ORM - Objektové relační mapování

Kryštof Jakůbek 2024

ORM

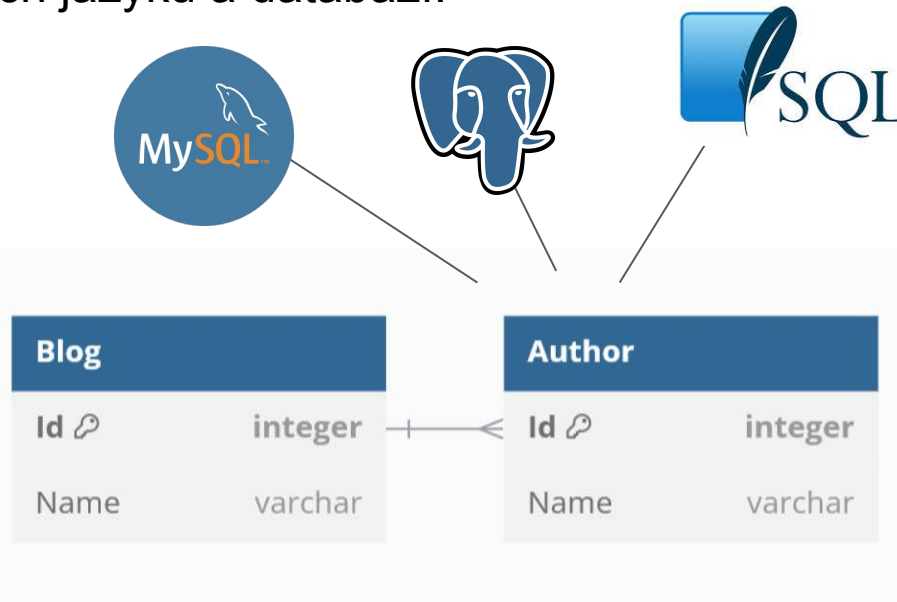
Systém pro propojení objektově orientovaných jazyků a databází.

Mapuje objekty na databázové objekty

```
public class Blog{
    public int Id;
    public string Name;
}

public class Author {
    public int Id;
    public string Name;
    public List<Blog> Blogs;
}
```

ORM



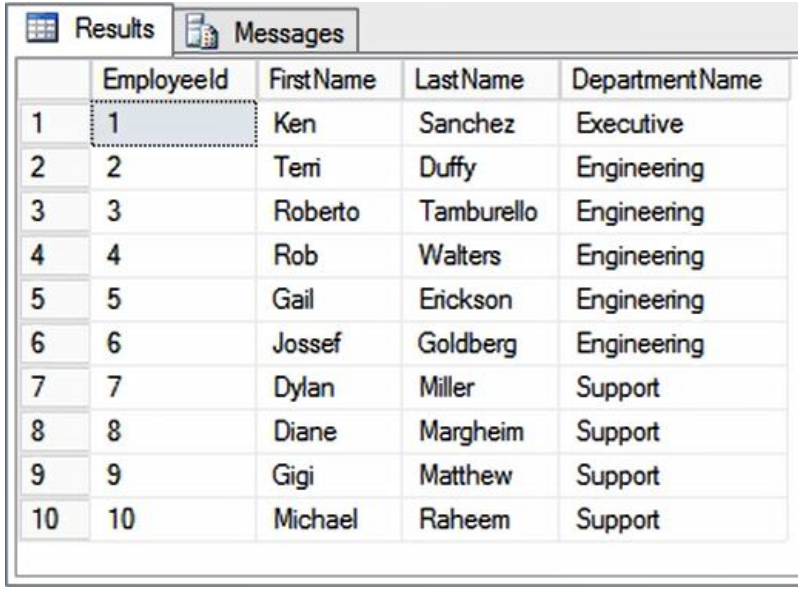
OOP ↔ SQL database

- Tabulka ↔ Třída
- Řádek ↔ Instance
- Sloupec ↔ Pole

SQL databáze: MySQL, SQLite, PostgreSQL

Specifické pro daný jazyk:

- C# - **Entity Framework**, Dapper
- Java - Hibernate
- Python - Django
- Javascript - Knex.js



Results | Messages

	EmployeeId	FirstName	LastName	DepartmentName
1	1	Ken	Sanchez	Executive
2	2	Terri	Duffy	Engineering
3	3	Roberto	Tamburello	Engineering
4	4	Rob	Walters	Engineering
5	5	Gail	Erickson	Engineering
6	6	Jossef	Goldberg	Engineering
7	7	Dylan	Miller	Support
8	8	Diane	Margheim	Support
9	9	Gigi	Matthew	Support
10	10	Michael	Raheem	Support

Working with Temporal Tables in SQL Server 2016 (Part 2)
September 2, 2016 — Leonard Lobel
https://lennilobel.wordpress.com/wp-content/uploads/2016/09/temporal02.png

Výhody / nevýhody

Abstraktnější:

- Nemusíme řešit konkrétní databázi
- Lze popisovat databázi čistě z programovacího jazyka
- Jednodušší použití z daného jazyka
- Lehčí neudělat chybu při ošetření vstupu

Méně kontroly:

- ORM nemusí podporovat všechny typy / operace
- přidání/změna/odstranění proměnné může změnit databázi (nebo naopak)

Třída contextu

```
public class Blog
{
    public int BlogId;
    public string Name;
    public string Body;
}
```

```
public class MyDbContext : DbContext
{
    public DbSet<Blog> Blogs;

    protected override void OnConfiguring(
                DbContextOptionsBuilder options) {
        string DbPath = "./blogs.db";
        options.UseSqlite($"Data Source={DbPath}");
        //...
    }
}
```

Propojení třídy a tabulky (konfigurace)

```
public class Blog
{
    public int BlogId;
    public string Name;
    public string Body;
}
```

```
options.Entity<Blog>()
    .HasKey(a => a.BlogId)

    .Property(a => a.Name)
    .HasColumnName("blog_name")
    .IsRequired()

    .Property(a => a.Body)
    .HasColumnName("blog_body");
```

Propojení třídy a tabulky (atributy)

- Deskriptivnější
- Méně opakování
- Více magické
- Omezenější

```
[Table("blogs")]
public class Blog
{
    [key]
    public int BlogId;
    [Column("tree_id"), Required]
    public string Name;
    [Column("tree_body")]
    public string Body;
}
```

Využití

```
using (var db = new MyDbContext())
{
    db.Blogs.Add(new Blog{ Name = "Name2" });
    db.Blogs.Add(new Blog{ Name = "Name1" });
    db.SaveChanges();
    //...
}
```

```
//...
var short_blogs = db.Blogs
    .Where(a => a.Body.Length <100)
    .OrderBy(b => b.Name)
    .skip(1)
    .ToList();

foreach (var blog in short_blogs)
    db.Blogs.remove(blog);

db.SaveChanges();
```

Relace 1:1

```
public class Blog
{
    public int Id;
    public string Name;
    public Author Author;
    public int AuthorId;
}
```

```
public class Author
{
    public int Id;
    public string Name;
    public Blog Blog;
}
```

```
public class MyDbContext : DbContext
{
    public DbSet<Blog> Blogs;
    public DbSet<Author> Authors;
}
```

Relace 1:1 - Propojení

```
options.Entity<Blog>()
    .HasOne(b => b.Author)
    .WithOne(a => a.Blog)
    .HasForeignKey<Blog>(b => b.AuthorId)
    .OnDelete(DeleteBehavior.NoAction);
```

Relace 1:1 - Propojení atributy (approx.)

```
public class Blog
{
    [Key]
    public int Id;
    public string Name;
    public Author Author;
}
```

```
public class Author
{
    [Key]
    [ForeignKey("Blog")]
    public int Id;
    public string Name;
    public Blog Blog;
}
```

Tabulka Blog a Author už nemají různé primární klíče, ale sdílí tentýž!

Relace 1:N

```
public class Blog
{
    public int Id;
    public string Name;
    public Author Author;
    public int AuthorId;
}
```

```
public class Author
{
    public int Id;
    public string Name;
    public List<Blog> Blogs;
}
```

```
public class MyDbContext : DbContext
{
    public DbSet<Blog> Blogs;
    public DbSet<Author> Authors;
}
```

Relace 1:N - Propojení

```
options.Entity<Blog>()
    .HasOne(b => b.Author)
    .WithMany(a => a.Blogs)
    .HasForeignKey<Blog>(b => b.AuthorId)
    .OnDelete(DeleteBehavior.NoAction);
```

```
public class Blog {
    [Key]
    public int Id;
    public string Name;
    public Author Author;
    [ForeignKey("Author")]
    public int AuthorId;
}
public class Author {
    public int Id;
    public string Name;
    public List<Blog> Blogs;
}
```

Relace N:N

```
public class Blog
{
    public int Id;
    public string Name;
    public List<Author>
Authors;
}
```

```
public class Author
{
    public int Id;
    public string Name;
    public List<Blog> Blogs;
}
```

```
public class BlogAuthor
{
    public int BlogId;
    public int AuthorId;
}
```

```
public class MyDbContext : DbContext
{
    public DbSet<Blog> Blogs;
    public DbSet<Author> Authors;
    public DbSet<BlogAuthor> BlogAuthors;
}
```

Relace M:N - Propojení

```
options.Entity<Blog>()
    .HasMany(b => b.Authors)
    .WithMany(a => a.Blogs)
    .UsingEntity<BlogAuthor>();
```

```
public class BlogAuthor
{
    [Key]
    [Column(Order = 1)]
    [ForeignKey("Blog")]
    public int BlogId;
    [Key]
    [Column(Order = 2)]
    [ForeignKey("Author")]
    public int AuthorId;
}
```

Načítaní dat

Pří provedení dotazu se načítají všechna propojená data “Eager loading”

- Pokud používáme všechna data rychlejší
- Načítání zbytečných dat

Pro zapnutí líného načítání “Lazy loading” stačí přidat při konfiguraci

```
protected override void OnConfiguring(DbContextOptionsBuilder options) {
    options
        .UseLazyLoadingProxies()
        .UseSqlite($"Data Source={DbPath}");
}
```