

## Project Description

The purpose of this project is creating a database system, which will be implemented in MS SQL and proper integrations to a website for Volkan Kozmetik. Database will store the sell and stock information of the company that is specialized in personal care products and cologne.

## Scope of the Project

We will be using Microsoft SQL to create our database. After entities and relations are described, the website should be ready. Each customer and company should see which categories are available. Each category should also include products that are produced and not out of stock. At the last step of the project we must implement both systems to each other.

## Business Requirements

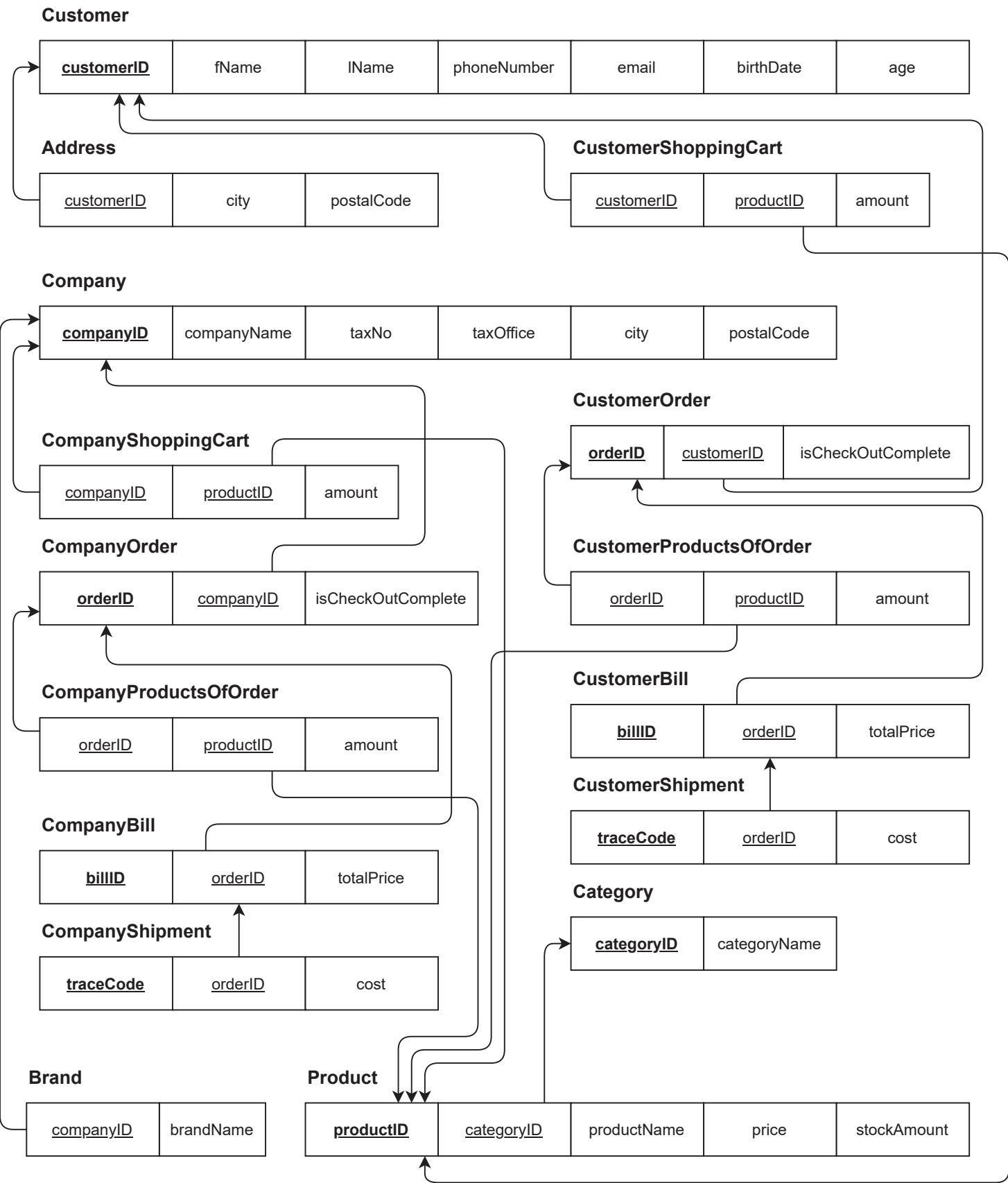
### Functional Business Requirement

1. **Extensible:** The database must be restatable and changeable according to the information of customers and companies. (Must)
2. **Accessible:** The employees who have access can reach the database from the website. (Must)
3. **Multiple User Support:** The employees or the customers can make work on the website at the same time. (Must)
4. **Data Change Support:** Forms will be used to allow users to change records according to their access level. (Must)
5. **Sorting:** Sorts should be generated quickly for allowing managers to reach first order given, highest amount etc. (Must)

### Non-Functional Business Requirement

1. **Microsoft SQL Server:** The database must be created using Microsoft SQL Server 2019. (Must)
2. **Entity Relation Diagram:** Create a diagram that includes the relations of the entities in the database. (Must)
3. **Table for Each Entity:** Create tables in the database according to the entity names with correct attributes. (Must)
4. **Achievable from Website:** Data must be achievable from website. (Must)
5. **Requirement Analysis Design:** Introduction of the project, list of entities and their definitions, list of functional and nonfunctional requirements. (Must)

# Volkan Kozmetik Database Mapping Diagram



## Tables

**Address:** The address of the customer that can be multivalued.

**Columns:** customerID (int), city (nvarchar(50)), postalCode (int).

**Keys:** customerID (FK)

**Brand:** The company's brand for representing produced products.

**Columns:** companyID (int), brandName (nvarchar(50))

**Keys:** companyID (FK)

**Category:** Cluster of products.

**Columns:** categoryID (int), categoryName (nvarchar(50))

**Keys:** companyID (FK)

**Company:** The table that buys huge amounts of products and uses the factory for their brands to be produced.

**Columns:** companyID (int) (+Identity) , taxNo (bigint) (+Unique), companyName (nvarchar(50))(+Unique), taxOffice (nvarchar(50)), address (nvarchar(50)).

**Keys:** companyID (PK)

**Company Bill:** Company bills are produced when the company checkouts the order.

**Columns:** billID(int) (+Identity), orderID(int), totalPrice(decimal(10,2)).

**Keys:** billID(PK), orderID(FK)

**Company Order:** Company orders are produced when the company creates an order for a product to be produced.

**Columns:** orderID(int) (+Identity) , companyID(int), isCheckOutComplete(int).

**Keys:** orderID(PK), companyID(FK)

**Company Product Of Order:** The product order that is the single element of the order.

**Columns:** orderID(int) , productID(int), amount(int).

**Keys:** orderID(FK), productID(FK)

**Trigger:** *sp\_autoDecrementStockAfterCompanyOrder*

**Company Shipment:** The shipment of the company that is created after the checkout is complete.

**Columns:** traceCode(int) (+Identity) , orderID(int), cost(int).

**Keys:** traceCode(PK), orderID(FK)

**Company Shopping Card:** The shopping card of the company created when the company representative is adding products.

**Columns:** companyID(int), productID(int), amount(int).

**Keys:** companyID(FK), productID(FK)

**Customer:** The table for the person who buys small amounts of products and regular customers.

**Columns:** customerID(int) (+Identity), fName (nvarchar(50)), lName (nvarchar(50)), phoneNumber(nvarchar(50)), email(nvarchar(50)), birthDate(smallerdatetime), age (int)(+Computed)

**Keys:** customerID (PK)

**Customer Bill:** Customer bills are produced when the customer checkouts the order.

**Columns:** billID(int) (+Identity), orderID(int), totalPrice(decimal(10,2)).

**Keys:** billID(PK), orderID(FK)

**Customer Order:** Customer orders are produced when the customer creates an order for a product to be produced.

**Columns:** orderID(int) (+Identity) , customerID(int), isCheckOutComplete(int).

**Keys:** orderID(PK), customerID(FK)

**Trigger:** *sp\_autoDecrementStockAfterCustomerOrder*

**Customer Product Of Order:** The product order that is the single element of the order.

**Columns:** orderID(int) , productID(int), amount(int).

**Keys:** orderID(FK), productID(FK)

**Customer Shipment:** The shipment of the customer that is created after the checkout is complete.

**Columns:** traceCode(int) (+Identity) , orderID(int), cost(int).

**Keys:** traceCode(PK), orderID(FK)

**Customer Shopping Card:** The shopping card of the customer created when the customer is adding products.

**Columns:** customerID(int), productID(int), amount(int).

**Keys:** customerID(FK), productID(FK)

**Product:** The shopping card of the customer created when the customer is adding products.

**Columns:** productID(int), categoryID(int), stockAmount(int), productName  
(nvarchar(50)), price (decimal(10,2)).

**Keys:** productID(PK), categoryID(FK)

## Views

### 1-) avgOrderPrice\_Istanbul

Display the average price of the all orders which was given in Istanbul.

```
-- Average price for orders in Istanbul
CREATE VIEW avgOrderPrice_Istanbul
as
select avg(cb.totalPrice) as averageCost
from Customer c inner join Address a on c.customerID = a.customerID inner join CustomerOrder co on c.customerID=co.customerID
inner join CustomerProductOfOrder cpo on cpo.orderID=co.orderID inner join CustomerBill cb on cb.orderID=cpo.orderID
where a.city = 'Istanbul'

select * from avgOrderPrice_Istanbul
```

119 %

Results Messages

	averageCost
1	22.444444

### 2-) mostOrderedProduct

Display the most ordered product

```
-- Most ordered product
CREATE VIEW mostOrderedProduct
as
select TOP 1 p.productID,p.productName , p.price, sum(cpo.amount) as TotalAmount
from Product p inner join CustomerProductOfOrder cpo on p.productID=cpo.productID
group by p.productID,p.productName , p.price
order by TotalAmount DESC

select * from mostOrderedProduct
```

119 %

Results Messages

	productID	productName	price	TotalAmount
1	23	Yilday Limon Kolonyası 1L	19.00	60

### 3-) orderCountByCity

Display the order counts according to the cities

```
CREATE VIEW orderCountByCity
as
select a.city , count(*) as totalOrderCity
from Customer c inner join Address a on c.customerID= a.customerID inner join CustomerOrder co on c.customerID=co.customerID
inner join CompanyProductOfOrder cpo on co.orderID = cpo.orderID
inner join CustomerBill cb on cb.orderID = cpo.orderID inner join CustomerShipment cs on cs.orderID=cb.orderID
group by a.city

select * from orderCountByCity
```

119 %

Results Messages

	city	totalOrderCity
1	Istanbul	18
2	Izmir	11

### 4-) orderCountByCategory

Display the order counts according to the categories

```
-- Number of total orders according to their categories
CREATE VIEW orderCountByCategory
as
select c.categoryName , sum(cpo.amount) as TotalCategorOrderAmont
from Product p inner join Category c on p.categoryID = c.categoryID
inner join CustomerProductOfOrder cpo on cpo.productID = p.productID
group by c.categoryName

select * from orderCountByCategory
```

119 %

Results Messages

	categoryName	TotalCategorOrderAmont
1	Cologne	129
2	Personal Care Product	18

## 5-) mostPurchasedCompany

Display the top 3 companies that spend most money

```
CREATE VIEW mostPurchasedCompany
as
select top 3 c.companyID , c.companyName, sum(cb.totalPrice) as totalAmountMoney
from Company c inner join CompanyOrder co on c.companyID=co.companyID inner join
CompanyProductOfOrder cpo on cpo.orderID=co.orderID inner join CompanyBill cb on cb.orderID = cpo.orderID
group by c.companyID , c.companyName
order by totalAmountMoney DESC

select * from mostPurchasedCompany
```

119 %

Results Messages

	companyID	companyName	totalAmountMoney
1	4	Gülseri Kozmetik	533500.00
2	5	Hobby	336000.00
3	9	Prolinex Kimya San. Ltd. Şti.	303500.00

## Triggers

### Trigger-1 : *sp\_autoDecrementStockAfterCustomerOrder*

**Definition:** This trigger updates the product stock amount when a customer orders a product.

**How it works:** By keeping track of what has been inserted to the customer's ordered product(s) list, trigger gets an inserted table. With the help of this table, trigger finds which product(s) have been purchased and decrements the stock amount of those products according to this situation.

### Screenshots:

```
sp_addProductAmount_RKIN\Berkin (52)*
exec sp_shipToCompany 12

Select * from CompanyShipment
select * from CompanyOrder
*/

--Trigger to Auto Decrement Stock when an order is placed by Customer
Create Trigger sp_autoDecrementStockAfterCustomerOrder
On CustomerProductOfOrder
After Insert
As
Begin
Update p
Set p.stockAmount=p.stockAmount-i.amount
From Product p inner join inserted i on p.productID=i.productID
End

--details of item which its id is 24
Select * From Product p where productID=24
--add item no 24 count of 1 to cart
exec sp_addIntoCustomerShoppingCart 13, 24, 1
--details of the shopping cart of the customer 13
Select * From CustomerShoppingCart Where customerID=13
--give an order
exec sp_addNewCustomerOrder 13
--order details of the customer 13
Select * From CustomerOrder Where customerID=13
--details of item which its id is 24
Select * From Product p where productID=24
```

100 %

Messages

Commands completed successfully.

Completion time: 2021-12-26T19:45:23.5335957+03:00

AS

Begin

Update p

Set p.stockAmount=p.stockAmount-i.amount

From Product p inner join inserted i on p.productID=i.productID

End

--details of item which its id is 24

Select \* From Product p where productID=24

--add item no 24 count of 1 to cart

exec sp\_addIntoCustomerShoppingCart 13, 24, 1

--details of the shopping cart of the customer 13

Select \* From CustomerShoppingCart Where customerID=13

--give an order

exec sp\_addNewCustomerOrder 13

--order details of the customer 13

Select \* From CustomerOrder Where customerID=13

--details of item which its id is 24

Select \* From Product p where productID=24

Results

productID	categoryID	productName	price	stockAmount	
24	1	Yılday Lavanta Kolonyası 1L	19.00	12500	total stock

customerID	productID	amount	
13	24	1	shopping cart info

orderID	customerID	isCheckoutComplete	
12	13	0	
13	13	1	
14	13	0	
15	13	0	
16	13	0	
17	13	0	order is placed

productID	categoryID	productName	price	stockAmount	
24	1	Yılday Lavanta Kolonyası 1L	19.00	12499	left stock



## Trigger-2 : *sp\_autoDecrementStockAfterCompanyOrder*

**Definition:** This trigger updates the product stock amount when a company orders a product.

**How it works:** By keeping track of what has been inserted to the company's ordered product(s) list, trigger gets an inserted table. With the help of this table, trigger finds which product(s) have been purchased and decrements the stock amount of those products according to this situation.

**Screenshots:**

```
sp_addProductAmou...RKIN\Berkin (52))* X
Select * from Product p where productID=24
*/
-----
--Trigger to Auto Decrement Stock when an order is placed by Company
Create Trigger sp_autoDecrementStockAfterCompanyOrder
On CompanyProductOfOrder
After Insert
As
Begin
Update p
Set p.stockAmount=p.stockAmount-i.amount
From Product p inner join inserted i on p.productID=i.productID
End

--details of item with ids are 24 and 30
Select * From Product p where productID=24 or productID=30
--add item no 24 and 30 count of 23 and 34 to cart
exec sp_addIntoCompanyShoppingCart 10, 24, 23
exec sp_addIntoCompanyShoppingCart 10, 30, 34
```

100 %

Messages

Commands completed successfully.

Completion time: 2021-12-26T19:53:46.3799504+03:00

```
sp_addProductAmou...RKIN\Berkin (52))* X
From Product p inner join inserted i on p.productID=i.productID
End

--details of item with ids are 24 and 30
Select * From Product p where productID=24 or productID=30
--add item no 24 and 30 count of 23 and 34 to cart
exec sp_addIntoCompanyShoppingCart 10, 24, 23
exec sp_addIntoCompanyShoppingCart 10, 30, 34
--details of the shopping cart of the company 10
Select * From CompanyShoppingCart where companyID=10
--give an order
exec sp_addNewCompanyOrder 10
--order details of the company 10
Select * From CompanyOrder where companyID=10
--details of item with ids are 24 and 30
Select * From Product p where productID=24 or productID=30
```

100 %

Results Messages

productID	categoryID	productName	price	stockAmount
24	1	Yilday Lavanta Kolonyasi 1L	19.00	12499
30	2	Yilday Vasein Pembe	14.00	5000

total stock

companyID	productID	amount
10	24	23
10	30	34

shopping cart info

orderID	companyID	isCheckoutComplete
13	10	0
14	10	0

order info

productID	categoryID	productName	price	stockAmount
24	1	Yilday Lavanta Kolonyasi 1L	19.00	12476
30	2	Yilday Vasein Pembe	14.00	4966

left stock

Query executed successfully. BERKIN (15.0 RTM) BERKIN\Berkin (52) Volkan Cosmetic Website 00:00:00

## Stored Procedures

### Stored Procedure-1: `sp_addProductAmount(productId, amountToAdd)`

**Definition:** Increments a product's stock amount by the given amount.

**How it works:** Works by updating the product table's stock amount.

**Screenshots:**

The first screenshot shows the SQL script for creating the stored procedure `sp_addProductAmount`. The script includes comments, parameter definitions, and the logic to update the stock amount of a product by a specified amount. It also includes a test execution command.

```
--STORED PROCEDURE TO INCREMENT PRODUCT AMOUNT
/*
Create Procedure sp_addProductAmount
@productID int,
@amountToAdd int
As
Begin
    Update p
    Set p.stockAmount = p.stockAmount+@amountToAdd
    From Product p
    Where p.productID=@productID
End
*/

/*
select * from Product

exec sp_addProductAmount 1, 10000

select * from Product
*/
```

The second screenshot shows the same script with the execution command `exec sp_addProductAmount 1, 10000` highlighted. Below the script, the results of the query are displayed in a table.

productID	categoryID	productName	price	stockAmount
1	1	Yilday Zeytin Kolonyasi Sprey	3.00	10000
2	1	Yilday Limon Kolonyasi Sprey	4.00	10000
3	1	Yilday Lavanta Kolonyasi Sprey	5.00	10000
4	1	Yilday Altin Damla Kolonyasi Sprey	5.00	10000
5	1	Yilday Tutun Kolonyasi Sprey	3.00	10000
6	1	Yilday Beyaz Zambak Kolonyasi Sprey	5.00	10000
7	1	Yilday Beyaz Incir Kolonyasi Sprey	2.00	10000
8	1	Yilday Zeytin Kolonyasi Mini	7.00	25000
9	1	Yilday Limon Kolonyasi Mini	8.00	25000

The results table shows that the stock amount for product 1 has been updated from 10000 to 20000.

## Stored Procedure-2: *sp\_removeProductAmount* (*productId*, *amountToAdd*)

**Definition:** Decrements a product's stock amount by the given amount.

**How it works:** Works by updating the product table's stock amount.

**Screenshots:**

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the SQL script for creating the stored procedure `sp_removeProductAmount`. The script includes a `DECLARE` statement for `@stock`, an `IF` statement to check if the amount to remove is less than the current stock, and an `UPDATE` statement to decrement the stock amount. The bottom pane shows the execution of the stored procedure with the command `exec sp_removeProductAmount 1, 10000`. Below the command, the results of the `select * from Product` query are displayed, showing the updated stock amount for product 1.

```
sp_addProductAmount.RKIN.Berkin (62)/* X
/*
--STORED PROCEDURE TO DECREMENT PRODUCT AMOUNT
/*
Create Procedure sp_removeProductAmount
@productId int,
@amountToRemove int
AS
Begin
Declare @stock int = (select p.stockAmount from Product p where p.productId=@productId)
If(@amountToRemove < @stock)
Begin
Update p
Set p.stockAmount = p.stockAmount - @amountToRemove
From Product p
where p.productId=@productId
End
Else
Begin
Update p
Set p.stockAmount = 0
From Product p
where p.productId=@productId
End
End
/*
110 %
Messages
Command completed successfully.
Completion time: 2021-12-27T18:21:45.1449080+03:00

110 %
Query executed successfully.
```

sp\_addProductAmount.RKIN.Berkin (62)/\* X

```
End
/*
/*
select * from Product
exec sp_removeProductAmount 1, 10000
select * from Product
/*
110 %
Results Messages
productID categoryID productName price stockAmount
1 1 Yilday Zeytin Kolonyasi Sprey 3.00 20000
2 2 Yilday Limon Kolonyasi Sprey 4.00 10000
3 3 Yilday Lavanta Kolonyasi Sprey 5.00 10000
4 4 Yilday Altin Damla Kolonyasi Sprey 5.00 10000
5 5 Yilday Tütün Kolonyasi Sprey 3.00 10000
6 6 Yilday Beyaz Zambak Kolonyasi Sprey 5.00 10000
7 7 Yilday Beyaz Incir Kolonyasi Sprey 2.00 10000
8 8 Yilday Zeytin Kolonyasi Mini 7.00 25000
9 9 Yilday Limon Kolonyasi Mini 8.00 25000
10 10 Yilday Lavanta Kolonyasi Mini 8.00 25000
11 11 Yilday Altin Damla Kolonyasi Mini 8.00 25000
12 12 Yilday Tütün Kolonyasi Mini 8.00 25000
13 13 Yilday Beyaz Zambak Kolonyasi Mini 7.00 25000
14 14 Yilday Beyaz Incir Kolonyasi Mini 7.00 25000
productID categoryID productName price stockAmount
1 1 Yilday Zeytin Kolonyasi Sprey 3.00 10000
2 2 Yilday Limon Kolonyasi Sprey 4.00 10000
3 3 Yilday Lavanta Kolonyasi Sprey 5.00 10000
4 4 Yilday Altin Damla Kolonyasi Sprey 5.00 10000
5 5 Yilday Tütün Kolonyasi Sprey 3.00 10000
6 6 Yilday Beyaz Zambak Kolonyasi ... 5.00 10000
7 7 Yilday Beyaz Incir Kolonyasi Sprey 2.00 10000
8 8 Yilday Zeytin Kolonyasi Mini 7.00 25000
9 9 Yilday Limon Kolonyasi Mini 8.00 25000
10 10 Yilday Lavanta Kolonyasi Mini 8.00 25000
11 11 Yilday Altin Damla Kolonyasi Mini 8.00 25000
12 12 Yilday Tütün Kolonyasi Mini 8.00 25000
13 13 Yilday Beyaz Zambak Kolonyasi ... 7.00 25000
14 14 Yilday Beyaz Incir Kolonyasi Mini 7.00 25000
15 15 Yilday Zeytin Kolonyasi Büyük 14.00 7500
16 16 Yilday Limon Kolonyasi Büyük 14.00 7500
17 17 Yilday Lavanta Kolonyasi Büyük 16.00 7500
Query executed successfully.
```

### Stored Procedure-3: *sp\_addNewProduct (category, name, price)*

**Definition:** Adds a new product.

**How it works:** Works by inserting given parametes into product table.

**Screenshots:**

The first screenshot shows the execution of a stored procedure in SQL Server Enterprise Manager. The command window displays the following SQL code:

```
sp_addProductAmou...RKIN\Berkir (62))" * X
exec sp_removeProductAmount 1, 10000

select * from Product
*/

--STORED PROCEDURE TO ADD NEW PRODUCT
/*
Create Procedure sp_addNewProduct
@category int,
@name nvarchar(50),
@price decimal(5, 2)
As
Begin
Insert into Product(categoryID, productName, price)
Values(@category, @name, @price)
End
*/
/*
```

The Messages pane shows the following output:

```
Commands completed successfully.
Completion time: 2021-12-26T18:23:12.8688003+03:00
```

The second screenshot shows the execution of the `sp_addNewProduct` stored procedure. The command window displays the following SQL code:

```
sp_addProductAmou...RKIN\Berkir (62))" * X
/*
select * from Product

exec sp_addNewProduct 1, 'Mavi Kolonya Tester', 10

select * from Product
*/
```

The Results pane shows the following table:

productID	categoryID	productName	price	stockAmount
18	1	Yilday Altin Damla Kolonyasi Buyuk	16.00	7500
19	1	Yilday Tutun Kolonyasi Buyuk	16.00	7500
20	1	Yilday Beyaz Zambak Kolonyasi Buy...	15.00	7500
21	1	Yilday Beyaz Incir Kolonyasi Buyuk	15.00	7500
22	1	Yilday Zeytin Kolonyasi 1L	18.00	12500
23	1	Yilday Limon Kolonyasi 1L	19.00	12500
24	1	Yilday Lavanta Kolonyasi 1L	19.00	12500
25	1	Yilday Altin Damla Kolonyasi 1L	19.00	12500
26	1	Yilday Tutun Kolonyasi 1L	18.00	12500
27	1	Yilday Beyaz Zambak Kolonyasi 1L	18.00	12500
28	1	Yilday Beyaz Incir Kolonyasi 1L	18.00	12500
29	2	Yilday Vasein Beyaz	17.00	5000
30	2	Yilday Vasein Pembe	14.00	5000
31	2	Yilday Aseton Pembe	15.00	7000
32	2	Yilday Aseton Mor	16.00	7000
33	2	Yilday Gulsuyu	17.00	15000
34	1	Fuar Zeytincegi Kolonyasi Sprey	7.00	15000
35	1	Fuar Zeytincegi Kolonyasi Buyuk	14.00	15000
36	1	Fuar Limon Kolonyasi Sprey	8.00	15000
37	1	Fuar Limon Kolonyasi Buyuk	16.00	15000
38	1	Mavi Kolonya Tester	10.00	0

#### Stored Procedure-4: *sp\_addIntoCustomerShoppingCart (customerID, productID, amount)*

**Definition:** Adds the given product by the given amount into the customer's shopping cart or updates existing amount.

**How it works:** Works by inserting to CustomerShoppingCart table or updating existing amount in the cart.

#### Screenshots:

The first screenshot shows the creation of the stored procedure `sp_addIntoCustomerShoppingCart`. The code is as follows:

```
--STORED PROCEDURE TO ADD INTO CUSTOMER SHOPPING CART
/*
Create Procedure sp_addIntoCustomerShoppingCart
@customerID int,
@productID int,
@amount int
As
Begin
    If (@productID in (Select productID From Product) and @amount <= (Select stockAmount From Product Where productID=@productID))
    Begin
        If (@productID not in (Select productID From CustomerShoppingCart Where customerID=@customerID))
        Begin
            Insert into CustomerShoppingCart
            Values(@customerID, @productID, @amount)
        End
        Else
        Begin
            Update CustomerShoppingCart
            Set amount=amount+@amount
            Where customerID=@customerID
        End
    End
End
Select * From CustomerShoppingCart Where customerID=13
```

The second screenshot shows the execution of the stored procedure. The code is as follows:

```
Begin
    If (@productID not in (Select productID From CustomerShoppingCart Where customerID=@customerID))
    Begin
        Insert into CustomerShoppingCart
        Values(@customerID, @productID, @amount)
    End
    Else
    Begin
        Update CustomerShoppingCart
        Set amount=amount+@amount
        Where customerID=@customerID
    End
End

Select * From CustomerShoppingCart Where customerID=13

exec sp_addIntoCustomerShoppingCart 13, 25, 12
exec sp_addIntoCustomerShoppingCart 13, 23, 15

Select * From CustomerShoppingCart Where customerID=13
*/
```

The Results tab shows the following data:

	customerID	productID	amount
1	13	25	12
2	13	23	15

### Stored Procedure-5: *sp\_addNewCustomerOrder (customerID)*

**Definition:** Creates a new order for the customer by using the values in customer's shopping cart. Empties the cart.

**How it works:** Works by inserting to customer order table and products of the order table. Deletes the cart's values.

#### Screenshots:

```
sp_addProductAmount.RKIN\Berkin (62)* X
```

```
/*
--STORED PROCEDURE TO ADD NEW CUSTOMER ORDER
*/
Create Procedure sp_addNewCustomerOrder
@customerID int
As
Begin
    Insert into CustomerOrder(customerID, isCheckoutComplete)
    Values(@customerID, 0)

    Declare @orderId int = SCOPE_IDENTITY()

    Insert into CustomerProductOfOrder
    Select @orderId, csc.productID, csc.amount
    From CustomerShoppingCart csc
    Where csc.customerID=@customerID

    Delete From CustomerShoppingCart Where customerID=@customerID

End
```

110 %

Messages

Commands completed successfully.

Completion time: 2021-12-26T18:29:57.2579011+03:00

sp\_addProductAmount.RKIN\Berkin (62)\* X

```
Select * from CustomerOrder
Select * from CustomerProductOfOrder cpo
exec sp_addNewCustomerOrder 13
Select * from CustomerOrder
Select * from CustomerProductOfOrder cpo
/*
```

62 %

	orderId	customerID	isCheckoutComplete
6	6	6	1
7	7	7	1
8	8	8	1
9	9	9	1
10	10	10	1
11	11	1	1
12	12	13	0
13	13	13	0

	orderId	productID	amount
14	7	8	1
15	8	30	4
16	9	31	2
17	10	33	12
18	11	1	31
19	12	23	24
20	13	23	36
21	13	25	12

	orderId	customerID	isCheckoutComplete
7	7	7	1
8	8	8	1
9	9	9	1
10	10	10	1
11	11	1	1
12	12	13	0
13	13	13	0
14	14	13	0

order info

	orderId	productID	amount
16	9	31	2
17	10	33	12
18	11	1	31
19	12	23	24
20	13	23	36
21	13	25	12
22	14	25	12
23	14	23	15

items in the order

### Stored Procedure-6: *sp\_addIntoCompanyShoppingCart* (*companyID*, *productId*, *amount*)

**Definition:** Adds the given product by the given amount into the company's shopping cart or updates existing amount.

**How it works:** Works by inserting to CompanyShoppingCart table or updating existing amount in the cart.

#### Screenshots:

The first screenshot shows the SQL script for creating the stored procedure `sp_addIntoCompanyShoppingCart`. The script includes a `CREATE PROCEDURE` statement with parameters `@companyID int`, `@productID int`, and `@amount int`. The logic uses an `IF` statement to check if the product is already in the cart for the given company. If not, it inserts a new record; otherwise, it updates the existing amount.

```
--STORED PROCEDURE TO ADD INTO COMPANY SHOPPING CART
Create Procedure sp_addIntoCompanyShoppingCart
@companyID int,
@productID int,
@amount int
As
Begin
    If (@productID in (Select productID From Product) and @amount <= (Select stockAmount From Product Where productID=@productID))
    Begin
        If (@productID not in (Select productID From CompanyShoppingCart Where companyID=@companyID))
        Begin
            Insert into CompanyShoppingCart
            Values(@companyID, @productID, @amount)
        End
        Else
        Begin
            Update CompanyShoppingCart
            Set amount=amount+@amount
            Where companyID=@companyID
        End
    End
End
```

The second screenshot shows the execution of the stored procedure. It includes the same script as above, followed by two `EXEC` statements to add items to the cart for company ID 6, and a final `SELECT` statement to view the cart contents.

```
sp_addProductAmou...RKIN\Berkin (62)* X
If (@productID in (Select productID From Product) and @amount <= (Select stockAmount From Product Where productID=@productID))
Begin
    If (@productID not in (Select productID From CompanyShoppingCart Where companyID=@companyID))
    Begin
        Insert into CompanyShoppingCart
        Values(@companyID, @productID, @amount)
    End
    Else
    Begin
        Update CompanyShoppingCart
        Set amount=amount+@amount
        Where companyID=@companyID
    End
End
End

Select * From CompanyShoppingCart Where companyID=6

exec sp_addIntoCompanyShoppingCart 6, 20, 12
exec sp_addIntoCompanyShoppingCart 6, 25, 54

Select * From CompanyShoppingCart Where companyID=6

*/
```

The Results tab shows the output of the `SELECT` statement, displaying two rows of data from the `CompanyShoppingCart` table for company ID 6.

	companyID	productID	amount
1	6	20	12
2	6	25	54

### Stored Procedure-7: *sp\_addNewCompanyOrder (companyID)*

**Definition:** Creates a new order for the company by using the values in company's shopping cart. Empties the cart.

**How it works:** Works by inserting to company order table and products of the order table. Deletes the cart's values.

#### Screenshots:

```
-- STORED PROCEDURE TO ADD NEW COMPANY ORDER

Create Procedure sp_addNewCompanyOrder
@companyID int
As
Begin
    Insert into CompanyOrder(companyID, isCheckoutComplete)
    Values(@companyID, 0)

    Declare @orderID int = SCOPE_IDENTITY()

    Insert into CompanyProductOfOrder
    Select @orderID, csc.productID, csc.amount
    From CompanyShoppingCart csc
    Where csc.companyID=@companyID

    Delete From CompanyShoppingCart Where companyID=@companyID

End

Select * from CompanyOrder
Select * from CompanyProductOfOrder cpo
```

Messages  
Commands completed successfully.  
Completion time: 2021-12-26T18:36:47.7012491+03:00

Results

orderID	companyID	isCheckoutComplete
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1
11	11	0

orderID	productID	amount
22	8	9
23	8	12
24	9	4
25	9	10
26	9	12
27	9	25
28	9	36
29	10	1

orderID	companyID	isCheckoutComplete
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1
11	11	0
12	6	0

order info

orderID	productID	amount
24	9	4
25	9	10
26	9	12
27	9	25
28	9	36
29	10	1
30	12	20
31	12	25

items in the order

Query executed successfully.



### Stored Procedure-8: *sp\_approveCustomerOrder (customerID)*

**Definition:** Completes the customer's checkout and creates a bill for the order.

**How it works:** Works by updating customer order table and inserting a new bill into the bill table. Calculates total cost for the products.

### Screenshots:

```
sp_addProductAmou...RKIN\Berkin (62) X
--STORED PROCEDURE TO APPROVE CUSTOMER ORDER
/*
Create Procedure sp_approveCustomerOrder
@orderID int
As
Begin
    Update co
    Set co.isCheckoutComplete=1
    From CustomerOrder co
    Where co.orderID=@orderID

    Insert into CustomerBill(orderID, totalPrice)
    Select @orderID, sum(dt.cost)
    From
    (Select cpo.amount * p.price as cost
    From CustomerProductOfOrder cpo inner join Product p on cpo.productID=p.productID
    Where cpo.orderID=@orderID) as dt
End
*/

110 %
Messages
Commands completed successfully.
Completion time: 2021-12-26T18:39:21.1879991+03:00
```

sp\_addProductAmou...RKIN\Berkin (62)\* X

```
Select * from CustomerOrder
Select * from CustomerBill
exec sp_approveCustomerOrder 13
Select * from CustomerOrder
Select * from CustomerBill
```

75 %

Results Messages

orderID	customerID	isCheckoutComplete
7	7	1
8	8	1
9	9	1
10	10	1
11	11	1
12	12	0
13	13	0
14	14	0

billID	orderID	totalPrice
4	4	13.00
5	5	5.00
6	6	2.00
7	7	7.00
8	8	56.00
9	9	30.00
10	10	204.00
11	11	93.00

orderID	customerID	isCheckoutComplete
7	7	1
8	8	1
9	9	1
10	10	1
11	11	1
12	12	0
13	13	1
14	14	0

billID	orderID	totalPrice
5	5	5.00
6	6	2.00
7	7	7.00
8	8	56.00
9	9	30.00
10	10	204.00
11	11	93.00
12	12	912.00

### Stored Procedure-9: *sp\_approveCompanyOrder (companyID)*

**Definition:** Completes the company's checkout and creates a bill for the order.

**How it works:** Works by updating company order table and inserting a new bill into the bill table. Calculates total cost for the products.

#### Screenshots:

```
sp_addProductAmou...RKIN\Berklin (62) X
--STORED PROCEDURE TO APPROVE COMPANY ORDER
/*
Create Procedure sp_approveCompanyOrder
@orderID int
As
Begin
    Update co
    Set co.isCheckoutComplete=1
    From CompanyOrder co
    Where co.orderID=@orderID

    Insert into CompanyBill(orderID, totalPrice)
    Select @orderID, sum(dt.cost)
    From
    (Select cpo.amount * p.price as cost
    From CompanyProductOfOrder cpo inner join Product p on cpo.productID=p.productID
    Where cpo.orderID=@orderID) as dt
End
*/
/*
110 %
Messages
Commands completed successfully.
Completion time: 2021-12-26T18:52:00.0672509+03:00
```

```
sp_addProductAmou...RKIN\Berklin (62)* X
Select * from CompanyOrder
Select * from CompanyBill
exec sp_approveCompanyOrder 12
Select * from CompanyOrder
Select * from CompanyBill
```

68 %

orderID	companyID	isCheckoutComplete
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1
11	11	0
12	12	1

billID	orderID	totalPrice
3	3	15000.00
4	4	106700.00
5	5	84000.00
6	6	24000.00
7	7	43000.00
8	8	27000.00
9	9	60700.00
10	31	93.00

orderID	companyID	isCheckoutComplete
5	5	1
6	6	1
7	7	1
8	8	1
9	9	1
10	10	1
11	11	0
12	12	1

billID	orderID	totalPrice
4	4	106700...
5	5	84000.00
6	6	24000.00
7	7	43000.00
8	8	27000.00
9	9	60700.00
10	31	93.00
11	32	1206.00

Query executed successfully.

**Stored Procedure-10:** *sp\_registerNewCustomer* (*fName*, *lName*, *phoneNumber*, *email*, *birthDate*)

**Definition:** Adds a new customer to the system.

**How it works:** Works by inserting personal details into Customer table and address details into address table.

**Screenshots:**

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the script for creating the stored procedure `sp_registerNewCustomer`. The script includes comments, parameter definitions, and the logic to insert customer and address details. The bottom pane shows the execution results, including a message indicating successful completion and a table of results.

```
sp_addProductAmou...RKIN\Berkin (62))* X
Select * From CompanyBill
*/
--STORED PROCEDURE TO REGISTER NEW CUSTOMER
Create Procedure sp_registerNewCustomer
@fName nvarchar(30),
@lName nvarchar(20),
@phoneNumber nvarchar(12),
@email nvarchar(50),
@birthdate smalldatetime,
@city nvarchar(50),
@postalcode int
As
Begin
Insert into Customer(fName, lName, phoneNumber, email, birthDate)
Values(@fName, @lName, @phoneNumber, @email, @birthdate)

Declare @customerID int = SCOPE_IDENTITY()

Insert into Address
Values(@customerID, @city, @postalcode)
End
/*
Select * From Customer
*/
```

Messages

Commands completed successfully.

Completion time: 2021-12-26T18:53:46.9481765+03:00

```
sp_addProductAmou...RKIN\Berkin (62))* X
Declare @customerID int = SCOPE_IDENTITY()
Insert into Address
Values(@customerID, @city, @postalcode)
End
/*
Select * From Customer c inner join Address a on a.customerID=c.customerID
exec sp_registerNewCustomer 'Ahmet', 'Gedik', '905305233421', 'ahmetgedik@gmail.com', '1995-12-14', 'Istanbul', 34321
Select * From Customer c inner join Address a on a.customerID=c.customerID
*/
```

Results

customerID	fName	lName	phoneNumber	email	birthDate	age	customerID	city	postalCode
5	Ozge	Tulum	905062239087	ozgetulum09@hotmail.com	1960-09-10 00:00:00	61	5	Istanbul	34098
6	Murat	Eksen	905064325055	murat.eksen@hotmail.com	1956-11-13 00:00:00	65	6	Izmir	35102
7	Mehmet	Turan	905065121212	turannmehmet@hotmail.com	1956-02-17 00:00:00	65	7	Izmir	35560
8	Ahmet	Taylan	905063212394	taylan_ahmet@hotmail.com	1993-10-28 00:00:00	28	8	Istanbul	34087
9	Ege	Kocdemir	90506387409	egekokdemir@icloud.com	1960-12-17 00:00:00	61	9	Istanbul	34056
10	Sila	Fenerci	905052312312	fenerisila@icloud.com	1956-01-25 00:00:00	65	10	Izmir	35210
11	Tunahan	Bay	905059876312	tunahanbay@hotmail.com	1993-09-17 00:00:00	28	11	Izmir	35457
12	Bihter	Barusoglu	905052131232	barus0bihter@hotmail.com	1960-03-17 00:00:00	61	12	Izmir	35803
13	Berkin	Polat	905305753102	bkrnpolat@gmail.com	2000-11-20 00:00:00	21	13	Istanbul	34821

customerID	fName	lName	phoneNumber	email	birthDate	age	customerID	city	postalCode
1	Mehmet	Etilen	905072231212	mehmetzebilen@gmail.com	1996-02-11 00:00:00	25	1	Izmir	35324
2	Ahmet	Celik	905075608960	ahmetcelik@icloud.com	1996-09-06 00:00:00	25	2	Izmir	35860
3	Aykut	Demir	905071232312	aykut_demir@gmail.com	1996-02-14 00:00:00	25	3	Istanbul	34323
4	Sena	Ayva	905062234589	mehmetzebilen@gmail.com	1993-03-28 00:00:00	28	4	Istanbul	34245
5	Ozge	Tulum	905062239087	ozgetulum09@hotmail.com	1960-09-10 00:00:00	61	5	Istanbul	34098
6	Murat	Eksen	905064325055	murat.eksen@hotmail.com	1956-11-13 00:00:00	65	6	Izmir	35102
7	Mehmet	Turan	905065121212	turannmehmet@hotmail.com	1956-02-17 00:00:00	65	7	Izmir	35560
8	Ahmet	Taylan	905063212394	taylan_ahmet@hotmail.com	1993-10-28 00:00:00	28	8	Istanbul	34087
9	Ege	Koc...	90506387409	egekokdemir@icloud.com	1960-12-17 00:00:00	61	9	Istanbul	34056
10	Sila	Fene...	905052312312	fenerisila@icloud.com	1956-01-25 00:00:00	65	10	Izmir	35210
11	Tuna...	Bay	905059876312	tunahanbay@hotmail.com	1993-09-17 00:00:00	28	11	Izmir	35457
12	Bihter	Bar...	905052131232	barus0bihter@hotmail.c...	1960-03-17 00:00:00	61	12	Izmir	35803
13	Berkin	Polat	905305753102	bkrnpolat@gmail.com	2000-11-20 00:00:00	21	13	Istanbul	34821
14	Ahmet	Gedik	905305233421	ahmetgedik@gmail.com	1995-12-14 00:00:00	26	14	Istanbul	34321

**Stored Procedure-11:** *sp\_registerNewCompany* (taxNo, companyName, taxOffice, address)

**Definition:** Adds a new company to the system.

**How it works:** Works by inserting given company details into Company table.

**Screenshots:**

The screenshot displays two windows from SQL Server Enterprise Manager. The top window, titled 'sp\_addProductAmou...RKIN\Berkin (62))', shows the definition of a stored procedure named 'sp\_registerNewCompany'. The code is as follows:

```
--STORED PROCEDURE TO REGISTER NEW COMPANY
/*
Create Procedure sp_registerNewCompany
@taxNo bigint,
@name nvarchar(50),
@taxOffice nvarchar(50),
@address nvarchar(50)
As
Begin
    Insert into Company(taxNo, companyName, taxOffice, address)
    Values(@taxNo, @name, @taxOffice, @address)
End
*/
```

The bottom window, also titled 'sp\_addProductAmou...RKIN\Berkin (62))', shows the execution of the stored procedure. The command executed is:

```
exec sp_registerNewCompany '7493756201', 'Protek Chemical', 'Tuzla', 'Tuzla,34940'
```

The 'Messages' pane below the command shows 'Commands completed successfully.' and 'Completion time: 2021-12-26T18:56:40.9683432+03:00'.

The 'Results' pane shows a table with 14 rows and 5 columns: companyID, taxNo, companyName, taxOffice, and address. The data is as follows:

companyID	taxNo	companyName	taxOffice	address
1	1064845672	Derin Kozmetik	Torbali	Izmir,35060
2	8365118171	Eq Products	Torbali	Izmir,35060
3	5248256336	Gokoğlu Kozmetik	Torbali	Izmir,35060
4	8715093295	Gülseel Kozmetik	Torbali	Izmir,35060
5	6250425958	Hobby	Torbali	Izmir,35060
6	3730424823	Kavlak Kozmetik	Ayvalık	Balkesir,10400
7	6312525334	Lotus	Torbali	Izmir,35060
8	1333782821	Mif Kozmetik	Torbali	Izmir,35060
9	7137114628	Prolinex Kimya San. Ltd. Şti.	Torbali	Izmir,35060
10	3627697305	Proxy Kozmetik	Torbali	Izmir,35060
11	2905378954	Sera Kozmetik	Torbali	Izmir,35060
12	1902564879	Since Kozmetik	Torbali	Izmir,35060
13	7384920463	Pama Kimya San.	Maltepe	Maltepe,34843
14	7493756201	Protek Chemical	Tuzla	Tuzla,34940

### Stored Procedure-12: *sp\_createNewCategory (categoryName)*

**Definition:** Adds a new category to the system.

**How it works:** Works by inserting given category name into category table.

**Screenshots:**

The first screenshot shows the execution of the stored procedure `sp_createNewCategory` with the parameter `'Lotion'`. The Messages pane indicates that the commands completed successfully.

The second screenshot shows the execution of the stored procedure `sp_createNewCategory` with the parameter `'Soap'`. The Results pane shows the contents of the `category` table, which now includes four rows:

categoryID	categoryName
1	Cologne
2	Personal Care Product
3	Lotion
4	Soap

### Stored Procedure-13: *sp\_shipToCustomer* (orderId)

**Definition:** Ships the billed-order to the customer. Calculates a shipment cost.

**How it works:** Works by inserting a new row into shipment table. SP links the given orderId to shipment details. Calculates a random cost depending on the total amount of the items.

#### Screenshots:

```
sp_addProductAmou...RKIN\Berkin (62))" * X
exec sp_createNewCategory 'Soap'
Select * From Category
*/
-----
--STORED PROCEDURE TO SHIP ORDER TO CUSTOMER
/*
Create Procedure sp_shipToCustomer
@orderId int
As
Begin
Insert into CustomerShipment(orderID, cost)
Select @orderId, sum(cpo.amount)*(RAND()*(10-5)+5)
From CustomerProductOfOrder cpo inner join CustomerOrder co on cpo.orderID=co.orderID
Where co.isCheckoutComplete=1 and cpo.orderID=@orderId
End
*/
/*
Select * from CustomerShipment
select * from CustomerOrder
exec sp_shipToCustomer 10
100 %
Messages
Commands completed successfully.
Completion time: 2021-12-26T18:59:49.0153463+03:00
```

```
sp_addProductAmou...RKIN\Berkin (62))" * X
Select * from CustomerShipment
select * from CustomerOrder
exec sp_shipToCustomer 13
Select * from CustomerShipment
select * from CustomerOrder
83 %
Results Messages
traceCode orderID cost
3 53 3 2
4 54 4 1
5 55 5 1
6 56 6 1
7 57 7 1
8 58 8 5
9 59 9 3
10 60 10 111
orderID customerID isCheckoutComplete
7 7 7 1
8 8 8 1
9 9 9 1
10 10 10 1
11 11 1 1
12 12 13 0
13 13 13 1
14 14 13 0
traceCode orderID cost
4 54 4 1
5 55 5 1
6 56 6 1
7 57 7 1
8 58 8 5
9 59 9 3
10 60 10 111
11 63 13 358
orderID customerID isCheckoutComplete
7 7 7 1
8 8 8 1
9 9 9 1
10 10 10 1
11 11 1 1
12 12 13 0
13 13 13 1
14 14 13 0
Query executed successfully.
```

### Stored Procedure-14: *sp\_shipToCompany* (orderId)

**Definition:** Ships the billed-order to the company. Calculates a shipment cost.

**How it works:** Works by inserting a new row into shipment table. SP links the given orderId to shipment details. Calculates a random cost depending on the total amount of the items.

**Screenshots:**

```
sp_addProductAmou...RKIN\Berkin (62))" -> X
select * from customerOrder
*/
--STORED PROCEDURE TO SHIP ORDER TO COMPANY
/*
Create Procedure sp_shipToCompany
@orderId int
As
Begin
Insert into CompanyShipment(orderID, cost)
Select @orderId, sum(cpo.amount)*(RAND()*(10-5)+5)
From CompanyProductOfOrder cpo inner join CompanyOrder co on cpo.orderID=co.orderID
where co.isCheckoutComplete=1 and cpo.orderID=@orderId
End
*/
/*
Select * from CompanyShipment
select * from CompanyOrder
exec sp_shipToCompany 10
Select * from CompanyShipment
select * from CompanyOrder
*/
83 %
Messages
Commands completed successfully.
Completion time: 2021-12-26T19:06:25.0479289+03:00
```

```
sp_addProductAmou...RKIN\Berkin (62))" -> X
Select * from CompanyShipment
select * from CompanyOrder
exec sp_shipToCompany 12
Select * from CompanyShipment
select * from CompanyOrder
83 %
Results Messages
traceCode orderID cost
3 4 3 150
4 5 4 1067
5 6 5 840
6 7 6 240
7 8 7 430
8 9 8 270
9 10 9 607
10 22 10 221
orderID companyID isCheckoutComplete
5 5 5 1
6 6 6 1
7 7 7 1
8 8 8 1
9 9 9 1
10 10 1 1
11 11 13 0
12 12 6 1
traceCode orderID cost
4 5 4 10...
5 6 5 840
6 7 6 240
7 8 7 430
8 9 8 270
9 10 9 607
10 22 10 221
11 23 12 556
orderID companyID isCheckoutComplete
5 5 5 1
6 6 6 1
7 7 7 1
8 8 8 1
9 9 9 1
10 10 1 1
11 11 13 0
12 12 6 1
Query executed successfully.
```