**UNIVERSIDADE DO OESTE DE SANTA CATARINA**

**CAMPUS DE SÃO MIGUEL DO OESTE**

**BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO**

EVERTO CARLOS HOLLEWEIGER

GUILHERME ZENIN

**DATACINE: GESTÃO PARA CINEMAS**

São Miguel Do Oeste/SC

2022

EVERTO CARLOS HOLLEWEIGER

GUILHERME ZENIN

DATACINE: GESTÃO PARA CINEMAS

Trabalho Acadêmico dos componentes de Banco de Dados II, Engenharia de Software I e Programação II do curso de Ciências da Computação da Universidade do Oeste de Santa Catarina no campus de São Miguel do Oeste.

Professores: Roberson Junior Fernandes Alves, Franciele Carla Petry e Otilia Donato Barbosa.

São Miguel Do Oeste/SC

2022

**LISTA DE ILUSTRAÇÕES**

**FIGURAS**

[Figura 1: Diagrama relacional proposto para o DataCine 8](#_Toc107246462)

[Figura 2: Script utilizado para criação do banco de dados. 9](#_Toc107246463)

[Figura 3: Script utilizado para criação da tabela filme\_premiacao. 10](#_Toc107246464)

[Figura 4: Script utilizado para criação da tabela filme\_premiacao. 10](#_Toc107246465)

[Figura 5: Informações inseridas na tabela cinema. 10](#_Toc107246466)

**SUMÁRIO**

[1 INTRODUÇÃO 5](#_Toc107249750)

[2 DESENVOLVIMENTO 6](#_Toc107249751)

[2.1 PESQUISAS E IDEIAS INICIAIS 6](#_Toc107249752)

[2.2 REQUISITOS E DIAGRAMA RELACIONAL 6](#_Toc107249753)

[2.3 SCRIPTS 8](#_Toc107249754)

[2.3.1 *Script* de criação do banco de dados. 9](#_Toc107249755)

[2.3.2 Script de geração das tabelas 9](#_Toc107249756)

[2.3.2 Script de insert das tabelas 10](#_Toc107249757)

[2.4 RELATÓRIOS DO MODELO DE NEGÓCIO 11](#_Toc107249758)

[2.5 DICIONÁRIO DE DADOS 11](#_Toc107249759)

[3 CONCLUSÃO 12](#_Toc107249760)

# 1 INTRODUÇÃO

O Sistema DataCine, é um projeto de *Software* desenvolvido para as empresas, cujo ramo de atividade trata-se de salas de cinema. O projeto foi desenvolvido pensando em atender cinemas de pequeno, médio e até grande porte, devido a suas inúmeras possibilidades de gestão.

A elaboração de um *Software* pode ser dividida em alguns seguimentos: o uso de metodologias conhecidas dentro da Engenharia de Software para análise de requisitos e necessidades do *Software* em si, bem como a utilização de métodos ágeis e diagramas, para que o projeto se mantenha organizado e explicativo durante todo o desenvolvimento para que chegue à conclusão conforme o planejado. A estruturação de um banco de dados de acordo com regras de negócio e por fim, o desenvolvimento do código de programação de acordo com todo o pressuposto antes realizado.

É importante ressaltar que o banco de dados e os relatórios utilizando linguagem SQL são os mesmos utilizados na fase inicial desse projeto realizado durante a 3ª fase do curso de Ciência da Computação, no decorrer da disciplina de Banco de Dados I. A partir deste modelo, foram introduzidas novas atualizações de melhorias e desenvolvimento de atividades para atender as novas requisições propostas.

Neste artigo serão apresentados, os métodos, *scripts*, diagramas e a descrição de cada fase do projeto, explicando a realização de cada tarefa. Todos os arquivos aqui mencionados poderão ser encontrados no repositório do GitHub, através do link: <https://github.com/evertoch/DataCine> .

# 2 DESENVOLVIMENTO

O desenvolvimento deste trabalho ocorreu de forma multidisciplinar, envolvendo os seguintes componentes curriculares: Banco de Dados II, Engenharia de Software I e Programação II, sendo assim, a fim de transcrever o projeto de maneira didática, o desenvolvimento teórico foi elaborado e dividido da mesma forma.

## 2.1 BANCO DE DADOS II

## **2.1.1 Contextualização e ideias iniciais**

Inicialmente, ao ser proposto a ideia de criação de um banco de dados, foi pensado em nichos pouco explorados, onde as opções de sistema fossem limitadas. Baseado nisso, surgiu a ideia do DataCine, um sistema voltado para o gerenciamento de cinemas. Com o segmento definido, foi realizado uma análise das reais necessidades que um Cinema pode possuir em sua rotina de trabalho.

Com todas as informações necessárias, foi elaborado o Modelo Relacional do banco de dados, sendo possível definir as entidades e atributos, bem como as especificações de cada um, sejam estes chave primária (*primary key*, ou simplesmente, PK), chave estrangeira (*foreign key*, ou simplesmente FK), campos não nulos (*not null),* e únicos *(unique).* Em adição, foram criados *scripts* de criação das tabelas, *inserts* de informações e o desenvolvimento de relatórios previamente requisitados.

Com base no modelo inicialmente feito, foi dado sequência com o desenvolvimento do projeto.

## **2.1.2 Modelo Relacional**

O modelo relacional representa os dados num Banco de Dados como uma coleção de tabelas (relações). Cada tabela terá um nome, que será único, e um conjunto de atributos com seus respectivos nomes e domínios. O objetivo do modelo relacional elaborado é explanação da estrutura do banco de dados em um esquema de fácil visualização, sendo uma peça fundamental para manutenções, implementações e correções que podem ocorrer no banco de dados, visto que o usuário terá a estrutura completa para eventuais consultas.

Para a nomenclatura dos componentes do diagrama, foi adotado a seguinte padronização: Tabelas principais utilizam o nome da informação que será guardada, sem utilizar letra maiúscula, acentuação ou caracteres especiais, exemplo: filme. Tabelas intermediárias (fruto da relação “N:N” ou “N:M”, lê-se muitos para muitos), utiliza o nome das duas tabelas principais separados por *underline*, exemplo: filme\_premiacao. Para as colunas, presente em cada tabela, utiliza as três primeiras letras de cada palavra, sem nenhum componente para separação, exemplo: durfil (ou seja, duração do filme).

O diagrama relacional elaborado foi construído utilizando a ferramenta Visual Paradigm e pode ser visto na figura 1 abaixo:

*Figura 1: Primeira versão modelo relacional*

Diagrama

Descrição gerada automaticamente

Fonte: Os autores (2022).

Visando uma estrutura sem redundância e com a possibilidade mínima de erros, o diagrama foi reestruturado seguindo as boas práticas propostas pela normalização das Formas Normais e implementando alterações previstas no projeto, como a adição de novos campos, alterações em campos já existentes e a adição de um controle de tickets e cardápio. Além disso, foi adicionado a tabela filme\_auditoria, responsável pelo armazenamento de logs de alterações que possam ocorrer na tabela filme, que será abordado com mais detalhes no decorrer deste trabalho.

O diagrama relacional em sua versão final elaborado pode ser visto na figura 2 abaixo e na íntegra, no diretório do projeto no GitHub (<https://github.com/evertoch/DataCine> Banco de Dados II >Mapa Relacional > DataCine.vpp).

*Figura 2: Modelo Relacional versão final.*

Diagrama

Descrição gerada automaticamente

Fonte: Os autores (2022).

## **2.1.3 *Scripts***

Com a parte teórica atualizada, foi iniciado a parte prática, ou seja, a criação do banco de dados baseado em todas as novas informações e no Modelo Relacional (MR) criado.

### 2.1.3.1 *Script* de criação do banco de dados.

O *script* de criação do banco de dados não sofreu, consistindo no seguinte padrão: create database + “nome do banco de dados”;, neste caso: create database datacine;.

A figura a seguir mostra o SQL utilizado:

Figura 3: Script utilizado para criação do banco de dados.



Fonte: Os autores (2022).

O arquivo completo poderá ser encontrado no repositório do GitHub com o nome “Create\_Database\_DataCine.sql”.

### 2.1.3.2 *Script* de geração das tabelas

Com novas informações acrescentadas no MR, foi necessário atualizar o *script* da geração das tabelas.

No padrão utilizado, o SQL é divido entra a criação da tabela, definição dos atributos, os tipos de dados de cada atributo, a chave primária da tabela e os comentários. O fim do código é reservado para geração de todas as demais *constraints,* composto, em sua grande maioria, pelas chaves estrangeiras.

A Figura 4 mostra o SQL utilizado para a criação da tabela filme\_premiacao:

Figura 4: Script utilizado para criação da tabela “filme\_premiacao”.

Texto

Descrição gerada automaticamente

Fonte: Os autores (2022).

A Figura 4 mostra o SQL utilizado para a criação das *constraints* da tabela, especificadas no final do arquivo.sql:

Figura 5: Script utilizado para criação da tabela filme\_premiacao.



Fonte: Os autores (2022).

O arquivo completo poderá ser encontrado no repositório do GitHub com o nome “Create\_Database\_Tables.sql”.

### 2.1.3.3 *Script* de inserção das tabelas

Com a estrutura do banco de dados pronta, foi realizado a inserção de dados em todas as tabelas, a fim de certificado o pleno funcionamento de toda a estrutura e para geração dos relatórios propostos no Modelo de Negócio. Para campos “auto incremento”, as informações sequenciais foram geradas pelo próprio Banco de Dados. Para os demais campos, a inserção ocorreu de forma manual com informações fictícias.

A Figura 6 mostra o SQL utilizado para a inserção na tabela cinema:

Figura 6: Script utilizado para a inserção na tabela “cinema”.



Fonte: Os autores (2022).

O arquivo completo poderá ser encontrado no repositório do GitHub com o nome “Insert\_Database\_Tables.sql”.

## 2.1.4 **Índices (*Index*)**

Por tratar-se de um banco de dados com pouco cadastro de dados, a criação de índices passa a ser apenas teórico, visto que o banco de dados não irá utilizá-los. Os índices foram adicionados em todas as chaves estrangeiras, e nas colunas que serão mais utilizadas na prática, como pode ser observado no Apêndice A (*Script* da criação dos índices). O arquivo na íntegra poderá ser encontrado no repositório do GitHub com o nome “Create\_Index.sql”.

## **2.1.5 Gatilhos (Triggers)**

As triggers são mecanismos ativados automaticamente quando uma ação pré-estabelecida acontece. As triggers podem ser utilizadas para estabelecer algumas regras de negócio, auditoria, controle das regras de integridade etc. No PostgreSQL, elas vêm acompanhadas de uma procedure ou uma function

A primeira trigger implementada é a “filme\_afeter\_tg”, que executa a function “filme\_log\_func” e é utilizada para gravar na tabela “filme\_auditoria” as informações que forem inseridas, alteradas ou excluídas da tabela “filme”. O script do gatilho pode ser visualizado no Apêndice B

A segunda trigger implementada é a disponibilidade\_before\_tg, que executa a function “verifica\_disponibilidade” e é utilizada sempre antes da venda de um ingresso para verificar se ainda há assentos vazios na sessão do filme. A busca fará uma comparação com a capacidade da sala com o total de ingressos vendidos para aquela sessão, caso o número de ingressos vendidos seja igual a capacidade da sala, o banco de dados não permitirá novas inserções na tabela de ingresso. O script do gatilho pode ser visualizado no Apêndice C.

O arquivo na íntegra contendo as duas triggers citadas, bem como seu funcionamento em detalhes, poderá ser encontrado no repositório do GitHub com o nome “Triggers.sql”.

## **2.1.6 Funções (Functions) e Procedures**

As functions ou também chamadas de procedure, são recursos muito utilizados no gerenciamento de banco de dados, pois mantém concentrada a lógica necessária para determinadas funções, tendo assim uma maior agilidade no retorno de informações importantes. Com base nas regras de negócios, foi implementado duas functions para o banco de dados.

A primeira function implementada é a “func\_bilheteria”, utilizada para calcular a bilheteria de um filme, trazendo o total de ingressos e o valor arrecadado. O parâmetro a ser passado na chamada da function deverá ser o ID do filme. O script da function pode ser visualizado no Apêndice D.

A segunda function implementada é a “func\_qtd\_pro”, utilizada para calcular a quantidade total de um determinado alimento vendido e o valor total de suas vendas (somando o valor de cada item). O parâmetro a ser passado na chamada da function deverá ser o ID do alimento. O script da function pode ser visualizado no Apêndice E.

O arquivo na íntegra contendo as duas functions citadas, bem como seu funcionamento em detalhes, poderá ser encontrado no repositório do GitHub com o nome “Procedures.sql”.

## **2.1.7 Configuração de políticas de acesso**

Com base nas possibilidades de gestão que o software DataCine oferece, foram pensadas em permissões para os usuários e suas respectivas funções. Para controle de funções, foram criados os seguintes grupos: TI, Gerente, Administrativo e Atendente.

Para o grupo TI foi concedido permissão total em todas as tabelas do banco de dados, pois será este o responsável por gerir a fim de buscar o melhor aproveitamento do sistema.

Para o grupo Gerente foi concedido apenas as permissões de visualizar (*select*), cadastrar (*insert*), atualizar (*update*) e excluir (*delete*), contudo tais permissões aplicam-se em todas as tabelas do sistema, visto que o mesmo realiza a gestão da empresa e deve ter acesso em todas as áreas.

Para o grupo Administrativo, foram concedidas as permissões visualizar (*select*), cadastrar (*insert*), atualizar (*update*) e excluir (*delete*) apenas nas tabelas respectivas às suas atribuições, ou seja, filme, diretor, genero, sala, sala\_filme, premiacao, filme\_premiacao.

Para o grupo Atendente, foram concedidas as permissões visualizar (*select*), cadastrar (*insert*), atualizar (*update*) e excluir (*delete*) apenas nas tabelas respectivas às suas atribuições, ou seja, filme, ticket, ingresso.

O script foi dividido em algumas etapas, começando pela criação dos grupos com base nos setores, e atribuições das permissões para estas. Por último, foi criado um usuário para cada grupo, atribuindo login e senha. O script pode ser visualizado no Apêndice C (Script da criação dos privilégios de usuários). O arquivo na íntegra poderá ser encontrado no repositório do GitHub com o nome “Criação das políticas de acesso.sql”.

## 2.4 RELATÓRIOS DO MODELO DE NEGÓCIO

Com toda a estrutura do banco de dados pronta e todas as informações inseridas, chegou a hora de testar o mesmo com os relatórios propostos pelo Modelo de Negócios. Os relatórios propostos foram os seguintes:

* Relacionar o nome do filme, tempo de duração e o gênero de todos os filmes. Ordene o relatório do filme mais longo(tempo) para o filme mais curto;
* Relacionar o nome do filme e o nome da sala para todos os filmes. Filtre somente filmes com duração mínima de 90 minutos. Ordene o relatório de forma descendente pelo nome da sala;
* Relacionar o código do filme, nome do filme, mês da bilheteria, quantidade total de ingressos vendidos no mês para meses entre anos de 2020 e 2021. Ordene o relatório do filme com mais vendas(em termos de quantidade de ingressos) para o filme com menos vendas;
* Relacionar o código do filme, nome do filme e o total de bilheteria(valores). Filtrar somente filmes, excetuando-se dramas, lançados a partir de 2018 e com restrição de idade para maiores de 12 anos. Ordene o relatório do filme com mais bilheteria para o filme com menos bilheteria.

O arquivo completo com os comandos SQL utilizados para gerar os relatórios propostos, poderá ser encontrado no repositório do GitHub com o nome “Relatório.sql”.

## 2.5 DICIONÁRIO DE DADOS

O dicionário de dados é um arquivo contendo toda a estrutura detalhada do banco de dados, bem como os comentários e informações inseridas. O mesmo foi gerado utilizando as informações cadastradas no Visual Paradigm.

O arquivo completo com o dicionário de dados completo, poderá ser encontrado no repositório do GitHub com o nome “Dicionário de dados.docx”.

# 3 CONCLUSÃO

Com base nas informações apresentadas neste trabalho, conclui-se ser de suma importância o planejamento antecipado de como proceder antes e durante o desenvolvimento do banco de dados. É visto também que a documentação das informações, seja ela através de comentários ou do dicionário de dados é fundamental para um bom controle das informações a serem desenvolvidas, além de facilitar os processos de criação e manutenção do banco de dados.

APÊNDICE A - *Script* de criação dos índices

-- Criação dos índices chaves estrangeiras

create index fun\_codcin\_sk

on funcionario(codcin);

create index fil\_idgen\_sk

on filme(idgen);

create index fil\_iddir\_sk

on filme(iddir);

create index ing\_id\_sal\_fil\_sk

on ingresso(id\_sal\_fil);

create index hor\_fun\_idfuncao\_sk

on horario\_funcionario(idfuncao);

create index hor\_fun\_idhor\_sk

on horario\_funcionario(idhor);

create index sal\_fil\_idfilm\_sk

on sala\_filme(idfilm);

create index sal\_fil\_idsala\_sk

on sala\_filme(idsala);

create index sal\_fil\_idhor\_sk

on sala\_filme(idhor);

create index fil\_pre\_idfilm\_sk

on filme\_premiacao(idfilm);

create index fil\_pre\_idpre\_sk

on filme\_premiacao(idpre);

create index tic\_idfilm\_sk

on ticket(idfilm);

create index tic\_idali\_sk

on ticket(idali);

-- Criação dos índices colunas mais usadas

create index fil\_nombr\_sk

on filme(nombr);

create index fil\_nomen\_sk

on filme(nomen);

create index ing\_valing\_sk

on ingresso(valing);

create index funcao\_nomfun\_sk

on funcao(nomfun);

create index funcionario\_nomfun\_sk

on funcionario(nomfun);

APÊNDICE B - Script de criação da trigger “filme\_afeter\_tg”

-- Criação da Procedure:

create or replace function filme\_log\_func()

returns trigger

as

$$

Begin

insert into filme\_auditoria (id\_filme\_new, id\_filme\_old, dat\_alt, usu\_alt, dml)

values (new.idfilm, old.idfilm, current\_timestamp, current\_user, TG\_OP);

return new;

END

$$

Language plpgsql;

-- Criação da Trigger:

create trigger filme\_afeter\_tg

after insert or update or delete

on filme

for each row

execute procedure filme\_log\_func();

APÊNDICE C - Script de criação da trigger “disponibilidade\_before\_tg”

create or replace function verifica\_disponibilidade()

returns trigger

as

$body$

declare

conta numeric;

capacidade numeric;

begin

select count(\*) into conta from ingresso i

inner join sala\_filme sf on i.id\_sal\_fil = sf.id\_sal\_fil

inner join sala s on sf.idsala = s.idsal

where i.id\_sal\_fil = new.id\_sal\_fil;

select capsal into capacidade from sala s

inner join sala\_filme sf on s.idsal = sf.idsala

inner join ingresso i on sf.id\_sal\_fil = i.id\_sal\_fil

where i.id\_sal\_fil = new.id\_sal\_fil

group by capsal;

if (conta >= capacidade) then

raise exception 'Não há mais assentos disponíveis para a sessão atual';

end if;

return new;

end

$body$

language plpgsql;

-- Criação da Trigger:

create trigger disponibilidade\_before\_tg

before insert on ingresso

for each row

execute procedure verifica\_disponibilidade();

APÊNDICE D - Script de criação da function “func\_bilheteria”

-- Criação da Procedure:

create or replace function func\_bilheteria(id\_film integer)

returns SETOF record as

$body$

declare

begin

return query

select id\_film, count(\*), sum(valing) from ingresso i

inner join sala\_filme sf on i.id\_sal\_fil = sf.id\_sal\_fil

inner join filme f on sf.idfilm = f.idfilm

where f.idfilm = id\_film;

return;

end;

$body$

language plpgsql;

-- Chamada da procedure com o parâmetro (id\_fil).

select \* from func\_bilheteria(2) as (Id\_filme integer, ingressos\_vendidos int8, valor\_total numeric);

APÊNDICE E - Script de criação da function “func\_qtd\_pro”

-- Criação da Procedure:

create or replace function func\_qtd\_pro(id\_alimento integer)

returns SETOF record as

$body$

declare

begin

return query

select desali, count(valali), sum(valali) from ticket t

inner join cardapio c ON t.idali = c.idali

where c.idali = id\_alimento

group by desali;

return;

end;

$body$

language plpgsql;

-- Chamada da Procedure:

select \* from func\_qtd\_pro(1) as (Descricao varchar, Quantidade int8, valor\_total int8);

APÊNDICE F - Script da criação dos privilégios de usuários

-- Criação dos grupos.

create group TI;

create group Gerente;

create group Administrativo;

create group Atendente;

-- Criação dos usuários.

create user everto

with

superuser

password 'everto123';

create user gerente1

with password 'gerente123';

create user administrativo1

with password 'administrativo123';

create user atendente1

with password 'atendente123';

-- Adicionando os usuários aos grupos;

grant ti to everto;

grant gerente to gerente1;

grant administrativo to administrativo1;

grant atendente to atendente1;

-- Conceções de permissões ao grupo "TI".

-- ao grupo de "TI" é concedido total privilégio em todas as tabelas.

grant all privileges

on all tables in schema public

to ti;

-- Conceções de permissões ao grupo "Gerente".

-- Ao grupo de "gerente" são concedidos apenas as permissões permitindo-o "mexer" de forma básica no banco, mas em todas as tabelas.

grant select, insert, delete, update

on all tables in schema public

to gerente;

-- Conceções de permissões ao grupo "Administrativo".

-- Ao grupo de "Administrativo" são concedidos algumas permissões básicas, mas tabelas pertinentes apenas a sua devida função.

grant select, insert, delete, update

on filme, diretor, genero, sala, sala\_filme, premiacao, filme\_premiacao

to administrativo;

-- Conceções de permissões ao grupo "atendente".

-- Ao grupo de "atendente" são concedidos as mesmas permissões que o grupo "administrativo", mas apenas nas tabelas pertinentes apenas a sua devida função.

grant select, insert, delete, update

on filme, ticket, ingresso

to atendente;