Bacharelado em Sistemas de Informação

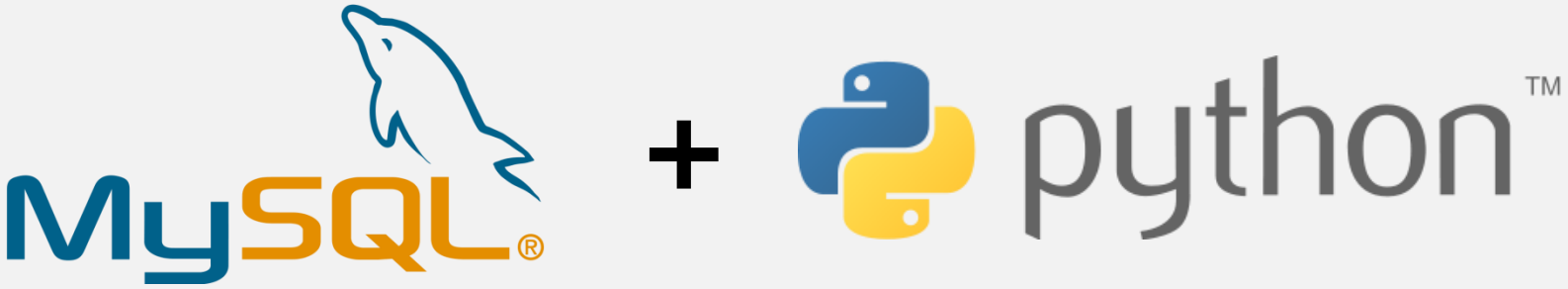
Banco de Dados
Aula 09
MySQL + Python

Dr. Diego Buchinger
diego.buchinger@udesc.br

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

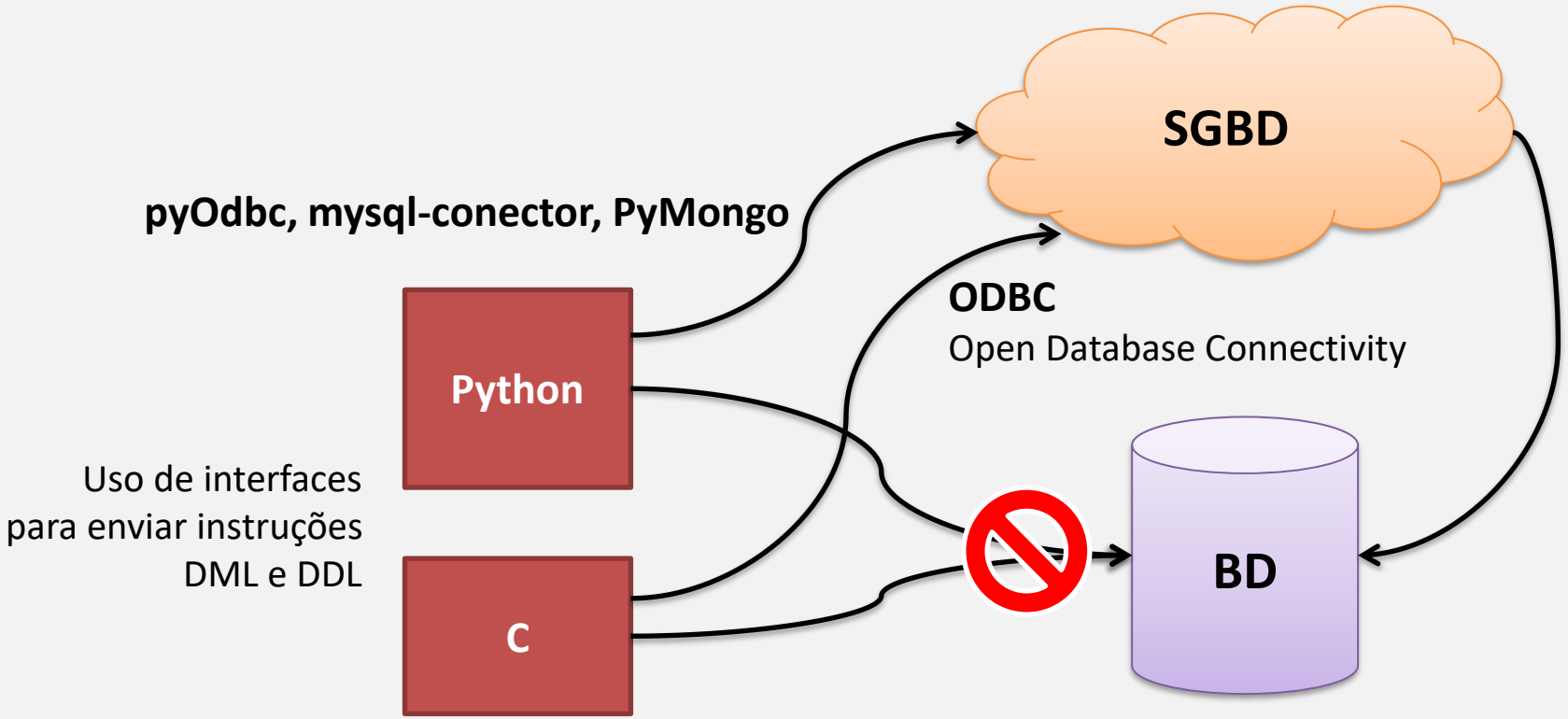
Introdução

Integrando MySQL em uma aplicação escrita em Python



Introdução

- Persistência de dados em aplicações
  - ~~Memória~~
  - Arquivo
  - Banco de Dados



Introdução

- **Instalando o conector**
  - Para utilizar o MySQL através do Python é preciso instalar um conector ou driver deste SGBD
  - Primeiro, atualizamos o sistema gerenciador de pacotes do python, o pip:
    - `python –m pip install --upgrade pip`
  - Na sequência, instalamos o conector
    - `python -m pip install mysql-connector-python`

Estes comandos devem ser executados no prompt de comando ou terminal!

Python + MySQL

- **Conectando-se ao Banco de Dados**
  - ❑ Com o conector instalado podemos importar a biblioteca de comandos de integração entre Python e MySQL
  - ❑ Veja como é fácil realizar a conexão e validá-la

```
import mysql.connector

conexao = mysql.connector.connect(
   host = "localhost",
   user = "nome de usuário",
   password = "senha usuário",
   database = "base de dados"
)

print(conexao)
if conexao.is_connected():
   print("Conectado no banco de dados")
else:
   print("Não foi possível conectar")
```

Python + MySQL

- **Realizando operações**
  - A maioria das operações utilizam um objeto cursor que é obtido através do objeto da conexão
  - Os comandos são executados através do objeto cursor e seu método execute()
  - Múltiplos resultados podem ser iterados e mostrados utilizando um laço de repetição for
  - Exemplo 1: mostrar as tabelas do banco de dados conectado
  - Exemplo 2: criar uma nova tabela

```
# ... conexão criada ...
cursor = conexao.cursor()
cursor.execute("show tables")
for tabela in cursor:
   print(tabela)
```

```
# ... conexão criada ...
cursor = conexao.cursor()
cursor.execute(
   "CREATE TABLE teste (\
      id INT AUTO_INCREMENT,\
      descricao VARCHAR(255),\
      PRIMARY KEY (id)\
   );"
)
```

Python + MySQL

- **Realizando operações**
  - ❑ As operações de seleção também utilizam o objeto cursor
  - ❑ Cada linha do resultado pode ser iterada através de uma repetição for e do método fetchall
  - ❑ Exemplo 1: mostrar id, nome e idade de todos os pacientes
  - ❑ Exemplo 2: mostrar o nome de cada especialidade exercida por pelo menos um médico e a quantidade de médicos que a exercem.

```
# ... conexão criada ...
cursor = conexao.cursor()
cursor.execute("
   SELECT codp, nome, idade
   FROM pacientes
")
result = cursor.fetchall():
for registro in result:
   print(registro)
```

```
# ... conexão criada ...
cursor = conexao.cursor()
cursor.execute("SELECT especialidade, \
   COUNT(especialidade) FROM medicos \
   GROUP BY (especialidade)
")
result = cursor.fetchall():
for registro in result:
   print(registro[0],"=>", registro[1])
```

Python + MySQL

- **Realizando operações**
  - ❑ As operações de inserção funcionam de modo similar, mas utilizam dois parâmetros:
    - A operação em sql com máscaras no lugar dos valores
    - Uma tupla de valores que substituem as máscaras
  - ❑ Ao final da operação deve ser executado o método commit
  - ❑ É possível verificar/mostrar quantos registros foram incluídos

```
# ... conexão criada ...
cursor = conexao.cursor()
sql = "INSERT INTO pacientes (codp, nome, idade, cpf) \
       VALUES (%s, %s, %s, %s)"
valores = (6, "Joao Aurélio", 45, "12345678901")
cursor.execute(sql, valores)
conexao.commit()
print(cursor.rowcount, "registro(s) inserido(s)")
```

Python + MySQL

- **Realizando operações**
  - ❑ As operações de atualização e remoção de registros funcionam de modo parecido com as inserções:

```
# ... conexão criada ...
cursor = conexao.cursor()
sql = "UPDATE pacientes SET nome = %s, idade = idade + 1 \
       WHERE codp = 6"
valores = ("Joao da Silva Sauro")
cursor.execute(sql, valores)
conexao.commit()
print(cursor.rowcount, "registro(s) afetado(s)")
```

```
# ... conexão criada ...
cursor = conexao.cursor()
cursor.execute("DELETE FROM pacientes WHERE codp = 6")
conexao.commit()
print(cursor.rowcount, "registro(s) removidos(s)")
```

Python + MySQL

- **Finalizando operações**
  - Após toda operação é muito importante fechar o objeto cursor com o método `close`.
  - Após o término de todas as operações desejadas é muito importante fechar a conexão utilizando seu objeto com o método `close` (ou `disconnect`).

```
# ... conexão criada ...
cursor = conexao.cursor()

# ... realiza uma ou várias operações ...

cursor.close()
conexao.close()
# conexao.disconnect
print("conexão com o MySQL fechada")
```

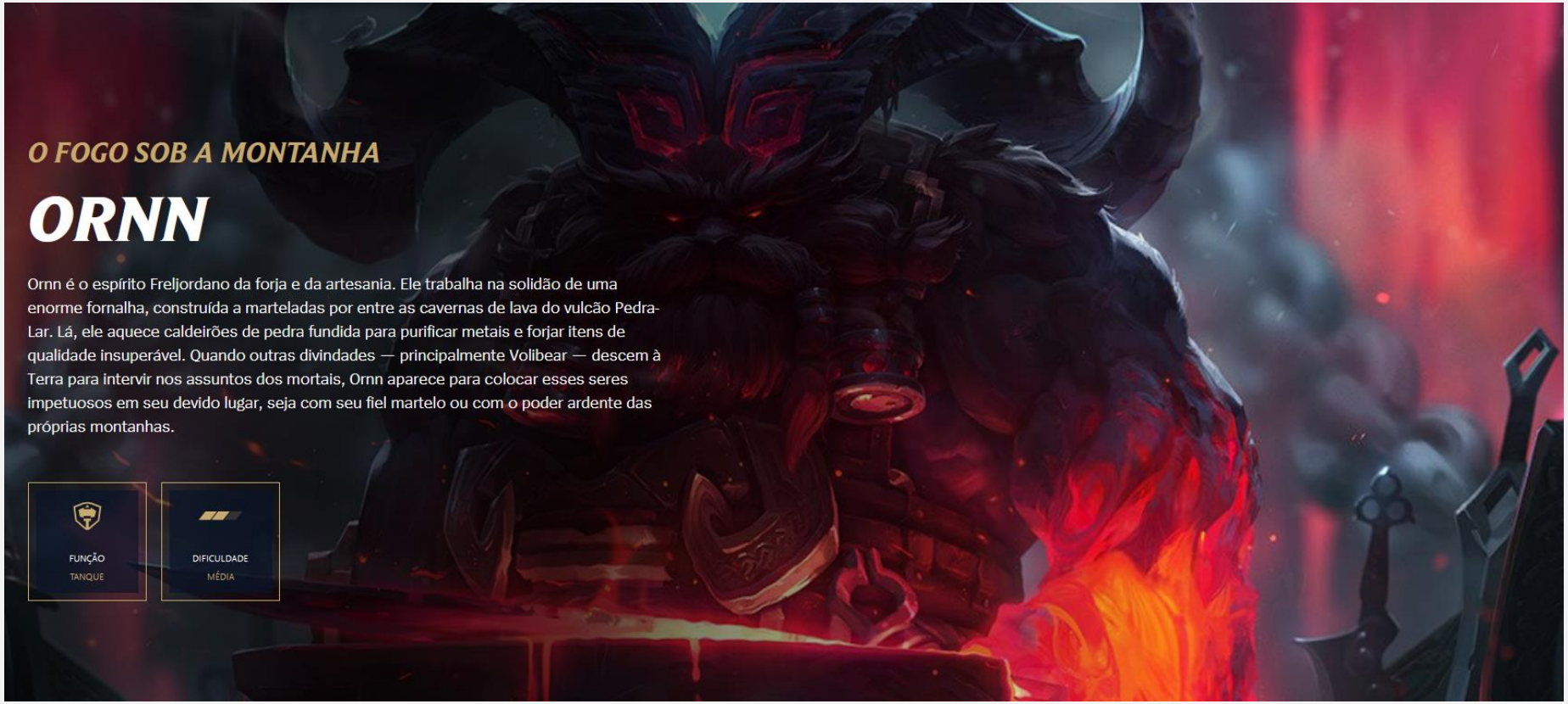
Atividades

Escreva um script em python para manipular os registros de pacientes do BD **Clinica**:

1. Crie uma classe BancoDados para gerenciar a conexão e as operações com o BD utilizando o usuário `suporte` (se a senha foi definida conforme orientado, ela deve ser: `$uport3CL1N1CA`).
2. Crie uma classe para representar os objetos de Pacientes.
3. Crie um método na classe BancoDados que consulta todos os pacientes da **Clinica**.
4. Crie um método na classe BancoDados que consulta um paciente específico da **Clinica**, através de seu código.
5. Crie um método na classe BancoDados que insere um novo paciente específico da **Clinica**, através de seu objeto.
6. Instancie um objeto do banco de dados e utilize os métodos criados

ORM - *Object Relational Mapping*

- A ponte entre orientação a objetos e Banco de Dados
- Cuidado, não confundir com ORNN



ORM - *Object Relational Mapping*

- A ponte entre orientação a objetos e Banco de Dados
- Prós:
  - ❑ Poupa tempo
  - ❑ Força uso de MVC deixando o código mais organizado
  - ❑ Abstrai SGBDs / simplifica migrações
- Contras:
  - ❑ Desempenho – algumas funcionalidades mais avançadas podem não ser tratadas corretamente pelos ORMs
  - ❑ Pode ser mais complicado aprender um ORM específico do que escrever suas próprias queries
  - ❑ Pode ser perigoso se o programador ficar dependente da ferramenta e não entender o que ela faz
- Exemplos:
  SQLAlchemy (python), Doctrini (PHP), Hibernate (Java)

ORM - *Object Relational Mapping*

- Instalação do SQLAlchemy
  - `python -m pip install sqlalchemy`
- Importações Principais:
  - `from sqlalchemy import create_engine, Column, String, Integer, DECIMAL, ForeignKey`
  - `from sqlalchemy.orm import declarative_base, sessionmaker, Relationship`
- Criando a conexão:
  - engine = create_engine("mysql+pymysql://USER:PASSWORD@localhost:3306/ DATABASE")
- Criando a classe base para as tabelas:
  - Base = declarative_base()
  - class MinhaClase(Base):

Este comandos deve ser executado no prompt de comando ou terminal!