OPPIMISTEHTÄVÄ 3

Kim Tiainen

5.4.2025

Projektin aloitin lataamalla tehtävän rungon GitHubista. Ennen varsinaisten tehtävien tekemistä minun täytyi kuitenkin varmistaa, että kehitysympäristö ja riippuvuudet olivat kunnossa. Asensin koneelle .NET SDK version 9 ja lisäsin projektin tarvitsemat Entity Framework Core –paketit:

- `Microsoft.EntityFrameworkCore`
- `Microsoft.EntityFrameworkCore.Tools`
- `Npgsql.EntityFrameworkCore.PostgreSQL`

Tietokantayhteyden luomiseksi käytin pgAdminia, jossa loin tietokannan nimeltä Oppimistehtävä 3. Yhdistin projektin tietokantaan käyttämällä seuraavaa komentoa, jolla generoitiin DbContext ja entiteettiluokat:

“dotnet ef dbcontext scaffold "Host=localhost;Port=5432;Database=Oppimistehtävä 3;Username=postgres;Password=******" Npgsql.EntityFrameworkCore.PostgreSQL -o Entities”

Tämän jälkeen sain projektin yhdistettyä tietokantaan onnistuneesti ja pääsin aloittamaan LINQ-kyselyiden toteutuksen.

Kyselyt	**Kuvaus**

Task01 Listaa kaikki asiakkaat

```
var customers = await _dbContext.Customers
// .AsNoTracking() // optional for read-only
.ToListAsync();

Console.WriteLine("=== TASK 01: List All Customers ===");

foreach (var c in customers)
{
    Console.WriteLine($"{c.FirstName} {c.LastName} - {c.Email}");
}
```

Task02 Näyttää jokaisen tilauksen tuotteiden lukumäärän

```
var orders = await _dbContext.Orders
```

```
                .Include(o => o.Customer)
                .Include(o => o.OrderItems)
                .Select(o => new
                {
                    OrderId = o.OrderId,
                    CustomerName = o.Customer.FirstName + " " +
o.Customer.LastName,
                    Status = o.OrderStatus,
                    ItemCount = o.OrderItems.Sum(oi => oi.Quantity)
                })
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== TASK 02: List Orders With Item Count ===");

            foreach (var order in orders)
            {
                Console.WriteLine($"Order #{order.OrderId} -
{order.CustomerName} - {order.Status} - Items: {order.ItemCount}");
            }
```

Task03 Listaa kaikki tuotteet hinnan mukaan laskevasti

```
            var products = await _dbContext.Products
                .OrderByDescending(p => p.Price)
                .Select(p => new { p.ProductName, p.Price })
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== Task 03: List Products By Descending Price
===");

            foreach (var product in products)
            {
                Console.WriteLine($"{product.ProductName} - {product.Price}
€");
            }
```

Task04 Näyttää "Pending"-tilaukset ja niiden kokonaisarvon

```
            var pendingOrders = await _dbContext.Orders
                .Where(o => o.OrderStatus == "Pending")
                .Include(o => o.Customer)
                .Include(o => o.OrderItems)
                .Select(o => new
                {
```

```
                    o.OrderId,
                    o.OrderDate,
                    CustomerName = o.Customer.FirstName + " " +
o.Customer.LastName,
                    TotalPrice = o.OrderItems.Sum(oi => (oi.UnitPrice *
oi.Quantity) - oi.Discount)
                })
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== Task 04: List Pending Orders With Total
Price ===");

            foreach (var order in pendingOrders)
            {
                Console.WriteLine($"Order #{order.OrderId} |
{order.CustomerName} | {order.OrderDate:d} | Total: {order.TotalPrice:0.00}
€");
            }
```

Task05 Laskee, montako tilausta kukin asiakas on tehnyt

```
            var orderCounts = await _dbContext.Orders
                .GroupBy(o => new { o.CustomerId, o.Customer.FirstName,
o.Customer.LastName })
                .Select(g => new
                {
                    CustomerName = g.Key.FirstName + " " + g.Key.LastName,
                    OrderCount = g.Count()
                })
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== Task 05: Order Count Per Customer ===");

            foreach (var c in orderCounts)
            {
                Console.WriteLine($"{c.CustomerName} - Orders:
{c.OrderCount}");
            }
```

Task06 Etsii 3 eniten rahaa käyttänyttä asiakasta

```
            var topCustomers = await _dbContext.Orders
                .Include(o => o.Customer)
                .Include(o => o.OrderItems)
```

```
                .GroupBy(o => new { o.CustomerId, o.Customer.FirstName,
o.Customer.LastName })
                .Select(g => new
                {
                    CustomerName = g.Key.FirstName + " " + g.Key.LastName,
                    TotalValue = g.Sum(o => o.OrderItems.Sum(oi =>
(oi.UnitPrice * oi.Quantity) - oi.Discount))
                })
                .OrderByDescending(c => c.TotalValue)
                .Take(3)
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== Task 06: Top 3 Customers By Order Value
===");

            foreach (var c in topCustomers)
            {
                Console.WriteLine($"{c.CustomerName} - Total Spent:
{c.TotalValue:0.00} €");
            }
```

Task07 Näyttää viimeisen 30 päivän tilaukset

```
            var recentDate = DateTime.Now.AddDays(-30);

            var recentOrders = await _dbContext.Orders
                .Include(o => o.Customer)
                .Where(o => o.OrderDate >= recentDate)
                .Select(o => new
                {
                    o.OrderId,
                    o.OrderDate,
                    CustomerName = o.Customer.FirstName + " " +
o.Customer.LastName
                })
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== Task 07: Recent Orders ===");

            foreach (var order in recentOrders)
            {
                Console.WriteLine($"Order #{order.OrderId} |
{order.OrderDate:d} | {order.CustomerName}");
            }
```

Task08 Tuotteiden myyntimäärät yhteensä

```
            var totalSold = await _dbContext.OrderItems
                .GroupBy(oi => new { oi.ProductId, oi.Product.ProductName })
                .Select(g => new
                {
                    g.Key.ProductName,
                    TotalQuantity = g.Sum(oi => oi.Quantity)
                })
                .OrderByDescending(p => p.TotalQuantity)
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== Task 08: Total Sold Per Product ===");

            foreach (var product in totalSold)
            {
                Console.WriteLine($"{product.ProductName} - Total Sold:
{product.TotalQuantity}");
            }
```

Task09 Tilaukset joissa on alennuksia

```
            var discountedOrders = await _dbContext.Orders
                .Include(o => o.Customer)
                .Include(o => o.OrderItems)
                    .ThenInclude(oi => oi.Product)
                .Where(o => o.OrderItems.Any(oi => oi.Discount > 0))
                .Select(o => new
                {
                    o.OrderId,
                    CustomerName = o.Customer.FirstName + " " +
o.Customer.LastName,
                    DiscountedProducts = o.OrderItems
                        .Where(oi => oi.Discount > 0)
                        .Select(oi => oi.Product.ProductName)
                        .ToList()
                })
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== Task 09: Discounted Orders ===");

            foreach (var order in discountedOrders)
            {
```

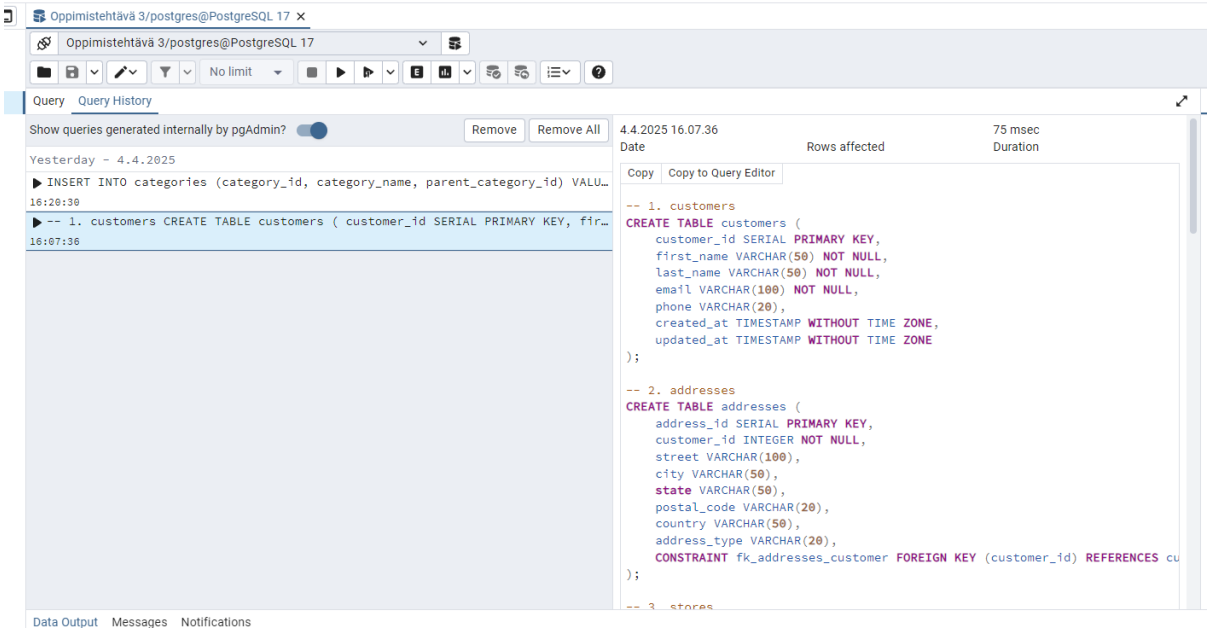
```
                var products = string.Join(", ", order.DiscountedProducts);
                Console.WriteLine($"Order #{order.OrderId} |
{order.CustomerName} | Discounted: {products}");
            }
```

Task10 Monivaiheinen kysely: esim. tuotteet "Electronics"-kategoriasta ja missä niitä varastoidaan

```
            var electronicsOrders = await _dbContext.Orders
                .Include(o => o.Customer)
                .Include(o => o.OrderItems)
                    .ThenInclude(oi => oi.Product)
                        .ThenInclude(p => p.Categories)
                .Where(o => o.OrderItems.Any(oi =>
                    oi.Product.Categories.Any(c => c.CategoryName ==
"Electronics")))
                .Select(o => new
                {
                    o.OrderId,
                    CustomerName = o.Customer.FirstName + " " +
o.Customer.LastName,
                    ElectronicsProducts = o.OrderItems
                        .Where(oi => oi.Product.Categories.Any(c =>
c.CategoryName == "Electronics"))
                        .Select(oi => oi.Product.ProductName)
                        .Distinct()
                        .ToList()
                })
                .ToListAsync();

            Console.WriteLine(" ");
            Console.WriteLine("=== Task 10: Advanced Query Example
(Electronics) ===");

            foreach (var order in electronicsOrders)
            {
                var products = string.Join(", ", order.ElectronicsProducts);
                Console.WriteLine($"Order #{order.OrderId} |
{order.CustomerName} | Electronics: {products}");
            }
```

Projektin haasteina oli yhdistää tietokanta projektiin, sillä ei ollut aikaisempaa kokemusta tämän tyylisestä projektista mutta internetin avulla ongelmista pääsin yli.

```
=== TASK 01: List All Customers ===
Ola Nordmann - ola.nordmann@example.com
Anna Svensson - anna.svensson@example.com

=== TASK 02: List Orders With Item Count ===
Order #1 - Ola Nordmann - Pending - Items: 2
Order #2 - Anna Svensson - Shipped - Items: 3

=== Task 03: List Products By Descending Price ===
Laptop Pro - 1299,00 ?
Smartphone X - 799,99 ?
Wireless Headphones - 199,50 ?

=== Task 04: List Pending Orders With Total Price ===
Order #1 | Ola Nordmann | 1.2.2025 | Total: 999,49 ?

=== Task 05: Order Count Per Customer ===
Anna Svensson - Orders: 1
Ola Nordmann - Orders: 1

=== Task 06: Top 3 Customers By Order Value ===
Anna Svensson - Total Spent: 1598,00 ?
Ola Nordmann - Total Spent: 999,49 ?

=== Task 07: Recent Orders ===

=== Task 08: Total Sold Per Product ===
Wireless Headphones - Total Sold: 3
Laptop Pro - Total Sold: 1
Smartphone X - Total Sold: 1

=== Task 09: Discounted Orders ===
Order #2 | Anna Svensson | Discounted: Laptop Pro

=== Task 10: Advanced Query Example (Electronics) ===
Order #1 | Ola Nordmann | Electronics: Smartphone X
Press any key to exit...
```


Oppimistehtävä 3/postgres@PostgreSQL 17
Oppimistehtävä 3/postgres@PostgreSQL 17
No limit
Query
Query History
Show queries generated internally by pgAdmin?
Remove
Remove All
Yesterday - 4.4.2025
INSERT INTO categories (category_id, category_name, parent_category_id) VALU…
16:20:30
-- 1. customers CREATE TABLE customers (customer_id SERIAL PRIMARY KEY, fir…
16:07:36
4.4.2025 16.07.36
Date
Rows affected
75 msec
Duration
Copy
Copy to Query Editor
-- 1. customers
CREATE TABLE customers (
customer_id SERIAL PRIMARY KEY,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL,
email VARCHAR(100) NOT NULL,
phone VARCHAR(20),
created_at TIMESTAMP WITHOUT TIME ZONE,
updated_at TIMESTAMP WITHOUT TIME ZONE
);
-- 2. addresses
CREATE TABLE addresses (
address_id SERIAL PRIMARY KEY,
customer_id INTEGER NOT NULL,
street VARCHAR(100),
city VARCHAR(50),
state VARCHAR(50),
postal_code VARCHAR(20),
country VARCHAR(50),
address_type VARCHAR(20),
CONSTRAINT fk_addresses_customer FOREIGN KEY (customer_id) REFERENCES cu
);
-- 3. stores
Data Output
Messages
Notifications