



Universidade Federal do Mato Grosso
Instituto de Computação

Livraria

Shoten

Trabalho Laboratório de Banco de Dados:

Gustavo Eiji Kida

Hiago de Sousa Patrício

Wilson Andrade Pereira de Souza

SUMARIO :

1. Introdução
2. Regras de Negocio
3. UML
4. Triggers
5. Relatorio
6. Otimização
7. Log/Backup
8. Cloud
9. DataLake
10. DataOps
11. SQL Tables
12. Crescimento de Dados
13. Versionamento

1. Introdução

A livraria Shoten é uma pequena empresa, possuindo 3 tipos de produtos principais: livros, mangás e revistas. Para realizar compras na loja, é necessário que o cliente faça um cadastro, caso não tenha se cadastrado antes. Com o cadastro, o cliente pode pagar mais barato virando vip na 100ª compra, além de outras promoções.

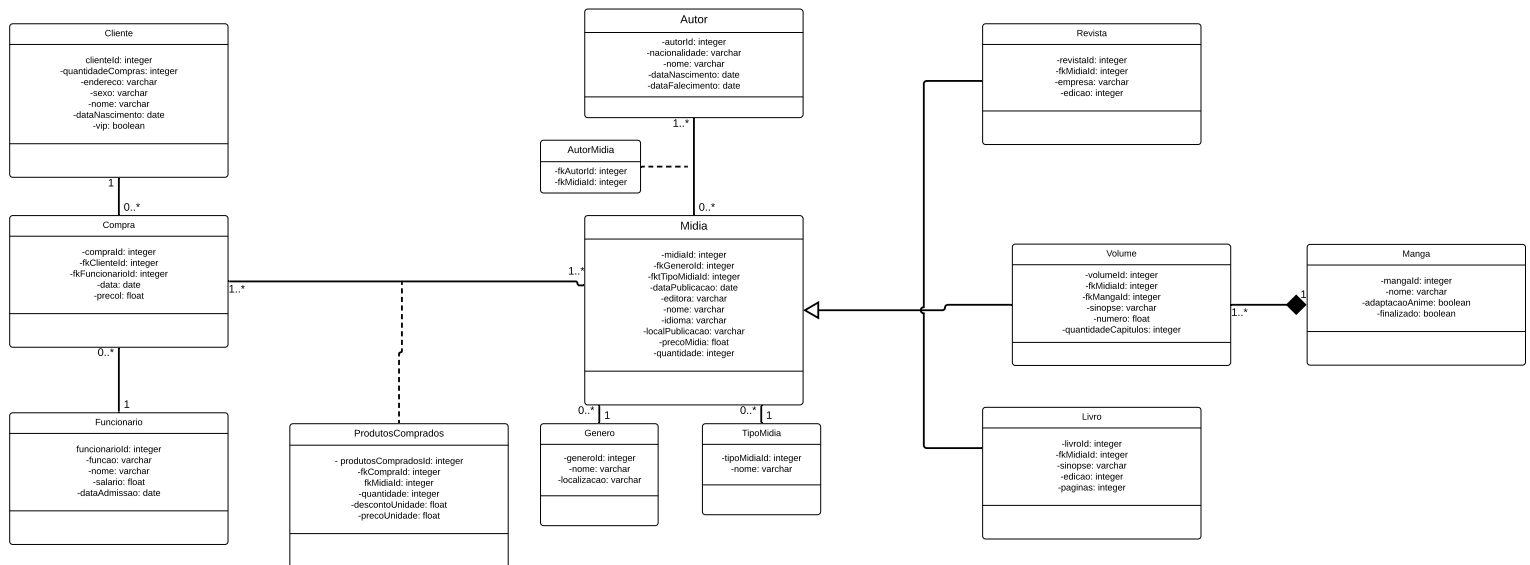
Os funcionários também são obrigados a realizar o cadastramento antes de começar a trabalhar, para ter as compras associadas ao vendedor e assim registrar suas vendas. Os autores dos produtos também são cadastrados, facilitando a oferta de obras desconhecidas do autor e facilitando a busca de produtos em geral. A livraria também possui um sistema de desconto de maneira a incentivar a compra e a leitura, com desconto em mangás completos, compras em grande quantidade e frequência de compras.

2. Regras de negócio

- 1 - Cliente pode ser cadastrado sem precisar comprar;
- 2 - Cliente pode fazer quantas compras quiser ou nenhuma;
- 3 - Funcionário pode ser cadastrado sem precisar participar de compra;
- 4 - Funcionário pode participar quantas compras quiser ou nenhuma;
- 5 - Uma compra deve ter 1 cliente e 1 funcionário participando;
- 6 - Uma compra deve possuir ao menos 1 mídia;
- 7 - A quantidade mínima de mídias na compra é 1;
- 8 - O preço de uma mídia é mutável;
- 9 - Uma mídia é um livro ou revista ou volume;
- 10 - Uma mídia deve pertencer a somente 1 gênero;
- 11 - Um gênero pode ser cadastro sem possuir mídia;
- 12 - Ao cadastrar um livro, revista ou volume deve-se associar a somente uma mídia;
- 13 - Um mangá é cadastrado sem precisar possuir volume;
- 14 - Um volume ao ser cadastrado deve se vincular a somente um mangá;
- 15 - Uma mídia deve possuir 1 ou mais autores;
- 16 - Um autor pode ser cadastrado sem ter feito mídia;
- 17 - Um autor pode se vincular a nenhuma ou quantas quiser;
- 18 - Na 100ª compra o cliente vira vip;
- 19 - Cliente vip tem 10% de desconto no valor inicial da compra;
- 20 - Cliente recebe 10% de desconto no valor da mídia a partir de 5 unidades dela;
- 21 - A cada 10 compras o cliente ganha 25% de desconto;

22 - Se o mangá estiver finalizado o cliente recebe 20% de desconto na compra de todos os volumes do mangá.

3. UML



4. Triggers

Foram utilizadas nove triggers, no desenvolvimento do SQL.

a_valor_atual_produto_comprado_TR: Copia o valor atual da mídia para a tabela produto comprado;

b_desconto_de_10_porcento_a_partir_de_5_unidades_TG: O cliente recebe 10% de desconto no valor inicial do produto na compra a partir de 5 unidades;

c_desconto_10_para_vip_TG: Cliente vip possui 10% de desconto no valor inicial da compra;

d_desconto_25_a_cada_10_compras_TG: A cada 10 compras o cliente ganha 25% de desconto;

e_desconto_de_20_porcento_na_colecao_completa_de_manga_TG: Se o mangá estiver finalizado, o cliente recebe 20% de desconto na compra de todos os volumes do mangá;

f_atualiza_compra_TR: Atualiza o valor da compra;

g_atualiza_quantidade_de_compras_TG: Atualiza a quantidade de compras;
h_cliente_vira_vip_depois_de_100_compras_TG: O cliente vira vip depois de 100 compras;

limita_uso_da_chave_da_midia_livro_TG, limita_uso_da_chave_da_midia_revista_TG, limita_uso_da_chave_da_midia_volume_TG: Verifica se uma mídia só está sendo referenciada por uma revista, livro ou volume.

-- Copiar o valor atual da midia para o produto comprado

-- Primeiro

CREATE OR REPLACE FUNCTION valor_atual_produto_comprado_FUNC()

RETURNS TRIGGER AS \$BODY\$

DECLARE

preco_midia FLOAT;

BEGIN

SELECT M.precoMidia

INTO preco_midia

FROM Midia M

WHERE NEW.fkMidiaId = M.midiaId;

NEW.precoUnidade = preco_midia;

RETURN NEW;

END;

\$BODY\$ LANGUAGE plpgsql;

CREATE TRIGGER a_valor_atual_produto_comprado_TR

BEFORE INSERT ON ProdutosComprados

FOR EACH ROW

EXECUTE PROCEDURE valor_atual_produto_comprado_FUNC();

-- Cliente recebe 10% de desconto no valor inicial do produto na compra a partir de 5 unidades

-- Segundo

CREATE OR REPLACE FUNCTION

desconto_de_10_porcento_a_partir_de_5_unidades_FUNC()

RETURNS TRIGGER AS \$BODY\$

BEGIN

IF NEW.quantidade >= 5 THEN

NEW.descontoUnidade = NEW.descontoUnidade + NEW.precoUnidade * 0.1;

END IF;

RETURN NEW;

```

END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER b_desconto_de_10_porcento_a_partir_de_5_unidades_TG
BEFORE INSERT ON ProdutosComprados
FOR EACH ROW
EXECUTE
desconto_de_10_porcento_a_partir_de_5_unidades_FUNC();
-- Cliente vip tem 10% de desconto no valor inicial da compra
-- Terceiro
CREATE OR REPLACE FUNCTION desconto_10_para_vip_FUNC()
RETURNS TRIGGER AS $BODY$
DECLARE
    bool_vip BOOLEAN;
BEGIN
    SELECT Cl.vip
    INTO bool_vip
    FROM Cliente Cl
    JOIN Compra C
    ON C.fkClienteId = Cl.clienteId
    WHERE NEW.fkCompraId = C.compraId;

    IF bool_vip = TRUE THEN
        NEW.descontoUnidade = NEW.descontoUnidade + NEW.precoUnidade * 0.1;
    END IF;
    RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER c_desconto_10_para_vip_TG
BEFORE INSERT ON ProdutosComprados
FOR EACH ROW
EXECUTE PROCEDURE desconto_10_para_vip_FUNC();

-- A cada 10 compras o cliente ganha 25% de desconto
-- Quarto
CREATE OR REPLACE FUNCTION desconto_25_a_cada_10_compras_FUNC()
RETURNS TRIGGER AS $BODY$
DECLARE
    quantidade_compras INTEGER;
BEGIN

```

```

SELECT Cl.quantidadeCompras
INTO quantidade_compras
FROM Cliente Cl
JOIN Compra C
ON Cl.clienteId = C.fkClienteId
WHERE NEW.fkCompraId = C.compraId;

```

```

    IF quantidade_compras > 0 AND quantidade_compras % 10 = 0 THEN
        NEW.descontoUnidade = NEW.descontoUnidade + NEW.precoUnidade * 0.25;
    END IF;
    RETURN NEW;

```

```

END;
$BODY$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER d_desconto_25_a_cada_10_compras_TG
BEFORE INSERT ON ProdutosComprados
FOR EACH ROW
EXECUTE PROCEDURE desconto_25_a_cada_10_compras_FUNC();

```

-- Se o mangá estiver finalizado o cliente recebe 20% de desconto na compra de todos os volumes do mangá

-- Quinto

```

CREATE OR REPLACE FUNCTION
desconto_de_20_porcento_na_colecao_completa_de_manga_FUNC()
RETURNS TRIGGER AS $BODY$
DECLARE

```

```

    finalizado BOOLEAN;

```

```

    mangaId INTEGER;

```

```

    quantidade_volumes_comprados INTEGER;

```

```

    quantidade_volumes_existentes INTEGER;

```

```

BEGIN

```

```

    SELECT M.finalizado

```

```

    INTO finalizado

```

```

    FROM Manga M

```

```

    JOIN Volume V

```

```

    ON V.fkMangaId = M.mangaId

```

```

    WHERE NEW.fkMidiaId = V.fkMidiaId;

```

```

    IF finalizado = TRUE THEN

```

```

        SELECT M.mangaId

```

```

        INTO mangaId

```

```

        FROM Manga M

```

```
JOIN Volume V
ON V.fkMangaId = M.mangaId
WHERE NEW.fkMidiaId = V.fkMidiaId;
```

```
SELECT COUNT (DISTINCT V.volumeId)
INTO quantidade_volumes_existentes
FROM Volume V
WHERE V.fkMangaId = mangaId;
```

```
SELECT COUNT (DISTINCT PC.produtosCompradosId)
INTO quantidade_volumes_comprados
FROM ProdutosComprados PC
JOIN Midia M
ON PC.fkMidiaId = M.midiaID
JOIN TipoMidia TP
ON M.fkTipoMidiaId = TP.tipoMidiaId
WHERE
    NEW.fkCompraId = PC.fkCompraId
    AND TP.nome = 'Volume';
```

```
                IF  quantidade_volumes_comprados  =  quantidade_volumes_existentes
THEN
    UPDATE ProdutosComprados PC
    SET descontoUnidade = descontoUnidade + precoUnidade * 0.2
    FROM Volume V
    WHERE
        -- Selecciona a compra
        NEW.fkCompraId = PC.fkCompraId AND
        -- Selecciona os volumes comprados
        PC.fkMidiaId = V.fkMidiaId;
        END IF;
    END IF;
    RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;
```

```
CREATE
e_desconto_de_20_porcento_na_colecao_completa_de_manga_TG
AFTER INSERT ON ProdutosComprados
FOR EACH ROW
```

TRIGGER


```

EXECUTE
desconto_de_20_porcento_na_colecao_completa_de_manga_FUNC());

-- Atualiza valor da compra
-- Sexto
CREATE OR REPLACE FUNCTION atualiza_compra_FUNC()
RETURNS TRIGGER AS $BODY$
DECLARE
    valor_final_produtos FLOAT;
BEGIN
    valor_final_produtos = (NEW.precoUnidade - NEW.descontoUnidade) *
NEW.quantidade;

    UPDATE Compra C
    SET preco = preco + valor_final_produtos
    WHERE C.compraId = NEW.fkCompraId;

    RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER f_atualiza_compra_TR
AFTER INSERT ON ProdutosComprados
FOR EACH ROW
EXECUTE PROCEDURE atualiza_compra_FUNC());

-- Atualiza quantidade de compras
-- Sétimo
CREATE OR REPLACE FUNCTION atualiza_quantidade_de_compras_FUNC()
RETURNS TRIGGER AS $BODY$
BEGIN
    UPDATE Cliente Cl
    SET quantidadeCompras = quantidadeCompras + 1
    WHERE NEW.fkClienteId = Cl.clienteId;

    RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER g_atualiza_quantidade_de_compras_TG
AFTER INSERT ON Compra
FOR EACH ROW

```

```
EXECUTE PROCEDURE atualiza_quantidade_de_compras_FUNC();
```

```
-- Depois de 100 compras o cliente vira vip
```

```
-- Oitavo
```

```
CREATE OR REPLACE FUNCTION
```

```
cliente_vira_vip_depois_de_100_compras_FUNC()
```

```
RETURNS TRIGGER AS $BODY$
```

```
DECLARE
```

```
    quantidade_compras INTEGER;
```

```
BEGIN
```

```
    SELECT Cl.quantidadeCompras
```

```
    INTO quantidade_compras
```

```
    FROM Cliente Cl
```

```
    WHERE Cl.clienteId = NEW.fkClienteId;
```

```
    IF quantidade_compras = 100 THEN
```

```
        UPDATE Cliente Cl
```

```
        SET vip = TRUE
```

```
        WHERE Cl.clienteId = NEW.fkClienteId;
```

```
    END IF;
```

```
    RETURN NEW;
```

```
END;
```

```
$BODY$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER h_cliente_vira_vip_depois_de_100_compras_TG
```

```
AFTER INSERT ON Compra
```

```
FOR EACH ROW
```

```
EXECUTE PROCEDURE cliente_vira_vip_depois_de_100_compras_FUNC();
```

```
-- Uma mídia só pode ser referenciada por uma revista, livro ou volume
```

```
--nono
```

```
CREATE OR REPLACE FUNCTION limita_uso_da_chave_da_midia_FUNC()
```

```
RETURNS TRIGGER AS $BODY$
```

```
DECLARE
```

```
    quantidadeUsadaFkMidiaId INTEGER;
```

```
BEGIN
```

```
    quantidadeUsadaFkMidiaId = 0;
```

```
    SELECT COUNT(*)
```

```
    INTO quantidadeUsadaFkMidiaId
```

```
    FROM livro l
```

```
    FULL JOIN revista r
```

```

ON l.fkmidiaid = r.fkmidiaid
FULL JOIN volume v
ON v.fkmidiaid = r.fkmidiaid
WHERE l.fkmidiaid = NEW.fkmidiaid;

IF(quantidadeUsadaFkMidiaId != 0) THEN
    RAISE EXCEPTION 'Chave já usada em outro produto';
END IF;
RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;

CREATE TRIGGER limita_uso_da_chave_da_midia_livro_TG
BEFORE INSERT ON livro
FOR EACH ROW
EXECUTE PROCEDURE limita_uso_da_chave_da_midia_FUNC();

CREATE TRIGGER limita_uso_da_chave_da_midia_revista_TG
BEFORE INSERT ON revista
FOR EACH ROW
EXECUTE PROCEDURE limita_uso_da_chave_da_midia_FUNC();

CREATE TRIGGER limita_uso_da_chave_da_midia_volume_TG
BEFORE INSERT ON volume
FOR EACH ROW
EXECUTE PROCEDURE limita_uso_da_chave_da_midia_FUNC();

```

5. Relatório

Foram feitos dois relatórios utilizando VIEW, o relatorio_view mostra a informação da compra feito pelo cliente, mostrando o nome do cliente, a data da compra, o funcionário que efetuou a compra e informações gerais dos produtos.

Já no relatorio2_view mostra os produtos vendidos organizado por autor para verificar qual o autor que mais vendeu, e além disso mostra o funcionário que efetuou essa venda.

---Relatorio dos produtos comprados pelos clientes

```

CREATE VIEW relatorio_view AS
SELECT c.nome as nomeCliente, co.data, co.preco, f.nome as nomeFuncionario,
m.nome as nomeMidia,
g.nome as nomeGenero, tm.nome as nomeTipo, pc.precoUnidade, pc.quantidade
FROM CLiente c
LEFT OUTER JOIN Compra co ON c.clienteId = co.fkClienteId
LEFT OUTER JOIN Funcionario f ON co.fkFuncionarioId = f.funcionarioId
LEFT OUTER JOIN ProdutosComprados pc ON co.compraId = pc.fkCompraId
LEFT OUTER JOIN Midia m ON pc.fkMidiaId = m.midiaId
LEFT OUTER JOIN TipoMidia tm ON m.fkTipoMidiaId = tm.tipoMidiaId
LEFT OUTER JOIN Genero g ON m.fkGeneroId = g.generoId
WHERE c.vip = TRUE
AND co.data > '2000-1-1'
AND co.preco > 100
ORDER BY nomeCLiente;

```

---Relatorio de autores que tiveram suas midias vendidas e por quais funcionarios

```

CREATE VIEW relatorio2_view AS
SELECT a.nome as nomeAutor, m.nome as nomeMidia, tm.nome as nomeTipo, g.nome
as nomeGenero,
f.nome as nomeFuncionario, co.data, pc.quantidade
FROM Autor a
LEFT OUTER JOIN AutorMidia am ON a.autorId = am.fkAutorId
LEFT OUTER JOIN Midia m ON am.fkMidiaId = m.MidiaId
LEFT OUTER JOIN Genero g ON m.fkGeneroId = g.generoId
LEFT OUTER JOIN TipoMidia tm ON m.fkTipoMidiaId = tm.tipoMidiaId
LEFT OUTER JOIN ProdutosComprados pc ON m.midiaId = pc.fkMidiaId
LEFT OUTER JOIN Compra co ON pc.fkCompraId = co.compraId
LEFT OUTER JOIN Funcionario f ON co.fkFuncionarioId = f.funcionarioId
WHERE pc.quantidade > 25
AND co.data > '2000-1-1'
ORDER BY nomeAutor;

```

Foi criado os relatórios utilizando a ferramenta jasper, e gerado um pdf que está na pasta “jasperreport-to-pdf” com um script java utilizando o Netbeans.

1 de 80

relatorio.pdf

79,8%

Vendas							
Compras feitas pelos clientes organizadas pelo nome do cliente							
data	preco	nomefuncio	nomemidia	nomegener	nometipo	precounida	quantidade
nome cliente		Aaron Williams					
02/05/09 00:	13799.552	Antonio Beck	Jessica Ward	Heather	Andrew	99.42	25
30/10/18 00:	7390.704	Cesar Smith	Laura Cruz	Michael	Jesse Duncan	69.98	66
30/10/18 00:	7390.704	Cesar Smith	Ann Butler	Brian Carr	Jesse Duncan	85.55	54
29/12/03 00:	12123.84	Jesse	Katrina	Jacob Charles	Jesse Duncan	48.85	72
28/12/01 00:	2357.12	Antonio Beck	Elizabeth	Tammy	Laura Acosta	36.83	80
19/02/17 00:	2676.421	Lisa	Jason Silva	Tina Banks	Laura Acosta	35.97	43
19/02/17 00:	2676.421	Lisa	Rachel Flynn	Richard	Jesse Duncan	75.25	1
19/02/17 00:	2676.421	Lisa	Heather Hart	Michael	Andrew	61.22	28
19/04/12 00:	4155.136	Stacy Ortiz	Anthony	Paula Powers	Jesse Duncan	58.67	38
19/04/12 00:	4155.136	Stacy Ortiz	Kelly Dickson	Michael	Andrew	76.26	11
19/04/12 00:	4155.136	Stacy Ortiz	Marc Glover	Tammy	Laura Acosta	53.14	40
29/12/03 00:	12123.84	Jesse	Brittany	Patricia	Laura Acosta	79.5	95
11/09/06 00:	3752.688	Pamela	Sarah Torres	Amanda	Laura Acosta	63.39	74
30/08/12 00:	6316.868	William	Elizabeth	Brian Carr	Jesse Duncan	98.33	4
31/10/09 00:	2583.008000	Pamela	Adam	Eric Brown	Laura Acosta	60.92	53
02/05/09 00:	13799.552	Antonio Beck	Michael	Tammy	Laura Acosta	39.52	57
30/08/12 00:	6316.868	William	Paul Ortega	Eric Brown	Laura Acosta	93.17	80
02/05/09 00:	13799.552	Antonio Beck	Anthony	Paula Powers	Jesse Duncan	58.67	77
02/05/09 00:	13799.552	Antonio Beck	Brittany	Patricia	Laura Acosta	79.5	95
19/08/07 00:	3384.688	Justin Ramos	Sheila Wong	Heather	Laura Acosta	62.28	18
12/05/09 00:	5313.072	Justin Ramos	Alexis Barnes	Tammy	Jesse Duncan	93.54	71
02/05/05 00:	1049.28	Stacy Ortiz	Steven	Heather	Andrew	43.72	30
03/06/01 00:	2234.936	Antonio Beck	Danielle	Paula Powers	Andrew	94.51	25
03/06/01 00:	2234.936	Antonio Beck	Tracy	Patricia	Andrew	71.82	6
11/07/02 00:	8873.104	Joshua Parker	Susan Harper	Amanda	Jesse Duncan	80.1	42
11/07/02 00:	8873.104	Joshua Parker	Tracy	Patricia	Andrew	71.82	38
20/12/02 00:	1075.712	Daniel	Tina Kerr	Jessica Finley	Laura Acosta	21.01	64
11/07/02 00:	8873.104	Joshua Parker	Brent Davis	Kevin Smith	Andrew	55.79	58
11/07/02 00:	8873.104	Joshua Parker	Tyler Martin	Heather	Jesse Duncan	32.04	55
13/09/06 00:	4953.879999	Daniel	Pam Smith	Manuel Gray	Jesse Duncan	51.46	4
19/12/16 00:	11163.472	Daniel	Carlos	Michael	Jesse Duncan	78.51	63

5

sexta-feira 10

Page 1 of 80

1

de 416

relatorio2.pdf

79,8%

1

2

3

4

5

Vendas por Autor

Analisar os autores que mais vendem

nomemidia	nometipo	nomegenero	nomefuncionari	data	quantidade
nome autor	Adam Good				
Anthony Williams	Jesse Duncan	Joanne Williams	Edward McIntyre	09/11/16 00:00	63
Christopher	Laura Acosta	Michael Mullen	Cesar Smith	28/08/10 00:00	43
Jeffery White	Jesse Duncan	Heather Rangel	Daniel Galvan	19/01/01 00:00	53
Anthony Williams	Jesse Duncan	Joanne Williams	Justin Ramos	16/12/18 00:00	36
Mary Acosta	Andrew Harvey	Jessica Finley	Antonio Beck	07/02/12 00:00	49
Christopher	Laura Acosta	Michael Mullen	Daniel Galvan	10/01/13 00:00	64
Jeffery White	Jesse Duncan	Heather Rangel	Jesse Stevenson	24/05/12 00:00	97
Anthony Williams	Jesse Duncan	Joanne Williams	Antonio Beck	06/03/18 00:00	52
Anthony Williams	Jesse Duncan	Joanne Williams	Daniel Galvan	15/02/03 00:00	93
Christopher	Laura Acosta	Michael Mullen	Stacy Ortiz	21/11/00 00:00	27
Christopher	Laura Acosta	Michael Mullen	Joshua Parker	01/06/09 00:00	36
Jeffery White	Jesse Duncan	Heather Rangel	Justin Ramos	01/10/02 00:00	40
Mary Acosta	Andrew Harvey	Jessica Finley	Antonio Beck	05/04/15 00:00	59
Mary Acosta	Andrew Harvey	Jessica Finley	Cesar Smith	05/01/11 00:00	86
Jeffery White	Jesse Duncan	Heather Rangel	Jesse Stevenson	23/06/08 00:00	78
Jeffery White	Jesse Duncan	Heather Rangel	Daniel Galvan	15/06/18 00:00	43
Mary Acosta	Andrew Harvey	Jessica Finley	Stacy Ortiz	10/08/00 00:00	32
Christopher	Laura Acosta	Michael Mullen	Joshua Parker	18/07/03 00:00	63
Christopher	Laura Acosta	Michael Mullen	William Sandoval	12/04/18 00:00	36
Anthony Williams	Jesse Duncan	Joanne Williams	Joan Floyd	21/07/00 00:00	55
Anthony Williams	Jesse Duncan	Joanne Williams	Joan Floyd	29/08/15 00:00	79
Jeffery White	Jesse Duncan	Heather Rangel	Stacy Ortiz	19/10/18 00:00	71
Mary Acosta	Andrew Harvey	Jessica Finley	Cesar Smith	30/04/14 00:00	95
Jeffery White	Jesse Duncan	Heather Rangel	Susan Shaffer	02/12/12 00:00	85
Mary Acosta	Andrew Harvey	Jessica Finley	Jesse Stevenson	29/05/05 00:00	33
Christopher	Laura Acosta	Michael Mullen	Pamela Garcia	02/01/09 00:00	75
Anthony Williams	Jesse Duncan	Joanne Williams	Jesse Stevenson	23/08/14 00:00	69
Mary Acosta	Andrew Harvey	Jessica Finley	Cesar Smith	23/08/15 00:00	50
Mary Acosta	Andrew Harvey	Jessica Finley	Edward McIntyre	21/09/15 00:00	93
Mary Acosta	Andrew Harvey	Jessica Finley	Joshua Parker	13/03/08 00:00	98
Jeffery White	Jesse Duncan	Heather Rangel	Cesar Smith	30/12/13 00:00	70

10

Page 1 of

416

6. Otimizações

Foram criados três índices para melhorar a velocidade de visualização dos relatórios. O índice `idx_compra_data` foi feito no campo “data” da tabela “compra”, pois todos os relatórios necessitam que esse campo seja verificado. Também foi criado o `idx_cliente_vip` feito no campo “vip” da tabela “cliente”, já que é utilizado no relatório 1 e em duas triggers. Além disso, o `idx_midia_nome`, feito no campo “nome” da tabela “mídia” para melhorar a busca de produtos pelo cliente.

```
CREATE INDEX idx_compra_data ON Compra (data);
CREATE INDEX idx_cliente_vip ON Cliente (vip);
CREATE INDEX idx_midia_nome ON Midia(nome);
ALTER INDEX idx_compra_data SET TABLESPACE tbs_indice;
ALTER INDEX idx_cliente_vip SET TABLESPACE tbs_indice;
ALTER INDEX idx_midia_nome SET TABLESPACE tbs_indice;
```

Foi criada duas tablespaces, a `tbs_livraria` armazena a tabela mídia pois é a tabela mais lida, por isso recomenda-se utilizar um disco que tenha uma boa velocidade de leitura. Já a `tbs_indice` armazena todos os índices e deve ser armazenado em um hd rápido ou um SSD para agilizar as pesquisas.

```
CREATE TABLESPACE tbs_livraria LOCATION '/livraria';
CREATE TABLESPACE tbs_indice LOCATION '/indice';
```

RELATORIO SEM INDEX

```
postgres=# explain analyze select * from relatorio_view;
                                         QUERY PLAN
-----
Sort (cost=242.92..242.92 rows=1 width=94) (actual time=1.659..1.659 rows=0 loops=1)
  Sort Key: c.nome
  Sort Method: quicksort  Memory: 25kB
    -> Nested Loop Left Join (cost=165.31..242.91 rows=1 width=94) (actual time=1.651..1.651 rows=0 loops=1)
      -> Nested Loop Left Join (cost=165.03..242.56 rows=1 width=84) (actual time=1.650..1.650 rows=0 loops=1)
        -> Nested Loop Left Join (cost=164.75..242.22 rows=1 width=74) (actual time=1.649..1.649 rows=0 loops=1)
          -> Nested Loop Left Join (cost=164.46..241.86 rows=1 width=56) (actual time=1.649..1.649 rows=0 loops=1)
            -> Nested Loop Left Join (cost=164.18..240.91 rows=1 width=44) (actual time=1.648..1.648 rows=0 loops=1)
              -> Hash Join (cost=163.90..239.89 rows=1 width=34) (actual time=1.648..1.648 rows=0 loops=1)
                Hash Cond: (co.fkclienteid = c.clienteid)
                -> Bitmap Heap Scan on compra co (cost=17.89..92.91 rows=367 width=24) (actual time=0.143..0.143 rows=1 loops=1)
                  Recheck Cond: (data > '2000-01-01'::date)
                  Filter: (preco > '100'::double precision)
                  Rows Removed by Filter: 2
                  Heap Blocks: exact=2
                  -> Bitmap Index Scan on idx_compra_cliente (cost=0.00..17.79 rows=735 width=0) (actual time=0.105..0.105 rows=729 loops=1)
                    Index Cond: (data > '2000-01-01'::date)
                -> Hash (cost=146.00..146.00 rows=1 width=18) (actual time=1.497..1.497 rows=0 loops=1)
                  Buckets: 1024  Batches: 1  Memory Usage: 8kB
                  -> Seq Scan on cliente c (cost=0.00..146.00 rows=1 width=18) (actual time=1.496..1.496 rows=0 loops=1)
                    Filter: vip
                    Rows Removed by Filter: 5000
              -> Index Scan using funcionario_pkey on funcionario f (cost=0.28..1.02 rows=1 width=18) (never executed)
                Index Cond: (co.fkfuncionarioid = funcionarioid)
            -> Index Scan using produtoscomprados_fkcompra_id_key on produtoscomprados pc (cost=0.28..0.93 rows=2 width=20) (never executed)
              Index Cond: (co.compra_id = fkcompra_id)
          -> Index Scan using midia_pkey on midia m (cost=0.28..0.37 rows=1 width=26) (never executed)
            Index Cond: (pc.fkmidia_id = midia_id)
        -> Index Scan using tipomidia_pkey on tipomidia tm (cost=0.28..0.34 rows=1 width=18) (never executed)
          Index Cond: (m.fktipomidia_id = tipomidia_id)
      -> Index Scan using genero_pkey on genero g (cost=0.28..0.34 rows=1 width=18) (never executed)
        Index Cond: (m.fkgenero_id = genero_id)
Planning Time: 2.754 ms
Execution Time: 1.796 ms
(34 rows)

postgres=#
```

RELATORIO COM INDEX EM VIP

```
Sort (cost=1092.62..1094.22 rows=638 width=78) (actual time=31.546..31.634 rows=1198 loops=1)
  Sort Key: a.nome
  Sort Method: quicksort Memory: 217kB
  -> Hash Left Join (cost=934.85..1062.90 rows=638 width=78) (actual time=25.830..28.926 rows=1198 loops=1)
    Hash Cond: (co.fkfuncionarioid = f.funcionarioid)
    -> Hash Join (cost=772.35..898.73 rows=638 width=68) (actual time=22.175..24.578 rows=1198 loops=1)
      Hash Cond: (a.autorid = am.fkautorid)
      -> Seq Scan on autor a (cost=0.00..95.00 rows=5000 width=18) (actual time=0.015..0.819 rows=5000 loops=1)
      -> Hash (cost=764.37..764.37 rows=638 width=58) (actual time=22.147..22.147 rows=1198 loops=1)
        Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 124kB
        -> Hash Right Join (cost=642.99..764.37 rows=638 width=58) (actual time=18.362..21.379 rows=1198 loops=1)
          Hash Cond: (g.generoid = m.fkgeneroid)
          -> Seq Scan on genero g (cost=0.00..90.00 rows=5000 width=18) (actual time=0.014..1.156 rows=5000 loops=1)
          -> Hash (cost=635.02..635.02 rows=638 width=48) (actual time=18.339..18.339 rows=1198 loops=1)
            Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 114kB
            -> Hash Right Join (cost=526.14..635.02 rows=638 width=48) (actual time=14.669..17.587 rows=1198 loops=1)
              Hash Cond: (tm.tipomidiaid = m.fktipomidiaid)
              -> Seq Scan on tipomidia tm (cost=0.00..78.33 rows=4833 width=18) (actual time=0.027..1.060 rows=4833 loops=1)
              -> Hash (cost=518.17..518.17 rows=638 width=38) (actual time=14.630..14.630 rows=1198 loops=1)
                Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 101kB
                -> Hash Join (cost=420.12..518.17 rows=638 width=38) (actual time=8.561..13.589 rows=1198 loops=1)
                  Hash Cond: (am.fkmidiaid = m.midiaid)
                  -> Seq Scan on autormidia am (cost=0.00..72.94 rows=4994 width=8) (actual time=0.014..1.659 rows=4994 loops=1)
                  -> Hash (cost=412.13..412.13 rows=639 width=42) (actual time=8.533..8.533 rows=568 loops=1)
                    Buckets: 1024 Batches: 1 Memory Usage: 51kB
                    -> Hash Join (cost=254.49..412.13 rows=639 width=42) (actual time=4.838..8.175 rows=568 loops=1)
                      Hash Cond: (m.midiaid = pc.fkmidiaid)
                      -> Seq Scan on midia m (cost=0.00..120.00 rows=5000 width=26) (actual time=0.008..1.406 rows=5000 loops=1)
                      -> Hash (cost=246.50..246.50 rows=639 width=16) (actual time=4.821..4.821 rows=568 loops=1)
                        Buckets: 1024 Batches: 1 Memory Usage: 35kB
                        -> Hash Join (cost=135.69..246.50 rows=639 width=16) (actual time=1.785..4.576 rows=568 loops=1)
                          Hash Cond: (pc.fkcompraid = co.compraid)
                          -> Seq Scan on produtoscomprados pc (cost=0.00..99.40 rows=4345 width=12) (actual time=0.014..1.577 rows=4326 loops=1)
                          Filter: (quantidade > 25)
                          Rows Removed by Filter: 666
                        -> Hash (cost=126.50..126.50 rows=735 width=12) (actual time=1.760..1.761 rows=729 loops=1)
                          Buckets: 1024 Batches: 1 Memory Usage: 40kB
                          -> Seq Scan on compra co (cost=0.00..126.50 rows=735 width=12) (actual time=0.011..1.452 rows=729 loops=1)
                          Filter: (data > '2000-01-01'::date)
                          Rows Removed by Filter: 4271
                    -> Hash (cost=100.00..100.00 rows=5000 width=18) (actual time=3.630..3.630 rows=5000 loops=1)
                      Buckets: 8192 Batches: 1 Memory Usage: 317kB
                      -> Seq Scan on funcionario f (cost=0.00..100.00 rows=5000 width=18) (actual time=0.013..1.654 rows=5000 loops=1)
Planning Time: 6.796 ms
Execution Time: 31.819 ms
(45 rows)
```

postgres=#

RELATORIO2 SEM INDEX

```
postgres=# explain analyze select * from relatorio_view;
                                QUERY PLAN
-----
Sort (cost=101.22..101.22 rows=1 width=94) (actual time=0.009..0.009 rows=0 loops=1)
  Sort Key: c.nome
  Sort Method: quicksort Memory: 25kB
  -> Nested Loop Left Join (cost=23.61..101.21 rows=1 width=94) (actual time=0.006..0.006 rows=0 loops=1)
    -> Nested Loop Left Join (cost=23.33..100.86 rows=1 width=84) (actual time=0.006..0.006 rows=0 loops=1)
      -> Nested Loop Left Join (cost=23.05..100.52 rows=1 width=74) (actual time=0.006..0.006 rows=0 loops=1)
        -> Nested Loop Left Join (cost=22.76..100.16 rows=1 width=56) (actual time=0.006..0.006 rows=0 loops=1)
          -> Nested Loop Left Join (cost=22.48..99.21 rows=1 width=44) (actual time=0.006..0.006 rows=0 loops=1)
            -> Hash Join (cost=22.20..98.19 rows=1 width=34) (actual time=0.006..0.006 rows=0 loops=1)
              Hash Cond: (co.fkclienteid = c.clienteid)
              -> Bitmap Heap Scan on compra co (cost=17.89..92.91 rows=367 width=24) (never executed)
                Recheck Cond: (data > '2000-01-01'::date)
                Filter: (preco > '100'::double precision)
                -> Bitmap Index Scan on idx_compra_cliente (cost=0.00..17.79 rows=735 width=0) (never executed)
                  Index Cond: (data > '2000-01-01'::date)
              -> Hash (cost=4.30..4.30 rows=1 width=18) (actual time=0.004..0.004 rows=0 loops=1)
                Buckets: 1024 Batches: 1 Memory Usage: 8kB
                -> Index Scan using idx_cliente on cliente c (cost=0.28..4.30 rows=1 width=18) (actual time=0.004..0.004 rows=0 loops=1)
                  Index Cond: (vlp = true)
                  Filter: vlp
            -> Index Scan using funcionario_pkey on funcionario f (cost=0.28..1.02 rows=1 width=18) (never executed)
              Index Cond: (co.fkfuncionarioid = funcionarioid)
          -> Index Scan using produtoscomprados_fkcompraid_fkmidiaid_key on produtoscomprados pc (cost=0.28..0.93 rows=2 width=20) (never executed)
            Index Cond: (co.compraid = fkcompraid)
        -> Index Scan using midia_pkey on midia m (cost=0.28..0.37 rows=1 width=26) (never executed)
          Index Cond: (pc.fkmidiaid = midiaid)
      -> Index Scan using tipomidia_pkey on tipomidia tm (cost=0.28..0.34 rows=1 width=18) (never executed)
        Index Cond: (m.fktipomidiaid = tipomidiaid)
    -> Index Scan using genero_pkey on genero g (cost=0.28..0.34 rows=1 width=18) (never executed)
      Index Cond: (m.fkgeneroid = generoid)
Planning Time: 0.673 ms
Execution Time: 0.045 ms
(32 rows)
```

postgres=#

RELATORIO2 COM INDEX EM DATA

```
Sort (cost=1057.29..1058.89 rows=638 width=78) (actual time=22.538..22.600 rows=1198 loops=1)
  Sort Keys: a:none
  Sort Method: quicksort Memory: 217kB
  -> Hash Left Join (cost=899.51..1027.57 rows=638 width=78) (actual time=18.852..20.697 rows=1198 loops=1)
    Hash Cond: (co.fkfuncionarioid = f.funcionarioid)
    -> Hash Join (cost=737.01..803.59 rows=638 width=68) (actual time=16.912..18.380 rows=1198 loops=1)
      Hash Cond: (a.autorid = am.fkautorid)
      -> Seq Scan on autor a (cost=0.00..95.00 rows=5000 width=18) (actual time=0.016..0.530 rows=5000 loops=1)
      -> Hash (cost=729.04..729.04 rows=638 width=58) (actual time=16.886..16.886 rows=1198 loops=1)
        Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 124kB
        -> Hash Right Join (cost=607.66..729.04 rows=638 width=58) (actual time=14.653..16.468 rows=1198 loops=1)
          Hash Cond: (g.generoid = m.fkgeneroid)
          -> Seq Scan on genero g (cost=0.00..90.00 rows=5000 width=18) (actual time=0.010..0.664 rows=5000 loops=1)
          -> Hash (cost=599.68..599.68 rows=638 width=48) (actual time=14.636..14.636 rows=1198 loops=1)
            Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 114kB
            -> Hash Right Join (cost=490.01..599.68 rows=638 width=48) (actual time=12.450..14.217 rows=1198 loops=1)
              Hash Cond: (tn.tipomidiaid = m.fktipomidiaid)
              -> Seq Scan on tipomidia tn (cost=0.00..78.33 rows=4833 width=18) (actual time=0.009..0.596 rows=4833 loops=1)
              -> Hash (cost=482.83..482.83 rows=638 width=38) (actual time=12.435..12.435 rows=1198 loops=1)
                Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 101kB
                -> Hash Join (cost=384.79..482.83 rows=638 width=38) (actual time=9.235..11.075 rows=1198 loops=1)
                  Hash Cond: (an.fkmidiaid = m.midiaid)
                  -> Seq Scan on autornidia an (cost=0.00..72.94 rows=4994 width=8) (actual time=0.018..0.847 rows=4994 loops=1)
                  -> Hash (cost=376.80..376.80 rows=639 width=42) (actual time=9.208..9.208 rows=568 loops=1)
                    Buckets: 1024 Batches: 1 Memory Usage: 51kB
                    -> Hash Join (cost=219.16..376.80 rows=639 width=42) (actual time=5.605..8.826 rows=568 loops=1)
                      Hash Cond: (m.midiaid = pc.fkmidiaid)
                      -> Seq Scan on midia m (cost=0.00..120.00 rows=5000 width=26) (actual time=0.011..1.279 rows=5000 loops=1)
                      -> Hash (cost=211.17..211.17 rows=639 width=16) (actual time=5.584..5.584 rows=568 loops=1)
                        Buckets: 1024 Batches: 1 Memory Usage: 35kB
                        -> Hash Join (cost=100.35..211.17 rows=639 width=16) (actual time=1.409..5.261 rows=568 loops=1)
                          Hash Cond: (pc.fkcomprauid = co.comprauid)
                          -> Seq Scan on produtoscomprados pc (cost=0.00..99.40 rows=4345 width=12) (actual time=0.018..2.061 rows=4326 loops=1)
                            Filter: (quantidade > 25)
                            Rows Removed by Filter: 666
                          -> Hash (cost=91.17..91.17 rows=735 width=12) (actual time=1.381..1.381 rows=729 loops=1)
                            Buckets: 1024 Batches: 1 Memory Usage: 40kB
                            -> Bitmap Heap Scan on compra co (cost=17.98..91.17 rows=735 width=12) (actual time=0.191..0.938 rows=729 loops=1)
                              Recheck Cond: (data > '2000-01-01'::date)
                              Heap Blocks: exact=60
                              -> Bitmap Index Scan on idx_compra_cliente (cost=0.00..17.79 rows=735 width=0) (actual time=0.163..0.163 rows=729 loops=1)
                                Index Cond: (data > '2000-01-01'::date)
                        -> Seq Scan on funcionario f (cost=0.00..100.00 rows=5000 width=18) (actual time=0.006..0.905 rows=5000 loops=1)
    -> Seq Scan on funcionario f (cost=0.00..100.00 rows=5000 width=18) (actual time=0.006..0.905 rows=5000 loops=1)
Planning Time: 11.511 ms
Execution Time: 22.794 ms
(47 rows)
```

7. Log/Backup

Os logs e Backups são feitos automaticamente pela AWS RDS todos os dias. Ele efetua um snapshot (Backup do momento exato), e são armazenados por até 7 dias.

8. Nuvem

Foi utilizado o AWS RDS para levar o banco de dados para nuvem. É preciso configurar a segurança da conexão, que nesse caso foi utilizado o acesso aberto.

RDS > Databases > postgres

postgres

Modify Actions

Summary

DB identifier postgres	CPU 3.00%	Info Available	Class db.t2.micro
Role Instance	Current activity 0 Sessions	Engine PostgreSQL	Region & AZ sa-east-1a

Connectivity & security Monitoring Logs & events Configuration Maintenance & backups Tags

Connectivity & security

Endpoint & port Endpoint postgres.czuc1renprc.sa-east-1.rds.amazonaws.com Port 5432	Networking Availability zone sa-east-1a VPC vpc-0a92926d Subnet group default-vpc-0a92926d Subnets subnet-b960dddf subnet-41ca711a subnet-c27ccf8b	Security VPC security groups default (sg-a65ecdda) (active) Public accessibility Yes Certificate authority rds-ca-2019 Certificate authority date Aug 22nd, 2024
--	---	--

9. Datalake

Foi utilizado o AWS S3, que permite o armazenamento de diversos dados em forma de *buckets*. Porém, nossa livreria não requer armazenamento de nenhum dado não estruturado.

Nome do bucket	Acesso	Região	Data de criação
livreria1	Público	América do Sul (São Paulo)	fev 16, 2020 9:05:21 PM GMT-0400

10. DataOps


Foi feito um pdf (Metadata of bookstore database.pdf) com os metadados do banco. Não foi possível colocar na nuvem pois era necessário uma licença paga.

postgres@postgres.czuc1renrprc.sa-east-1.rds.amazonaws.com

1. Tables

1.1. Table: public.autor

Columns

	Name	Data type	Description / Attributes
	autorid	integer	Default: nextval('autor_autorid_seq'::regclass)
	nacionalidade	character varying(255)	Nome do país em português
	nome	character varying(255)	Nome completo
	datanascimento	date	Data de nascimento
	datafalecimento	date	Data de falecimento, se tiver morrido

Linked from

	Table	Join	Title / Name / Description
	public.autormidia	public.autor .autorid = public.autormidia.fkautorid	autormidia_fkautorid_fkey

Unique keys

	Columns	Name / Description
	autorid	autor_pkey

1.2. Table: public.autormidia

Columns

	Name	Data type	Description / Attributes
	fkautorid	integer	References: public.autor
	fkmidiaid	integer	References: public.midia

Links to

	Table	Join	Title / Name / Description
	public.autor	public.autormidia .fkautorid = public.autor.autorid	autormidia_fkautorid_fkey
	public.midia	public.autormidia .fkmidiaid = public.midia.midiaid	autormidia_fkmidiaid_fkey

Unique keys

	Columns	Name / Description
	fkautorid, fkmidiaid	autormidia_pkey

1.3. Table: public.cliente

Columns

	Name	Data type	Description / Attributes
	clienteid	integer	Default: nextval('cliente_clienteid_seq'::regclass)

11. SQL Tabelas

```
DROP TABLE IF EXISTS ProdutosComprados;
DROP TABLE IF EXISTS Compra;
DROP TABLE IF EXISTS Cliente;
DROP TABLE IF EXISTS Funcionario;
DROP TABLE IF EXISTS AutorMidia;
DROP TABLE IF EXISTS Autor;
DROP TABLE IF EXISTS Revista;
DROP TABLE IF EXISTS Volume;
DROP TABLE IF EXISTS Manga;
DROP TABLE IF EXISTS Livro;
DROP TABLE IF EXISTS Midia;
DROP TABLE IF EXISTS TipoMidia;
DROP TABLE IF EXISTS Genero;
```

```
CREATE TABLE Genero (
    generoId SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    localizacao VARCHAR(255) NOT NULL
);
```

```
CREATE TABLE TipoMidia(
    tipoMidiaId SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    UNIQUE (nome)
);
```

```
CREATE TABLE Midia (
    midiaId SERIAL PRIMARY KEY,
    fkGeneroId INTEGER REFERENCES Genero(generoId) NOT NULL,
    fkTipoMidiaId INTEGER NOT NULL REFERENCES TipoMidia(tipoMidiaId),
    dataPublicacao date NOT NULL,
    editora VARCHAR(255) NOT NULL,
    nome VARCHAR(255) NOT NULL,
    idioma VARCHAR(255) NOT NULL,
    localPublicacao VARCHAR(255) NOT NULL,
    precoMidia FLOAT NOT NULL CHECK(precoMidia >= 0),
```

```
    quantidade INTEGER NOT NULL CHECK(quantidade >= 0)
);
```

```
CREATE TABLE Livro (
    livroId SERIAL PRIMARY KEY,
    fkMidiaId INTEGER REFERENCES Midia(midiaId) NOT NULL,
    sinopse VARCHAR(255) NOT NULL,
    edicao INTEGER NOT NULL CHECK(edicao > 0),
    paginas INTEGER NOT NULL CHECK(paginas > 0),
    UNIQUE(fkMidiaId)
);
```

```
CREATE TABLE Manga (
    mangaId SERIAL PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    adaptacaoAnime BOOLEAN NOT NULL,
    finalizado BOOLEAN NOT NULL
);
```

```
CREATE TABLE Volume (
    volumeId SERIAL PRIMARY KEY,
    fkMidiaId INTEGER REFERENCES Midia(midiaId) NOT NULL,
    fkMangaId INTEGER REFERENCES Manga(mangaId) NOT NULL,
    sinopse VARCHAR(255) NOT NULL,
    numero FLOAT NOT NULL CHECK(numero >= 0),
    quantidadeCapitulos INTEGER NOT NULL CHECK(quantidadeCapitulos > 0),
    UNIQUE(fkMidiaId)
);
```

```
CREATE TABLE Revista (
    revistaId SERIAL PRIMARY KEY,
    fkMidiaId INTEGER REFERENCES Midia(midiaId) NOT NULL,
    empresa VARCHAR(255) NOT NULL,
    edicao INTEGER NOT NULL CHECK(edicao > 0),
    UNIQUE(fkMidiaId)
);
```

```
CREATE TABLE Autor (
    autorId SERIAL PRIMARY KEY,
    nacionalidade VARCHAR(255) NOT NULL,
    nome VARCHAR(255) NOT NULL,
```

```
dataNascimento DATE NOT NULL,  
dataFalecimento DATE NOT NULL  
);
```

```
CREATE TABLE AutorMidia (  
    fkAutorId INTEGER REFERENCES Autor(autorId),  
    fkMidiaId INTEGER REFERENCES Midia(midiaId),  
    UNIQUE(fkAutorId, fkMidiaId),  
    PRIMARY KEY(fkAutorID, fkMidiaId)  
);
```

```
CREATE TABLE Funcionario (  
    funcionarioId SERIAL PRIMARY KEY,  
    funcao VARCHAR(255) NOT NULL,  
    nome VARCHAR(255) NOT NULL,  
    salario FLOAT NOT NULL CHECK(salario > 0),  
    dataAdmissao DATE NOT NULL  
);
```

```
CREATE TABLE Cliente (  
    clienteId SERIAL PRIMARY KEY,  
    quantidadeCompras INTEGER NOT NULL DEFAULT 0,  
    endereco VARCHAR(255) NOT NULL,  
    sexo VARCHAR(255) NOT NULL,  
    nome VARCHAR(255) NOT NULL,  
    dataNascimento DATE NOT NULL,  
    vip BOOLEAN NOT NULL DEFAULT FALSE  
);
```

```
CREATE TABLE Compra (  
    compraId SERIAL PRIMARY KEY,  
    fkCLienteId INTEGER REFERENCES Cliente(clienteId) NOT NULL,  
    fkFuncionarioId INTEGER REFERENCES Funcionario(funcionarioId) NOT NULL,  
    data DATE NOT NULL,  
    preco FLOAT NOT NULL CHECK(preco >= 0) DEFAULT 0  
);
```

```
CREATE TABLE ProdutosComprados (  
    produtosCompradosId SERIAL PRIMARY KEY,  
    fkCompraId INTEGER REFERENCES Compra(compraId) NOT NULL,  
    fkMidiaId INTEGER REFERENCES Midia(midiaId) NOT NULL,  
    quantidade INTEGER NOT NULL CHECK(quantidade > 0),
```

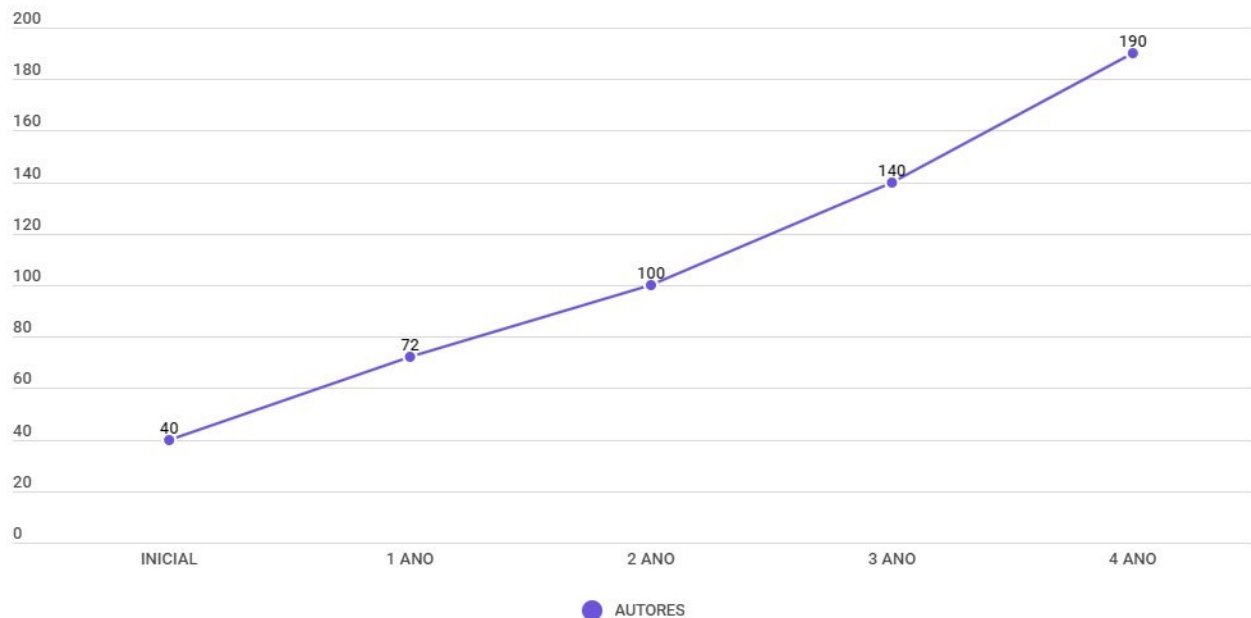
```
precoUnidade FLOAT CHECK(precoUnidade >= 0),
descontoUnidade FLOAT NOT NULL CHECK(descontoUnidade >= 0) DEFAULT 0,
CHECK (precoUnidade >= descontoUnidade),
UNIQUE(fkCompraId, fkMidiaId)
);
```

12. Crescimento de dados

Autor inicia com 40 e aumenta com base na função:

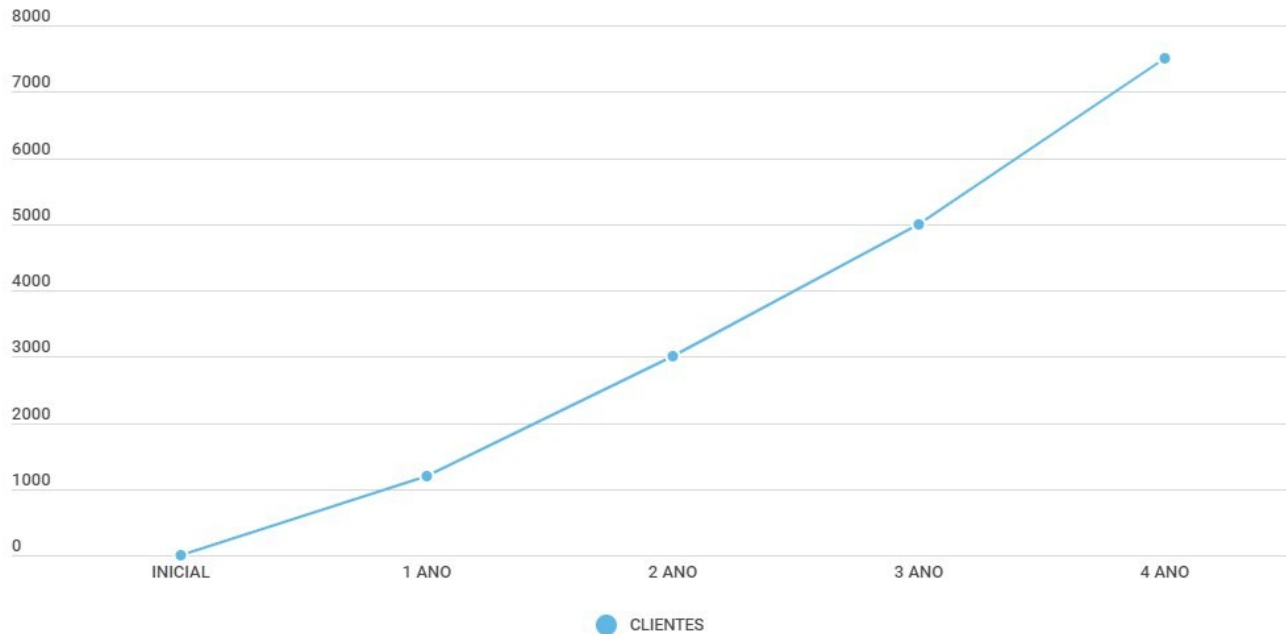
$$\text{Autor} = \text{int}(\text{Autor} + ((20 + \text{ano}) ** (1 + (\text{ano}/100 * 5))))$$

Sendo 20 o valor base de aumento anual e 5% de crescimento médio



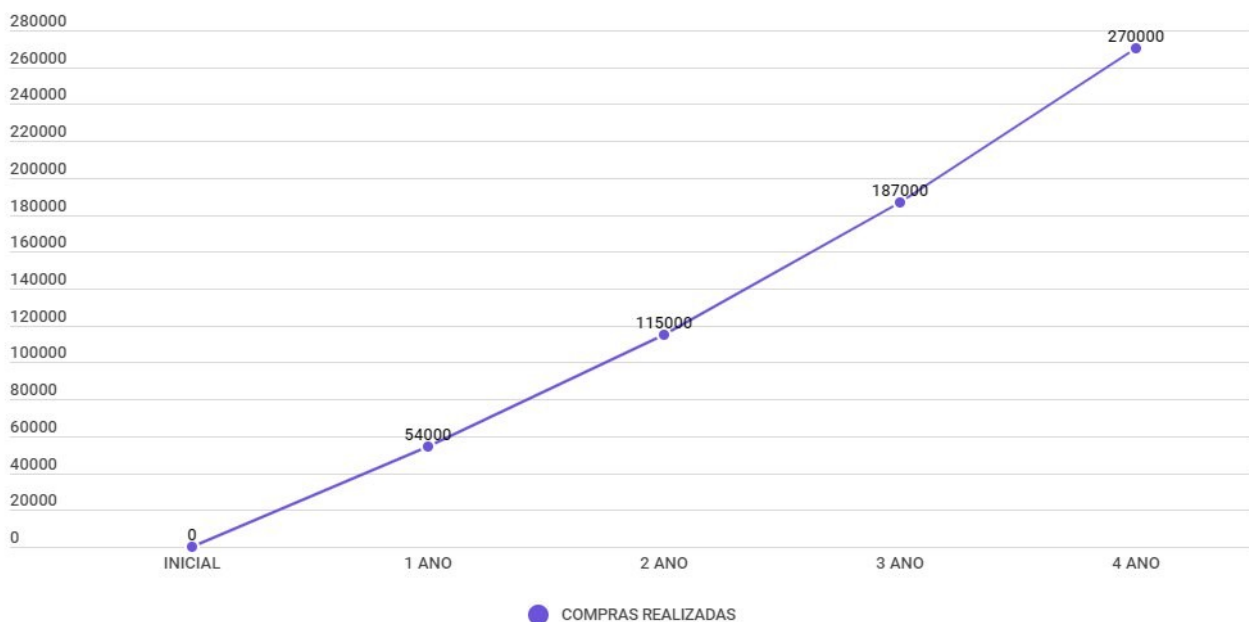
Cliente inicia com 0 e aumenta com base na função :
$$\text{Clientes} = \text{int}(\text{Clientes} + ((720 + \text{ano}) ** (1 + (\text{ano}/100 * 5))))$$

Sendo 720 o valor base de aumento anual e 5% de crescimento médio



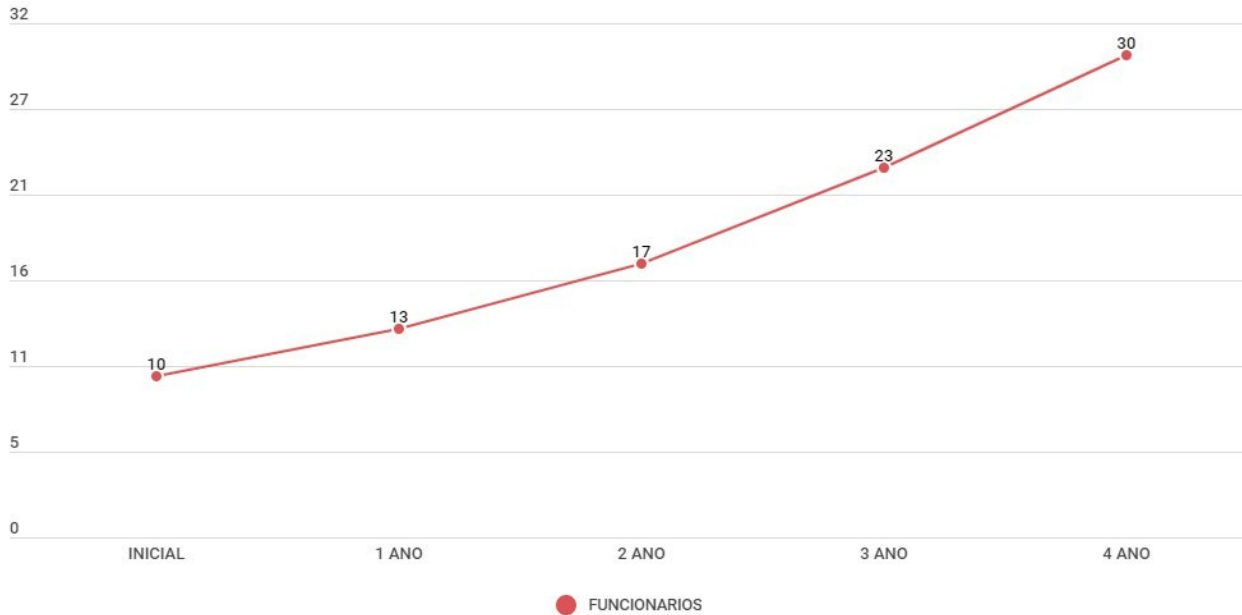
Compras Realizadas inicia com 0 e aumenta com base na função :
$$\text{comprasRealizadas} = \text{int}(\text{comprasRealizadas} + ((36000 + \text{ano}) ** (1 + (\text{ano}/100 * 2))))$$

Sendo 36000 o valor base de aumento anual e 2% o crescimento médio



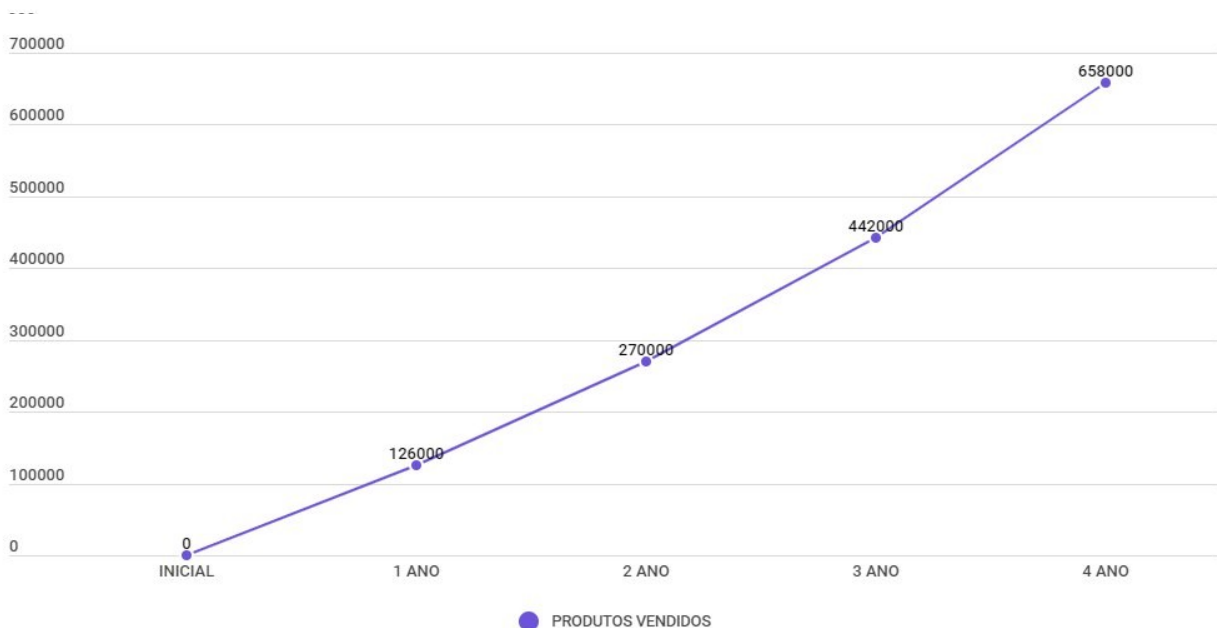
Funcionario inicia com 10 e aumenta com base na função :
$$\text{Funcionarios} = \text{int}(\text{Funcionarios} + ((2 + \text{ano}) ** (1 + (\text{ano}/100 * 5))))$$

Sendo 2 o valor base de aumento anual e 5% o crescimento médio



Produtos Vendidos inicia com 0 e aumenta com base na função :
$$\text{produtosVendidos} = \text{int}(\text{produtosVendidos} + ((72000 + \text{ano}) ** (1 + (\text{ano}/100 * 2))))$$

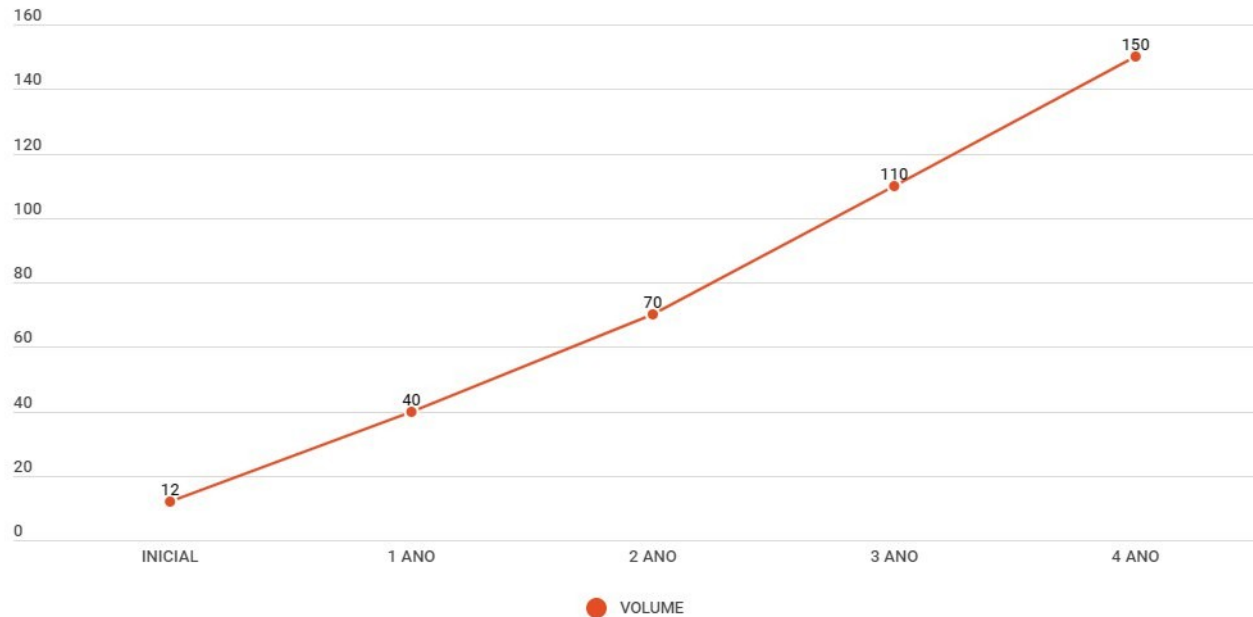
Sendo 72000 o valor base de aumento anual e 2% o crescimento médio



Volume inicia com 12 e aumenta com base na função :

$$\text{Volume} = \text{int}(\text{volume} + ((20 + \text{ano}) ** 1 + (\text{ano}/100 * 5)))$$

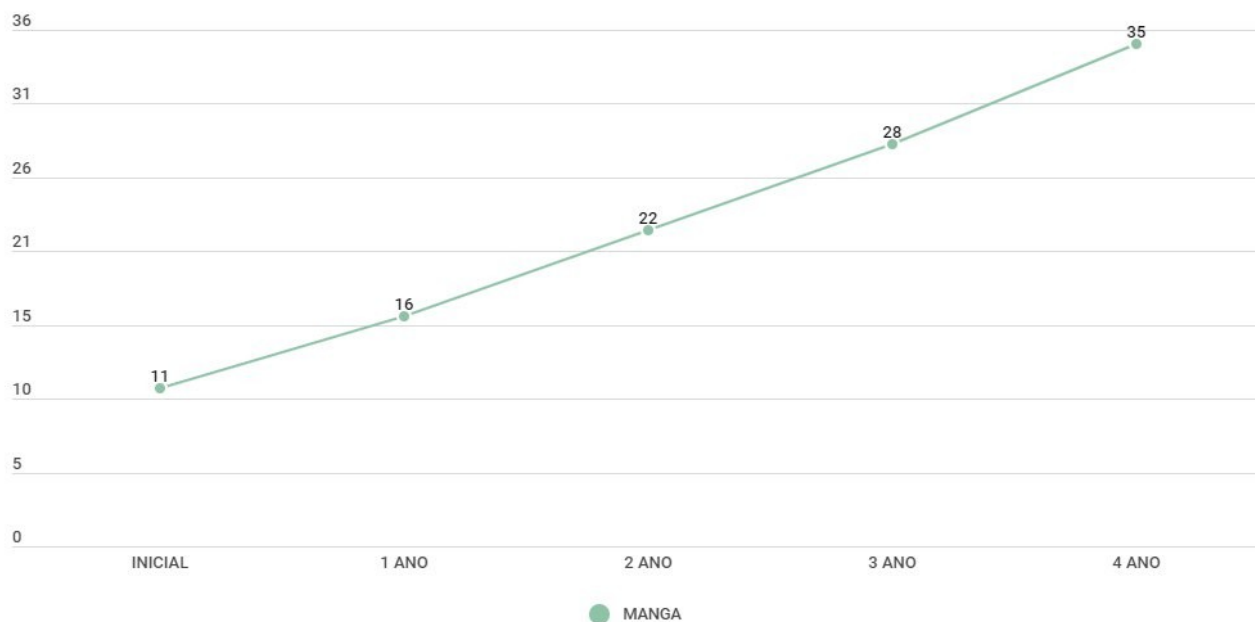
Sendo 12 o valor base de aumento anual e 5% o crescimento médio



Manga inicia com 11 e aumenta com base na função :

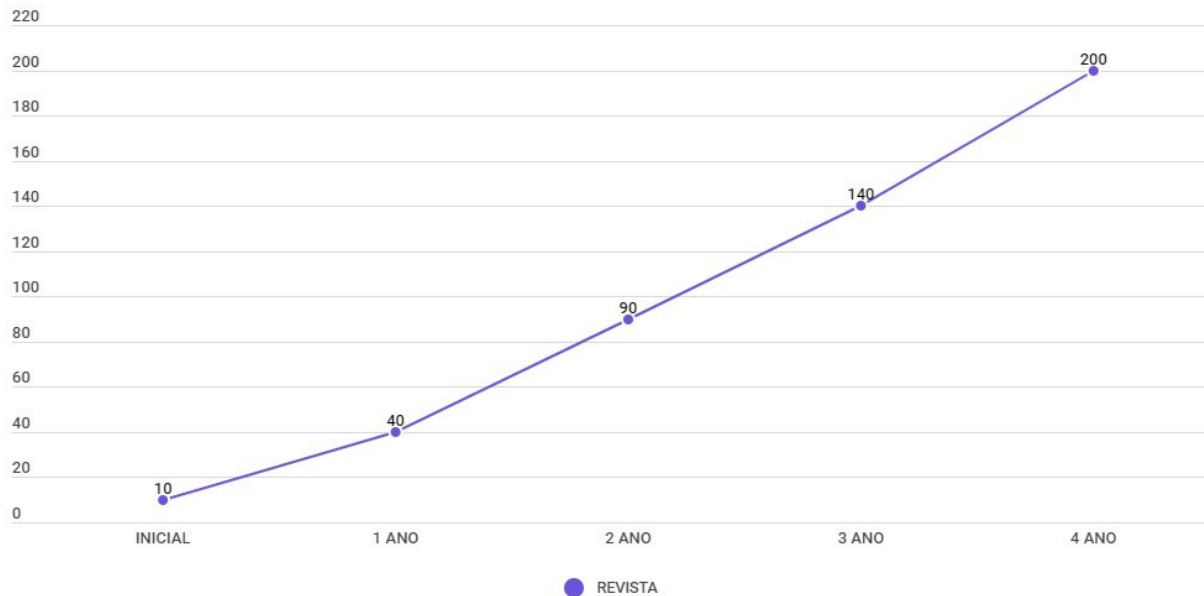
$$\text{Manga} = \text{int}(\text{Manga} + ((12 + \text{ano}) ** (1 + (\text{ano}/100 * 5))))$$

Sendo 12 o valor base de aumento anual e 5% o crescimento médio



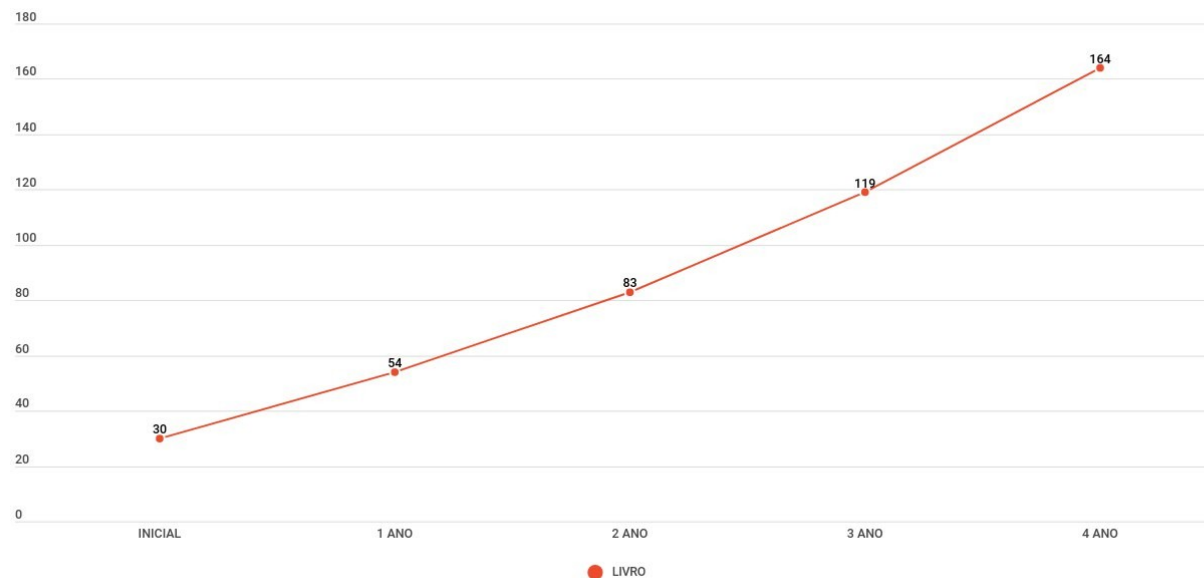
Revista inicia com 10 e aumenta com base na função :
$$\text{Revista} = \text{int}(\text{Revista} + ((24 + \text{ano}) ** (1 + (\text{ano}/100 * 5))))$$

Sendo 24 o valor base de aumento anual e 5% o crescimento médio



Livro inicia com 30 e aumenta com base na função :
$$\text{Livro} = \text{int}(\text{Livro} + ((20 + \text{ano}) ** (1 + (\text{ano}/100 * 5))))$$

Sendo 20 o valor base de aumento anual e 5% o crescimento médio



13. Versionamento

Foi utilizado o flyway para fazer o versionamento. Todos os SQL's estão dentro da pasta flyway-6.0.7/sql.

