# UNIDADE: TÉCNICAS DE PROJETO DE SISTEMAS RAD


## *Integração com linguagens de programação(conexão, ORM, entre outras técnicas)*

***Prof. Roberson Alves***

# AGENDA

- ✓ **Integração BD e Linguagens**
- ✓ **Drivers e APIs**
- ✓ **ORM**
- ✓ **Versionamento de Schemas**

# INTEGRAÇÃO BD E LINGUAGENS

- **Todas as linguagens de programação tem suporte a BD;**
- **Acesso geralmente é feito por drivers;**
- **BD pode também ter driver específico.**

# DRIVERS E APIS

- **São os recursos de software, responsáveis por permitir o acesso a determinado BD;**
- **Permitem realizar a conexão, consulta, inserção, atualização, controle transacional, entre outros recursos.**




# ORM

- **ORM = OBJECT-RELATIONAL MAPPING OU MAPEAMENTO OBJETO-RELACIONAL**
- **O descasamento dos paradigmas: Relacional X OO**
- **Há vários pontos onde o modelo relacional é incompatível com o modelo de objetos**
  - **Granularidade**
  - **Herança e polimorfismo**
  - **Identidade**
  - **Associações**
  - **Navegação em grafos**

# ORM - INCOMPATIBILIDADES DOS PARADIGMAS

- **Problema da identidade**
  - **No mundo relacional, existe um critério de igualdade:**
    - **Chave-primária**
  - **No mundo Java há dois**
    - **Igualdade de referência (testado com ==)**
    - **Equivalência (testado com equals())**
  - **Além disso, mapeamento pode associar vários objetos a uma mesma tabela!**
  - **Complicações adicionais**
    - **Chaves naturais**
    - **Chaves compostas**

# ORM - INCOMPATIBILIDADES DOS PARADIGMAS

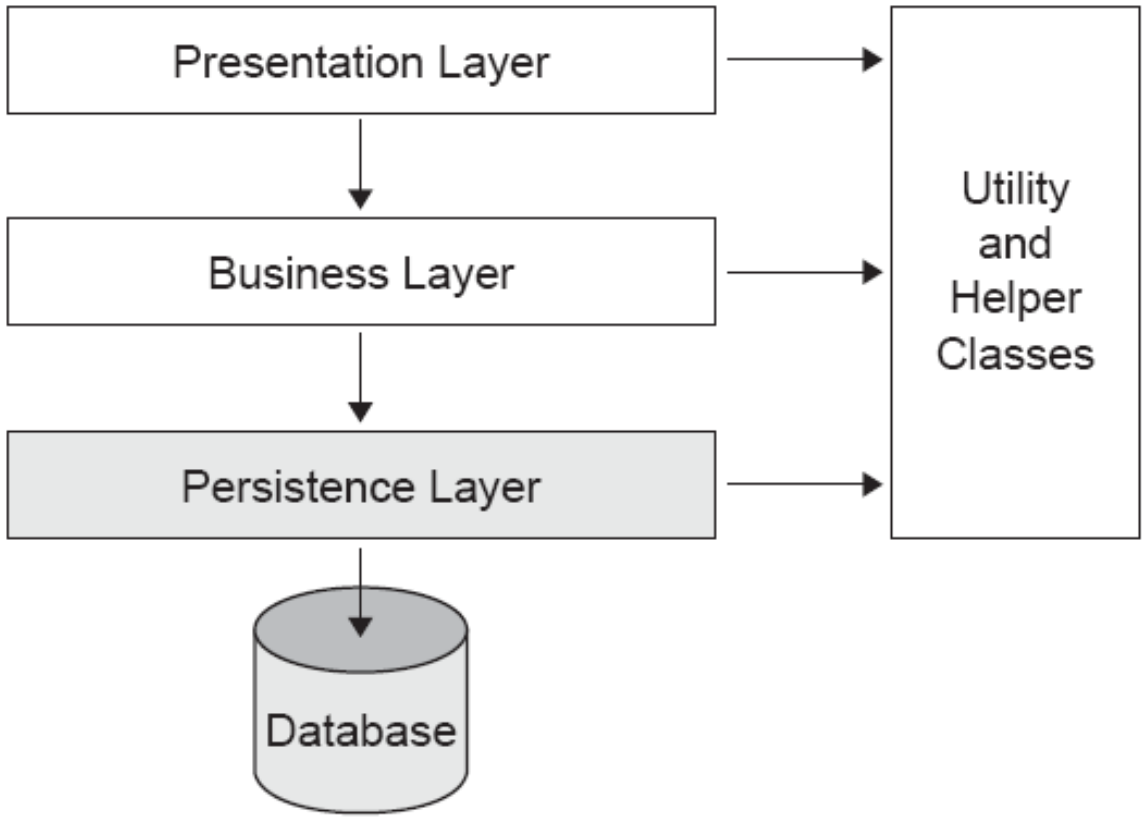
- **Problema das Associações**
  - **Java representa associações como referências (ou coleções de) referências para objetos**
    - **São inerentemente direcionais**
    - **Para implementar associações bidirecionais, é preciso criar referências dos dois lados da associação**
    - **Referências dos dois lados podem ser associações M-N**
  - **No mundo relacional, associações são representadas por chaves estrangeiras**
    - **Não são inerentemente direcionais**
    - **Pode-se criar associações arbitrárias com projeção e *joins***
    - **Associações M-N requerem tabela extra**

# ORM - INCOMPATIBILIDADES DOS PARADIGMAS

- **Problema da Navegação em grafos**
  - **Grafos de objetos vs. table joins**
  - **Navegação em objetos**
    - **Pula-se de um objeto para outro: objeto.getA().getB() sem a definição de um caminho previamente definido**
    - **Equivalente a fazer um query para cada pulo (nó do grafo)**
    - **Portanto, a forma mais natural de navegar entre objetos em Java é a forma menos eficiente de recuperar dados em SQL**
  - **Solução: usar joins para minimizar queries**
    - **Porém, é preciso traçar o caminho de navegação antes!**

# ORM - SOLUÇÃO

- **Usar uma camada de persistência para lidar com as incompatibilidades dos paradigmas**
  - **Requer arquitetura com separação em camadas que concentram-se em um interesse predominante**
  - **Solução recomendada pelos padrões JEE**



# ORM - SOLUÇÃO

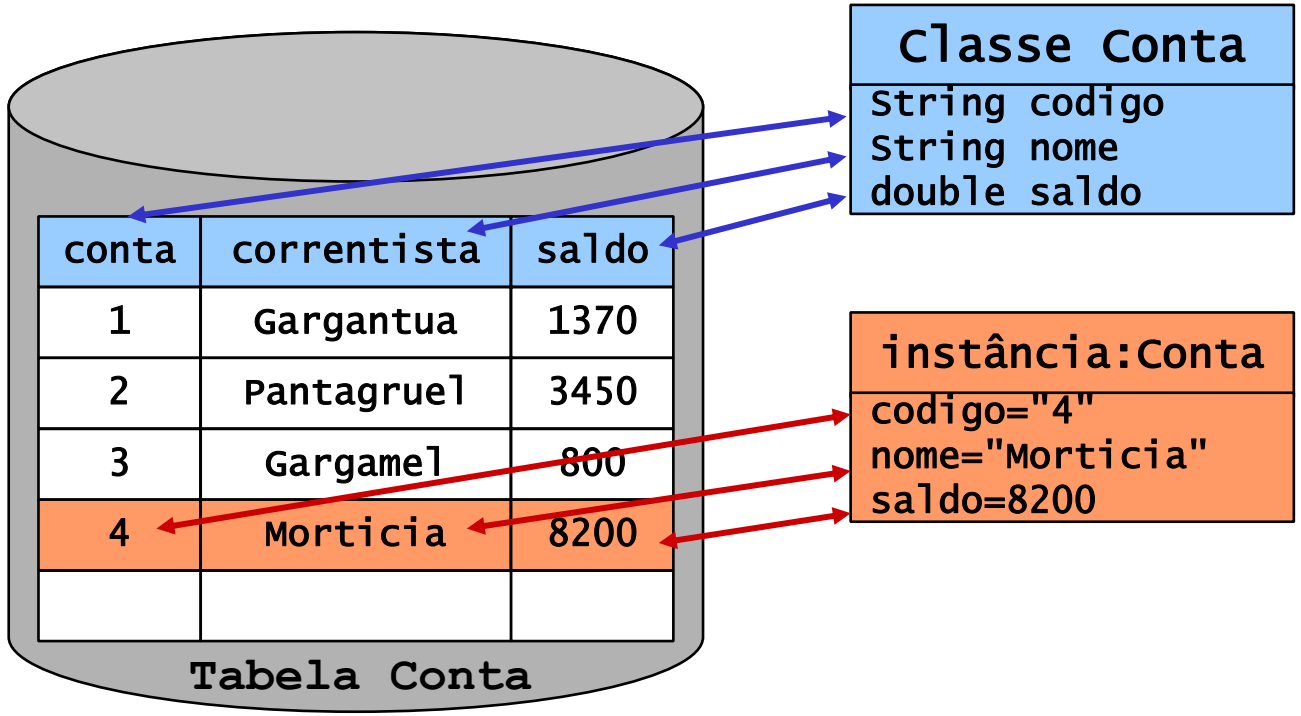
- **Criar uma camada de persistência usando JDBC.**
- **O padrão mais usado é o DAO – Data Access Object**
  - **Isola todas as chamadas ao banco (SQL) em um objeto e fornece uma API via interface para clientes**
  - **Clientes são objetos de negócio que desconhecem a tecnologia de persistência usada**
- **Mas criar uma boa camada de persistência exige trabalho**
  - **É preciso implementar eficientemente todo o SQL de acesso, relacionamentos, atualização, etc. e integrar com APIs de transações, cache, etc**

# ORM - SOLUÇÃO

- **É basicamente implementar todo o mapeamento Objeto Relacional "a mão";**
- **Opa! O que é mesmo Mapeamento objeto-relacional (ORM)?**

# ORM

- **Princípios básicos**
  - **Classes são mapeadas a tabelas (esquemas)**
  - **Instâncias (objetos) são mapeadas a registros (linhas)**



# ORM

- **Porém quando falamos em ORM, estamos nos referindo normalmente ao processo automatizado;**
- **Uma solução de ORM completa normalmente nos oferecerá**
  - **Uma API para realização de operações básicas de criação, leitura, atualização e remoção (CRUD) em objetos de classes persistentes**
  - **Uma linguagem ou API para especificar consultas sobre classes e suas propriedades**
  - **Um recurso para especificar meta dados de mapeamento**
  - **Uma técnica que permita à implementação interagir com objetos transacionais (para realizar funções de otimização)**

# EXEMPLOS DE ORM

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping auto-import="true">
    <class name="br.com.meuprojeto.bean.Venda" schema="venda"
        table="venda" dynamic-update="true">
        <id name="id" type="integer">
            <generator class="sequence">
                <param name="sequence">venda.venda_id_seq</param>
            </generator>
        </id>
        <property name="data" type="date" column="data" />
        <property name="observacao" type="string" column="observaca

        <many-to-one name="cliente" class="br.com.meuprojeto.bean.C
            column="id_cliente" cascade="none" outer-join="false" /
        <many-to-one name="atendente" class="br.com.meuprojeto.bean
            column="id_atendente" cascade="none" outer-join="false"
```

```java
@Entity
public class User implements Serializable {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  @Column(length = 20, nullable = false)
  private String username;

  @OneToOne(cascade = CascadeType.ALL)
  @JoinColumn(name = "user_profile_id")
  private UserProfile userProfile;
```

```python
from django.db import models

#DataFlair #DjangoTutorials
# Create your models here.
class Customer(models.Model):
    name = models.CharField(max_length=255)

class Vehicle(models.Model):
    name = models.CharField(max_length=255)
    customer = models.OneToOneField(
        Customer,
        on_delete=models.CASCADE,
        related_name='vehicle'
    )
```

# BIBLIOTECAS E FRAMEWORKS ORM



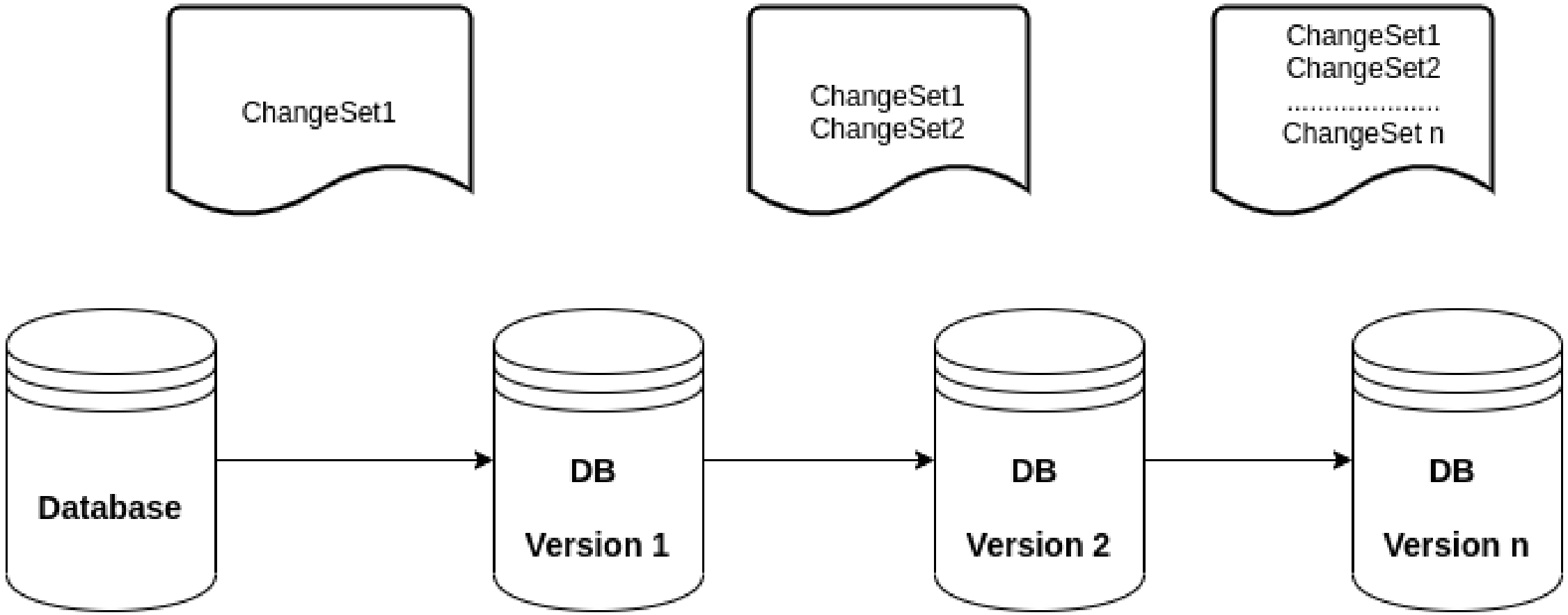




# VERSIONAMENTO DE SCHEMAS(ESQUEMAS)

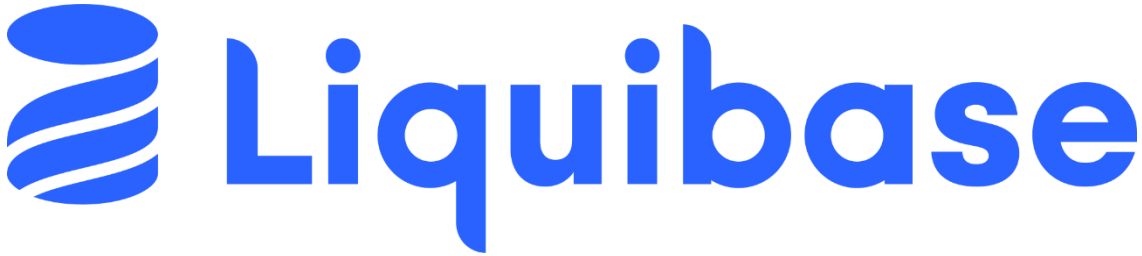
- **Refere-se a "versionar" a estrutura do banco de dados;**
- **Por que fazer isso?**
  - **A estrutura de dados muda com frequência. FATO!**
  - **Na maioria das vezes liberamos scripts SQL para atualizar o nosso schema do BD;**
  - **Mas como saber em que versão a base de dados do usuário/cliente está?**

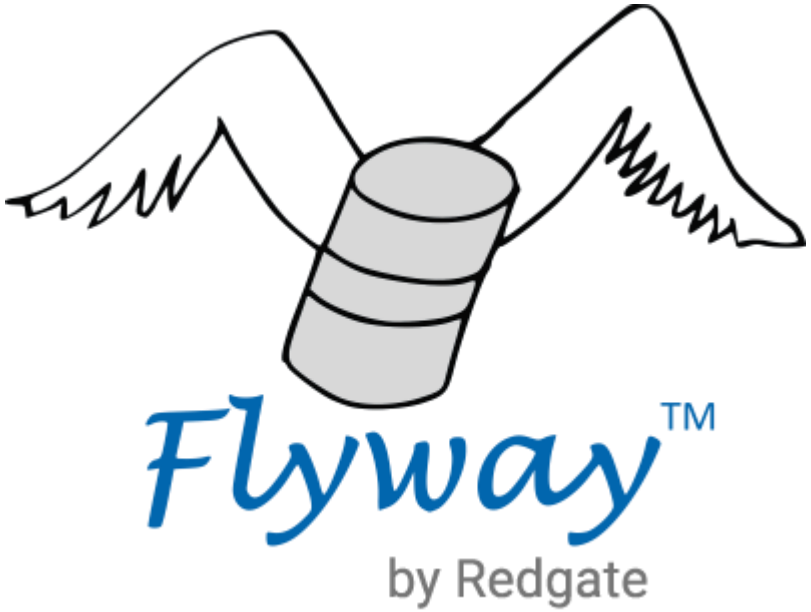
# VERSIONAMENTO DE SCHEMAS(1)



# VERSIONAMENTO DE SCHEMAS(2)

- **Algumas ferramentas que podem ser utilizadas:**



# VERSIONAMENTO DE SCHEMAS(3)

- **O que essas ferramentas oferecem:**
  - **Evoluir o banco de dados por meio de scripts de versionamento;**
  - **Controla a versão do banco de dados;**
  - **Suporta rollback em caso de problemas;**
  - **Automatizar a execução de scripts SQL;**
  - **Registrar todas as mudanças executas na estrutura do BD;**
  - **É possível reverter versões.**

# VERSIONAMENTO DE SCHEMAS: MÃOS NA MASSA

- **Vamos fazer um exemplo prático com Spring Data e Liquibase;**
- **Baixar o projeto exemplo disponível;**
- **Projeto utiliza:**
  - **Um modelo simples de BD;**
  - **Maven;**
  - **Spring Data; e**
  - **Liquibase.**

# VERSIONAMENTO DE SCHEMAS: MÃOS NA MASSA

- **Para vocês fazerem:**
  - **Adicione um método findByNameLike(String name) customizado no repositório Country;**
  - **Adicione um método findByfirstNameStartsWith(String firstName);**
  - **Adicionem mais um changeset que adiciona uma constraint na tabela Employee, coluna email, que obriga a coluna a conter o símbolo @;**