**Online Restaurant Management System**

****

**Course:** CSE250 Database Management System

**Semester:** 4

**Section:** 2

**Faculty:** Prof. Shefali Naik

|  |  |  |
| --- | --- | --- |
| **Name** | **Enrollment No.** | **Course** |
| Meghna Kanade | AU1940134 | B-Tech CSE |
| Jeenal Shah | AU1940156 | B-Tech CSE |
| Hrutika Patel | AU1940182 | B-Tech CSE |

**Description of Project**

This is a Python project for a simple "**Online Restaurant Management System**," written in Python and MySQL. The main goal of this system is to solve the issue of the restaurant's current manual table booking, order, and billing system by converting it into an easy and error-free method using the Python Online Restaurant Management System. Currently, when we have a go-to restaurant, the waiter comes and takes our order, writes it in the diary, and then manually tells the order to the cook, and when we create a bill, the waiter goes to the cashier counter and manually tells the customer order details, but with this system, the waiter can directly store customer order details.

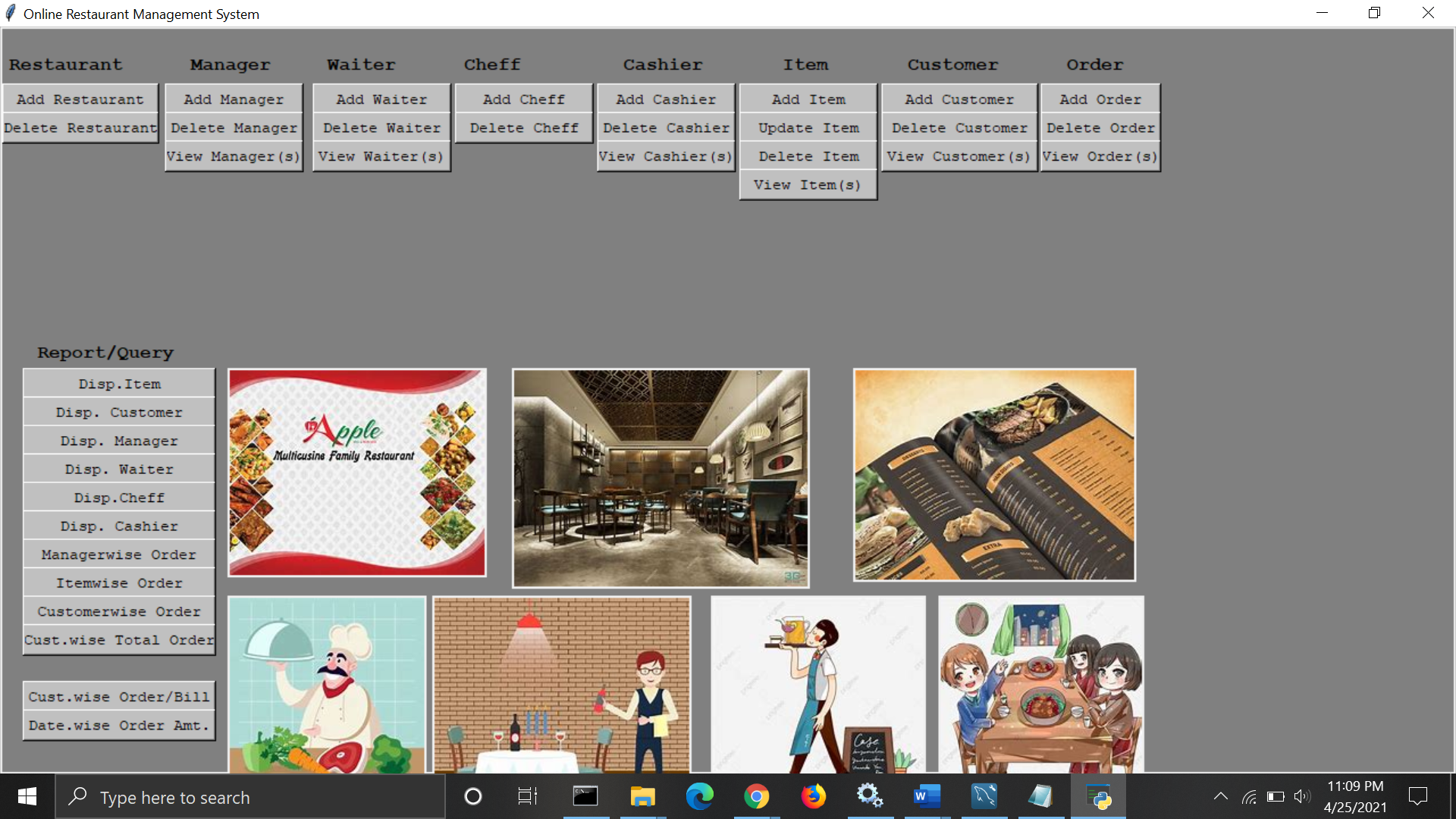


Figure 1: User interface screenshot

**Modules of Online Restaurant Management System**

Following are the Modules of Online Restaurant Management System.

1. Order Management Module
2. Billing Management Module
3. Food Item Module
4. Tax Management Module
5. Customer Management Module
6. Staff Management Module

**Feature of Online Restaurant Management System**

1. There is only User under this system.
2. User can Set up Online Restaurant Management System.
3. User can Add Update Delete and View Food Item Data.
4. User can Add Delete and View Cashier details.
5. User can Add Update Delete and View Restaurant Table details.
6. User can Add and Delete Chef Data.
7. User can Create Update delete and View Restaurant Order data.
8. User can Create Edit Delete and View Restaurant Billing Data.
9. User can view data of Today sales, Yesterday sales, Last 7 Days sales and All-time sales data also.
10. Cashier User can View list of billing data, he can check the status of bill and he can take print of bill.
11. User can Create Update Delete and View Restaurant Manager Data.
12. User can Create Update Delete and View Restaurant Customers Data.
13. The system will show a system date based on the current date.
14. If a user enters an incorrect price for a food item, the device will check it against the database and automatically correct it if it is incorrect.
15. After selecting a food itemNo, when you search for an item, the item's details are automatically displayed.
16. The system will also show the order amount by date.
17. The system displays the order and bill for each individual customer.

**Web Technology Used in Online Restaurant Management System**

There following Web Technology has been used to build this Online Restaurant Management System.

Server-Side

1. Python
2. MySQL Database

**Entity-Relationship Diagram**

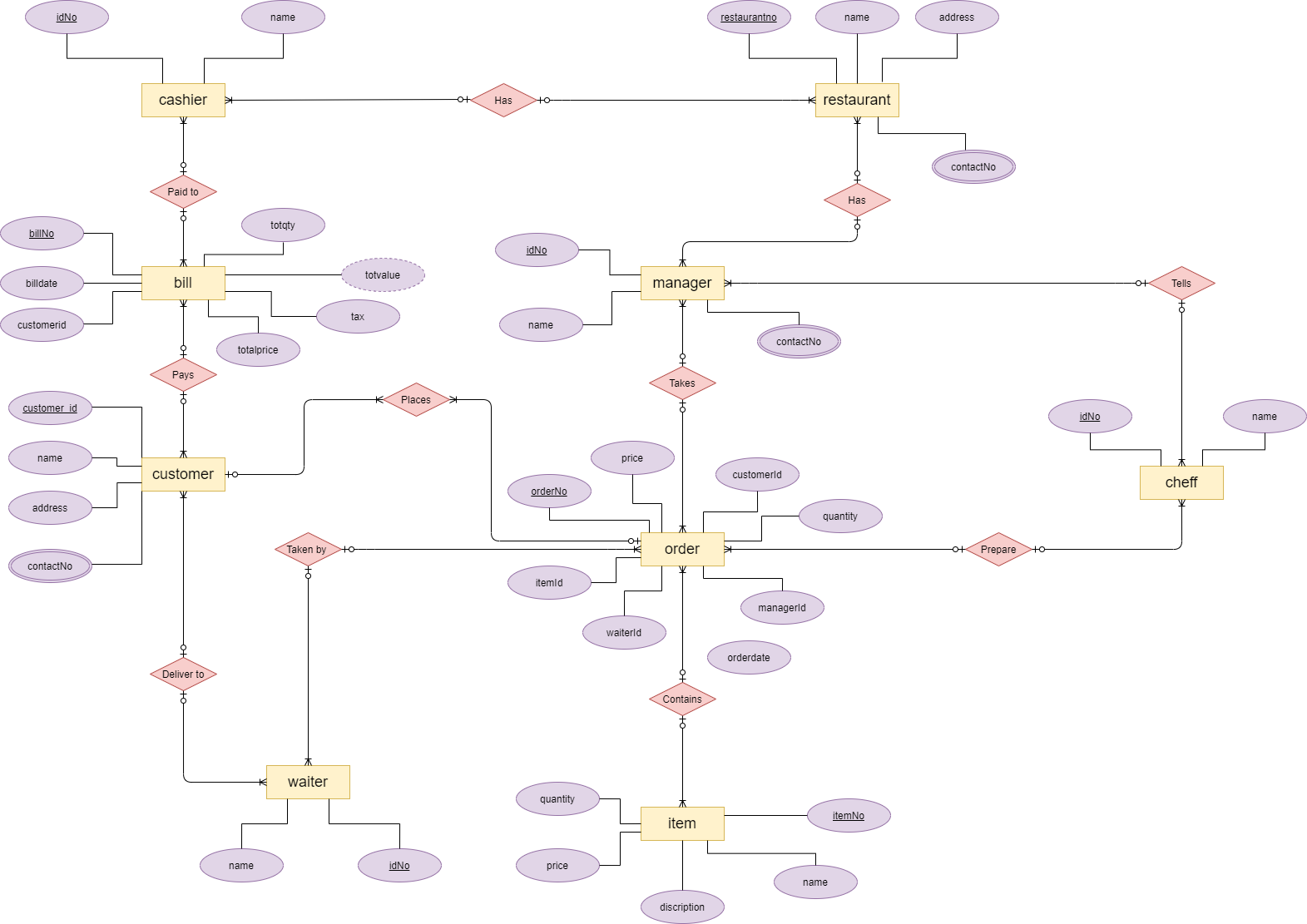


Figure 2: ER diagram. Created using - https://www.diagrams.net/blog/insert-sql

**Database Design (Table Design)**

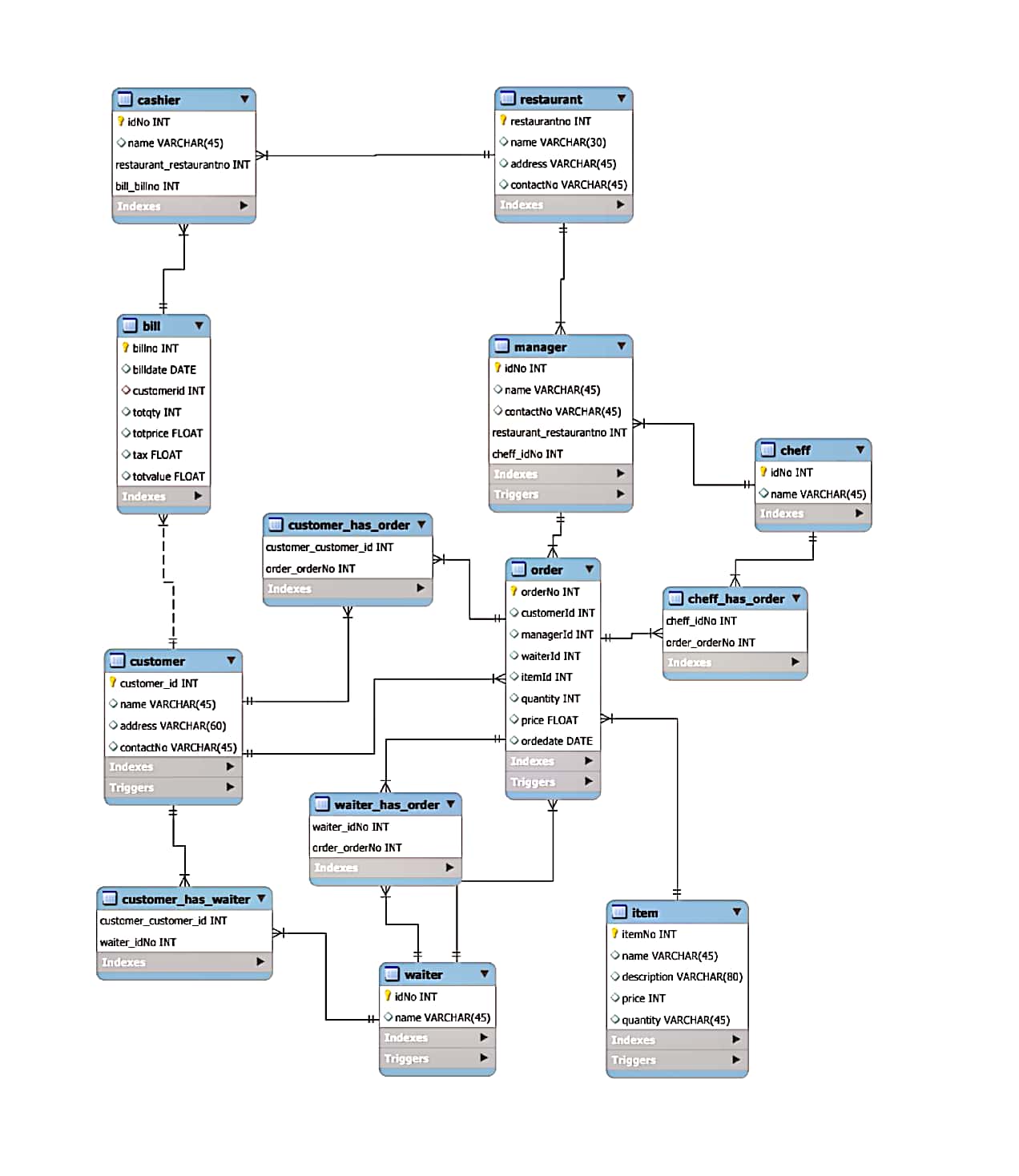


Figure 3: Schema of the project. Screenshot taken from MySQL.

**Procedures and Triggers**

**Procedures**

**1. This practice displays the name of the cashier. (With this, we can locate a cashier's information as well as a linked database.)**

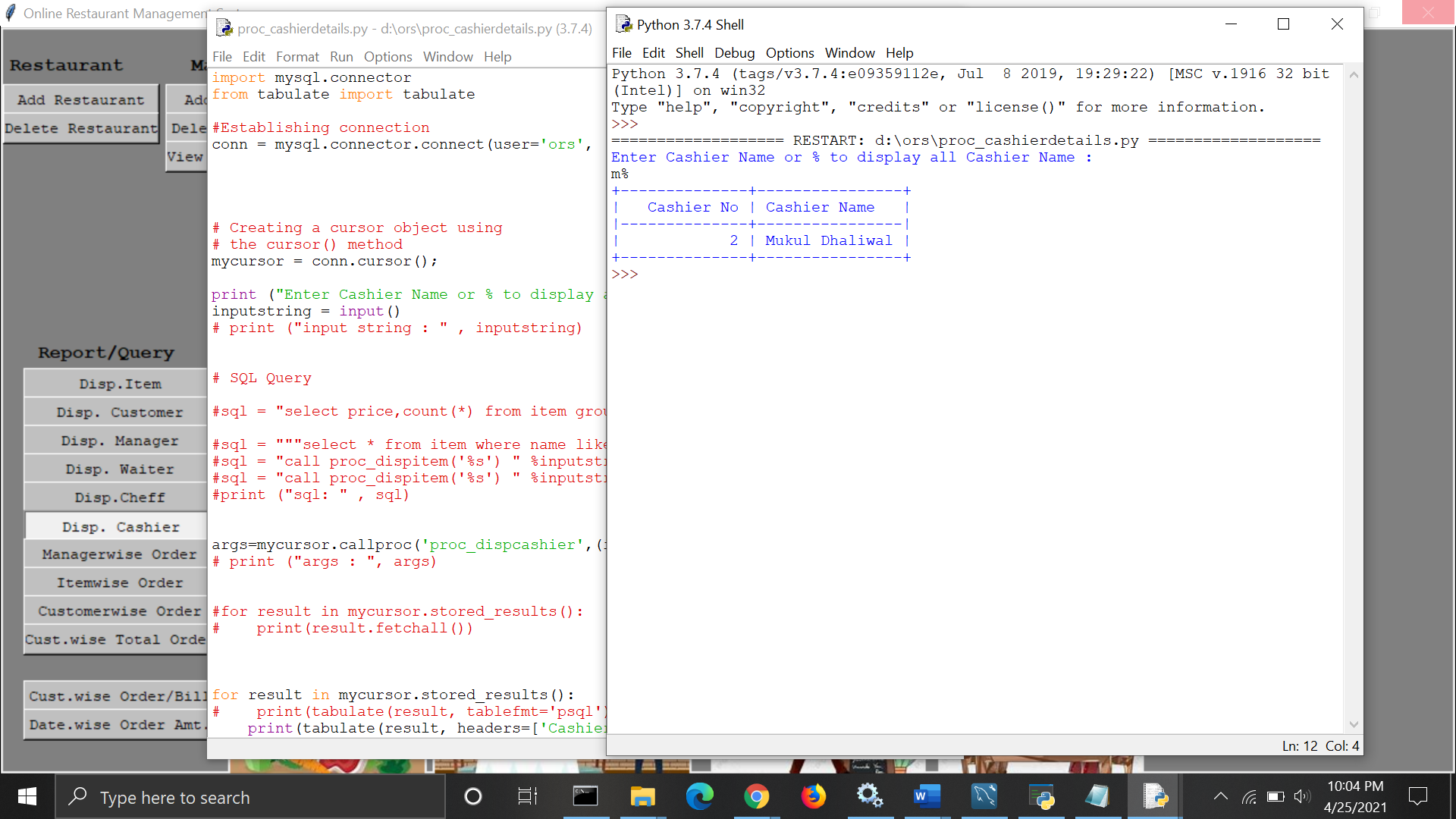
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_dispcashier`(  in cashiername varchar(45)  )  BEGIN  select \* from cashier where name like cashiername ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_dispcashier('m%');  call proc\_dispcashier('%'); |

**Output on the frontend:**



**2. This practice displays the name of the cheff. (With this, we can locate a cheff's information as well as a linked database.)**

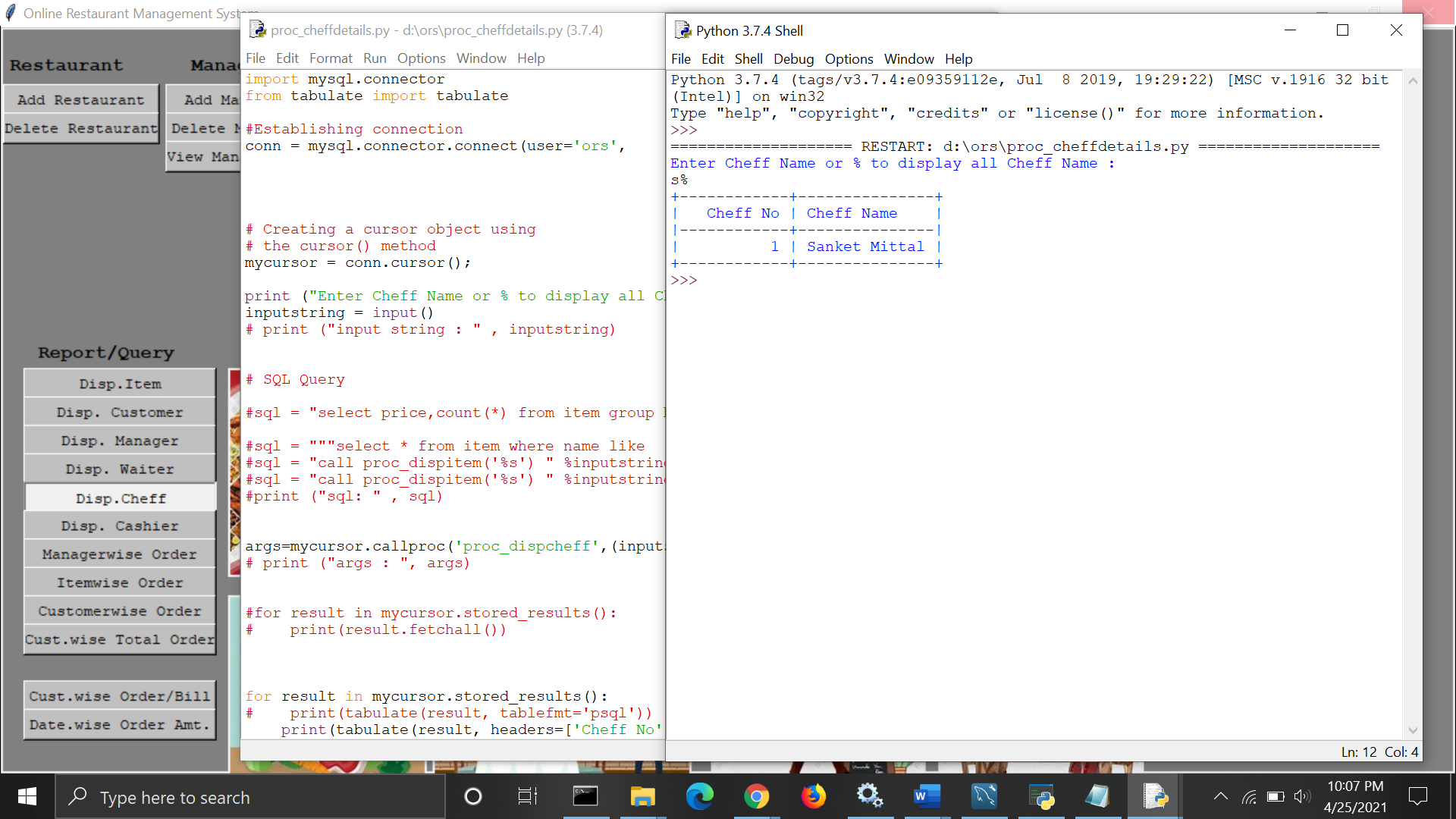
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_dispcheff`(  in cheffname varchar(45)  )  BEGIN  select \* from cheff where name like cheffname ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_dispcheff('s%');  call proc\_dispcheff('%'); |

**Output on the frontend:**



**3. This practice displays the name of the customers. (With this, we can locate a customer's information as well as a linked database.)**

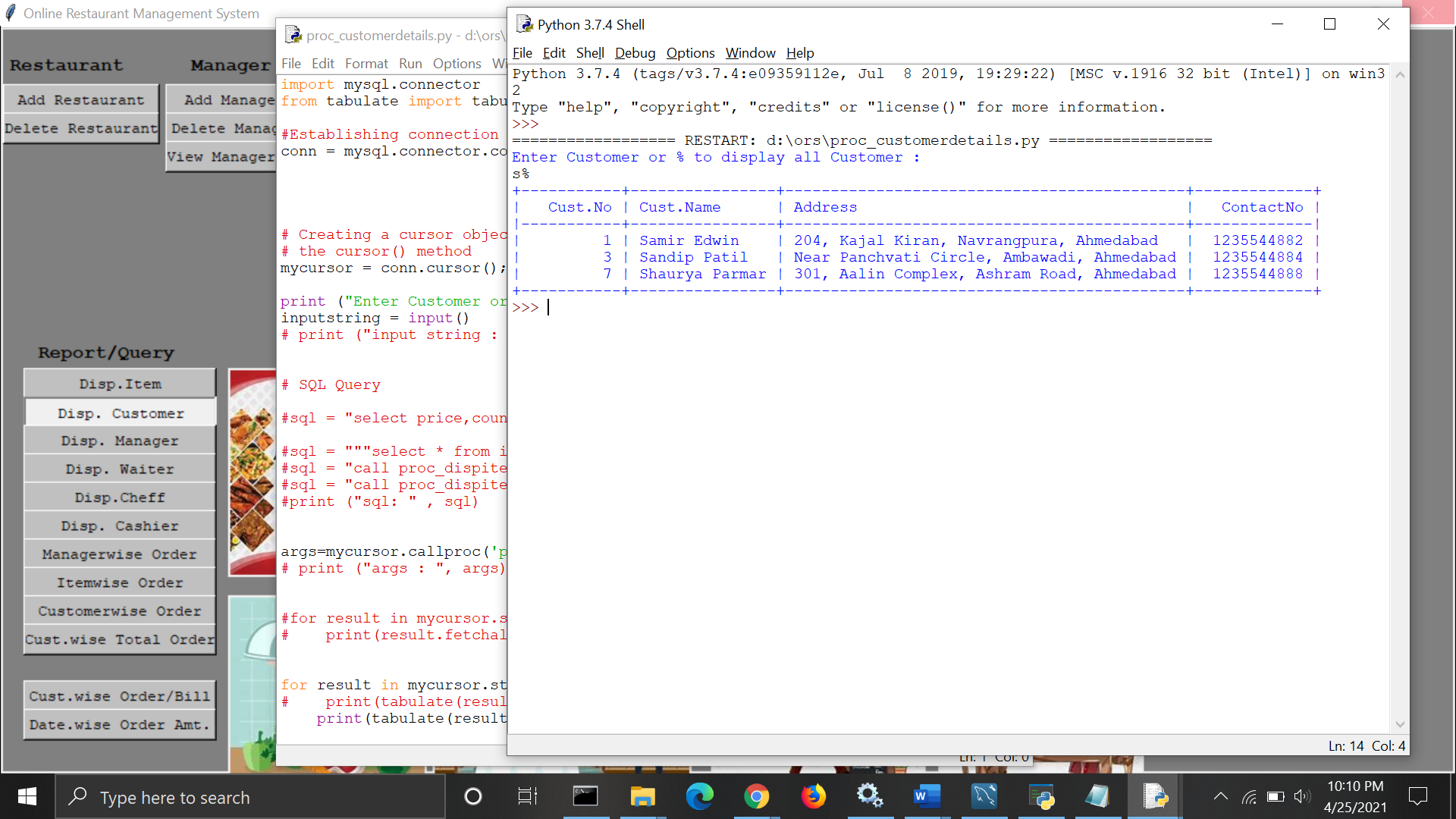
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_dispcust`(  in custname varchar(45)  )  BEGIN  select \* from customer where name like custname ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_dispcust('s%');  call proc\_dispcust('%'); |

**Output on the frontend:**



**4. This practice displays the name of the manager. (With this, we can locate a manager's information as well as a linked database.)**

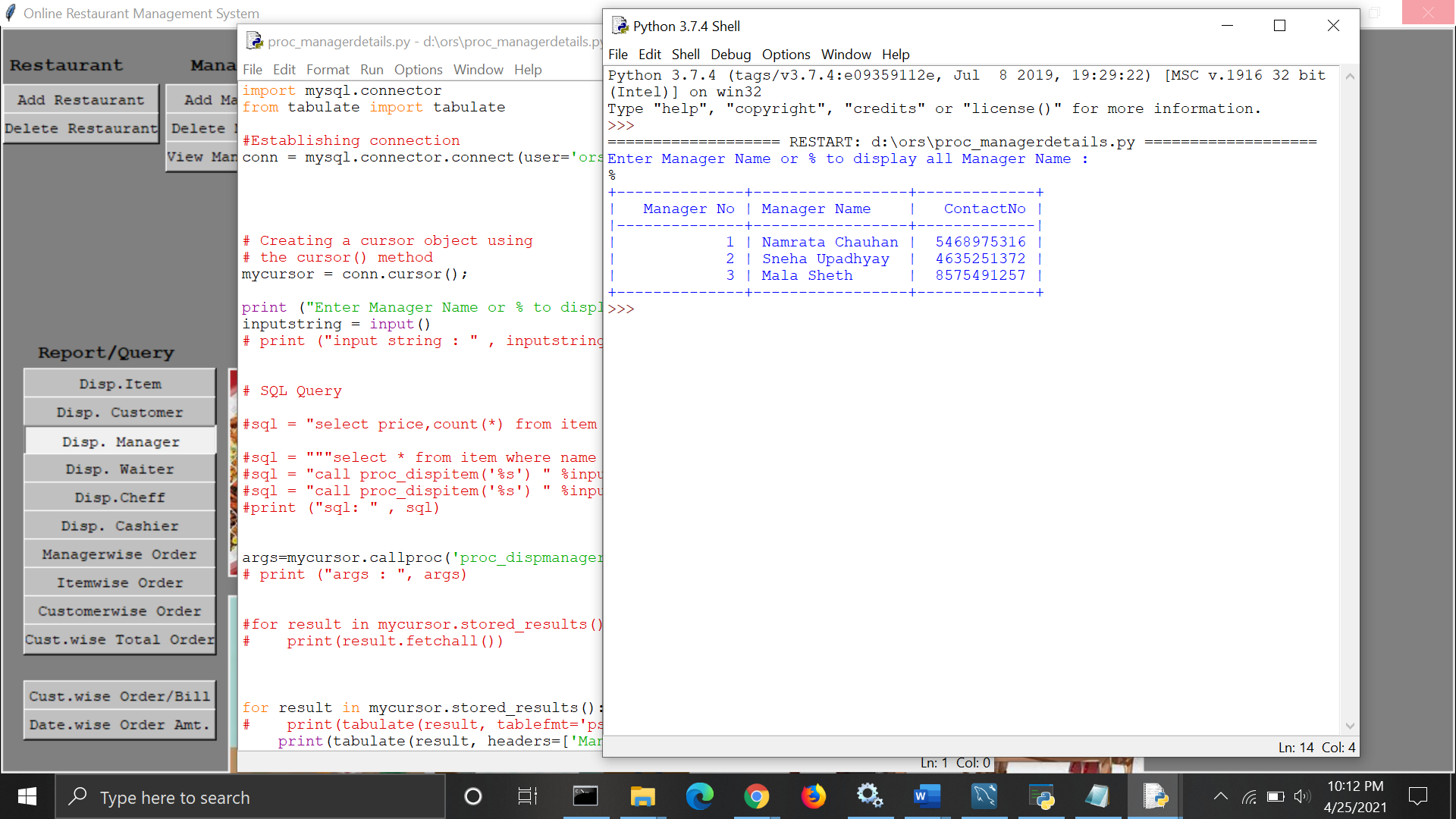
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_dispmanager`(  in managername varchar(45)  )  BEGIN  select \* from manager where name like managername ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_dispmanager('s%');  call proc\_dispmanager('%'); |

**Output on the frontend:**



**5. This practice displays the name of the waiter. (With this, we can locate a waiter's information as well as a linked database.)**

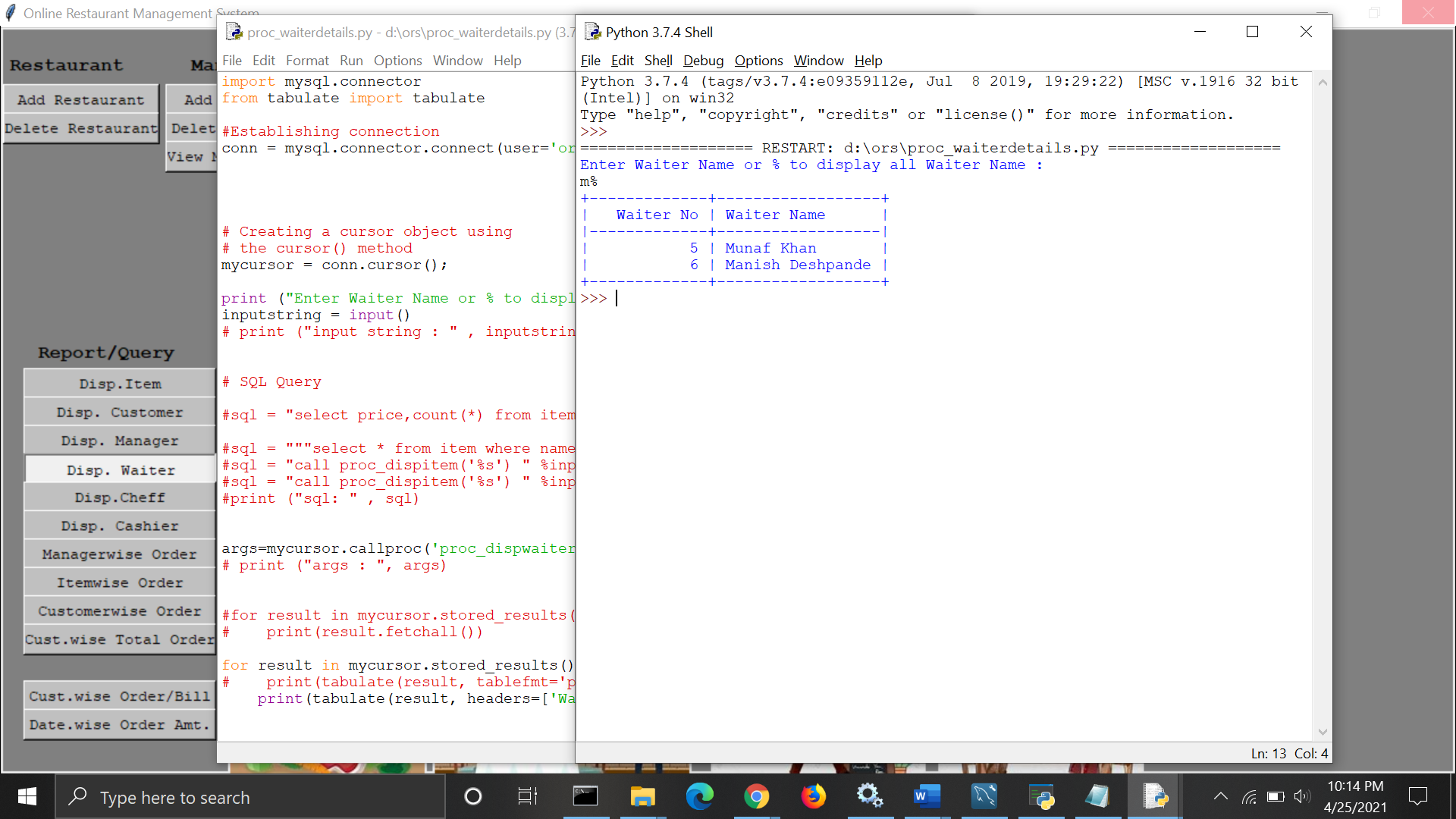
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_dispwaiter`(  in waitername varchar(45)  )  BEGIN  select \* from waiter where name like waitername ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_dispwaiter('m%') ;  call proc\_dispwaiter('%') ; |

**Output on the frontend:**



**6. This procedure retrieves an item's number, name, price, and quantity from the item table and displays them. (We can find item descriptions and related databases using this method.)**

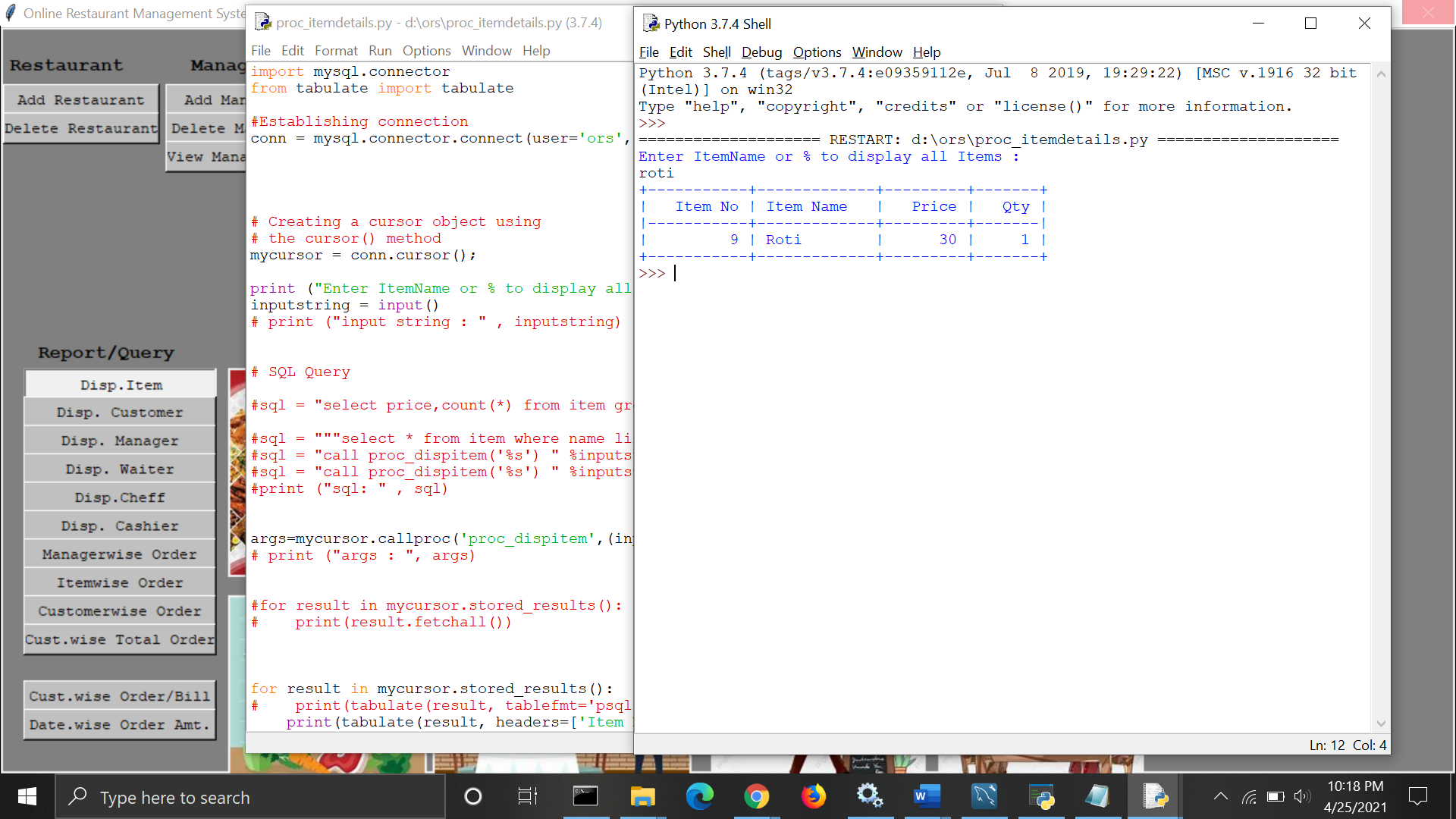
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_dispitem`(  in itemname char(45)  )  BEGIN  select itemno,name,price,quantity  from item where name like itemname ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_dispitem('roti%');  call proc\_dispitem('%'); |

**Output on the frontend:**



**7. This technique displays a list of items in chronological order. (We can use this to locate an object as well as date-specific information. We can also identify most sold and popular item.)**

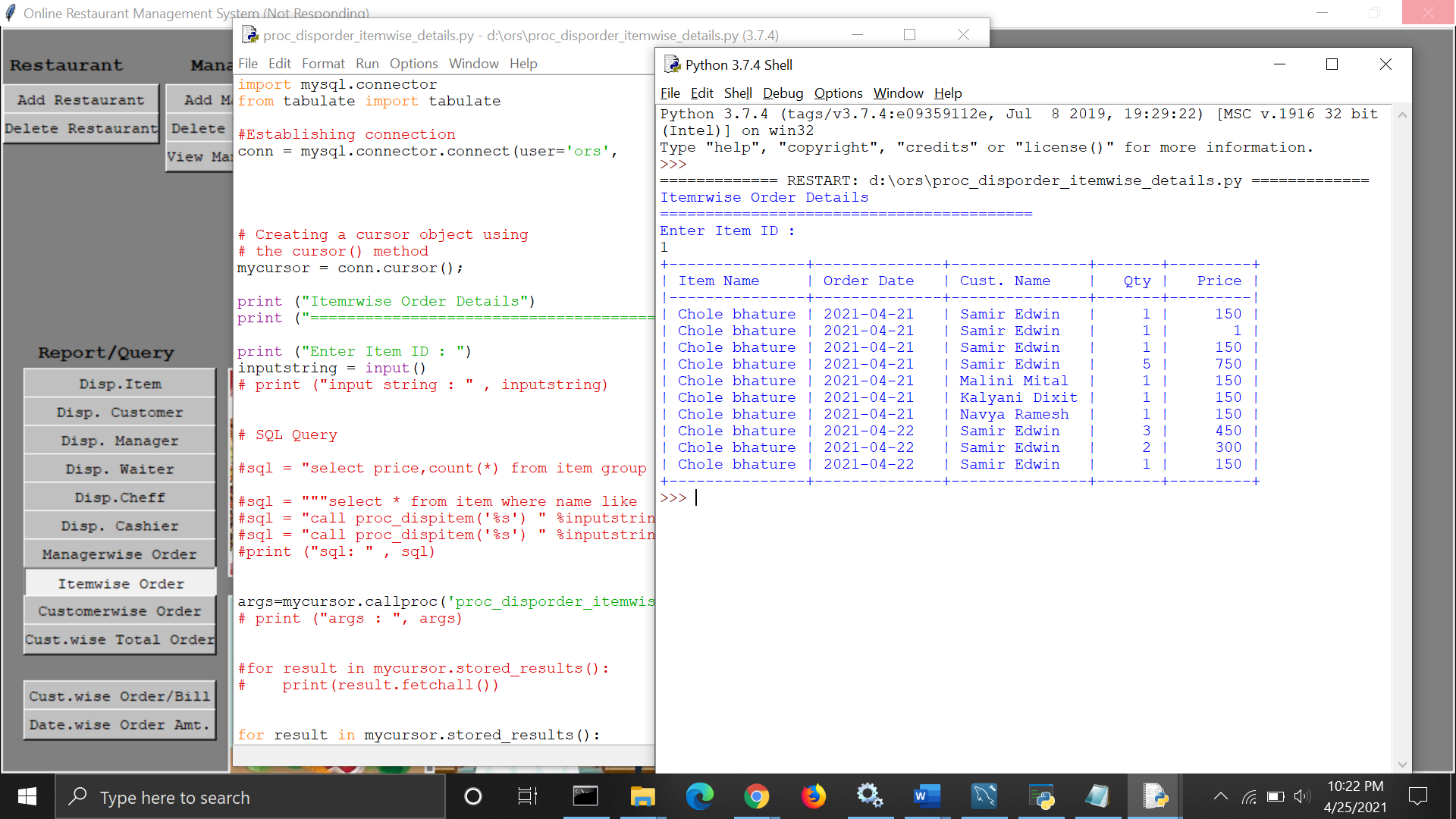
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_disporder\_itemwise`(  in itemnumber int  )  BEGIN  SELECT i.name, o.orderdate, c.name, o.quantity , o.price  FROM ors.order o , item i , customer c  where o.itemId = i.itemNo  and o.customerId = c.customer\_id  and o.itemId = itemnumber  order by o.itemId, o.orderdate, o.customerId ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_disporder\_itemwise('1'); |

**Output on the frontend:**



**8. This method shows a manager wise order. (We can use this to locate a specific order placed by that manager.)**

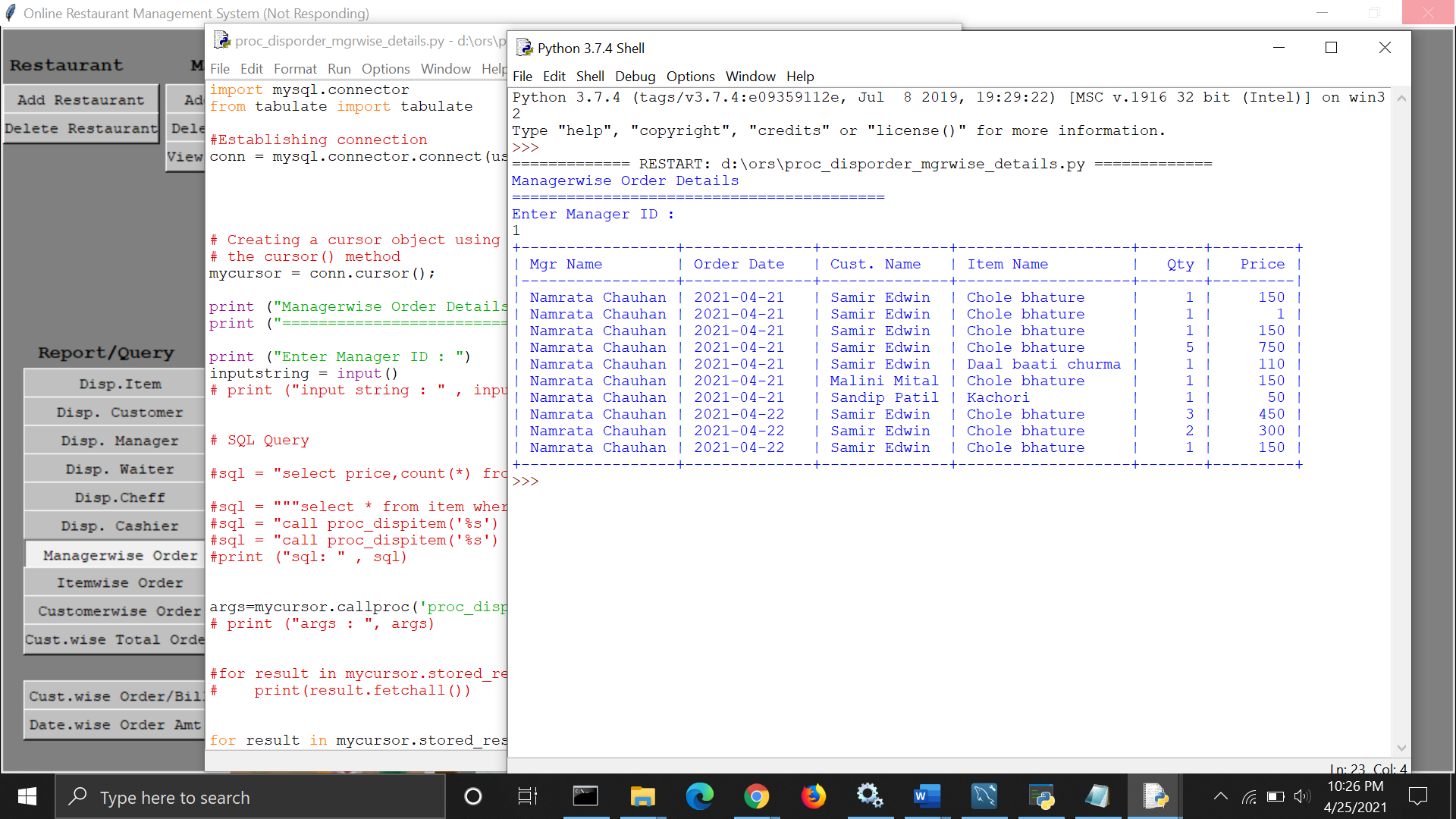
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_disporder\_mgrwise`(  in mgrno int  )  BEGIN  SELECT m.name, o.orderdate, c.name, i.name, o.quantity , o.price  FROM ors.order o , manager m , item i , customer c  where o.managerid = m.idno  and o.itemId = i.itemNo  and o.customerId = c.customer\_id  and o.managerId = mgrno  order by o.managerId, o.orderdate, o.customerId, o.itemId ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_disporder\_mgrwise('1'); |

**Output on the frontend:**



**9. This process displays a customer wise order. (We can figure out how many things the customer ordered using this information.)**

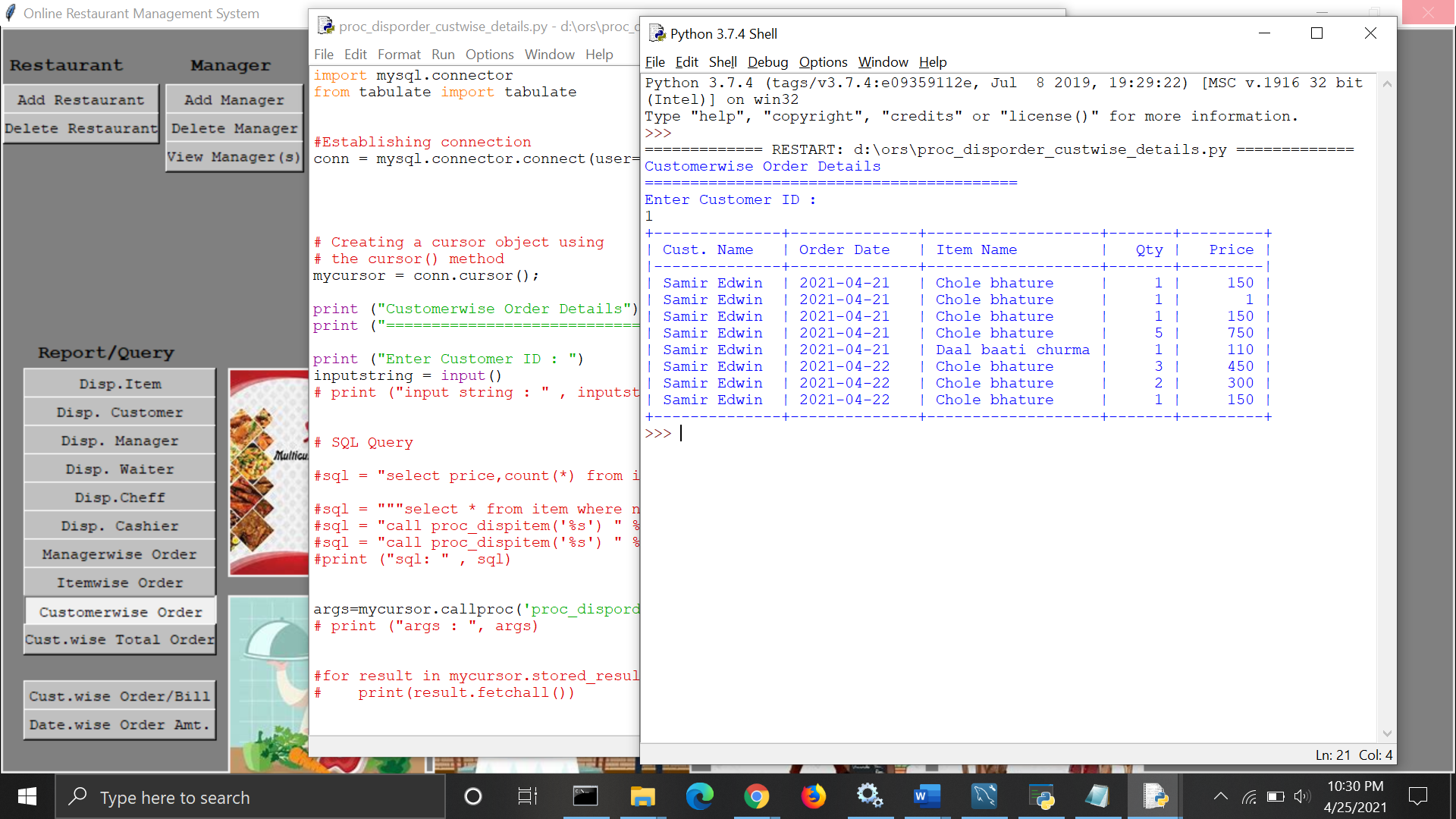
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_disporder\_custwise`(  in custnumber int  )  BEGIN  SELECT c.name, o.orderdate, i.name, o.quantity , o.price  FROM ors.order o , item i , customer c  where o.itemId = i.itemNo  and o.customerId = c.customer\_id  and o.customerId = custnumber  order by o.customerId , o.orderdate, o.itemId ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_disporder\_custwise('1'); |

**Output on the frontend:**



**10. This procedure display a customer wise total order. (With this we can find a that how many items that customer ordered)**

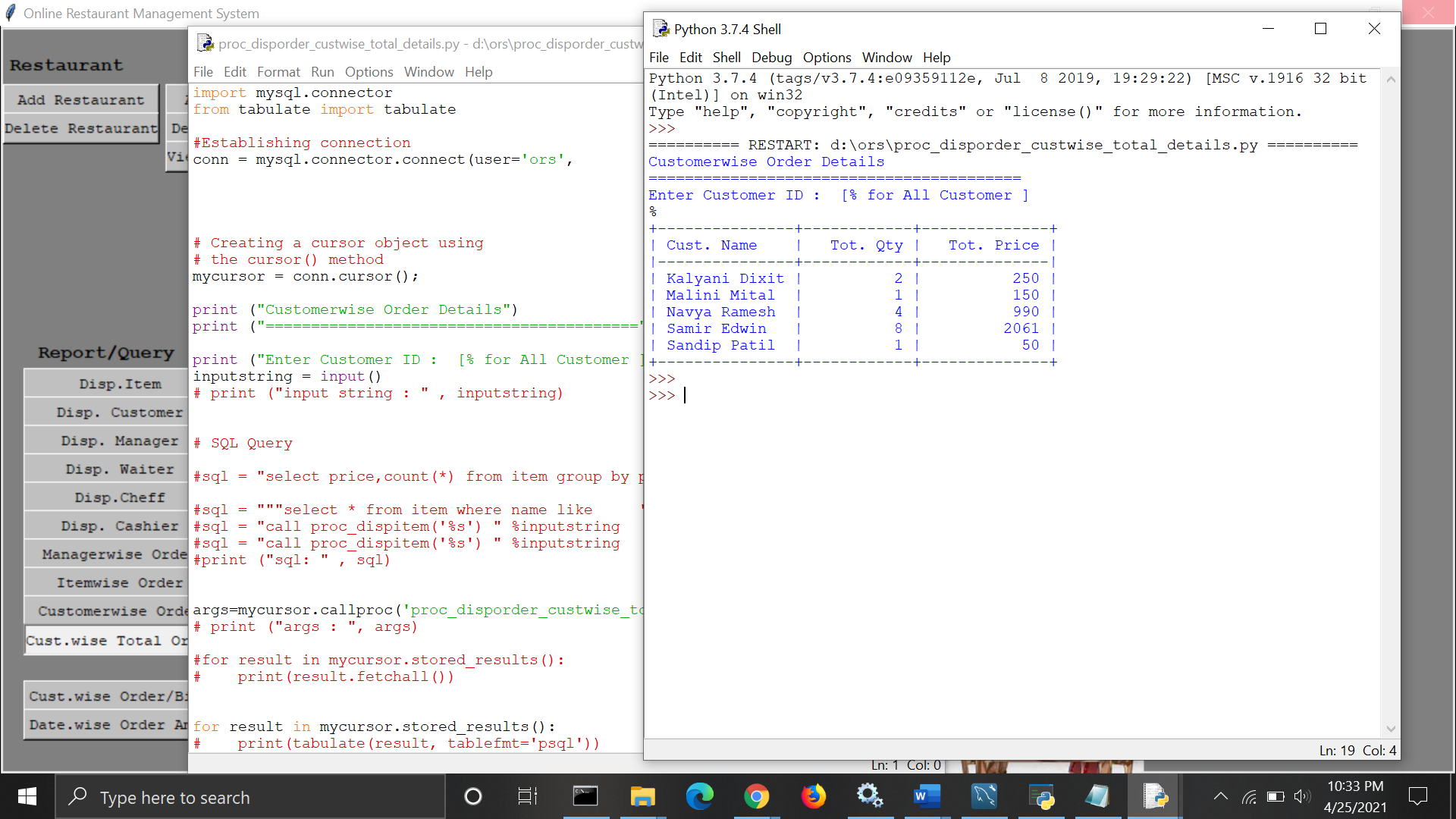
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_disporder\_custwise\_total`(  in custnumber varchar(45)  )  BEGIN  SELECT c.name, count(o.quantity) , sum(o.price)  FROM ors.order o , item i , customer c  where o.itemId = i.itemNo  and o.customerId = c.customer\_id  and o.customerId like custnumber||'%'  group by c.name  order by c.name ;  END |

**Statement to call procedure:**

|  |
| --- |
| call proc\_disporder\_custwise\_total('1');  call proc\_disporder\_custwise\_total('%'); |

**Output on the frontend:**



**Triggers**

**1. When we delete customer data, it also deletes data from other tables such as orders and bills.**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` TRIGGER `customer\_BEFORE\_DELETE`  BEFORE DELETE ON `customer`  FOR EACH ROW BEGIN  delete from ors.bill where customerId = OLD.customer\_id ;  delete from ors.order where customerId = OLD.customer\_id ;  END |

**2. When itemNo is removed from the item table. It will also exclude the item's data from the order table.**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` TRIGGER `item\_BEFORE\_DELETE`  BEFORE DELETE ON `item`  FOR EACH ROW BEGIN  delete from ors.order where itemid = OLD.itemno ;  END |

**3. When the managerId column in the manager table is removed. It will also exclude the manager's information from the order table.**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` TRIGGER `manager\_  BEFORE\_DELETE` BEFORE DELETE ON `manager`  FOR EACH ROW BEGIN  delete from ors.order where managerid = OLD.idno ;  END |

**4.** **When we put in a new order The currentdate will be automatically changed in Ordrdate (sysdate). It will also automatically change the item price based on the latest order quantity and price from the item table.**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` TRIGGER `order\_BEFORE\_INSERT`  BEFORE INSERT ON `order`  FOR EACH ROW BEGIN  Declare  v\_price float;  begin  set NEW.orderdate=curdate();  set @v\_price := (select price from item where itemno=new.itemid);  set NEW.price = NEW.quantity \* @v\_price ;  END;  END |

**5. When waiterId is removed from the waiter table. It will also exclude the waiter's information from the order table.**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` TRIGGER `waiter\_BEFORE\_DELETE`  BEFORE DELETE ON `waiter`  FOR EACH ROW BEGIN  delete from ors.order where waiterid = OLD.idno ;  END |

**Note: When the trigger fires, the data is automatically deleted or inserted in the backend. This can be seen from the backend.**

**Query**

**1. This query displays each customer's order as well as their bill.**

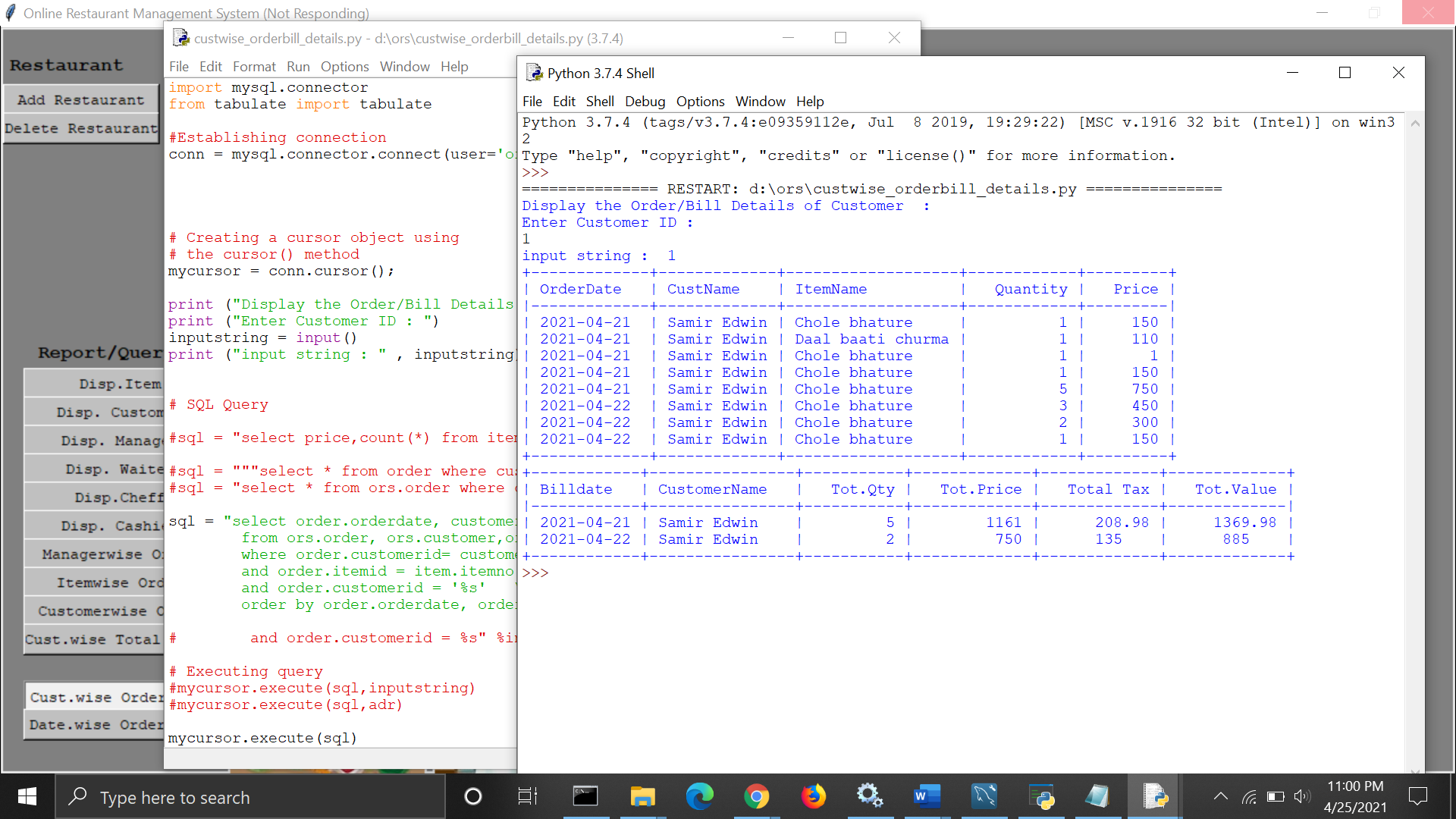
**Code:**

|  |
| --- |
| CREATE DEFINER=`ors`@`%` PROCEDURE `proc\_billsummary`(  in billdate date,  in custno int )  BEGIN  DECLARE v\_totqty int ;  DECLARE v\_totprice float ;  DECLARE v\_tax float;  DECLARE v\_totvalue float ;  Declare v\_x int default null;  Declare v\_y int default null;  select count(o.quantity), sum(o.price),  sum(o.price) \* 0.18 , (sum(price)+ (sum(o.price) \* 0.18 )) , 1  into v\_totqty, v\_totprice , v\_tax , v\_totvalue , v\_x  from ors.order o  where o.orderdate = billdate  and o.customerid = custno  group by o.orderdate, o.customerid;  select 1 into v\_y  from bill b  where b.billdate = billdate  and b.customerid = custno  group by b.billdate, b.customerid ;  if (v\_x = 1 and v\_y is null ) then  insert into bill(billdate,customerid,totqty,totprice,tax,totvalue) values (billdate,custno,v\_totqty,v\_totprice,v\_tax,v\_totvalue );  elseif (v\_x = 1 and v\_y = 1 ) then  update bill b  set b.totqty = v\_totqty,  b.totprice = v\_totprice,  b.tax = v\_tax,  b.totvalue = v\_totvalue  where b.billdate = billdate  and b.customerid = custno ;  end if;  commit;  END |

**Output:**

|  |
| --- |
| call proc\_billsummary(curdate(),1);  select \* from bill; |

**Output on the frontend:**



**2. This query displays total price, quantity, and orders by date.**

**Code:**

|  |
| --- |
| select o.orderdate , count(\*) , sum(o.quantity) , sum(o.price)  from ors.order o  group by o.orderdate  order by o.orderdate;  select o.orderdate , count(\*) , sum(o.quantity) , sum(o.price)  from ors.order o  where o.orderdate = '%s'  group by o.orderdate  order by o.orderdate; |

**Output on the frontend:**

