Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

2019/2020

Construção de Sistemas de Software

Padrões para a Camada de Acesso aos Dados
— *Gateways* —

Padrões para as várias camadas

Apresentação	MVC Front controller Page controller Page template
Negócio	Domain Model Transaction Script Table Module
Dados	Row data gateway Table data gateway Active Record Data Mapper

Camada de Dados

- Como vimos anteriormente, a **Camada de Dados** (***Data Source Layer***) é uma das principais camadas das EA, responsável pela comunicação com base de dados, sistemas de mensagens, gestores de transações ou outros sistemas externos
- Vamos nos focar numa das situações mais comuns, que é ter bases de dados relacionais como fontes de dados
- Existem **dois padrões** para organizar a camada de dados baseados em ***gateways*** — i.e., objetos que encapsulam o acesso a um sistema externo ou recurso
  - o código de acesso a uma determinada API é embrulhado numa classe cujo interface é o de um objeto normal
  - os outros objetos acedem ao recurso através da *gateway*, que traduz as chamadas aos seus métodos nas tarefas que são exigidas pela API do recurso

Row Data Gateway

- Uma ***Row Data Gateway*** é um objeto que funciona como uma *gateway* para um **registo** de uma fonte de dados
  - permite que os clientes acedam e manipulem o registo usando as primitivas da linguagem de programação, encapsulando todas as especificidades da API que permite aceder à fonte de dados
- Sendo uma ***gateway*** este objeto não realiza qualquer tarefa que não esteja relacionada com a manipulação do registo que representa (só tem lógica de acesso à bd)

id	designation	phonenumber	vatnumber	discountid
1	xico	912345678	128542663	1
2	maria	963456789	197672337	1

- Esta abordagem para a camada de dados funciona especialmente bem com o ***Transaction Script*** na camada de negócio

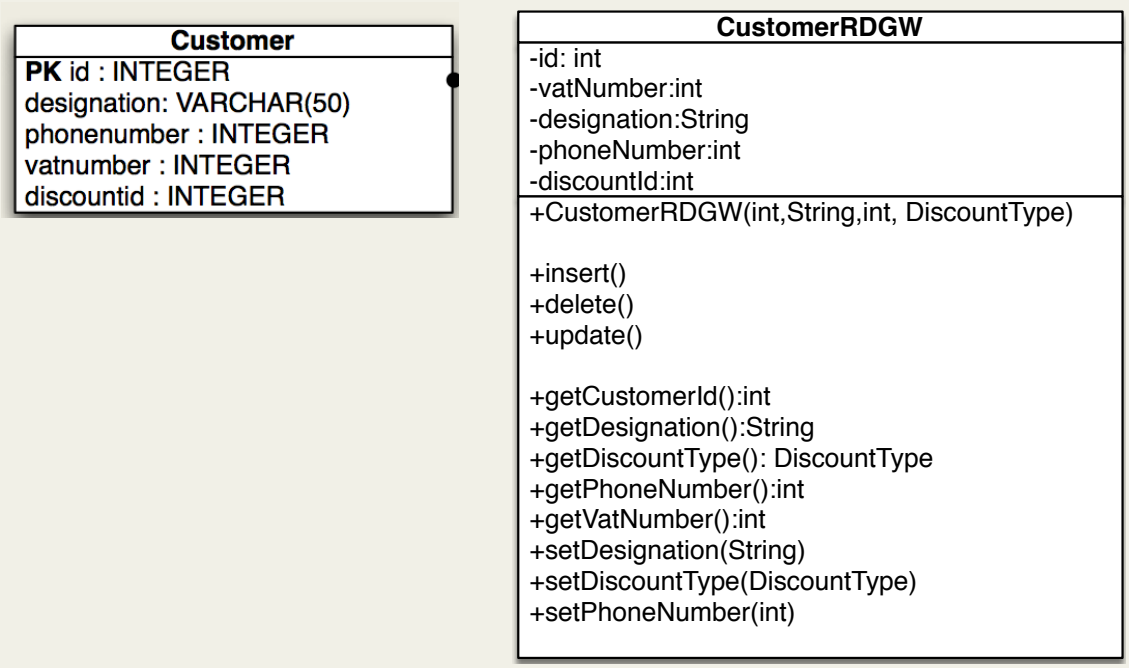
Row Data Gateway

Em termos de organização do código:

- O **esquema** de cada tabela **Table** é representado por uma **classe** **TableRDGateway** cujos **objetos** representam os **registos** de **Table**

- A classe **TableRDGateway**
  - tem um **atributo** por cada **coluna** de **Table** (com o tipo correspondente)
  - tem construtores que recebem os valores dos atributos necessários para a construção
  - tem ***getters*** e ***setters*** para os diferentes atributos
  - com métodos de instância ***insert()***, ***delete()*** e ***update()***/ ***updateXYZ()*** para inserir um registo, apagar ou alterar o registo representado pelo *this* em **Table**

Row Data Gateway: Exemplo



Row Data Gateway: Exemplos de Utilização

- **Inserir um novo cliente**

  1. Criar o objeto **CustomerRDGW** correspondente ao cliente
  2. Preencher os atributos do objeto, via **construtor** ou através de chamadas aos ***setters***
  3. chamar o método ***insert()***

- **Alterar ou apagar cliente**

  Tendo o objeto **CustomerRDGW** correspondente ao registo do cliente a alterar ou modificar

  1. Chamar ***setters*** para alterar o objeto (caso se pretenda alterar)
  2. Chamar o método ***update()*** ou ***delete()*** para se alterar ou apagar o correspondente registo na tabela da BD

Row Data Gateway: Operações de procura

Adicionalmente é preciso decidir onde colocar os métodos que permitem fazer os tipos de procura requeridos pela lógica de negócio

1. **métodos de classe** em **TableRDGateway**

   -- impede o uso de polimorfismo, que é útil se quisermos ter diferentes implementações dos métodos de procura para diferentes fontes de dados

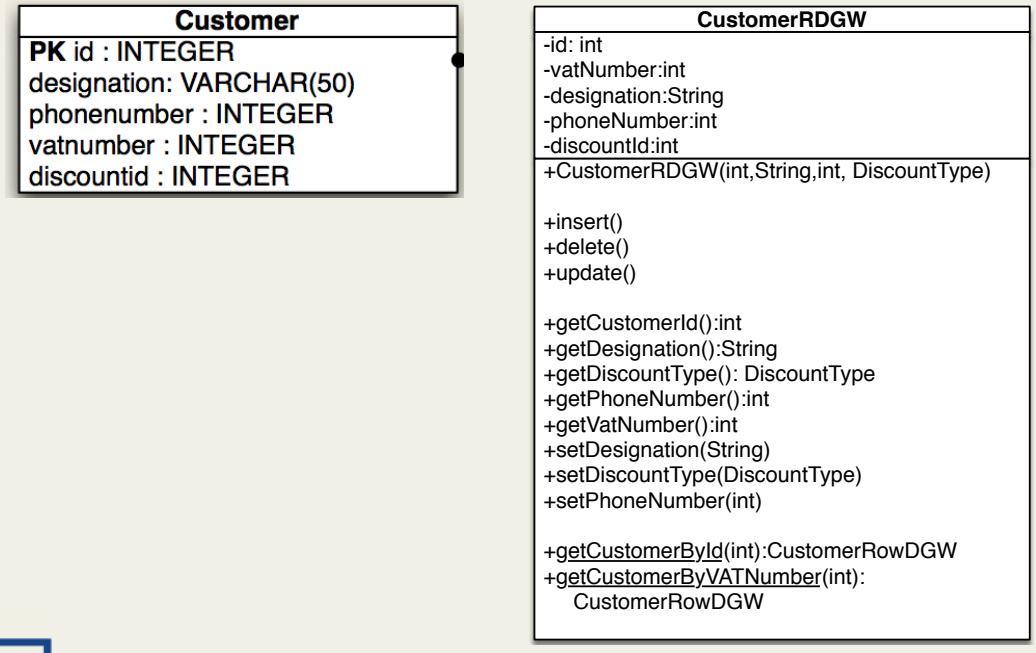
2. **métodos de instância** de uma classe **TableFinder**

   os objetos desta classe tem a responsabilidade de fazer as pesquisas e construir os respetivos *gateways* (i.e., objetos do tipo **TableRDGateway**)

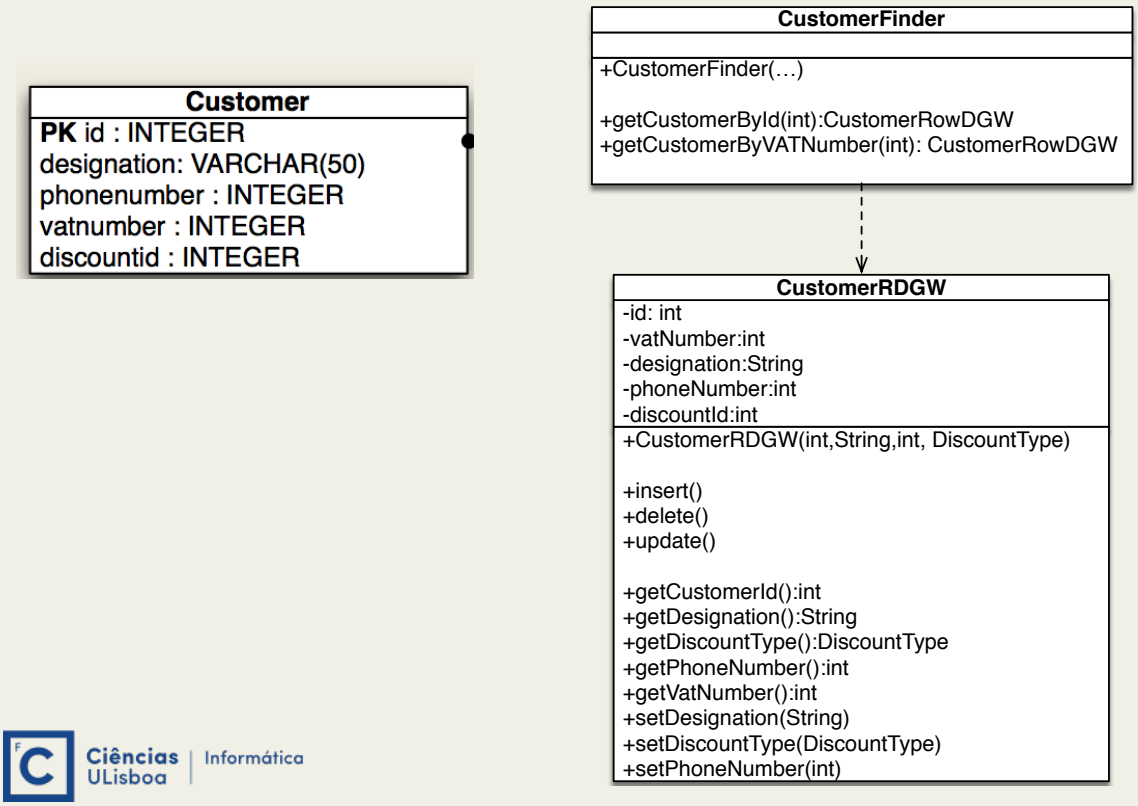
   ++ a classe *gateway* pode ser gerada automaticamente

   ++ a parte de procura está isolada numa única classe

Row Data Gateway: Exemplo



Row Data Gateway: Exemplo



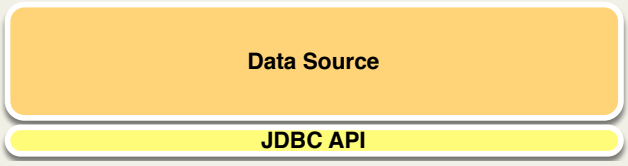
JDBC *detour*

- **JDBC = Java DataBase Connectivity**

  API Java que permite aceder a praticamente todos os tipos de fontes de dados, nomeadamente bases de dados relacionais e *spreadsheets*.

  - **java.sql**
  - **javax.sql**

- Estabelece um contrato entre
  - quem programa a aplicação e
  - quem fornece a ligação à base de dados específica
    - Oracle
    - MySql
    - Derby
    - ...



JDBC Modelo de Programação

É um modelo Cliente/Servidor

1. Cliente cria uma **ligação** à fonte de dados física (servidor de base de dados)
2. Cliente envia **comandos ou interrogações** SQL para serem executados através dessa ligação
3. Servidor envia de volta um **conjunto de resultados**
4. Cliente processa os resultados
5. Cliente fecha a ligação

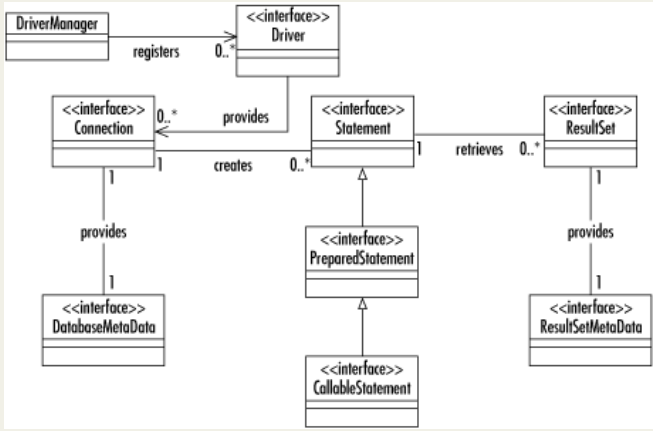
Notar que

- Todos os erros são reportados através de **exceções**
- Os conjuntos de resultados têm associados **meta-dados**

JDBC *detour*

Principais Elementos da API

- **DriverManager** – Uma fábrica para ligações à fonte de dados física
- **Connection** - Uma ligação (sessão) com a base de dados que permite criar e executar comandos (e gerir transações)
- **Statement** – Um comando SQL (*update* ou *query*) a ser executado através de uma ligação
- **PreparedStatement** – Versão de comandos mais segura e eficiente (*cached* e evita injeção de SQL)
- **ResultSet** – Uma estrutura de dados que contém informação que resulta da execução de *queries*.



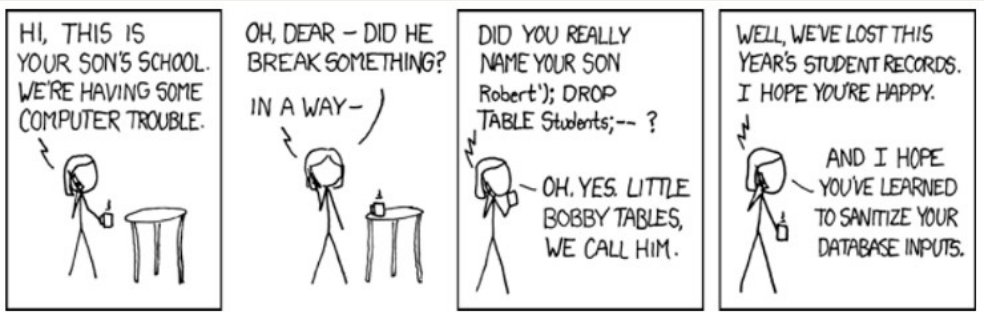
JDBC *detour:* Statement vs PreparedStatement

```
Statement st = con.createStatement();
st.executeUpdate("update STUDENT set NAME =" + student.getName()+ …);

//pre-compiled (once) and DB-side caching
PreparedStatement pst =
    con.prepareStatement("update STUDENT set NAME = ? where ID = ?");
pst.setString(1, student.getName());
pst.setInt(2, student.getId());
pst.executeUpdate();
```

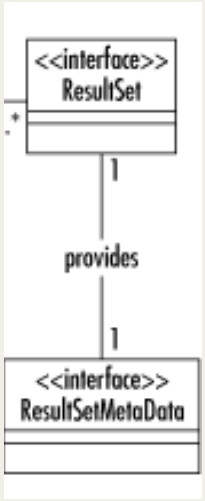
+ seguro, + eficiente, + legível

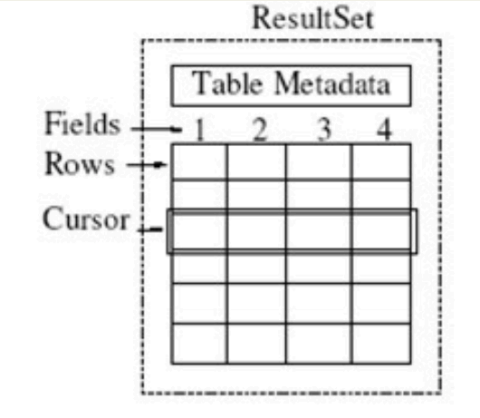
https://xkcd.com/327/



JDBC *detour*

- **ResultSet** – Uma estrutura de dados (em forma de tabela) que contém informação que resulta da execução de *queries*.
  - Valores das colunas acedidos por índice ou nome.
  - Meta-dados descrevem os nomes das colunas e os seus tipos.





JDBC Exemplo

CustomerRDGW
-id: int -vatNumber:int -designation:String -phoneNumber:int -discountId:int
+CustomerRDGW(int,String,int, Discount

Customer
PK id : INTEGER designation: VARCHAR(50) phonenumber : INTEGER vatnumber : INTEGER discountid : INTEGER

```
/**
 * Table name
 */
private static final String TABLE_NAME = "customer";

/**
 * Field names
 */
private static final String ID_COLUMN_NAME = "id";
private static final String VAT_NUMBER_COLUMN_NAME = "vatnumber";
private static final String DESIGNATION_COLUMN_NAME = "designation";
private static final String PHONE_NUMBER_COLUMN_NAME = "phonenumber";
private static final String DISCOUNT_ID_COLUMN_NAME = "discount_id";
```

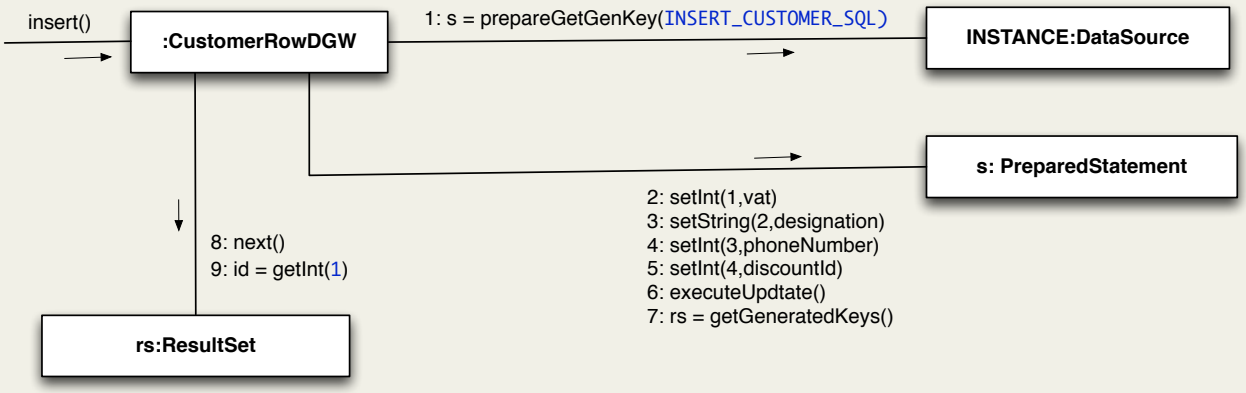
JDBC Exemplo

CustomerRDGW
-id: int -vatNumber:int -designation:String -phoneNumber:int -discountId:int
+CustomerRDGW(int,String,int, Discount +insert() +delete()

Customer
PK id : INTEGER designation: VARCHAR(50) phonenumber : INTEGER vatnumber : INTEGER discountid : INTEGER

```
/**
 * insert a customer
 */
private static final String INSERT_CUSTOMER_SQL =
    "insert into " + TABLE_NAME + " " +
        "(" + ID_COLUMN_NAME + ", " +
              VAT_NUMBER_COLUMN_NAME + ", " +
              DESIGNATION_COLUMN_NAME + ", " +
              PHONE_NUMBER_COLUMN_NAME + ", " +
              DISCOUNT_ID_COLUMN_NAME + ") " +
    "values (DEFAULT, ?, ?, ?, ?)";
```


JDBC Exemplo

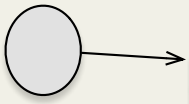


JDBC Exemplo

```
public void insert () throws PersistenceException {
    try (PreparedStatement statement =
                DataSource.INSTANCE.prepareGetGenKey(INSERT_CUSTOMER_SQL)) {
        // set statement arguments
        statement.setInt(1, vat);
        statement.setString(2, designation);
        statement.setInt(3, phoneNumber);
        statement.setInt(4, discountId);
        // executes SQL
        statement.executeUpdate();
        // Gets customer Id generated automatically by the database engine
        try (ResultSet rs = statement.getGeneratedKeys()) {
            rs.next();
            id = rs.getInt(1);
        }
    } catch (SQLException e) {
        throw new PersistenceException ("Internal error!", e);
    }
}
```


Table Data Gateway

- Uma ***Table Data Gateway*** é um objeto que funciona como uma *gateway* para uma tabela de uma base de dados
- Permite que os clientes acedam e manipulem os registos da tabela usando as primitivas da linguagem de programação, encapsulando todas as especificidades da API que permite aceder à fonte de dados



id	designation	phonenumber	vatnumber	discountid
1	xico	912345678	128542663	1
2	maria	963456789	197672337	1

- Esta abordagem para a camada de dados é especialmente adequada para o ***Table Module*** na camada de negócio, mas também é uma alternativa para o ***Transaction Script***

Table Data Gateway

Em termos de organização do código:

- O **esquema** de cada tabela **Table** é representado por uma **classe** **TableTDGateway**

- A classe **TableRDGateway**
  - tem métodos que permitem fazer os tipos de procura requeridos pela lógica de negócio e que permitem inserir novos registos, apagar um registo ou alterar um registo e que
    - fazem o mapeamento dos parâmetros de entrada em comandos SQL
    - quando retornam informação, é na forma de conjuntos de resultados (***Record Set/Result Set***)
  - pode oferecer métodos para facilitar a extração de informação dos conjuntos de resultados

Table Data Gateway: Exemplo

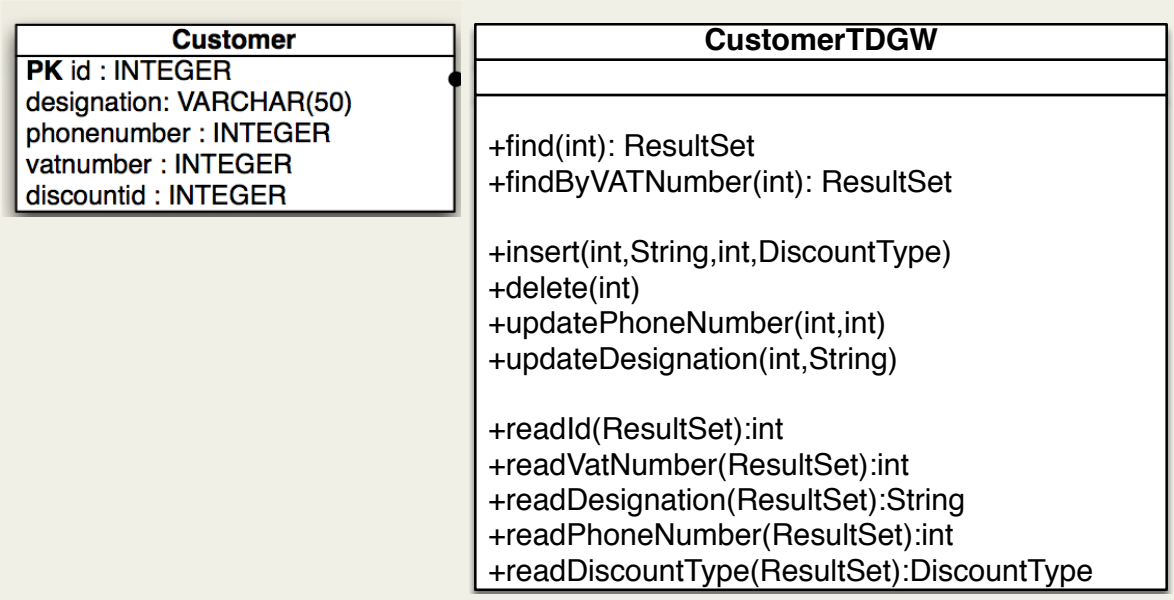


Table Data Gateway: Exemplos de Utilização

- **Inserir um novo cliente**
  1. Obter o objeto **CustomerTDGW** (geralmente só existe um no sistema)
  2. Chamar o método ***insert*** que trata da criação com os dados de input necessários (notar que pode haver mais do que um)

- **Apagar cliente**
  1. Obter o objeto **CustomerTDGW**
  2. Chamar o método ***delete*** com o valor do *id* (chave) como input

Combinações de Padrões para Camadas de Negócio e Dados

- ***Transaction Script***
  - *RowDataGateway*
  - *TableDataGateway*

    se é mais conveniente trabalhar com os conjuntos de resultados nos *scripts*

- ***Table Module***
  - *TableDataGateway*

- ***Domain Model***

Sumário

- Dois padrões de organização da camada de dados e utilização de *gateways*: *row data gateway* (organização por linha) e *table data gateway* (organização por tabela).
- Discussão sobre a utilização dos *gateways* (*row* e *table*) mais adequados pata os diferentes padrões da organização da camada de negócio.
- A implementação dos padrões *row data gateway* e *table data gateway* usando a API JDBC do Java SE, recorrendo nomeadamente aos interfaces *DriverManager*, *Connection*, *PreparedStatement* e *ResultSet*.
- Ilustração com o exemplo do *SaleSys*.