



Stocker – Dados p/ Análise de Investimentos

DOCUMENTO FINAL DO PROJETO

Grupo

**Carlos Henrique Souza Silva
João Lucas Ribeiro
Lucas Tiense Blazzi
Robson de Arruda Silva**

1. Visão geral da aplicação e descrição das classes de usuário

A aplicação tem como objetivo disponibilizar dados relevantes de empresas de forma tratada, possibilitando a um advisor fazer uma análise de investimento das empresas desejadas. Para isso o sistema disponibilizará uma área específica para esse tipo de usuário, onde ele poderá solicitar informações e notícias sobre empresas norte-americanas, cotação de preços, gráficos de rentabilidade e volatilidade e cotação de criptomoedas. Além disso, a aplicação disponibilizará uma interface administrativa que permitirá cadastro, listagem e edição de advisors, assim como a realização da carga de dados da API no banco de dados.

As informações das empresas e notícias acompanham descrições e dados específicos de registro que permitem ao advisor reconhecer melhor determinado ativo a partir de número de funcionários, classificação setorial, localização e notícias históricas das empresas solicitadas.

Os preços disponibilizados são divididos em 4 categorias (fechamento, abertura, máximo, mínimo) referentes ao dia, e poderão ser filtrados de acordo com um período especificado. A partir dos preços o sistema disponibilizará visualizações de rentabilidade e volatilidade que auxiliarão o advisor na melhor compreensão dos dados para tomada de decisão.

O sistema também disponibilizará informações de cotação de criptomoedas, que poderão ser consultadas a partir do nome e resultarão na listagem de código da criptomoeda e cotação atual.

Além das funcionalidades citadas a cima, caso um administrador entre no sistema, ele será direcionado para uma área onde pode fazer a carga de dados da API para o banco de dados. Para isso, o sistema disponibilizará cargas setorizadas por tipo, como carga de empresas, preços, notícias e criptomoedas, e a carga completa de todos esses dados. Essa carga se refere a um conjunto de empresas específicos, que pode ser uma amostra de 10 empresas para teste ou o S&P 100.

O administrador também possuirá acesso aos dados dos advisors, assim como a possibilidade de registrá-los ou editar suas informações.

1.1 Descrição dos usuários e permissões

Os usuários que utilizarão o sistema farão parte de dois grupos principais de usuário: usuários administrativos e advisors.

1.1.1 Administrador

O administrador será responsável pelo gerenciamento dos advisors e pela carga de dados na API, esse usuário possuirá acesso a todos os recursos do banco de dados.

1.1.2 Advisor

Os usuários pertencentes ao grupo “advisor” possuirão acesso limitado aos recursos do banco de dados, sendo possível apenas permissão de leitura nas tabelas e com maior limitação na tabela de usuários, a qual não possuirá nenhum privilégio.

1.1.3 Login

Além das roles citadas acima, será criada uma role auxiliar para melhorar a segurança da aplicação. A role “login” será a role de acesso inicial ao sistema e só possuirá acesso a função de login e consequentemente a leitura da tabela de usuários.

2. Requisitos funcionais

Abaixo estão listados os requisitos funcionais que determinarão as principais funções que serão realizadas pelo sistema em questão de armazenamento de dados.

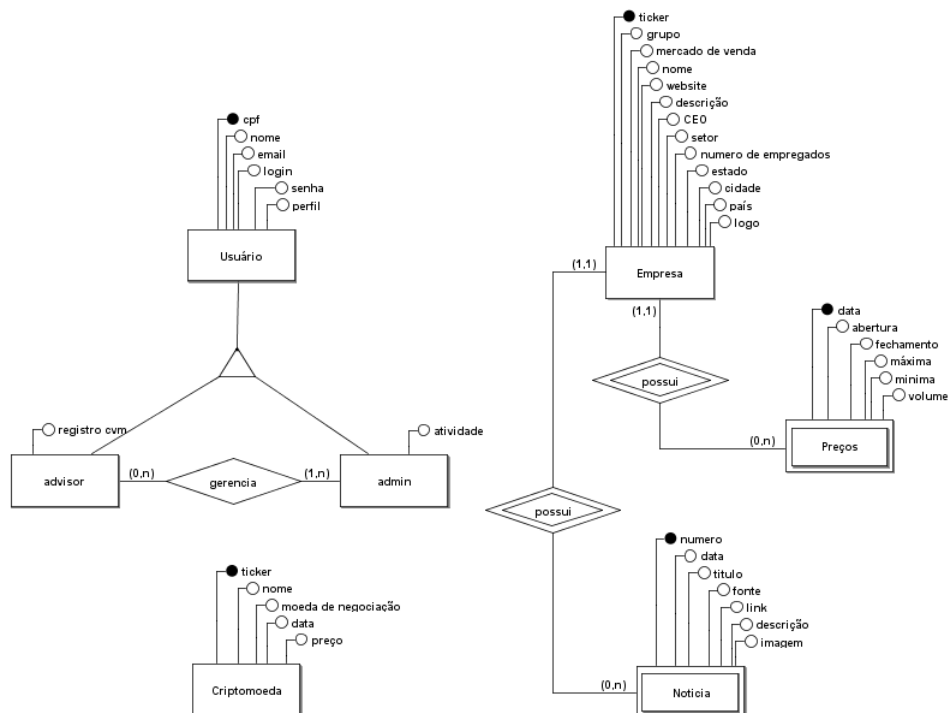
Número	Descrição	Prioridade
RF1	<p>O s.d.p. manter informações sobre os usuários, contendo os seguintes dados: cpf, nome, email, login, senha, tipo de usuário. Existem dois tipos de usuários no sistema: admin ou advisor.</p> <p>Os usuários do tipo advisor, além das informações de usuário, possuirão também um registro cvm, atestando sua atuação como consultor de investimentos.</p> <p>Cada usuário registrado no sistema só poderá ser consultor ou administrador, não existirão usuários pertencentes a ambos os grupos.</p> <p>Um admin pode criar vários advisors e um advisor sempre será criado por um admin.</p>	(E)
RF2	<p>O s.d.p. manter informações sobre as empresas, contendo os seguintes dados: ticker, nome, mercado de venda, grupo, website, descrição de atividade, CEO, setor, número de empregados, estado, cidade, país e logo da empresa</p> <p>Cada empresa possuirá nenhuma ou várias cotações de preço.</p> <p>Se uma determinada empresa for excluída, todos os</p>	(E)



	<p>preços associados a ela também serão.</p> <p>Cada empresa possuirá nenhuma ou várias notícias associadas a ela.</p> <p>Se uma determinada empresa for excluída, todas as notícias associadas a ela também serão.</p>	
RF3	<p>O s.d.p. manter informações sobre os preços, contendo os seguintes dados: data referente, valor de abertura, valor de fechamento, valor máximo, valor mínimo e volume de trades.</p> <p>Os preços poderão ser identificados a partir da associação do ticker (símbolo) da empresa com a data referente ao preço.</p> <p>Cada registro de preços deve necessariamente estar associado a uma e somente uma empresa.</p>	(E)
RF4	<p>O s.d.p. manter informações sobre as notícias, contendo os seguintes dados: data referente, título, fonte, link de acesso, descrição e imagem.</p> <p>Cada registro de notícia deve necessariamente estar associado a uma e somente uma empresa.</p>	(E)
RF5	<p>O s.d.p. manter informações sobre criptomoedas, contendo os seguintes dados: ticker (símbolo), nome, moeda de negociação, preço de fechamento e data de referência.</p>	(E)



3. Modelo Conceitual (Modelo Entidade Relacionamento - DER)



4. Mapeamento da API para o BD

Para a elaboração do modelo relacional do banco de dados foi coletada uma amostra de dados da API, sendo feito o levantamento dos tipos de dados e também do tamanho máximo dos campos, calculado através de um script que percorreu os dados, para evitar desperdício de memória. A partir disso tivemos os seguintes resultados:

Company

Column	Type	Max Length
symbol	VARCHAR	5
name	VARCHAR	37
exchange	VARCHAR	33
industry	VARCHAR	119
website	VARCHAR	33
description	TEXT	1505

CEO	VARCHAR	23
sector	VARCHAR	45
employees	INT	-
state *	VARCHAR	14
city *	VARCHAR	18
country *	VARCHAR	14
logo	VARCHAR	67

(*) pode não ter dados na API.

News

Column	Type	Max Lenght
symbol	VARCHAR	5
date	DATE	-
title	VARCHAR	264
source	VARCHAR	35
url	VARCHAR	86
description	TEXT	1900
image	VARCHAR	84

Crypto

Column	Type	Max Lenght
symbol	VARCHAR	13
name	VARCHAR	24
currency	VARCHAR	4
price	REAL	17

Esse mapeamento de tipos e tamanho máximo de caracteres resultou no schema sql listado abaixo:



```
CREATE SCHEMA IF NOT EXISTS stocker|
CREATE TYPE stocker.profile AS ENUM (
    'admin',
    'advisor',
);

CREATE TABLE IF NOT EXISTS stocker.user(
    cpf BIGINT NOT NULL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    username VARCHAR(20) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    email VARCHAR(50) NOT NULL,
    profile stocker.profile NOT NULL,
    cvm_license INT UNIQUE
);

CREATE TABLE IF NOT EXISTS stocker.company (
    symbol VARCHAR(5) NOT NULL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    exchange VARCHAR(40) NOT NULL,
    industry VARCHAR(130) NOT NULL,
    website VARCHAR(40),
    description TEXT,
    CEO VARCHAR(30) NOT NULL,
    sector VARCHAR(50) NOT NULL,
    employees INT,
    state VARCHAR(20),
    city VARCHAR(20),
    country VARCHAR(20),
    logo VARCHAR(80)
);
```



```
CREATE TABLE IF NOT EXISTS stocker.price(  
    symbol VARCHAR(5) NOT NULL,  
    date DATE NOT NULL,  
    close REAL NOT NULL,  
    high REAL NOT NULL,  
    low REAL NOT NULL,  
    open REAL NOT NULL,  
    volume INT NOT NULL,  
    FOREIGN KEY(symbol) REFERENCES stocker.company(symbol) ON DELETE CASCADE,  
    PRIMARY KEY (symbol, date)  
);  
  
CREATE TABLE IF NOT EXISTS stocker.news(  
    id SERIAL PRIMARY KEY,  
    symbol VARCHAR(10) NOT NULL,  
    date DATE NOT NULL,  
    title VARCHAR(200) NOT NULL,  
    source VARCHAR(100),  
    url VARCHAR(100),  
    description TEXT,  
    image VARCHAR(100),  
    FOREIGN KEY(symbol) REFERENCES stocker.company(symbol) ON DELETE CASCADE  
);  
  
CREATE TABLE IF NOT EXISTS stocker.crypto (  
    symbol VARCHAR(15) NOT NULL PRIMARY KEY,  
    name VARCHAR(25) NOT NULL,  
    currency VARCHAR(5) NOT NULL,  
    price REAL NOT NULL,  
    price_date DATE NOT NULL  
);
```

5. Definição de funções

Para a utilização inicial da plataforma foi definida uma função de login para verificar o usuário do cliente que deseja acessar a plataforma e definir seu profile no sistema. A utilização da função nesse caso foi feita para melhoria de segurança, podendo limitar o acesso aos recursos e também pelo fato da utilização de criptografia para salvar as senhas do usuário:



```
CREATE EXTENSION pgcrypto;

CREATE OR REPLACE FUNCTION login(user_input VARCHAR(20), pass_input VARCHAR(20)) RETURNS VARCHAR(20) AS $$
    SELECT u.profile FROM stocker.user u WHERE u.username = user_input AND u.password = crypt(pass_input, password)
$$ LANGUAGE SQL;
```

Além da função de login também foi criada uma função para a busca de preços por período (get_prices) para tentativa de melhorar performance, no entanto essa função não foi utilizada e o motivo será explicado nos próximos tópicos (7. Melhorias de performance e otimização).

```
CREATE OR REPLACE FUNCTION get_prices(symbol_input VARCHAR(200), period_input VARCHAR(10)) RETURNS SETOF stocker.price AS $$
BEGIN
    IF period_input = '1m' THEN
        RETURN QUERY
        SELECT * FROM stocker.price p WHERE p.symbol = ANY(string_to_array(symbol_input, ' ')) and p.date BETWEEN (CURRENT_DATE - interval '1 month') AND CURRENT_DATE;
    END IF;
    IF period_input = '6m' THEN
        RETURN QUERY
        SELECT * FROM stocker.price p WHERE p.symbol = ANY(string_to_array(symbol_input, ' ')) and p.date BETWEEN (CURRENT_DATE - interval '6 month') AND CURRENT_DATE;
    END IF;
    IF period_input = '1y' THEN
        RETURN QUERY
        SELECT * FROM stocker.price p WHERE p.symbol = ANY(string_to_array(symbol_input, ' ')) and p.date BETWEEN (CURRENT_DATE - interval '1 year') AND CURRENT_DATE;
    END IF;
    IF period_input = '2y' THEN
        RETURN QUERY
        SELECT * FROM stocker.price p WHERE p.symbol = ANY(string_to_array(symbol_input, ' ')) and p.date BETWEEN (CURRENT_DATE - interval '2 year') AND CURRENT_DATE;
    END IF;
    IF period_input = '5y' THEN
        RETURN QUERY
        SELECT * FROM stocker.price p WHERE p.symbol = ANY(string_to_array(symbol_input, ' ')) and p.date BETWEEN (CURRENT_DATE - interval '5 year') AND CURRENT_DATE;
    END IF;
    IF period_input = 'max' THEN
        RETURN QUERY
        SELECT * FROM stocker.price p WHERE p.symbol = ANY(string_to_array(symbol_input, ' '));
    END IF;
END;
$$ LANGUAGE 'plpgsql'
```

6. Melhorias de segurança

Para demonstração de controle de acesso fizemos a divisão de acesso do sistema em dois tipos de usuários: admin e advisor. Além disso, temos uma role específica para a verificação do login no sistema (role: login).

- A role admin possui todos os privilégios do schema.
- A role advisor não possui permissão na tabela user, e possui apenas permissão de leitura nas demais.
- A role login só possui acesso de execução da function login e consequentemente de leitura da tabela user

O admin pode registrar advisors na tabela user, esses advisors possuirão uma senha de acesso que é criptografada com salt antes de ser registrada, utilizando a extensão pgcrypto. A query para esse procedimento pode ser encontrada em `utils/db_query.py` -> `insert_user_query`, e a função de login que utiliza essa extensão foi listada acima.



A partir dessas definições tivemos o código resultante:

```
CREATE ROLE login WITH LOGIN PASSWORD 'stocker_login';
GRANT USAGE ON SCHEMA stocker TO login;
GRANT EXECUTE ON FUNCTION login TO login;
GRANT SELECT ON TABLE stocker.user TO login;

CREATE ROLE advisor WITH LOGIN PASSWORD 'stocker_advisor';
GRANT USAGE ON SCHEMA stocker TO advisor;
GRANT SELECT ON TABLE stocker.company TO advisor;
GRANT SELECT ON TABLE stocker.crypto TO advisor;
GRANT SELECT ON TABLE stocker.news TO advisor;
GRANT SELECT ON TABLE stocker.price TO advisor;
```

7. Melhorias de performance e otimização

Além da otimização na construção do schema, foram analisados os tempos de execução de algumas queries, buscando possibilidades de melhoria no desempenho.

- Consulta de preços por período

O primeiro caso foi a query de preços, como essa query possui como entrada um intervalo de duas datas, surgiu a dúvida se o desempenho seria melhor a partir do processamento das datas no postgresql ou no python. Assim, foi montada a function `get_prices` (stocker.sql) e comparada com a query `price_series_query2` (utils/db_query.py)

14

EXPLAIN ANALYZE SELECT * FROM stocker.price p WHERE p.symbol = ANY(string_to_array(('AAPL TSLA'), ' ')) and p.date BETWEEN '2019-06-29' AND '2021-06-29';

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Bitmap Heap Scan on price p (cost=29.38..1059.32 rows=984 width=28) (actual time=0.111..0.203 rows=1000 loops=1)

2

Recheck Cond: (((symbol)::text = ANY (({AAPL,TSLA})::text[])) AND (date >= '2019-06-29'::date) AND (date <= '2021-06-29'::date))

3

Heap Blocks: exact=9

4

-> Bitmap Index Scan on price_pkey (cost=0.00..29.14 rows=984 width=0) (actual time=0.103..0.103 rows=1000 loops=1)

5

Index Cond: (((symbol)::text = ANY (({AAPL,TSLA})::text[])) AND (date >= '2019-06-29'::date) AND (date <= '2021-06-29'::date))

6

Planning Time: 0.086 ms

7

Execution Time: 0.263 ms

16

EXPLAIN ANALYZE SELECT * FROM get_prices(('AAPL AMZN'), '2y')

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Function Scan on get_prices (cost=0.25..10.25 rows=1000 width=48) (actual time=1.139..1.195 rows=1006 loops=1)

2

Planning Time: 0.028 ms

3

Execution Time: 1.250 ms

Como podemos ver pelo resultado, o processamento das datas pelo python trouxe um resultado mais satisfatório (0.263 ms / 1.250 ms), se tornando a abordagem utilizada.

- Consulta de cryptos por nome

Para a otimização desse cenário foi pensado a criação de um índice na coluna name da tabela crypto, para comprovar a eficácia dessa hipótese as análises foram feitas:

3

EXPLAIN ANALYZE SELECT * FROM stocker.crypto c WHERE c.name like '%BTC%'

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Seq Scan on crypto c (cost=0.00..32.14 rows=418 width=31) (actual time=0.034..0.570 rows=355 loops=1)

2

Filter: ((name)::text ~~ '%BTC%':text)

3

Rows Removed by Filter: 1176

4

Planning Time: 0.101 ms

5

Execution Time: 0.595 ms

20

CREATE extension pg_trgm;

21

CREATE INDEX name_crypto_idx ON stocker.crypto USING gin (name gin_trgm_ops);

22

EXPLAIN ANALYZE SELECT * FROM stocker.crypto c WHERE c.name like '%BTC%'

Data Output

Explain

Messages

Notifications

QUERY PLAN

text

1

Bitmap Heap Scan on crypto c (cost=11.24..29.46 rows=418 width=31) (actual time=0.036..0.103 rows=355 loops=1)

2

Recheck Cond: ((name)::text ~~ '%BTC%':text)

3

Heap Blocks: exact=12

4

-> Bitmap Index Scan on name_crypto_idx (cost=0.00..11.13 rows=418 width=0) (actual time=0.028..0.028 rows=355 loops=1)

5

Index Cond: ((name)::text ~~ '%BTC%':text)

6

Planning Time: 0.083 ms

7

Execution Time: 0.131 ms

Na criação do índice também foi utilizada a extensão [pg_trgm](#) que torna o processo de reconhecimento de cadeia de caracteres mais rápido, a partir dos resultados (0.595 ms / 0.131 ms), o método de indexação com o pg_trgm foi utilizado.

- Validação de login

Por fim, foi feita a tentativa de criação de um índice na tabela user sobre as colunas user e password com o objetivo de melhorar a velocidade de validação de usuário. Os testes não justificaram a implementação do índice já que os resultados foram muito similares e um indexador múltiplo com pouca diferença de performance não justificaria o uso extra de memória.

8. Contagem das tabelas após a carga de dados

Obs: a tabela user foi criada para demonstração de roles e seus dados não são resultados da carga da API.




company: 101 registros

price: 125.470 registros

news: 1.005 registros

crypto: 1.531 registros

Para a contagem de rows foi executada uma [function de row count](#).

	 table_schema name	 table_name name	 count_rows_of_table integer
1	stocker	price	125470
2	stocker	crypto	1531
3	stocker	news	1005
4	stocker	company	101
5	stocker	user	5