To Make Aggregation Queries, That Enable Us To Make Statistics, We Can use:

from django.db.models.aggregates import Count, Sum, Max, Min, Avg

from store.models import Product, OrderItem, Order

def say\_hello\_26(request):

    products\_count = Product.objects\

        .aggregate(

            Count('id'),

            count=Count('id'),

            min\_price=Min('unit\_price'),

            max\_price=Max('unit\_price'),

        )

    return render(request, 'hello.html', {

        'products\_count': products\_count,

        'name': 'Jafar-Loka',

    })

def say\_hello\_27(request):

    result\_1 = Order.objects.aggregate(count=Count('id'))

    result\_2 = OrderItem.objects.filter(product\_\_id = 1).aggregate(units\_sold=Sum('quantity'))

    result\_3 = Order.objects.filter(customer\_\_id=1).aggregate(count=Count('id'))

    result\_4 = Product.objects.filter(collection\_\_id = 3)\

    .aggregate(

        min\_price   =Min('unit\_price'),

        avg\_price   = Avg('unit\_price'),

        max\_price   =Max('unit\_price'))

    return render(request, 'hello.html', {

        'name': 'Jafar Loka',

        'result\_1': result\_1,

        'result\_2': result\_2,

        'result\_3': result\_3,

        'result\_4': result\_4

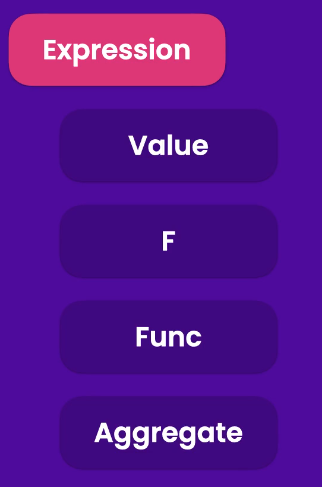
    })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If We Want To Add New Fields To The Model Object, We Can Use annotate-Method, That Return Also queryset-object:

**Note 1**: We Must Pass Expression Object To The New Field.

**Note 2**: We Can Pass Mixed Of Expressions To The New Field.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The Code Like This:

from django.db.models import Q, F, Value

def say\_hello\_28(request):

    queryset = Customer.objects.annotate(is\_new=Value(True), new\_id=F('id') + 1)

    return render(request, 'hello.html', { 'name': 'Jafar Loka', 'customers': queryset })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

We Can Use annotate-Method, For Creating New Mixed Value Of Fields:

**Note**: We Can Also, Use Django Database Functions To Make The Same Result.

from django.db.models import Q, F, Value, Func

from django.db.models.functions import Concat

def say\_hello\_28(request):

    queryset = Customer.objects.annotate(

        is\_new=Value(True),

        new\_id=F('id') + 1,

        full\_name= Func(F('first\_name'), Value(' '), F('last\_name'), function='CONCAT'),

        full\_name\_02 = Concat('first\_name', Value(' '), 'last\_name')

    )

    return render(request, 'hello.html', { 'name': 'Jafar Loka', 'customers': queryset })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If We Set Aggregation Function Like: Count, Inside The annotate-Method, Then Django Will Group The Results By The Fields That We Use.

def say\_hello\_29(request):

    queryset = Customer.objects.annotate(

        is\_new=Value(True),

        new\_id=F('id') + 1,

        full\_name= Func(F('first\_name'), Value(' '), F('last\_name'), function='CONCAT'),

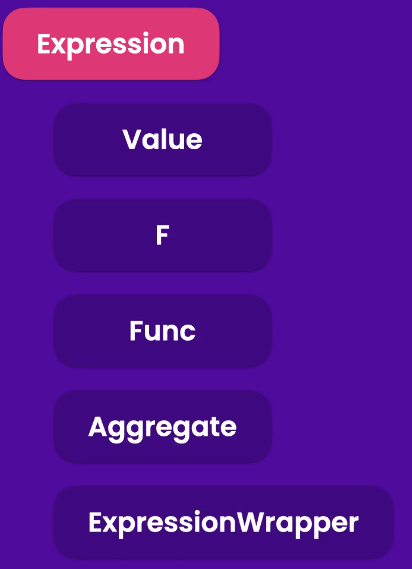
        full\_name\_02 = Concat('first\_name', Value(' '), 'last\_name'),

        order\_count = Count('order')

    )

    return render(request, 'hello.html', { 'name': 'Jafar Loka', 'customers': queryset })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Make Calculation Between Different Types Of Models:

from django.db.models import Q, F, Value, Func, ExpressionWrapper, DecimalField

def say\_hello\_30(request):

    queryset = Product.objects.annotate(

        discount\_price = ExpressionWrapper(

            F('unit\_price') \* 0.8,

            output\_field=DecimalField())

    )

    return render(request, 'hello.html', { 'name': 'Jafar Loka', 'products\_02': queryset })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Get The Data From ContentType Fields:

from django.contrib.contenttypes.models import ContentType

from tags.models import TaggedItem

def say\_hello\_31(request):

    content\_type = ContentType.objects.get\_for\_model(Product)

    # Here We Use select\_related To Avoid The Problems Of

    # Loading Many Tags

    tagItems = TaggedItem.objects\

        .select\_related('tag')\

        .filter(

            content\_type = content\_type,

            object\_id=1

    )

    return render(request, 'hello.html', {'name': 'Jafar Loka', 'tagsItem': tagItems})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

We Must Understand The Query Set Cache.

Cache Will Be Happened Only When The Entire QuerySet Evaluated, **Ex**:

*list(queryset)*

*queryset[0]*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Create Custom Manger For Specific Model:

**Note**: get\_tags\_for Is Custom Method For Getting The Data That We Want.

class TaggedItemManager(models.Manager):

    def get\_tags\_for(self, obj\_type, obj\_value):

        content\_type = ContentType.objects.get\_for\_model(obj\_type)

        # Here We Use select\_related To Avoid The Problems Of

        # Loading Many Tags

        return TaggedItem.objects\

            .select\_related('tag')\

            .filter(

                content\_type = content\_type,

                object\_id=obj\_value

        )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Then We Override The Objects-Attribute Of The Model:

class TaggedItem(models.Model):

    objects = TaggedItemManager()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Save The Collection Data, That Maybe Come With Request OR From Coder:

**Note 1**: Like Laravel; Collection-Class Has Also create-Method, But It Doesn’t Have Auto-Complete For Attributes

**Note 2:** Also; For Saving Featured Product, We Can use collection\_featured\_product\_id=1

def save\_collection\_example\_1(request):

    collection = Collection()

    collection.title = "Video Games"

    collection.featured\_product = Product(pk=1)

    collection.save()

    return render(request, 'hello.html', { 'name': 'Jafar Loka' })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Note**: This Will Raise An Error: Manager isn't accessible via Collection instances

# collection = collection.objects.select\_related('featured\_product')

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Update The Collection-Class With Reading Data, To Populate It:

    collection = Collection.objects\

        .select\_related('featured\_product')\

        .get(pk=16) # In This Way We Can Populate All Fields

    collection.title = "New Video Games-07"

    collection.save()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Update The Object Using Managers, Then We Need To Filter First, Else Every Thing Will Be Updated:

**Note**: Here Update-Method Will Return The Number Of Rows That Updated.

def update\_collection\_example\_2(request):

    Collection.objects.filter(pk=18).update(

        title="New Video Games-18",

        featured\_product=None

    )

    return render(request, 'hello.html', { 'name':'Jafar Loka' })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For Deleting Object/Objects From Table, We Can use *delete-Method* For Objects OR QuerySets.

If We Use QuerySets, Then We Should Use *Filter()-Method*.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Create Transactions, That Can Be Used For Committing OR Rollbacking:

from django.db import transaction

# This Can Be Used AS Decorator OR Using With-Keyword

@transaction.atomic()

def test\_transaction\_example\_1(request):

    order = Order()

    order.customer\_id = 1

    order.save()

    orderItem = OrderItem()

    orderItem.order = order

    orderItem.product\_id = 1

    orderItem.quantity = 1

    orderItem.unit\_price = 1

    orderItem.save()

    return render(request, 'hello.html',{ 'name': 'Jafar Loka' })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For More Control On The Transaction Operator, We Use Context Manger Using with-Keyword:

# This Can Be Used AS Decorator OR Using With-Keyword

def test\_transaction\_example\_1(request):

    with transaction.atomic():

        order = Order()

        order.customer\_id = 1

        order.save()

        orderItem = OrderItem()

        orderItem.order = order

        orderItem.product\_id = 1

        orderItem.quantity = 1

        orderItem.unit\_price = 1

        orderItem.save()

    return render(request, 'hello.html',{ 'name': 'Jafar Loka' })

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Execute RAW SQL, We Can Use raw-Method, That Return Raw QuerySet Object, Not Query Set Object.

def test\_raw\_sql(request):

    products = Product.objects.raw('SELECT \* FROM store\_order')

    return render(request, 'hello.html', {'name': 'Jafar Loka', 'raw\_result': products})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Use Stored Procedure We Can Use Cursor:

from django.db import connection

def test\_raw\_sql\_2(request):

    # cursor = connection.cursor()

    # products = cursor.execute('SELECT id, title FROM store\_product')

    # row = cursor.fetchone() # In This Way We Return The Result

    # cursor.close() # In Production We Use try-except-finally Block,

    # And Close The Cursor In Finally-Block, OR We Can Use

    # with-Keyword For Context Manager.

    # The Best Way To Use Cursor, By Using with-Keyword

    with connection.cursor() as cursor:

        cursor.callproc('Procedure Name Here', params=[1, 2, 'a'])

    return render(request, 'hello.html', {'name': 'Jafar Loka'})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

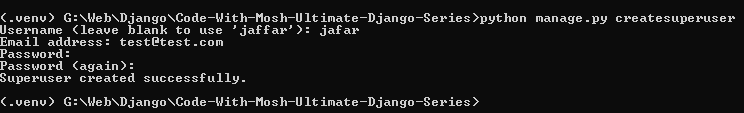
To Access The Admin Interface Of Our Project, Go To URL: <http://project.url.com:port_if_exists/admin>

* Ex: <http://localhost:8000/admin>

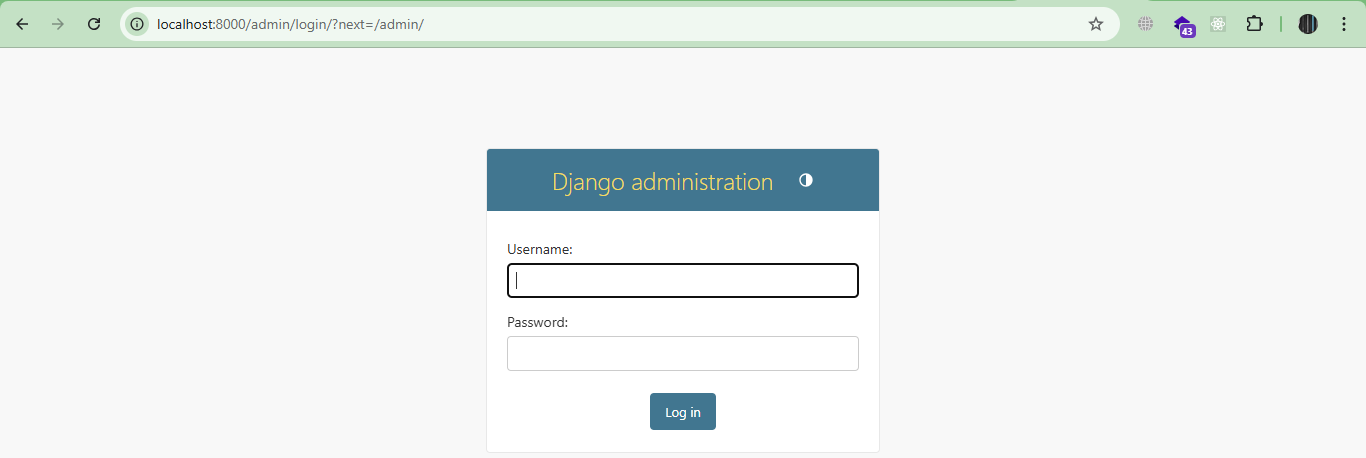
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

To Create User For Admin Interface, Run Command: *python manage.py createsuperuser*

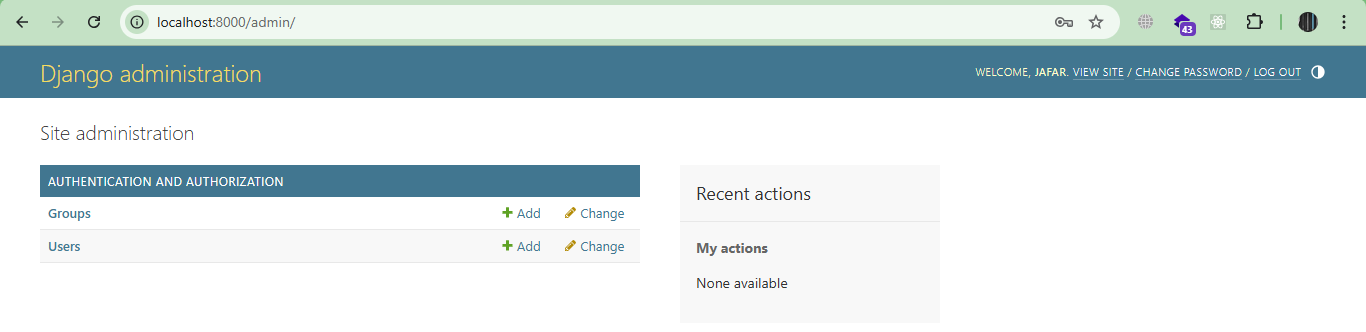




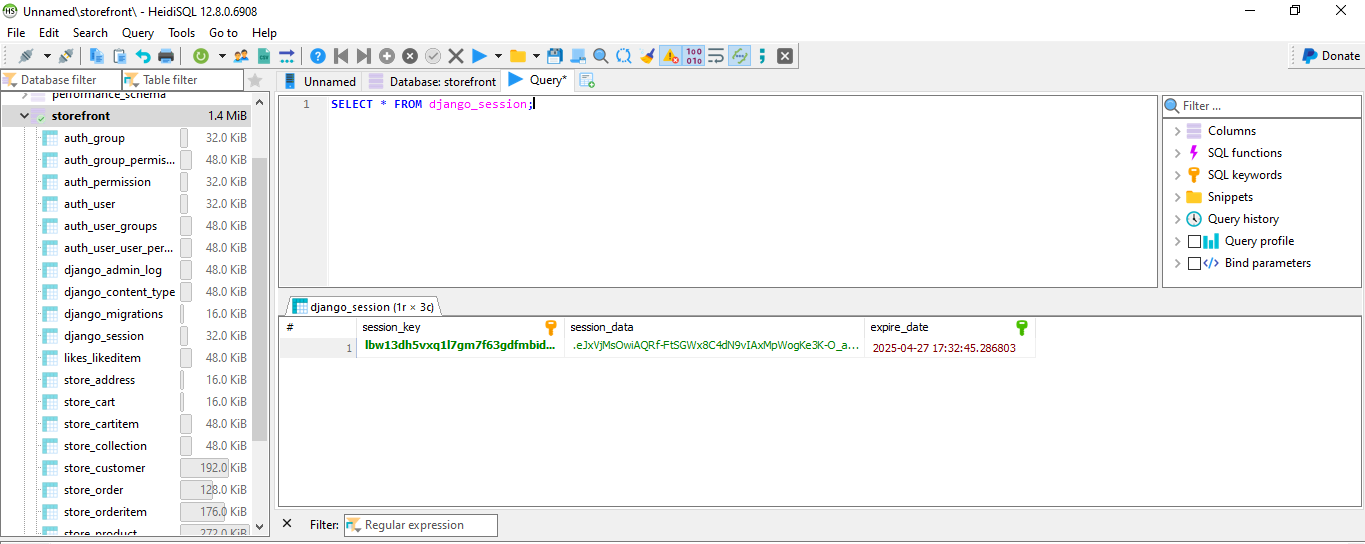
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



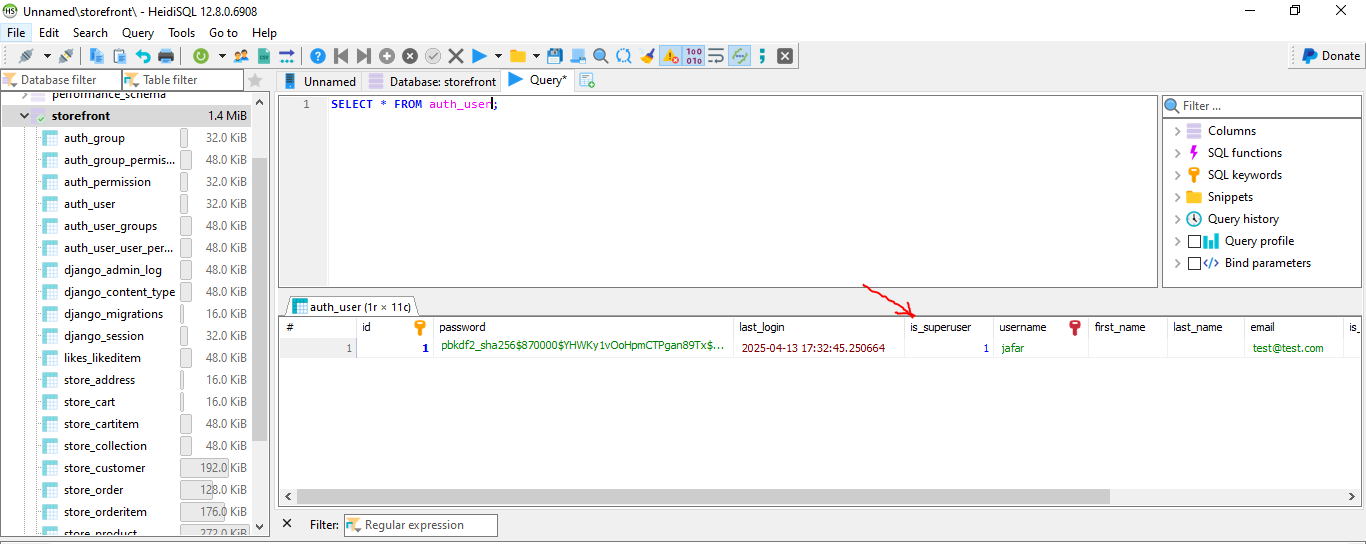
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*