Escola Superior
de Tecnologia
e Gestão
Politécnico de Coimbra

Relatório Projeto Final

Bases de Dados

Modelação de dados recorrendo a Diagramas de Entidade-Relacionamento

Autores:
André Sequeira
a2022123691@estgoh.ipc.pt

Mariana Magalhães
a2022147454@estgoh.ipc.pt

Data: Janeiro 2024

Resumo

Este relatório é relativo ao Projeto Final da cadeira de Bases de Dados. Que tem como objetivo a análise conceptual e física de uma base de dados assim como a definição do seu plano de transações, geração do código SQL e estudo das dependências funcionais.

Palavras-chave

Análise Relacional, Concorrência, Dependências Funcionais, Gestão de Concorrência, Históricos, Modelo Conceptual, Modelo Físico, Normalização, Plano de Transacções, Restrições de Integridade, Tempo.

Índice

Lista de Figuras

Lista de Acrónimos

BD	Bases de Dados
DBMS	*Database Management System*
ER	Modelo Entidade-Relacionamento
ESTGOH	Escola Superior de Tecnologia e Gestão
FNBC	Forma Normal de *Boyce-Codd*
IPC	Instituto Politécnico de Coimbra
SGBD	Sistema Gestor de Bases de Dados
SQL	*Structured Query Language*

1. Introdução

Este relatório tem como objetivo descrever um projeto de base de dados para uma papelaria/livraria de apoio a uma escola. O projeto abrange a análise conceptual e física da base de dados, incluindo a definição do plano de transações, código SQL e estudo das dependências funcionais.

Esta base de dados precisa fazer a gestão de informações como unidades curriculares, apontamentos, estudantes, docentes, funcionários, materiais para venda, livros assim como os respectivos autores e editoras, fornecedores e ainda registar processos tanto de entrega de apontamentos como de confirmação de vendas .

Ao longo do relatório, serão apresentados o modelo conceptual através de um diagrama Entidade-Relacionamento, o modelo físico, e o código SQL para criação e manipulação da base de dados tendo em conta os requisitos pedidos no enunciado. Também serão delineados o plano de transações, visando a integridade dos dados, e o estudo das dependências funcionais.

2. Objetivos e Metodologias

2.1. Ferramentas e Tecnologias

PowerDesigner

Visual Studio Code (extensões: SQLTools e SQLTools MySQL/MariaDB/TiDB)

2.2. Planeamento

Este projeto foi iniciado no dia 19 de dezembro durante a aula de Base de Dados I e concluído após 9 dias de trabalho, totalizando aproximadamente 30 horas de trabalho.

Durante o projeto, criamos e conectamos várias tabelas de banco de dados.Esta etapa do trabalho foi realizada com a contribuição de todos os elementos do grupo, pois era a parte mais complexa e servia de base para o restante trabalho realizado. Sendo preciso que todos os elementos estejam de acordo com as decisões tomadas para prosseguir com o restante trabalho.

3. Análise da Instituição

3.1. Descrição Geral

A instituição educacional, representada pela base de dados, tem várias tabelas para gerenciar suas operações diárias. As tabelas criadas foram Apontamentos, Autor, Docente, Editora, Encomendas, Entregas, Estudantes, Fornecedores, Funcionarios, Livro, Materiais, UC e Vendas. Cada tabela armazena informações específicas sobre diferentes aspectos da instituição.

Por exemplo, a tabela Estudantes armazena informações sobre os estudantes, como nome, morada, telefone e email. A tabela Funcionarios armazena informações sobre os funcionários, como nome, morada, telefone, email, grau, categoria, ordenado, data de contrato, escalão, etc.

Relativamente ao tópico estatísticas, não foi feita nenhuma tabela pois pela nossa interpretação não achamos uma tabela necessária, uma vez que seria algo que um programa fizesse, ou seja, que fosse buscar os dados necessários num dado intervalo de tempo para calcular as estatísticas pretendidas.

A tabela Livro armazena informações sobre os livros, como título, área, ano de edição, número de edição, ISBN, custo de compra, preço de venda, resumos, número de páginas, formatos, etc. A tabela Materiais armazena informações sobre os materiais, como código do fabricante, estoque, custo de compra, custo de venda, fabricante, categoria, localização, etc.

A tabela Vendas armazena informações sobre as vendas, como ID da venda, ID do funcionário, custo total, etc. A tabela verificaçao_venda armazena informações sobre a verificação de vendas, como ID da venda, ID do material, ID do estudante, ID do docente, produto, data da venda, etc.

Além disso, o sistema também inclui chaves estrangeiras para estabelecer relacionamentos entre as tabelas. Por exemplo, a tabela Apontamentos tem uma chave estrangeira ID_UC que referencia a tabela UC. Isso significa que cada apontamento está associado a uma unidade curricular específica.

Em resumo, este sistema de gerenciamento de banco de dados é projetado para gerenciar todas as operações da instituição educacional, desde o gerenciamento de estudantes e funcionários até o gerenciamento de livros e vendas.

3.2. Regras da Instituição

Descrevem-se sintaticamente as regras da papelaria:

- Um docente pode leccionar zero, uma ou mais disciplinas num ano.
- Todas as disciplinas que são leccionadas têm um docente responsável.
- Uma unidade curricular pode ter zero, um ou mais apontamentos associados.
- Um apontamento tem sempre de ter apenas uma disciplina.
- Cada estudante pode fazer o levantamento de zero, um ou mais apontamentos
- Cada entrega é feita a um estudante
- Um apontamento pode ser entregue várias vezes
- Cada unidade curricular pode ter zero, um ou mais livros relativos a esta.
- Cada livro é relativo apenas a uma unidade curricular.
- Cada livro tem apenas uma editora.
- Uma editora pode ter vários livros.
- Uma editora tem contrato com vários autores.
- Cada autor pode apenas ter contrato com uma editora de cada vez.
- Cada livro é considerado um material para ser vendido.
- Um material pode não ser um livro.
- Os funcionários fazem encomendas aos fornecedores.
- Uma encomenda é feita apenas a um fornecedor.
- Cada fornecedor pode receber zero, uma ou mais encomendas.

- Um funcionário pode fazer várias encomendas.
- Uma encomenda é sempre feita apenas por um funcionário .
- Cada venda da papelaria é feita apenas por um funcionário.
- Um funcionário pode efetuar zero, uma ou mais vendas.
- As vendas são verificadas através da gravação da data em que foram feitas.
- Quando há uma venda ocorre uma verificação do material
- Cada material pode ser verificado várias vezes.
- Cada estudante ou docente pode efetuar zero, uma ou mais compras na secretaria.

4. Modelo Conceptual

4.1. Diagrama Entidade-Relacionamento

Tendo por base a descrição e as regras apresentadas na secção anterior, foi construído o seguinte diagrama conceptual (Figura 4-1):

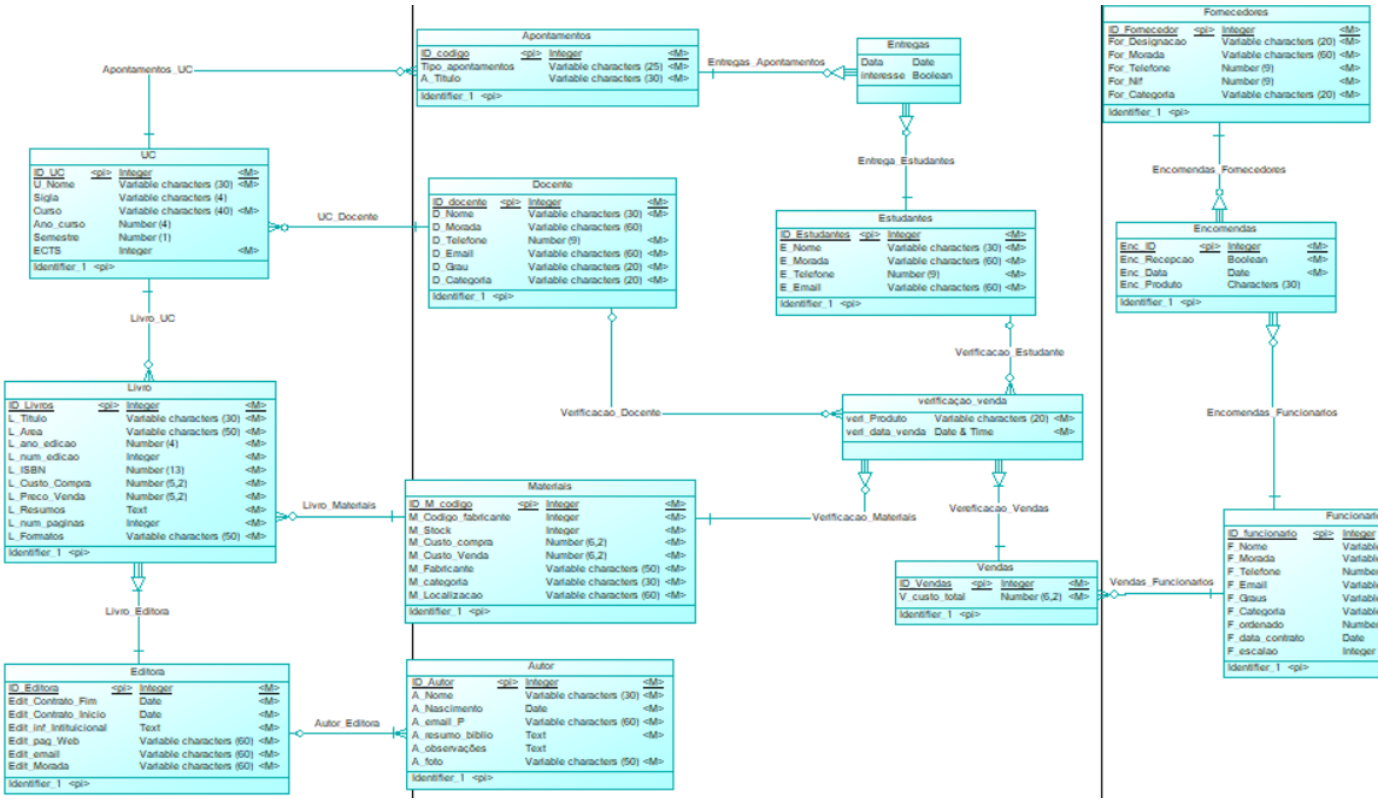


Figura 4-1 Diagrama Entidade-Relacionamento, Modelo Conceptual.

5. Modelo Físico

A partir do diagrama conceptual, estruturado na secção anterior, é gerado o diagrama físico (Figura 5-1), que representa as tabelas e chaves forasteiras que são criadas no Sistema Gestor de Bases de Dados (SGBD), como resultado da aplicação das regras de relacionamentos binários entre entidades.

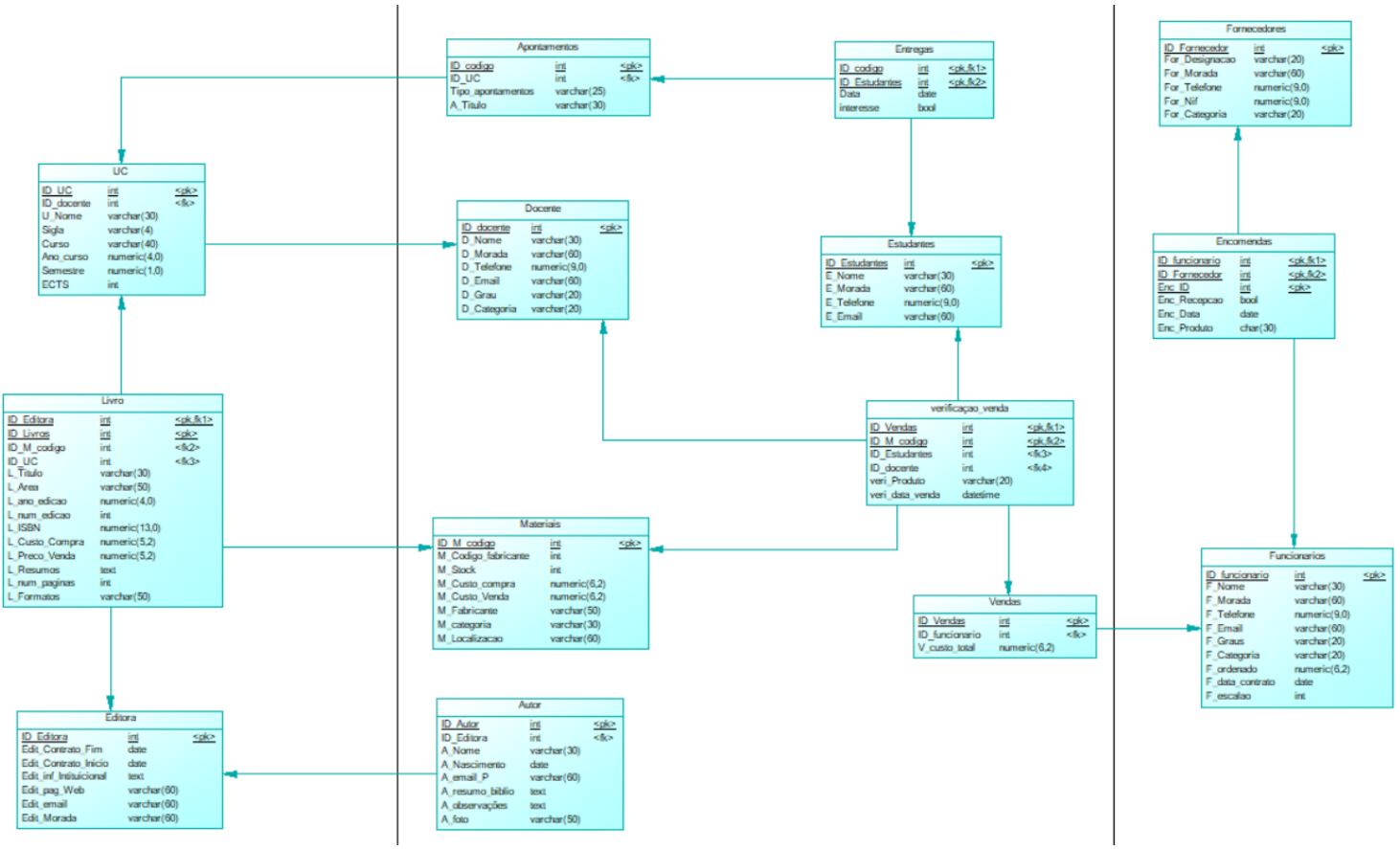


Figura 5-1 Modelo Físico.

Do modelo físico apresentado atrás destacam-se os seguintes aspectos:

- As restrições de chave estrangeira são usadas para manter a integridade referencial entre as tabelas. Por exemplo, a tabela Apontamentos tem uma chave estrangeira que referencia a tabela UC, garantindo que só se possam inserir apontamentos para unidades curriculares existentes.

- Cada campo nas tabelas tem um tipo de dados específico, como int, varchar, date, bool, numeric, e text, que são apropriados para a natureza dos dados que serão armazenados e ainda os limites de tamanho de cada dado para otimizar o seu uso de armazenamento e garantir que estejam alinhados com os requisitos reais dos dados. Além disso, há restrições como not null que garantem que dados essenciais sejam sempre fornecidos.

- Algumas tabelas utilizam o atributo AUTO_INCREMENT para gerar automaticamente um valor único para a chave primária, enquanto outras, como Livro e Encomendas, usam chaves primárias compostas, que combinam múltiplos campos para identificar unicamente um registro.

6. Código de Criação das Tabelas

Concluídos os modelos conceptual e físico, é possível gerar o código SQL que permite a criação da base de dados e respectivas tabelas no SGBD. É importante que o código apresentado seja compatível com o SGBD definido.

```
/*==============================================================*/
/* DBMS name:         MySQL 5.0                                 */
/* Created on:        06/01/2024 23:55:21                       */
/*==============================================================*/


/*criar base de dados*/
DROP DATABASE IF EXISTS trabalho;
CREATE DATABASE trabalho;
USE trabalho;


drop table if exists Apontamentos;


drop table if exists Autor;


drop table if exists Docente;


drop table if exists Editora;


drop table if exists Encomendas;


drop table if exists Entregas;


drop table if exists Estudantes;


drop table if exists Fornecedores;


drop table if exists Funcionarios;
```

```
drop table if exists Livro;

drop table if exists Materiais;

drop table if exists UC;

drop table if exists Vendas;

drop table if exists verificaçao_venda;

/*==============================================================*/
/* Table: Apontamentos                                          */
/*==============================================================*/
create table Apontamentos
(
   ID_codigo          int not null,
   ID_UC              int not null,
   Tipo_apontamentos  varchar(25) not null,
   A_Titulo           varchar(30) not null,
   primary key (ID_codigo)
);

/*==============================================================*/
/* Table: Autor                                                 */
/*==============================================================*/
create table Autor
(
   ID_Autor           int not null,
   ID_Editora         int,
   A_Nome             varchar(30) not null,
```

```
   A_Nascimento       date not null,
   A_email_P          varchar(60) not null,
   A_resumo_biblio    text not null,
   A_observações      text,
   A_foto             varchar(50) not null,
   primary key (ID_Autor)
);

/*==============================================================*/
/* Table: Docente                                               */
/*==============================================================*/
create table Docente
(
   ID_docente         int not null,
   D_Nome             varchar(30) not null,
   D_Morada           varchar(60),
   D_Telefone         numeric(9,0) not null,
   D_Email            varchar(60) not null,
   D_Grau             varchar(20) not null,
   D_Categoria        varchar(20) not null,
   primary key (ID_docente)
);

/*==============================================================*/
/* Table: Editora                                               */
/*==============================================================*/
create table Editora
(
   ID_Editora         int not null,
   Edit_Contrato_Fim  date not null,
   Edit_Contrato_Inicio date not null,
```

```
   Edit_inf_Intituicional text not null,
   Edit_pag_Web       varchar(60) not null,
   Edit_email         varchar(60) not null,
   Edit_Morada        varchar(60) not null,
   primary key (ID_Editora)
);

/*==============================================================*/
/* Table: Encomendas                                            */
/*==============================================================*/
create table Encomendas
(
   ID_funcionario     int not null,
   ID_Fornecedor      int not null,
   Enc_ID             int not null,
   Enc_Recepcao       bool not null,
   Enc_Data           date not null,
   Enc_Produto        char(30),
   primary key (ID_funcionario, ID_Fornecedor, Enc_ID)
);

/*==============================================================*/
/* Table: Entregas                                              */
/*==============================================================*/
create table Entregas
(
   ID_codigo          int not null,
   ID_Estudantes      int not null,
   Data               date,
   interesse          bool,
   primary key (ID_codigo, ID_Estudantes)
```

```
);


/*==============================================================*/
/* Table: Estudantes                                            */
/*==============================================================*/
create table Estudantes
(
   ID_Estudantes      int not null,
   E_Nome             varchar(30) not null,
   E_Morada           varchar(60) not null,
   E_Telefone         numeric(9,0) not null,
   E_Email            varchar(60) not null,
   primary key (ID_Estudantes)
);


/*==============================================================*/
/* Table: Fornecedores                                          */
/*==============================================================*/
create table Fornecedores
(
   ID_Fornecedor      int not null,
   For_Designacao     varchar(20) not null,
   For_Morada         varchar(60) not null,
   For_Telefone       numeric(9,0) not null,
   For_Nif            numeric(9,0) not null,
   For_Categoria      varchar(20) not null,
   primary key (ID_Fornecedor)
);


/*==============================================================*/
/* Table: Funcionarios                                          */
```

```
/*==============================================================*/
create table Funcionarios
(
   ID_funcionario     int not null,
   F_Nome             varchar(30) not null,
   F_Morada           varchar(60) not null,
   F_Telefone         numeric(9,0) not null,
   F_Email            varchar(60) not null,
   F_Graus            varchar(20) not null,
   F_Categoria        varchar(20) not null,
   F_ordenado         numeric(6,2) not null,
   F_data_contrato    date not null,
   F_escalao          int not null,
   primary key (ID_funcionario)
);

/*==============================================================*/
/* Table: Livro                                                 */
/*==============================================================*/
create table Livro
(
   ID_Editora         int not null,
   ID_Livros          int not null,
   ID_M_codigo        int not null,
   ID_UC              int not null,
   L_Titulo           varchar(30) not null,
   L_Area             varchar(50) not null,
   L_ano_edicao       numeric(4,0) not null,
   L_num_edicao       int not null,
   L_ISBN             numeric(13,0) not null,
   L_Custo_Compra     numeric(5,2) not null,
```

```
   L_Preco_Venda      numeric(5,2) not null,
   L_Resumos          text not null,
   L_num_paginas      int not null,
   L_Formatos         varchar(50) not null,
   primary key (ID_Editora, ID_Livros)
);


/*==============================================================*/
/* Table: Materiais                                             */
/*==============================================================*/
create table Materiais
(
   ID_M_codigo        int not null,
   M_Codigo_fabricante  int not null,
   M_Stock            int not null,
   M_Custo_compra     numeric(6,2) not null,
   M_Custo_Venda      numeric(6,2) not null,
   M_Fabricante       varchar(50) not null,
   M_categoria        varchar(30) not null,
   M_Localizacao      varchar(60) not null,
   primary key (ID_M_codigo)
);


/*==============================================================*/
/* Table: UC                                                    */
/*==============================================================*/
create table UC
(
   ID_UC              int not null,
   ID_docente         int not null,
   U_Nome             varchar(30) not null,
```

```
   Sigla              varchar(4),

   Curso              varchar(40) not null,

   Ano_curso          numeric(4,0),

   Semestre           numeric(1,0),

   ECTS               int not null,

   primary key (ID_UC)

);


/*==============================================================*/

/* Table: Vendas                                                */

/*==============================================================*/

create table Vendas

(

   ID_Vendas          int not null,

   ID_funcionario     int not null,

   V_custo_total      numeric(6,2) not null,

   primary key (ID_Vendas)

);


/*==============================================================*/

/* Table: verificaçao_venda                                      */

/*==============================================================*/

create table verificaçao_venda

(

   ID_Vendas          int not null,

   ID_M_codigo        int not null,

   ID_Estudantes      int,

   ID_docente         int,

   veri_Produto       varchar(20) not null,

   veri_data_venda    datetime not null,

   primary key (ID_Vendas, ID_M_codigo)
```

```
);


alter table Apontamentos add constraint FK_Apontamentos_UC foreign key (ID_UC)
	references UC (ID_UC) on delete restrict on update restrict;


alter table Autor add constraint FK_Autor_Editora foreign key (ID_Editora)
	references Editora (ID_Editora) on delete restrict on update restrict;


alter table Encomendas add constraint FK_Encomendas_Fornecedores foreign key
(ID_Fornecedor)
	references Fornecedores (ID_Fornecedor) on delete restrict on update restrict;


alter table Encomendas add constraint FK_Encomendas_Funcionarios foreign key
(ID_funcionario)
	references Funcionarios (ID_funcionario) on delete restrict on update
restrict;


alter table Entregas add constraint FK_Entrega_Estudantes foreign key (ID_Estudantes)
	references Estudantes (ID_Estudantes) on delete restrict on update restrict;


alter table Entregas add constraint FK_Entregas_Apontamentos foreign key (ID_codigo)
	references Apontamentos (ID_codigo) on delete restrict on update restrict;


alter table Livro add constraint FK_Livro_Editora foreign key (ID_Editora)
	references Editora (ID_Editora) on delete restrict on update restrict;


alter table Livro add constraint FK_Livro_Materiais foreign key (ID_M_codigo)
	references Materiais (ID_M_codigo) on delete restrict on update restrict;


alter table Livro add constraint FK_Livro_UC foreign key (ID_UC)
	references UC (ID_UC) on delete restrict on update restrict;


alter table UC add constraint FK_UC_Docente foreign key (ID_docente)
```

```
       references Docente (ID_docente) on delete restrict on update restrict;



alter table Vendas add constraint FK_Vendas_Funcionarios foreign key (ID_funcionario)

       references Funcionarios (ID_funcionario) on delete restrict on update
restrict;



alter table verificaçao_venda add constraint FK_Vereficacao_Vendas foreign key
(ID_Vendas)

       references Vendas (ID_Vendas) on delete restrict on update restrict;



alter table verificaçao_venda add constraint FK_Verificacao_Docente foreign key
(ID_docente)

       references Docente (ID_docente) on delete restrict on update restrict;



alter table verificaçao_venda add constraint FK_Verificacao_Estudante foreign key
(ID_Estudantes)

       references Estudantes (ID_Estudantes) on delete restrict on update restrict;



alter table verificaçao_venda add constraint FK_Verificacao_Materiais foreign key
(ID_M_codigo)

       references Materiais (ID_M_codigo) on delete restrict on update restrict;
```

7. Funcionalidades da Aplicação

Neste capítulo são apresentados os comandos SQL (*Structured Query Language*) que permitem manipular os dados da base de dados.

7.1. Pesquisa de Todas as Notas de um Aluno

Listagem de todos os produtos, indicando o valor total de vendas desse produto e quantas vezes foi vendido:

```
SELECT  veri_Produto,
  SUM(V_custo_total) AS Total_de_Vendas,
  COUNT(*) AS Quantidade_de_Vendas
FROM verificaçao_venda, vendas
GROUP BY  veri_Produto
```

Vamos calcular a soma dos valores na coluna V_custo_total (o valor de da compra) para cada grupo de veri_Produto (os produtos vendidos), para obter o total de vendas (custo total) para cada produto.
Depois conta o número total de linhas para cada grupo de veri_Produto, para obtermos a quantidade total de vendas para cada produto.
Agrupa os resultados pelo campo veri_Produto, ou seja, os cálculos de total de vendas e quantidade de vendas são feitos para cada produto individualmente.

Pesquisa de produtos vendidos dentro de um intervalo de tempo;

```
SELECT veri_Produto
FROM verificaçao_venda
WHERE veri_data_venda BETWEEN 'data_inicial' AND 'data_final'
GROUP BY veri_Produto;
```

Aqui para obtermos o intervalo de tempo usamos a cláusula BETWEEN e substituir 'data_inicial' e 'data_final' pelo intervalo de tempo pretendido.

Pesquisa de produtos (alternativos) da mesma categoria de um determinado produto:

```
SELECT veri_Produto
FROM verificaçao_venda, Materiais
WHERE veri_Produto != 'produto1' AND
 M_categoria = (SELECT M_categoria FROM Materiais WHERE veri_Produto = 'produto1');
```

Neste contexto o objetivo seria recuperar produtos alternativos (diferentes do produto que pretende 'produto1') mas que pertencem à mesma categoria que 'produto1'.
É feito veri_Produto != 'produto1' para garantir que o produto alternativo não seja igual a 'produto1' (o que pretende). Depois é feita uma subconsulta para retornar a categoria do 'produto1'.

Apresentar o total (valor) de vendas por cada mês do presente ano:

```
SELECT YEAR(veri_data_venda) AS Ano, MONTH(veri_data_venda) AS Mes, SUM(V_custo_total)
AS Total_de_Vendas
FROM verificaçao_venda, Vendas
WHERE YEAR(veri_data_venda) = YEAR(CURRENT_DATE())
GROUP BY Ano, Mes
ORDER BY Ano, Mes;
```

Neste comando, YEAR(veri_data_venda) retorna o ano da data de venda e MONTH(veri_data_venda) retorna o mês da data de venda.
SUM(veri_valor_venda) calcula o valor total de vendas para cada mês.
A cláusula WHERE filtra as vendas para apenas o ano atual (CURRENT_DATE()).
A cláusula GROUP BY Agrupa os resultados pelo ano e mês, permitindo calcular o total de vendas para cada mês.
ORDER BY Ano, Mes ordena os resultados pelo ano e mês, garantindo uma apresentação ordenada no resultado final.

Listar a quantidade de vendas por hora dos últimos 5 anos:

```
SELECT DATE_FORMAT(veri_data_venda, '%Y-%m-%d %H') AS Hora,
 COUNT(*) AS Total_de_Vendas
FROM verificaçao_venda
WHERE veri_data_venda >=  NOW() - INTERVAL 5 YEAR
GROUP BY Hora
ORDER BY Hora DESC;
```

Utilizamos a função DATE_FORMAT() para formatar a data e a hora de cada venda, o argumento '%Y-%m-%d %H' especifica que queremos o ano, mês, dia e hora. O resultado é armazenado na nova coluna Hora.

A função COUNT()é utilizada para contar o número de vendas para cada hora e uma vez que estamos a agrupar as vendas por hora (usando GROUP BY Hora), COUNT(*) retornara o número de vendas para cada hora.
Acláusula WHERE limita as vendas consideradas para as últimas 5 anos, ou seja, a expressão NOW() - INTERVAL 5 YEAR é usada para subtrair 5 anos da data e hora atuais.

A função NOW() retorna a data e hora atuais. O operador - é usado para subtrair um intervalo de tempo da data e hora atuais. O comando INTERVAL 5 YEAR especifica um intervalo de 5 anos. Portanto, NOW() - INTERVAL 5 YEAR retorna a data e hora exatas de 5 anos atrás.
Por fim utilizamos a cláusula GROUP BY para agrupa as vendas por hora. Isso significa que para cada hora única, haverá uma linha no resultado com o número de vendas para essa hora.
ORDER BY para ordenar as horas em ordem decrescente o que significa que o resultado mostra primeiro a hora com o maior número de vendas.

Listar o número de vendas realizadas por cada funcionário durante os últimos 2 anos:

```
SELECT F.ID_funcionario,
 (SELECT COUNT(*) FROM Vendas WHERE ID_funcionario = F.ID_funcionario ) AS vendas_efetuadas
FROM Vendas, verificaçao_venda , Funcionarios F
WHERE veri_data_venda >=  NOW() - INTERVAL 2 YEAR
GROUP BY F.ID_funcionario
ORDER BY vendas_efetuadas DESC;
```

Utilizamos uma subconsulta para contar o número de vendas para cada funcionário.
Em seguida, limitamos as vendas para os últimos 2 anos utilizando a mesma forma anterior (NOW() - INTERVAL 2 YEAR). Por fim utilizamos a cláusula GROUP BY para agrupa os resultados pelo ID do funcionário e ORDER BY para ordenar os resultados pelo número de vendas efetuadas por funcionário em ordem decrescente.

Listagem das encomendas de produtos realizadas, mas ainda não recebidas nos últimos dois anos:

```
SELECT *
FROM encomendas
WHERE Enc_Recepcao == FALSE
AND Enc_Data_Entrega >= NOW() - INTERVAL 2 YEAR;
```

O que ele vai fazer é, basicamente selecionar tudo da tabela encomendas e vai mostrar todas as entregas que não foram feitas nos últimos 2 anos.

Identificar para cada cliente o produto mais vendido:

```
SELECT ID_Vendas, veri_Produto,  COUNT(*) AS NumeroDeVendas
FROM
 (SELECT * FROM verificaçao_venda WHERE ID_docente IS NOT NULL AND ID_Estudantes IS NOT NULL ) AS vendas_clientes
GROUP BY ID_Vendas, veri_Produto
ORDER BY NumeroDeVendas DESC;
```

Uma vez que queremos identificar clientes que realizaram compras o seu ID não pode ser nulo, logo é feita uma subconsulta que seleciona apenas as vendas que têm um

docente e um estudante e é feito um COUNT(*) para contar o número de vendas para cada combinação de ID_Vendas e veri_Produto.
E por último agrupamos os resultados pelo ID_Vendas e veri_Produtoe ordenamos por ordem decrescente com base no número de vendas.

Pesquisa dos clientes com mais compras de um determinado produto:

```
SELECT veri_Produto,

 COUNT(*) AS NumerodeVendas

FROM verificaçao_venda

WHERE veri_Produto = 'nome do produto'

GROUP BY veri_Produto

ORDER BY NumerodeVendas DESC;
```

Substituir 'nome do produto' pelo produto que deseja , depois o COUNT(*) conta número de compras do produto. Por fim, agrupamos os resultados pelo nome do produto, de modo que se obtenha um total de compras para cada produto e ordenamos os resultados ordem decrescente pelo número de compras.

Listagem do número total e valor total de vendas por produto e por cliente:

```
SELECT veri_Produto,
        COUNT(*) AS NumeroDeVendas,
        SUM(V_custo_total) AS ValorTotalDeVendas
FROM verificaçao_venda, vendas
GROUP BY veri_Produto
ORDER BY   NumeroDeVendas DESC, ValorTotalDeVendas DESC;
```

SUM(V_custo_total) para somar os valores da coluna V_custo_total, que é o valor total das vendas do produto.
GROUP BY para agrupar os resultados pelo nome do produto, de modo que você obtenha um total de vendas para cada produto.
ORDER BY NumeroTotalDeVendas DESC, ValorTotalDeVendas DESC, para ordenar os resultados pelo número total de vendas e pelo valor total de vendas em ordem decrescente.

Indicação dos clientes mais faltosos (alunos que requereram em diversas ocasiões fotocópias, mas não procederam ao respectivo pagamento):

```
SELECT ID_Estudantes,
 COUNT(*) AS NumeroDeaquisicoes,
 SUM(IF (ID_Vendas IS NULL ,1, 0)) AS NumeroDeFaltasDePagamento
FROM verificaçao_venda
GROUP BY ID_Estudantes
ORDER BY NumeroDeFaltasDePagamento DESC, NumeroDeaquisicoes DESC;
```

Aqui fazemos um SUM(IF(ID_Vendas IS NULL, 1, 0)) que soma o número de vezes que ID_Vendas é NULL para cada ID_Estudantes. Se ID_Vendas for NULL, isso significa que há uma falta de pagamento, então adicionamos 1 à soma. Caso contrário, adicionamos 0. Renomeamos essa coluna como NumeroDeFaltasDePagamento. COUNT(*) conta o número total de aquisiçoes de fotocópias do estudante.
Por fim, agrupamos os resultados pelo ID_Estudante, de modo a que se obtenha um total de aquisições e faltas de pagamento para cada estudante e ordenamos os resultados pelo número de faltas de pagamento e pelo número total de aquisiçoes em ordem decrescente.

8. Gestão da Concorrência: Plano de Transações

Neste capítulo, vamos discutir os planos transacionais necessários para o correto funcionamento do nosso sistema. Vamos abordar situações que podem resultar em acessos concorrentes de escrita aos mesmos registros e discutir como lidar com essas situações.

Para começar devemos ter a noção que uma transação é um conjunto de operações que são executadas como uma única unidade, caso todas as operações de uma transação forem bem-sucedidas, a transação será confirmada (commit), caso contrário se alguma operação falhar, toda a transação será revertida (rollback).

Uma situação que pode resultar em acessos concorrentes de escrita é quando dois funcionários tentam registrar a venda de um livro ao mesmo tempo, neste caso, ambos os funcionários estão a tentar modificar o mesmo registro ao mesmo tempo, como o MySQL usa bloqueios para prevenir os acessos concorrentes que podem levar a inconsistências nos dados, uma das transações será colocada em espera até que a outra termine.

Uma forma de lidar com essa situação seria usar transações. Uma transação é uma série de operações que são executadas como uma única unidade de trabalho. Se qualquer operação dentro da transação falhar, toda a transação é revertida, garantindo a consistência dos dados.

COMMIT;

Insere Livro;

COMMIT;

Por fim, uma situação em que pode ocorrer um rollback é quando uma transação não consegue ser concluída com sucesso, quando uma das operações falhar e todas as alterações feitas na transação serão revertidas. Por exemplo, se uma operação de inserção falha devido a uma violação de chave primária, a transação inteira será revertida para manter a integridade dos dados.

8.1. Transacção de Inserção de Pessoas

Inserir uma nova pessoa na base de dados envolve várias etapas que devem ser todas bem-sucedidas (COMMIT), ou o processo deve ser revertido (ROLLBACK) se alguma etapa falhar.
Primeiro, precisamos inserir os dados das pessoas na tabela Estudantes ou Funcionarios, dependendo se é um estudante ou um funcionário, como estamos a usar MySQL, que suporta a propriedade AUTO_INCREMENT para as chaves primárias, não precisamos de nos preocupar em gerar um ID único para a nova pessoa. O MySQL fará isso automaticamente.
Depois de inserirmos esses dados, precisamos inserir mais informações sobre a pessoa em outras tabelas, como Docente ou Autor. Novamente, você pode confiar no AUTO_INCREMENT para gerar um ID único para a nova entrada.

COMMIT;
INSERT INTO Estudantes (E_Nome, E_Morada, E_Telefone, E_Email)

VALUES ('João Silva', 'Rua das Flores, 123', 91245678, 'joao@gmail.com');

INSERT INTO Docente (ID_docente, D_Nome, D_Morada, D_Telefone, D_Email, D_Grau, D_Categoria)

VALUES (LAST_INSERT_ID(), 'João Silva', 'Rua das Flores, 123', 91345678, 'joao@gmail.com', 'PhD', 'Professor');

COMMIT;

Neste exemplo, LAST_INSERT_ID() é uma função que retorna o último valor gerado para uma coluna AUTO_INCREMENT, usamos isso para obter o ID do estudante recém-inserido e usamo-lo como o ID do docente.
Se algum desses comandos falhar, todo o processo será revertido, e nenhum dado será inserido na base de dados, o que nos garante a integridade dos dados.

9. Estudo da Normalização

9.1. Normalização da Tabela Salas

- O seguinte esquema representa o diagrama de dependências funcionais da tabela produtos(Materiais):

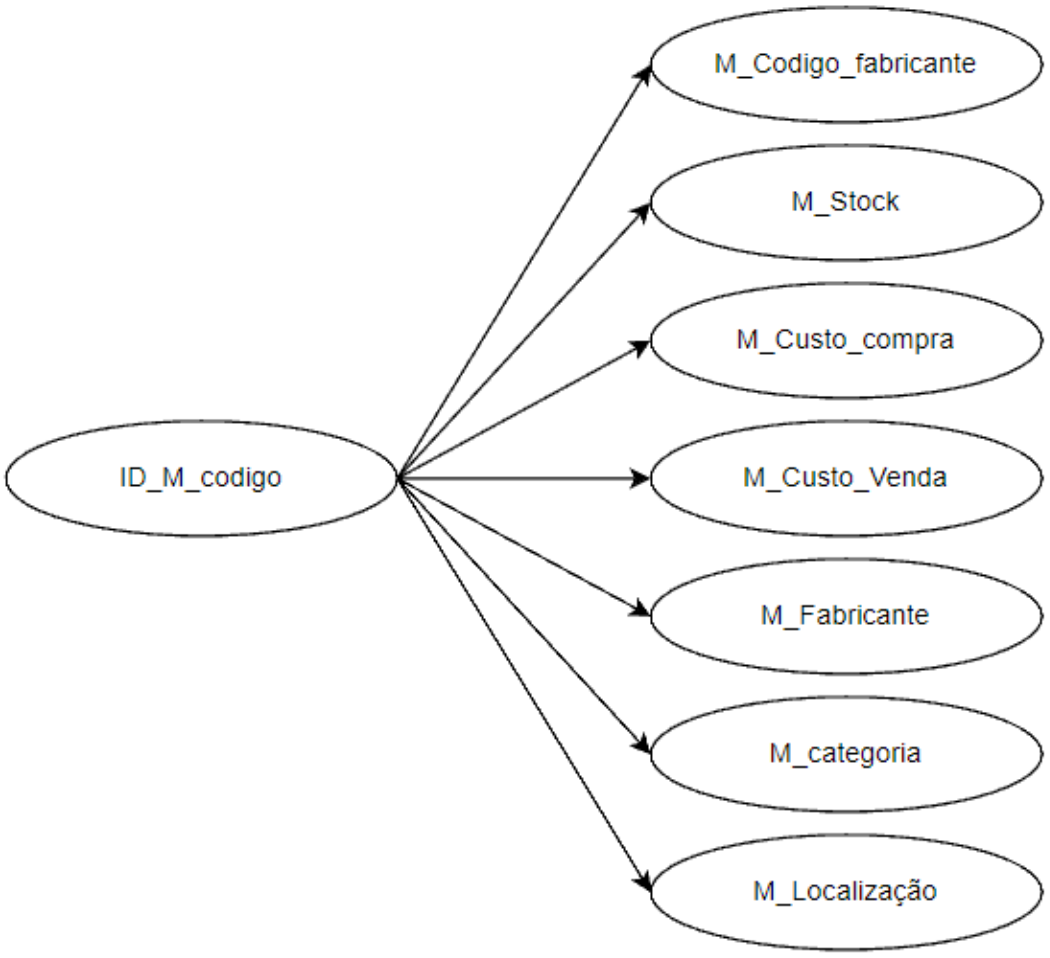


Figura 9-1 Diagrama de dependências da tabela Materiais.

O conjunto de chaves candidatas é constituído unicamente pelo atributo {`ID_M_codigo`}, enquanto o conjunto de chaves determinantes é também constituído apenas pelo atributo {`ID_M_codigo`}.

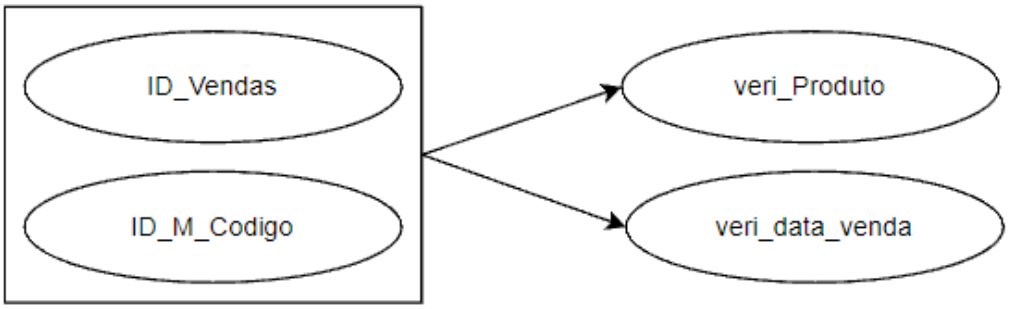
Como as chaves candidatas são iguais às chaves determinantes, significa que a tabela se encontra na **Forma Normal de Boyce-Codd** (FNBC), estando assim normalizada.

- A nossa tabela vendas foi dividida em 2 tabelas, o seguinte esquema representa o diagrama de dependências funcionais da primeira tabela, Vendas:

O conjunto de chaves candidatas é constituído unicamente pelo atributo {ID_Vendas}, enquanto o conjunto de chaves determinantes é também constituído apenas pelo atributo {ID_Vendas}.

Como as chaves candidatas são iguais às chaves determinantes, significa que a tabela se encontra na **Forma Normal de Boyce-Codd** (FNBC), estando assim normalizada.

- O seguinte esquema representa o diagrama de dependências funcionais da segunda tabela, Verificação_vendas:



O conjunto de chaves candidatas é constituído pelos atributos {ID_Vendas, `ID_M_codigo`}, o conjunto de chaves determinantes é também constituído pelos atributos {ID_Vendas, `ID_M_codigo`}.

Como as chaves candidatas são iguais às chaves determinantes, significa novamente que a tabela se encontra na **Forma Normal de Boyce-Codd** (FNBC), estando assim normalizada.

- Relativamente a uma tabela “categorias”, na nossa interpretação do trabalho acabamos por não pensar em “categorias” como uma tabela mas sim como um atributo relativo aos materiais, docentes. Não tendo assim uma tabela para fazer a normalização neste ponto.

10. Conclusões

Este trabalho pôs à prova os nossos conhecimentos de bases de dados e da linguagem SQL para a criação e manipulação de uma base de dados com o fim de ser utilizada na papelaria de uma escola para fazer a gestão de entrega de apontamentos assim como registos de vendas tanto entre os elementos da escola como dos fornecedores de materiais.

As grandes forças deste trabalho incluem a capacidade de SQL para manipular e analisar grandes volumes de dados, bem como a habilidade de formular consultas precisas para obter informações específicas. Também destacamos a eficácia das funções de agregação e das cláusulas GROUP BY, ORDER BY e WHERE para organizar e analisar os dados.

No entanto, também identificamos algumas limitações, tais como, embora o SQL seja uma ferramenta poderosa, pode ser desafiador formular consultas complexas que atendam a todos os requisitos, especialmente quando se trata de lidar com grandes conjuntos de dados. Além disso, a eficácia do SQL depende muito da estrutura adequada de uma base de dados, o que em alguns casos pode ser um desafio.

Em termos pessoais, este trabalho foi uma grande oportunidade para aprender mais sobre SQL e para aplicar em prática tudo o que aprendemos nas aulas de Base De Dados, também foi desafiador navegar por complexidades como por exemplo a otimização de consultas.

Para futuros projetos, poderíamos explorar outras ferramentas de análises e manipulação de dados, tais como python, PHP, etc, que podem vir a oferecer mais opções para manipular e analisar dados. Outra coisa que seria bom pesquisar seria métodos para otimizar a performance do SQL quando estamos a lidar com grandes conjuntos de dados.

11. Referências

stevenbranigan82. (2017) *MySQL (now() - interval 1 month) for this year only*, *Stack Overflow*.
Available at: https://stackoverflow.com/questions/39462114/mysql-now-interval-1-month-for-this-year-only (Accessed: January 2024).

Mike B (2017) *What exactly is interval 1 hour checking?*, *Database Administrators Stack Exchange*.
Available at: https://dba.stackexchange.com/questions/163090/what-exactly-is-interval-1-hour-checking (Accessed: January 2024).

Sr. Enthusiast (2013) - *how to write (current date - 1 year) in teradata? - community*.
Available at: https://support.teradata.com/community?id=community_question&sys_id=4f278f231b97fb00682ca8233a4bcbf9 (Accessed: January 2024).

Crystal (2023) *SQL: Alternative option for using 'Interval "1" year' function*, *Stack Overflow*.
Available at: https://stackoverflow.com/questions/75594585/sql-alternative-option-for-using-interval-1-year-function (Accessed: January 2024).

MySQL 8.0 Reference Manual :: 12.7 date and time functions (no date) *MySQL*.
Available at: https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html (Accessed: January 2024).

Mari Muridi. (2014) *How can I add year, month, day, hour, minute, second to mysql date using date_add?*, *Stack Overflow*.
Available at: https://stackoverflow.com/questions/23999026/how-can-i-add-year-month-day-hour-minute-second-to-mysql-date-using-date-ad (Accessed: January 2024).

W3School(No date) *MySQL date_add() function*.
Available at:
https://www.w3schools.com/sql/func_mysql_date_add.asp (Accessed: January 2024).