**Dr BR Ambedkar National institute of Technology**

**Jalandhar**

**DBMS LAB PROJECT REPORT**



Topic: Developing a database management system for an ecommerce webapp based on Django Rest framework in Full Stack

Submitted By: Submitted to:

Anubhav Gupta : 19124014 Dr Amritpal Singh  
Ayush Gupta : 19124020

**Introduction:**

Django is a python-based web framework that is employed here to build the back end of website. The Django framework is compatible with multiple **RDS** like Postgres, MySql, and Sqlite3 databases.

In this Project We employ MySQL database hosted on   
<https://remotemysql.com/> with the following credentials:

Username: qYfI2UxniT

Password: UVPaaAO31J

Database Name: qYfI2UxniT

Now the database is connected with the Django using a python configuration file called settings.py

Under the Following code.

# Database

# https://docs.djangoproject.com/en/2.0/ref/settings/#databases

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'qYfI2UxniT',

        'USER': 'qYfI2UxniT',

        'PASSWORD': 'UVPaaAO31J',

        'HOST': 'remotemysql.com',

        'PORT': '3306',

    }

}

The Schema for Tables are defined in a Models.py file.

The attributes of each tables are defined in a python class and are related to SQL schema as follows:

class Blog (models.Model):

    name         = models.CharField(max\_length=50)

    content      = HTMLField()

    author       = models.CharField(max\_length=20)

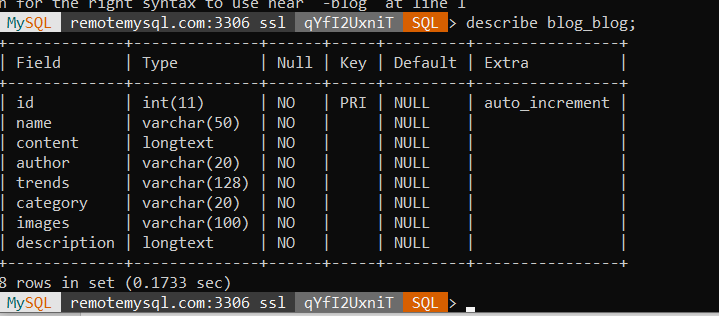
    trends       = models.CharField(choices=Trend\_CHOICES, max\_length=128)

    category     = models.CharField(max\_length=20)

    images       = models.ImageField(upload\_to= 'images/blog',default='')

    description  = models.TextField(default='')

Now Django is configured in a way that there are multiple apps, and each app is configured with its own logic and functionality. Now the app name for above file is blog so corresponding table that’s made in MySql corresponding to this code is as follows:



Here a Primary Key called id is always set by default in Django with auto increment constraint.

Now we can observe how python Class attributes are translated to MySql by Django here:

1. The CharField() has a max\_length argument and is translated to varchar(max\_length) in corresponding MySql.

2. Whereas HtmlField and TextField have corresponding longtext fields to store large strings of data.

3. We define a imageField in Python that store the address of image file in the directory tree as varchar(100) in backend because we know that we cannot store files in a database.

Once we define the table in a class it is migrated to database using the following commands:

1. python manage.py makemigrations

2. python manage.py migrate --run-syncdb

These migrate the database and create appropriate changes to it

**Features and Entities Used :**

The webapp is made with following features and these have corresponding Schemas defined along with the python implementation: -

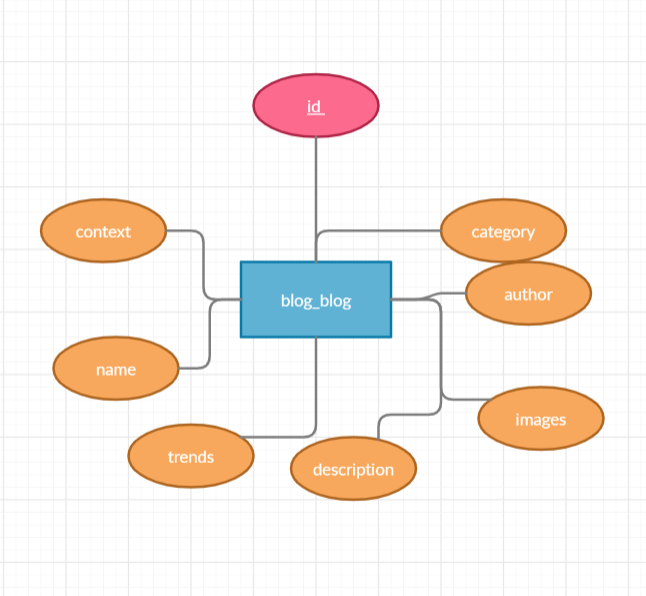
1. **Blogs**

This is Blogging module based in tinymce text-editor plugin and it stored Html as a string in database

1. **Newsletter**
   1. **Email**
   2. **News**
2. **Shop**
3. **Orders**
4. **Categories**
5. **Order\_Items (items in cart)**
6. **Products listed in Shop**
7. **Contact Form**
8. **E-books**

**The Schemas:**

The Schemas of each entities as defined in python and MySQL are as follows:

**1.blog\_blog:**

Trend\_CHOICES = (

   ('Y', 'Yes'),

   ('N', 'No')

)

class Blog (models.Model):

    name         = models.CharField(max\_length=50)

    content      = HTMLField()

    author       = models.CharField(max\_length=20)

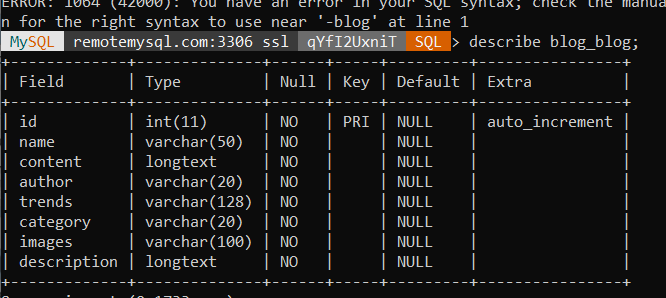
    trends       = models.CharField(choices=Trend\_CHOICES, max\_length=128)

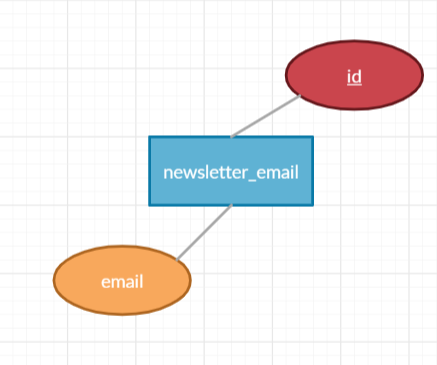
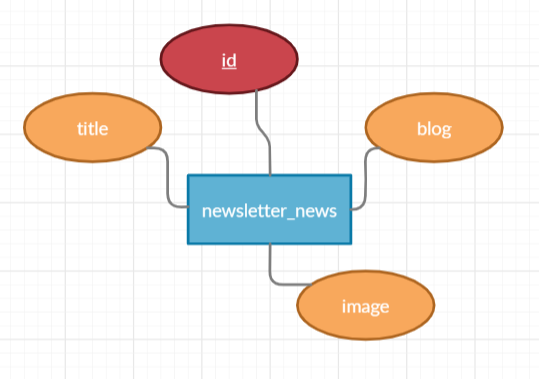
    category     = models.CharField(max\_length=20)

    images       = models.ImageField(upload\_to= 'images/blog',default='')

    description  = models.TextField(default='')

this corresponds to following table definition in mysql



******2.Newsletter:**

This module contains 2 Tables, and this provide a subscribe to newsletter functionality to user,

The Subscribe button on website is linked with two Queries,

Once the user Enters the Email in the html box, a filtered query is made to database to check if the user already exists in table,

In Django it looks like

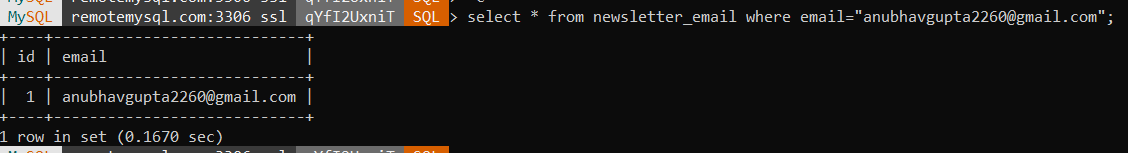
 if Email.objects.filter(email=email).exists():

                 messages.info(request, 'You are already in our Family')

it is equivalent to Mysql Syntax as:

**Select \* from newsletter\_email where email=email;**

(email being a string variable)



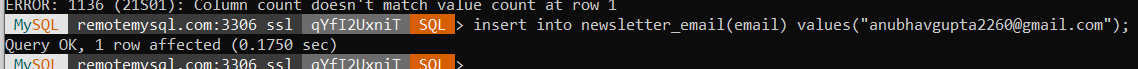
If the user Does not exist the next step is to add the user to database and it is done as:

 data = Email(email=email)

                data.save()

this is equivalent to MySql syntax as:

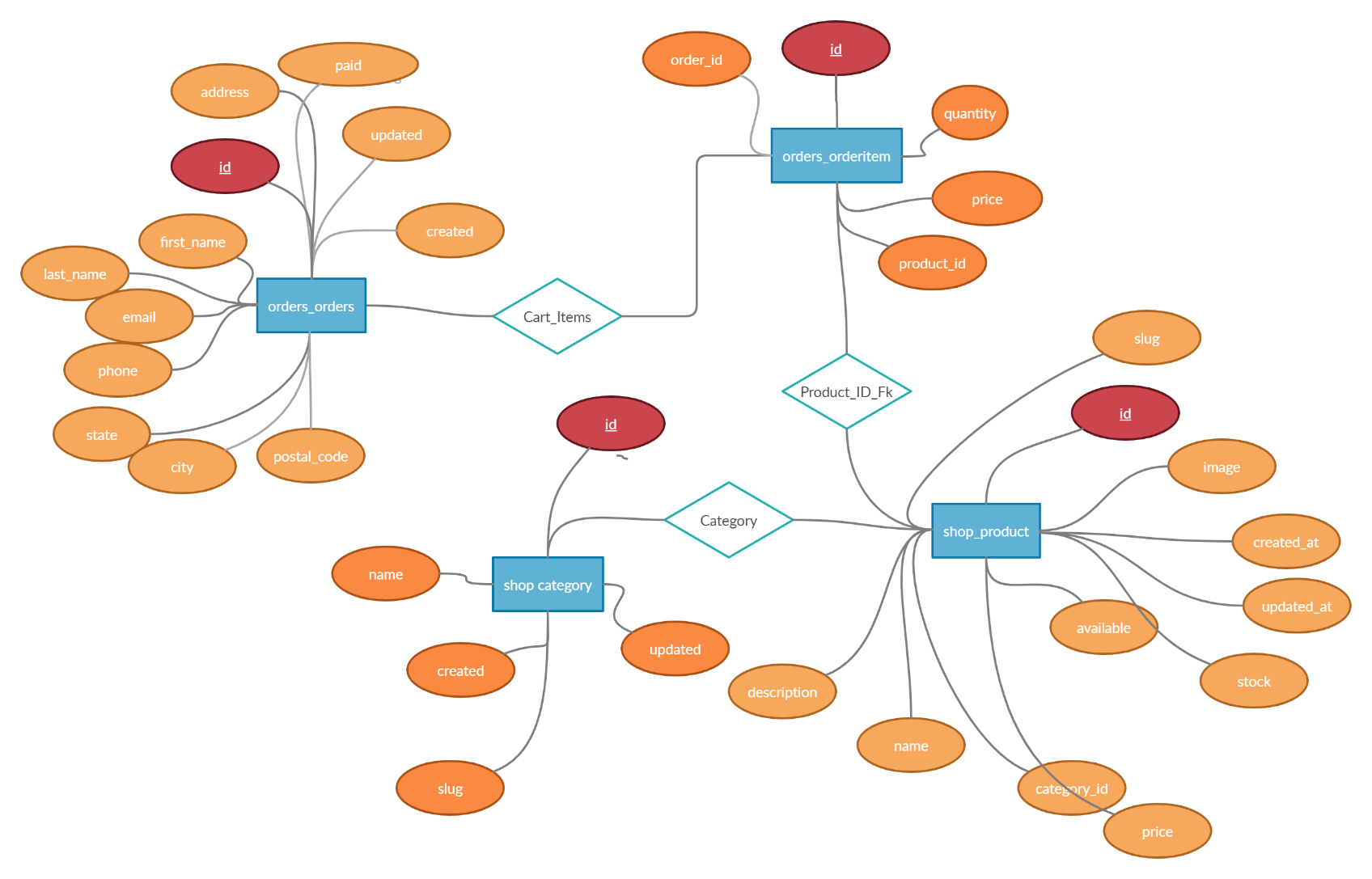
**Insert into newsletter\_email(email) values(email);**

****

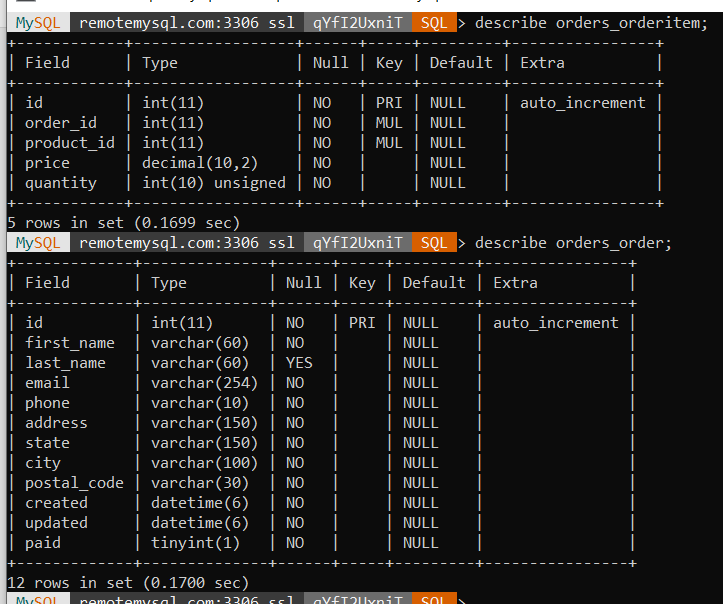
email here is a string variable

**3. Shop:**

This is the core module of our webapp and it contains 4 tables interlinked as shown in the following ERD:



Here the Entity **orders\_orders**  is related to **orders\_orderitem** using a foreignkey constraint and that is



Here we can clearly see that the *mul* key of table orders\_orderitem corresponds to orders\_order’s id that is primary key   
making a many to one field :  
Logically speaking a single order can have multiple order Items stored in orders\_orderitem table.

This is achieved in python by following code

class Order(models.Model):

    first\_name = models.CharField(max\_length=60)

    last\_name = models.CharField(max\_length=60,null=True,blank=True)

    email = models.EmailField()

    phone=models.CharField(max\_length=10)

    address = models.CharField(max\_length=150)

    state=models.CharField(max\_length=150)

    city = models.CharField(max\_length=100)

    postal\_code = models.CharField(max\_length=30)

    created = models.DateTimeField(default=timezone.now)

    updated = models.DateTimeField(default=timezone.now)

    paid = models.BooleanField(default=False)

class OrderItem(models.Model):

    order = models.ForeignKey(Order, related\_name='items',

 on\_delete=models.CASCADE)

    product = models.ForeignKey(Product,

related\_name='order\_items',

 on\_delete =models.CASCADE)

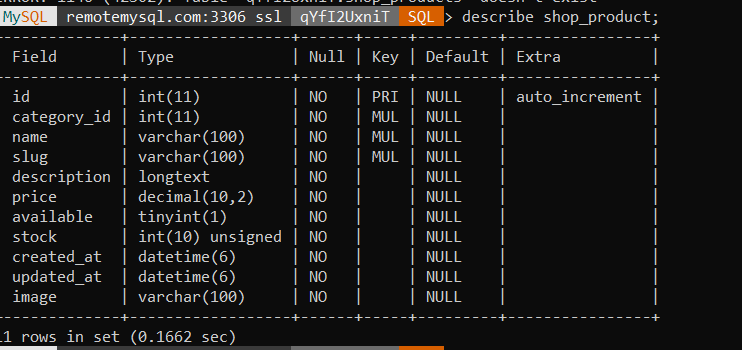
    price = models.DecimalField(max\_digits=10, decimal\_places=2)

    quantity = models.PositiveIntegerField(default=1)

here in ForiegnKey field Models.cascade ensure that on deletion of the field entry the corresponding foreign key entries are also deleted from the database.

**3.Shop\_products:**

The Shop\_products is related to Shop\_category using same concept of one to many field i.e a foreign key   
here a single category can hold multiple products, Also the Product id is a Foreign key in order items table hence using this relation   
we can trace back a product that has been ordered and hence we can update the Stock in our inventory

****

The same in Django is Described as :

class Product(models.Model):

    category = models.ForeignKey(Category, related\_name='products',

on\_delete=models.CASCADE)

    name = models.CharField(max\_length=100, db\_index=True)

    slug = models.SlugField(max\_length=100, db\_index=True)

    description = models.TextField(blank=True)

    price = models.DecimalField(max\_digits=10, decimal\_places=2)

    available = models.BooleanField(default=True)

    stock = models.PositiveIntegerField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

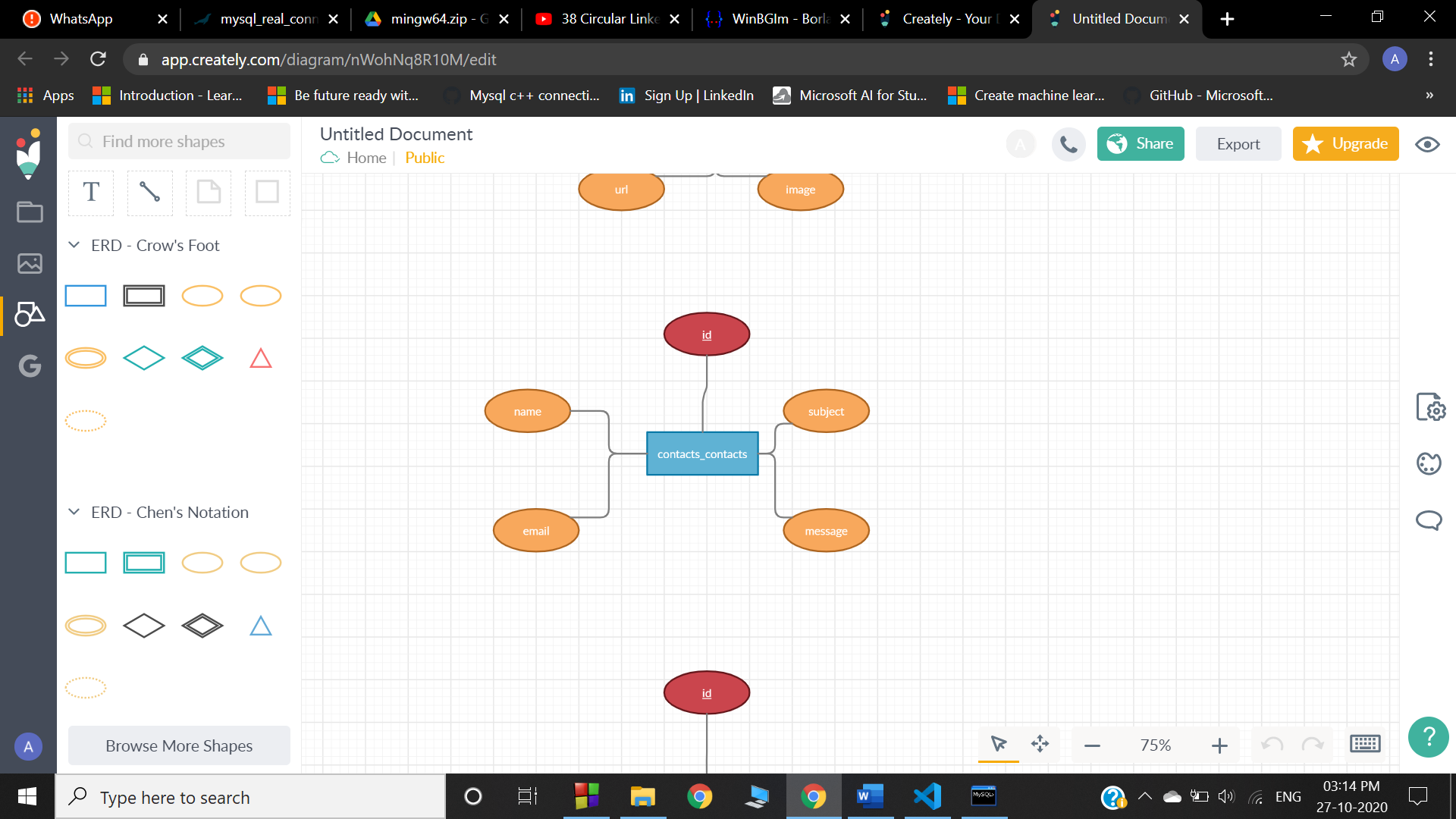
    image = models.ImageField(upload\_to='products/%Y/%m/%d', blank=True)

**4.Shop\_category:**

This is a table that contains a foreign key to products table hence the *mul* relation can help categorisation.

The SlugField in Django is used to have a callback link which is used to call category when called , I can redirect to the category of product

**4. Contact Form**



This module contains only one table with id as key attribute and name, email, subject, messages as other attributes.

This is achieved in python by following code

class Contact(models.Model):

    name     = models.CharField(max\_length=50)

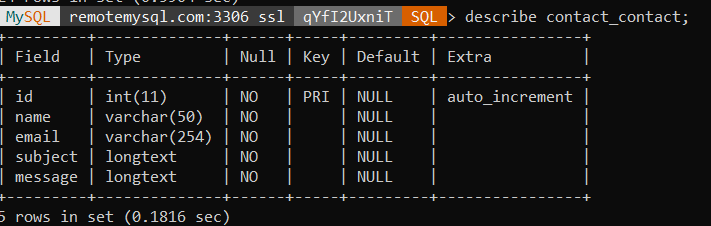
    email    = models.EmailField()

    subject  = models.TextField()

    message  = models.TextField()

Here name can be of maximum 50 character length.

The Real table created in Mysql in Backend is shown below



**The filtered Queries:**

The syntax of making a Query in Django is different to that of MySQL but Django being a cross platform database Framework ,

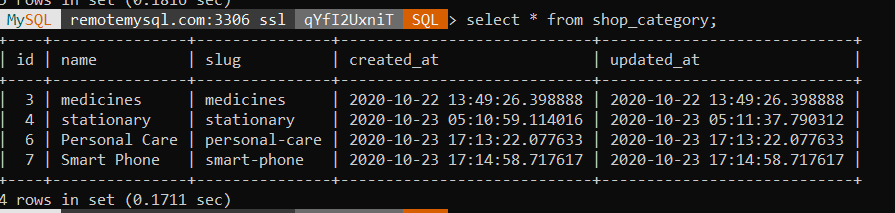
Can translate the Python query to a SQL query.

The Queries made in Python for the Webapp resemble Mysql syntax in the following way:

**Shop App:**

categories = Category.objects.all()

this is equivalent in Mysql as :



products = Product.objects.filter(available=True)

This is Equivalent in MySql as :

**Select \* from shop\_product where available = 1;**

products = Product.objects.filter(category=category)

This is Equivalent in MySql as :

**Select \* from shop\_product where category = “ ”;**

for item in cart:

                    OrderItem.objects.create(

                        order=order,

                        product=item['product'],

                        price=item['price'],

                        quantity=item['quantity']

                    )

This Query is used to insert data into Table

**Field Lookups:**

This was an extremely interesting part in Django that is used to make the search module in webapp wherein user can search for a specific product using a Query that is similar to **Like** of MySql in reality

 if request.method == 'GET':

        query= request.GET.get('q')

        if query is not None:

            lookups= Q(name\_\_icontains=query)

            results= Product.objects.filter(lookups).distinct()

            context = {

                'category': category,

                'categories': categories,

                'products': results

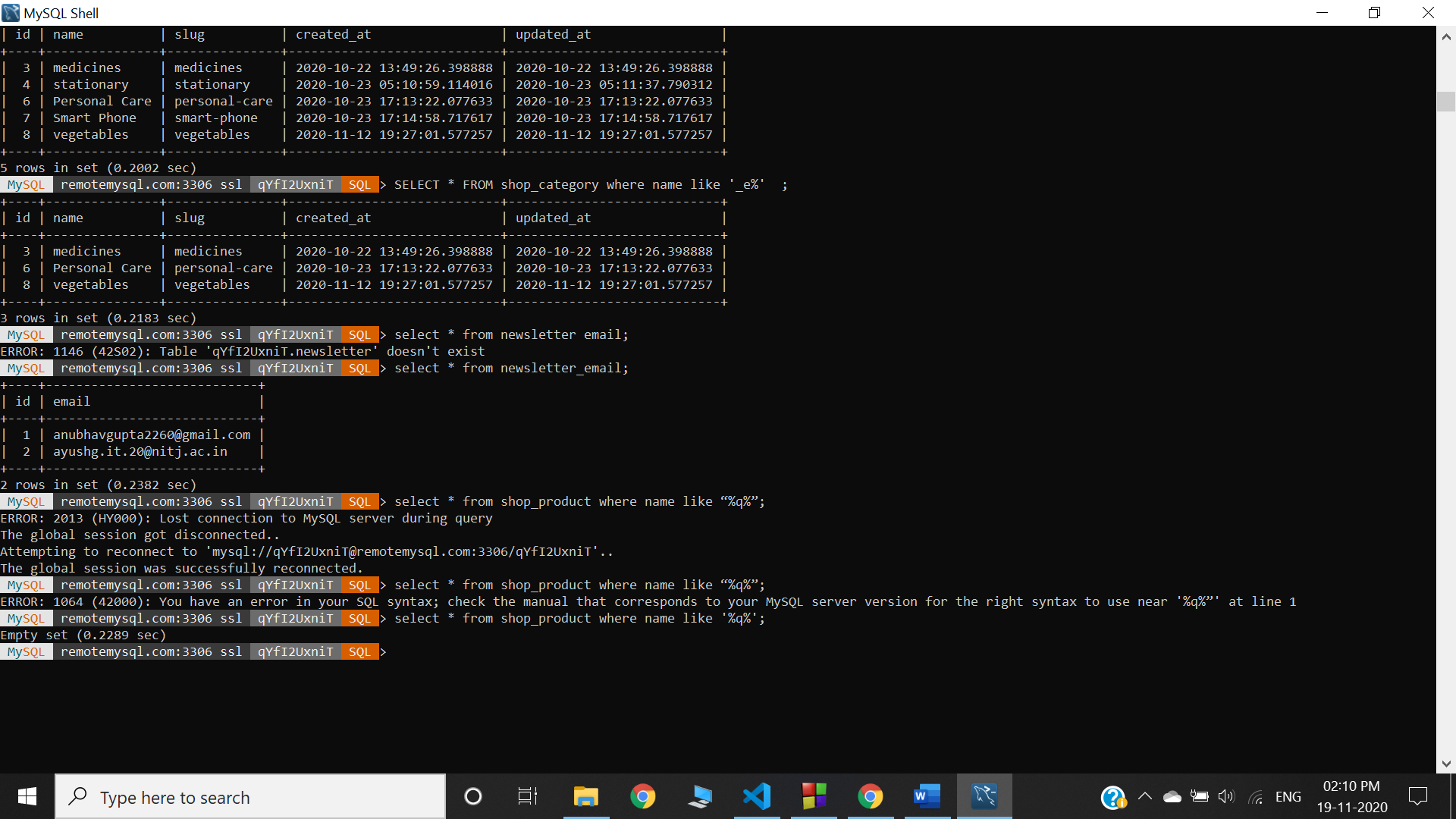
            }

The above can be roughly translated to MySQL syntax as:

First a Html form submits a string from the search box and that is passed to create a lookup using **icontains** on the Name attribute:

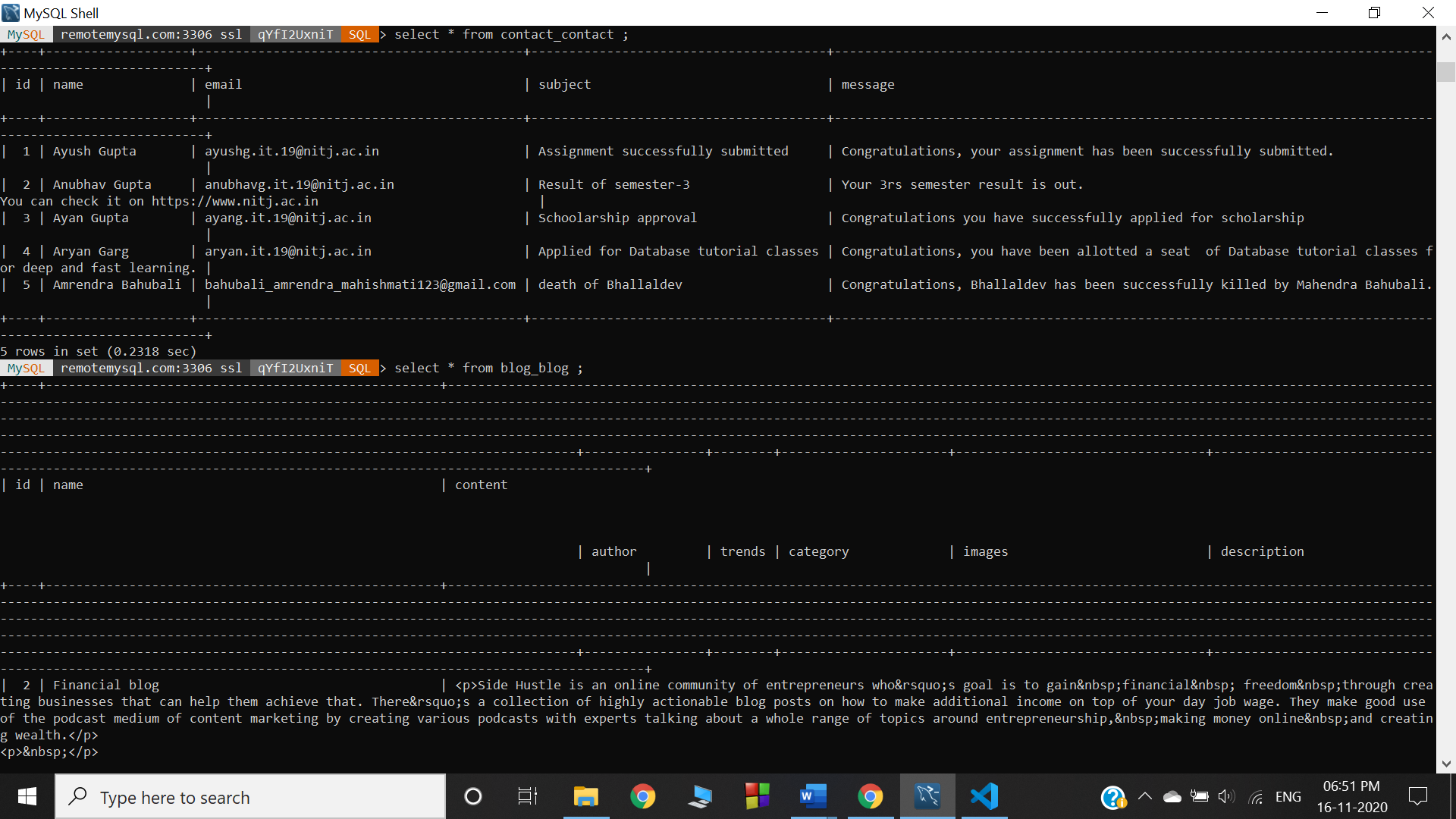
This is similar to like as

**select \* from shop\_product where name like “%q%”;**



q here can be a string that is taken from front end during form submission,

an improvement to this can be using Ajax in front end to send requests to server without reloading the page and to make it more user friendly and responsive.

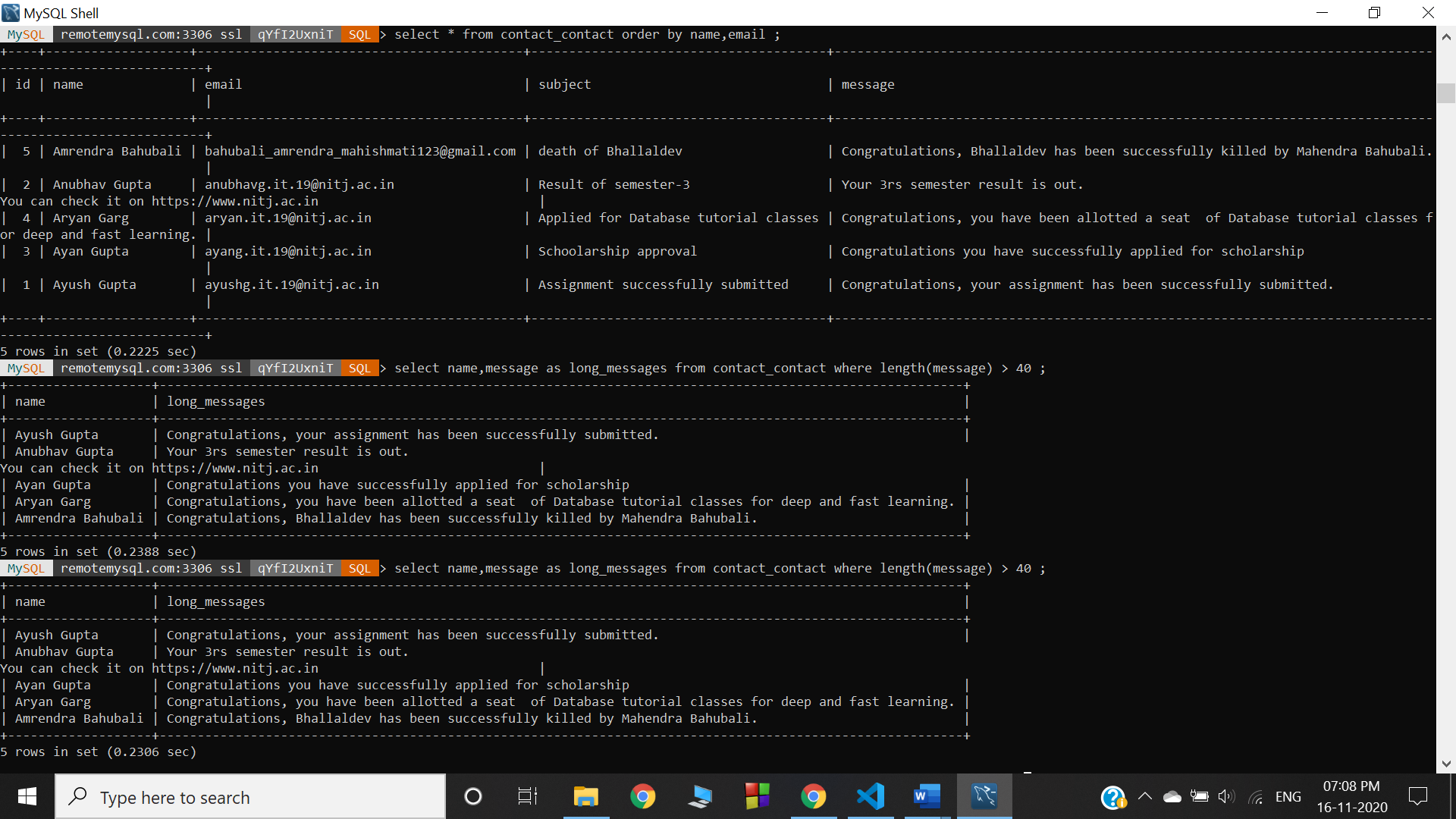
**select \* from contact\_contact ;**

**In Blog App:**

The Blog app has following Queries:

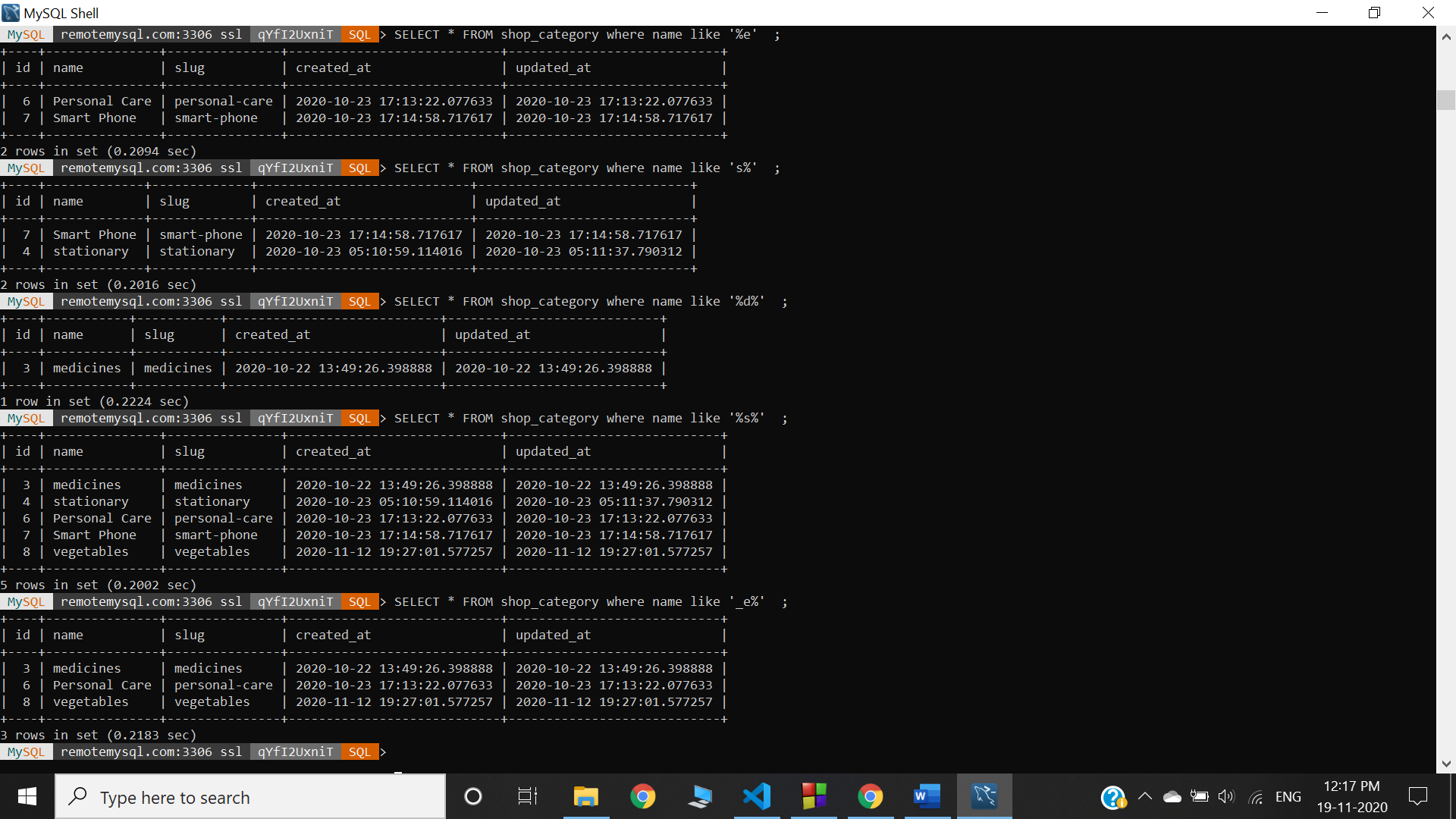
This query will show all rows and columns from table (contact\_contact).

**select \* from contact\_contact order by name,email ;**



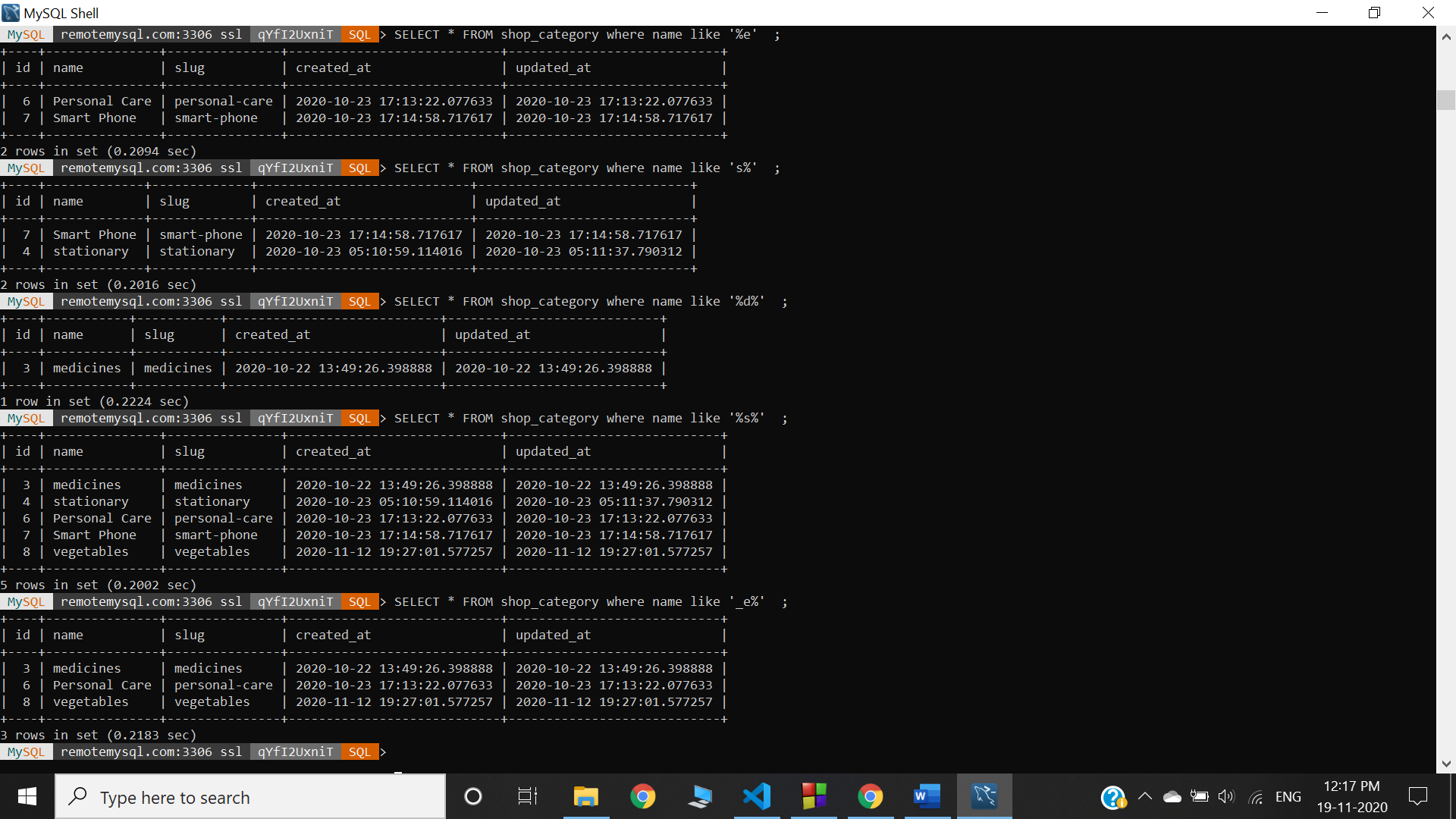
Order by will display all rows in ascending order by default in order of name. But if two names are same then output will be in order of email.

**SELECT \* FROM shop\_category where name like '%e';**



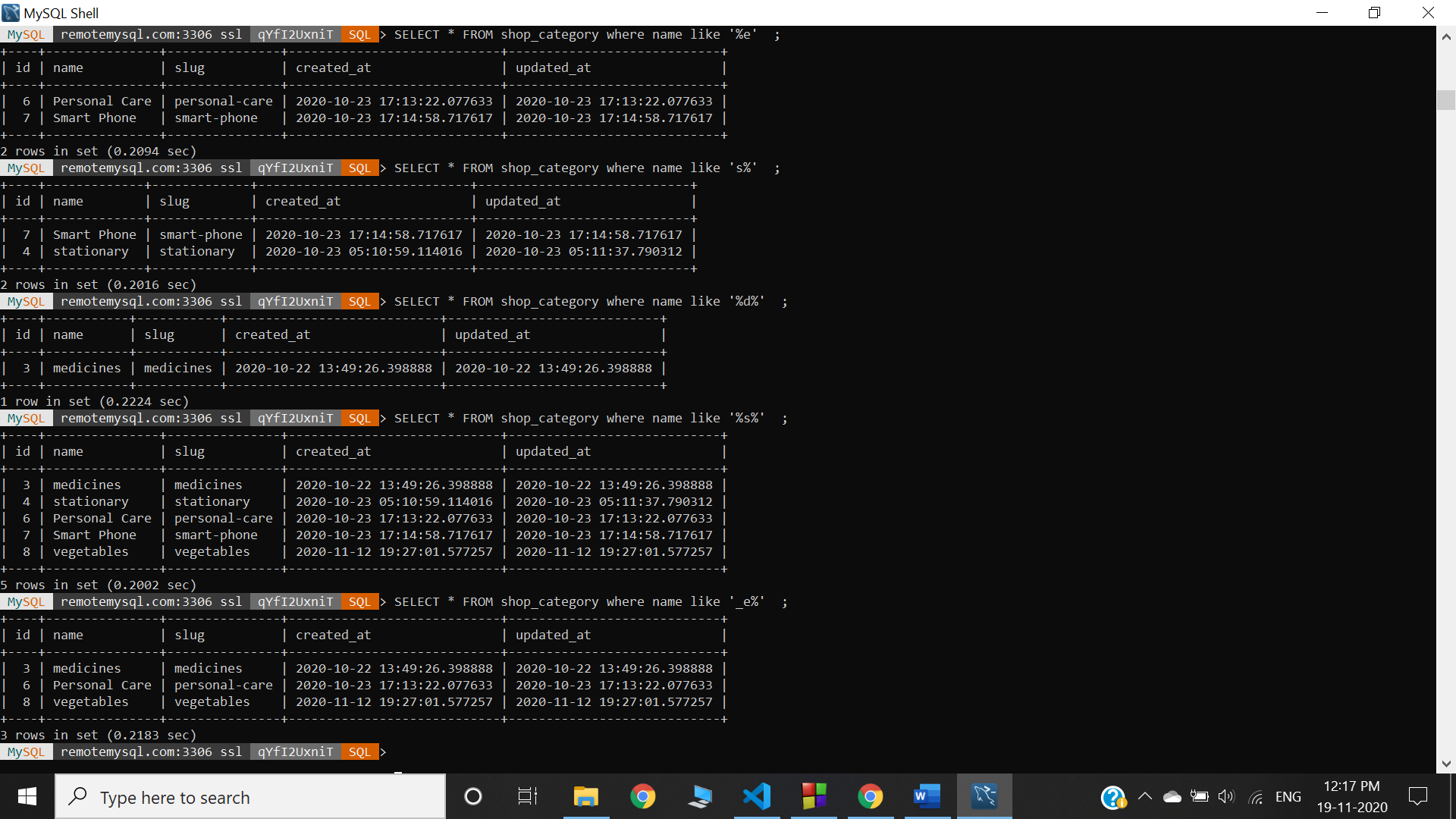
The **LIKE operator** is used in a WHERE clause to search for a specified pattern in a column. Here, rows with names ending with e are displayed.

**SELECT \* FROM shop\_category where name like 's%';**



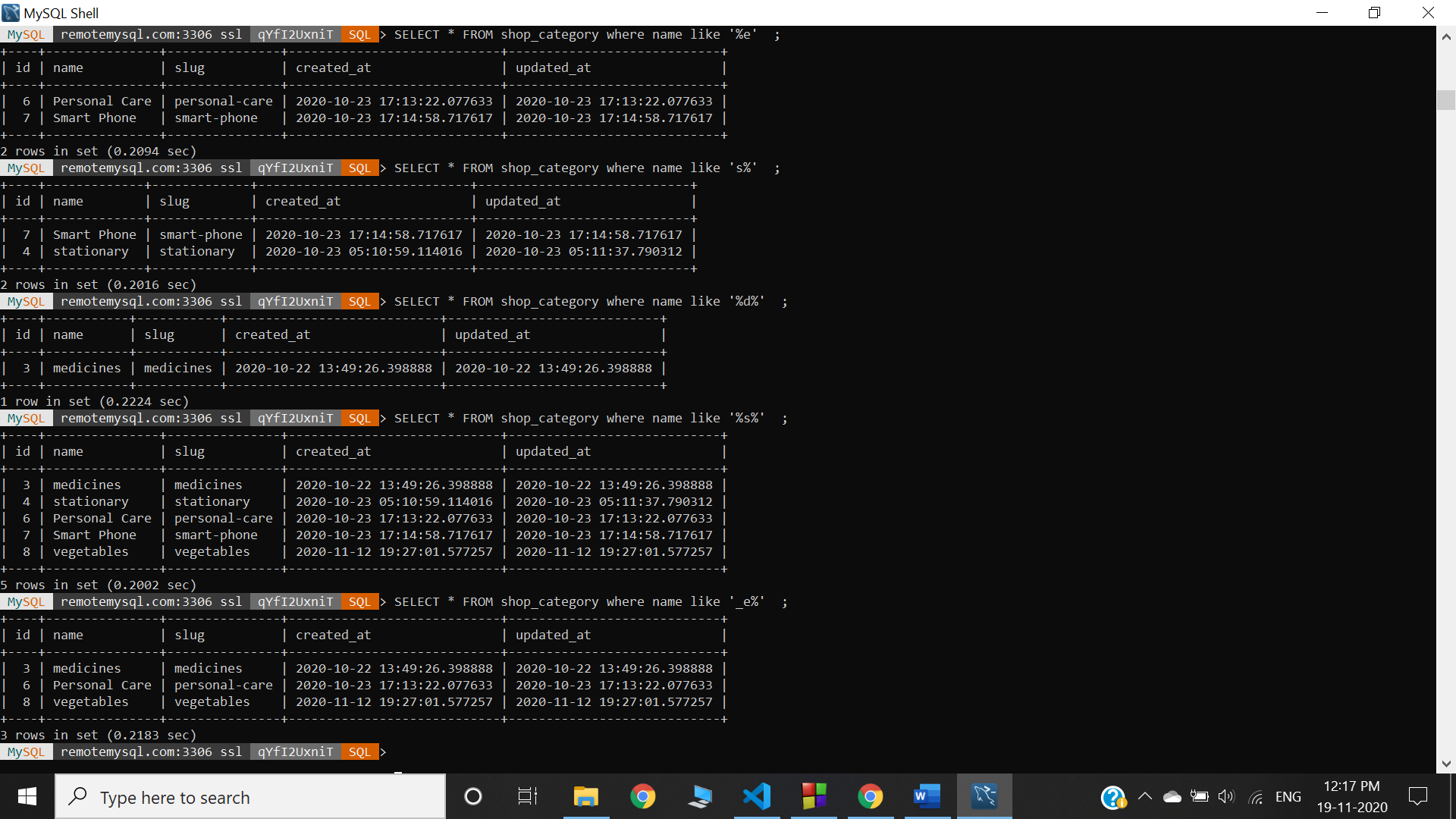
In above query, rows with names starting with s are displayed.

**SELECT \* FROM shop\_category where name like '%s%' ;**



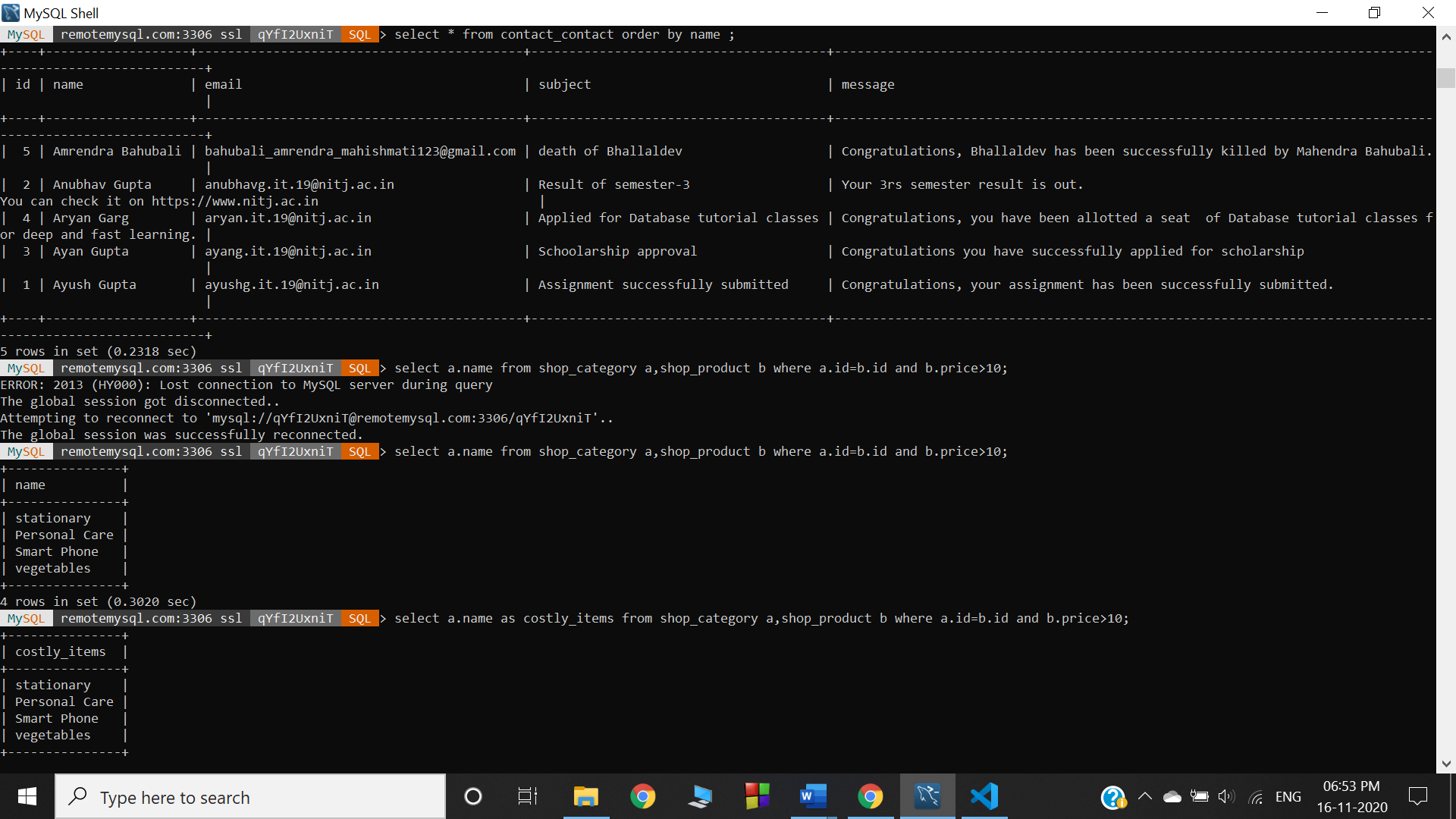
In above query, rows with names having letter ‘s’ anywhere are displayed.

**SELECT \* FROM shop\_category where name like '\_e%' ;**



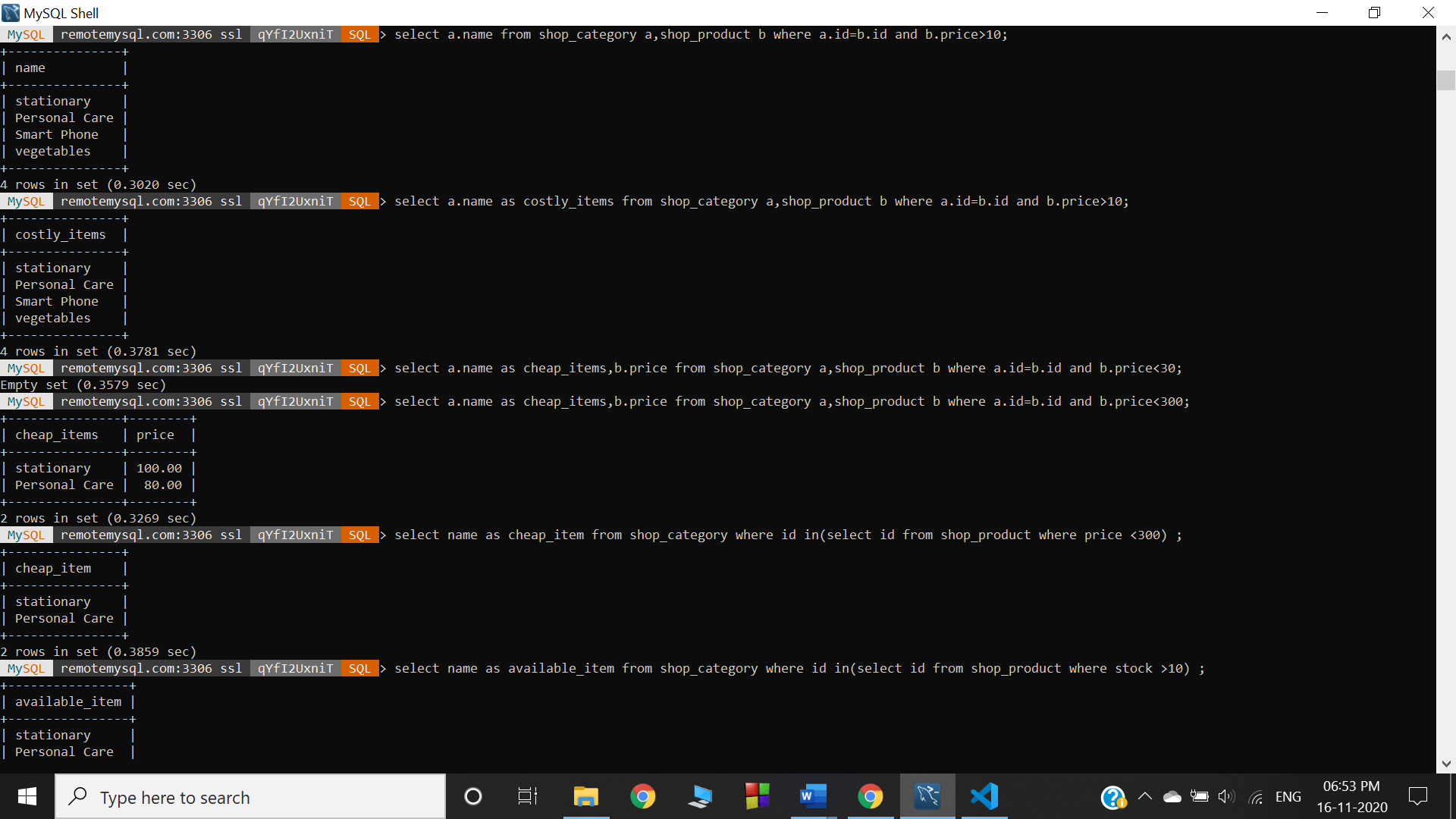
In above query, rows with names having ‘e’ at second place are displayed.

**select a.name as costly\_items from shop\_category a,shop\_product b where a.id=b.id and b.price>10;**



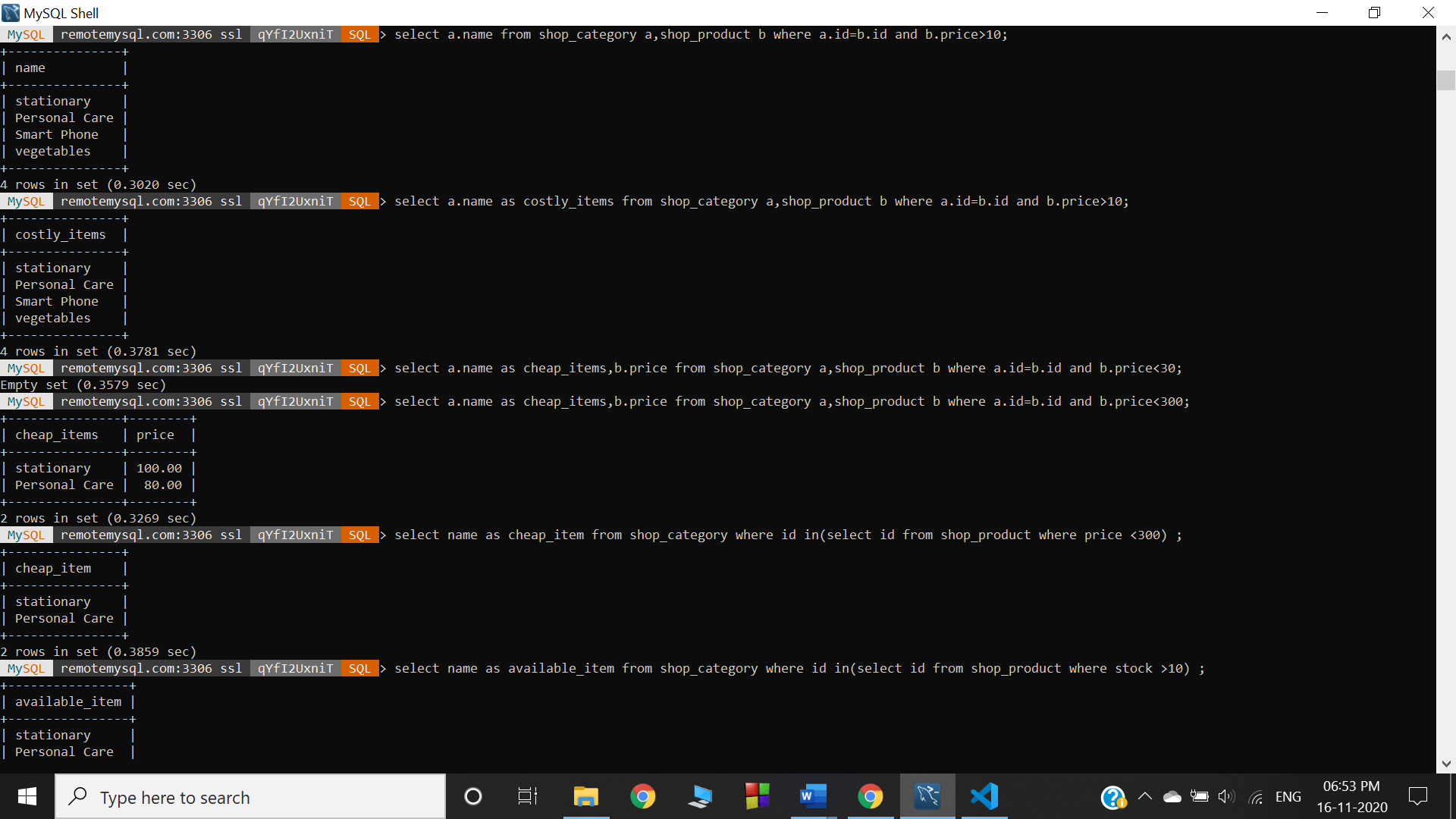
Here we have joined two tables shop\_category and shop\_product and named them as a and b to get the required output.

**select a.name as cheap\_items,b.price from shop\_category a,shop\_product b where a.id=b.id and b.price<300;**

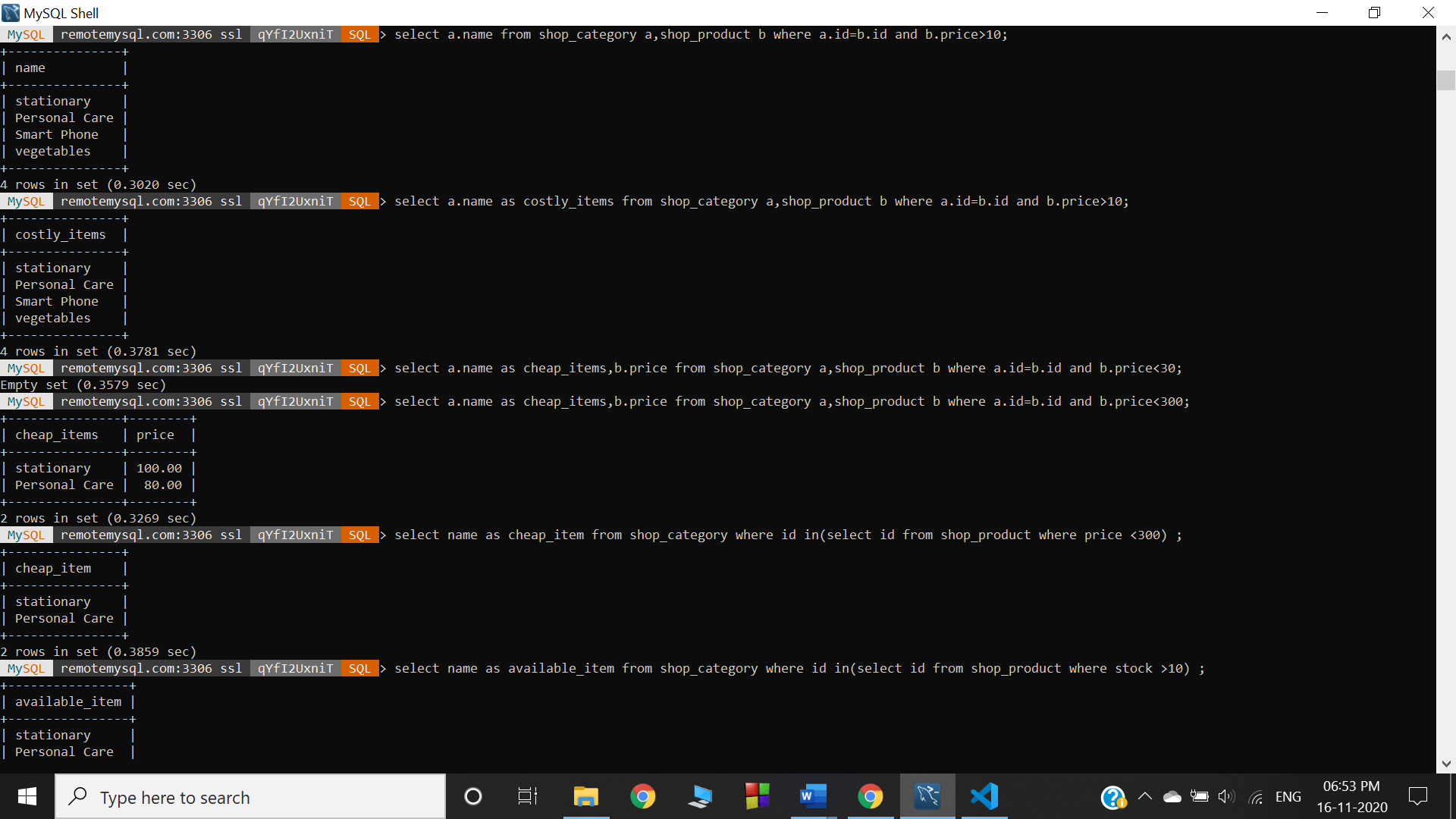


We can also rename columns using keyword ‘as’ as we done in above example. In above query we have displayed name as cheap items and their prices for items having price less than Rs300.

**select name as cheap\_item from shop\_category where id in(select id from shop\_product where price <300) ;**

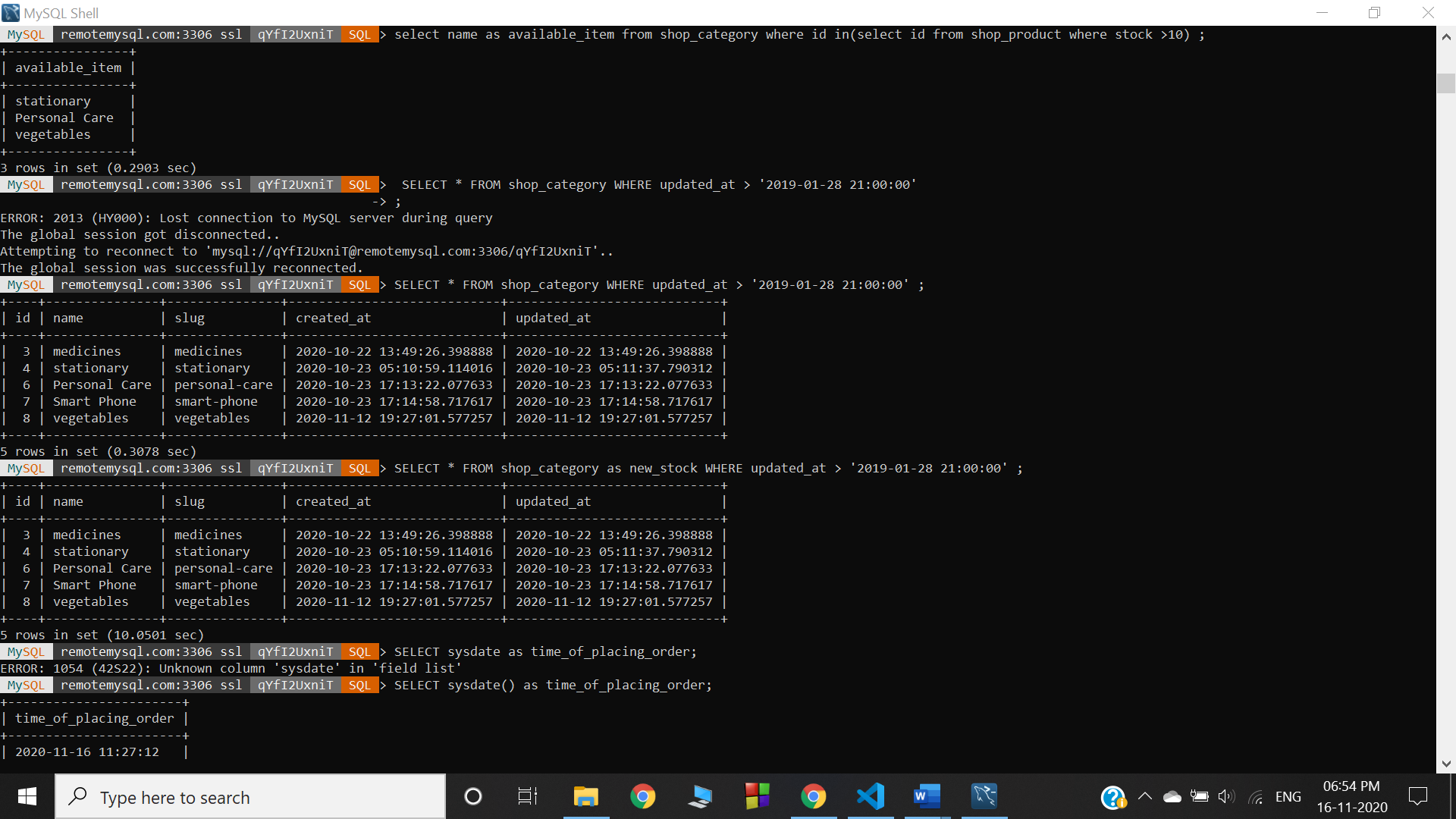


**select name as available\_item from shop\_category where id in(select id from shop\_product where stock >10) ;**



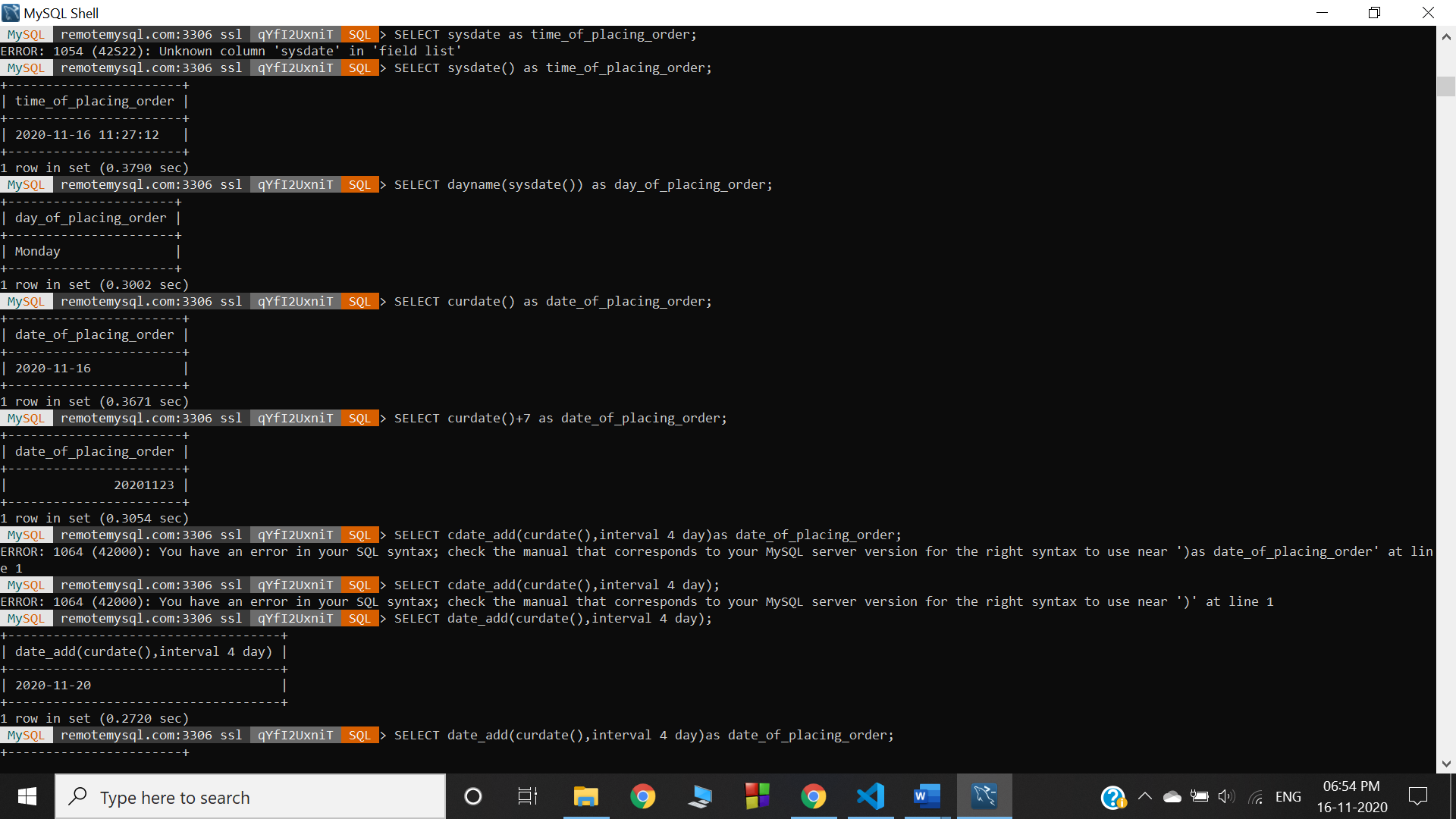
In above **two** examples we have nested the queries. The output of query written in bracket will be used to perform the outer query. ‘in’ is the keyword used to receive the output of nested query.

**SELECT \* FROM shop\_category WHERE updated\_at > '2019-01-28 21:00:00' ;**



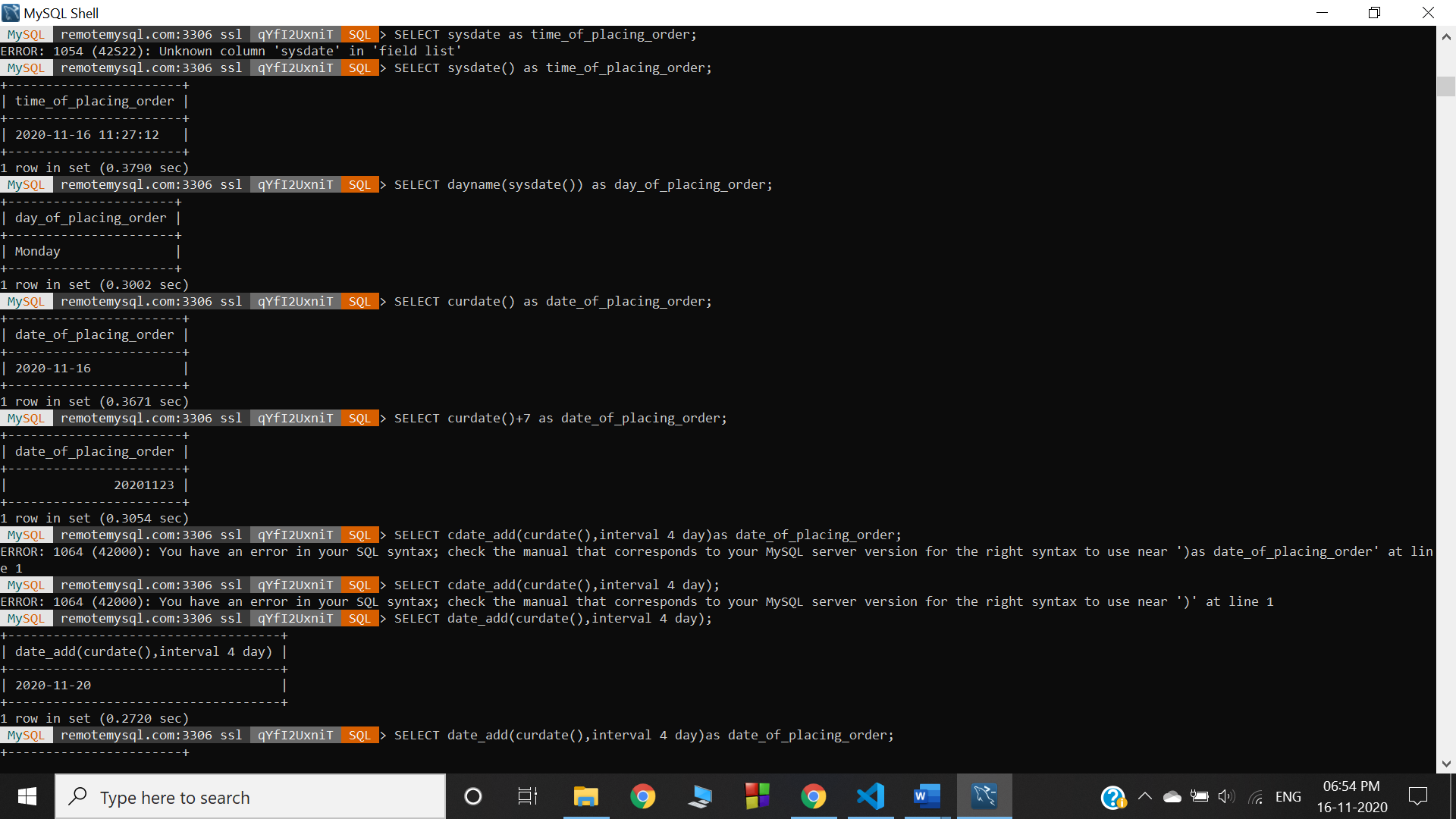
Above query is performed to get output where time in the column ‘updated\_at’ is greater than the time mentioned in ‘’ in the query.

**SELECT sysdate() as time\_of\_placing\_order;**



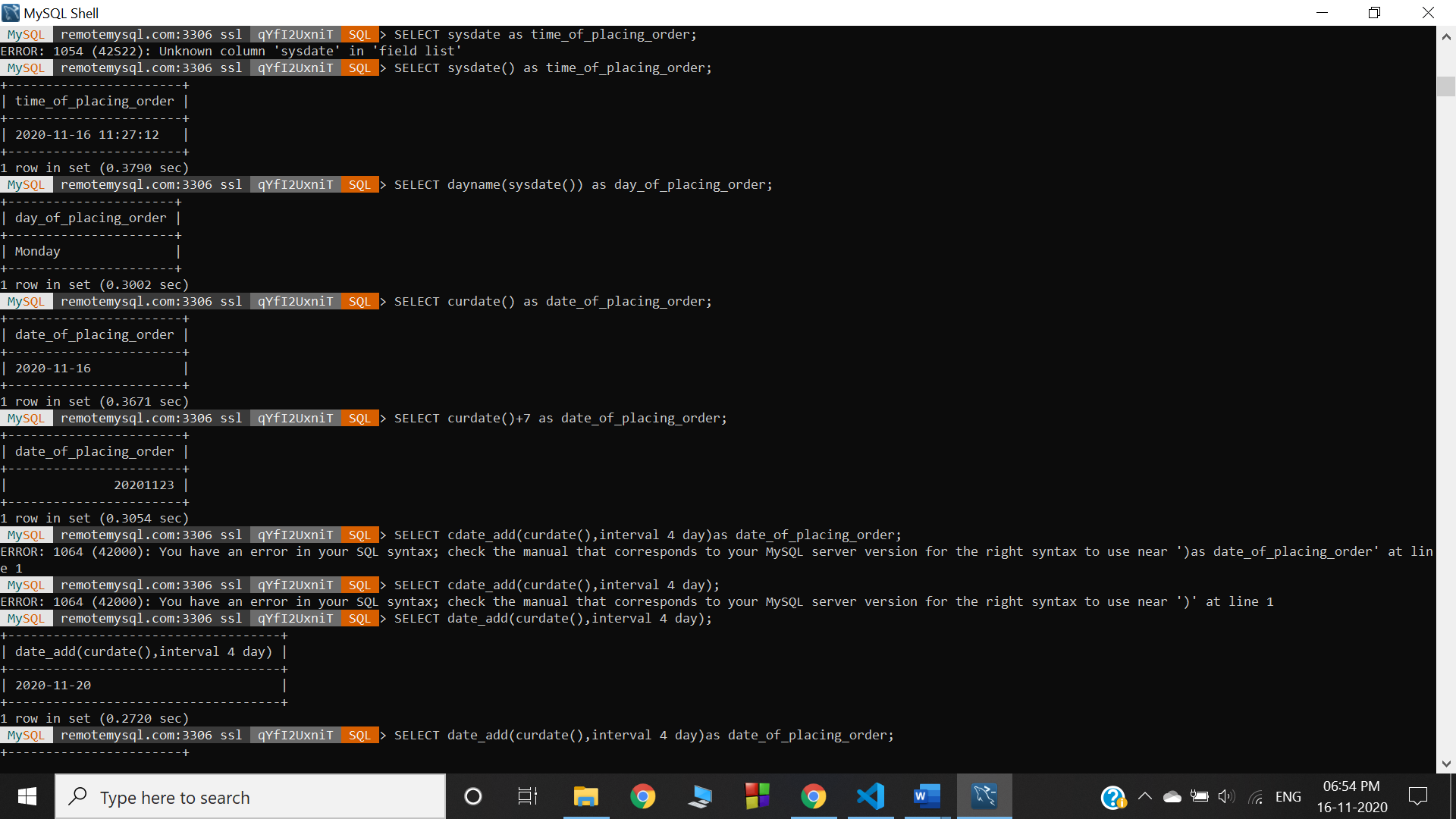
The function sysdate() gives current date and time. In the above query we are giving the live time as time of placing order.

**SELECT dayname(sysdate()) as day\_of\_placing\_order;**



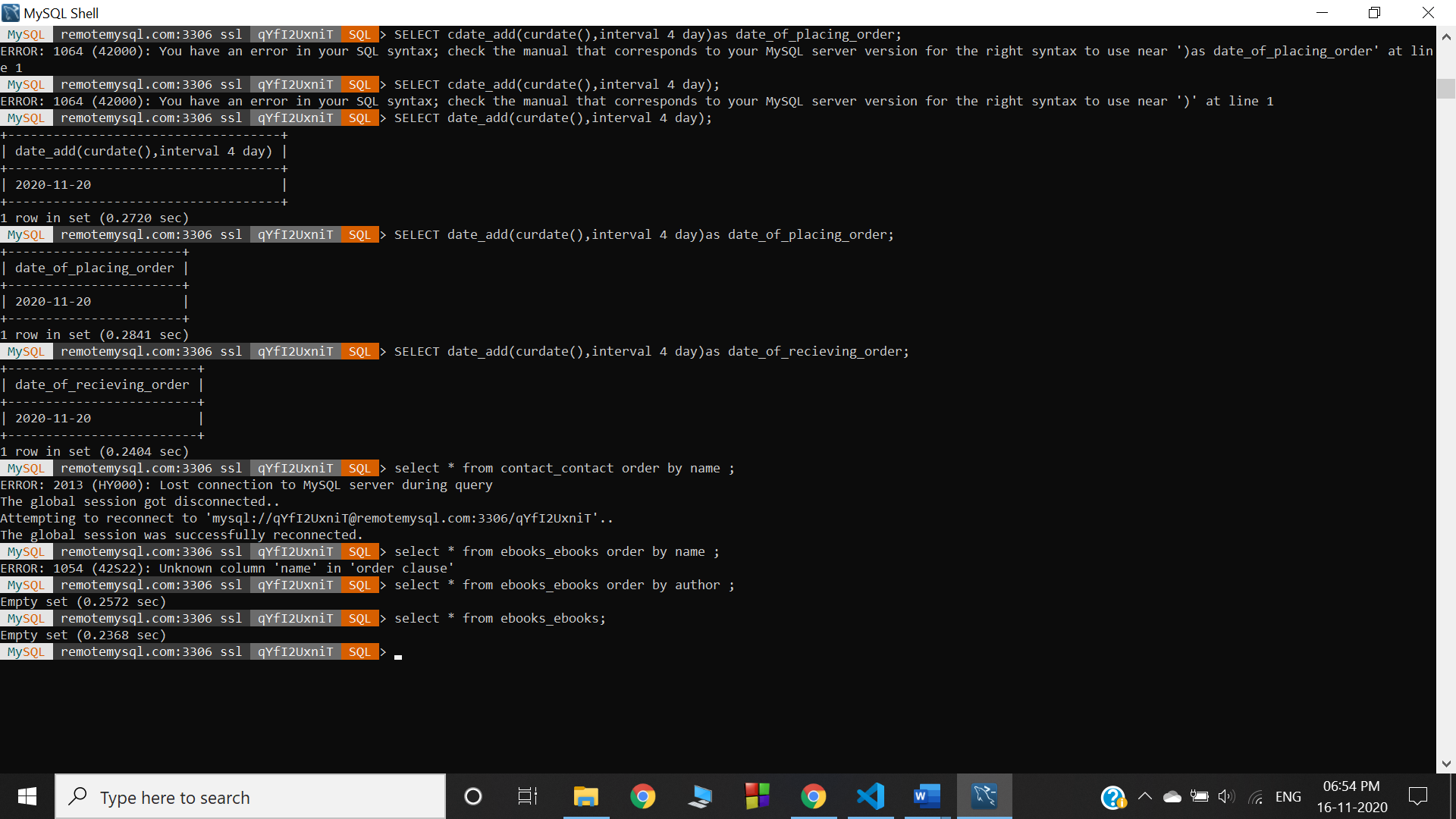
Dayname() gives the name of the day as mentioned above.

**SELECT curdate() as date\_of\_placing\_order;**

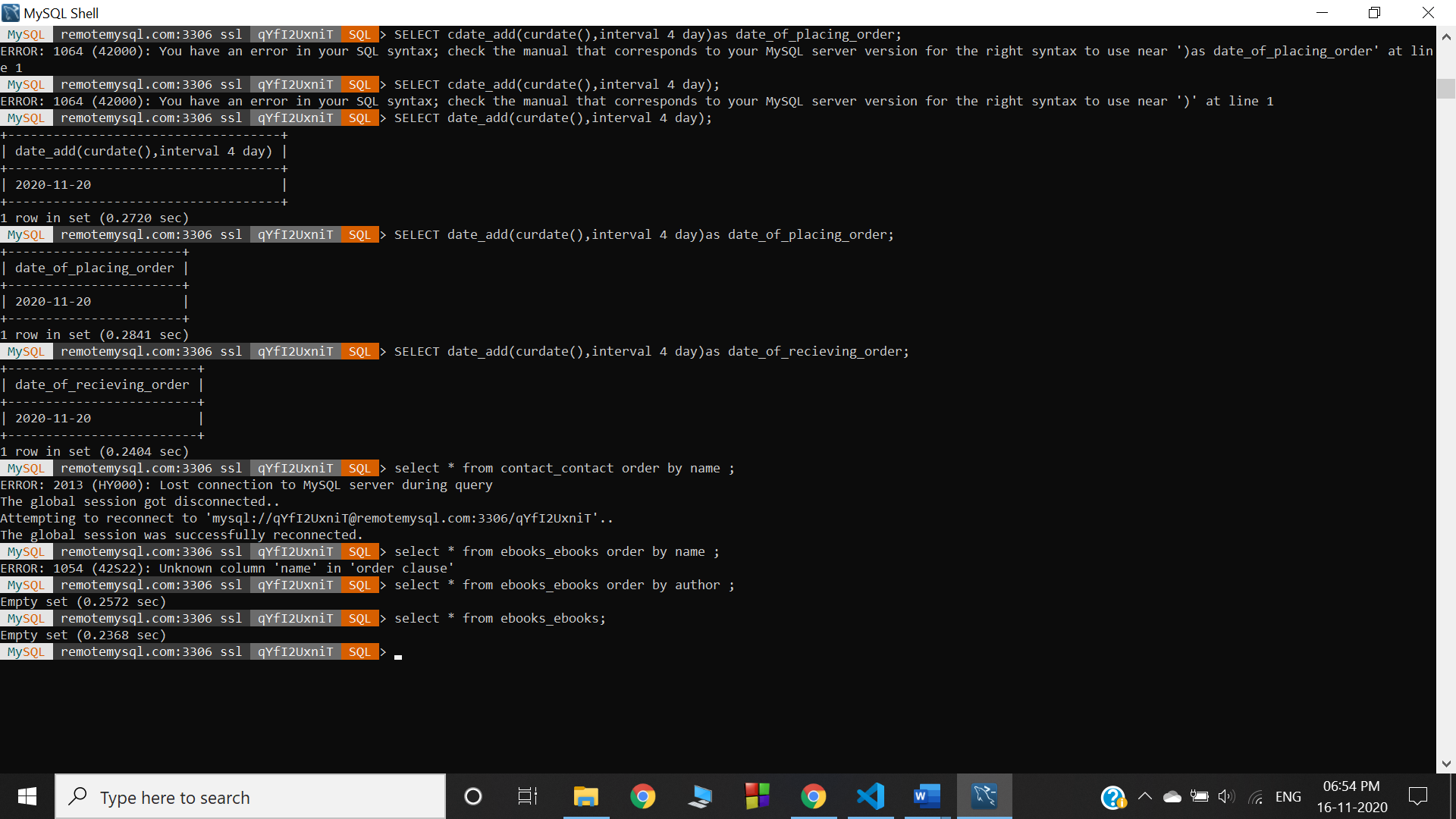


The function curdate() gives current date . In the above query we are giving the live date as date of placing order.

**SELECT date\_add(curdate(),interval 4 day)as date\_of\_placing\_order;**

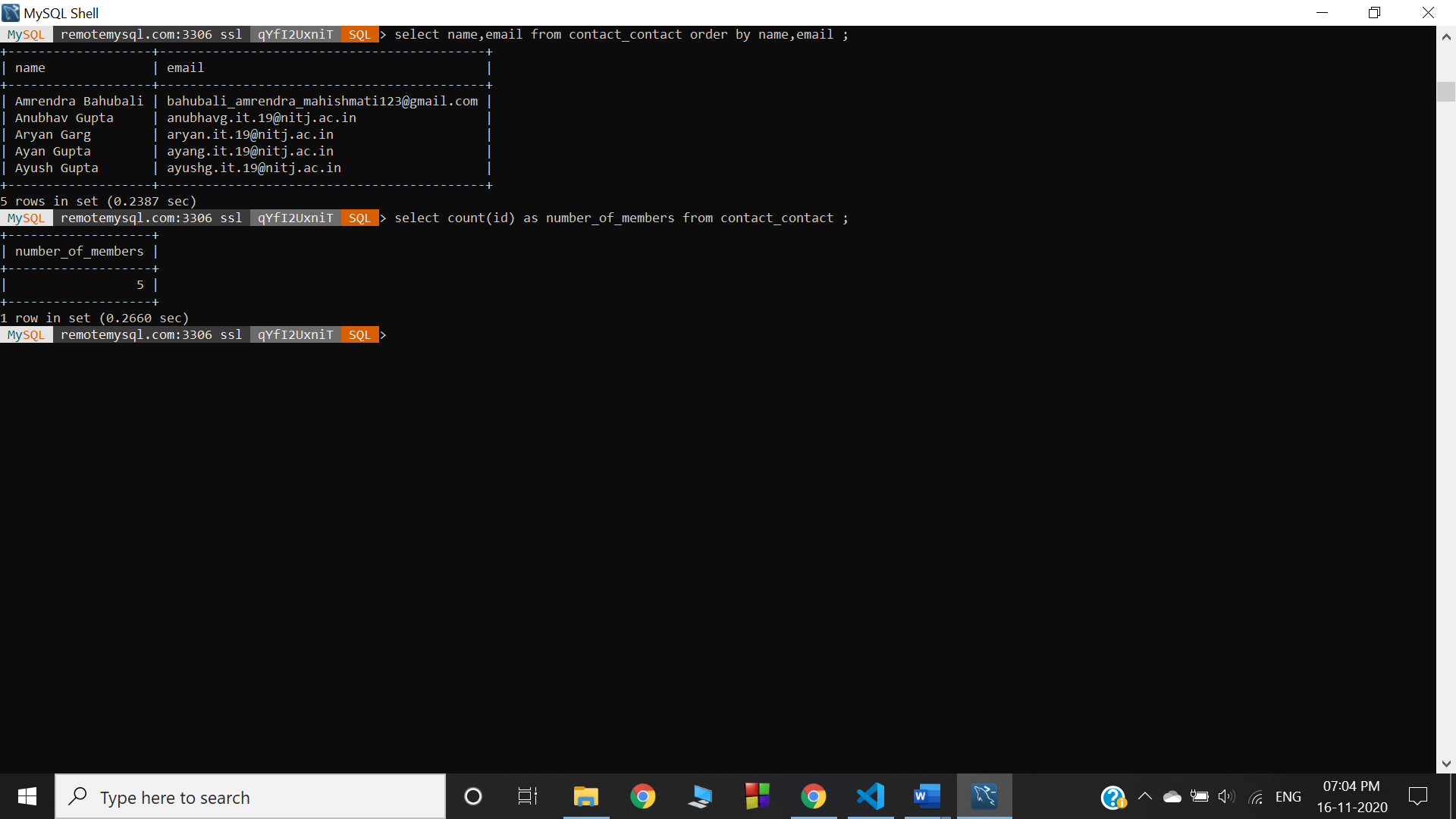


**SELECT date\_add(curdate(),interval 4 day)as date\_of\_recieving\_order;**



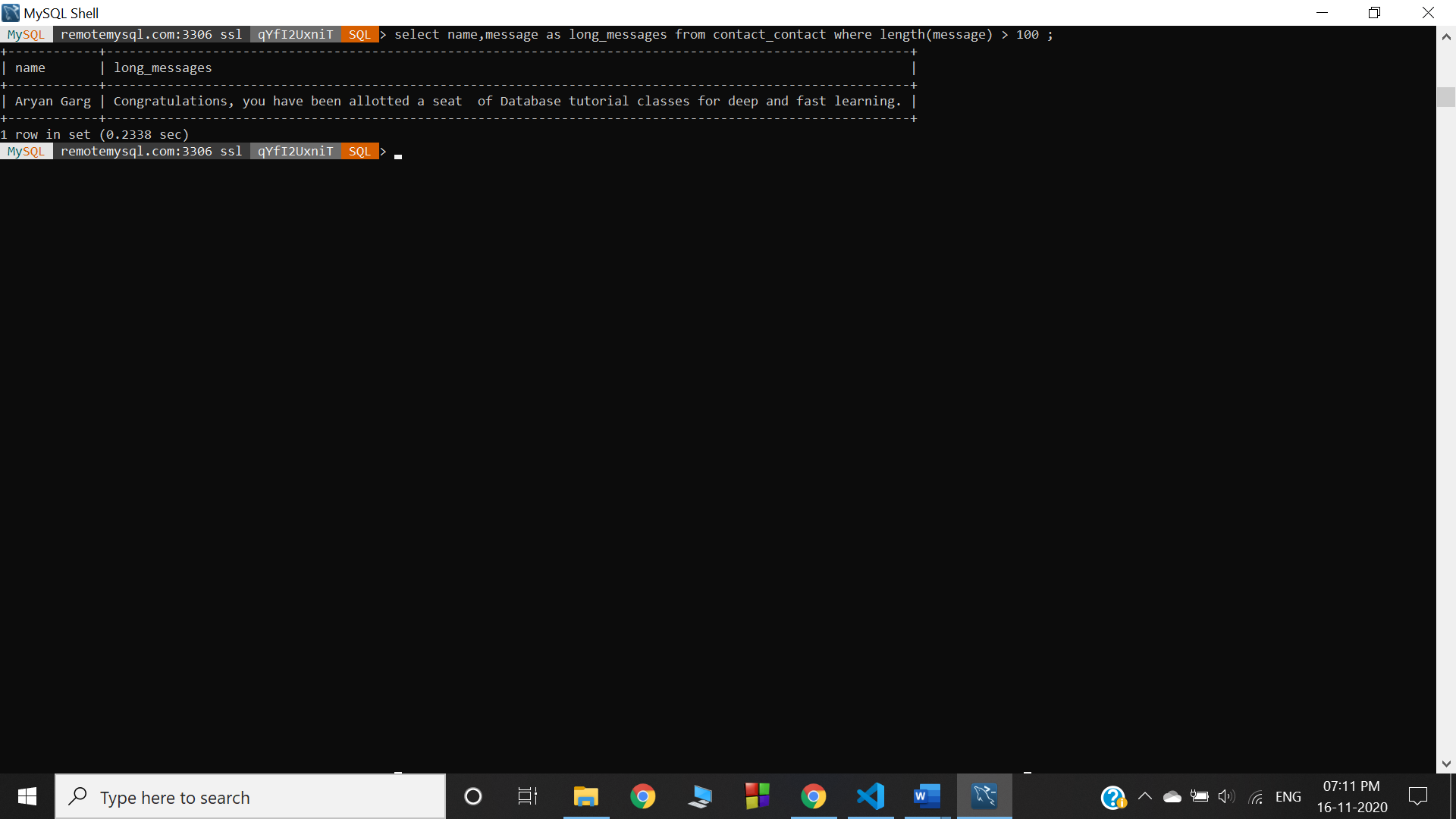
The function date\_add() is used the mention time interval that may be in days minutes or hours in the date mentioned. In the above two queries an interval of 4 days is added to curdate() as it takes normally 4 days to deliver order.

**select count(id) as number\_of\_members from contact\_contact ;**



The function count() is used to count number of rows.

**select name,message as long\_messages from contact\_contact where length(message) > 100 ;**



The function length () is used to count the number of letters. In above query columns name and messages are displayed with messages having letters more than 100.

blog   = Blog.objects.filter(trends = 'Y')

this gives out trending blogs that can be listed on homepage of our site

mysql syntax is equivalent to:

**select \* from blog\_blog where trends = “Y”;**

    context   = Blog.objects.all()

this is to render blogs on Blogs page under command :

**select \* from blog\_blog where trends = “Y”;**

    context = Blog.objects.filter(name=name)

this particular Query is used to filter a blog and to render it’s detailed description to user, Hence this equivalent to MySql as

**Select \* from blog\_blog where name = “”;**

**In E-Books App:**

    ebooks = Ebooks.objects.filter(trends = 'Y')

this gives out trending ebooks that can be listed on homepage of our site

mysql syntax is equivalent to:

**select \* from Ebook\_ebook where trends = “Y”;**

    ebooks   = Ebooks.objects.all()

**select \* from Ebook\_ebook ;**

**In Newsletter App:**

 if Email.objects.filter(email=email).exists():

                 messages.info(request, 'You are already in our Family')

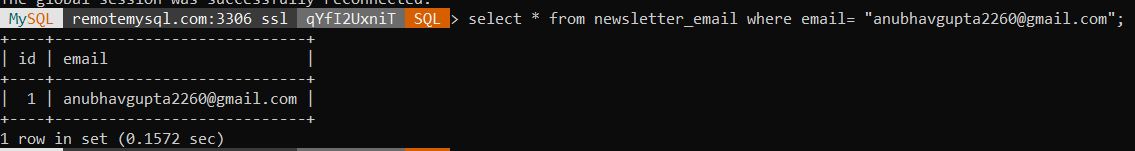
            else:

                data = Email(email=email)

                data.save()

This query is made to database to prevent redundancy and help validating the fields of subscription form,

This is given in Mysql as:



****

            context = News.objects.all()

This query retrieves data to be fed into newsletter and is used to send newsletter to emails stored in the database