

# Theory: Queries and filters

🕒 20 minutes    0 / 5 problems solved

Skip this topic

Start practicing

613 users solved this topic. Latest completion was about 4 hours ago.

There is not much sense in having a database if you don't know how to get data from it. Let's find out how it works in Django!

First, you should get familiar with the Model object manager. We will use it to get and filter the data for a particular model. Once you learn the syntax rules, you'll be able to easily make queries to your database. It will give you the flexibility to retrieve any objects you want.

Reading the data is the most common operation for a web application. The clients get data from the server more often than modify or delete it.

## §1. Model Object Manager

An instance of the Model class represents a single row in the table of your database. To start working with a set of rows you should call the **Model Object Manager** methods.

The Manager is a special class to get object(s) from the database and modify them. To access the Manager of your model, you should get the attribute *"objects"* of the `Model` class.

Right now we are working on a *tournament* application for a Quidditch league. The season is coming, but the website is not ready yet! The wizards from Hogwarts get used to working with the books and papers but know nothing about databases. Fortunately, you don't need magic to start querying and searching. We create models `Team` and `Player` and that's how we define them:

```
1 from django.db import models
2
3
4 class Team(models.Model):
5     name = models.CharField(max_length=64)
6
7
8 class Player(models.Model):
9     height= models.FloatField()
10
11     name = models.CharField(max_length=64)
12
13     team = models.ForeignKey(Team, on_delete=models.CASCADE)
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

It's not necessary to give an alias name to the Manager, you can use its methods simply like this: `Team.objects.filter(name="Ballycastle Bats")`. You can choose what you like more, but for clarity, we will access it directly in all the examples.

This small snippet helps you fill the tables with the data:

Current topic:

[Queries and filters](#) ...

Topic depends on:

- ✓ [Indexes](#) Stage 3 7★ ...
- ✓ [Dictionary methods](#) Stage 1 ...
- ✓ [Exception handling](#) 3★ ...
- ✗ [Django ORM](#) ...

Topic is required for:

- [Field lookups](#) ...
- [Object modification with ORM](#) ...

Table of contents:

- 1 [Queries and filters](#)
- [§1. Model Object Manager](#)
- [§2. Get an Object](#)
- [§3. Filtering Objects](#)
- [§4. Conclusion](#)
- [Feedback & Comments](#)

```
1 falmouth_falcons = Team.objects.create(name="Falmouth Falcons")
2 montrose_magpies = Team.objects.create(name="Montrose Magpies")
3
4
5 Player.objects.create(name="Karl Broadmoor", height=180, team=falmouth_falcons)
6
7 Player.objects.create(name="Kevin Broadmoor", height=183, team=falmouth_falcons)
8
9 Player.objects.create(name="Alasdair Maddock", height=175, team=montrose_magpies)
10
11 Player.objects.create(name="Lennox Campbell", height=197, team=montrose_magpies)
```

Remember that you should migrate your models before using it!

## §2. Get an Object

One step at a time, we will start from getting the team we want and then move on to getting a distinct player.

Unlike Python's `dict` *get* method, the Manager's *get* method may raise an `Exception`. You should keep in mind two rules:

- You can only pass the parameters with the names of the fields of your model or with valid field lookups;
- You should be sure that with this query you will get exactly one object.

We will carefully choose the parameters for our first query. Our `Team` model has two fields: *id* and *name*. The *id* field is generated automatically for every model, though we do not specify it explicitly.

We are sure that we have a team named "Falmouth Falcons". Let's try to get it with the Manager:

```
1 falcons = Team.objects.get(name="Falmouth Falcons")
```

Looks fine. But what happens if we get a nonexistent team?

```
1 tornados = Team.objects.get(name="Tutshill Tornados")
```

This call raises a `Team.DoesNotExist` exception. To prevent this situation and keep our program from crashing, you can wrap this call in *try-except* construction:

```
1 try:
2     tornados = Team.objects.get(name="Tutshill Tornados")
3 except Team.DoesNotExist:
4     ...
```

Let's try to get the "*Karl Broadmoor*" player account from the database:

```
1 karl_broadmoor = Player.objects.get(name="Karl Broadmoor")
```

Karl plays for Falmouth Falcons, so we get his account with no errors, but suppose you want to make a query that returns multiple objects:

```
1 falcons = Team.objects.get(name="Falmouth Falcons")
2 falcon_player = Player.objects.get(team=falcons)
```

You will not get a player, but a `Player.MultipleObjectsReturned` exception.

It seems that life is not that easy with the get queries. Sometimes we get an object, sometimes we get an error and we're never sure what happens next. Data may change and our valid call will start raising an `Exception`. You may turn to other Manager's methods and see what they can do for you.

## §3. Filtering Objects

Like the standard Python *filter* function, the Manager's *filter* method returns only the objects that match the query. You don't have to know initially how many objects it will return, so it's safer than the *get* method.

The only rule is similar to the first rule for the *get* method:

- You can only pass parameters with names of the fields of your model or with valid field lookups.

Now we'll try to make our queries without fear of `DoesNotExist` and `MultipleObjectReturned` situations. We modify our call to:

```
1 | tornados = Team.objects.filter(name="Tutshill Tornados")
```

Although we do not have Tornados in the database, no exception is raised. So what is the difference between these two methods? The answer is the return type. The *get* method always returns an instance of a particular model, while the *filter* method returns the `QuerySet`.

`QuerySet` is a wrapper for a set of objects in the database. `QuerySet` and `Manager` have much in common, so you can easily convert one into another. You can think of `QuerySet` as of another type of `Manager`.

To retrieve an object from the `QuerySet` you can iterate it over or get the item by the index as you get it from the Python's `list`.

```
1 | tornados = Team.objects.filter(name="Tutshill Tornados")
2 | if len(tornados) == 1:
3 |     tornados_team = tornados[0]
```

This call is safe, so you can change the model and condition and it will still work.

Also, we want to get a "Falmouth Falcons" player. Let's do it with the combination of *filter* and *first* methods:

```
1 | falcons = Team.objects.get(name="Falmouth Falcons")
2 | falcon_player = Player.objects.filter(team=falcons).first()
```

Success!

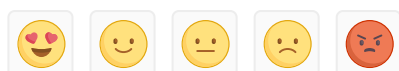
The last pitfall you should consider is that the *first* method does not raise any exceptions: if no objects are found, it returns `None`. So before accessing any properties of an object be sure that it's not `None`.

## §4. Conclusion

It's likely that getting data from a database is an operation you will frequently use. We started polishing our skills by getting and filtering data. We found out how to retrieve a single object and a `QuerySet` to work with them as we work with other Python classes. However, the main purpose of Django is to provide the tools to make web services, and you can easily apply your query skills for doing analytics and reports.

 Report a typo

44 users liked this theory. 3 didn't like it. What about you?



Start practicing

[Comments \(4\)](#)

[Hints \(0\)](#)

[Useful links \(0\)](#)

[Show discussion](#)