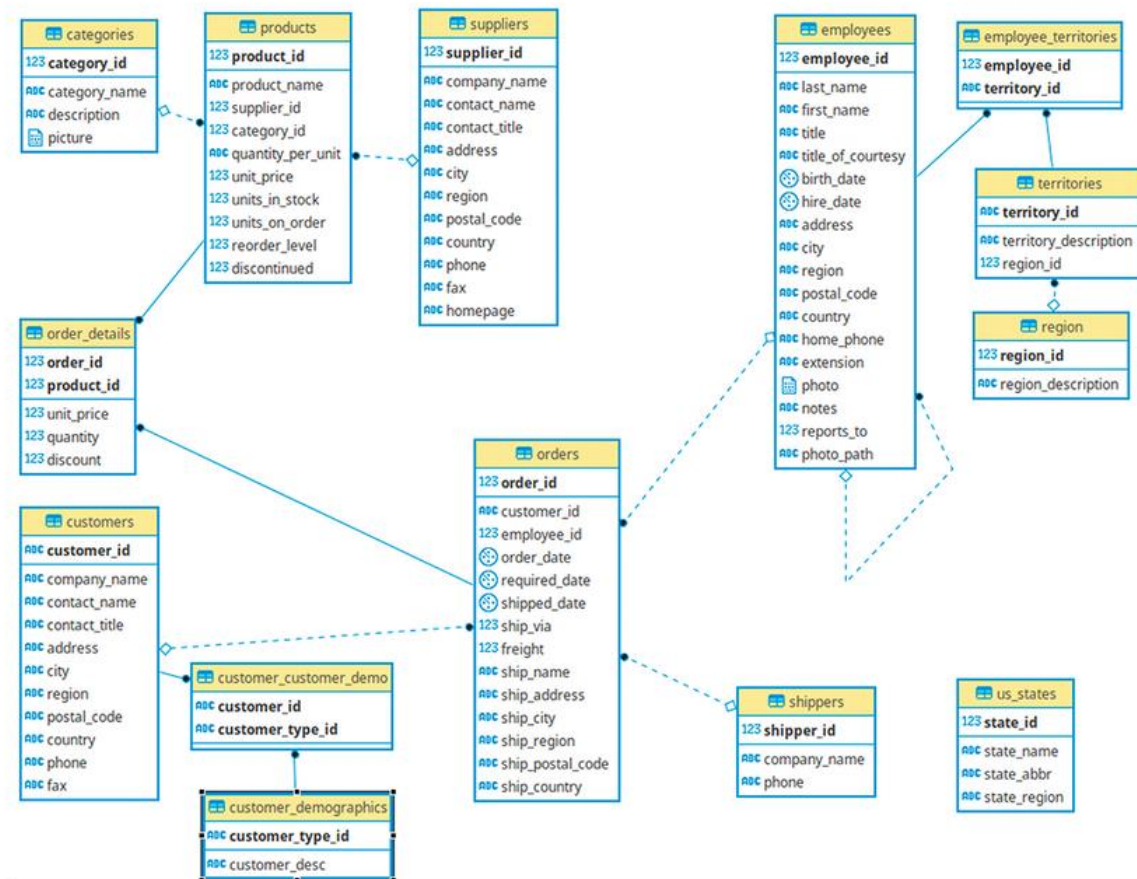


Tomasz Obuchowski - bazy danych 2 Projekt

Temat: Oprogramowanie składania zamówień w bazie Northwind w bazie danych postgresql używając frameworka .net Entity Framework.

Projekt zaczął się od znalezienia implementacji bazy Northwind w postgresql. Nie zaimplementowałem jej sam, gdyż nie to było istotą mojego projektu i byłoby marnowaniem czasu. Znalazłem w internecie i wykonałem kod sql do stworzenia i zapelnienia Northwind'a danymi w postgresql. Jego diagram wyglądał następująco:



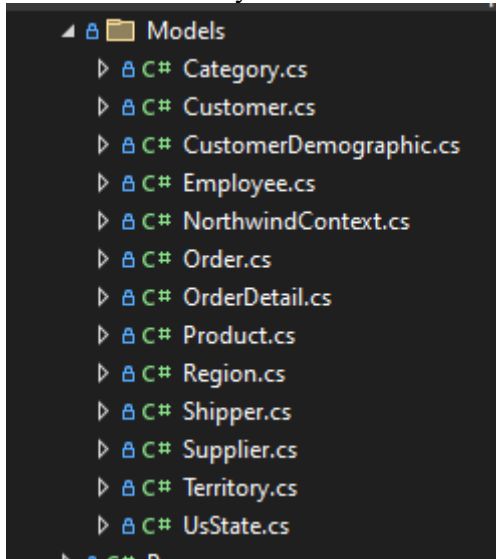
Jako serwera do hostowania bazy danych używam dockera.

Mając już bazę danych zacząłem moją aplikację od podejścia database first. Zainstalowałem wszystkie potrzebne narzędzia po czym w nuget manager console wpisałem komendę generującą klasy odpowiadające tabelą w bazie danych:

Scaffold-DbContext

"Host=localhost;Port=55432;Timeout=300;CommandTimeout=300;Database=northwind;Username=postgres;Password=postgres" Npgsql.EntityFrameworkCore.PostgreSQL -OutputDir Models

Komenda ta stworzyła folder Models który zawiera wszystkie klasy i context mojej bazy danych.



Stworzyła się również migracja i byłem już gotowy do pisania programu. Pomimo że c# jest językiem obiektowym to w Program.cs można pisać proceduralnie. Cały kod napisany przeze mnie będzie w Program.cs, ponieważ gdy mam wybór wolę podejście proceduralne, w szczególności w tak małym projekcie. Postanowiłem że moja aplikacja będzie prostym programem o tekstowym UI, za pomocą której można będzie wykonywać dodawanie zamówień.

Początkowo nie zrozumiałem zadania i dodałem obsługę dodawania supplierów shipperów oraz produktów, dlatego poza obsługą zamówień będzie jeszcze możliwość właśnie tego.

Główna pętla programu wygląda więc następująco:

```
while (true)
{
    string text = Console.ReadLine();
    // whether operation succeeded or not and why
    string answer="";
    switch (parseCmd(text))
    {
        case 0:
            continue;
        case 1:
            answer=addOrder();
            break;
        case 2:
            answer = deleteOrder();
            break;
        case 3:
            answer = modifyOrder();
            break;
        case 4:
            answer=addShipper();
            break;
        case 5:
            answer=addProduct();
            break;
        case 6:
            answer=addSupplier();
            break;
        case 7:
            Console.Clear();
            Console.WriteLine("Exiting...");
            return 0;
    }
    Console.Clear();
    writeUI(answer);
}
```

Czeka ona na stdin w konsoli i wywołuje odpowiednią funkcję o jej wartości. Komenda parseCmd wygląda następująco:

```

int parseCmd(string text){
    if (text == "add order" || text == "1")
    {
        return 1;
    }
    else if (text == "delete order" || text == "2")
    {
        return 2;
    }
    else if (text == "modify order" || text == "3")
    {
        return 3;
    }
    else if (text=="add shipper" || text == "4")
    {
        return 4;
    }
    else if (text == "add product" || text == "5")
    {
        return 5;
    }
    else if (text == "add supplier" || text == "6")
    {
        return 6;
    }
    else if (text == "exit" || text == "7")
    {
        return 7;
    }

    else return 0;
}

```

Funkcja writeUI(string) ma postać:

```

void writeUI(string answer){
    Console.WriteLine(answer);
    Console.WriteLine(" - Northwind - \n" +
        "Choose a command:\n" +
        "-add order\n-delete order\n-modify order\n-add shipper\n-add product\n-add supplier\n-exit");
}

```

1 reference

Wypisuje na ekran informacje o powodzeniu bądź porażce poprzedniej akcji (answer) i możliwe do wykonania komendy. Uruchomienie programu wygląda następująco:

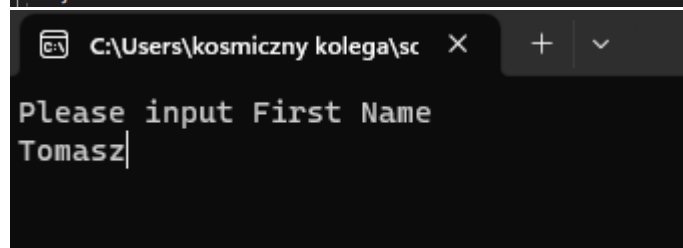
```
- Northwind -  
Choose a command:  
-add order  
-delete order  
-modify order  
-add shipper  
-add product  
-add supplier  
-exit  
|
```

Opis wszystkich funkcji wraz z kodem sql który wywołuje:

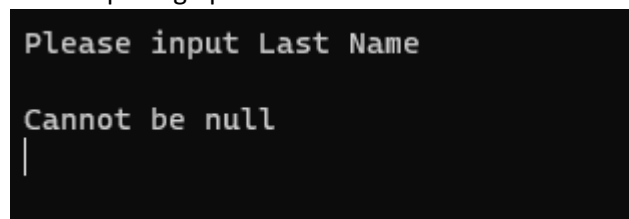
1. addOrder():

Najpierw pobiera od użytkownika informacje na temat pól w orders:

```
string addOrder()  
{  
    using var context = new NorthwindContext();  
    string answer = "";  
    string[] fieldLabels = { "First Name", "Last Name", "address", "Region", "PostalCode", "Country", "city" };  
    List<String> fields = new List<String>();  
  
    for (int i = 0; i < fieldLabels.Length; i++)  
    {  
        Console.Clear();  
        Console.WriteLine($"Please input {fieldLabels[i]}");  
        string temp = Console.ReadLine();  
        if (temp == "")  
        {  
            while (temp == "")  
            {  
                Console.WriteLine("Cannot be null");  
                temp = Console.ReadLine();  
            }  
            fields.Add(temp);  
        }  
        else  
        {  
            fields.Add(temp);  
        }  
    }  
}
```



W razie pustego pola:



Potem pobiera z bazy danych customera jeżeli istnieje, a w przeciwnym przypadku tworzy nowy obiekt Customer. W razie pobierania customera z bazy danych wykonywane jest następujące zapytanie:

```
debug: 21.06.2023 13:48:56.263 RelationalEventId.CommandExecuting[20100] (Microsoft.EntityFrameworkCore.Database.Command)  
Executing DbCommand [Parameters=[], CommandType='Text', CommandTimeout='300']  
SELECT c.customer_id, c.address, c.city, c.company_name, c.contact_name, c.contact_title, c.country, c.fax, c.phone, c.postal_code, c.region  
FROM customers AS c
```

Potem zaczyna się pętla odpowiedzialna za stworzenie listy zamówień i uzupełnienie jej zamówieniami sprecyzowanymi przez użytkownika:

```
bool ifEnd = false;
while (!ifEnd)
{
    context.ChangeTracker.Clear();
    List<OrderDetail> orders = new List<OrderDetail>();
    try
    {
        bool ifContinue = true;
        while (ifContinue)
        {
            string[] fieldLabels2 = { "what product you want to buy (id)", "how much of it" };
            List<String> fields2 = new List<string>();

            Product newProd = new Product();
            for (int i = 0; i < fieldLabels2.Length; i++)
            {
                Console.Clear();
                Console.WriteLine($"Please input {fieldLabels2[i]}");
                string temp = Console.ReadLine();
                if (temp == "")
                {
                    while (temp == "")
                    {
                        Console.WriteLine("Cannot be null");
                        temp = Console.ReadLine();
                    }
                    fields2.Add(temp);
                }
            }
        }
    }
}
```

, jeżeli produkty podane przez użytkownika istnieją to po podaniu ilości do dodania do zamówienia, jeżeli jest wystarczająco produktu dodaj do listy OrderDetail nowy OrderDetail

```
else if (i == 0)
{
    while (true)
    {
        newProd = context.Products.Where(x => x.ProductId == Convert.ToInt16(temp)).AsNoTracking().FirstOrDefault();
        if (newProd != null)
        {
            fields2.Add(temp);
            break;
        }
        Console.Clear();
        Console.WriteLine("There's no such Item, choose a correct one");
        temp = Console.ReadLine();
        while (temp == "")
        {
            Console.Clear();
            Console.WriteLine("Cannot be null");
            temp = Console.ReadLine();
        }
    }
}
else
{
    fields2.Add(temp);
}
if (i == 1 && newProd.UnitsInStock < Convert.ToInt16(fields2[1]))
{
    while (newProd.UnitsInStock < Convert.ToInt16(fields2[1]))
    {
        Console.Clear();
        Console.WriteLine($"There's not enough of this product please select less than {newProd.UnitsInStock} units or 0 to continue\n");
        temp = Console.ReadLine();
        fields2[1] = temp;
    }
}
if (fields2[1] != "0")
{
    orders.Add(new OrderDetail { ProductId = newProd.ProductId, Discount = 0, UnitPrice = (float)(newProd.UnitPrice * Convert.ToInt16(fields2[1])), Quantity = Convert.ToInt16(fields2[1])
}
```

Do bazy danych wysłane jest tutaj za każdym razem zapytanie o zwrócenie produktu, na razie bez śledzenia go w kontekście:

```
RelationalCommandCache.QueryExpression(
    Projection Mapping:
    EmptyProjectionMember -> Dictionary<IProperty, int> { [Property: Product.ProductId (short) Required PK AfterSave:Throw, 0], [Property: Product.CategoryId (short?) FK Index, 1],
    SELECT TOP(1) p.product_id, p.category_id, p.discontinued, p.product_name, p.quantity_per_unit, p.reorder_level, p.supplier_id, p.unit_price, p.units_in_stock, p.units_on_order
    FROM products AS p
    WHERE p.product_id = @__ToInt16_0,
```

Po czym w zależności od odpowiedzi użytkownika kontynuuj

```

        orders.Add(new OrderDetail { ProductId = newProd.ProductId, Quantity = 1 });
    }
    Console.Clear();
    Console.WriteLine("add another product to order? y/n\n");
    ifContinue = Console.ReadLine() == "y";
}

```

Po zakończeniu składania zamówień, pobiera dane z bazy danych i zmienia `unit_in_stock` zamówionych produktów oraz dodaje wszystko i próbuje zapisać zmiany.

Fracht jest dla ułatwienia obliczany jako jakaś wartość razy ilość obiektów, a pracownik i shipper są losowani.

```

double fr = 0;
foreach (OrderDetail order in orders)
{
    order.OrderId = newestid;
    foreach (Product product in context.Products) {
        if (product.ProductId == order.ProductId)
        {
            product.UnitsInStock = (short?)(product.UnitsInStock - order.Quantity);
            break;
        }
    }
    context.OrderDetails.Add(order);
    fr += order.Quantity * 0.51;
}

var rnd = new Random(DateTime.Now.Millisecond);
short ticks = (short)rnd.Next(1, 6);
short shipp = (short)rnd.Next(1, 3);
Order newOrder = new Order
{
    OrderId = newestid,
    CustomerId = id,
    EmployeeId = ticks,
    OrderDate = DateOnly.FromDateTime(DateTime.Now),
    ShippedDate = null,
    ShipVia = shipp,
    Freight = (float)fr,
    ShipName = firstName + " " + lastName,
    ShipAddress = address,
    ShipCity = city,
    ShipRegion = region,
    ShipPostalCode = postalCode,
    ShipCountry = country
};
if (!if) {
    ifEnd = true;
    context.Customers.Add(customer);
    ifEnd = false;
}
context.Orders.Add(newOrder);
context.SaveChanges();
answer = "Order added!";
ifEnd = true; break;
}

```

Wykonane zapytania to zczytanie products

```

SELECT p.product_id, p.category_id, p.discontinued, p.product_name, p.quantity_per_unit, p.reorder_level, p.supplier_id, p.unit_price, p.units_in_stock, p.units_on_order
FROM products AS p

```

I dodanie wszystkich zamówień, przykładowo:

```

INSERT INTO orders (order_id, customer_id, employee_id, freight, order_date, required_date, ship_address, ship_city, ship_country, ship_name, ship_postal_code, ship_region, ship_via, shipped_date)
VALUES (@p0, @p1, @p2, @p3, @p4, @p5, @p6, @p7, @p8, @p9, @p10, @p11, @p12, @p13);
UPDATE products SET units_in_stock = @p14
WHERE product_id = @p15;
INSERT INTO order_details (order_id, product_id, discount, quantity, unit_price)
VALUES (@p16, @p17, @p18, @p19, @p20);

```

Jeżeli customer został dodany to próbuje go również dodać.

Wszystko jest w bloku try catch żeby m.in. kontrolować `concurrencyException` i sytuacje w których klient którego próbowano dodać został dodany podczas działania programu i

kontrolować czy ilość units_in_stock jest ≥ 0 poprzez constraint dodany przeze mnie do bazy danych w migracji:

```
#nullable disable

namespace repl.Migrations
{
    /// <inheritdoc />
    public partial class AddConstraintToProd : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.Sql("ALTER TABLE products ADD CONSTRAINT CK_units_in_stock CHECK (units_in_stock >= 0)");
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
        }
    }
}
```

Block catch:

```
catch (DbUpdateConcurrencyException)
{
    answer = "couldn't add order due to concurrency exception";
    ifEnd = true;
}
catch (DbUpdateException)
{
    if (ifEnd)
    {
        answer = "couldn't add order due to unique constraints (customer added mid-order)";
    }
    else
    {
        Console.Clear();
        Console.WriteLine("sorry the products you were trying to purchase are no longer available try again? y/n\n");
        ifEnd = (Console.ReadLine() != "y");
        answer = "couldn't add order due to products >0 constraint";
    }
}
```

W przypadku gdy klient został dodany podczas wykonywania programu funkcja zostaje przerwana i żeby kontynuować trzeba zacząć od początku, ale w przypadku gdy produkt zostanie wykupiony przed saveChanges() w zależności od odpowiedzi użytkownika można spróbować jeszcze raz złożyć zamówienia bez potrzeby ponownego dodawania danych kontaktowych.

Przykładowy wynik funkcji gdy wszystkie dane są poprawne dla nowego klienta Klient Klient:

W Orders:

11095	KLKL	2	2023-06-21	[null]	[null]	1	18.36	Klient Klient	adress	City	Region	123123	Country
-------	------	---	------------	--------	--------	---	-------	---------------	--------	------	--------	--------	---------

W Order_details:

	order_id [PK] smallint	product_id [PK] smallint	unit_price real	quantity smallint	discount real
1	11095	6	300	12	0
2	11095	33	2.5	1	0
3	11095	61	655.5	23	0

ilość produktów na stanie również się zmieniła

Z:

	product_id [PK] smallint	product_name character varying	supplier_id smallint	category_id smallint	quantity_per_unit character varying	unit_price real	units_in_stock smallint	units_on_order smallint	reorder_level smallint	discontinued integer
1	6	Grandma's Boysenberry Spread	3	2	12 - 8 oz Jars	120	108	0	25	0
2	33	Geitost	15	4	500 g	112	111	0	20	0
3	61	Sirup d'érable	29	2	24 - 500 ml bottles	113	90	0	25	0

do:

units_in_stock smallint	ui sr
96	
110	
67	

Klient został dodany:

	customer_id [PK] character varying	company_name character varying	contact_name character varying	contact_title character varying	address character varying	city character varying	region character varying	postal_code character varying	country character varying	phone character varying	fax character varying
87	WARTH	Wartian Herkku	Pirkko Koskitalo	Accounting Manager	Torikatu 38	Oulu	[null]	90110	Finland	981-443655	981-443655
88	WELU	Wellington Importadora	Paula Parente	Sales Manager	Rua do Mercado, 12	Resende	SP	08737-363	Brazil	(14) 555-8122	[null]
89	WHITC	White Clover Markets	Karl Jablonski	Owner	305 - 14th Ave. S. Suite 38	Seattle	WA	98128	USA	(206) 555-4112	(206) 555-4115
90	WILMK	Wilman Kala	Matti Karttunen	Owner/Marketing Assistant	Keskuskatu 45	Helsinki	[null]	21240	Finland	90-224 8858	90-224 8858
91	WOLZA	Wolski Zajazd	Zbyszek Piestrzeniewicz	Owner	ul. Filitowa 68	Warszawa	[null]	01-012	Poland	(26) 642-7012	(26) 642-7012
92	TOCZ	Tomasz Czowiek	Tomasz Czowiek	[null]	here	Paris	here	11111	France	[null]	[null]
93	NITO	Nie Tomasz	Nie Tomasz	[null]	there	there	there	123123	there	[null]	[null]
94	XXXX	xxx xxx	xxx xxx	[null]	xxx	xxx	xxx	444444	xxx	[null]	[null]
95	ZZZZ	zzz zzz	zzz zzz	[null]	zzz	zzz	zzz	123451	zzz	[null]	[null]
96	KLKL	Klient Klient	Klient Klient	[null]	adress	City	Region	123123	Country	[null]	[null]

2. modifyOrder()

modyfikuje order o orderid podanym przez użytkownika

```
using NorthwindContext context = new NorthwindContext();
string answer = "";
Console.Clear();
Console.WriteLine("What order to modify?\n");
string temp = Console.ReadLine();
Order order = context.Orders.Where(x => x.OrderId == Convert.ToInt16(temp)).FirstOrDefault();
if (order != null)
{
    Console.Clear();
    Console.WriteLine("what to modify?\n" +
        "0 - employee id\n" +
        "1 - required date\n" +
        "2 - shipped date\n" +
        "3 - ship via\n" +
        "4 - freight\n" +
        "5 - ship name\n" +
        "6 - ship address\n" +
        "7 - ship city\n" +
        "8 - ship region\n" +
        "9 - postal code\n" +
        "10 - country\n" +
        "11 - quantity of products on order");
    temp = Console.ReadLine();
    try
    {
        switch (temp)
        {
            case "0":
                Console.Clear();
                Console.WriteLine("enter new employee id\n");
                order.EmployeeId = Convert.ToInt16(Console.ReadLine());
                context.SaveChanges();
                answer = "employee id changed";
                break;
        }
    }
}
```

```

what to modify?
0 - employee id
1 - required date
2 - shipped date
3 - ship via
4 - freight
5 - ship name
6 - ship address
7 - ship city
8 - ship region
9 - postal code
10 - country
11 - quantity of products on order

```

Wysłane zapytanie o order do bazy :

```

Executed DbCommand (2ms) [Parameters=[@__ToInt16_0=?' (DbType = Int16)], CommandType='Text', CommandTimeout='300']
SELECT o.order_id, o.customer_id, o.employee_id, o.freight, o.order_date, o.required_date, o.ship_address, o.ship_city, o.ship_country, o.ship_name, o.ship_postal_code, o.ship_region, o.ship_via, o.shipped_date
FROM orders AS o
WHERE o.order_id = @__ToInt16_0
LIMIT 1

```

Wszystkie case'y poza 11. wyglądają dokładnie tak samo z dokładnością do tego co zmieniają.

```

case "11":
    temp = "";
    while (temp.Length == 0)
    {
        Console.Clear();
        Console.WriteLine("which product?\n");
        temp = Console.ReadLine();
    }
    OrderDetail orderDetail1 = context.OrderDetails.Where(x => x.OrderId == order.OrderId && x.ProductId == Convert.ToInt16(temp)).FirstOrDefault();
    if (orderDetail1 != null)
    {
        temp = "";
        foreach (Product product in context.Products)
        {
            if (product.ProductId == orderDetail1.ProductId)
            {
                Console.Clear();
                Console.WriteLine("what should the quantity be?\n");
                temp = Console.ReadLine();
                product.UnitsInStock = (short?)(product.UnitsInStock + orderDetail1.Quantity - Convert.ToInt16(temp));
                orderDetail1.Quantity = Convert.ToInt16(temp);
                orderDetail1.UnitPrice = (float)(product.UnitPrice * orderDetail1.Quantity);
                answer = "order modified succesfully";
            }
        }
        context.SaveChanges();
    }
    else
    {
        answer = "Couldn't find your product in order";
    }
    break;

```

Case 11 zmienia ilość w podanym order_detail zmieniając również wartość units_in_stock w produktach. Cały switch jest w bloku try catch:

```

}
catch (DbUpdateConcurrencyException)
{
    answer = "couldn't modify order due to concurrency exception";
}
catch (DbUpdateException)
{
    answer = "couldn't modify order due to constraints exception";
}
}

```

W razie gdyby zmienić ilość produktów zamówionych w sposób który powodowałby zmniejszenie wartości units_in_stock na ≥ 0 zostanie wywołany dbupdateexception. Działanie dla dodanego wcześniej orderu dla modyfikacji ilości zamówionych produktów id 6 na 14:

	order_id [PK] smallint	product_id [PK] smallint	unit_price real	quantity smallint	discount real
1	11095	6	1680	14	0
2	11095	33	112	1	0
3	11095	61	2599	23	0

	product_id [PK] smallint	product_name character varying	supplier_id smallint	category_id smallint	quantity_per_unit character varying	unit_price real	units_in_stock smallint	units_on_order smallint	reorder_level smallint	discontinued integer
1	33	Geftost	15	4	500 g	112	110	0	20	0
2	61	Sirop d'érable	29	2	24 - 500 ml bottles	113	67	0	25	0
3	6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	120	94	0	25	0

3. deleteOrder()

usuwa order o podanym id jeżeli istnieje i nie został już dostarczony dodając do magazynu produkty wcześniej zamówione:

```
string deleteOrder()
{
    using var context = new NorthwindContext();
    string answer = "";
    Console.Clear();
    Console.WriteLine("What order to delete?\n");
    string temp = Console.ReadLine();
    Order order = context.Orders.Where(x => x.OrderId == Convert.ToInt16(temp)).FirstOrDefault();
    if (order == null)
    {
        answer = "no such order exists";
    }
    else if (order.ShippedDate == null)
    {
        try
        {
            List<OrderDetail> orderDetail = context.OrderDetails.Where(x => x.OrderId == order.OrderId).ToList();
            foreach (OrderDetail orderDetails in orderDetail)
            {
                Product prod = context.Products.Where(x => x.ProductId == orderDetails.ProductId).FirstOrDefault();
                prod.UnitsInStock = (short ?)(prod.UnitsInStock + orderDetails.Quantity);
                context.Remove(orderDetails);
            }
            context.Remove(order);
            context.SaveChanges();
            answer = "order deleted succesfully";
        }
        catch (DbUpdateConcurrencyException)
        {
            answer = "couldn't delete order due to concurrency exception";
        }
        catch (DbUpdateException)
        {
            answer = "couldn't delete order due to constraint exception (shouldn't be possible?)";
        }
    }
    else
    {
        answer = "the order was already sent so it can't be deleted";
    }
    return answer;
}
```

Dla usunięcia wcześniej dodanego orderu:

```
EXECUTE sp_executesql N'
SELECT o.order_id, o.customer_id, o.employee_id, o.freight, o.order_date, o.required_date, o.ship_address, o.ship_city, o.ship_country, o.ship_name, o.ship_postal_code, o.ship_region, o.ship_via, o.shipped_date
FROM orders AS o
WHERE o.order_id = @__ToInt16_0
LIMIT 1'
```

```

DELETE FROM order_details
WHERE order_id = @p0 AND product_id = @p1;
DELETE FROM order_details
WHERE order_id = @p2 AND product_id = @p3;
DELETE FROM order_details
WHERE order_id = @p4 AND product_id = @p5;
UPDATE products SET units_in_stock = @p6
WHERE product_id = @p7;
UPDATE products SET units_in_stock = @p8
WHERE product_id = @p9;
UPDATE products SET units_in_stock = @p10
WHERE product_id = @p11;
DELETE FROM orders
WHERE order_id = @p12;

```

Wynik:

order_id	product_id	unit_price	quantity	discount
[PK] smallint	[PK] smallint	real	smallint	real

Query	Query History
-------	---------------

```
1 select * from order_details where order_id =11095
```

order_id	customer_id	employee_id	order_date	required_date	shipped_date	ship_via	freight	ship_name	ship_address	ship_city	ship_region	ship_postal_code	ship_country
[PK] smallint	character varying	smallint	date	date	date	smallint	real	character varying	character varying	character varying	character varying	character varying	character varying

Query	Query History	Scratch Pad x
-------	---------------	---------------

```
1 select * from orders where order_id =11095
```

product_id	product_name	supplier_id	category_id	quantity_per_unit	unit_price	units_in_stock	units_on_order	reorder_level	discontinued
[PK] smallint	character varying	smallint	smallint	character varying	real	smallint	smallint	smallint	integer
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	120	108	0	25	0
33	Geitost	15	4	500 g	112	111	0	20	0
61	Sirop d'érable	29	2	24 - 500 ml bottles	113	90	0	25	0

Query	Query History	Scratch Pad x
-------	---------------	---------------

```
1 select * from products where product_id = 6 OR product_id = 33 or product_id =61
```

Reszta funkcji nie jest opisana gdyż nie były celem tego projektu.