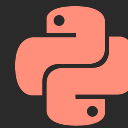




O incrível e pequeno banco
de dados de documentos

Live de Python # 246



1. Banco de Documentos

É de comer?

2. O TinyDB

Um básico introdutório

3. Um CRUD

Criando, pesquisando, alterando, deletando e etc...

4. Além da primeira página

Conceitos importantes e extensões



picpay.me/dunossauro



apoia.se/livedepython



pix.dunossauro@gmail.com



Ajude o projeto <3



Ademar Peixoto, Adilson Herculano, Adriano Ferraz, Alemão, Alexandre Harano, Alexandre Lima, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alfredo Braga, Alisson Souza, Alysson Oliveira, André Azevedo, André Mesquita, André Paula, Aquiles Coutinho, Arnaldo Turque, Aslay Clevisson, Aurelio Costa, Bernardo At, Bernardo Fontes, Bruno Almeida, Bruno Barcellos, Bruno Batista, Bruno Freitas, Bruno Lopes, Bruno Ramos, Caio Nascimento, Carlos Ramos, Christian Semke, Claudemir Firmino, Cristian Firmino, Damianth, Daniel Freitas, Daniel Wojcickoski, Danilo Boas, Danilo Segura, Danilo Silva, David Couto, David Kwast, Davi Goivinho, Davi Souza, Dead Milkman, Delton Porfiro, Denis Bernardo, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino, Diogo Paschoal, Edgar, Eduardo Silveira, Eduardo Tolmasquim, Emerson Rafael, Erick Andrade, Érico Andrei, Everton Silva, Fabiano, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Correa, Fábio Thomaz, Fabricio Biazotto, Fabricio Patrocinio, Felipe Augusto, Felipe Rodrigues, Fernanda Prado, Fernando Celmer, Firehouse, Flávio Meira, Francisco Neto, Francisco Silvério, Gabriel Espindola, Gabriel Mizuno, Gabriel Moreira, Gabriel Paiva, Gabriel Ramos, Gabriel Simonetto, Geanderson Costa, Geizielder, Giovanna Teodoro, Giuliano Silva, Guilherme Beira, Guilherme Felitti, Guilherme Gall, Guilherme Silva, Guionardo Furlan, Gustavo Suto, Haelmo, Haelmo Almeida, Harold Gautschi, Heitor Fernandes, Helvio Rezende, Henrique Andrade, Higor Monteiro, Italo Silva, Janael Pinheiro, Jean Victor, Jefferson Antunes, Joelson Sartori, Jonatas Leon, Jônatas Oliveira, Jônatas Silva, Jorge Silva, José Gomes, Joseíto Júnior, Jose Mazolini, Josir Gomes, Juan Felipe, Juan Gutierrez, Juliana Machado, Julio Franco, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kálita Lima, Kaneson Alves, Leandro Silva, Leandro Vieira, Leonardo Mello, Leonardo Nazareth, Lucas Carderelli, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Pavelski, Lucas Schneider, Lucas Simon, Lucas Valino, Luciano Filho, Luciano Ratamero, Luciano Teixeira, Luis Eduardo, Luiz Duarte, Luiz Lima, Luiz Paula, Luiz Perciliano, Mackilem Laan, Marcelo Araujo, Marcelo Campos, Marcio Moises, Marco Mello, Marcos Gomes, Maria Clara, Marina Passos, Mateus Lisboa, Mateus Ribeiro, Mateus Silva, Matheus Silva, Matheus Vian, Mauricio Fagundes, Mírian Batista, Mlevi Lsantos, Murilo Carvalho, Murilo Viana, Natan Cervinski, Nathan Branco, Ocimar Zolin, Otávio Carneiro, Patricia Minamizawa, Patrick Felipe, Pedro Henrique, Pedro Pereira, Peterson Santos, Pytonyc, Rafael Faccio, Rafael Lopes, Rafael Romão, Raimundo Ramos, Ramayana Menezes, Regis Santos, Renato José, Renato Oliveira, Renê Barbosa, Rene Bastos, Rene Pessoto, Ricardo Silva, Riverfount, Rjribeiro, Robson Maciel, Rodrigo Barretos, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Vaccari, Rodrigo Vieira, Rui Jr, Samanta Cicilia, Samuel Santos, Selmison Miranda, Téo Calvo, Thiago Araujo, Thiago Borges, Thiago Curvelo, Thiago Souza, Tony Dias, Tyrone Damasceno, Vinícius Costa, Vinicius Stein, Walter Reis, William Vitorino, Willian Lopes, Wilson Duarte, Wilson Neto, Yros Aguiar, Zeca Figueiredo



Obrigado você



SQL para quê?

Docum
entos

Bancos de dados



Quando pensamos em bancos de dados, geralmente aprendemos a pensar em tabelas, dados tabulares.

Nome	Email	Telefone
Eduardo	eu@dunossauro.live	999999999999
Fausto	fausto@dunossauro.live	998888888888

Embora isso seja comum, geralmente existe um planejamento para que isso possa ser tabular, **normalizado** e também **evoluído**

Um problema comum!



Se em um determinado dia, tivermos que armazenar dois telefones, como faríamos?

Nome	Email	Telefone
Eduardo	eu@dunossauro.live	999999999999
Fausto	fausto@dunossauro.live	998888888888

Um problema comum!




Se em um determinado dia, tivermos que armazenar dois telefones, como faríamos?

Nome	Email	Telefone	Telefone 2
Eduardo	eu@dunossauro.live	999999999999	NULL
Fausto	fausto@dunossauro.live	998888888888	997777777777

Um problema comum!

Uma solução tradicional



ID	UID	Telefone	Telefone 2
1	1	999999999999	NULL
2	1	998888888888	997777777777

ID	Nome	Email
1	Eduardo	eu@dunossauro.live
2	Fausto	fausto@dunossauro.live

Um problema comum!

ID	Nome	Email
1	Eduardo	eu@dunossauro.live
2	Fausto	fausto@dunossauro.live

ID	UID	Rua	Número	CEP
1	1	Avenida ..	42	33333333
2	2	Rua ...	982	22222222

ID	UID	Telefone	Telefone 2
1	1	999999999999	NULL
2	2	998888888888	997777777777

ID	UID	Cargo	Departamento
1	1	1	1
2	2	2	5

ID	Cargo
1	QA
2	Dev

Orientado a documentos

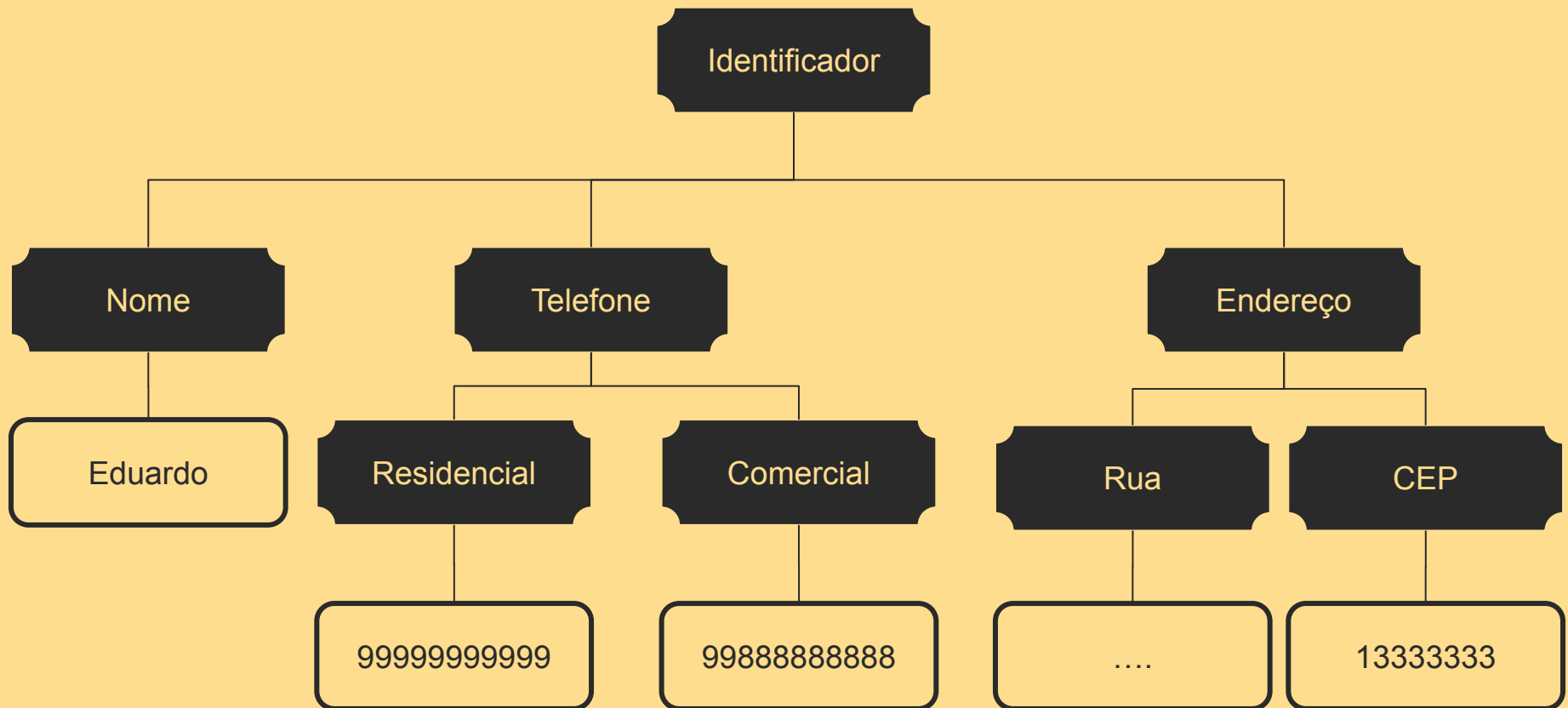


Normalmente pensamos em tabelas e esquemas de banco.

Tabelas são "inflexíveis", pois a estrutura tem que ser sempre mantidas, mantendo uma "estrutura" padrão.

A ideia dos documentos é ser mais flexível.

Dados semi-estruturados



Tipos semiestrutturados



- Json
- Bson
- XML
- YAML
- ...

Json



JavaScript Object Notation é talvez a forma mais popular que temos como exemplos de dados semiestruturados.

```
{  
  "field": "value",  
  "chave": "valor",  
  "nome": "Eduardo"  
}
```

```
{  
  "nome": "Eduardo",  
  "telefones": ["999999999", "888888888"]  
  "endereço": {  
    "logradouro": "Rua dos bobos",  
    "numero" : 0  
  }  
}
```

```
{  
  "nome": "Eduardo",  
  "telefones": ["999999999", "888888888"] # array  
  "endereço": { # sub documento  
    "logradouro": "Rua dos bobos",  
    "numero" : 0  
  }  
}
```


Tiny DB

0 básico
introdutório



TinyDB é um banco de dados focado em ser pequeno, totalmente escrito em Python **puro**, sem nenhuma dependência externa, extensível e com uma API com foco na "felicidade de quem usa".

- **Criador:** Markus Siemens
- **Commit inicial:** Julho de 2013
- **Release Atual:** 4.8.0, lançada em junho de 2023
- **Licença:** MIT
- **Compatível com 3.7+ e PyPy**

História



TinyDB foi criado pelo Markus ainda durante a graduação em engenharia elétrica durante um trabalho de Freelance para selecionar cerca de 6.000 fotos (em EXIF) de diferentes fotógrafos.

Ele procurou algumas alternativas "autocontidas" (sem dependências) nada era simples o suficiente e ele acabou criando um "super dicionário" com diversas funções para facilitar o manuseio das imagens. Isso acabou se tornando o TinyDB.



`pip install tinydb`



Instalação



0 básico [exemplo_00.py]



O TinyDB consiste em um arquivo JSON ***aberto***. Caso o arquivo exista abre o arquivo, caso não, ele criará o arquivo.

```
1  from tinydb import TinyDB
2
3  database = TinyDB('database.json')
```

0 básico [exemplo_01.py]



A inserção básica de dados!

```
1  from tinydb import TinyDB
2
3  database = TinyDB('database.json')
4
5  id_ = database.insert(
6      {'data': 1, 'value': [1, 2, 3], 'name': 'dunossauro'}
7  )
8
9  print(id_)
```

0 básico [exemplo_01.py]



A inserção básica de dados!

```
1  from tinydb import TinyDB
2
3  database = TinyDB('database.json')
4
5  id_ = database.insert(
6      {'data': 1, 'value': [1, 2, 3], 'name': 'dunossauro'}
7  )
8
9  print(id_)
```

Os valores devem ser possíveis de serem convertidos em json

0 básico [exemplo_02.py]



Objetos que não podem ser serializados em JSON resultarão em **TypeError**

```
1  from datetime import datetime
2  from tinydb import TinyDB
3
4  database = TinyDB('database.json')
5
6  database.insert({'datetime': datetime.now()})
7  # TypeError:
8  # Object of type datetime is not JSON serializable
```


Operações padrões [exemplo_03.py]



O objeto TinyDB implementa diversas operações simples, mas bastante importantes:

- TinyDB.**all**(): Mostra todos os valores no banco
- TinyDB.**truncate**(): Deleta todos os valores da base
- **iter**(TinyDB): Itera sobre todos os valores da base (pode ser um **for**)

```
1  from tinydb import TinyDB
2
3  db = TinyDB('database.json')
4
5  db.insert_multiple([
6      {'nome': 'Dunossauro', 'idade': 30},
7      {'nome': 'Fausto', 'idade': 7},
8      {'nome': 'Maria'},
9      {'nome': 'Martha', 'hobbies': ['tomar uma', 'fazer bolos']},
10 ])
11
12 print(db.all())
13
14 for doc in db:
15     print(doc)
16
17 db.truncate()
18
19 print(db.all())
```

Propriedades de JSON [exemplo_04.py]



A forma padrão de armazenamento do Tiny é usando o objeto json do python, então algumas propriedades podem ser passadas a ele. Isso se refletirá no arquivo.

```
1  from tinydb import TinyDB
2
3  db = TinyDB(
4      'database.json',
5      indent=4,
6      sort_keys=True
7  )
```

CRUD

Criando, pesquisando,
alterando, deletando e etc...

Operações



Quando pensamos em operações de banco, falamos em **CRUD** de dados

- **Criação:** Como inserir registros na base
 - TinyDB.**insert**(): Insere um documento
 - TinyDB.**insert_multiple**(): Insere N documentos
- **Recuperação:**
 - TinyDB.**search**(): Busca por dados usando uma **Query**
 - TinyDB.**get**(doc_id=1): Busca dados por ID
 - TinyDB.**all**(): Recupera todos os dados
- **Atualização (Update):**
 - TinyDB.**update**(campos, **Query**): Atualiza registros
- **Deleção:**
 - TinyDB.**remove**(**Query**): Remove registros
 - TinyDB.**tuncate**(): Deleta a base toda

Linguagem de Busca (Query language) [exemplo_05.py]



Parte importante na hora de recuperar, alterar e deletar dados é como encontrar eles na base de dados. Para isso precisamos fazer buscas ou Queries

```
1  from tinydb import TinyDB, Query
2  database = TinyDB('database.json')
3
4  database.insert_multiple([
5      {'data': 1}, {'data': 2},
6      {'data': 3, 'person': 'José'}
7  ])
8
9  Busca = Query()
10
11  database.search(Busca.data >= 2)
12  # [{'data': 2}, {'data': 3, 'person': 'José'}]
```

Query Language



O objeto Query nos trás diversas abstrações interessantes:

- Query.**nome_do_atributo.sub_atributo**...
- Query['**atributo**']

Isso pode ser usado com qualquer operador de comparação:

- ==
- !=
- <
- >
- ...

```
Busca = Query( )
```

```
database.search(Busca.data >= 2)
```

```
database.search(Busca.data == 2)
```

```
database.search(Busca.data != 2)
```

Modificadores



Modificadores permitem aplicar condições lógicas nas buscas

```
Busca = Query()
```

```
# Negação (quem não chama John)
```

```
db.search(~ (User.name == 'John'))
```

```
# E (quem se chama John e tem 30 ou mais)
```

```
db.search((User.name == 'John') & (User.age <= 30))
```

```
# Ou (Quem se chama John ou Bob)
```

```
db.search((User.name == 'John') | (User.name == 'Bob'))
```


Funções prontas



- Query.**atributo.exists()**: Lista somente os que tem X atributo
- Query.**atributo.matches**('[aZ]*'): Lista usando expressão regular
- Query().**fragment**({ 'foo': True, 'bar': False }): Procura por matches
- Query.**atributo.any**(['valor_1', 'valor_2']): Procura qualquer valor
- Query.**atributo.all**(['valor_1', 'valor_2']): Procura quem tenha a lista de valores
- Query.**atributo.one_of**(['valor_1', 'valor_2']): Procura quem tenha uma das chaves

Where



A função **where** simplifica a feitura de buscas, pois não é necessário chamar o objeto **Query**:

```
from tinydb import TinyDB, where

database = TinyDB('database.json')

database.insert_multiple([{'data': 1}, {'data': 2}])

db.search(where('data') >= 1)
db.search(~(where('data') == 1))
```

Update



Agora que sabemos fazer as buscas, podemos atualizar registros.

```
db = TinyDB('database.json')

db.update({'person': 'Fausto'}, Busca.data == 1)

db.update_multiple([
    ({'person': 'Fausto'}, Busca.data == 1),
    ({'data': 1}, where('person') == 'José'),
])

db.upsert( # Atualiza, caso não acha insere! <3
    {'name': 'Dudu', 'logged-in': True}, Busca.name == 'Fausto'
)
```



Operações para Updates

Em alguns momentos não queremos mudar somente os campos existentes, queremos adicionar, deletar e etc.. Para isso temos os operadores

```
from tinydb import TinyDB, where
from tinydb import operations

db = TinyDB('database.json')

db.update(operations.delete('idade'), where('nome') == 'Fausto')

db.update(
    operations.set('hobbies', ['andar de skate']),
    where('data') == 1
)
```



Operações para Updates

Em alguns momentos não queremos adicionar, deletar e etc.. Pa

```
from tinydb import TinyDB, where
from tinydb import operations
```

```
db = TinyDB('database.json')
```

```
db.update(operations.delete('idade'), where('nome') == 'Fausto')
```

```
db.update(
    operations.set('hobbies', ['andar de skate']),
    where('data') == 1
)
```

Existem outros operadores:

- set: Para atribuir um valor
- increment: para adicionar mais 1
- decrement: para subtrair 1
- subtract: butrai N para o campo
- add: Adiciona N para o campo

Delete



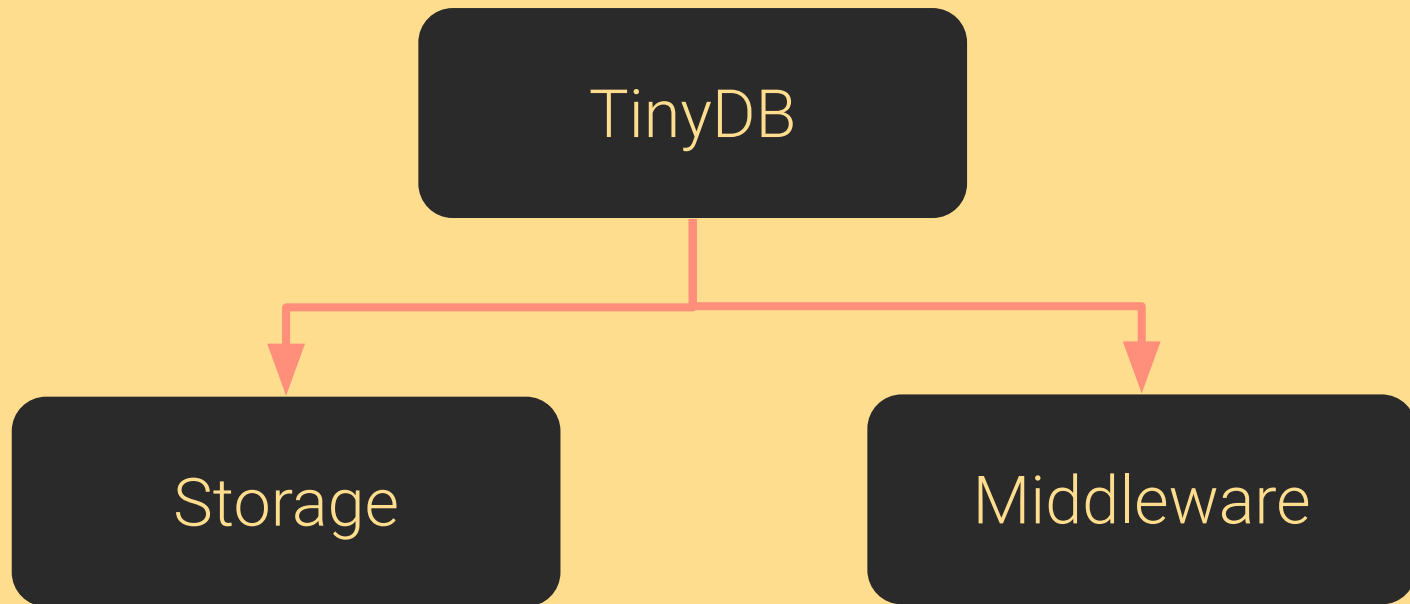
Para deletar registros temos o método **remove**

```
from tinydb import TinyDB, where  
  
db = TinyDB( 'database.json' )  
  
db.remove(where( 'nome' ) == 'Fausto' )
```

Além do CRUD

Indo além da
primeira página

Pequeno, mas plugável



Tipos de armazenamento (Storages)



Embora por padrão seja usado JSON para armazenar os dados, o Tiny também conta com diversos outros tipos de armazenamento.

Padrões:

- **MemoryStorage**: Volátil, quando a aplicação acaba, o banco é destruído
- **JSONStorage**: O padrão, cria um arquivo JSON

Extensões:

- **BetterJSONStorage**: Storage com Orjson (+10x de velocidade) e compactado com Blosc2

Storage em memória



```
1  from tinydb.storages import MemoryStorage
2
3  db = TinyDB(storage=MemoryStorage)
```

Orjson



```
1  from pathlib import Path
2  from tinydb import TinyDB
3  from BetterJSONStorage import BetterJSONStorage
4
5  path = Path('relative/path/to/file.db')
6
7  with TinyDB(path, access_mode="r+", storage=BetterJSONStorage) as db:
8      db.insert({'int': 1, 'char': 'a'})
9      db.insert({'int': 1, 'char': 'b'})
```

pip install BetterJSONStorage

Cache



```
1  from tinydb.storages import JSONStorage
2  from tinydb.middlewares import CachingMiddleware
3  db = TinyDB(
4      '/path/to/db.json', storage=CachingMiddleware(JSONStorage)
5  )
```

Serialização [pip install tinydb-serialization]



— □ ×

```
1  from tinydb import TinyDB, Query
2  from tinydb.storages import JSONStorage
3  from tinydb_serialization import SerializationMiddleware
4  from tinydb_serialization.serializers import DateTimeSerializer
5  from datetime import datetime
6
7  serialization = SerializationMiddleware(JSONStorage)
8  serialization.register_serializer(DateTimeSerializer(), 'TinyDate')
9
10 db = TinyDB('db.json', storage=serialization)
11 db.insert({'date': datetime(2000, 1, 1, 12, 0, 0)})
12 db.all()
13 # [{'date': datetime.datetime(2000, 1, 1, 12, 0)}]
```

Por que usar o TinyDB?

- **minúsculo:** O código-fonte atual tem 1.800 linhas de código (com cerca de 40% documentação) e testes de 1600 linhas.
- **orientado a documentos:** como [o MongoDB](#), você pode armazenar qualquer documento (representado como `dict`) no TinyDB.
- **otimizado para sua felicidade:** TinyDB foi projetado para ser simples e divertido de usar, fornecendo uma API simples e limpa.
- **escrito em Python puro:** TinyDB não precisa de um servidor externo (como por exemplo, [PyMongo](#)) nem quaisquer dependências do PyPI.
- **funciona em Python 3.5+ e PyPy:** TinyDB funciona em todas as versões modernas de Python e PyPy.
- **potentemente extensível:** você pode estender facilmente o TinyDB criando novos storages ou modificar o comportamento dos storages com Middlewares.
- **Cobertura de teste de 100%:** Nenhuma explicação necessária.

Resumindo: se você precisa de um banco de dados simples com uma API limpa que simplesmente funcione sem muita configuração, o TinyDB pode ser a escolha certa para você.

<https://tinydb.readthedocs.io/en/latest/intro.html#why-use-tinydb>

Por que **não** usar o TinyDB?

- Você precisa de **recursos avançados** como:
 - acesso de vários processos ou threads (por exemplo, ao usar Flask!),
 - criando índices para tabelas,
 - um servidor HTTP,
 - gerenciar relacionamentos entre tabelas ou similares,
 - Garantias ACID .
- Você está realmente preocupado com **o desempenho** e precisa de alta velocidade base de dados.

Resumindo: se você precisa de recursos avançados ou alto desempenho, TinyDB é o banco de dados errado para você – considere usar bancos de dados como SQLite , Buzhug , CodernityDB ou MongoDB .

<https://tinydb.readthedocs.io/en/latest/intro.html#why-not-use-tinydb>



picpay.me/dunossauro



apoia.se/livedepython



pix.dunossauro@gmail.com



Ajude o projeto <3



Referências

- Documentação: <https://tinydb.readthedocs.io/en/latest/index.html>
- Entrevista com Markus Siemens: <https://youtu.be/d5WL6OuVx90>
- Artigo do Markus: <https://m-siemens.de/blog/2016/01/tinydb-past-present-and-future>