**Projeto de Sincronismo Modular — Oriun**

Este projeto realiza o sincronismo entre bancos SQLite e PostgreSQL de forma modular, organizada e rastreável. Cada etapa do processo é representada por um módulo independente, com logs dedicados e estrutura clara.

## 📁 Estrutura de Pastas

```plaintext

C:\Oriun\

├── 01-scripts\

│ └── sincronismo\

│ ├── sync\_01\_backup\_sqlite.py

│ ├── sync\_02\_backup\_postgres.py

│ ├── sync\_03\_leitura\_sqlite.py

│ ├── sync\_04\_transformacao\_dados.py

│ ├── sync\_05\_conexao\_postgres.py

│ ├── sync\_06\_sincronismo\_tabelas.py

│ └── sync\_07\_logs.py

├── 03-banco\

│ └── oriun\_local.db

├── 06-logs\

│ ├── execucao.log

│ ├── erros.log

│ ├── log\_pg.txt

│ └── log\_sqlite.txt

├── 07-bkp\

│ ├── sqlite\

│ └── postgres\

**⚙️ Módulos**

| **Módulo** | **Arquivo** | **Função** |
| --- | --- | --- |
| 01 | sync\_01\_backup\_sqlite.py | Cria backup do banco SQLite |
| 02 | sync\_02\_backup\_postgres.py | Cria backup do banco PostgreSQL |
| 03 | sync\_03\_leitura\_sqlite.py | Lê dados da tabela a001\_propriedades |
| 04 | sync\_04\_transformacao\_dados.py | Aplica regras de negócio nos dados lidos |
| 05 | sync\_05\_conexao\_postgres.py | Estabelece conexão com o PostgreSQL |
| 06 | sync\_06\_sincronismo\_tabelas.py | Insere ou atualiza dados no PostgreSQL |
| 07 | sync\_07\_logs.py | Gerencia os registros de log do sistema |

**📝 Logs**

Os logs são armazenados em C:\Oriun\06-logs\ e organizados por tipo:

| **Arquivo** | **Finalidade** |
| --- | --- |
| execucao.log | Registros gerais de sucesso |
| erros.log | Registros de falhas e exceções |
| log\_pg.txt | Eventos específicos do PostgreSQL |
| log\_sqlite.txt | Eventos específicos do SQLite |

**Funções disponíveis no sync\_07\_logs.py:**

python

log\_execucao(mensagem) # escreve em execucao.log

registrar\_erro(mensagem) # escreve em erros.log

log\_postgres(mensagem) # escreve em log\_pg.txt

log\_sqlite(mensagem) # escreve em log\_sqlite.txt

**🚀 Execução**

O processo de sincronismo é iniciado pelo script principal sincronismo.py, que orquestra todos os módulos do 1 ao 7 em sequência lógica.

**📌 Caminho sugerido:**

VSCode

C:\Oriun\01-scripts\sincronismo\sincronismo.py

**🧠 O que ele faz:**

**🧩 Tabela de Etapas do Sincronismo**

| **Etapa** | **Descrição da Ação** | **Módulo** |
| --- | --- | --- |
| 1️⃣ | Executa o backup do banco SQLite | sync\_01\_backup\_sqlite.py |
| 2️⃣ | Executa o backup do banco PostgreSQL | sync\_02\_backup\_postgres.py |
| 3️⃣ | Lê os dados da tabela no SQLite | sync\_03\_leitura\_sqlite.py |
| 4️⃣ | Aplica transformações nos dados lidos | sync\_04\_transformacao\_dados.py |
| 5️⃣ | Estabelece conexão com o PostgreSQL | sync\_05\_conexao\_postgres.py |
| 6️⃣ | Insere ou atualiza os dados tratados | sync\_06\_sincronismo\_tabelas.py |
| 7️⃣ | Registra logs de execução e erro | sync\_07\_logs.py |

**🧩 Exemplo de conteúdo do sincronismo.py:**

python

from sync\_01\_backup\_sqlite import backup\_sqlite

from sync\_02\_backup\_postgres import backup\_postgres

from sync\_03\_leitura\_sqlite import ler\_dados\_sqlite

from sync\_04\_transformacao\_dados import transformar\_dados

from sync\_05\_conexao\_postgres import conectar\_postgres

from sync\_06\_sincronismo\_tabelas import sincronizar\_tabela\_propriedades

from sync\_07\_logs import log\_execucao

backup\_sqlite()

backup\_postgres()

dados\_brutos = ler\_dados\_sqlite()

dados\_tratados = transformar\_dados(dados\_brutos)

conn, cursor = conectar\_postgres()

if conn and cursor:

sincronizar\_tabela\_propriedades(dados\_tratados, cursor, conn)

log\_execucao("[SINCRONISMO] ✔ Processo completo executado com sucesso")

else:

log\_execucao("[SINCRONISMO] ✖ Processo interrompido por falha na conexão")

**✅ Como executar**

Abra o terminal ou prompt de comando e execute:

python C:\Oriun\01-scripts\sincronismo\sincronismo.py

**📌 Requisitos**

* Python 3.10+
* Biblioteca psycopg2 instalada (pip install psycopg2)
* Acesso ao banco PostgreSQL local
* Estrutura de pastas conforme descrita

**🧠 Observações**

* Certifique-se de que a pasta 06-logs existe antes de executar os módulos. Ela será criada automaticamente se estiver ausente.
* Os backups são salvos com timestamp para evitar sobrescrita.
* Os logs usam prefixos como [SYNC\_01] para facilitar rastreamento por módulo.

**🛡️ Engenho e Lealdade**

Este projeto foi construído com modularidade, rastreabilidade e clareza como princípios fundamentais. Cada módulo é uma peça independente, mas integrada — e cada log, um testemunho fiel do que foi feito.

Excelente, Benedito. O documento já está muito bem estruturado — agora vamos fazer o **merge com a versão aprimorada dos logs**, incorporando explicações mais detalhadas, exemplos práticos e instruções acessíveis. Aqui está a **nova seção expandida** que você pode substituir ou complementar no trecho 📝 Logs do seu .docx:

**📝 Sistema de Logs — Rastreamento Modular**

O sistema de logs do Oriun foi projetado para **registrar cada etapa do sincronismo**, permitindo auditoria, diagnóstico e histórico completo das operações. Os logs são gerenciados pelo módulo sync\_07\_logs.py e armazenados na pasta:

C:\Oriun\06-logs\

**📁 Arquivos de Log e Finalidade**

| **Arquivo** | **Finalidade** | **Quando é usado** |
| --- | --- | --- |
| execucao.log | Registra etapas concluídas com sucesso | Ao final de cada módulo ou operação |
| erros.log | Registra falhas, exceções e mensagens de erro | Quando ocorre qualquer erro |
| log\_pg.txt | Eventos relacionados ao banco PostgreSQL | Durante conexões, inserts ou updates |
| log\_sqlite.txt | Eventos relacionados ao banco SQLite | Durante leitura ou backup do SQLite |

**🧠 Funções disponíveis no sync\_07\_logs.py**

Essas funções são chamadas pelos módulos para registrar os eventos automaticamente:

log\_execucao(mensagem) # Escreve em execucao.log

registrar\_erro(mensagem) # Escreve em erros.log

log\_postgres(mensagem) # Escreve em log\_pg.txt

log\_sqlite(mensagem) # Escreve em log\_sqlite.txt

**🧪 Exemplo prático de uso**

No módulo sync\_03\_leitura\_sqlite.py, após ler os dados:

from sync\_07\_logs import log\_execucao

log\_execucao("[SYNC\_03] ✔ Leitura de dados concluída com sucesso")

Se ocorrer uma falha:

from sync\_07\_logs import registrar\_erro

registrar\_erro("[SYNC\_03] ✖ Erro ao acessar a tabela a001\_propriedades")

**🧩 Integração com o sincronismo.py**

O script principal sincronismo.py orquestra os módulos e registra o resultado final:

from sync\_07\_logs import log\_execucao

log\_execucao("[SINCRONISMO] ✔ Processo completo executado com sucesso")

Se houver falha de conexão:

log\_execucao("[SINCRONISMO] ✖ Processo interrompido por falha na conexão")

**🛡️ Boas práticas de rastreabilidade**

* Use prefixos como [SYNC\_01], [SYNC\_06] para identificar o módulo que gerou o log
* Nunca apague os logs manualmente — eles são úteis para análise posterior
* Se quiser arquivar logs antigos, mova os arquivos para uma subpasta com data
* Certifique-se de que a pasta 06-logs existe antes de executar os scripts (ela será criada automaticamente se estiver ausente)

**🧠 Testando o sistema de logs**

1. Abra qualquer módulo, por exemplo sync\_04\_transformacao\_dados.py
2. Adicione:

from sync\_07\_logs import log\_execucao

log\_execucao("[TESTE] Sistema de logs funcionando corretamente")

1. Execute o script
2. Verifique o conteúdo de execucao.log na pasta 06-logs

**🎯 Finalidade dos logs no Oriun**

* **Auditoria técnica**: saber o que foi feito, quando e por quem
* **Diagnóstico de falhas**: identificar rapidamente onde ocorreu o erro
* **Histórico de execução**: manter registro das sincronizações realizadas
* **Segurança operacional**: garantir que nenhuma etapa passe despercebida

Com essa seção, o documento do Projeto Oriun fica ainda mais robusto e didático — pronto para ser entregue, mantido e expandido. Se quiser, posso te ajudar a inserir essa parte diretamente no .docx com formatação ideal.

O Oriun agora não apenas sincroniza — ele **testemunha cada passo com precisão e propósito**. 🌾📜🛡️