NÁSKOK
DÍKY
ZNALOSTEM

PROFINIT

Object Relational Mapping

Ing. Tomáš Vichta, Ing. Pavel Kopecký

1. března 2022

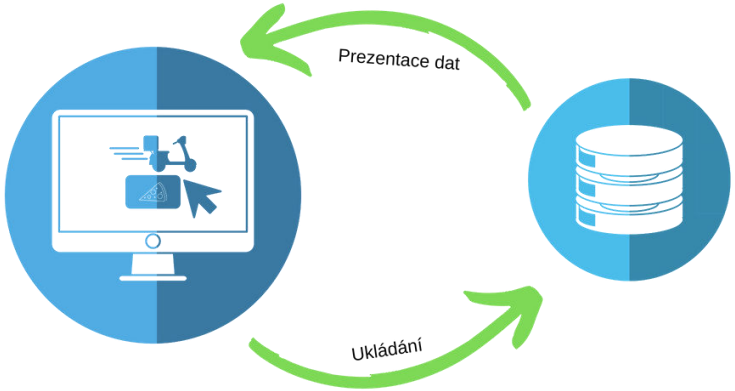
Osnova

1. Co je ORM a základní vlastnosti
2. Mapování modelů
3. ORM entita a stav
4. Dotazování
5. ORM v architektuře aplikace

PROFINIT

Co je ORM? A proč ho vůbec používat?

- ORM = **O**bject-**R**elational **M**apping
- Mapování mezi relačním světem (SQL) na objektový svět
  - Z dat v DB tabulkách vytváří objekty, a opačně
- Sleduje stav/změny objektů, aby věděl co jak uložit do DB
- Přístup k datům je sekundární task z pohledu programátora
  - Nesmí překážet a zdržovat od hlavního tasku - řešení business problémů



PROFINIT

Bez ORM

```
string connectionString =
    "Data Source=(local);Initial Catalog=Northwind;"
    + "Integrated Security=true";

// Provide the query string with a parameter placeholder.
string queryString =
    "SELECT ProductID, UnitPrice, ProductName from dbo.products "
        + "WHERE UnitPrice > @pricePoint "
        + "ORDER BY UnitPrice DESC;";

// Specify the parameter value.
int paramValue = 5;

// Create and open the connection in a using block. This
// ensures that all resources will be closed and disposed
// when the code exits.
using (SqlConnection connection =
    new SqlConnection(connectionString))
{
```

```
using (SqlConnection connection =
    new SqlConnection(connectionString))
{
    // Create the Command and Parameter objects.
    SqlCommand command = new SqlCommand(queryString, connection);
    command.Parameters.AddWithValue("@pricePoint", paramValue);

    // Open the connection in a try/catch block.
    // Create and execute the DataReader, writing the result
    // set to the console window.
    try
    {
        connection.Open();
        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Console.WriteLine("\t{0}\t{1}\t{2}",
                reader[0], reader[1], reader[2]);
        }
        reader.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.ReadLine();
}
```

PROFINIT

S ORM (MS Entity Framework)

```
using (var context = new SalesContext())
{
    foreach (var order in context.SalesOrderHeaders
            .Where(h => h.SalesOrderDetails.Any(d => d.UnitPrice > 2500)))
    {
        Console.WriteLine($"{order.SalesOrderNumber}: {order.Customer.Person.FullName}");
    }
}

Console.ReadKey();
```

PROFINIT

Lightweight ORM (Dapper)

```
using (var connection = new SqlConnection(conString))
{
    var orderDetails = connection.Query<OrderDetail>(
        "SELECT TOP 10 * FROM OrderDetails").ToList();
}
```

PROFINIT

Základní princip ORM

- Navrhnete třídy pro vaší business doménu
- Řeknete ORM, kde je DB a jak se mapují DB tabulky na třídy
- V ORM otevřete „session“ / „context“
- Pracujete s pseudo-kolekcemi objektů
  - Filtrujete, přidáváte, odstraňujete, upravujete
- Uložíte session => všechny změny se promítnou do DB
- ORM framework za vás
  - Spravuje DB connections
  - Vytváří a vykonává DB commandy
  - S výsledků query hydratuje objekty
  - Sleduje vámi provedené změny objektů (jen některé ORM frameworky)
  - Objekty převádí na SQL parametry a SQL příkazy
- Ne všechny ORM umí vše uvedené
  - Jen mapování: MyBatis, Dapper, ...
  - Plnotučné: Entity Framework, Hibernate, NHibernate

Příklad: Create - Read - Update - Delete v EF

```
using var session = new PeopleContext();
using var transaction = session.Database.BeginTransaction();
var alice = new Customer("Alice", "Sheldon");
session.Customers.Add(alice);

var bob = session.Customers.First(c => c.FirstName == "Bob");
bob.AssignEmail("bob@example.com");

var expiredCustomers = session.Customers.Where(c => c.IsExpired);
session.RemoveRange(expiredCustomers);
session.SaveChanges();

transaction.Commit();
```

PROFINIT

Výhody a nevýhody ORM

- Výhody
  - Práce s daty tak, jak je to v objektovém jazyku přirozené
  - Zapouzdření DB logiky
    - Chceme se věnovat doméně a neřešit (příliš) DB
  - Přepoužitelnost, testovatelnost, udržitelnost
    - S ORM lze snadněji zajistit, než přímo s nativním API či SQL
      - např. složité grafy objektů
- Nevýhody
  - Ztrácíte kontrolu nad SQL
  - Velká obecnost, musí fungovat spolehlivě => ztráta výkonu
  - Další složitá vrstva v aplikaci => ztráta výkonu
  - Objektový jazyk svádí k iterativnímu zpracování (for cykly)
    - SQL je mnohem efektivnější v hromadných operacích
      - UPDATE …. WHERE …
      - INSERT …. SELECT …
- Nevýhody lze vhodným použitím ORM minimalizovat
- ORM je vhodné pro 95% případů v běžných OLTP aplikacích

PROFINIT

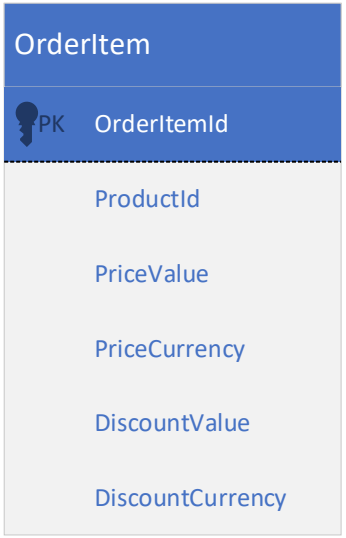
Historie přístupu k RDBMS z objektových jazyků

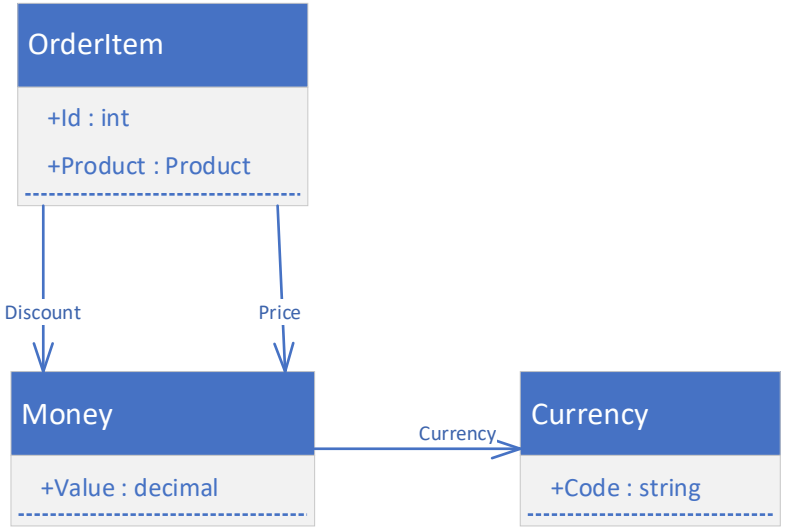
- 1994: TopLink (for SmallTalk)
- 1996: *JDK 1.0*, TopLink (for Java), ADO (ActiveX Data Objects)
- 1997: JDBC
- 2001: Hibernate
- 2002: *.NET 1.0*
- 2007: *.NET 3.5* – LINQ + první ORM
- 2008: Entity Framework

The Object-Relational Impedance Mismatch

Granuralita

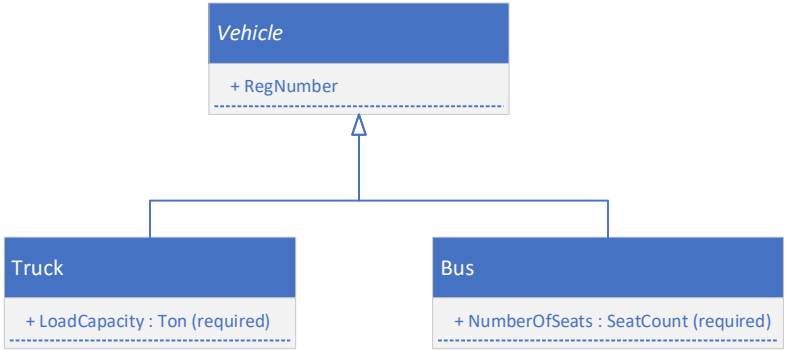
- 1 tabulka vs. více objektů





PROFINIT

Dědičnost



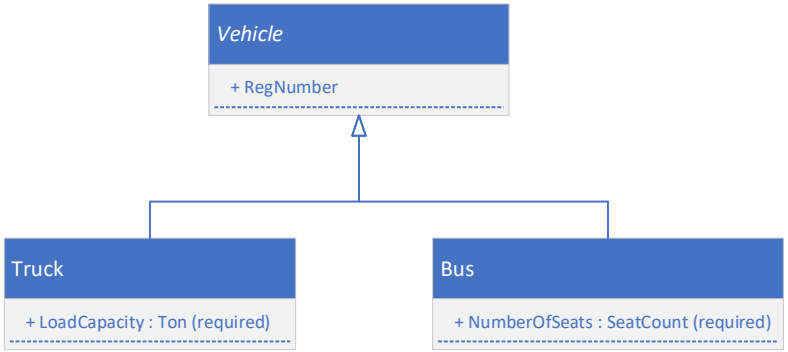
Vehicle	
PK Id	
RegNumber	NOT NULL
Type	NOT NULL
LoadCapacity	NULL
NumberOfSeats	NULL

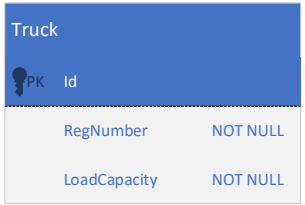
Table per hierarchy (parent only)

- Type = diskriminátor ("Truck", "Bus")
- Nutnost NULL
  - příliš volný relační model
  - složitější dotazování

PROFINIT

Dědičnost





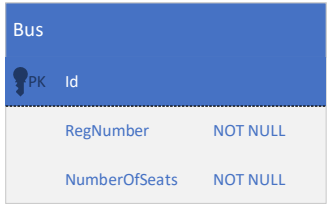
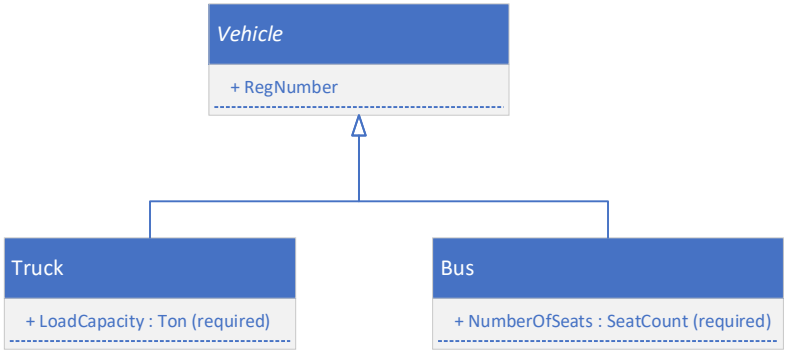


Table per concrete class (child only)

- Problém unikátnosti klíčů
- V DB modelu není vazba vyjádřena
- Složitější dotazy na společné atributy nadřazeného typu - UNION

PROFINIT

Dědičnost



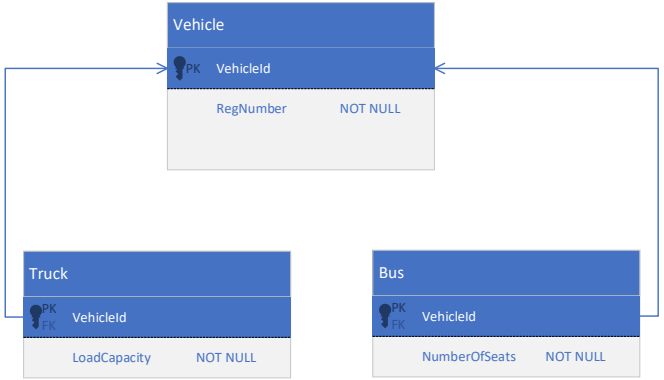


Table per type (parent child)

- Složité dotazy, výkonově nejhorší chování
- Relační model nedokáže zakázat vícenásobnou dědičnost

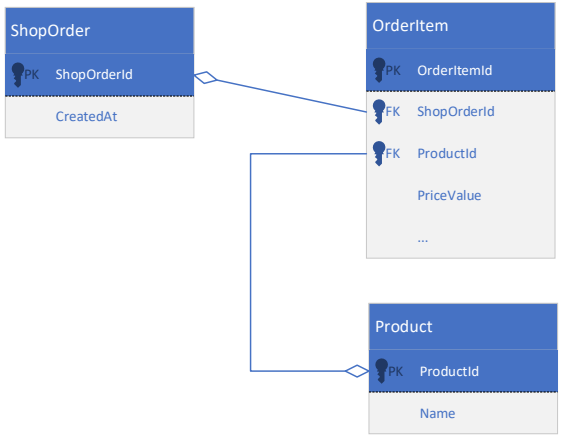
PROFINIT

Identita

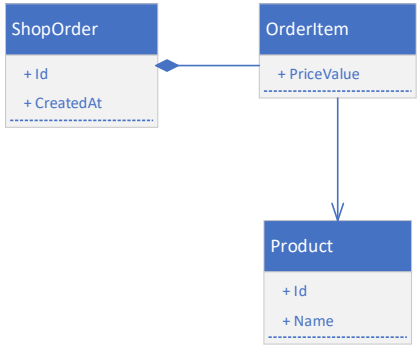
- Relační model
  - (Primární) klíč
- Objektový model
  - Reference na objekt
  - Identita: `a == b`
  - Hodnotová rovnost: `a.equals(b)`

Navigace

- Relační model
  - Cizí klíče
  - Nativně obousměrné vazby



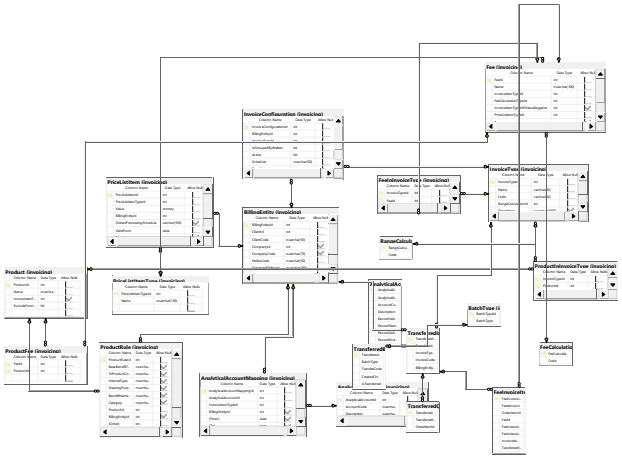
- Objektový model
  - Jednosměrné reference (možné a výhodné)



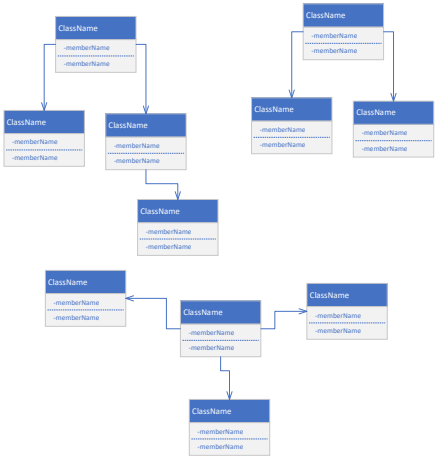
PROFINIT

Navigace

- Relační model
  - Obousměrný "god model"



- Objektový model
  - Lze dělit do izolovaných transakčně konzistentních agregátů
  - Zapouzdření, loose coupling, Domain driven design



PROFINIT

Plochá vs. hierarchická struktura

- Relační model
  - Flattening queries (JOIN)
  - Možná projekce vybraných sloupců

```
SELECT c.LastName, i.Price, i.Quantity
FROM customer c
JOIN shopOrder o ON
o.customerId=c.customerId
JOIN orderItem i ON i.orderId=o.orderId
```

- Objektový model
  - Primárně graf celých objektů
  - Průchod grafem

```
foreach(var customer in customers)
{
    foreach(var order in customer.Orders)
    {
        foreach(var item in order.OrderItems)
        {
            var shoppingInfo = new CustomerShoppingInfo(
                customer.LastName,
                item.Price,
                item.Quantity);
        }
    }
}
```

PROFINIT

Mapování modelů

Mapování

- Vlastní propojení Objektového a Relačního modelu
  - nejčastěji v imperativním kódu nebo xml
  - velké odlišnosti a možnosti mezi frameworky
  - Tabulky, views, procedury, funkce, sekvence, ...

Příklady definice mapování

```
vendorEntity.HasKey(vendor => vendor.Id);
vendorEntity.Property(vendor => vendor.Id).UseHiLo("VendorIdSequence");
vendorEntity.Property(vendor => vendor.CompanyName).IsRequired().HasMaxLength(30);
var emailConverter = new ValueConverter<EmailAddress, string>(
    email => email.Value,
    value => EmailAddress.Create(value).Value);
vendorEntity.Property(v => v.ContactEmail).HasConversion(emailConverter);

vendorEntity.Property(v => v.Version).IsConcurrencyToken().ValueGeneratedOnAddOrUpdate();
```

```
public class Product
{
    [Key]
    [Column("ProductId")]
    public int Id { get; set; }

    [MaxLength(30)]
    [Required]
    public string Name { get; set; }
}
```

PROFINIT

Příklady definice mapování

```
<hibernate-mapping>
  <class name="Vendor" table="tblVENDOR">
    <id name="id" type="int" column="VendorId">
      <generator class="native"/>
    </id>
    <property name="CompanyName" column="name" type="string"/>
  </class>
</hibernate-mapping>
```

```
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select id, subject, author from Blog where id = #{id}
  </select>
</mapper>
```

PROFINIT

ORM entita a stav

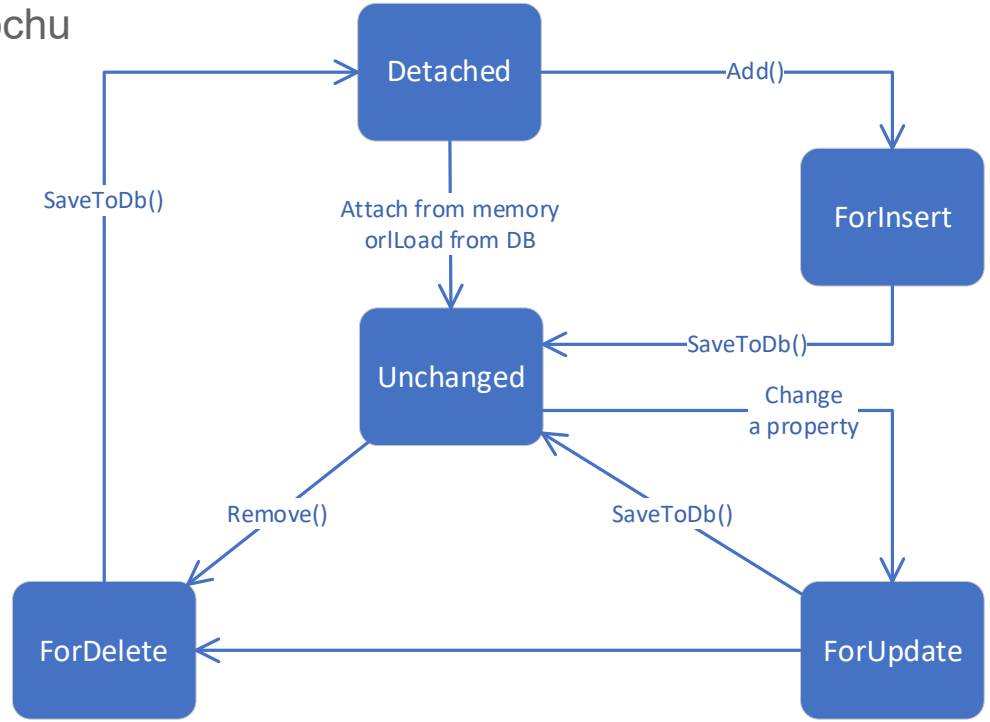
Příklad: Create - Read - Update - Delete v EF

```
using var session = new PeopleContext();

var alice = new Customer("Alice", "Sheldon");
session.Customers.Add(alice);

var bob = session.Customers.First(c => c.FirstName == "Bob");
bob.AssignEmail("bob@example.com");

var expiredCustomers = session.Customers.Where(c => c.IsExpired);
session.RemoveRange(expiredCustomers);

session.SaveChanges();
```

PROFINIT

Stav ORM entity v session

- Příklad, v každém ORM trochu jiné
- Sleduje stav objektu
- Některé ORM sledují i stav properties



PROFINIT

Připojený režim

- Jedna session po celý život aplikace!
- Načtení entity z DB → prezentace na UI → úprava entit → uložení
  - I opakovaně, ale jednou načtená entita typicky zůstane v session
- Možné u jednoduchých aplikací, které neopouštějí scope
  - Konzolová aplikace, Windows služba, tlustý klient
- I tam lze použít jen pro malé scénáře!
- Riziko výkonových problémů - mnoho trackovaných entit (některé ORM)
- Možný problém s nevalidní entitou, pokud selže uložení do DB

Odpojený režim

- Nová session pro každou unit of work (operace, web request)
- Typický režim pro většinu aplikací

PROFINIT

Problémy stavu entit

- Každý ORM framework jiné chování, nutno zohlednit v návrhu aplikace
- Graf objektů: Vložím řádek objednávky. Má se vložit i nová objednávka (ForInsert), nebo se jedná o existující (Unchanged)?
- Stejný objekt připojím do session opakovaně jako nový
- Během zpracování mojí session se změní řádek v DB
- Dojde k chybě při ukládání, stav objektu v paměti se liší od DB
- Rollback transakce - obnoví se i objekt?
- Caching
- Views nad tabulkou - různé entity ze stejného zdroje

Příklad nevhodného návrhu práce se stavem v EF

```
// Example: order 123 has many order items. Load order with all items.
var order = LoadOrder(123);

// Apply discount to 1st item
var firstOrderItem = order.Items.First();
firstOrderItem.Discount = new Money(100, Currency.Czk);

// What really will be updated in DB?
UpdateOrderItem(firstOrderItem);

public ShopOrder LoadOrder(int orderId)
{
    using var session = new ShopContext();
    var order = session.Orders.Include(o=>o.Items).Single(o => o.Id == orderId);
    return order;
}

public void UpdateOrderItem(OrderItem orderItem)
{
    using var session = new ShopContext();
    session.Update(orderItem);
    session.SaveChanges();
}
```

Transakční zpracování a concurrency

- DB transakce
  - Často samostatný roundtrip - zbytečné prodloužení transakce

```
using var tran = dbContext.Database.BeginTransaction(IsolationLevel.ReadCommitted);
// ...
tran.Commit();
```

- Distribuované transakce
  - Riziko extrémních výkonových problémů - 2fázový commit je pomalý
- Optimistic concurrency
  - Většina ORM má vestavěnou podporu, obvykle ale jen na 1 property v entitě
  - I to je často akceptovatelné řešení

```
UPDATE ... WHERE Id=@Id AND ConcurrencyColumn=@OrigValue
SELECT @@ROWCOUNT
```

- Nutno řešit deadlocky, serializable konflikty, connection resiliency
  - Chování závislé na DB serveru

PROFINIT

Dotazování

Dotazování

- Celý objektový graf entit
- Projekce do pomocných data transfer objektů
- Je třeba myslet na výkon - objem dat, složitost dotazu
- Která část kódu se překládá do SQL a běží v databázi?

```
dbContext.Customers
    .Where(c => c.LastName == "novák")
    .ToList()
    .OrderBy(c => c.LastName);
```

PROFINIT

Problémy dotazování: hluboký entitní graf

```
var customers = dbContext.Customers
        .Include(c => c.Orders)
        .ThenInclude(o => o.OrderItems)
        .ThenInclude(i => i.Product)
        .ToList();
```

```
SELECT *
FROM customer c
LEFT JOIN shopOrder o ON o.customerId=c.customerId
LEFT JOIN orderItem i ON i.orderId=o.orderId
LEFT JOIN product p ON p.ProductId=i.ProductId
```

Customer columns	ShopOrder columns	OrderItem columns	Product columns
Customer 1	Order 1	Item 1	Product 1
Customer 1	Order 1	Item 2	Product 2
Customer 1	Order 1	Item 3	Product 3
Customer 1	Order 1	Item 4	Product 4
Customer 1	Order 1	Item 5	Product 5
Customer 1	Order 2	Item 6	Product 2
Customer 1	Order 2	Item 7	Product 3
Customer 1	Order 2	Item 8	Product 10
Customer 2	Order 10	Item 9	Product 11
Customer 2	Order 10	Item 10	Product 2
Customer 2	Order 10	Item 11	Product 3
Customer 2	Order 10	Item 12	Product 40
Customer 2	Order 10	Item 13	Product 50
Customer 2	Order 11	Item 14	Product 2
Customer 2	Order 11	Item 15	Product 3
Customer 2	Order 11	Item 16	Product 10

Duplicitní údaj

PROFINIT

Problémy dotazování: Zpracování ve smyčklách

```
foreach (var customer in dbContext.Customers)
{
    var orders = dbContext.Orders.Where(o => o.CustomerId == customer.Id);
    foreach (var order in customer.Orders)
    {
        var items = dbContext.OrderItems.Where(i => i.OrderId == order.Id);
        foreach (var item in order.OrderItems)
        {
            // ...
```

PROFINIT

NULL, case sensitivity, accent sensitivity

- ne-porovnatelnost null?

```
dbContext.Customers.Where(c => c.ManagerId == null);

dbContext.Customers.Where(c => c.ManagerId != 123);
```

  - Odlišná NULL logika
    - C# (null == null): true, (null != null): false
    - SQL: (NULL == NULL): NULL, (NULL <> NULL): NULL

- databáze často case/accent-insensitive

```
dbContext.Customers.Where(c => c.LastName == "novák");
```

PROFINIT

Lazy loading

- Reference nebo kolekce se načte, až když na ní v kódu přistoupím
- Některé ORM umí lazy loadovat i jednotlivé properties
- Podporuje většina ORM, často zapnuto implicitně
- Na první dojem jednodušší na použití

```
var customers = dbContext.Customers.ToList();
foreach (var customer in customers)
{
    foreach (var order in customer.Orders)
    {
        foreach (var item in order.OrderItems)
        {
            // ...
        }
    }
}
```

- Může způsobit extrémně mnoho roundtripů do DB
- V okamžiku přístupu musí být dostupné spojení do DB
- Chyba programátora je v kódu špatně odhalitelná
- Oprávněné použití jen ve výjimečných případech

PROFINIT

Eager loading

- Musím explicitně definovat které části objektového grafu chci načíst z DB
- To ale téměř vždy vím

```
var customers = dbContext.Customers
                .Include(c => c.Orders)
                .ThenInclude(o => o.OrderItems)
                .ToList();
foreach (var customer in customers)
{
    foreach (var order in customer.Orders)
    {
        foreach (var item in order.OrderItems)
        {
            // ...
        }
    }
}
```

PROFINIT

Výkonové vlastnosti

- Relační DB navrženy pro
  - velký objem dat
  - hromadné (dávkové) zpracování (UPDATE ... WHERE ...)
- OOP: základní jednotka je jeden objekt
- Složité dotazy: SQL v kódu = obejití mapovací vrstvy
- ORM nevhodné pro reporting, analytiku, ETL

Závěr

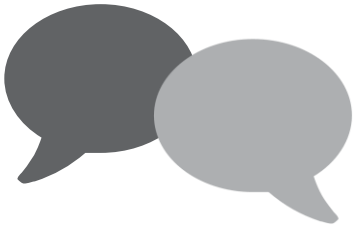
- ORM je velmi užitečné, ale není vhodné na vše
- Nutné kompromisy v objektovém a/nebo relačním modelu
- Na větší databázi nutná znalost SQL a optimalizace
- Žádné ORM nemůže být plně platformě nezávislé

PROFINIT

Co si zapamatovat

- Které 2 základní úlohy většina ORM zajišťuje
- Jaké jsou principiální rozdíly mezi objektovým a relačním modelem
- Co je lazy loading a v čem je problematický
- Pro co je ORM vhodné a pro co není
- K čemu ORM využívá stav entity a jak souvisí s aktuálními daty v DB

PROFINIT



Diskuze

Děkujeme
za pozornost

PROFINIT
NÁSKOK DÍKY ZNALOSTEM

Profinit EU, s.r.o.
Tychonova 2, 160 00 Praha 6 | Telefon + 420 224 316 016

Web
www.profinit.eu

LinkedIn
linkedin.com/company/profinit

Twitter
twitter.com/Profinit_EU

Facebook
facebook.com/Profinit.EU

Youtube
Profinit EU