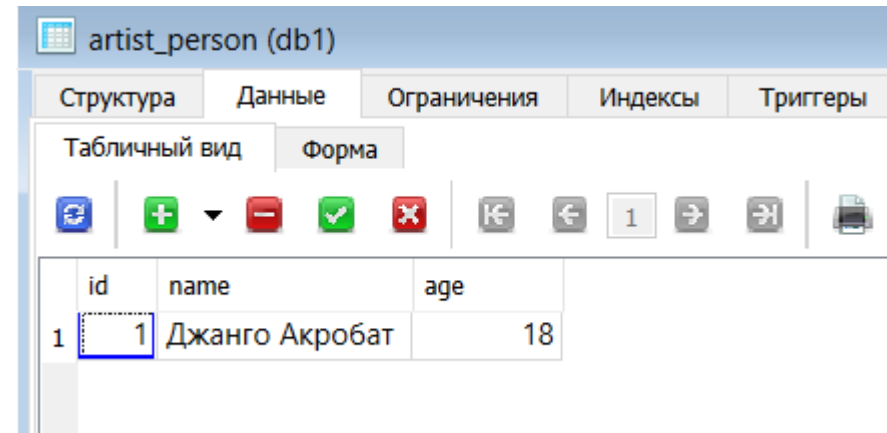
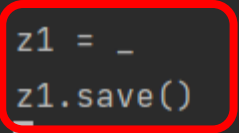
**Как указать фреймворку, что надо сохранить запись в таблице?**


Создадим переменную, например, z1 и присвоим ей то, что получили на предыдущем этапе:

z1 = \_ (подчерк – это объект, который сохраняет последние действия)

z1.save() - сохраняем данные в таблицу и посмотрим таблицу в SQLiteStudio:

```
>>> from artist.models import Person
>>> Person(name="Джанго Акробат",age=18)
<Person: Person object (None)>
>>> z1 = _
>>> z1.save()
>>>
```



Структура	Данные	Ограничения	Индексы	Триггеры
Табличный вид    Форма				
				
id	name	age		
1	Джанго Акробат	18		



Используя переменную z1, мы можем оперировать со всеми полями записи таблицы.

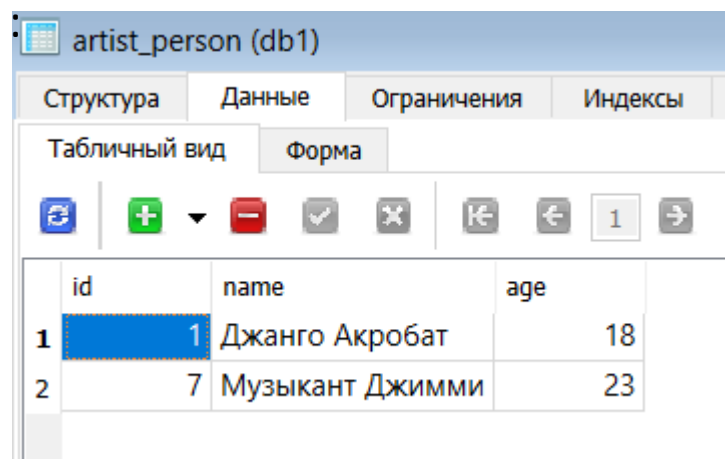
```
>>> z1.name
'Джанго Акробат'
>>> z1.age
18
>>>
```

Чтобы посмотреть, какой запрос был выполнен для добавления записи:

```
>>> from django.db import connection
>>> connection.queries
[{'sql': 'INSERT INTO "artist_person" ("name", "age") VALUES (\''Джанго Акробат\'', 18)', 'time': '0.031'}]
>>>
```

Добавим еще запись и посмотрим таблицу в SQLiteStudio:

```
>>> z2 = Person(name="Музыкант Джимми", age=23)
>>> z2.save()
>>>
```



The screenshot shows the SQLiteStudio interface with the 'artist\_person' table selected. The table has two columns: 'id' and 'name', and one column: 'age'. The table contains two records: one with id 1, name 'Джанго Акробат', and age 18; and another with id 2, name 'Музыкант Джимми', and age 23.

	id	name	age
1	1	Джанго Акробат	18
2	7	Музыкант Джимми	23



Если мы повторим вывод коллекции `connection.queries`, то увидим оба запроса:

```
>>> connection.queries
```

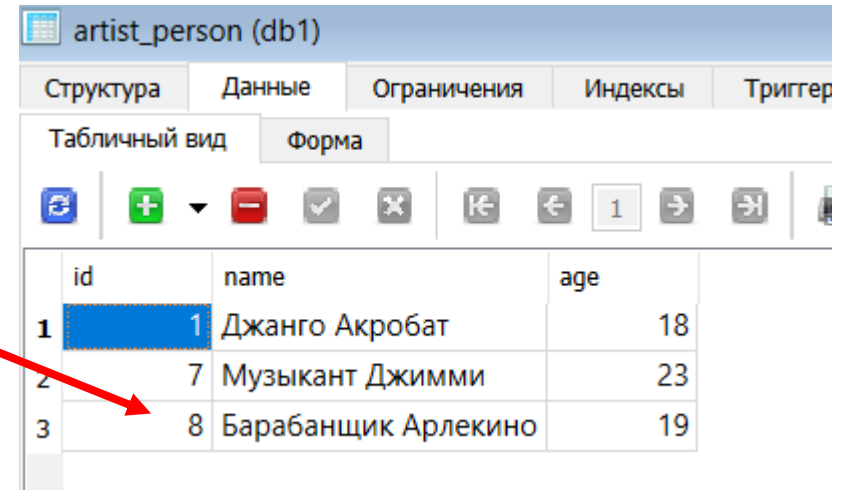
```
[{'sql': 'INSERT INTO "artist_person" ("name", "age") VALUES (\'Джанго Акробат\',  
18)', 'time': '0.031'},
```

```
{'sql': 'INSERT INTO "artist_person" ("name", "age") VALUES (\'Музыкант  
Джимми',  
23)', 'time': '0.016'}]
```



Можно добавить  
запись так:

```
>>> z3=Person()
>>> z3.name='Барабанщик Арлекино'
>>> z3.age=19
>>> z3.save()
>>>
```



Структура			
Данные			
Ограничения			
Индексы			
Триггер			
Табличный вид			
Форма			
id			
name			
age			
1	Джанго Акробат	18	
2	Музыкант Джимми	23	
3	Барабанщик Арлекино	19	

Благодаря тому, что запросы являются “ленивыми”, можно их создавать, а сохранять данные позже по мере надобности.

Каждый класс модели содержит специальный статический объект # objects. Этот объект берется из базового класса Model и представляет ссылку на специальный класс manager:

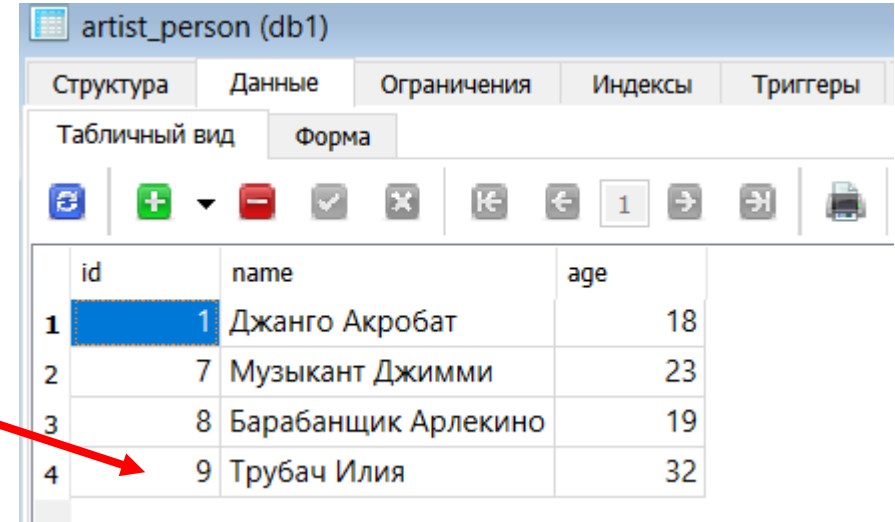
```
>>> # objects
>>> Person.objects
<django.db.models.manager.Manager object at 0x000001472336BAC0>
>>>
```

Этот объект называют менеджером записей. Он имеет несколько полезных методов.



Метод create(). Не требует .save(). Запись сразу попадает в базу данных.

```
>>> z4=Person.objects.create(name='Трубач Илия', age=32)
```



artist\_person (db1)

Структура Данные Ограничения Индексы Триггеры

Табличный вид Форма

	id	name	age
1	1	Джанго Акробат	18
2	7	Музыкант Джимми	23
3	8	Барабанщик Арлекино	19
4	9	Трубач Илия	32

```
>>> z4=Person.objects.create(name='Трубач Илия', age=32)
>>> z4.pk
9
>>> z4.name
'Трубач Илия'
>>> z4.age
32
>>>
```

Через переменную z4 можем  
посмотреть значения полей  
записи





Можно сделать то же самое без использования переменной:

```
>>> Person.objects.create(name='Саксофонист Билли', age=27)
<Person: Person object (10)>
>>>
```

artist\_person (db1)

Структура Данные Ограничения Индексы

Табличный вид Форма

	id	name	age
1	1	Джанго Акробат	18
2	7	Музыкант Джимми	23
3	8	Барабанщик Арлекино	19
4	9	Трубоч Илия	32
5	10	Саксофонист Билли	27



Как при помощи менеджера **читать данные из таблицы:**

```
>>> Person.objects.all()  
<QuerySet [  
  <Person: Person object (1)>, <Person: Person object (7)>, <Person: Person object (8)>, <Person: Person object (9)>, <Person: Person object (10)>]>
```

Чтобы выводились названия полей, надо в описание модели добавить:

```
def __str__(self):  
    return self.name
```

```
models x  
1  from django.db import models  
2  
3  # Create your models here.  
4  
5  
6  class Person(models.Model):  
7      name = models.CharField(max_length=20)  
8      age = models.IntegerField()  
9  
10     def __str__(self):  
11         return self.name  
12
```

self – это ссылка на текущий экземпляр записи.

Чтобы изменения в файле models сохранились, надо перезапустить оболочку **shell**.

Для этого сначала надо выйти **exit()**

Потом снова войти: **python manage.py shell**



```
>>> from artist.models import Person
>>> Person.objects.all()
<QuerySet [<Person: Джанго Акробат>, <Person: Музыкант Джимми>, <Person: Барабанщик Арлекино>, <Person: Трубоч Илия>, <Person: Саксофонист Билли>]>
>>> █
```

```
>>> z = _
>>> z[0]
<Person: Джанго Акробат>
>>> z[1]
<Person: Музыкант Джимми>
>>> z[2]
<Person: Барабанщик Арлекино>
>>> █
```

Вопрос: как из списка **QuerySet** выбирать отдельные записи и поля записей?

Присвоим список переменной: `z = _` и далее к элементу списка **QuerySet** можно обращаться по индексу: `z[0]`, `z[1]` и т.д.

```
>>> z[0].name
'Джанго Акробат'
>>> z[0].age
18
>>> z[1].name
'Музыкант Джимми'
>>> z[1].age
23
>>> █
```

И для вывода полей: `z[0].name` `z[0].age` и т. д.



```
>>> len(z)
5
>>> for i in z:
...     print(i.name)
...
Джанго Акробат
Музыкант Джимми
Барабанщик Арлекино
Трубач Илия
Саксофонист Билли
>>> █
```

Для вывода числа записей : `len(z)`,  
для вывода поля `name` всех записей – цикл **for**

```
>>> Person.objects.filter(id=5)
<QuerySet []>
>>> Person.objects.filter(id='5')
<QuerySet []>
>>> Person.objects.filter(id=8)
<QuerySet [<Person: Барабанщик Арлекино>]>
>>> █
```

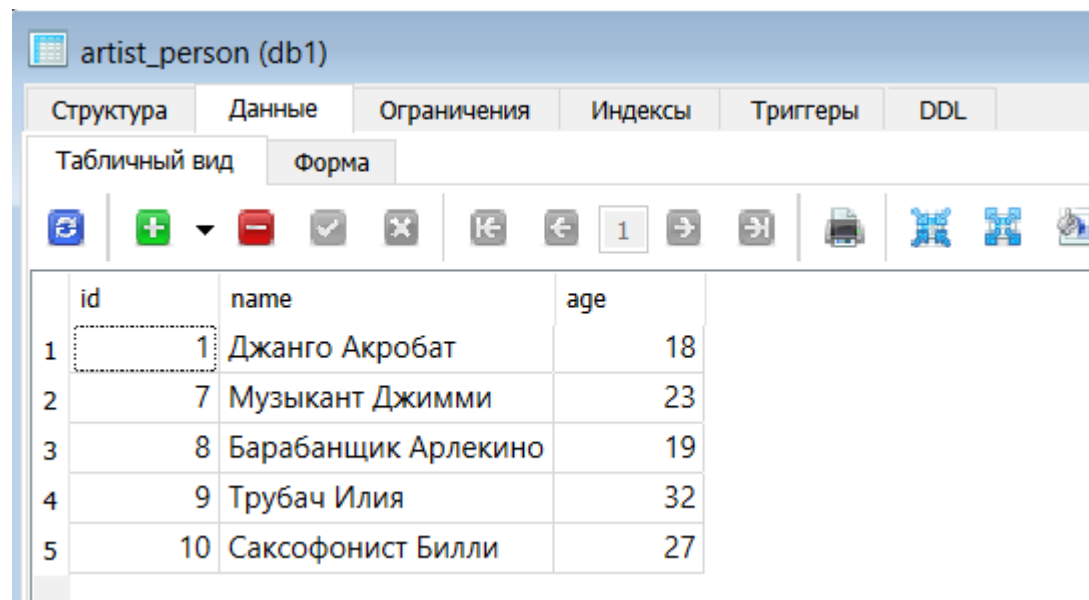
Использование метода `filter()`



Как при наложении фильтра задать условие для вывода нескольких записей?

- <имя атрибута>\_\_gte >=
- < имя атрибута >\_\_lte <=

```
>>> Person.objects.filter(id__gte=6)
<QuerySet [<Person: Музыкант Джимми>, <Person: Барабанщик Арлекино>, <Person: Трубач Илия>, <Person: Саксофонист Билли>]>
>>> Person.objects.filter(id__lte=4)
<QuerySet [<Person: Джанго Акробат>]>
>>>
```



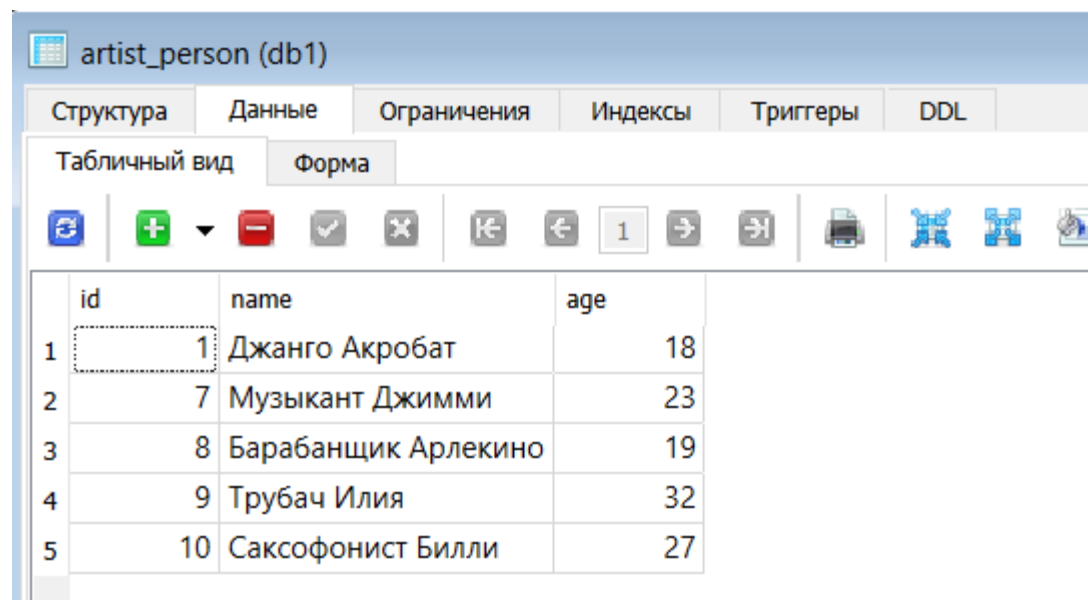
The screenshot shows a database management interface for a table named 'artist\_person' in a database named 'db1'. The interface includes tabs for 'Структура' (Structure), 'Данные' (Data), 'Ограничения' (Constraints), 'Индексы' (Indexes), 'Триггеры' (Triggers), and 'DDL'. The 'Данные' tab is selected, and the view is set to 'Табличный вид' (Table view). Below the tabs is a toolbar with various icons for table operations. The table itself has three columns: 'id', 'name', and 'age'. It contains five rows of data, with the first row highlighted.

	id	name	age
1	1	Джанго Акробат	18
2	7	Музыкант Джимми	23
3	8	Барабанщик Арлекино	19
4	9	Трубач Илия	32
5	10	Саксофонист Билли	27



## Метод exclude() - за исключением

```
>>> Person.objects.exclude(id=7)  
<QuerySet [<Person: Джанго Акробат>, <Person: Барабанщик Арлекино>, <Person: Трубач Илия>, <Person: Саксофонист Билли>]>
```



The screenshot shows a database management interface for a table named 'artist\_person' in a database named 'db1'. The interface has several tabs: 'Структура' (Structure), 'Данные' (Data), 'Ограничения' (Constraints), 'Индексы' (Indexes), 'Триггеры' (Triggers), and 'DDL'. The 'Данные' tab is selected, and within it, the 'Табличный вид' (Table view) sub-tab is active. Below the tabs is a toolbar with various icons for table operations. The main area displays a table with the following data:

	id	name	age
1	1	Джанго Акробат	18
2	7	Музыкант Джимми	23
3	8	Барабанщик Арлекино	19
4	9	Трубач Илия	32
5	10	Саксофонист Билли	27



Для выбора конкретной записи можно использовать методы `filter()` и `get()`.

Но использовать метод `get()` предпочтительнее, потому что в случае, если запись не будет найдена, метод `get()` всегда сгенерирует исключение.

```
>>> Person.objects.filter(id=7)
<QuerySet [<Person: Музыкант Джимми>]>
>>> Person.objects.get(id=7)
<Person: Музыкант Джимми>
>>> 
```



Для сортировки записей можно использовать метод `order_by()`

```
>>> Person.objects.order_by('name')
<QuerySet [<Person: Барабанщик Арлекино>, <Person: Джанго Акробат>, <Person: Музыкант Джимми>, <Person: Саксофонист Билли>, <Person: Трубач Илия>]>
>>> Person.objects.filter(pk__gte=7).order_by('name')
<QuerySet [<Person: Барабанщик Арлекино>, <Person: Музыкант Джимми>, <Person: Саксофонист Билли>, <Person: Трубач Илия>]>
>>>
```

Для обратной сортировки следует указать знак минус в имени поля: `'-name'`

```
>>> Person.objects.filter(pk__gte=7).order_by('-name')
<QuerySet [<Person: Трубач Илия>, <Person: Саксофонист Билли>, <Person: Музыкант Джимми>, <Person: Барабанщик Арлекино>]>
>>>
```

artist\_person (db1)

Структура Данные Ограничения Индексы Три

Табличный вид Форма

	id	name	age
1	1	Джанго Акробат	18
2	7	Музыкант Джимми	23
3	8	Барабанщик Арлекино	19
4	9	Трубач Илия	32
5	10	Саксофонист Билли	27





## Как можно изменять записи в таблице.

Сначала читаем запись в переменную: `z = Person.objects.get(pk=7)`

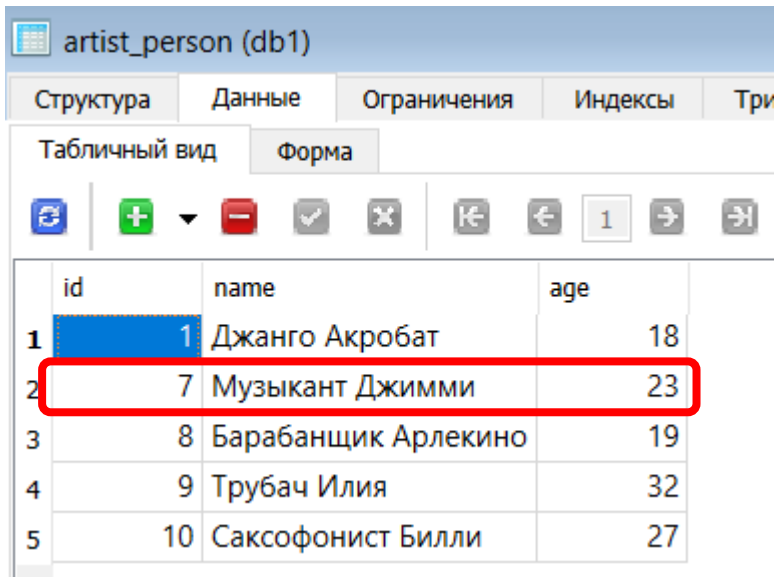
Затем присвоить новые значения полям и сохранить объект:

`z.name = "Флейтист Амиго"`

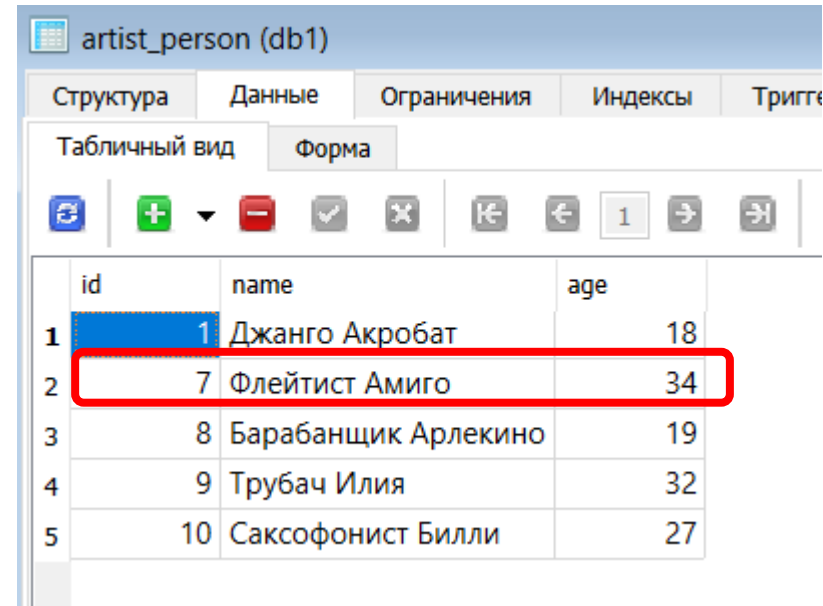
`z.age = 34`

`z.save()`

```
>>> z = Person.objects.get(pk=7)
>>> z.name = "Флейтист Амиго"
>>> z.age = 34
>>> z.save()
>>>
```



	id	name	age
1	1	Джанго Акробат	18
2	7	Флейтист Амиго	23
3	8	Барабанщик Арлекино	19
4	9	Трубач Илия	32
5	10	Саксофонист Билли	27



	id	name	age
1	1	Джанго Акробат	18
2	7	Флейтист Амиго	34
3	8	Барабанщик Арлекино	19
4	9	Трубач Илия	32
5	10	Саксофонист Билли	27

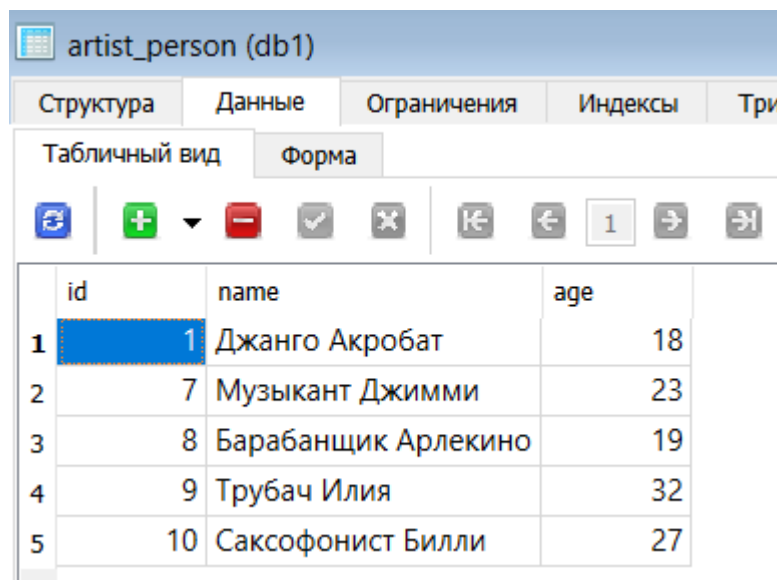


## Удаление записей. Метод delete().

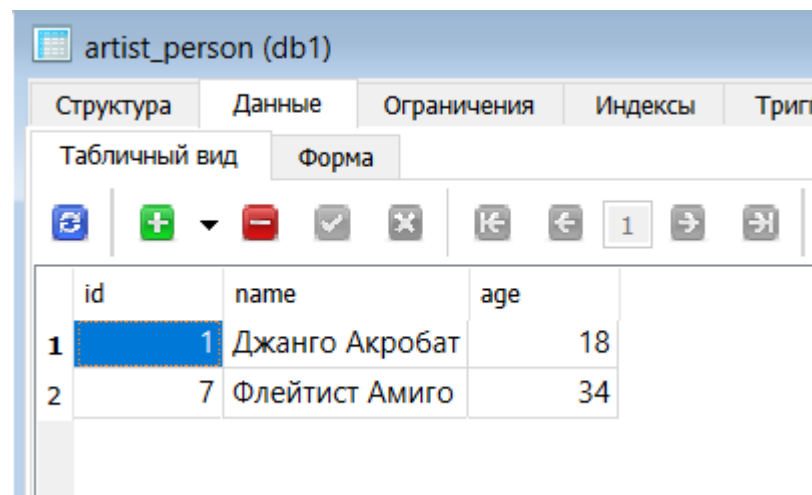
Сначала выбираем записи: `z = Person.objects.filter(id__gte=8)`

Затем удалим: `z.delete()`

```
>>> z = Person.objects.filter(id__gte=8)
>>> z.delete()
(3, {'artist.Person': 3})
>>>
```



	id	name	age
1	1	Джанго Акробат	18
2	7	Музыкант Джимми	23
3	8	Барабанщик Арлекино	19
4	9	Трубоч Илия	32
5	10	Саксофонист Билли	27



	id	name	age
1	1	Джанго Акробат	18
2	7	Флейтист Амиго	34



Более подробно о том, как управлять записями таблиц через модели, можно почитать в документации:

<http://djbook.ru/rel3.0/topics/db/queries.html>



- Конец части 2. Продолжение следует...

