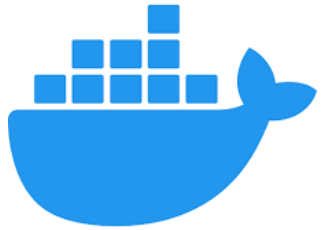
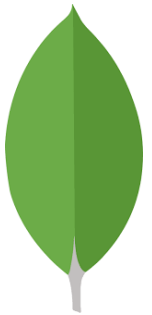
DATABASE APPLICATION





"A LIGA":

CARLOS ALBERTO RAMALHO,

GUSTAVO PORTELA PACHÊCO,

JOSÉ GABRIEL DE OLIVEIRA LINO,

MARCIO SOBEL,

RAFAEL ANTÔNIO RIBEIRO GALVÃO MENDES

ROADMAP

- Escolher Datasets
- Converter datasets para um banco MySQL
- Escolher uma tecnologia NoSQL
- Converter o banco MySQL para a tecnologia escolhida

DATASET ESCOLHIDO

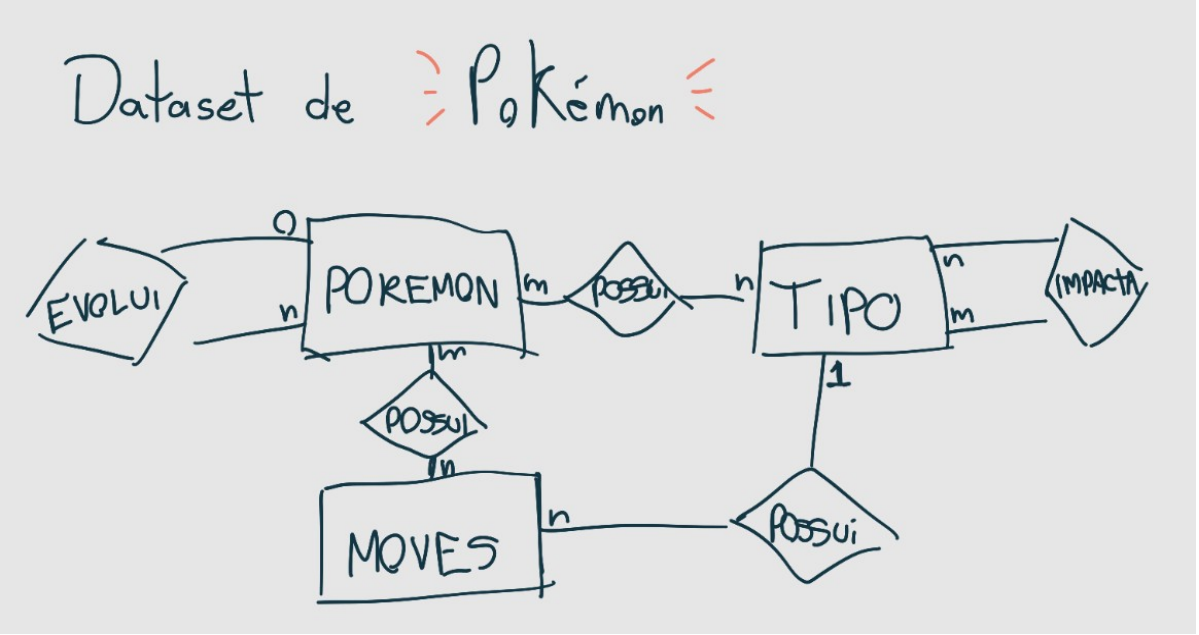
- Pokémon!
- Abstração relativamente simples
- Famoso

TECNOLOGIA ESCOLHIDA

- MongoDB!
- Familiaridade
- Referência NoSQL Documental

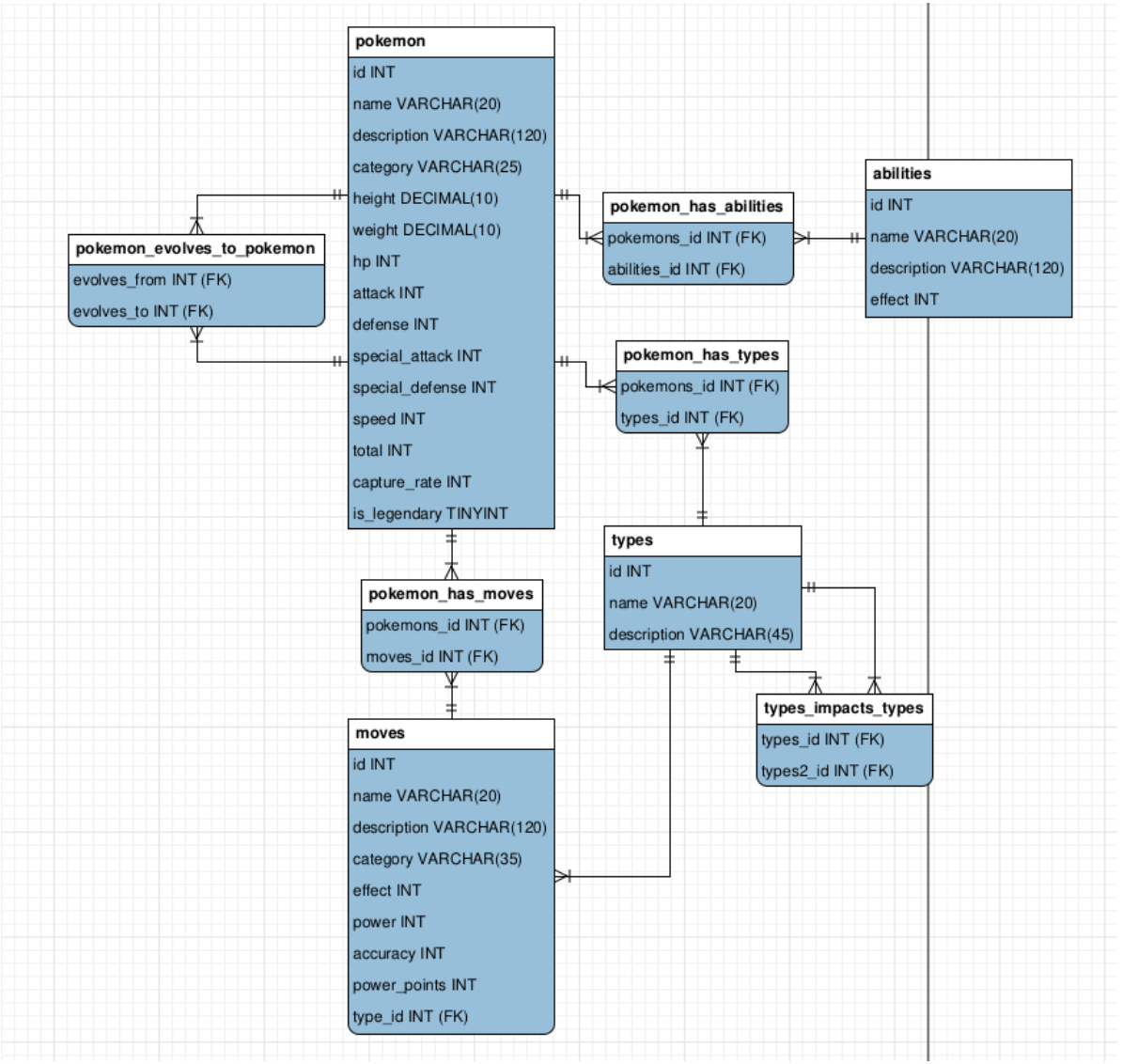
PASSO 1

- Planejar e modelar o banco de dados



PASSO 1

- Planejar e modelar o banco de dados



PASSO 2

- Transformar os arquivos para SQL

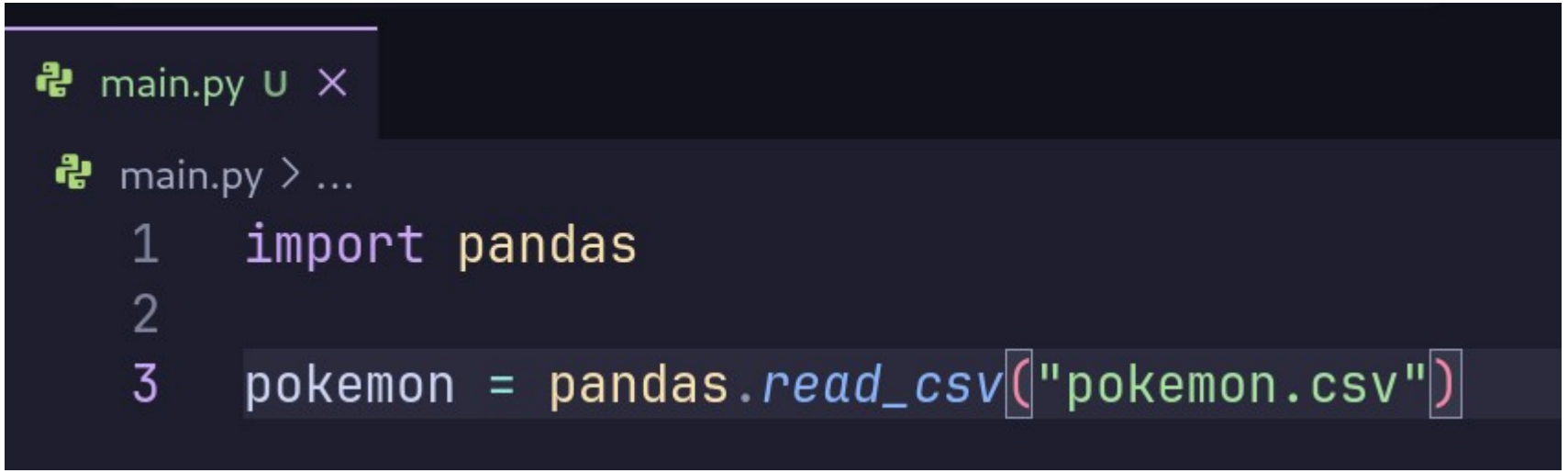
PASSO 2

- Transformar os arquivos para SQL
- Como?

SOBRE O PANDAS

- Leitura de uma variedade de arquivos
- Manipulação complexa com alta flexibilidade
- Permite o envio direto para o servidor SQL

DANDO INÍCIO


main.py U
main.py > ...
1 import pandas
2
3 pokemon = pandas.read_csv("pokemon.csv")

tutorial.py U

src > tutorial.py > ...

```python
import pandas

pokemon = pandas.read_csv("pokemon.csv")
print(pokemon)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
        id  ndex     species                   forme  ... percent-female pre-evolution    egg-group1 egg-group2
0        1     1   Bulbasaur                Bulbasaur  ...          0.125           NaN       Monster      Grass
1        2     2     Ivysaur                  Ivysaur  ...          0.125     Bulbasaur       Monster      Grass
2        3     3    Venusaur                 Venusaur  ...          0.125       Ivysaur       Monster      Grass
3        4     4  Charmander               Charmander  ...          0.125           NaN       Monster     Dragon
4        5     5  Charmeleon               Charmeleon  ...          0.125    Charmander       Monster     Dragon
...    ...   ...         ...                      ...  ...            ...           ...           ...        ...
1056  1058   778     Mimikyu     Mimikyu (Busted Form)  ...          0.500           NaN     Amorphous        NaN
1057  1059   778     Mimikyu               Mimikyu (2)  ...          0.500           NaN     Amorphous        NaN
1058  1060   778     Mimikyu               Mimikyu (3)  ...          0.500           NaN     Amorphous        NaN
1059  1061   784     Kommo-o               Kommo-o (1)  ...          0.500           NaN        Dragon        NaN
1060  1062   801    Magearna  Magearna (Original Color)  ...            NaN           NaN  Undiscovered        NaN

[1061 rows x 26 columns]
```

CRIANDO RELAÇÕES NO PANDAS

tutorial.py U

src > tutorial.py > ...

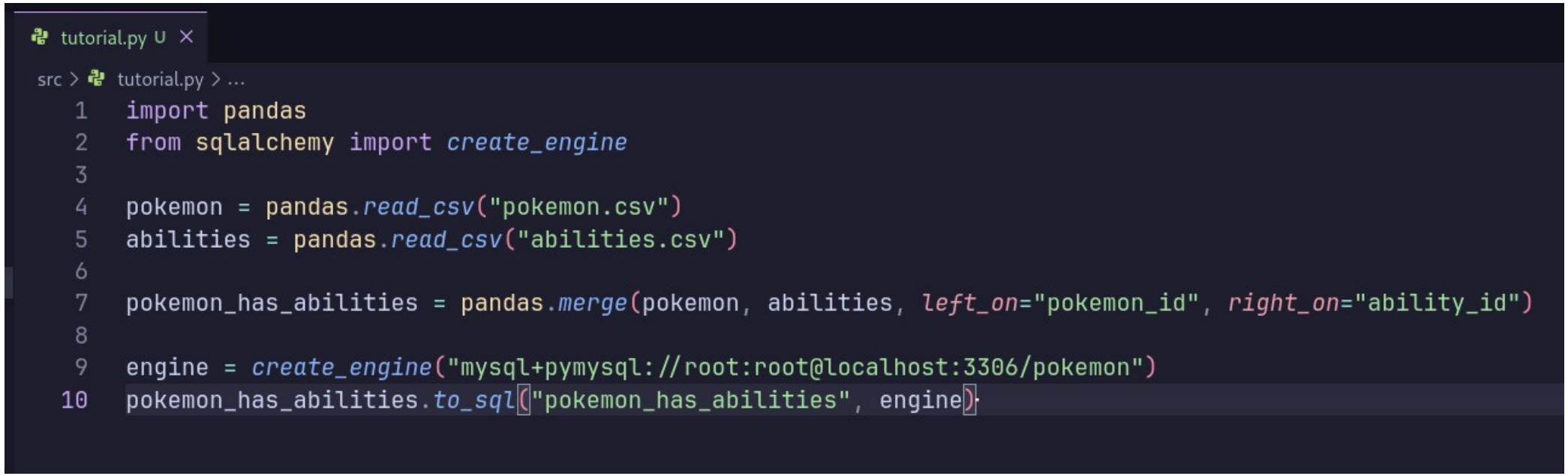
```python
import pandas

pokemon = pandas.read_csv("pokemon.csv")
abilities = pandas.read_csv("abilities.csv")

pokemon_has_abilities = pandas.merge(pokemon, abilities, left_on="pokemon_id", right_on="ability_id")
print(pokemon_has_abilities)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
      pokemon_id  ability_id
0              1          65
1              2          65
2              3          65
5              4          66
6              5          66
...          ...         ...
1006          76         206
1008          88         223
1010          89         223
1076         678         172
1126         745          99

[2007 rows x 2 columns]
```

ENVIANDO PRO SQL

- SQLAlchemy
- PyMySQL

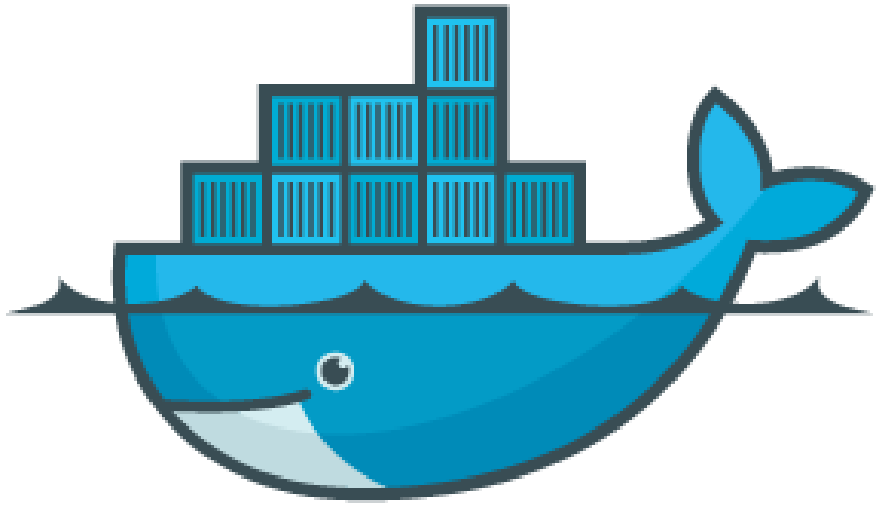

tutorial.py U
src > tutorial.py > ...
1 import pandas
2 from sqlalchemy import create_engine
3
4 pokemon = pandas.read_csv("pokemon.csv")
5 abilities = pandas.read_csv("abilities.csv")
6
7 pokemon_has_abilities = pandas.merge(pokemon, abilities, left_on="pokemon_id", right_on="ability_id")
8
9 engine = create_engine("mysql+pymysql://root:root@localhost:3306/pokemon")
10 pokemon_has_abilities.to_sql("pokemon_has_abilities", engine)

```
sqlalchemy.exc.OperationalError: (pymysql.err.OperationalError) (2003, "Can't connect
 to MySQL server on 'localhost' ([Errno 111] Connection refused)")
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

```
sqlalchemy.exc.OperationalError: (pymysql.err.OperationalError) (2003, "Can't connect
 to MySQL server on 'localhost' ([Errno 111] Connection refused)")
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

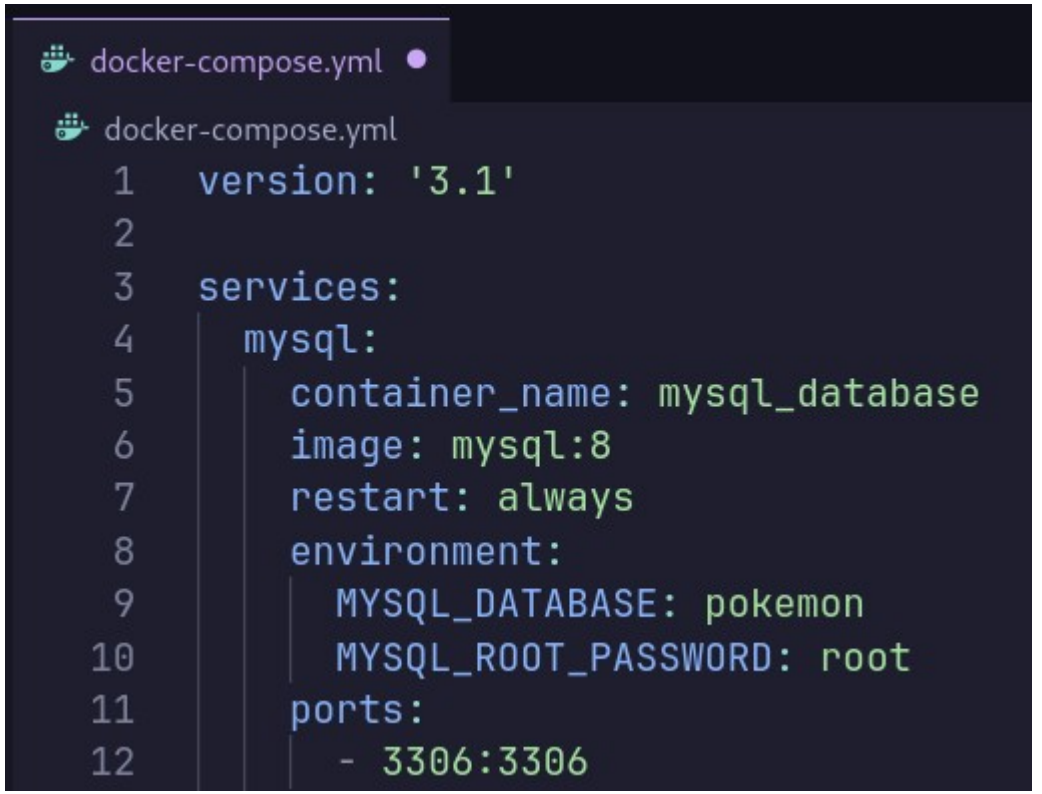
CONHECENDO O DOCKER

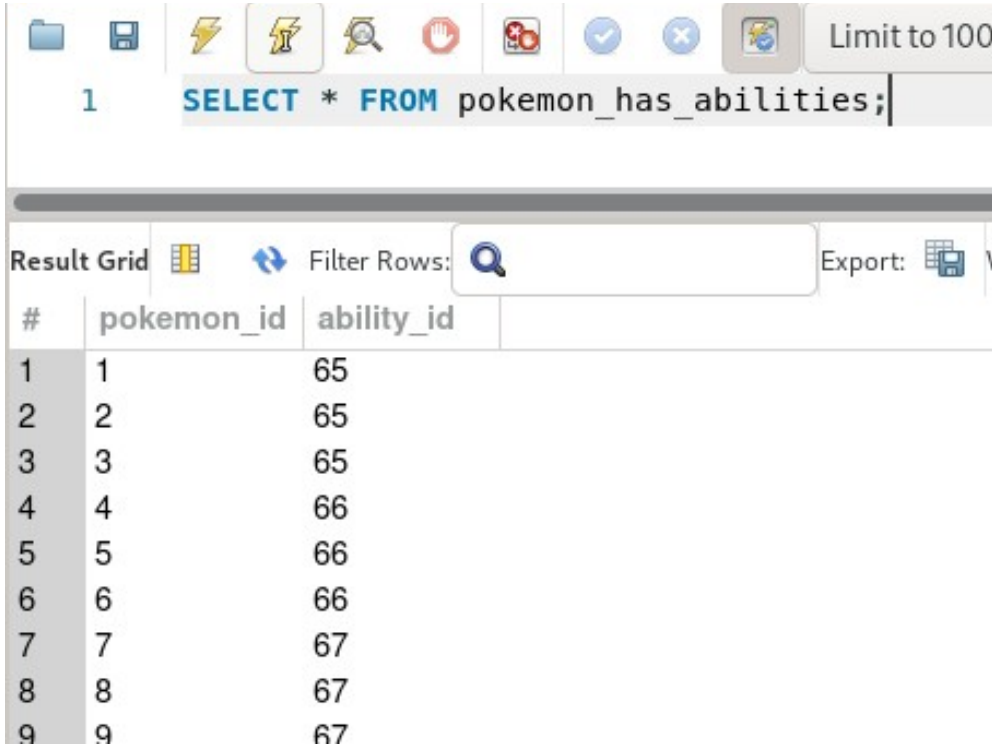
- Virtualização
- Containers isolados
- Fácil compartilhamento
- Configuração "única"



DOCKER COMPOSE

- "Receita" de configurações





#	pokemon_id	ability_id
1	1	65
2	2	65
3	3	65
4	4	66
5	5	66
6	6	66
7	7	67
8	8	67
9	9	67

src > tutorial.py > ...

```python
import pandas
from sqlalchemy import create_engine

pokemon = pandas.read_csv("pokemon.csv")
abilities = pandas.read_csv("abilities.csv")

pokemon_has_abilities = pandas.merge(pokemon, abilities, left_on="pokemon_id", right_on="ability_id")

engine = create_engine("mysql+pymysql://root:root@localhost:3306/pokemon")

pokemon.to_sql("pokemon", engine)
abilities.to_sql("abilities", engine)
pokemon_has_abilities.to_sql("pokemon_has_abilities", engine)

```

```
Limit to 1000 rows
1  SELECT p.pokemon AS pokemon, GROUP_CONCAT(DISTINCT a.ability) AS abilities
2  FROM pokemon p
3  JOIN pokemon_has_abilities pha ON p.pokedex_number = pha.pokemon_id
4  JOIN abilities a ON a.id = pha.ability_id
5  WHERE p.pokemon = 'oshawott'
6  GROUP BY p.pokemon;
```

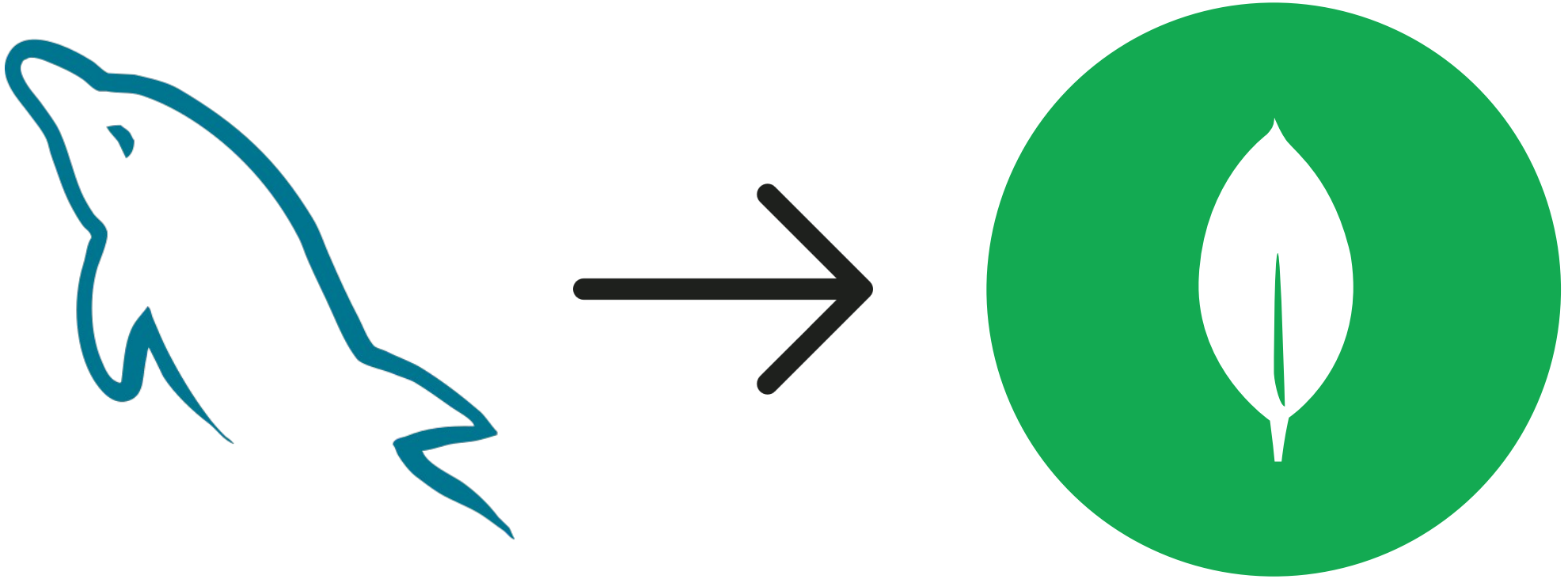
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

#	pokemon	abilities
1	Oshawott	Shell Armor,Torrent

src > convertToServer.py > ...

```python
import ast
import cleanup
import pandas as pd
from getEngine import *
from getDataFrame import *
from resetDatabase import *

def csvToSQL() -> None:
    pokemon = getDataFrame("pokemon")
    abilities = getDataFrame("abilities")
    type_chart = getDataFrame("type-chart")
    types = type_chart.copy()
    moves = getDataFrame("moves")

    pokemonWithMoves = getDataFrame("pokedex")
    pokemonWithMoves["Moves"] = pokemonWithMoves["Moves"].apply(ast.literal_eval)
    pokemonDescriptions = getDataFrame("poki_descs")
    pokemonLegendary = getDataFrame("pokemonLegendary")[["pokedex_number", "legendary", "mythical"]]
    pokemon = pd.merge(pokemon, pokemonWithMoves, left_on="ndex", right_on="Id")
    pokemon = pd.merge(pokemon, pokemonDescriptions, left_on="species", right_on="name", how="left")
    pokemon = pd.merge(pokemon, pokemonLegendary, left_on="ndex", right_on="pokedex_number").drop(columns=["pokedex_number"])

    abilities = createAutoIncrementColumn(abilities, "ability_id")

    types = pokemon[["type1"]].rename(columns={"type1": "type"}).drop_duplicates()
    types = createAutoIncrementColumn(types, "type_id")
    types_lowercase = types["type"].apply(lambda x: pd.Series(str.lower(x))).rename(columns={0: "type"})
    types_lowercase["type_id"] = types["type_id"]

    moves = moves.replace("—", numpy.nan)
    moves["accuracy"] = moves["accuracy"].str.replace("%", "").astype("Int64")
    moves = moves.rename(columns={"id": "move_id"})

    ########## RELATIONSHIPS ##########
    # pokemon has abilities
```

MIGRANDO PARA O MONGODB



DOCKER COMPOSE

```yaml
mongo:
  image: mongo:7
  container_name: mongo_database
  restart: always
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: root
  ports:
    - 27017:27017
```

```python
def get_table(table_name: str) -> pd.DataFrame:
    mysql_engine = create_engine("mysql+pymysql://root:root@localhost:3306/pokemon")
    return pd.read_sql(f"SELECT * FROM {table_name}", mysql_engine)
```

```python
pokemon = get_table("pokemon")
types = get_table("types")
moves = get_table("moves")
abilities = get_table("abilities")

pokemon_has_types = get_table("pokemon_has_types")
pokemon_has_moves = get_table("pokemon_has_moves")
type_effectiveness = get_table("type_effectiveness")
pokemon_has_abilities = get_table("pokemon_has_abilities")
pokemon_evolves_to_pokemon = get_table("pokemon_evolves_to_pokemon")
```

```python
# pokemon_has_types
pokemon = pd.merge(pokemon, pokemon_has_types, left_on="pokedex_number", right_on="pokemon_id")
```

```python
pokemon_group = pokemon.groupby(["pokedex_number", "pokemon"]).aggregate({"type_id": list}).reset_index()
```

```python
pokemon_group = pokemon_group.drop(columns=["pokemon"]).rename(columns={"type_id": "type_ids"})
```

```python
pokemon = (pd.merge(pokemon, pokemon_group, on="pokedex_number")
           .drop(columns=["pokemon_id", "type_id"])
           .drop_duplicates(subset=["pokedex_number"])
)
```

```python
def merge_and_group(
        dataframe: pd.DataFrame,
        relationship_dataframe: pd.DataFrame,
        left_on: str,
        right_on: str,
        groupby: list,
        drop_columns: list,
        old_column_name: str,
        new_column_name: str = "",
    ) -> pd.DataFrame:
    dataframe = pd.merge(dataframe, relationship_dataframe, left_on=left_on, right_on=right_on, how="left")
    # https://stackoverflow.com/questions/64235312/how-to-implodereverse-of-pandas-explode-based-on-a-column
    grouped = dataframe.groupby(groupby).agg({old_column_name: list}).reset_index()
    grouped = grouped.drop(columns=drop_columns)
    if new_column_name:
        grouped = grouped.rename(columns={old_column_name: new_column_name})
        dataframe = pd.merge(dataframe, grouped, on=left_on).drop(columns=[right_on, old_column_name]).drop_duplicates(subset=[left_on])
    else:
        dataframe = pd.merge(dataframe, grouped, on=left_on).drop(columns=[right_on]).drop_duplicates(subset=left_on)
        dataframe = dataframe.drop(columns=[f"{old_column_name}_x"]).rename(columns={f"{old_column_name}_y": old_column_name})
    return dataframe
```

```
######### HANDLE RELATIONSHIPS #########
pivots = ["pokedex_number", "pokemon"]

# pokemon_has_types
pokemon = merge_and_group(pokemon, pokemon_has_types, "pokedex_number", "pokemon_id", pivots, ["pokemon"], "type_id", "type_ids")

# pokemon_has_moves
pokemon = merge_and_group(pokemon, pokemon_has_moves, "pokedex_number", "pokemon_id", pivots, ["pokemon"], "move_id", "move_ids")

# pokemon_has_abilities
pokemon = merge_and_group(pokemon, pokemon_has_abilities, "pokedex_number", "pokemon_id", pivots, ["pokemon"], "ability_id", "ability_ids")

# pokemon_evolves_to_pokemon
pokemon = merge_and_group(pokemon, pokemon_evolves_to_pokemon, "pokedex_number", "evolves_from", pivots, ["pokemon"], "evolves_to")
```

```
######### RENAME ID COLUMNS TO _ID #########
pokemon = pokemon.rename(columns={"pokedex_number": "_id"})
abilities = abilities.rename(columns={"id": "_id"})
types = types.rename(columns={"id": "_id"})
moves = moves.rename(columns={"id": "_id"})

######### CONVERT FROM DATAFRAME TO DICTS #########
pokemon = pokemon.to_dict("records")
types = types.to_dict("records")
moves = moves.to_dict("records")
abilities = abilities.to_dict("records")
```

```
########## SEND TO MONGODB ##########
mongodb_client = MongoClient("mongodb://root:root@localhost:27017")
mongodb_client.drop_database("pokemon")
db = mongodb_client["pokemon"]

db.pokemon.insert_many(pokemon)
db.types.insert_many(types)
db.moves.insert_many(moves)
db.abilities.insert_many(abilities)
db.type_effectiveness.insert_many(type_effectiveness)
```

```
admin> show databases
admin    100.00 KiB
config   108.00 KiB
local     72.00 KiB
pokemon  728.00 KiB
admin>
```

```
admin> use pokemon
switched to db pokemon
pokemon> show collections
abilities
moves
pokemon
type_effectiveness
types
pokemon>
```

VANTAGENS DA ARQUITETURA NoSQL

- Não relacional
- Cresce horizontalmente (Não verticalmente)
- Não é estruturado

RELACIONAMENTOS

Relacionamentos 1-N e N-M são tradados da mesma forma

```
{
    _id: 19,
    move: "Fly",
    tipo_id: 18
},
{
    _id: 45,
    pokemon: "Growl",
    tipo_id: 5
}
```

db.moves.aggregate([{ $match: { $or: [{ move: "Fly" }, { move: "Growl" }] } }])

RELACIONAMENTOS

Relacionamentos 1-N e N-M são tradados da mesma forma

```
{
    _id: 25,
    pokemon: "Pikachu",
    evolui_para: [ 26 ]
},
{
    _id: 133,
    pokemon: "Eevee",
    evolui_para: [ 134, 135, 136
        196, 197, 470, 471, 700]
}
```

db.pokemon.aggregate([{ $match: { $or: [{ pokemon: "Pikachu" }, { pokemon: "Eevee" }] } }])

RELACIONAMENTOS

Tabelas podem ser referenciadas através do ID

```
{
    _id: 25,
    pokemon: "Pikachu",
    evolui_para: [
        {
            _id: 26,
            pokemon: "Raichu"
        }
    ]
}
```

```
db.pokemon.aggregate([
        { $lookup: { from: "pokemon", foreignField: "_id", localField: "evolves_to", as: "evolves_to" } },
        { $match: { pokemon: "Pikachu" } }
])
```