Danilo Campana Fuchs - 1906755

Prof. Leandro Batista de Almeida

Introdução a Banco de Dados

26 de junho de 2023

Trabalho Final - Gerenciador de Banco de Dados em Python

Este trabalho consiste em um gerenciador de banco de dados (DBMS), escrito em Python, capaz de operações básicas de SELECT, INSERT, UPDATE e DELETE.

Instalação

Ambiente: Python 3.9+

Dependências:

- tabulate: Mostrar dados em formato tabular
- mysql-connector-python: Importar MySQL
- psycopg[binary]: Importar PostgreSQL

```
pip install -r requirements.txt

# Ou usando gerenciador poetry (recomendado)
poetry install
```

Interface de Usuário

O banco de dados é disponibilizado através de interface de linha de comando (CLI), com 3 comandos para importar dados e um para execução de query SQL. Tabelas são mostradas de forma tabular. Por padrão, selects possuem limite de 100 linhas, podendo ser configuradas pela keyword LIMIT.

```
usage: simple_db.py [-h] [--execute EXECUTE] [--import-csv IMPORT_CSV]
[--import-pg IMPORT_PG] [--import-mysql] [--user USER] [--password PASSWORD]
[--host HOST] [--port PORT]
                    [--database DATABASE]

optional arguments:
  -h, --help            show this help message and exit
  --execute EXECUTE     Execute query
  --import-csv IMPORT_CSV
```

```
                        Import CSV files from directory. Input directory
name
  --import-pg IMPORT_PG
                        Import from Postgres database. Input connection
string (e.g. postgresql://user:password@localhost:5432/database)
  --import-mysql        Import from MySQL database. Use --user, --password,
--host, --port, --database args to specify connection details
  --user USER           User name for MySQL database
  --password PASSWORD   Password for MySQL database
  --host HOST           Host for MySQL database
  --port PORT           Port for MySQL database
  --database DATABASE   Database name for MySQL database
```

Arquitetura

Arquivo base: **simple_db.py**, um arquivo **query_*.py** para cada operação

Todas as operações necessitam dos dados das tabelas afetadas em memória. Quando é necessário ler uma tabela, todas as linhas do seu CSV são lidas do disco, interpretadas de acordo com o tipo das colunas e carregados em memória como objetos Python. Operações subsequentes de filtragem, ordenação, alteração, join, limitação, etc. são efetuadas sobre esses dados, que podem ser então persistidos novamente em disco.

O banco é representado pelas classes *Database, Table* e *Column*, que são representados no arquivo persistido *db_data/meta.json*. Os dados em memória de uma tabela é um *ResultSet*, e é sobre ele que são efetuadas as operações de filtragem, join, limitação, etc.

Em alto nível, o tratamento de queries seguem a seguinte ordem:

1. Detecção da operação (SELECT, INSERT, UPDATE, DELETE)
2. *Parse* do SQL para classe Python equivalente e validação de sintaxe
3. Validação da classe em relação à definição do banco de dados
4. Leitura dos dados em disco
5. Operações em memória (opcional)
6. Persistência dos dados tratados em disco (opcional)
7. Renderização para o usuário

Queries SQL são interpretadas pelo DBMS com algoritmo próprio, que analisa a string sequencialmente, utilizando expressões regulares e condicionais para extrair os tokens e juntá-los em suas classes *Select, Insert, Update, Delete, Where* e *OrderBy*. As *keywords* são

case-insensitive, mas valores em string dentro das queries têm sua capitalização preservada. O algoritmo é maleável em relação à espaços em branco nas queries, como quebras de linha, tab, espaços duplos e falta de espaçamento entre vírgulas e parênteses. Os algoritmos de *parsing* possuem testes unitários em *pytest*.

A ordem de execução do select é: Leitura, Join, Where, Seleção de campos, Order By, Limit.

Operações implementadas

SELECT:

- Seleção de campos ou *
- Inner join: **JOIN**
    - ON: Qualquer comparação sobre as tabelas compostas (mesma comparação de WHERE)
    - USING
    - Uso de nomes explícitos de tabelas nas queries (*employees.emp_no*)
- Filtro: **WHERE**
    - Até 1 where adicional com AND ou OR
    - Operadores >, <, >=, <=, =, !=
- Ordenamento: **ORDER BY**
    - Um campo
    - ASC ou DESC

INSERT

- Uma linha por vez
- Inserção de uma seleção de campos

UPDATE

- Com cláusula **WHERE** opcional

DELETE

- Com cláusula **WHERE** opcional

Tipos de dados

int: Inteiro com precisão ilimitada, conforme implementado no tipo *int* do Python. Persistido em formato texto.

float: Ponto flutuante de dupla precisão (*double*), conforme implementado no tipo *float* do Python. Persistido em formato texto com 4 casas decimais.

str: String com tamanho variável ilimitado, conforme implementado no tipo *str* do Python. Persistido em formato texto UTF-8.

datetime: Data e tempo com precisão de microssegundos. Persistido em formato ISO 8601

Persistência

Um arquivo CSV por tabela, na pasta *db_data*. A primeira coluna de todas as tabelas é *__id*, um inteiro sequencial gerenciado totalmente pelo DBMS.

O *__id* é necessário para correlacionar linhas durante updates e deletes. Foi escolhido gerar um id sequencial pelo DBMS ao invés de utilizar campos existentes na tabela, para evitar as complexidades da importação e definição de chaves primárias, que devem garantir unicidade, podem ser compostas, etc. Para cada tabela, existe um campo *next_id* no arquivo de metadados para evitar ids duplicados.

db_data/departments.csv

```
__id,dept_no,dept_name
0,d009,Customer Service
1,d005,Development
2,d002,Finance
3,d003,Human Resources
4,d001,Marketing
5,d004,Production
6,d006,Quality Management
7,d008,Research
8,d007,Sales
```

db_data/meta.json

```
{
  "database": {
    "name": "employees",
    "tables": [
```

```
      {
        "name": "departments",
        "columns": [
          {
            "name": "__id",
            "type": "int"
          },
          {
            "name": "dept_no",
            "type": "str"
          },
          {
            "name": "dept_name",
            "type": "str"
          }
        ],
        "file": "departments.csv",
        "next_id": 10
      },
...
```

Ingestão de dados

Tabelas podem ser configuradas manualmente no arquivo de metadados ou serem importadas de arquivos CSV, tabelas MySQL ou PostgreSQL.

CSV

Escolher uma pasta contendo arquivos CSV com cabeçalhos. O nome dos arquivos serão os nomes das tabelas importadas. Como arquivos CSV não possuem identificação de tipo para seus campos, o usuário deve cadastrá-los manualmente através do prompt. As linhas originais são então carregadas em memória, convertidas para os tipos internos do banco de dados e persistidas ao final.

MySQL

Utilizando a biblioteca oficial *mysql-connector-python*, é feita uma query pelas tabelas do banco, e para cada tabela suas colunas e seus respectivos tipos, que são correlacionados com os tipos internos. É feita então uma query para todos os itens da tabela original, que são convertidos para os tipos internos e então persistidos em CSV.

O usuário deve fornecer o nome do banco de dados e pode fornecer usuário (root), senha (root), host (localhost), porta (3306).

PostgreSQL

Da mesma forma que para MySQL, porém utilizando a biblioteca *psycopg* para execução das queries. Como as queries e o formato de resposta são ligeiramente diferentes, a implementação é independente, porém segue estrutura semelhante.

O usuário deve fornecer uma connection string no seguinte formato:

postgresql://postgres:123456@localhost/employees

Exemplos de queries

Select simples

```
python simple_db.py --execute "SELECT * FROM EMPLOYEES LIMIT 10"

  emp_no  birth_date    first_name    last_name    gender    hire_date
--------  ------------  ------------  -----------  --------  -----------
   10001  1953-09-02    Georgi        Facello      M         1986-06-26
   10002  1964-06-02    Bezalel       Simmel       F         1985-11-21
   10003  1959-12-03    Parto         Bamford      M         1986-08-28
   10004  1954-05-01    Chirstian     Koblick      M         1986-12-01
   10005  1955-01-21    Kyoichi       Maliniak     M         1989-09-12
   10006  1953-04-20    Anneke        Preusig      F         1989-06-02
   10007  1957-05-23    Tzvetan       Zielinski    F         1989-02-10
   10008  1958-02-19    Saniya        Kalloufi     M         1994-09-15
   10009  1952-04-19    Sumant        Peac         F         1985-02-18
   10010  1963-06-01    Duangkaew     Piveteau     F         1989-08-24
```

Select com Where e Order By

```
python simple_db.py --execute 'SELECT * FROM EMPLOYEES WHERE birth_date >
"1950-01-01" ORDER BY birth_date ASC LIMIT 10'

  emp_no  birth_date           first_name    last_name     gender
hire_date
--------  -------------------  ------------  ------------  --------
-------------------
   65308  1952-02-01 00:00:00  Jouni         Pocchiola     M
1985-03-10 00:00:00
   87461  1952-02-01 00:00:00  Moni          Decaestecker  M
1986-10-06 00:00:00
   91374  1952-02-01 00:00:00  Eishiro       Kuzuoka       M
1992-02-12 00:00:00
  207658  1952-02-01 00:00:00  Kiyokazu      Whitcomb      M
1988-07-26 00:00:00
  237571  1952-02-01 00:00:00  Ronghao       Schaad        M
1988-07-10 00:00:00
  406121  1952-02-01 00:00:00  Supot         Remmele       M
1989-01-27 00:00:00
```

```
   12282  1952-02-02 00:00:00  Tadahiro      Delgrange     M
1997-01-09 00:00:00
   13944  1952-02-02 00:00:00  Takahito      Maierhofer    M
1989-01-18 00:00:00
   22614  1952-02-02 00:00:00  Dung          Madeira       M
1989-01-24 00:00:00
```

Select com Where + And

```
python simple_db.py --execute "SELECT * FROM employees WHERE gender = 'M'
AND hire_date > '1989-01-01' LIMIT 10"

  emp_no  birth_date    first_name    last_name    gender    hire_date
--------  ------------  ------------  -----------  --------  -----------
   10001  1953-09-02    Georgi        Facello      M         1986-06-26
   10002  1964-06-02    Bezalel       Simmel       F         1985-11-21
   10003  1959-12-03    Parto         Bamford      M         1986-08-28
   10004  1954-05-01    Chirstian     Koblick      M         1986-12-01
   10005  1955-01-21    Kyoichi       Maliniak     M         1989-09-12
   10006  1953-04-20    Anneke        Preusig      F         1989-06-02
   10007  1957-05-23    Tzvetan       Zielinski    F         1989-02-10
   10008  1958-02-19    Saniya        Kalloufi     M         1994-09-15
   10009  1952-04-19    Sumant        Peac         F         1985-02-18
   10010  1963-06-01    Duangkaew     Piveteau     F         1989-08-24
```

Select com Where comparando 2 colunas

```
python simple_db.py --execute "SELECT * FROM employees WHERE hire_date >
birth_date LIMIT 10"

  __id    emp_no  birth_date           first_name    last_name    gender
hire_date
------  --------  -------------------  ------------  -----------  --------
-------------------
     0     10001  1953-09-02 00:00:00  Georgi        Facello      M
1986-06-26 00:00:00
     1     10002  1964-06-02 00:00:00  Bezalel       Simmel       F
1985-11-21 00:00:00
     2     10003  1959-12-03 00:00:00  Parto         Bamford      M
1986-08-28 00:00:00
     3     10004  1954-05-01 00:00:00  Chirstian     Koblick      M
1986-12-01 00:00:00
     4     10005  1955-01-21 00:00:00  Kyoichi       Maliniak     M
1989-09-12 00:00:00
     5     10006  1953-04-20 00:00:00  Anneke        Preusig      F
1989-06-02 00:00:00
     6     10007  1957-05-23 00:00:00  Tzvetan       Zielinski    F
1989-02-10 00:00:00
     7     10008  1958-02-19 00:00:00  Saniya        Kalloufi     M
1994-09-15 00:00:00
```

```
     8     10009  1952-04-19 00:00:00  Sumant        Peac         F
1985-02-18 00:00:00
     9     10010  1963-06-01 00:00:00  Duangkaew     Piveteau     F
1989-08-24 00:00:00
```

Select com Join On

```
python simple_db.py --execute 'SELECT departments.dept_no,
dept_manager.emp_no FROM departments JOIN dept_manager ON
dept_manager.dept_no = departments.dept_no WHERE dept_manager.dept_no =
"d006"'

departments.dept_no      dept_manager.emp_no
---------------------  ---------------------
d006                                  110725
d006                                  110765
d006                                  110800
d006                                  110854
```

Select com Join Using

```
python simple_db.py --execute 'SELECT departments.dept_no,
dept_manager.emp_no FROM departments JOIN dept_manager USING (dept_no) WHERE
departments.dept_no = "d006"'

departments.dept_no      dept_manager.emp_no
---------------------  ---------------------
d006                                  110725
d006                                  110765
d006                                  110800
d006                                  110854
```

Insert

```
python simple_db.py --execute "INSERT INTO departments(dept_no, dept_name)
VALUES ('d999', 'Test department')"

Inserted row
```

Update

```
python simple_db.py --execute "UPDATE departments SET dept_name = 'Test
department 2' WHERE dept_no = 'd999'"

Updated 1 row: __id=8
```

Delete

```
python simple_db.py --execute "DELETE FROM departments WHERE dept_no =
'd999'"

Deleted 1 row: __id=9
```