

Persistência de objetos


Prof. Daniel S. Kaster

Departamento de Computação
Universidade Estadual de Londrina
dskaster@uel.br

Introdução

- Fazer a persistência de um objeto significa
 - salvar o seu estado em memória não volátil (considerando todas as suas propriedades, inclusive as herdadas, bem como suas associações)
- O resultado da persistência é
 - o objeto pode ser reinstanciado num momento posterior exatamente com o mesmo estado que possuía ao ser persistido
- Persistência envolvendo modelos diferentes enfrenta o problema da impedância entre modelos
 - Esforço de mapeamento necessário para que modelos diferentes trabalhem em conjunto

Abordagens básicas de persistência

- Serialização de objetos em mídia não volátil
- Uso de um SGBD orientado a objetos
- Uso de um SGBD relacional e uma técnica de mapeamento objeto-relacional (ORM) 

ORM é um problema difícil

- "Object/relational mapping is the Vietnam of Computer Science." (Ted Neward)
 - "In the case of Vietnam, the United States political and military apparatus was faced with a deadly form of the Law of Diminishing Returns. In the case of automated Object/Relational Mapping, it's the same concern – that early successes yield a commitment to use O/R-M in places where success becomes more elusive, and over time, isn't a success at all due to the overhead of time and energy required to support it through all possible use-cases."

Uma visão mais geral de ORM

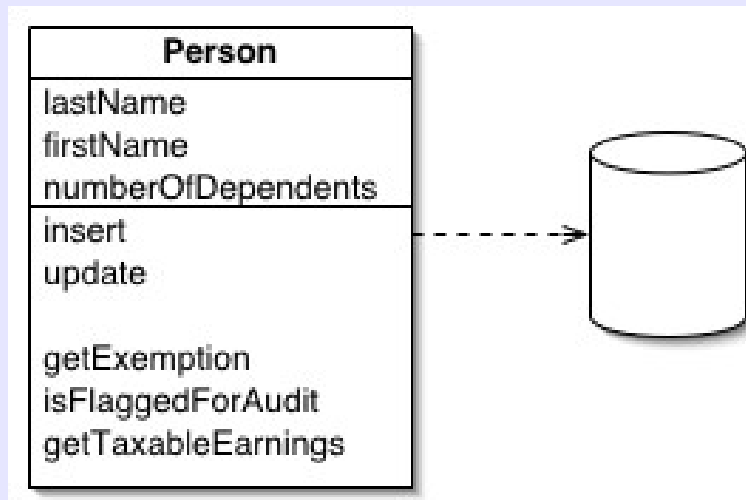
- "Essentially what you are doing is synchronizing between two quite different representations of data, one in the relational database, and the other in-memory. Although this is usually referred to as object-relational mapping, there is really nothing to do with objects here." (Martin Fowler)
- "Essentially the ORM (tools) can handle about 80-90% of the mapping problems, but that last chunk always needs careful work by somebody who really understands how a relational database works." (Martin Fowler)

Padrões de projeto para ORM

- Há vários padrões de projeto propostos para suportar tarefas de ORM
- Padrões interessantes para ORM
 - Active Record
 - Data Mapper
 - Row Data Gateway
 - Table Data Gateway
 - Data Access Objects (DAO)
 - Command Query Responsibility Segregation (CQRS)

Active Record

- Consiste em um objeto que mapeia para uma linha de uma tabela ou visão, encapsulando o acesso ao banco de dados e adicionando lógica do domínio
- O objeto do domínio contém também a lógica de acesso ao banco de dados



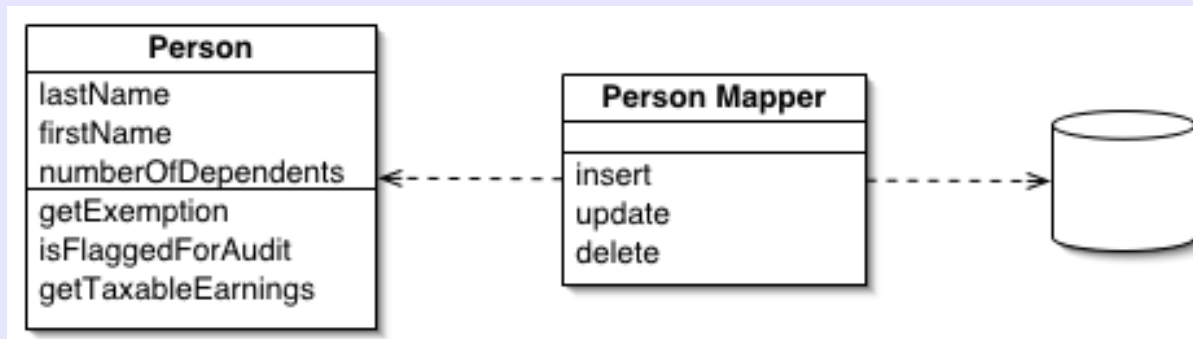
Active Record(2)

- Exemplo:

```
$persistentPerson = new Person();  
$persistentPerson->load(456);  
echo $persistentPerson->firstname;
```

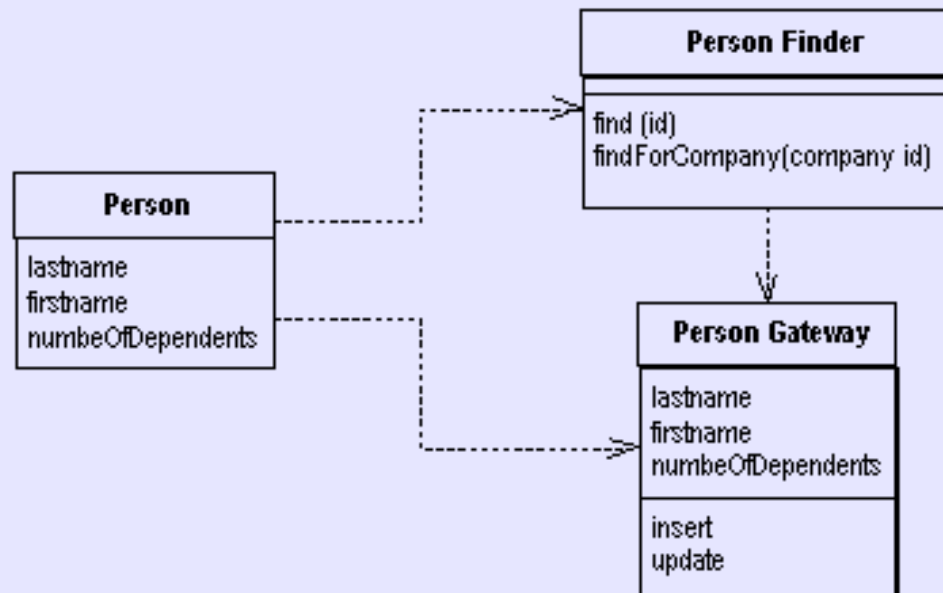

Data Mapper

- Consiste em uma camada de *mappers* que transferem dados entre um objeto em memória e uma fonte de dados, mantendo-os independentes



Row Data Gateway

- Consiste em um objeto que é um *gateway* para um único registro em uma fonte de dados
- Há uma instância por registro e todos os detalhes a respeito do acesso ao registro na fonte de dados são encapsulados nesta interface



Row Data Gateway(2)

- Exemplo: Zend_Db_Table_Row

```
$personGateway = new PersonGateway();  
$person = $personGateway->fetchRow($personGateway->select()->where('id = ?', 1));  
echo $person->firstname;
```

Table Data Gateway

- Consiste em um objeto que atua como um *gateway* para uma tabela (ou visão) de um banco de dados
- Uma instância manipula todas as linhas da tabela e encapsula as instruções para acesso à tabela (ou visão)

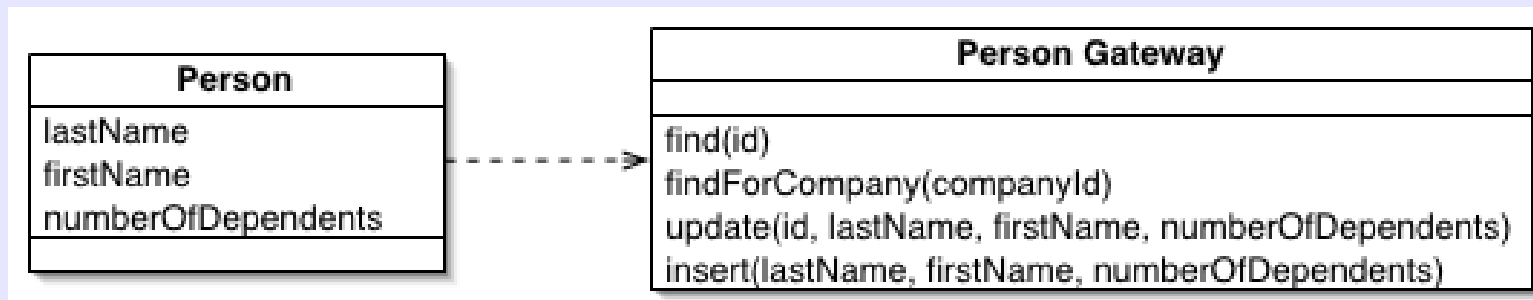


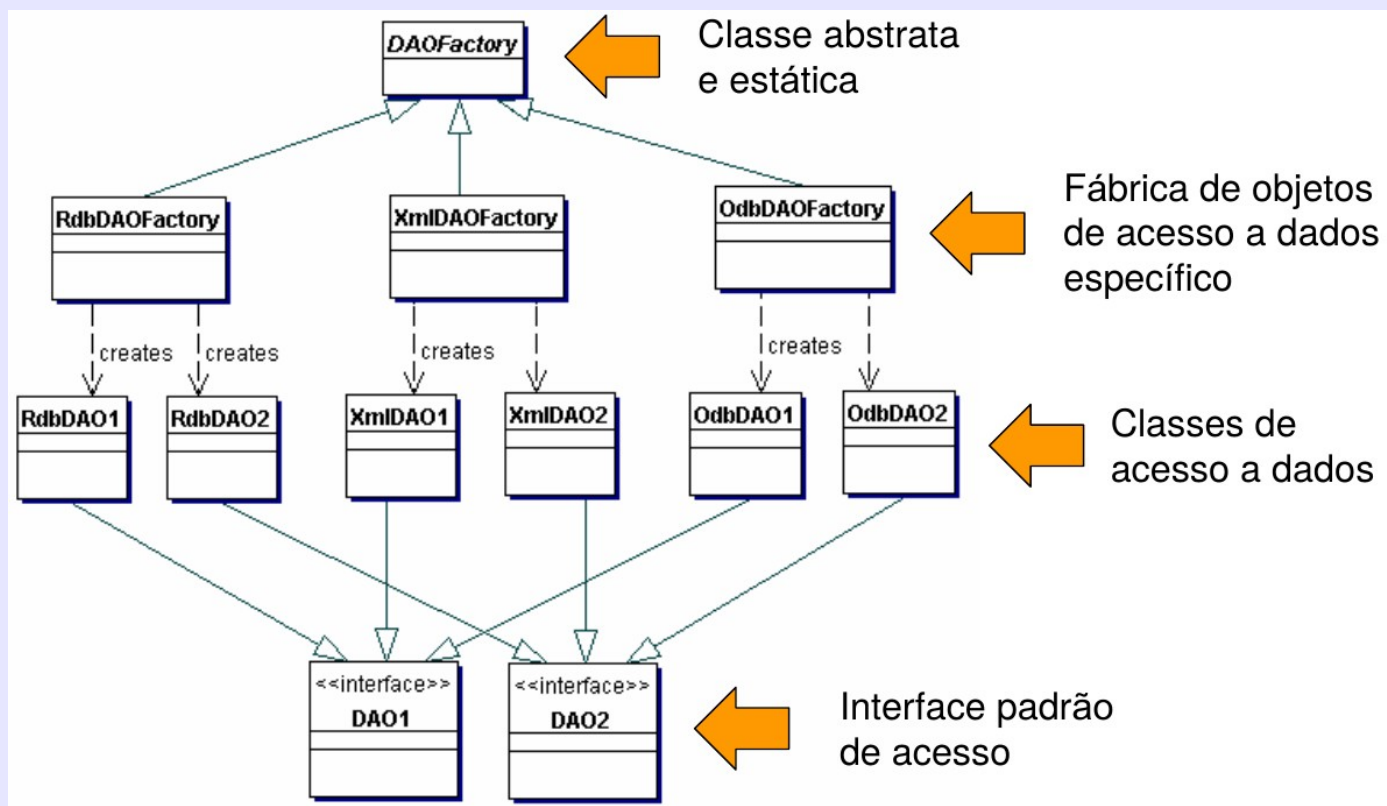
Table Data Gateway(2)

- Exemplo: Zend_Db_Table

```
$personGateway = new PersonGateway();  
$persons = $personGateway->find('Silva');  
$person = $persons->current();  
echo $person->firstname;
```

Data Access Objects

- Consistem em uma camada de abstração que encapsula todo o acesso à fonte de dados referente a objetos do domínio (nem *anemic objects* e nem *god objects*)

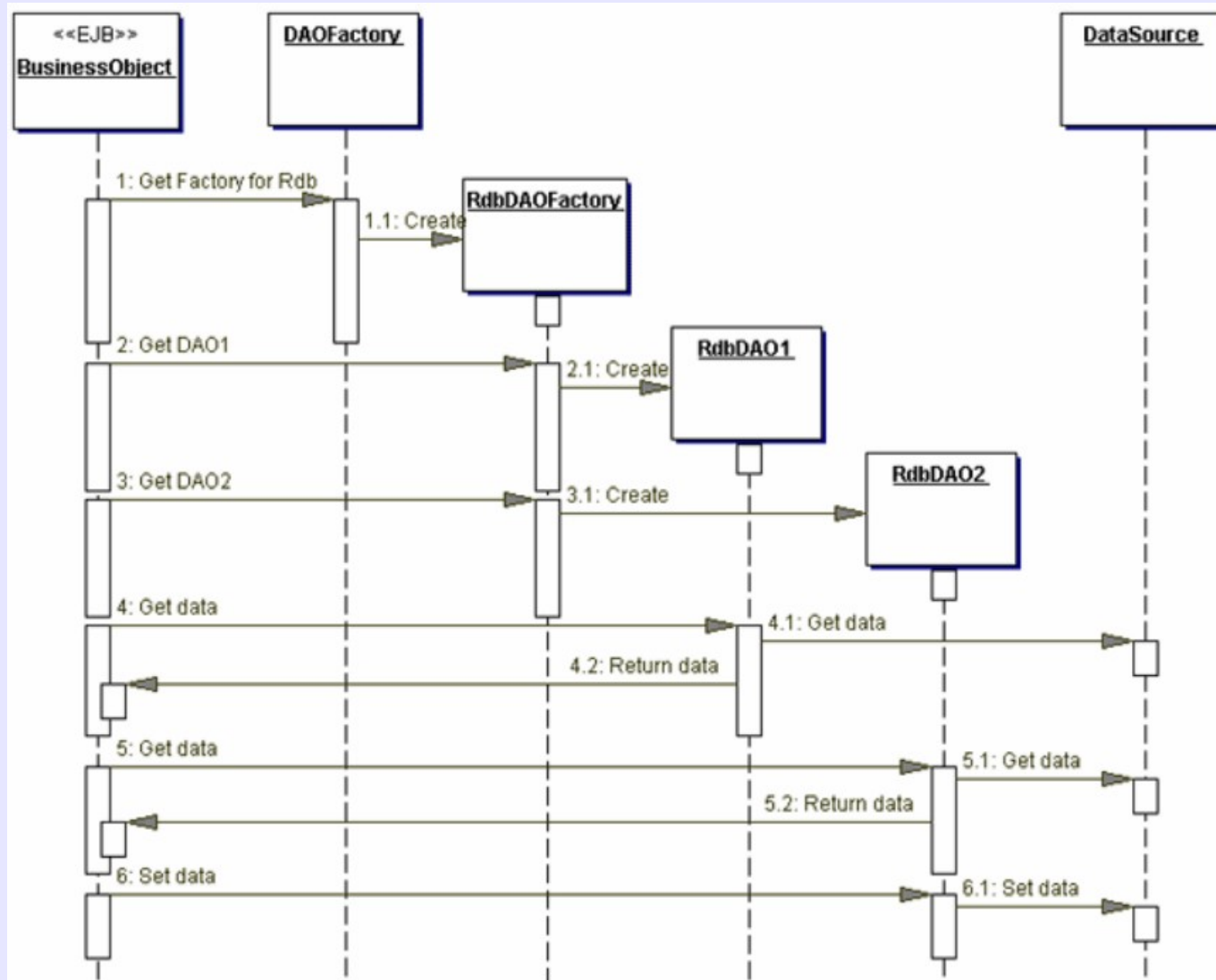


Data Access Objects(2)

- Exemplo:

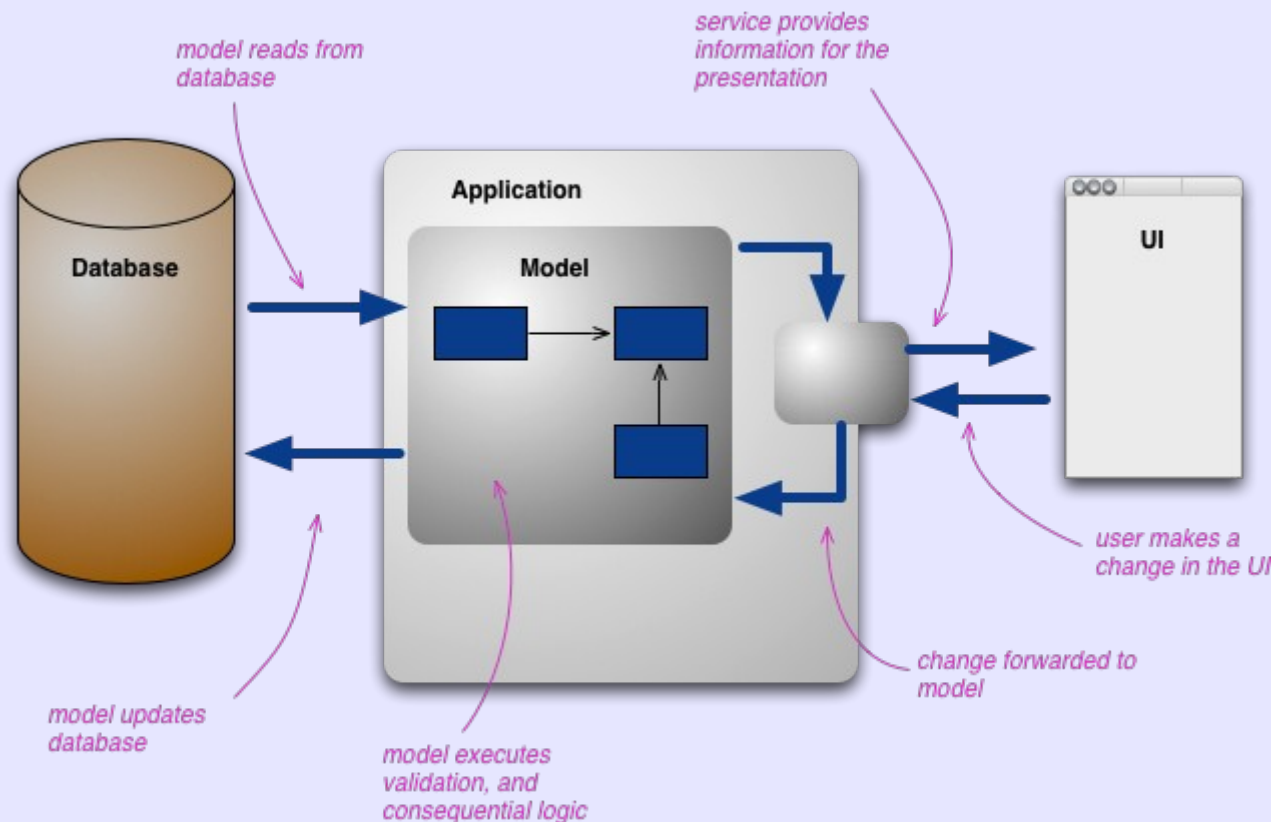
```
$DAOFactory = new RdbDAOFactory();  
$personDAO = $DAOFactory->getPersonDAO();  
$person = $personDAO->findById(456)  
echo $person->firstname;
```

Data Access Objects(3)



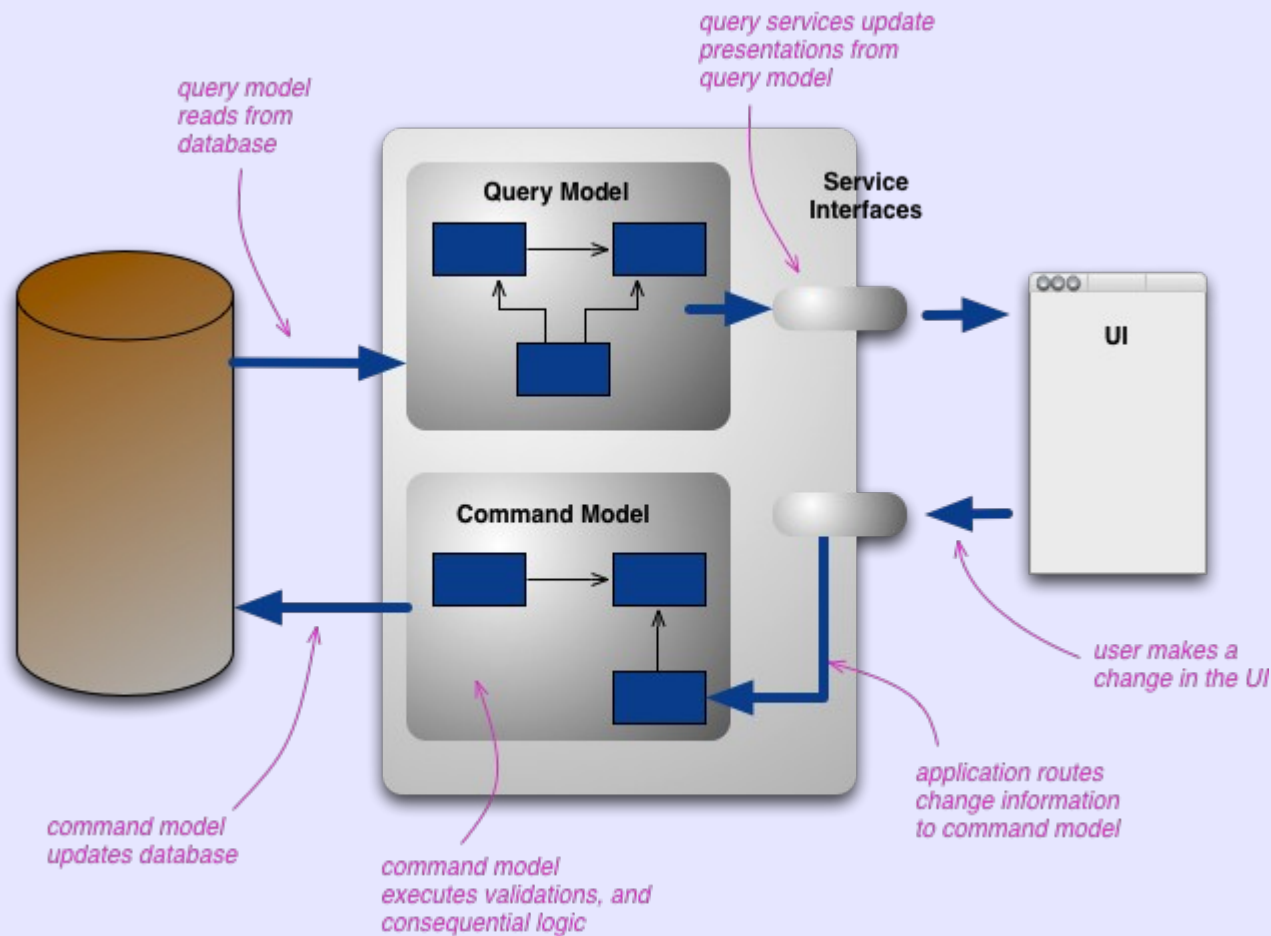
Command Query Responsibility Segregation

- Consiste em uma separação do modelo conceitual em modelos separados para atualização (*command*) e apresentação (*query*)
- Modelo CRUD



Command Query Responsibility Segregation(2)

- Modelo CQRS



Cuidados ao usar uma técnica de ORM

- Uso de lazy/eager loading
 - O uso incorreto pode produzir um número desnecessário de acessos ao banco de dados ou gerar estados inconsistentes
 - É recomendado fazer o *prefetch* de todas as entidades necessárias considerando "unidades" de lógica de negócios (ou utilizar um mecanismo consistente de cache de objetos)
- Uso de ORM para operações em batch
 - Evitar instanciar e persistir objetos individualmente
- Uso eficiente de agregações de dados
 - Normalmente, é preferível utilizar recursos eficientes dos SGBDs para agregar dados do que carregar individualmente objetos e efetuar a agregação sobre os objetos

Ex: eager loading no Hibernate

- O framework de persistência Hibernate faz uso de lazy loading por padrão, mas que pode ser modificado para eager

```
@Entity
@Table(name = "funcionario")
public class FuncionarioLazy {

    @Column
    private String nome;

    @Column
    private long idade;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_endereco")
    private Endereco endereco;
```

```
@Entity
@Table(name = "funcionario")
public class FuncionarioEager {

    @Column
    private String nome;

    @Column
    private long idade;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "id_endereco")
    private Endereco endereco;
```

Ex: eager loading no SQLAlchemy

- O SQLAlchemy também usa lazy loading por padrão (lazy="select"), mas que pode ser modificado
- Modificação da configuração no mapeamento

```
class Parent(Base):  
    __tablename__ = "parent"  
  
    id = Column(Integer, primary_key=True)  
    children = relationship("Child", lazy="joined")
```

- Modificação *per-query basis*

```
# set children to load eagerly with a join  
session.query(Parent).options(joinedload(Parent.children)).all()
```

Ex: batch updates no Hibernate

- Da forma padrão, os updates são enviados em instruções separadas para o SGBD

```
@Transactional
@Test
public void whenNotConfigured_ThenSendsInsertsSeparately() {
    for (int i = 0; i < 10; i++) {
        School school = createSchool(i);
        entityManager.persist(school);
    }
    entityManager.flush();
}
```

Ex: batch updates no Hibernate(2)

- Mas, é possível enviar em batches, para obter mais eficiência

```
public Properties hibernateProperties() {  
    Properties properties = new Properties();  
    properties.put("hibernate.jdbc.batch_size", "5");  
  
    // Other properties...  
    return properties;  
}
```

```
@Transactional  
@Test  
public void whenFlushingAfterBatch_ThenClearsMemory() {  
    for (int i = 0; i < 10; i++) {  
        if (i > 0 && i % BATCH_SIZE == 0) {  
            entityManager.flush();  
            entityManager.clear();  
        }  
        School school = createSchool(i);  
        entityManager.persist(school);  
    }  
}
```

Referências

- ELMARSI, R.; NAVATHE, S. B.; Fundamentals of database systems. Addison-Wesley, 6rd edition, 2010.
- <http://java.dzone.com/articles/martin-fowler-orm-hate>
- <http://martinfowler.com/eaCatalog/>
- <http://blog.fedecarg.com/2009/03/12/domain-driven-design-and-data-access-strategies/>
- FERREIRA, J. E.; TAKAI, O. K. Persistência de objetos (slides). IME-USP