

CHEGOU O MOMENTO DE FINALIZAR

PYTHON É A SUA **CONEXÃO COM O BANCO DE DADOS**



6

LISTA DE FIGURAS

Figura 1 – Logotipo Oracle SQL Developer	8
Figura 2 – Tela de configuração do Oracle	9
Figura 3 – Tela de instalação do cx_Oracle	10
Figura 4 – Tela de instalação do Pandas	11
Figura 5 – Teste que causou erro no programa maioria	13
Figura 6 – Visualização das colunas na tabela	28
Figura 7 – Visualização das linhas na tabela	29
Figura 8 – Visualização do contexto de banco de dados	30
Figura 9 – Visualização do Menu da Aplicação	31
Figura 10 – Status da tabela vazia	38
Figura 11 – Execução da rotina de cadastro	39
Figura 12 – Status da tabela depois do cadastro	40
Figura 13 – Execução da rotina de listagem dos registros	42
Figura 14 – Status dos dados gravados na Tabela	45
Figura 15 – Informação a ser editada	45
Figura 16 – Execução da rotina de Alteração	45
Figura 17 – Informação editada	46
Figura 18 – Informação não encontrada	46
Figura 19 – Visualização da linha que será excluída	48
Figura 20 – Execução da rotina de exclusão	49
Figura 21 – Visualização da exclusão da linha 4	49
Figura 22 – Visualização de todos os registros na tabela	50
Figura 23 – Visualização da confirmação da exclusão de todos os registros	51
Figura 24 – Visualização da tabela vazia	52

LISTA DE TABELAS

Tabela 1 – Conjunto de dados	27
Tabela 2 – Estrutura da tabela NOTAS_1A	29

EXEMPLO

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Programa (errado) que verifica se uma pessoa é maior de idade	13
Código-fonte 2 – Rotina de divisão de dois números	15
Código-fonte 3 – Sintaxe do try except	17
Código-fonte 4 – Execução try except na divisão de dois números	17
Código-fonte 5 – Execução try except com a captura da constante de erro na divisão de dois números	19
Código-fonte 6 – Programa com try except com a exibição da constante de erro na divisão de dois números	20
Código-fonte 7 – Rotina original para testar a divisão por zero	21
Código-fonte 8 – Tratamento do erro da divisão por zero	22
Código-fonte 9 – Sintaxe do try com o else	23
Código-fonte 10 – Aplicando o else no try	23
Código-fonte 11 – Aplicando o finally no try	24
Código-fonte 12 – Sintaxe completa do try	26
Código-fonte 13 – Script da tabela Petshop no Oracle	32
Código-fonte 14 – Rotina de conexão do servidor e usuário	34
Código-fonte 15 – Instanciando os cursores para fazer o CRUD	35
Código-fonte 16 – Apresentação do Menu e leitura da escolha do item	36
Código-fonte 17 – Rotina de cadastro	37
Código-fonte 18 – Rotina de listagem dos registros	41
Código-fonte 19 – Rotina de Alteração de um registro	44
Código-fonte 20 – Rotina de exclusão de um registro	47
Código-fonte 21 – Rotina de exclusão de todos os registros	51
Código-fonte 22 – Código fonte completo da aplicação	58

LISTA DE COMANDOS DE PROMPT DO SISTEMA OPERACIONAL

Comando de prompt 1 – Comando de instalação do cx_Oracle	9
Comando de prompt 2 – Comando de instalação do Pandas	10
Comando de prompt 3 – Execução assertiva da divisão de dois números	16
Comando de prompt 4 – Execução com valor errado da divisão de dois números...	16
Comando de prompt 5 – Execução com valor válido na divisão de dois números....	17
Comando de prompt 6 – Execução do erro tratado com try except da divisão de dois números	18
Comando de prompt 7 – Captura da constante de erro ValueError	18
Comando de prompt 8 – Execução com a captura da constante de erro ValueError	19
Comando de prompt 9 – Execução com a exibição da descrição da constante de erro ValueError.....	21
Comando de prompt 10 – Execução da rotina sem o tratamento de divisão por zero	22
Comando de prompt 11 – Execução da rotina com a exibição da divisão no else....	24
Comando de prompt 12 – Execução da rotina com sucesso e finally	25
Comando de prompt 13 – Execução da rotina sem sucesso e finally	25
Comando de prompt 14 – Apresentação do menu.....	36

SUMÁRIO

1 PYTHON E A SUA CONEXÃO COM O BANCO DE DADOS	7
1.1 Considerações	7
1.2 Pré-requisitos	7
1.3 Instalação dos Recursos	8
2 TRATAMENTO DE ERROS	11
2.1 Tipos de erros	12
2.1.1 Erro de Sintaxe	12
2.1.2 Erro de Lógica	13
2.1.3 Erro de Semântica	14
2.1.4 Erro circunstancial	14
2.1.5 Necessidade de tratar os erros	14
2.2 Tratamento de erros com o TRY	15
2.2.1 Except no Try	15
2.2.2 Except <ConstanteErro> no Try	18
2.2.3 Except exibindo o conteúdo do erro	20
2.2.4 Múltiplos excepts no mesmo try	21
2.2.5 Try com else	22
2.2.6 Try com finally	24
3 BANCO DE DADOS	26
3.1 Dado ou informação	27
3.2 Campos colunas	28
3.3 Registros linhas	28
3.4 Tabela	29
3.5 Banco de dados	30
4 O PROJETO	30
4.1 Tela principal do Projeto	31
4.2 Script do banco e tabela	32
5 CODIFICANDO O PROJETO	33
5.1 Conectando o servidor e o usuário	33
5.2 Apresentação do Menu	35
5.3 Cadastrar PET	37
5.4 Listar PETs	40
5.5 Alterar PET	43
5.6 Excluir PET	47
5.7 EXCLUIR TODOS OS PETS	50
5.8 Código-fonte da Aplicação	53
GLOSSÁRIO	59

1 PYTHON E A SUA CONEXÃO COM O BANCO DE DADOS

Chegou o grande momento! A partir de agora faremos a conexão do Python com o sistema gerenciador de banco de dados Oracle. Além da conexão com o Banco de dados, veremos as operações simples para manipular os registros de uma tabela.

1.1 Considerações

Para que este projeto funcione adequadamente, todas as ferramentas e complementos devem ser devidamente instalados na sua máquina e na IDE de sua preferência.

A ideia deste capítulo não é explorar o conceito de banco de dados, que na verdade esta é uma outra disciplina que vocês terão no próximo ano, o que usaremos são os conceitos suficientes para funcionar esta aplicação com o Python.

Como já trabalhamos consistência de dados (se é numérico ou não, em branco...) eu não aplicarei estes conceitos no projeto para que o foco seja apenas no novo aprendizado.

1.2 Pré-requisitos

Não basta digitarmos o código em Python, mandar executar e simplesmente tudo vai funcionar. Quando usamos ferramentas ou objetos externos (neste caso o Oracle Sql Developer), precisamos configurar o ambiente para que as ferramentas se comuniquem adequadamente.

Os pré-requisitos necessários na sua máquina para que tudo funcione sem problemas são:

- O sistema gerenciador de banco de dados Oracle SQL Developer;
- Instalar no Python o complemento 'cx_Oracle'. Ele tem a função de fazer com que o interpretador Python se comunique com o Oracle;

- Instalar no Python o complemento 'Pandas'. A sua função é a facilidade para uma melhor organização e apresentação dos dados capturados do Oracle e apresentados no Python.

1.3 Instalação dos Recursos

Primeiramente vamos precisar de ter na nossa máquina o software 'Oracle Sql Developer'.



Figura 1 – Logotipo Oracle SQL Developer
Fonte: Oracle (2022)

Link para download:

<https://www.oracle.com/tools/downloads/sqldev-downloads.html>

Depois de instalado o Oracle, precisamos ter as permissões - servidor, usuário e senha - devidamente cadastradas no provedor Oracle. Cadastre as suas permissões com as identificações de aluno FIAP.

Python e a sua conexão com o Banco de dados

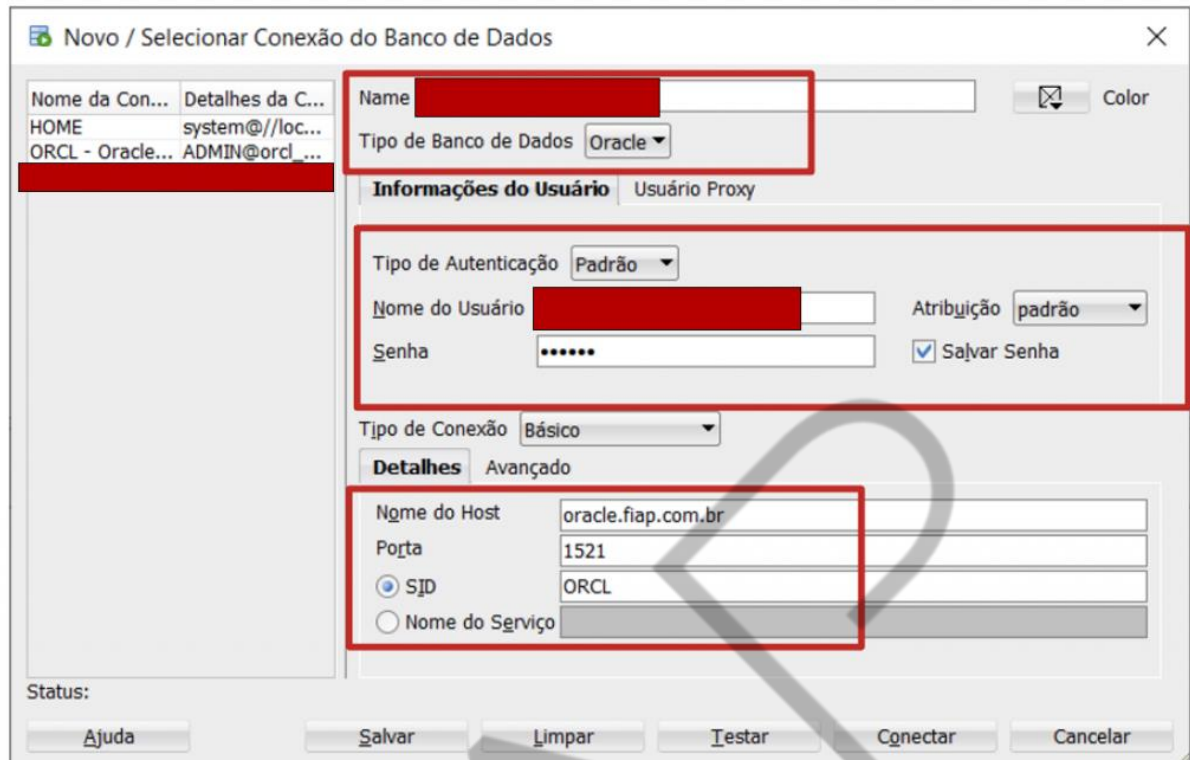


Figura 2 – Tela de configuração do Oracle
Fonte: Elaborado pelo autor (2023)
Os dados de acesso são de responsabilidade do usuário

No Python, devemos instalar via terminal o complemento 'cx_Oracle' com a seguinte linha de comando:

PROMPT:

```
pip install cx Oracle
```

Comando de prompt 1 – Comando de instalação do cx_Oracle
Fonte: Elaborado pelo autor (2023)

Desta forma o complemento será instalado na aplicação. Aparecerá algo semelhante a imagem abaixo:

Python e a sua conexão com o Banco de dados

```
(venv) MacBook-Air-de-Edson:pythonFontes2023 edsondeoliveira$ pip install cx_Oracle
Collecting cx_Oracle
  Using cached cx_Oracle-8.3.0.tar.gz (363 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: cx_Oracle
  Building wheel for cx_Oracle (pyproject.toml) ... done
  Created wheel for cx_Oracle: filename=cx_Oracle-8.3.0-cp311-cp311-macosx_10_9_universal2.whl size=376472 sha256=7285d1913576f9d57c11dd579703fef318551ec0e885d0fbac7a84942a389e05
  Stored in directory: /Users/edsondeoliveira/Library/Caches/pip/wheels/76/e2/f3/0bec6af62fe70c9c937eeeba545dc840dbaca94114363177a8
Successfully built cx_Oracle
Installing collected packages: cx_Oracle
Successfully installed cx_Oracle-8.3.0

[notice] A new release of pip available: 22.3.1 -> 23.0
[notice] To update, run: pip install --upgrade pip
(venv) MacBook-Air-de-Edson:pythonFontes2023 edsondeoliveira$
```

Figura 3 – Tela de instalação do cx_Oracle
Fonte: Elaborado pelo autor (2023)

Outro complemento que deve ser instalado no Python é o 'Pandas'. Digite a linha de comando:

PROMPT:

```
pip install pandas
```

Comando de prompt 2 – Comando de instalação do Pandas
Fonte: Elaborado pelo autor (2023)

Da mesma forma o complemento será instalado no Python dentro da aplicação que vamos criar:

```
(venv) MacBook-Air-de-Edson:pythonFontes2023 edsondeoliveira$ pip install pandas
Collecting pandas
  Downloading pandas-1.5.3-cp311-cp311-macosx_10_9_x86_64.whl (11.9 MB)
    11.9/11.9 MB 6.1 MB/s eta 0:00:00
Collecting python-dateutil>=2.8.1
  Using cached python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
Collecting pytz>=2020.1
  Using cached pytz-2022.7.1-py2.py3-none-any.whl (499 kB)
Collecting numpy>=1.21.0
  Downloading numpy-1.24.2-cp311-cp311-macosx_10_9_x86_64.whl (19.8 MB)
    19.8/19.8 MB 6.3 MB/s eta 0:00:00
Collecting six>=1.5
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: pytz, six, numpy, python-dateutil, pandas
Successfully installed numpy-1.24.2 pandas-1.5.3 python-dateutil-2.8.2 pytz-2022.7.1 six-1.16.0

[notice] A new release of pip available: 22.3.1 -> 23.0
[notice] To update, run: pip install --upgrade pip
(venv) MacBook-Air-de-Edson:pythonFontes2023 edsondeoliveira$
```

Figura 4 – Tela de instalação do Pandas
Fonte: Elaborado pelo autor (2023)

2 TRATAMENTO DE ERROS

Toda aplicação está sujeita a erros em sua execução, seja por culpa do usuário, do programador ou das circunstâncias que rondam a aplicação. Ainda mais quando estamos trabalhando com conexões externas à aplicação, como é o caso de conexão com banco de dados. Logo, devemos ter um cuidado extra.

Por isso, antes de entrarmos no projeto, vamos falar um pouco sobre o tratamento de erros. Feliz o programador que testa bem a sua aplicação antes de colocá-la em produção.

Colocar aplicação em Produção: quando a aplicação já está disponível ao usuário final.

Podemos dizer também que feliz a empresa que, além do próprio DEV, tem um profissional de teste (QA – Quality Assurance), porque este é treinado e tem habilidades específicas para encontrar falhas em sistemas através de técnicas como caixa branca (poder manipular internamente os componentes e códigos do sistema), caixa preta (modo funcional, como um usuário do sistema) ou testes automatizados (em que um código é desenvolvido para testar a aplicação).

Não importa em qual camada estejamos analisando, existem tipos de erros e estes erros devem ser tratados para que não eclodam nas mãos do usuário.

2.1 Tipos de erros

Existem vários tipos de erros, mas vou citar aqui apenas os mais importantes para uma melhor experiência neste capítulo.

2.1.1 Erro de Sintaxe

Este é o clássico erro para aquele que ainda é novo em programação ou está aprendendo uma nova linguagem.

O erro de compilação é aquele em que é desrespeitada alguma escrita ou regra do comando.

Vamos a alguns exemplos.

Se digitarmos a linha de código: `Print("Isso é Fiap")`, aparecerá o erro ***NameError: name 'Print' is not defined. Did you mean: 'print'?***, em outras palavras o interpretador quer dizer que o comando `Print` não existe e ainda sugere o nome correto que é o `'print'`.

Porque isso se o `print` existe? Na verdade o `'P'` do `print` está em maiúscula e a regra do Python é que todos os comandos sejam escritos totalmente com letras minúsculas. Logo, o correto é `print("Isso é Fiap")`.

Outro exemplo, vamos utilizar a mesma instrução. Se o programador digitar `print("Isso é Fiap)`, ocorrerá outro erro de compilação: ***SyntaxError: unterminated string literal (detected at line 1)***, em outras palavras a string não foi terminada, ou seja, as aspas da mensagem foram abertas, mas não fechadas.

O erro de sintaxe sempre fará execução da aplicação abortar (terminar e voltar ao sistema operacional).

2.1.2 Erro de Lógica

O erro de lógica é o tipo de erro no qual o programa não é abortado, mas os valores apresentados estão errados.

Por exemplo, o código abaixo informa se a pessoa é maior de idade ou não (considerando ao menos 18 anos para ser maior de idade):

PYTHON:

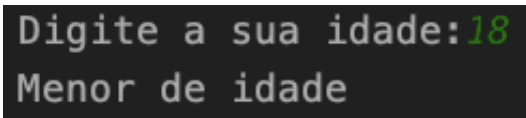
```
idade = int(input("Digite a sua idade:"))  
if idade > 18:  
    print("Maior de idade")  
else:  
    print("Menor de idade")
```

Código-fonte 1 – Programa (errado) que verifica se uma pessoa é maior de idade

Fonte: Elaborado pelo autor (2023)

Não há erro de compilação, o programa ‘funcionou’, mas, se o usuário digitar a idade 18, veja a resposta que será dada:

EXECUÇÃO:



```
Digite a sua idade:18  
Menor de idade
```

Figura 5 – Teste que causou erro no programa maioridade

Fonte: Elaborado pelo autor (2023)

Menor de idade? 18 não é maior de idade? Sim! Este programa está com erro de lógica, ou seja, o usuário não pensou direito, ele deveria ter colocado o = ao lado do sinal de > para incluir o 18 como maior de idade.

O if deveria ser `if idade >= 18`.

2.1.3 Erro de Semântica

O erro de Semântica ocorre quando não há erro de compilação, ou seja, a linha está escrita corretamente, mas eventualmente o programa pode dar um erro que não é o de lógica.

Por exemplo, o comando `arq = open("arquivo.txt", "r")` está escrito corretamente, certo? Ele abre o `arquivo.txt` para leitura. Caso o `arquivo.txt` esteja no disco, ele será aberto e segue o fluxo da aplicação, mas se o arquivo não estiver no disco ocorrerá o erro `Traceback (most recent call last): File "testes.py", line 1, in <module> arq = open("arquivo.txt", "r") FileNotFoundError: [Errno 2] No such file or directory: 'arquivo.txt'`, em outras palavras foi dito que o `arquivo.txt` não foi encontrado no disco.

2.1.4 Erro circunstancial

O erro circunstancial ocorre quando fatos alheios à aplicação ocorrem, fatos estes que não dependem da ação do usuário ou do DEV. Podemos citar como a perda da conexão com o servidor seja por falta de energia ou internet, delay de acesso, falha da internet do lado do usuário, banco de dados aberto em modo exclusivo pelo administrador entre outros erros diversos.

2.1.5 Necessidade de tratar os erros

Não importa o erro, até o momento, sempre os tratamos com camadas de `ifs` acompanhado de funções ou métodos como: `isdigit()`, `isnumeric()`, `isempty()`, `isdate()`, entre outros.

Até os erros de semântica era tranquilo tratar os erros desta forma, mas, a partir dos erros circunstanciais, fica difícil prever o erro com os `ifs`. Então, neste tópico aprenderemos a usar o `try`, uma estrutura específica para tratar os erros.

2.2 Tratamento de erros com o TRY

A grande vantagem de utilizarmos o TRY ao invés de ifs encadeados é o fato de que, independentemente do erro, o programa não é abortado. Em caso de erro, o fluxo do programa continua operando normalmente. Desta forma, ele atende erros previstos e imprevistos.

No Python há complementos dentro do TRY, são eles: except, else e finally. Utilizaremos a mesma problemática para explicarmos este comando e seus complementos.

Basicamente o comando Try executa uma rotina e dependendo da falha que ocorrer com esta rotina, ele aciona os seus complementos.

Este comando é muito utilizado em situações em que a problemática envolva arquivos e banco de dados por ambos poderem retornar uma infinidade de erros circunstanciais.

2.2.1 Except no Try

O except é acionado todas as vezes que houver um erro na rotina. Para exemplificarmos, vamos considerar a problemática de pegar dois valores e efetuar a divisão de um pelo outro.

Primeiramente, vamos analisar a rotina:

PYTHON:

```
print("Cálculo da divisão: ")
valor1 = float(input("Digite o valor 1: "))
valor2 = float(input("Digite o valor 2: "))
divisao = valor1 / valor2
print(f"Divisão: {divisao}")
```

Código-fonte 2 – Rotina de divisão de dois números
Fonte: Elaborado pelo autor (2023)

Python e a sua conexão com o Banco de dados

Esta rotina recebe dois valores do usuário e faz o casting dos dados digitados para float, haja vista que o input captura informações str, efetua a divisão e exibe o resultado.

Executando esta rotina sem valores errados, ela funcionará normalmente, por exemplo:

PROMPT:

```
Cálculo da divisão:  
Digite o valor 1: 10  
Digite o valor 2: 4  
Divisão: 2.5
```

Comando de prompt 3 – Execução assertiva da divisão de dois números
Fonte: Elaborado pelo autor (2023)

Nada de anormal ocorreu, ele dividiu 10 por 4 e resultou 2.5. Vamos sugerir um erro. Ao invés de digitarmos um número, vamos digitar 'A' – que é um valor não numérico – na variável `valor2`:

PROMPT:

```
Cálculo da divisão:  
Digite o valor 1: 10  
Digite o valor 2: A  
Traceback (most recent call last):  
  File "testes.py", line 3, in <module>  
    valor2 = float(input("Digite o valor 2: "))  
ValueError: could not convert string to float: 'A'
```

Comando de prompt 4 – Execução com valor errado da divisão de dois números
Fonte: Elaborado pelo autor (2023)

Repare que a execução do programa foi abortada e apareceu um erro: **ValueError: could not convert string to float: 'A'**. que quer dizer que não foi possível converter o dado 'A' para string para poder fazer a conta.

Pelo erro não ter sido tratado a execução foi abortada. Vamos tratá-lo com o try except. Antes, vejamos a sintaxe do try com o except:

PYTHON:

```
try:
    <ROTINA                que                será                executada>
except:
    <executa este complemento caso ocorra um erro na ROTINA>
```

Código-fonte 3 – Sintaxe do try except
Fonte: Elaborado pelo autor (2023)

A regra é simples, logo após o try colocamos a rotina que desejamos executar, caso ocorra um erro o fluxo do programa executará o except.

Vamos implementar o try na nossa rotina:

PYTHON:

```
try:
    print("Calculo da divisão: ")
    valor1 = float(input("Digite o valor 1: "))
    valor2 = float(input("Digite o valor 2: "))
    divisao = valor1 / valor2
    print(f"Divisão: {divisao}")
except:
    print("Ocorreu algum erro")
```

Código-fonte 4 – Execução try except na divisão de dois números
Fonte: Elaborado pelo autor (2023)

A rotina fica logo após o try: e qualquer erro que ocorra ele executa os comandos que estão dentro do except, neste caso, o print.

Agora vamos testar, na primeira execução, vamos colocar dados sem erros:

PROMPT:

```
Cálculo da divisão:
Digite o valor 1: 35
Digite o valor 2: 7
Divisão: 5.0
```

Comando de prompt 5 – Execução com valor válido na divisão de dois números
Fonte: Elaborado pelo autor (2023)

Nada de anormal ocorreu, a rotina foi executada normalmente.

Agora vamos testar digitando um valor não numérico na variável `valor2`:

PROMPT:

```
Cálculo da divisão:  
Digite o valor 1: 23  
Digite o valor 2: B  
Ocorreu algum erro
```

Comando de prompt 6 – Execução do erro tratado com `try except` da divisão de dois números
Fonte: Elaborado pelo autor (2023)

Na variável `valor 2` inserimos a letra 'B', apesar de ser um erro a execução do programa não foi terminada. Assim apareceu a mensagem de erro "Ocorreu algum erro", mas o programa continuou sendo executado.

2.2.2 Except <ConstanteErro> no Try

Temos como melhorar a mensagem de erro deixando-a mais específica, pois o erro apresentado foi genérico. Vamos aprender como capturar o erro pela sua constante.

Todas as constantes de erro têm nomes e descrições específicas determinada pelo interpretador.

Repare que na primeira execução, a constante que representa o erro foi anunciada:

PROMPT:

```
Traceback (most recent call last):  
  File "testes.py", line 3, in <module>  
    valor2 = float(input("Digite o valor 2: "))  
ValueError: could not convert string to float: 'A'
```

Comando de prompt 7 – Captura da constante de erro `ValueError`
Fonte: Elaborado pelo autor (2023)

Python e a sua conexão com o Banco de dados

O nome da constante deste erro é **ValueError** e a sua descrição é *could not convert string to float: 'A'*; em outras palavras, houve erro no dado informado e consequentemente a conversão de 'A' para float() não pode ser executada.

Vamos então tratar este erro específico:

PYTHON:

```
try:
    print("Calculo da divisão: ")
    valor1 = float(input("Digite o valor 1: "))
    valor2 = float(input("Digite o valor 2: "))
    divisao = valor1 / valor2
    print(f"Divisão: {divisao}")
except ValueError:
    print("ERRO! Digite um valor numérico")
except:
    print("Ocorreu algum erro")
```

Código-fonte 5 – Execução try except com a captura da constante de erro na divisão de dois números
Fonte: Elaborado pelo autor (2023)

Na rotina acrescentamos `except ValueError:` caso ocorra o erro de valor, o fluxo entrará nesta constante específica e, consequentemente, exibirá a mensagem “ERRO! Digite um valor numérico” (mensagem definida pelo programador) e caso ocorra um erro diferente deste, entrará no `except:` genérico e mostrará “Ocorreu algum erro”.

Vamos simular o erro de valor:

PROMPT:

```
Cálculo da divisão:
Digite o valor 1: 234
Digite o valor 2: @
ERRO! Digite um valor numérico
```

Comando de prompt 8 – Execução com a captura da constante de erro ValueError
Fonte: Elaborado pelo autor (2023)

Em valor 2 foi digitado '@' e a mensagem de erro foi específica: "Erro, digite um valor numérico". Em outro tipo de erro, seria apresentado "Ocorreu um erro" e, caso os valores fossem coerentes, ele faria o cálculo da divisão normalmente.

2.2.3 Except exibindo o conteúdo do erro

Outra possibilidade é, ao invés de exibirmos uma mensagem de erro personalizada, exibir a mensagem da própria constante de erro. A vantagem está que o erro é informado sem a necessidade de formularmos uma mensagem específica. A desvantagem é que a mensagem de erro aparece em inglês e isso pode confundir o usuário.

Vejamos a implementação:

PYTHON:

```
try:
    print("Calculo da divisão: ")
    valor1 = float(input("Digite o valor 1: "))
    valor2 = float(input("Digite o valor 2: "))
    divisao = valor1 / valor2
    print(f"Divisão: {divisao}")
except ValueError as erroValorErrado:
    print(erroValorErrado)
except:
    print("Ocorreu algum erro")
```

Código-fonte 6 – Programa com try except com a exibição da constante de erro na divisão de dois números

Fonte: Elaborado pelo autor (2023)

Na linha de código `except ValueError as erroValorErrado:` criamos um aliás (com o 'as') e demos um nome ao erro (`erroValorErrado`).

O nome do erro é criado pelo programador com as mesmas regras para nomearmos variáveis.

Assim, caso ocorra o erro, será exibido através do print o alias que conterà a descrição original do erro. Note a diferença:

PROMPT:

```
Cálculo da divisão:  
Digite o valor 1: #  
could not convert string to float: '#'
```

Comando de prompt 9 – Execução com a exibição da descrição da constante de erro ValueError
Fonte: Elaborado pelo autor (2023)

Depois de digitado '#' em valor 1, a mensagem apresentada será: `could not convert string to float: '#'`. Esta é a mensagem original do erro.

2.2.4 Múltiplos excepts no mesmo try

Tratamos o erro de valor, mas o diferencial do try é que ele pode tratar diversos erros diferentes.

Sabemos que um outro erro que pode existir nesta rotina é o usuário digitar 0 (zero) no segundo valor e causar uma 'divisão por zero', algo que não é possível na matemática nem em um cálculo no computador. Inicialmente, vamos simular este erro com a rotina sem o try:

PYTHON:

```
print("Cálculo da divisão: ")  
valor1 = float(input("Digite o valor 1: "))  
valor2 = float(input("Digite o valor 2: "))  
divisao = valor1 / valor2  
print(f"Divisão: {divisao}")
```

Código-fonte 7 – Rotina original para testar a divisão por zero
Fonte: Elaborado pelo autor (2023)

Caso seja digitado o valor 0 em valor 2, veja o erro que despertará:

PROMPT:

```
Cálculo da divisão:
Digite o valor 1: 10
Digite o valor 2: 0
Traceback (most recent call last):
  File "testes.py", line 4, in <module>
    divisao = valor1 / valor2
ZeroDivisionError: float division by zero
```

Comando de prompt 10 – Execução da rotina sem o tratamento de divisão por zero
Fonte: Elaborado pelo autor (2023)

A constante do erro é **ZeroDivisionError** e a sua descrição é float Division by zero. Para inserirmos no try fica fácil! Basta anotarmos a constante de erro **ZeroDivisionError** e a inserirmos em um novo except:

PYTHON:

```
try:
    print("Calculo da divisão: ")
    valor1 = float(input("Digite o valor 1: "))
    valor2 = float(input("Digite o valor 2: "))
    divisao = valor1 / valor2
    print(f"Divisão: {divisao}")
except ValueError as erroValorErrado:
    print(erroValorErrado)
except ZeroDivisionError:
    print("ERRO! Não há divisão por zero.")
except:
    print("Ocorreu algum erro")
```

Código-fonte 8 – Tratamento do erro da divisão por zero
Fonte: Elaborado pelo autor (2023)

Há uma nova mensagem para este novo erro de divisão por zero. Caso seja encontrado um terceiro erro, o fluxo do programa cairá no `except`:

2.2.5 Try com else

O else em um if tem a característica de negação da condição. O else no while ou for é executado caso não haja interrupção incondicional (com break) no laço. O else no

try é mais parecido com a segunda situação, ou seja, caso não ocorra nenhum erro na rotina do try, o else é executado. Sua sintaxe é:

PYTHON:

```
try:
    <Rotina>
except:
    <Se houver erro>
else:
    <Se não houver erro>
```

Código-fonte 9 – Sintaxe do try com o else
Fonte: Elaborado pelo autor (2023)

Vejam esta adição na nossa problemática:

PYTHON:

```
try:
    print("Calculo da divisão: ")
    valor1 = float(input("Digite o valor 1: "))
    valor2 = float(input("Digite o valor 2: "))
    divisao = valor1 / valor2
except ValueError as erroValorErrado:
    print(erroValorErrado)
except ZeroDivisionError:
    print("ERRO! Não há divisão por zero.")
except:
    print("Ocorreu algum erro")
else:
    # Caso não ocorra erro, exibe o resultado da divisão
    print(f"Divisão: {divisao}")
```

Código-fonte 10 – Aplicando o else no try
Fonte: Autor - 2023

Concordam que a exibição do resultado da divisão só deve ser exibida caso não haja erro? Então, modificamos a linha: `print(f"Divisão: {divisao}")`. Colocamos esta linha dentro do `else:`, em outras palavras, será exibido o resultado da divisão caso não ocorra erro.

PROMPT:

```
Cálculo da divisão:  
Digite o valor 1: 12  
Digite o valor 2: 6  
Divisão: 2.0
```

Comando de prompt 11 – Execução da rotina com a exibição da divisão no else
Fonte: Elaborado pelo autor (2023)

2.2.6 Try com finally

O `except` trata os erros, o `else` trata o **não** erro, agora o complemento `finally` do `try` simplesmente significa “Execute o `finally` independentemente de existir erro ou não”, em outras palavras, é a finalização da estrutura `try`.

Um exemplo da aplicação do `finally`: trabalhando com arquivos, independentemente de existir ou não erros, o arquivo deve ser fechado com `close()`, então poderíamos implementar o `finally` para fechar o arquivo. Vamos adaptar o `finally` na nossa problemática:

PYTHON:

```
try:  
    print("Calculo da divisão: ")  
    valor1 = float(input("Digite o valor 1: "))  
    valor2 = float(input("Digite o valor 2: "))  
    divisao = valor1 / valor2  
except ValueError as erroValorErrado:  
    print(erroValorErrado)  
except ZeroDivisionError:  
    print("ERRO! Não há divisão por zero.")  
except:  
    print("Ocorreu algum erro")  
else:  
    # Caso não ocorra erro, exibe o resultado da divisão  
    print(f"Divisão: {divisao}")  
finally:  
    # Executa independentemente da rotina ter erros ou não  
    print("Finalizada a operação de divisão!")
```

Código-fonte 11 – Aplicando o `finally` no `try`
Fonte: Elaborado pelo autor (2023)

Python e a sua conexão com o Banco de dados

Vamos fazer o primeiro teste neste código com valores coerentes:

PROMPT:

```
Cálculo da divisão:  
Digite o valor 1: 15  
Digite o valor 2: 9  
Divisão: 1.6666666666666667  
Finalizada a operação de divisão!
```

Comando de prompt 12 – Execução da rotina com sucesso e finally
Fonte: Elaborado pelo autor (2023)

Dividindo 15 por 9 resultou 1.66, ou seja, foi possível executar o cálculo, não apareceram mensagens de erros, o `else` foi executado exibindo a mensagem `Divisão: 1.6666666666666667` e o `finally` fechou a estrutura com a mensagem `Finalizada a operação de divisão!` Agora vamos testar os valores com erros de divisão por zero.

PROMPT:

```
Cálculo da divisão:  
Digite o valor 1: 23  
Digite o valor 2: 0  
ERRO! Não há divisão por zero.  
Finalizada a operação de divisão!
```

Comando de prompt 13 – Execução da rotina sem sucesso e finally
Fonte: Elaborado pelo autor (2023)

Nesta execução, depois de digitado o valor 0 (zero) em valor 2, acarretou o `ERRO! Não há divisão por zero.`, mesmo assim o `finally` foi executado exibindo a mensagem `Finalizada a operação de divisão!`

Depois de vista a estrutura de erros `try` e todos os seus complementos, fica assim a sintaxe completa:

PYTHON:

```
try:
    <ROTINA que será executada>
except <constanteErro>:
    <caso ocorra o erro específico 1>
except <constanteErro> as <nomeErro>:
    <caso ocorra o erro específico 2>
else:
    <caso não ocorra erro>
finally:
    <executa com ou sem erros>
```

Código-fonte 12 – Sintaxe completa do try
Fonte: Elaborado pelo autor (2023)

Com o aprendizado de tratamento de erros, a nossa aplicação estará mais fortalecida no que diz respeito a falhas sistêmicas.

3 BANCO DE DADOS

Este capítulo é um spoiler sobre uma das novas disciplinas que vocês terão no próximo ano letivo: Banco de dados.

Depois de evoluirmos no aprendizado de programação, no qual todos os conceitos fundamentais foram vistos e trabalhados, vale agora pensar em organizar, salvar e recuperar os dados fornecidos pelo usuário de forma prática: este é o conceito de um sistema gerenciador de banco de dados.

Como dito, falaremos neste tópico o mínimo para você aprender a fazer a conexão com o Oracle (SGBD), afinal vocês terão este assunto em uma disciplina e se você for visualizar de forma macro há até cursos de Banco de dados em faculdades.

3.1 Dado ou informação

Já li vários autores e vi vários professores falando sobre este tema: “Diferença de dado e informação”. Humildemente, por não ser a minha área, vou descrever a minha visão que pelo menos ajudará os leigos a entenderem esta temática. Veja a planilha abaixo:

RA	NOME	NOTA
23141	ANDREA BEATRIZ MARINI SILVA	4,0
23147	ARTUR DINIZ TORRES	8,0
22153	BARBARA DE MELO	9,0
23136	BRUNO SALES	10,0
23152	CARLOS HENRIQUE DA SILVA	7,0
23103	CAMILA FERNANDES	5,0
23125	CATARINA OLIVEIRA	6,0
23173	CAUE SOBRINHO	4,0
23147	CRISTIANO FILHO	6,0
23151	DAVI LEMOS	7,0
18060	FABIANO NOBREGA CANDIDO	8,0
19136	FELIPE MELO	8,0

Tabela 1 – Conjunto de dados
Fonte: Elaborado pelo autor (2023)

Esta planilha está completada com DADOS.

Se quisermos saber a nota que o aluno CRISTIANO tirou, teríamos que passar a informação CRISTIANO FILHO e o valor resultante seria a INFORMAÇÃO 6,0.

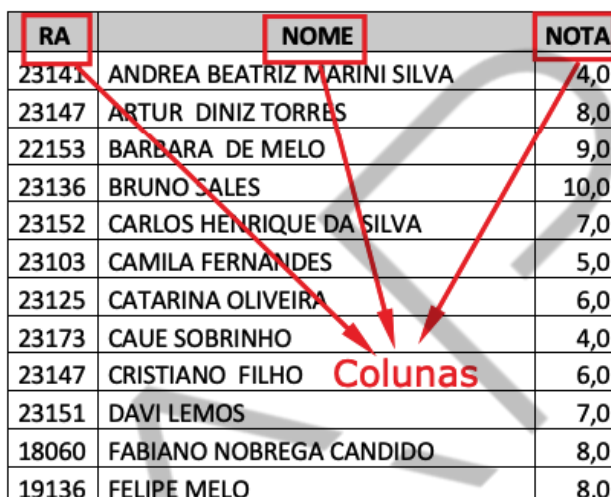
Outro caso, qual o RA do aluno FELIPE MELO? O nome FELIPE MELO é uma informação de referência e o 19136 é a INFORMAÇÃO resultante.

Mais um exemplo, mostre o RM dos alunos que tiraram 6,0, o resultado será as INFORMAÇÕES: 23125 e 23147.

Resumidamente, DADOS é o universo com todos os dados armazenados e INFORMAÇÃO é um subconjunto deste universo que será útil para algum propósito do usuário na manipulação do sistema.

3.2 Campos | colunas

Utilizando a mesma imagem temos os campos ou colunas. Eles dão o nome a mesma classificação de informações, neste caso RA, Nome e Nota. Ou seja, dentro da coluna RA só existem dados armazenados correspondentes a RA e assim sucessivamente com os demais campos.



RA	NOME	NOTA
23141	ANDREA BEATRIZ MARINI SILVA	4,0
23147	ARTUR DINIZ TORRES	8,0
22153	BARBARA DE MELO	9,0
23136	BRUNO SALES	10,0
23152	CARLOS HENRIQUE DA SILVA	7,0
23103	CAMILA FERNANDES	5,0
23125	CATARINA OLIVEIRA	6,0
23173	CAUE SOBRINHO	4,0
23147	CRISTIANO FILHO	6,0
23151	DAVI LEMOS	7,0
18060	FABIANO NOBREGA CANDIDO	8,0
19136	FELIPE MELO	8,0

Figura 6 – Visualização das colunas na tabela
Fonte: Elaborado pelo autor (2023)

Cada coluna tem um **nome** constante e tem um **tipo** (como ocorre em variáveis: float, int, str...). Dentro de cada coluna são inseridos dados relacionados somente àquela coluna: como no caso de NOTA, dentro desta coluna há somente números float.

3.3 Registros | linhas

Os registros são conhecidos como as informações de uma linha em sua totalidade, ou seja, absorvendo um dado de cada campo.

Observe a imagem:

RA	NOME	NOTA
23141	ANDREA BEATRIZ MARINI SILVA	4,0
23147	ARTUR DINIZ TORRES	8,0
22153	BARBARA DE MELO	9,0
23136	BRUNO SALES	10,0
23152	CARLOS HENRIQUE DA SILVA	7,0
23103	CAMILA FERNANDES	5,0
23125	CATARINA OLIVEIRA	6,0
23173	CAUE SOBRINHO	4,0
23147	CRISTIANO FILHO	6,0
23151	DAVI LEMOS	7,0
18060	FABIANO NOBREGA CANDIDO	8,0
19136	FELIPE MELO	8,0

Figura 7 – Visualização das linhas na tabela
Fonte: Elaborado pelo autor (2023)

O contexto das informações relacionadas ao CARLOS HENRIQUE DA SILVA em sua totalidade, como o RA 23152 e a nota 7,0 é caracterizada como um registro, ou seja, todas as informações da mesma entidade.

3.4 Tabela

Tabela é o conjunto de todas estas informações. Em outras palavras, a relação de todas as linhas e colunas. Toda tabela tem uma estrutura para o armazenamento dos dados.

Toda tabela deve ter um nome que a diferencie das outras tabelas do banco de dados, por exemplo, para a tabela abaixo, poderíamos dar o nome de NOTAS_1A. A estrutura desta tabela é:

Tabela: NOTAS_1A.

CAMPO	TIPO	TAMANHO
RA	Int	-
Nome	VarChar	50
Nota	Float	-

Tabela 2 – Estrutura da tabela NOTAS_1A
Fonte: Elaborado pelo autor (2023)

3.5 Banco de dados

Conhecendo os conceitos citados, agora fica fácil entender o que é um banco de dados. Um banco de dados nada mais é do que o acúmulo de várias tabelas que podem ser alimentadas pelo mesmo sistema.

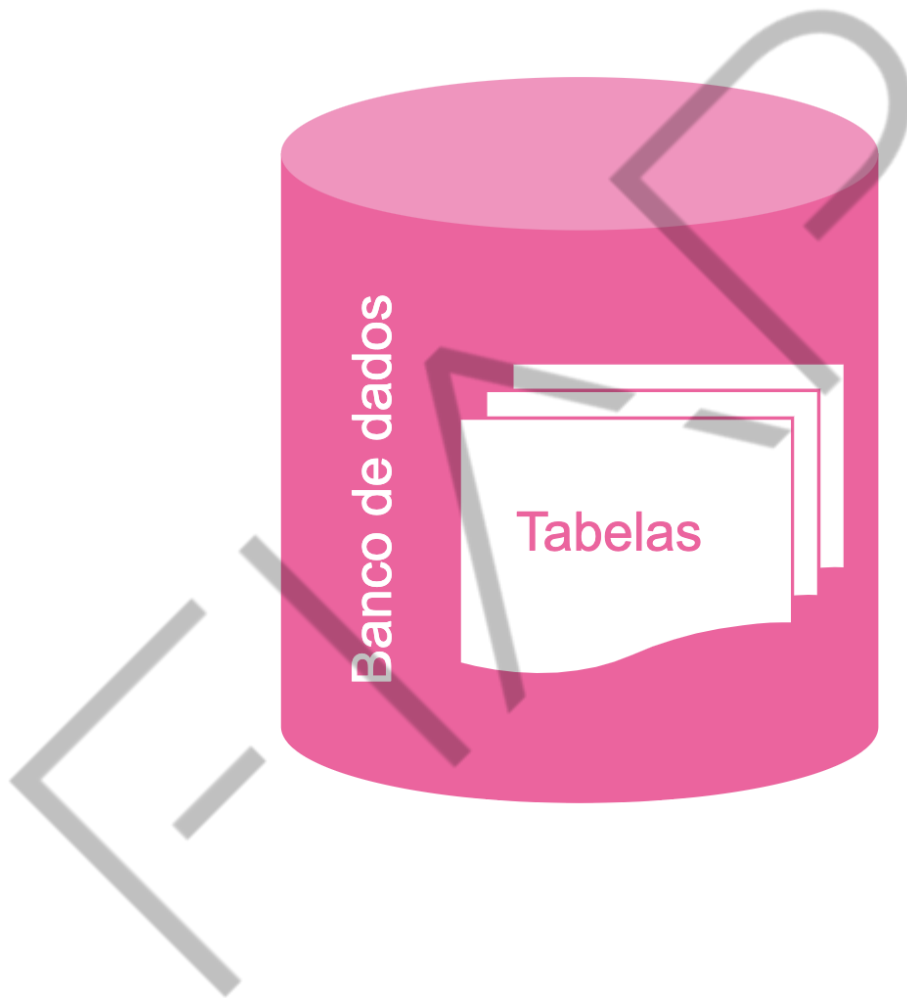


Figura 8 – Visualização do contexto de banco de dados
Fonte: Elaborado pelo autor (2023)

4 O PROJETO

Pronto! Agora podemos começar a desenvolver o projeto.

Este projeto é um simples CRUD – abreviatura de Create (criar), Read (ler), Update (atualizar) e Delete (excluir) registros – em uma tabela de Petshop.

4.1 Tela principal do Projeto

Este projeto terá aparência do menu da imagem abaixo:

EXECUÇÃO:

```
---- CRUD - PETSHOP ----

1 - Cadastrar Pet
2 - Listar Pets
3 - Alterar Pets
4 - Excluir pets
5 - EXCLUIR TODOS OS REGISTROS
6 - SAIR

Escolha -> _
```

Figura 9 – Visualização do Menu da Aplicação
Fonte: Elaborado pelo autor (2023)

Segue a definição dos itens do menu:

- O item **“1 – Cadastrar Pet”**: cadastrará um novo Pet gerando automaticamente um ID (identificação que o distingue dos demais registros).
- O item **“2 – Listar Pets”**: capturará todos os pets gravados na tabela e os apresentará na tela.
- O item **“3 – Alterar Pets”**: fará a edição do registro caso o ID solicitado exista.
- O item **“4 – Excluir Pets”**: apagará o registro da tabela caso o ID solicitado exista.
- O item **“5 – EXCLUIR TODOS OS REGISTROS”**: eliminará todos os registros da tabela incondicionalmente.

- **O item “6 – SAIR”:** termina a execução da aplicação.

Este será o norte que seguiremos para construir cada rotina (item).

4.2 Script do banco e tabela

Para desenvolver a aplicação usaremos uma tabela simples. A fim de que não haja conflito estrutural no Banco de dados, execute o seguinte script dentro do servidor do seu banco de dados:

SCRIPT:

```
create table petshop(  
    id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
    tipo_pet VARCHAR2(30),  
    nome_pet VARCHAR2(30),  
    idade INT  
);
```

Código-fonte 13 – Script da tabela Petshop no Oracle
Fonte: Elaborado pelo autor (2023)

Basicamente, este script cria uma tabela com o nome **petshop**, um campo **ID**, que é chave primária e gerado automaticamente, um campo **tipo_pet**, que armazena o tipo de animal, **nome_pet** que é o nome do animal e, por fim, a **idade** do pet, que é do tipo inteiro.

Chave primária é a escolha do campo na tabela onde não haja redundância (repetições) em seu conteúdo e o seu conteúdo não pode ser vazio.

Detalhando tecnicamente os campos da tabela petshop:

- **‘id’:** será gerado automaticamente e é chave primária;
- **‘tipo_pet’:** será do tipo texto com tamanho máximo de 30. Servirá para armazenar se o pet é um cachorro, gato...
- **‘nome_pet’:** será do tipo texto com o tamanho máximo de 30. Servirá para armazenar o nome do pet.
- **‘idade’:** será do tipo inteiro e armazenará a idade do pet.

5 CODIFICANDO O PROJETO

Configuradas as ferramentas e adicionados os complementos no Python, agora iremos trabalhar com a codificação.

Para um melhor entendimento, cada item do menu será tratado separadamente para um melhor entendimento da aplicação e no final deste capítulo colocarei o código-fonte na íntegra.

5.1 Conectando o servidor e o usuário

Quando tratamos conexão com banco de dados (Oracle) pelo Python, a primeira coisa que devemos nos preocupar é de importar as bibliotecas necessárias na aplicação e efetuar a conexão com o servidor e no servidor o usuário.

PYTHON

```
# Importação dos módulos
import os
import cx_Oracle
import pandas as pd

# Try para tentativa de Conexão com o Banco de Dados
try:
    # Conecta o servidor
    dsnStr = cx_Oracle.makedsn("oracle.fiap.com.br", "?????",
    "ORCL")
    # Efetua a conexão com o Usuário
    conn = cx_Oracle.connect(user='PF1530', password="???????",
    dsn=dsnStr)
    # Cria as instruções para cada módulo
    inst_cadastro = conn.cursor()
    inst_consulta = conn.cursor()
    inst_alteracao = conn.cursor()
    inst_exclusao = conn.cursor()
except Exception as e:
    # Informa o erro
    print("Erro: ", e)
    # Flag para não executar a Aplicação
    conexao = False
else:
    # Flag para executar a Aplicação
    conexao = True
```

Sobre as três primeiras linhas:

A linha 'import os' nos dá a permissão de executar comandos do sistema operacional; como 'cls' - limpar a tela - no Windows ou 'clear' no Linux ou Macos.

A importação de 'import cx_Oracle' habilita os métodos relacionados a manipulação do Oracle dentro da nossa aplicação.

A linha 'import pandas' nos dá melhores recursos para visualização e organização dos dados.

Vale lembrar que muitos dos métodos destas bibliotecas serão executados no decorrer do código.

Nas demais linhas:

Utilizamos um `try except else` para fazer a conexão com o servidor e usuário.

Neste caso é melhor usar um `try` para tratar os erros (ao invés de `if's`) porque podemos ter erros operacionais e circunstanciais. Quando isso ocorre, não podemos prever todos os erros, assim, com o `try`, ele trata todos os erros sem abortar a operação da aplicação.

Na linha `dsnStr = cx_Oracle.makedsn("oracle.fiap.com.br", "1521", "ORCL")` configuramos os dados do Host dentro da nossa aplicação Python. Partimos do princípio de que você tenha um cadastro no servidor (Oracle) com as suas devidas permissões.

A linha `conn = cx_Oracle.connect(user='PF1530', password="??????", dsn=dsnStr)` conecta o usuário com no servidor de banco de dados. Vale ressaltar que o user e a senha '?????' devem ser trocados pelos seus dados de acesso.

As linhas abaixo são os objetos que serão executados no decorrer da aplicação para cada tipo de operação que executaremos na tabela:

PYTHON:

```
inst_cadastro = conn.cursor()  
inst_consulta = conn.cursor()  
inst_alteracao = conn.cursor()  
inst_exclusao = conn.cursor()
```

Código-fonte 15 – Instanciando os cursores para fazer o CRUD
Fonte: Elaborado pelo autor (2023)

O `except` captura o erro e apresenta-o caso ocorra.

Usamos um flag chamado `conexao` para controlar se a conexão obteve êxito (True) ou não (False): a partir deste estado, a aplicação prosseguirá (apresentando o menu) ou será finalizada.

Um **flag**, em programação, é o tipo de variável capaz de controlar as ações do fluxo da aplicação através dos seus estados lógicos True ou False.

5.2 Apresentação do Menu

Considerando que o flag `conexao` esteja com o conteúdo True (foi possível estabelecer a conexão) é apresentado o menu:

PYTHON:

```
# Enquanto o flag conexao estiver apontado com True a aplicação
é executada
while conexao:
    # Limpa a tela via SO
    os.system('cls')

    # Apresenta o menu
    print("---- CRUD - PETSHOP ----")
    print("""
1 - Cadastrar Pet
2 - Listar Pets
3 - Alterar Pet
4 - Excluir Pet
5 - EXCLUIR TODOS OS PETS
6 - SAIR
""")

    # Captura a escolha do usuário
    escolha = int(input(margem + "Escolha -> "))
```

Código-fonte 16 – Apresentação do Menu e leitura da escolha do item
Fonte: Elaborado pelo autor (2023)

PROMPT:

```
---- CRUD - PETSHOP ----

1 - Cadastrar Pet
2 - Listar Pets
3 - Alterar Pets
4 - Excluir pets
5 - EXCLUIR TODOS OS REGISTROS
6 - SAIR

Escolha -> 6
```

Comando de prompt 14 – Apresentação do menu
Fonte: Elaborado pelo autor (2023)

Na variável `escolha` é aguardada a digitação da escolha do usuário.

5.3 Cadastrar PET

Nesta operação, o usuário digitará o tipo de pet, nome e idade para efetuar o cadastro na tabela. A identificação do Pet (número inteiro único) é gerada automaticamente de forma sequencial a partir dos que já estão cadastrados na tabela.

PYTHON:

```
# VERIFICA QUAL A ESCOLHA DO USUÁRIO
match escolha:

    # CADASTRAR UM PET
    case 1:
        try:
            print("----- CADASTRAR PET -----\\n")
            # Recebe os valores para cadastro
            tipo = input(margem + "Digite o tipo.....: ")
            nome = input(margem + "Digite o nome.....: ")
            idade = int(input(margem + "Digite a idade....: "))

            # Monta a instrução SQL de cadastro em uma string
            cadastro = f""" INSERT INTO petshop (tipo_pe,
nome_pet, idade)VALUES ('{tipo}', '{nome}', {idade}) """

            # Executa e grava o Registro na Tabela
            inst_cadastro.execute(cadastro)
            conn.commit()
        except ValueError:
            print("Digite um número na idade!")
        except:
            print("Erro na transação do BD")
        else:
            print("\\nDados GRAVADOS")
            input("Presione ENTER")
```

Código-fonte 17 – Rotina de cadastro
Fonte: Elaborado pelo autor (2023)

Se a escolha digitada foi a 1, entrará neste `case 1` e, primeiramente, pedirá as informações (tipo, nome e idade) do pet. Lembrando que somente a idade é do tipo `int`.

Python e a sua conexão com o Banco de dados

Em seguida criamos uma instrução SQL em uma variável string chamada `cadastro` e concatenando nela as informações digitadas pelo usuário.

O identificador `inst_cadastro` é o cursor – identificador que permite executar a instrução SQL nele contido via Python – que criamos na abertura das conexões. Acompanhado do método `execute`, ele executa a instrução SQL passada por parâmetro. O `commit` consoma a gravação magnética dos dados no Banco de dados.

Os dois `excepts` criados são para tratar os eventuais erros de **não digitar um número** na idade e caso ocorra alguma **falha no Banco de dados**.

Este é o status da tabela `petshop` no Banco de dados (visão Oracle) depois de darmos um `select`:

Considere o termo “darmos um select” como a instrução SQL “`SELECT * FROM PETSHOP`” que significa ‘*Selecionar todos os registros da tabela Petshop*’:

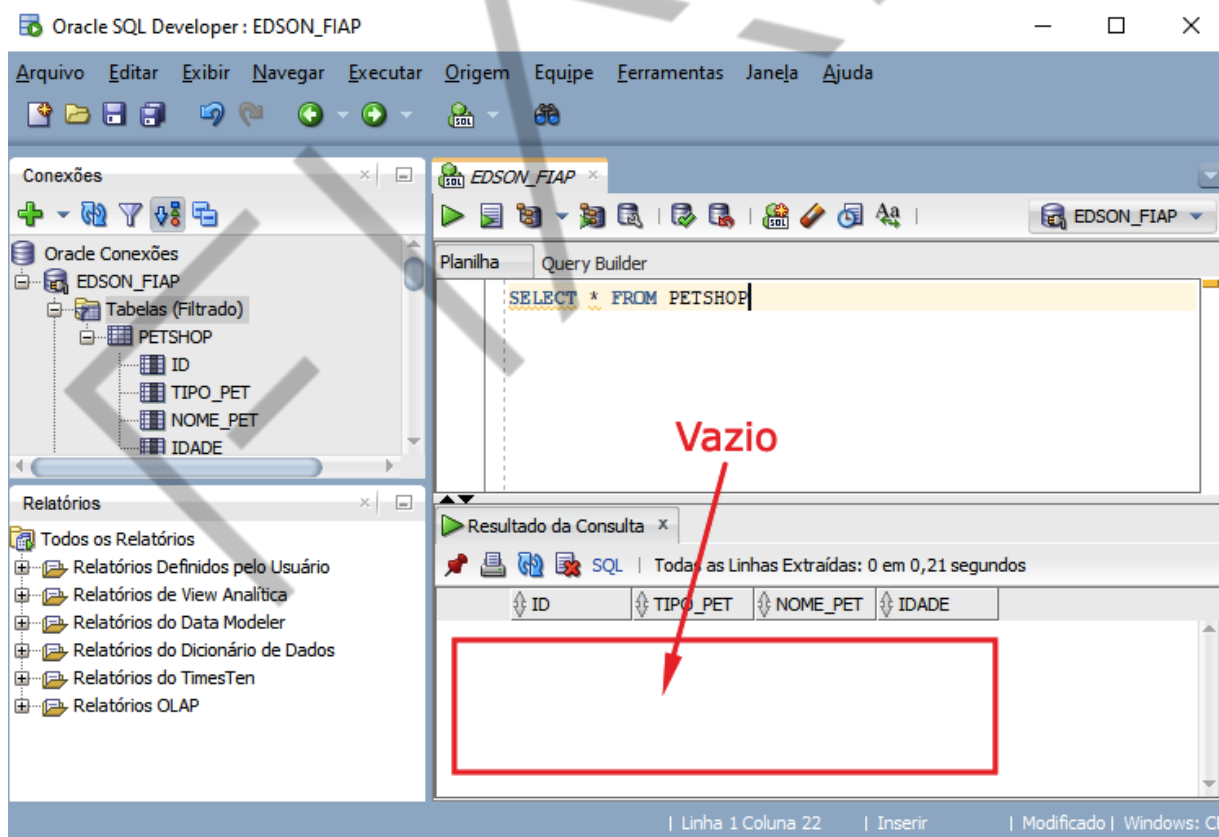


Figura 10 – Status da tabela vazia
Fonte: Elaborado pelo autor (2023)

A tabela está vazia. Vamos executar a rotina Python escrita logo acima:

EXECUÇÃO:

```
0---- CRUD - PETSHOP ----

    1 - Cadastrar Pet
    2 - Listar Pets
    3 - Alterar Pet
    4 - Excluir Pet
    5 - EXCLUIR TODOS OS PETS
    6 - SAIR

Escolha -> 1
0----- CADASTRAR PET -----

    Digite o tipo....: Cachorro
    Digite o nome....: Bob
    Digite a idade...: 12

Dados GRAVADOS
Presione ENTER
```

Figura 11 – Execução da rotina de cadastro
Fonte: Elaborado pelo autor (2023)

Depois de exibido o menu, foi escolhida a opção '1 – Cadastrar Pet' e apareceram as informações para serem digitadas e, ao final, foi cadastrado o registro do PET. Veja como ficou o registro no Banco de Dados:

Python e a sua conexão com o Banco de dados

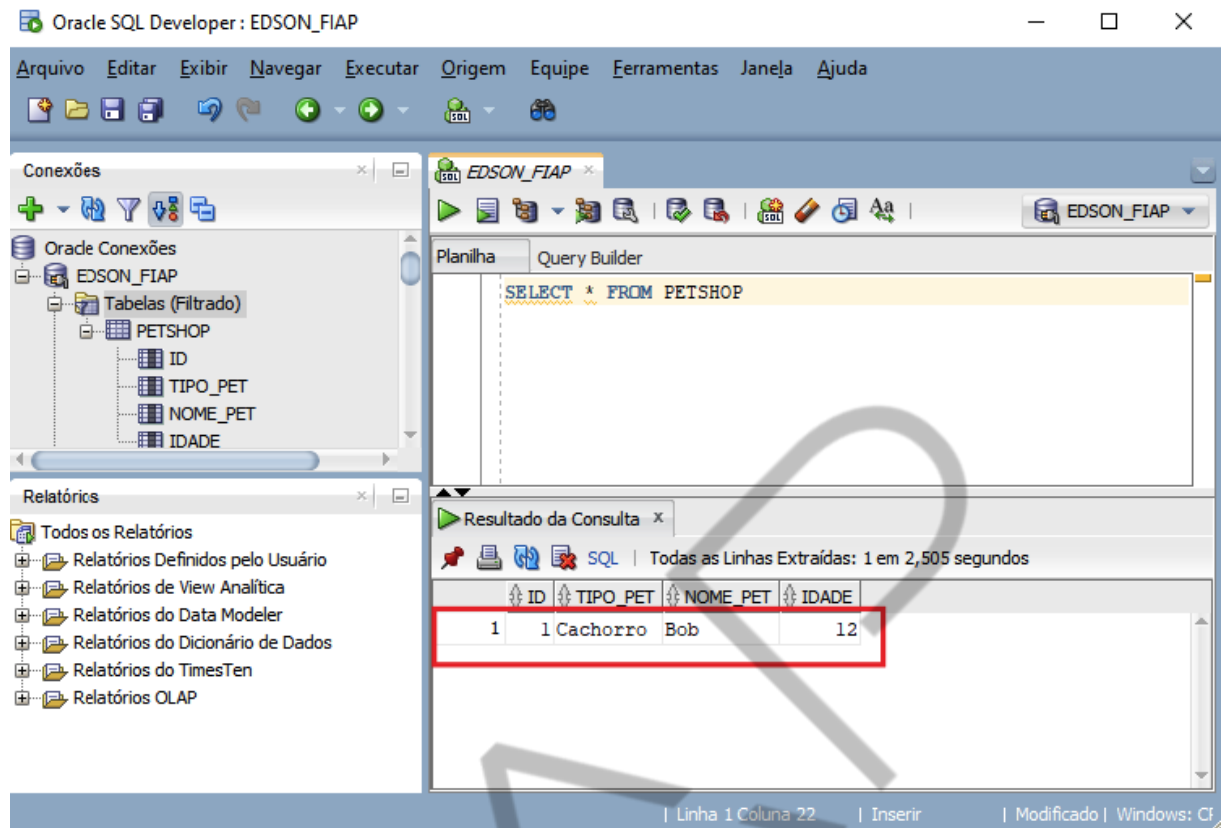


Figura 12 – Status da tabela depois do cadastro
Fonte: Elaborado pelo autor (2023)

O Registro do cachorro Bob foi devidamente cadastrado no banco de dados e o ID (Identificação única do registro) foi gerado automaticamente.

5.4 Listar PETs

Com o passar do tempo, a tabela é alimentada com inúmeros registros através da rotina '1 – Cadastrar Pet'. A rotina '2 - Listar Pets' seria o complemento desta e tem como principal objetivo apresentar o conteúdo dos registros da tabela na aplicação Python.

Segue a rotina 2:

PYTHON:


```
case 2:
    print("----- LISTAR PETS -----\\n")
    lista_dados = [] # Lista para captura de dados do Banco

    # Monta a instrução SQL de seleção de todos os registros da
    # tabela
    inst_consulta.execute('SELECT * FROM petshop')
    # Captura todos os registros da tabela e armazena no objeto
    data
    data = inst_consulta.fetchall()

    # Insere os valores da tabela na Lista
    for dt in data:
        lista_dados.append(dt)

        # ordena a lista
        lista_dados = sorted(lista_dados)

    # Gera um DataFrame com os dados da lista utilizando o
    # Pandas
    dados_df = pd.DataFrame.from_records(lista_dados,
        columns=['Id', 'Tipo', 'Nome', 'Idade'], index='Id')

    # Verifica se não há registro através do dataframe
    if dados_df.empty:
        print(f"Não há um Pets cadastrados!")
    else:
        print(dados_df) # Exibe os dados selecionados da tabela
        print("\\nLISTADOS!")
        input("Pressione ENTER")
```

Código-fonte 18 – Rotina de listagem dos registros
Fonte: Elaborado pelo autor (2023)

Ao selecionar a escolha 2, listar pets, inicialmente, será criada uma lista de nome `lista_dados[]`, que terá como objetivo o de armazenar os dados retornados da consulta na tabela.

A linha `inst_consulta.execute('SELECT * FROM petshop')` seleciona incondicionalmente todos os registros da tabela, armazenando os dados no cursor `inst_consulta`.

Python e a sua conexão com o Banco de dados

O método `fetchall()` contido na linha de comando `data = inst_consulta.fetchall()` retorna as linhas do resultado da consulta em uma lista para que os dados possam ser manipulados na aplicação Python.

No laço `for`, os dados capturados e armazenados no objeto `data` foram inseridos na `lista_dados()`.

O `dataframe`, método da biblioteca `pandas`, na linha de comando `dados_df = pd.DataFrame.from_records(lista_dados, columns=['Id', 'Tipo', 'Nome', 'Idade'], index='Id')` gera uma tabela (matriz) para exibir mais facilmente os dados na aplicação Python. Os argumentos contidos no método `from_records` nada mais são do que os títulos que colocamos em cada coluna, ordenado pelo campo `ID`.

Por fim, o `if` trata a situação do `dataframe` (ou tabela) estar vazio ou não. Veja a execução da rotina:

EXECUÇÃO:

```
0---- CRUD - PESHOP ----

1 - Cadastrar Pet
2 - Listar Pets
3 - Alterar Pet
4 - Excluir Pet
5 - EXCLUIR TODOS OS PETS
6 - SAIR

Escolha -> 2
0----- LISTAR PETS -----

      Tipo      Nome  Idade
Id
1  Cachorro      Bob    12
2      Gato  Princesa    3
3  Papagaio      Zé     4
4  Cachorro      Thor    8

LISTADOS!
Pressione ENTER
```

Figura 13 – Execução da rotina de listagem dos registros
Fonte: Elaborado pelo autor (2023)

5.5 Alterar PET

O princípio básico para editarmos algo em um registro é que ele exista. Então, para fazermos esta rotina, inicialmente, verificamos a existência do registro. Assim, faremos uma consulta singular.

Consulta singular é aquela que pode retornar UM ou NENHUM registro. Veja a Rotina:

PYTHON

```
# ALTERAR OS DADOS DE UM REGISTRO
case 3:
    try:
        # ALTERANDO UM REGISTRO
        print("----- ALTERAR DADOS DO PET -----\\n")

        lista_dados = [] # Lista para captura de dados da
tabela

        pet_id = int(input(margem + "Escolha um Id: ")) #
Permite o usuário escolher um Pet pelo id

        # Constroi a instrução de consulta para verificar a
existencia ou não do id
        consulta = f""" SELECT * FROM petshop WHERE id =
{pet_id}"""
        inst_consulta.execute(consulta)
        data = inst_consulta.fetchall()

        # Preenche a lista com o registro encontrado (ou não)
        for dt in data:
            lista_dados.append(dt)

        # analisa se foi encontrado algo
        if len(lista_dados) == 0: # se não há o id
            print(f"Não há um pet cadastrado com o ID =
{pet_id}")
            input("\\nPressione ENTER")
        else:
            # Captura os novos dados
            novo_tipo = input(margem + "Digite um novo tipo:
")

            novo_nome = input(margem + "Digite um novo nome:
")

            nova_idade = input(margem + "Digite uma nova
idade: ")
```

Python e a sua conexão com o Banco de dados

```
# Constroi a instrução de edição do registro com os novos dados
alteracao = f"""
UPDATE petshop SET tipo_pet='{novo_tipo}',
nome_pet='{novo_nome}', idade='{nova_idade}' WHERE id={pet_id}
"""
inst_alteracao.execute(alteracao)
conn.commit()
except ValueError:
    print("Digite um número na idade!")
except:
    print("Erro na transação do BD")
else:
    print("\nDados ATUALIZADOS!")
input("Presione ENTER")
```

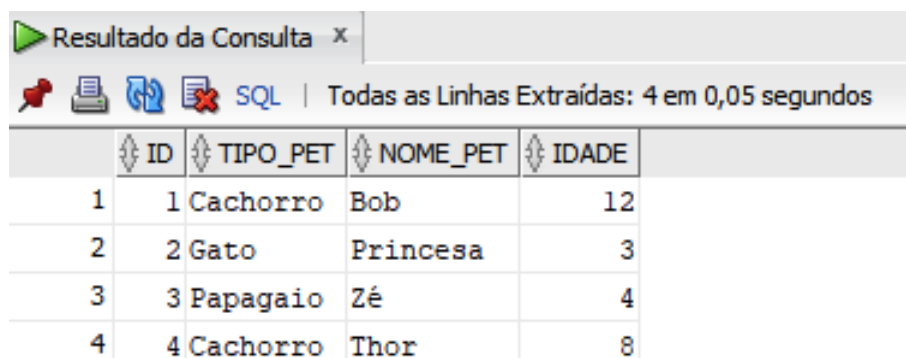
Código-fonte 19 – Rotina de Alteração de um registro
Fonte: Elaborado pelo autor (2023)

Dentro de um `try` criamos uma lista vazia e pedimos para o usuário digitar o ID do pet que ele deseja editar. A pesquisa criada na variável `consulta` será `'consulta = f"SELECT * FROM petshop WHERE id = {pet_id}"` desta forma, depois de executada, a instrução SQL retornará 0 ou 1 registro porque neste caso há a cláusula `Where`.

A cláusula `Where` no SQL filtra a pesquisa. No caso acima `'... WHERE id = {pet_id}'` quer dizer 'Se o valor digitado pelo usuário estiver contido em algum registro da tabela.

Executada a consulta, é verificado se o registro existe ou não. Se existe, são solicitados novos dados para o usuário, gerada a instrução `UPDATE` com estes dados para atualizar o registro.

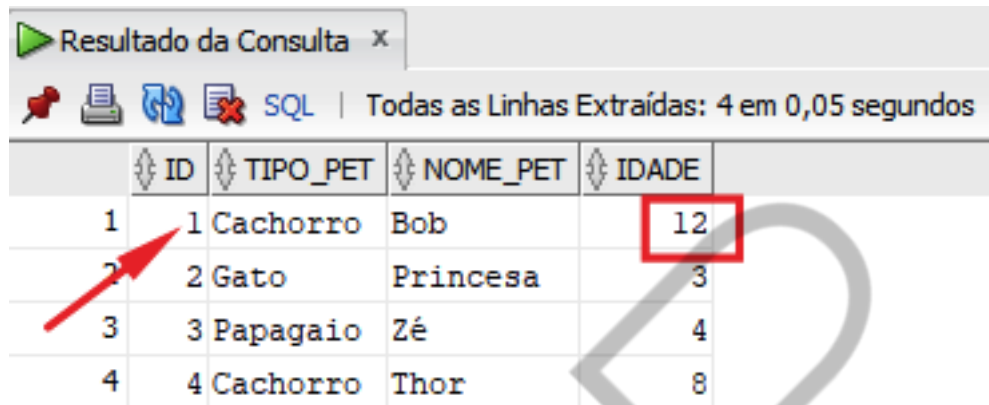
O status dos registros no Oracle é:



	ID	TIPO_PET	NOME_PET	IDADE
1	1	Cachorro	Bob	12
2	2	Gato	Princesa	3
3	3	Papagaio	Zé	4
4	4	Cachorro	Thor	8

Figura 14 – Status dos dados gravados na Tabela
Fonte: Elaborado pelo autor (2023)

Vamos editar a idade do Cachorro Bob, cujo ID é 1:



	ID	TIPO_PET	NOME_PET	IDADE
1	1	Cachorro	Bob	12
2	2	Gato	Princesa	3
3	3	Papagaio	Zé	4
4	4	Cachorro	Thor	8

Figura 15 – Informação a ser editada
Fonte: Elaborado pelo autor (2023)

Vamos sugerir que ele fez aniversário e a idade dele agora é 13.

EXECUÇÃO:

```
0---- CRUD - PETSHOP ----

1 - Cadastrar Pet
2 - Listar Pets
3 - Alterar Pet
4 - Excluir Pet
5 - EXCLUIR TODOS OS PETS
6 - SAIR

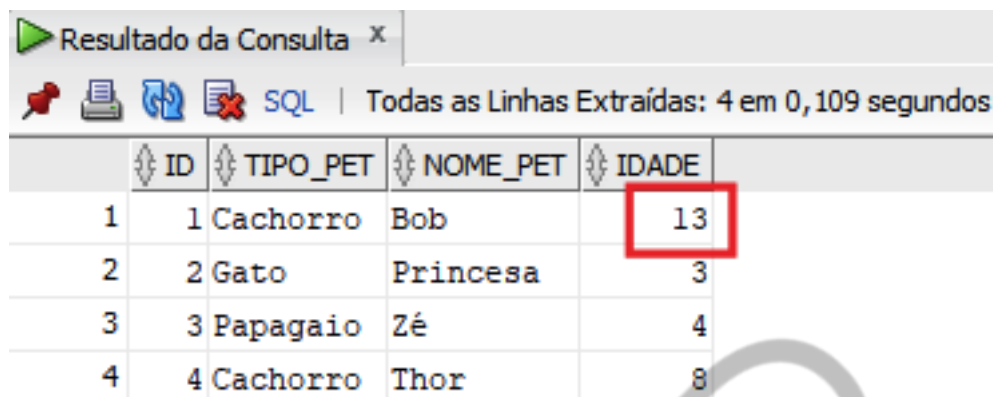
Escolha -> 3
0----- ALTERAR DADOS DO PET -----

Escolha um Id: 1
Digite um novo tipo: Cachorro
Digite um novo nome: Bob
Digite uma nova idade: 13

Dados ATUALIZADOS!
Presione ENTER
```

Figura 16 – Execução da rotina de Alteração
Fonte: Elaborado pelo autor (2023)

Veja que a idade foi atualizada na Tabela:



	ID	TIPO_PET	NOME_PET	IDADE
1	1	Cachorro	Bob	13
2	2	Gato	Princesa	3
3	3	Papagaio	Zé	4
4	4	Cachorro	Thor	8

Figura 17 – Informação editada
Fonte: Elaborado pelo autor (2023)

Para analisar o “poder” da nossa aplicação, vamos sugerir editar um registro que tenha o ID inexistente, vamos sugerir o id 23:

EXECUÇÃO:

```
0---- CRUD - PETSHOP ----  
  
1 - Cadastrar Pet  
2 - Listar Pets  
3 - Alterar Pet  
4 - Excluir Pet  
5 - EXCLUIR TODOS OS PETS  
6 - SAIR  
  
Escolha -> 3  
0----- ALTERAR DADOS DO PET -----  
  
Escolha um Id: 23  
Não há um pet cadastrado com o ID = 23  
  
Pressione ENTER
```

Figura 18 – Informação não encontrada
Fonte: Elaborado pelo autor (2023)

A aplicação informou que o ID 23 não existe, logo (e obviamente), nem permitiu que fossem digitados os novos dados.

5.6 Excluir PET

Como ocorreu com a edição do registro, para excluir um registro ele deve existir. Primeiramente, é efetuada a consulta singular e em caso de sucesso é efetuada a exclusão do Registro.

Segue a rotina caso a escolha seja a 4:

PYTHON:

```
# EXCLUIR UM REGISTRO
case 4:
    print("----- EXCLUIR PET -----\\n")
    lista_dados = [] # Lista para captura de dados da tabela
    pet_id = input(margem + "Escolha um Id: ") # Permite o
usuário escolher um Pet pelo ID
    if pet_id.isdigit():
        pet_id = int(pet_id)
        consulta = f"\" SELECT * FROM petshop WHERE id =
{pet_id}\""
        inst_consulta.execute(consulta)
        data = inst_consulta.fetchall()

        # Insere os valores da tabela na lista
        for dt in data:
            lista_dados.append(dt)

        # Verifica se o registro está cadastrado
        if len(lista_dados) == 0:
            print(f"Não há um pet cadastrado com o ID = {pet_id}")
        else:
            # Cria a instrução SQL de exclusão pelo ID
            exclusao = f"DELETE FROM petshop WHERE id={pet_id}"
            # Executa a instrução e atualiza a tabela
            inst_exclusao.execute(exclusao)
            conn.commit()
            print("\\nPet APAGADO!") # Exibe mensagem caso haja
sucesso
        else:
            print("O Id não é numérico!")
            input("Pressione ENTER") # Pausa o loop para a leitura da
mensagem
```

Código-fonte 20 – Rotina de exclusão de um registro
Fonte: Elaborado pelo autor (2023)

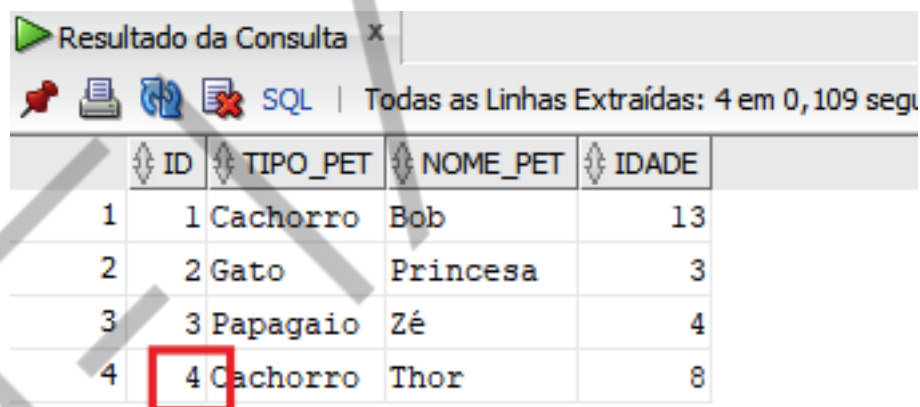
Solicitamos ao usuário o ID a ser excluído. Diferentemente das outras rotinas, ao invés de usar o `try`, preferi tratar o problema de o ID ser ou não número com o `if pet_id.isdigit()`: esta forma diferente foi apenas para agregar e mostrar que podemos fazer a mesma rotina de formas diferentes.

O método `.isdigit()` verifica se o conteúdo da variável string é ou não um número.

Considerando que o id digitado é um número, efetuamos um Select para ver se o registro existe ou não (da mesma forma que fizemos com a alteração). Caso ele exista, montamos com DELETE uma instrução sql para excluir o registro filtrado pelo id: `exclusao = f"DELETE FROM petshop WHERE id={pet_id}"`.

Tome cuidado com comandos SQL que modifiquem o conteúdo da tabela, se não for colocada a cláusula WHERE, todos os registros serão modificados | excluídos.

Sabemos que na tabela o id 4 é relacionado ao cachorro Thor:



ID	TIPO_PET	NOME_PET	IDADE
1	Cachorro	Bob	13
2	Gato	Princesa	3
3	Papagaio	Zé	4
4	Cachorro	Thor	8

Figura 19 – Visualização da linha que será excluída
Fonte: Elaborado pelo autor (2023)

Vamos excluir este registro? Voltemos à aplicação:

EXECUÇÃO:


```
0---- CRUD - PESHOP ----

1 - Cadastrar Pet
2 - Listar Pets
3 - Alterar Pet
4 - Excluir Pet
5 - EXCLUIR TODOS OS PETS
6 - SAIR

Escolha -> 4
0----- EXCLUIR PET -----

Escolha um Id: 4

Pet APAGADO!
Pressione ENTER
```

Figura 20 – Execução da rotina de exclusão
Fonte: Elaborado pelo autor (2023)

Para conferir se a exclusão obteve sucesso, vamos dar um Select no Oracle para ver se de fato o registro foi excluído:

Resultado da Consulta x

SQL | Todas as Linhas Extraídas: 3 em 0,023 segundos

	ID	TIPO_PET	NOME_PET	IDADE
1	1	Cachorro	Bob	13
2	2	Gato	Princesa	3
3	3	Papagaio	Zé	4

Figura 21 – Visualização da exclusão da linha 4
Fonte: Elaborado pelo autor (2023)

Sim, o ID 4 foi excluído. Assim como na edição de registros, caso o id não exista na tabela o fluxo da aplicação interrompe o processo de exclusão.

5.7 EXCLUIR TODOS OS PETS

Apesar de atualmente não ser usual a exclusão de registros, ela pode ser feita. Estamos nos atendo ao aprendizado do CRUD, então “Deletar” registros faz parte das operações fundamentais em tabelas. Primeiramente, vejamos o status inicial da Tabela Petshop:

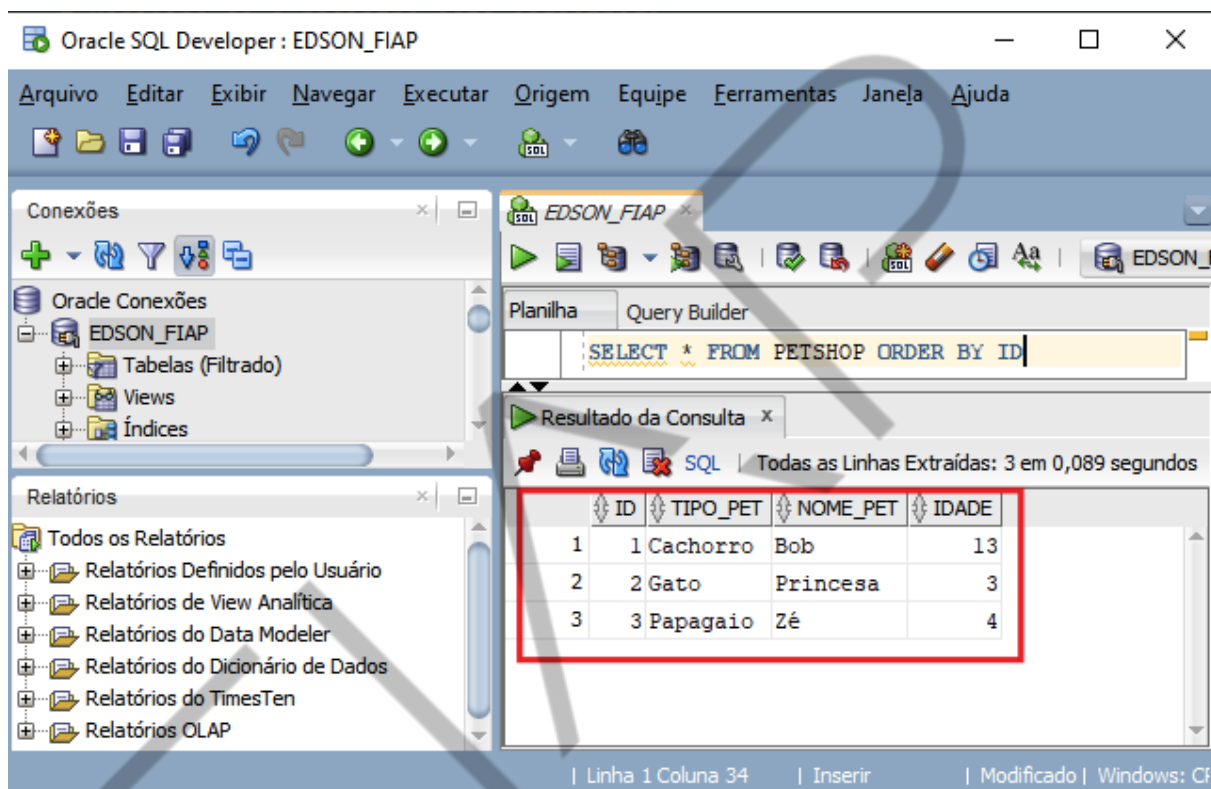


Figura 22 – Visualização de todos os registros na tabela
Fonte: Elaborado pelo autor (2022)

Na tabela temos três registros adicionados.

Vamos considerar, então, que a escolha do usuário foi o item '5 – EXCLUIR TODOS OS PETS'.

Depois da escolha da opção 5, pela operação ser delicada, por segurança, o ideal é perguntarmos para o usuário a confirmação da exclusão de todos os registros ou permitir que ele desista.

EXECUÇÃO:

```
0---- CRUD - PETSHOP ----

1 - Cadastrar Pet
2 - Listar Pets
3 - Alterar Pet
4 - Excluir Pet
5 - EXCLUIR TODOS OS PETS
6 - SAIR

Escolha -> 5

!!!!!! EXCLUI TODOS OS DADOS TABELA !!!!!

CONFIRMA A EXCLUSÃO DE TODOS OS PETS?
[S]im ou [N]ÃO?
```

Figura 23 – Visualização da confirmação da exclusão de todos os registros
Fonte: Elaborado pelo autor (2023)

Com a escolha do item 5 será executada a rotina:

PYTHON:

```
# EXCLUIR TODOS OS REGISTROS
case 5:
    print("\n!!!!!! EXCLUI TODOS OS DADOS TABELA !!!!!\n")
    confirma = input(margem + "CONFIRMA A EXCLUSÃO DE TODOS OS
PETS?\n[S]im ou [N]ÃO?")
    if confirma.upper() == "S":
        # Apaga todos os registros
        exclusao = "DELETE FROM petshop"
        inst_exclusao.execute(exclusao)
        conn.commit()

        # Depois de excluir todos os registros ele zera o ID
        data_reset_ids = """ ALTER TABLE petshop MODIFY(ID
GENERATED AS IDENTITY (START WITH 1)) """
        inst_exclusao.execute(data_reset_ids)
        conn.commit()

        print("\nTodos os registros foram excluídos!")
    else:
        print("Operação cancelada pelo usuário!")
        input("Pressione ENTER") # Pausa o loop para a leitura da
mensagem
```

Código-fonte 21 – Rotina de exclusão de todos os registros

Fonte: Elaborado pelo autor (2023)

Depois de lida a resposta do usuário ('S' ou 'N'), sobre excluir os registros, o `if confirma.upper()` analisa se foi digitado 'S'[im], se positivo, então é construída a instrução SQL `DELETE * FROM petshop` e jogada na variável `exclusao`. Na sequência, é executada a instrução de exclusão de todos os registros na tabela e a mensagem 'Todos os registros foram excluídos!' aparecerá para o usuário.

Agora, se o usuário digitar 'N'[ão], a operação é cancelada e aparece a mensagem: 'Operação cancelada pelo usuário!'.

Considerando que a tabela tinha 3 registros, logo, foi gerado automaticamente até o ID 3. Depois de excluídos os registros, os IDs não são 'zerados'. Assim, se alguém incluir um novo registro, ele iniciará do 4 e não do 1, como seria o esperado. Para resolver este problema, adicionamos a instrução `ALTER TABLE petshop MODIFY (ID GENERATED AS IDENTITY (START WITH 1))` que simplesmente faz com que o ID seja gerado automaticamente a partir do ID 1. Depois de executada esta rotina veja que a tabela ficou vazia:

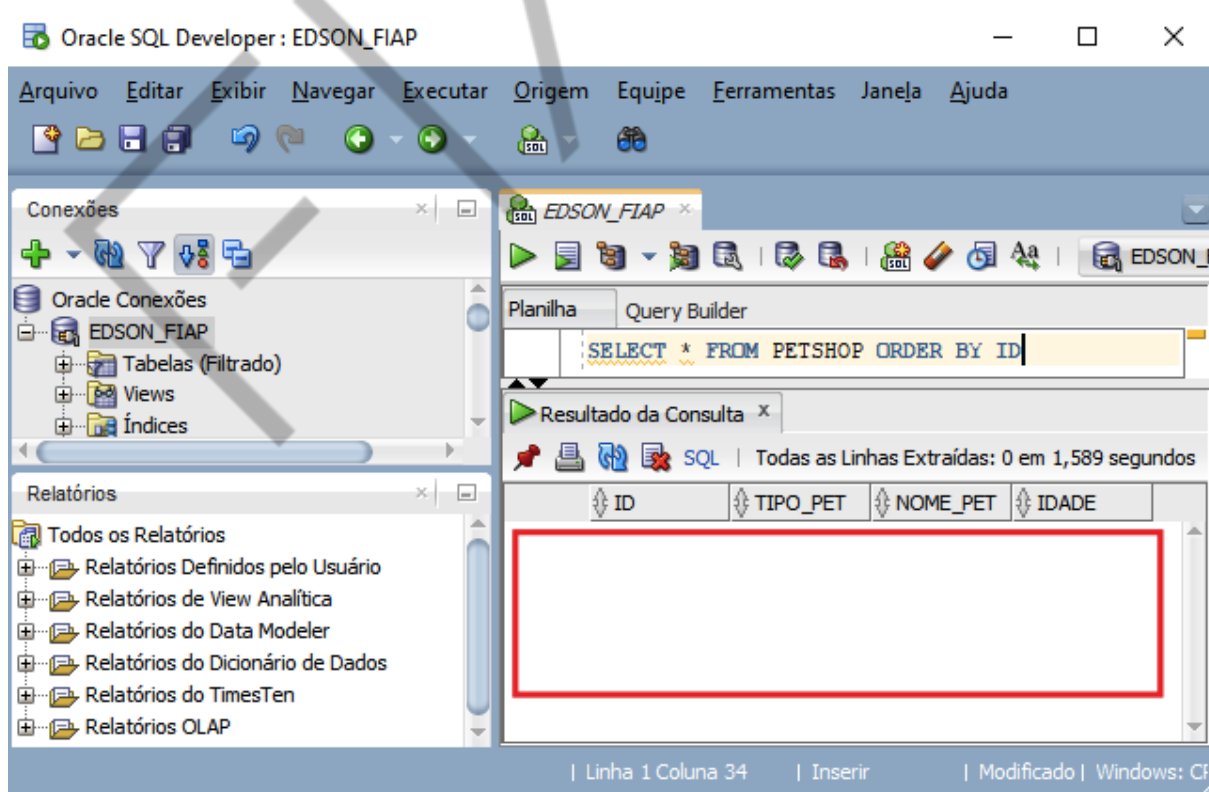


Figura 24 – Visualização da tabela vazia
Fonte: Elaborado pelo autor (2023)

No modelo atual, dificilmente as aplicações excluem registros. Elas colocam campos 'flags' como 'Ativo'. Se este campo estiver com o conteúdo True, quer dizer que o registro está disponível na aplicação, agora se o conteúdo for False, a aplicação não o "enxerga". De qualquer forma, o registro permanece na Base para ser trabalhado posteriormente se houver necessidade.

5.8 Código-fonte da Aplicação

Para explicar a aplicação, foram usadas as rotinas separadas. Neste tópico colocarei o código-fonte completo.

Lembre-se apenas que você deve ter o seu acesso ao Oracle e criar uma tabela idêntica (siga o script passado) na sua base para que o código funcione plenamente.

Por motivos óbvios apaguei meu usuário e senha da conexão.

Lembre-se também que o objetivo deste capítulo é tratar os novos conhecimentos sobre conexão com Oracle, portanto, não fui cuidadoso em relação a consistências diversas, como vazio, não números porque já vimos este assunto em outros momentos.

Segue o código:

PYTHON:

```
# Importação dos módulos
import os
import cx_Oracle
import pandas as pd

# Try para tentativa de Conexão com o Banco de Dados
try:
    # Conecta o servidor
    dsnStr = cx_Oracle.makedsn("oracle.fiap.com.br", "1521",
    "ORCL")
    # Efetua a conexao com o Usuário
    conn = cx_Oracle.connect(user='PF????', password="???????",
    dsn=dsnStr)
    # Cria as instruções para cada módulo
    inst_cadastro = conn.cursor()
    inst_consulta = conn.cursor()
    inst_alteracao = conn.cursor()
    inst_exclusao = conn.cursor()
except Exception as e:
    # Informa o erro
    print("Erro: ", e)
```

Python e a sua conexão com o Banco de dados

```
# Flag para não executar a Aplicação
conexao = False
else:
    # Flag para executar a Aplicação
    conexao = True

margem = ' ' * 4 # Define uma margem para a exibição da
aplicação

# Enquanto o flag conexao estiver apontado com True
while conexao:
    # Limpa a tela via SO
    os.system('cls')

    # Apresenta o menu
    print("----- CRUD - PETSHOP -----")
    print("""
1 - Cadastrar Pet
2 - Listar Pets
3 - Alterar Pet
4 - Excluir Pet
5 - EXCLUIR TODOS OS PETS
6 - SAIR
""")

    # Captura a escolha do usuário
    escolha = input(margem + "Escolha -> ")

    # Verifica se o número digitado é um valor numérico
    if escolha.isdigit():
        escolha = int(escolha)
    else:
        escolha = 6
        print("Digite um número.\nReinicie a Aplicação!")

    os.system('cls') # Limpa a tela via SO

    # VERIFICA QUAL A ESCOLHA DO USUÁRIO
    match escolha:

        # CADASTRAR UM PET
        case 1:
            try:
                print("----- CADASTRAR PET -----\\n")
                # Recebe os valores para cadastro
                tipo = input(margem + "Digite o tipo.....: ")
                nome = input(margem + "Digite o nome.....: ")
                idade = int(input(margem + "Digite a idade....: "
            ))

            # Monta a instrução SQL de cadastro em uma
```

Python e a sua conexão com o Banco de dados

```
string
    cadastro = f""" INSERT INTO petshop (tipo_pet,
nome_pet, idade)VALUES ('{tipo}', '{nome}', {idade}) """

    # Executa e grava o Registro na Tabela
    inst_cadastro.execute(cadastro)
    conn.commit()
except ValueError:
    # Erro de número não digitar um numero na
idade
    print("Digite um número na idade!")
except:
    # Caso ocorra algum erro de conexão ou no BD
    print("Erro na transação do BD")
else:
    # Caso haja sucesso na gravação
    print("\n##### Dados GRAVADOS #####")

# LISTAR TODOS OS PETS
case 2:
    print("----- LISTAR PETs -----\\n")
    lista_dados = [] # Lista para captura de dados do
Banco

    # Monta a instrução SQL de seleção de todos os
registros da tabela
    inst_consulta.execute('SELECT * FROM petshop')
    # Captura todos os registros da tabela e armazena
no objeto data
    data = inst_consulta.fetchall()

    # Insere os valores da tabela na Lista
    for dt in data:
        lista_dados.append(dt)

    # ordena a lista
    lista_dados = sorted(lista_dados)

    # Gera um DataFrame com os dados da lista
utilizando o Pandas
    dados_df = pd.DataFrame.from_records(lista_dados,
columns=['Id', 'Tipo', 'Nome', 'Idade'], index='Id')

    # Verifica se não há registro através do dataframe
    if dados_df.empty:
        print(f"Não há um Pets cadastrados!")
    else:
        print(dados_df) # Exibe os dados selecionados
da tabela

    print("\n##### LISTADOS! #####")
```

Python e a sua conexão com o Banco de dados

```
# ALTERAR OS DADOS DE UM REGISTRO
case 3:
    try:
        # ALTERANDO UM REGISTRO
        print("----- ALTERAR DADOS DO PET -----\\n")

        lista_dados = [] # Lista para captura de
dados da tabela

        pet_id = int(input(margem + "Escolha um Id:
")) # Permite o usuário escolher um Pet pelo id

        # Constroi a instrução de consulta para
verificar a existencia ou não do id
        consulta = f""" SELECT * FROM petshop WHERE id
= {pet_id}"""
        inst_consulta.execute(consulta)
        data = inst_consulta.fetchall()

        # Preenche a lista com o registro encontrado
(ou não)
        for dt in data:
            lista_dados.append(dt)

        # analisa se foi encontrado algo
        if len(lista_dados) == 0: # se não há o id
            print(f"Não há um pet cadastrado com o ID
= {pet_id}")
            input("\\nPressione ENTER")
        else:
            # Captura os novos dados
            novo_tipo = input(margem + "Digite um novo
tipo: ")
            novo_nome = input(margem + "Digite um novo
nome: ")
            nova_idade = input(margem + "Digite uma
nova idade: ")

            # Constroi a instrução de edição do
registro com os novos dados
            alteracao = f"""
UPDATE petshop SET tipo_pet='{novo_tipo}',
nome_pet='{novo_nome}', idade='{nova_idade}' WHERE id={pet_id}
"""
            inst_alteracao.execute(alteracao)
            conn.commit()
        except ValueError:
            print("Digite um número na idade!")
        except:
            print(margem + "Erro na transação do BD")
```


Python e a sua conexão com o Banco de dados

```
        else:
            print("\n##### Dados ATUALIZADOS! #####")

# EXCLUIR UM REGISTRO
case 4:
    print("----- EXCLUIR PET -----")
    lista_dados = [] # Lista para captura de dados da
tabela
    pet_id = input(margem + "Escolha um Id: ") #
Permite o usuário escolher um Pet pelo ID
    if pet_id.isdigit():
        pet_id = int(pet_id)
        consulta = f"SELECT * FROM petshop WHERE id
= {pet_id}"

        inst_consulta.execute(consulta)
        data = inst_consulta.fetchall()

        # Insere os valores da tabela na lista
        for dt in data:
            lista_dados.append(dt)

        # Verifica se o registro está cadastrado
        if len(lista_dados) == 0:
            print(f"Não há um pet cadastrado com o ID
= {pet_id}")
        else:
            # Cria a instrução SQL de exclusão pelo ID
            exclusao = f"DELETE FROM petshop WHERE
id={pet_id}"

            # Executa a instrução e atualiza a tabela
            inst_exclusao.execute(exclusao)
            conn.commit()
            print("\n##### Pet APAGADO! #####") #
Exibe mensagem caso haja sucesso
        else:
            print("O Id não é numérico!")

# EXCLUIR TODOS OS REGISTROS
case 5:
    print("\n!!!!!! EXCLUI TODOS OS DADOS TABELA
!!!!!!")
    confirma = input(margem + "CONFIRMA A EXCLUSÃO DE
TODOS OS PETS? [S]im ou [N]ão?")
    if confirma.upper() == "S":
        # Apaga todos os registros
        exclusao = "DELETE FROM petshop"
        inst_exclusao.execute(exclusao)
        conn.commit()

        # Depois de excluir todos os registros ele
zera o ID
```

Python e a sua conexão com o Banco de dados

```
        data_reset_ids = """ ALTER TABLE petshop
MODIFY(ID GENERATED AS IDENTITY (START WITH 1)) """
        inst_exclusao.execute(data_reset_ids)
        conn.commit()

        print("##### Todos os registros foram
excluídos! #####")
    else:
        print(margem + "Operação cancelada pelo
usuário!")

    # SAI DA APLICAÇÃO
    case 6:
        # Modificando o flag da conexão
        conexao = False

    # CASO O NUMERO DIGITADO NÃO SEJA UM DO MENU
    case _:
        input(margem + "Digite um número entre 1 e 6.")

    # Pausa o fluxo da aplicação para a leitura das
informações
    input(margem + "Pressione ENTER")
else:
    print("Obrigado por utilizar a nossa aplicação! :)")
```

Código-fonte 22 – Código fonte completo da aplicação
Fonte: Elaborado pelo autor (2023)

Queridos alunos, com este capítulo encerramos o primeiro ano letivo. Espero que tenham gostado

GLOSSÁRIO

Script	Sequência de comandos.
Banco de dados	Local em que os dados são organizados.
Registro	Linha da tabela.
Campo	Coluna da Tabela.
Tabela	Junção dos campos com os registros.
CRUD	As quatro operações fundamentais em banco de dados: Cadastro, consulta, alteração e exclusão.
Tratamento de erros	Prever todas as possibilidades onde pode ocorrer um erro de operação do usuário.