FUNDAÇÃO FAT

Modelagem e Desenvolvimento de Banco de Dados

Desenvolvedor de *Salesforce*

Sumário

Apresentação

Sejam bem-vindos a aula de nivelamento de conceitos de Modelagem e Desenvolvimento de Banco de Dados, preparatório para o *Salesforce Platform Developer Credential.* Neste curso, iremos explorar os principais conceitos e técnicas de Modelagem e Desenvolvimento de Banco de Dados.

Os bancos de dados são uma parte fundamental da tecnologia da informação e estão presentes em praticamente todos os sistemas que utilizamos no nosso dia a dia, desde redes sociais e serviços de e-mail até sistemas bancários e aplicativos de smartphones. Este material foi desenvolvido para fornecer uma base sólida em modelagem e desenvolvimento de bancos de dados, mesmo para aqueles que não possuem conhecimento prévio no assunto.

O objetivo principal deste material é ensinar os conceitos básicos de bancos de dados, suas características, como modelá-los e diagramá-los, além de introduzir a linguagem SQL, que é fundamental para a interação com bancos de dados relacionais. Ao final deste curso, você estará apto a criar e manipular bancos de dados simples, entender a importância da modelagem correta e utilizar comandos básicos de SQL.

1. Introdução aos Bancos de Dados

1.1. O que é um Banco de Dados?

Um banco de dados é uma coleção organizada de informações ou dados estruturados, geralmente armazenados eletronicamente em um sistema de computador. Os bancos de dados são gerenciados por sistemas de gerenciamento de bancos de dados (**SGBDs**), que permitem a criação, manutenção e utilização dos dados de maneira eficiente..

1.2. História e Evolução dos Bancos de Dados

Os primeiros sistemas de bancos de dados surgiram na década de 1960, com o modelo hierárquico e o modelo de rede. Nos anos 1970, o modelo relacional foi proposto por Edgar F. Codd, revolucionando a maneira como os dados eram organizados e acessados. Este modelo se tornou a base para muitos dos **SGBDs** modernos, como o **MySQL**, **PostgreSQL** e **Oracle**.

1.3. Importância dos Bancos de Dados no Mundo Atual

Hoje em dia, os bancos de dados são essenciais para praticamente todas as áreas, desde a administração de empresas até a saúde e a educação. Eles permitem o armazenamento seguro e a recuperação rápida de grandes volumes de dados, facilitando a tomada de decisões e a automação de processos.

2. Conceitos Básicos de Bancos de Dados

2.1. Definições e Terminologias

Antes de avançarmos, é importante compreender alguns termos básicos:

- **Dados**: Informações brutas que podem ser processadas para gerar conhecimento.
- **Informação**: Dados processados e organizados de maneira que tenham significado.
- **Tabela**: Estrutura que organiza dados em linhas (registros) e colunas (campos).
- **Registro**: Conjunto de dados relacionados em uma tabela.
- **Campo**: Unidade individual de dados em um registro.

2.2. Características de um Banco de Dados

Os principais aspectos que definem um banco de dados incluem:

- **Persistência**: Dados são armazenados de forma permanente.
- **Confiabilidade**: Integridade e consistência dos dados são mantidas.
- **Segurança**: Controle de acesso e proteção contra perda de dados.
- **Eficiência**: Rápida recuperação e manipulação dos dados.

Tipos de Bancos de Dados

Os bancos de dados podem ser classificados de várias maneiras:

- **Relacionais**: Baseados no modelo relacional, utilizando tabelas (e.g., MySQL, PostgreSQL).
- **Não-Relacionais** (**NoSQL**): Incluem bancos de dados de documentos, chave-valor, colunar e grafos (e.g., MongoDB, Redis, Cassandra).

3. Modelagem de Dados

Modelagem de dados é o processo de criar uma representação visual dos dados e suas relações dentro de um sistema. Isso ajuda a planejar a estrutura do banco de dados de maneira lógica e eficiente.

3.1. Modelagem Conceitual

Na fase de modelagem conceitual, utilizamos diagramas ER (Entidade-Relacionamento) para representar entidades, atributos e relacionamentos. Esta fase é independente de SGBD específico.

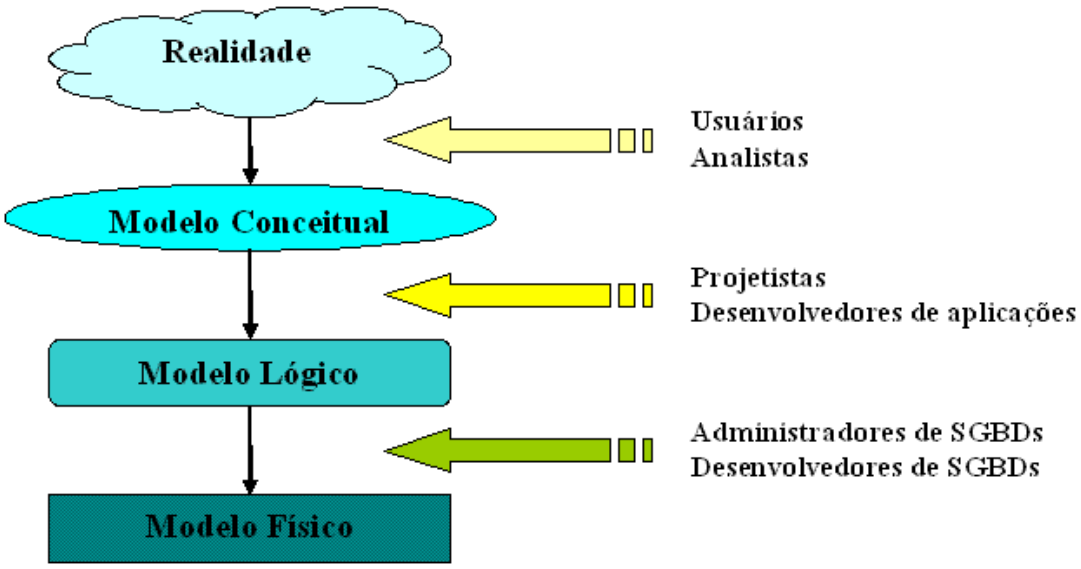
3.2. Modelagem Lógica

A modelagem lógica refina a modelagem conceitual, adicionando detalhes específicos, como chaves primárias e estrangeiras, e normalizando os dados para reduzir redundâncias.

3.3. Modelagem Física

Finalmente, a modelagem física traduz o modelo lógico em um esquema que pode ser implementado em um **SGBD** específico, definindo tipos de dados, índices e outras considerações de performance.

Figura 1- Relação entre os modelos de bancos de dados



3.4. Ferramentas de Modelagem de Dados

A modelagem de dados é um processo essencial no desenvolvimento de bancos de dados, e o uso de ferramentas apropriadas pode facilitar significativamente esse trabalho. As ferramentas de modelagem de dados ajudam a criar, visualizar e manter modelos de dados, garantindo que as estruturas de dados sejam precisas e eficientes. A seguir, apresentamos algumas das principais ferramentas utilizadas na modelagem de dados:

3.4.1. ER/Studio

Descrição: ER/Studio é uma ferramenta robusta de modelagem de dados usada para criar diagramas ER, realizar engenharia reversa em bancos de dados existentes e colaborar em projetos de modelagem.

Recursos: Suporte para múltiplos SGBDs, modelagem conceitual, lógica e física, visualização de relacionamentos e dependências de dados.

Benefícios: Facilita a comunicação entre desenvolvedores e partes interessadas, garante a integridade dos dados e melhora a qualidade do design do banco de dados.

3.4.2. Microsoft Visio

Descrição: Microsoft Visio é uma ferramenta de diagramação amplamente utilizada que também suporta a criação de diagramas ER e outros tipos de diagramas técnicos.

Recursos: Grande variedade de templates e formas, integração com outros produtos Microsoft, capacidade de criar diagramas ER detalhados.

Benefícios: Interface intuitiva e familiar para usuários do Microsoft Office, flexibilidade para criar diversos tipos de diagramas.

3.4.3. Lucidchart

Descrição: Lucidchart é uma ferramenta de diagramação baseada em nuvem que permite a criação colaborativa de diagramas ER e outros tipos de diagramas.

Recursos: Colaboração em tempo real, integração com diversas plataformas (Google Drive, Confluence, etc.), fácil compartilhamento e exportação de diagramas.

Benefícios: Ideal para equipes distribuídas, simplicidade de uso e acessibilidade via navegador web.

3.4.4. MySQL Workbench

Descrição: MySQL Workbench é uma ferramenta de design e administração de banco de dados desenvolvida especificamente para o MySQL.

Recursos: Modelagem de dados, engenharia reversa, geração de scripts SQL, administração de banco de dados.

Benefícios: Integrada ao MySQL, gratuita, interface amigável para modelagem e administração de bancos de dados.

Considerações na Escolha de Ferramentas de Modelagem

Ao escolher uma ferramenta de modelagem de dados, é importante considerar os seguintes fatores:

- **Compatibilidade com SGBDs**: Verifique se a ferramenta suporta o(s) sistema(s) de gerenciamento de banco de dados que você utiliza.
- **Facilidade de Uso:** Avalie a interface e a curva de aprendizado da ferramenta.
- **Recursos e Funcionalidades:** Considere as funcionalidades específicas necessárias para o seu projeto, como engenharia reversa, colaboração em equipe, geração de scripts SQL, etc.
- **Custo:** Compare os custos das ferramentas, levando em conta o orçamento disponível e o valor agregado que a ferramenta proporciona.

4. A Estrutura de um Banco de Dados

Um determinado conjunto de dados é chamado de **tabela**: uma matriz de linhas de colunas. A tabela é um conjunto de dados semelhantes. As **colunas** de uma tabela são chamadas de **campos** e as **linhas** de **registros** ou **tuplas**. Os **registros** representam cada elemento do conjunto representado pela tabela. Os **campos** são as características similares entre os registros e que servem para individualizar cada registro.

Os tipos mais comumente utilizados são os campos de texto, os campos de data e os campos numéricos, sendo necessários, em algumas situações, tipos mais especializados. Vejamos os campos que em geral são utilizados em bancos de dados:

4.1. Campo de texto

Contém uma cadeia de caracteres (**string**) alfanuméricos.

4.2. Campo numérico

Contém número. Este número pode ser um inteiro ou um número real (ponto flutuante). Uma especificação do número real é o campo do tipo moeda (**currency**).

4.3. Campo de data e hora

Armazenam data e hora. Internamente este armazenamento se dá em forma de número inteiro que representa o número de segundos contido entre a data e/ou hora armazenada e uma data/hora padrão (geralmente 00:00:00 de 1900). Este inteiro é conhecido como timestamp (marca de tempo). Alguns **SGBDs** possuem o tipo de campo data e hora e alternativamente o tipo timestamp.

4.4. Campo lógico

Também chamado de campo booleano, é um campo que pode armazenar somente dois tipos de valores: **verdadeiro** ou **falso**.

4.5. Campo memo

Também conhecido como campo de texto longo. É um campo especial que pode conter textos de comprimento variável. Em geral são utilizados para textos de grande extensão.

4.6. Campo binário

É um tipo de campo especial para armazenamento de objetos binários, ou **BLOBS - Binary Large Objects**. O **BLOB** é utilizado para armazenar imagens, sons, vídeos, planilhas, textos formatados, etc.

4.7. Campo autonumerado

É um campo numérico cujo valor é atribuído pelo **SGBD** para garantir que seja único para cada registro. Costuma ser utilizado para identificar registros sendo comum a sua utilização como chave primária.

4.8. Campo calculado

Campo especial criado pelo responsável pelo banco de dados, capaz de efetuar cálculos.

4.9. Chave Primária

A **chave primária** de uma tabela é o campo da tabela (chave primária simples) ou o conjunto de campos (chave primária composta) que identifica univocamente um registro, por isso, os valores neste(s) campo(s) devem ser exclusivos para cada registro. Ela também serve para definir uma ordem padrão para a ordenação de uma tabela.

4.10. Chave Estrangeira

Uma **chave estrangeira** é um campo de uma tabela que recebe o valor de chave primária de outra tabela formando assim uma ligação entre as duas tabelas.

4.11. Índices

Um **índice** é essencialmente uma estrutura de ordenação baseada em um campo escolhido. A definição de índices permite que o **SGBDs** crie pré-ordenações

para os campos escolhidos otimizando o tempo de respostas a consultas feitas requisitando a ordenação por estes campos. A definição de índices traz outro benefício: ela pode acelerar muito a busca de registros específicos no banco de dados quando a chave de busca desejada é o campo sobre o qual foi criado o índice.

4.12. Visão

Uma **visão** (**view**) é um texto com uma consulta em **SQL** que pode ser referenciado como uma fonte de dados por uma outra consulta. São recomendadas para o suporte de consultas simples e emissão de relatórios.

4.13. Procedimento Armazenado

Um **procedimento armazenado** (**stored procedure**) é um dos métodos mais populares de mover o processamento para perto dos dados. Um procedimento armazenado é um lote de comandos, direcionados ao **SGBD**, armazenados dentro do próprio **SGBD**.

4.14. Gatilho

Um **gatilho** (**trigger**) é um procedimento armazenado especial associado a eventos de uma tabela. Ele não pode ser diretamente executado como o procedimento armazenado; ao invés disso, ele é disparado somente em resposta a um evento de inserção, atualização, ou deleção em uma tabela.

4.15. Transação

Uma **transação** é uma sequência de tarefas que juntas constituem uma unidade lógica de trabalho. Todas as tarefas devem ser completadas ou, caso uma delas falhe a transação inteira deve ser abortada. A qualidade de um **SGBD** é mensurada, entre outras características, pela sua aderência às propriedades **ACID** de uma transação.

ACID é um acrônimo para quatro propriedades interdependentes: **atomicidade, consistência, isolamento e durabilidade**. Muito da arquitetura de qualquer **SGBD** relacional atualmente é fundamentada sobre estas propriedades. Bancos de dados estão muito associados à ideia de integridade de transações, tendo

que considerar a questão da concorrência. Muitos usuários requisitando seleções e modificações de dados simultaneamente fazem da concorrência uma questão importante para integridade transacional, que deve ser garantida por um **SGBD**.

Os tipos de dados definem o tipo de valor que uma coluna em uma tabela pode armazenar. Alguns tipos de dados comuns incluem:

- **Inteiros**: Números inteiros (e.g., INT, BIGINT).
- **Decimais**: Números com casas decimais (e.g., DECIMAL, FLOAT).
- **Texto**: Sequências de caracteres (e.g., VARCHAR, TEXT).
- **Datas e Horas**: Datas e horários (e.g., DATE, TIMESTAMP).

Escolha Adequada de Tipos de Dados

Escolher o tipo de dado correto é crucial para a eficiência e integridade do banco de dados. Por exemplo, use **INT** para idades, **VARCHAR** para nomes, e **DATE** para datas de nascimento.

Exemplos Práticos:

- Nome do Cliente: VARCHAR(100)
- Idade do Cliente: INT
- Preço do Produto: DECIMAL(10,2)
- Data de Criação: TIMESTAMP

5. Relacionamento entre Tabelas

5.1. Relacionamento entre Tabelas

Os relacionamentos entre tabelas são fundamentais em bancos de dados relacionais, pois definem como as tabelas se conectam e interagem umas com as outras. Eles são utilizados para assegurar a integridade referencial e para organizar os dados de maneira eficiente. Existem três tipos principais de relacionamentos: **um-para-um, um-para-muitos e muitos-para-muitos**.

5.1.1. Relacionamento Um-para-Um (1:1)

Cada registro em uma tabela está associado a no máximo um registro em outra tabela, e vice-versa.

Exemplo: Uma pessoa e um passaporte, onde cada pessoa tem um único passaporte e cada passaporte pertence a uma única pessoa.

5.1.2. Relacionamento Um-para-Muitos (1:N)

Um registro em uma tabela pode estar associado a muitos registros em outra tabela, mas cada registro na segunda tabela está associado a no máximo um registro na primeira tabela.

Exemplo: Uma empresa e seus empregados, onde uma empresa emprega muitos empregados, mas cada empregado trabalha para uma única empresa.

5.1.3. Relacionamento Muitos-para-Muitos (N:M)

Muitos registros em uma tabela podem estar associados a muitos registros em outra tabela.

Exemplo: Alunos e cursos, onde um aluno pode se inscrever em vários cursos e cada curso pode ter vários alunos inscritos.

5.2. Cardinalidade

A cardinalidade define o número de ocorrências de uma entidade que pode se relacionar com o número de ocorrências de outra entidade. Ela é representada pelo mínimo e máximo número de vezes que uma entidade pode estar associada a outra.

5.2.1. Cardinalidade (0,1)

(0,1) para a Entidade A: Um registro da Entidade A pode estar associado a nenhum ou a um único registro da Entidade B.

Exemplo: Uma pessoa pode ou não possuir um passaporte.

5.2.2. Cardinalidade (1,1)

(1,1) para a Entidade A: Um registro da Entidade A deve estar associado a exatamente um registro da Entidade B.

Exemplo: Cada cidadão deve estar registrado em pelo menos uma cidade.

5.2.3. Cardinalidade (0,*)

(0,*) para a Entidade A: Um registro da Entidade A pode estar associado a nenhum, um ou muitos registros da Entidade B.

Exemplo: Uma loja pode ter zero ou muitos clientes.

5.2.4. Cardinalidade (1,*)

(1,*) para a Entidade A: Um registro da Entidade A deve estar associado a pelo menos um ou muitos registros da Entidade B.

Exemplo: Cada pedido deve ter pelo menos um item, e pode ter muitos.

Compreender os relacionamentos entre tabelas e a cardinalidade é crucial para a modelagem eficaz de dados. Esses conceitos ajudam a definir como as tabelas interagem e garantem a integridade referencial, facilitando consultas eficientes e

precisas no banco de dados. A prática de desenhar e analisar diagramas ER com cardinalidade é uma habilidade essencial para qualquer desenvolvedor ou administrador de banco de dados.

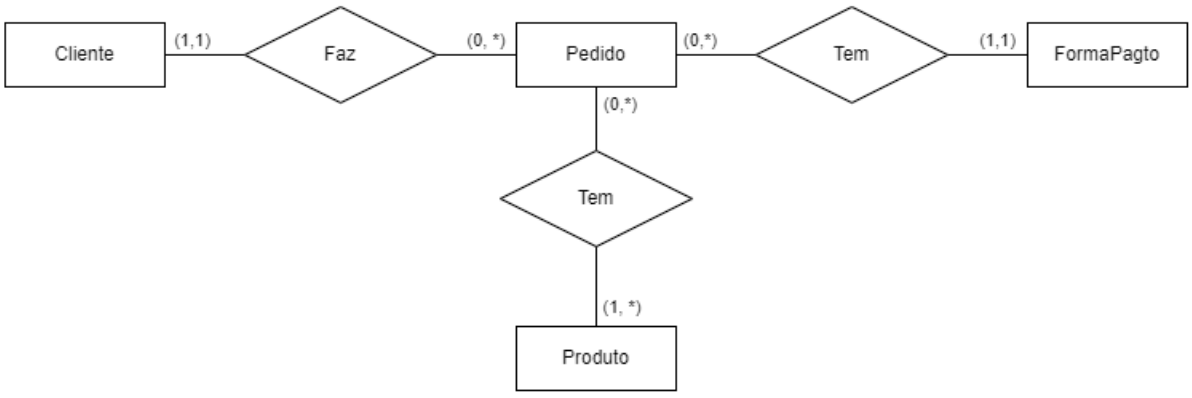
6. Diagramação de Banco de Dados

Os diagramas **ER** (**Entidade-Relacionamento**) são uma ferramenta essencial na modelagem de dados. Eles representam visualmente as entidades, atributos e os relacionamentos entre eles em um sistema de banco de dados.

O Diagrama abaixo exemplica a criação de um sistema simples de controle de pedidos, considerando apenas as entidades e seus relacionamentos. Onde podemos "ler" o diagrama da seguintes forma:

"Um cliente pode fazer nenhum ou muitos pedidos e cada pedido pertence a um único cliente. Cada pedido possui uma forma de pagamento e devem possuir no mínimo um produto. Os produtos pode estar em vários pedidos ou em nenhum.

Figura 2 – DER Conceitual de um Sistema de Controle de Pedidos



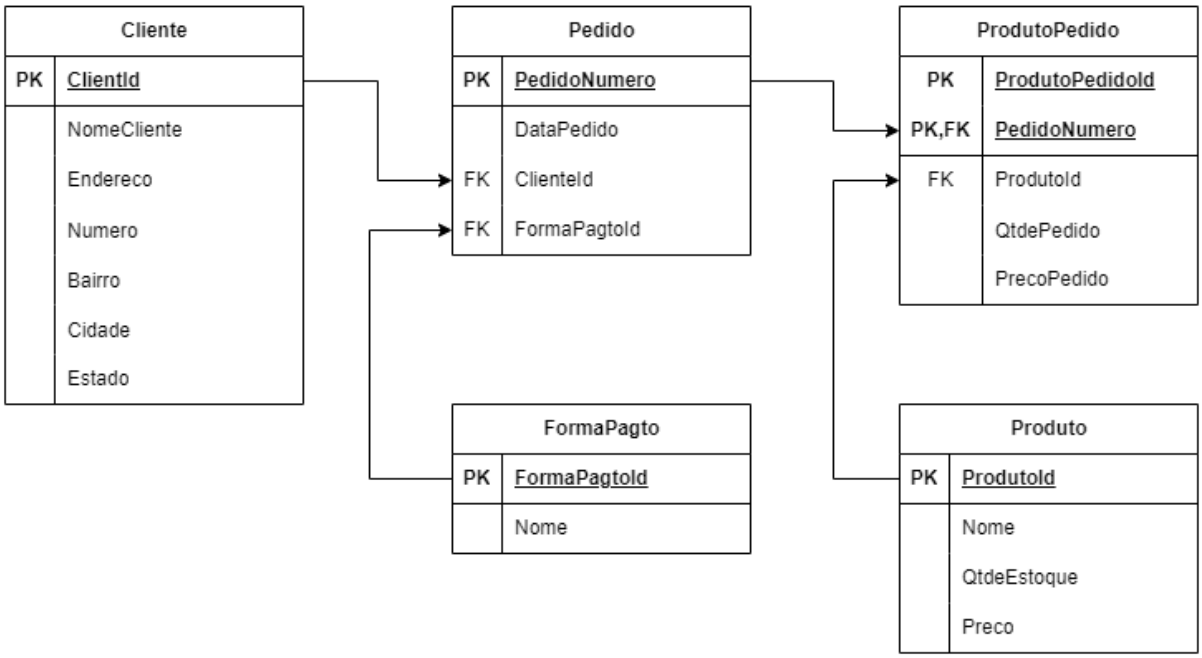
Componentes de um Diagrama ER:

- **Entidades**: Representam objetos ou conceitos que armazenam dados, como "Cliente" ou "Pedido". No diagrama acima, são os retângulos.
- **Atributos**: São as propriedades ou características das entidades, como "Nome" ou "Data de Nascimento". Não constam do diagrama acima.
- **Relacionamentos**: Indicam como as entidades estão conectadas entre si, como "Cliente faz Pedido". No diagrama acima são os losangos.
- **Cardinalidade**: Indicam o grau do relacionamento. No diagrama acima, temos a cardinalidade no formato mínimo e máximo (1,1), (1,*) entre outros.

Como cada pedido deve possuir um ou mais produtos e cada produto pode estar em mais de um pedido, sendo este um relacionamento de **Muitos-para-Muitos,** precisamos então criar uma nova tabela, para armazenar ambas as chaves-estrangeiras, e demais informações necessárias a esse relacionamento.

No diagrama abaixo, foram incluidos os campos de cada tabela, bem como as identificações de **chaves-primárias** (**PK**) e **chaves-estrangeiras** (**FK**).

Figura 3 – DER Lógico de um Sistema de Controle de Pedidos



Vale ressaltar que na entidade **Produto**, temos os atributos, **QtdeEstoque** e **Preco**, esse campos representam os valores atuais de cada produto. Na entidade

ProdutoPedido, que é uma entidade-fraca, resultante do relacionamento de muitos-para-muitos entre **Produto** e **Pedido**, temos **QtdePedido** e **PrecoPedido**, que representam esses mesmos valores, porem de forma temporal, ou seja, no momento em que o pedido foi feito.

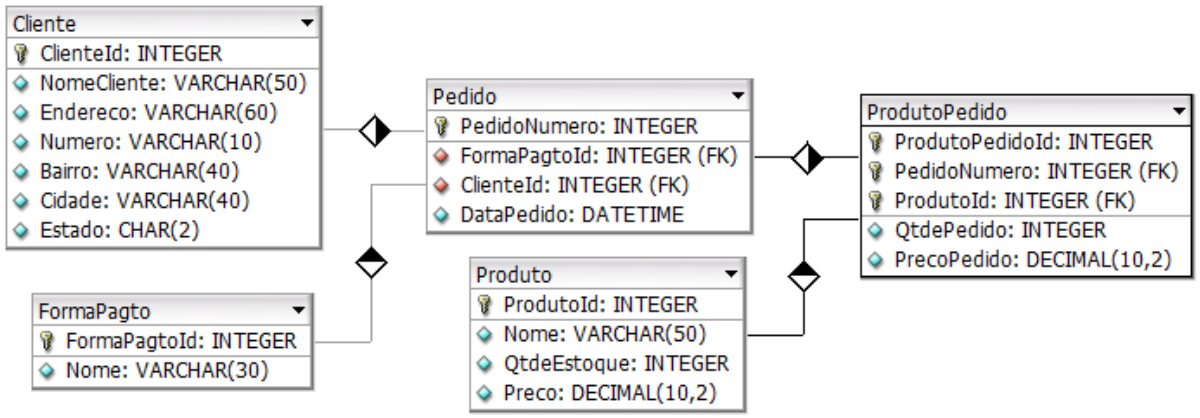
Portanto para criação de um Diagrama ER, devemos seguir os seguintes passos:

1. **Identificar as Entidades**: Determine quais são os principais objetos que serão armazenados no banco de dados.
2. **Definir os Atributos**: Liste as características importantes de cada entidade.
3. **Estabelecer os Relacionamentos**: Desenhe como as entidades interagem e se relacionam entre si.

Os diagramas anteriores foram desenvolvidos utilizando a ferramenta online gratuita do google, **diagrams.net**, que pode ser usada para diagramações simples. Para criações mais profissionais e com possibilidade de exportação e geração de códigos, utilize as ferramentas previamente mencionadas no capitulo 3.4.

Outra ferramenta que pode ser utilizada nesta tarefa é o **DBDesignerFork**, outra ferramenta gratuita, pequena e simples, mas que foi descontinuada e pode apresentar bugs e falhas. O Diagrama abaixo, representando o **Modelo Físico** do exemplo do Sistema de Pedidos, foi desenvolvido nessa ferramenta.

Figura 4 – DER Físico de um Sistema de Controle de Pedidos



7. Regras Formais e Normalização

7.1. O que é Normalização?

Normalização é o processo de organizar os dados em um banco de dados para reduzir a redundância e melhorar a integridade dos dados. A normalização envolve decompor tabelas grandes em tabelas menores e definir relacionamentos entre elas.

Formas Normais (1NF, 2NF, 3NF, BCNF)

- **Primeira Forma Normal (1NF):** Elimina grupos de repetição, garantindo que cada coluna contenha apenas valores atômicos.
- **Segunda Forma Normal (2NF):** Remove dependências parciais de chaves primárias.
- **Terceira Forma Normal (3NF):** Remove dependências transitivas entre colunas.
- **Forma Normal de Boyce-Codd (BCNF):** É uma versão mais rigorosa da 3NF.

Exemplos de Normalização:

- **1NF:** Divida uma tabela de "Pedidos" que tem múltiplos produtos em várias linhas.
- **2NF:** Separe informações do cliente em uma tabela diferente se houver dependência parcial.

- **3NF:** Certifique-se de que os atributos não-chave dependam apenas da chave primária.

8. Introdução ao SQL

8.1. O que é SQL?

SQL (Structured Query Language) é a linguagem padrão utilizada para gerenciar e manipular bancos de dados relacionais. Com **SQL**, você pode criar, ler, atualizar e excluir dados no banco de dados.

8.2. Comandos Básicos de SQL

- **CREATE TABLE:** Cria uma nova tabela no banco de dados.
- **INSERT INTO:** Insere novos dados em uma tabela.
- **SELECT**: Consulta dados de uma tabela.
- **UPDATE**: Atualiza dados existentes em uma tabela.
- **DELETE**: Remove dados de uma tabela.

8.3. Consultas Simples

Criar Tabela

```sql
CREATE TABLE Clientes (
  ID INT PRIMARY KEY,
  Nome VARCHAR(100),
  Idade INT,
  Cidade VARCHAR(50)
);
```

Inserir Dados:

```sql
INSERT INTO Clientes (ID, Nome, Idade, Cidade)
VALUES (1, 'João Silva', 30, 'São Paulo');
```

Selecionar Dados:

```sql
SELECT Nome, Cidade FROM Clientes WHERE Idade > 25;
```

Atualizar Dados:

```sql
UPDATE Clientes SET Cidade = 'Rio de Janeiro' WHERE ID = 1;
```

Deletar Dados:

```sql
DELETE FROM Clientes WHERE ID = 1;
```

Exercícios Práticos

Faça uma analise dos estudos de caso apresentados a seguir e crie diagramas de acordo com a necessidade de cada caso. Em seguida tente criar os bancos de dados utilizando SQL.

Cada exercício pode ter algumas diferenças em sua resolução.

Estudo de Caso: Vicio por TV

Uma pessoa qualquer possui uma assinatura de um pacote de uma empresa de canais televisivos. Esta pessoa possui tantos canais que muitas vezes desconhece, ou não lembra, em que canal é transmitido os programas que deseja assistir. É necessário criar um controle de canais, tipo de programação e dos programas, contendo, nome do programa, descrição, dia da semana, horário, duração, canal e tipo de programa.

Estudo de Caso: Clínica Veterinária

Uma clínica veterinária quer informatizar seus atendimentos, para isso decidiu contratar uma empresa especializada para criar um cadastro de clientes e seus animais. A clínica deseja que cada cliente tenha um código(id), nome, endereço, telefone; cada cliente pode ter um ou mais animais e cada animal pertence a uma raça de uma determinada espécie e possui ainda nome, data de nascimento, sexo, cor, peso e altura. Com base nestas informações desenvolva o diagrama de banco de dados para este projeto.

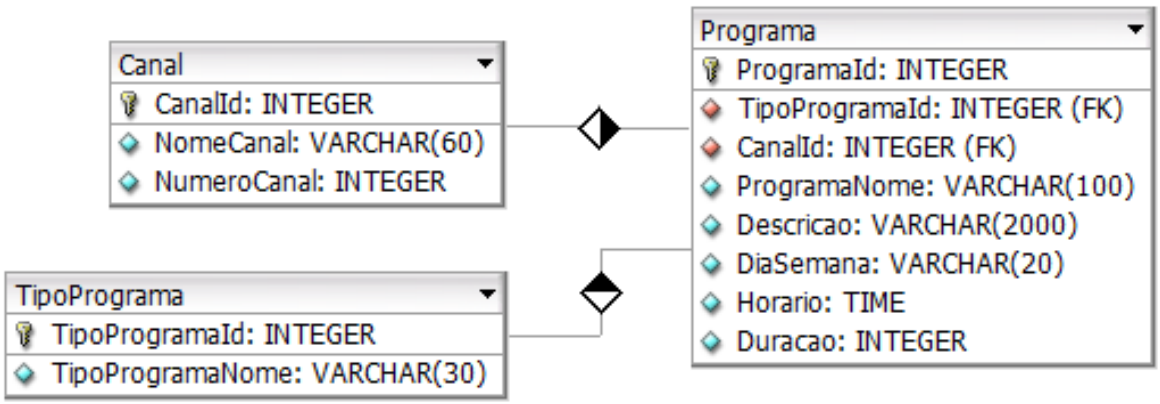
Estudo de Caso: Agência Imobiliária

Nossa empresa organiza aluguéis de imóveis de propriedade de ambos os proprietários privados e empresariais. É atribuído a cada proprietário de imóvel um número único para identificação, o nome do proprietário (nome completo para uma pessoa ou nome de uma empresa), seu endereço (que consiste do nome da rua, número, bairro, CEP, cidade e estado), e-mail e números de telefone e celular. Cada propriedade é identificada por um número único, o número do proprietário (cada propriedade pode possuir apenas um proprietário), o endereço da propriedade e seu tipo (Casa, Apartamento, Loja, Garagem, entre outros).

Locatário é o termo que se refere a uma pessoa ou empresa que assinou um contrato de locação de uma propriedade. Cada contrato de locação é identificado por um número único, registrada a data de assinatura do contrato, a data de início e término do período de locação, a propriedade, o proprietário e o locatário. Cada locatário pode alugar muitas propriedades, ele é identificado por um número único, seu nome, endereço completo, e-mail e números de telefone e celular. Com base na descrição acima, desenvolva o diagrama de banco de dados.

Resolução dos Exercícios Práticos

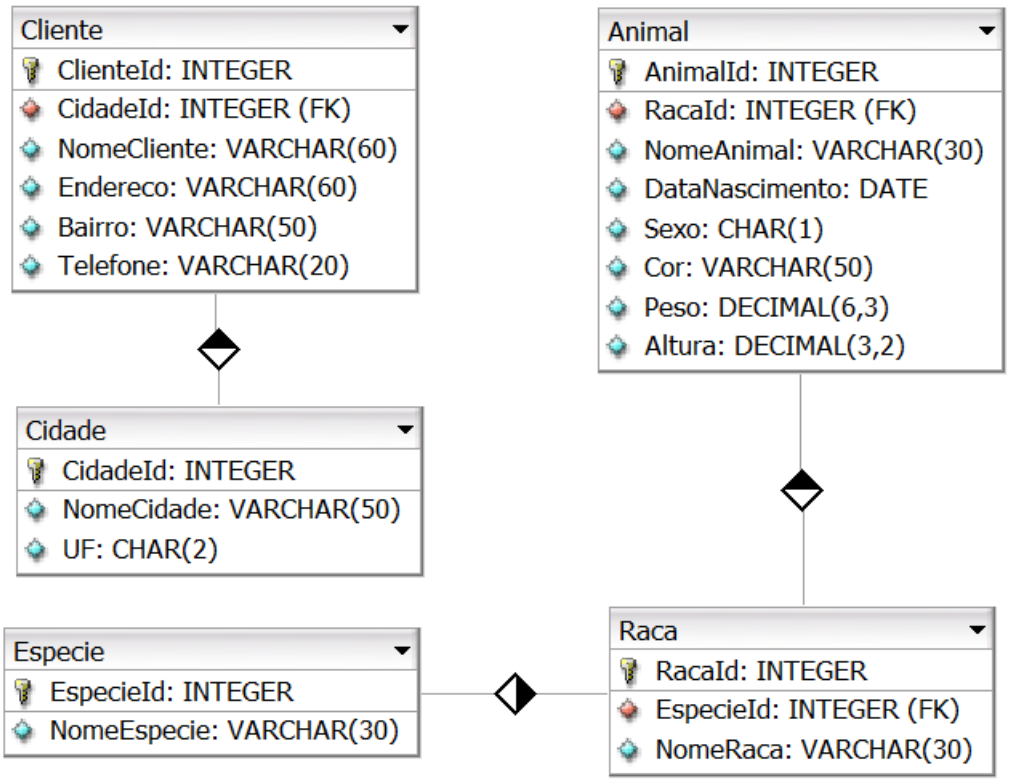
Estudo de Caso: Vicio por TV




```
CREATE TABLE TipoPrograma (
  TipoProgramaId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
  TipoProgramaNome VARCHAR(30) NOT NULL,
  PRIMARY KEY(TipoProgramaId)
);

CREATE TABLE Canal (
  CanalId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  NomeCanal VARCHAR(60) NOT NULL,
  NumeroCanal INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(CanalId)
);

CREATE TABLE Programa (
  ProgramaId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  TipoProgramaId INTEGER UNSIGNED NOT NULL,
  CanalId INTEGER UNSIGNED NOT NULL,
  ProgramaNome VARCHAR(100) NOT NULL,
  Descricao VARCHAR(2000),
  DiaSemana VARCHAR(20) NOT NULL,
  Horario TIME,
  Duracao INTEGER UNSIGNED,
  PRIMARY KEY(ProgramaId),
  FOREIGN KEY(CanalId) REFERENCES Canal(CanalId),
  FOREIGN KEY(TipoProgramaId) REFERENCES TipoPrograma(TipoProgramaId)
);
```

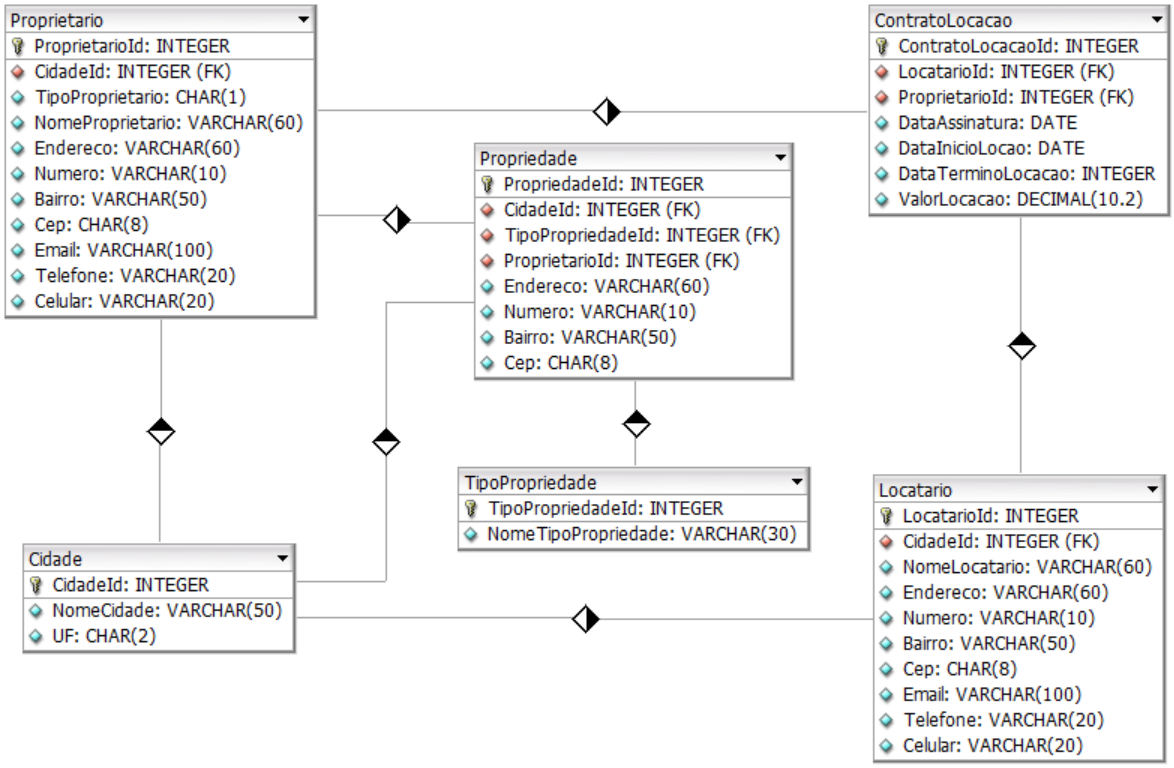
Estudo de Caso: Clínica Veterinária




```
CREATE TABLE Especie (
  EspecieId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  NomeEspecie VARCHAR(30) NOT NULL,
  PRIMARY KEY(EspecieId)
);


CREATE TABLE Cidade (
  CidadeId INTEGER UNSIGNED  NOT NULL AUTO_INCREMENT,
  NomeCidade VARCHAR(50) NOT NULL,
  UF CHAR(2)  NOT NULL,
  PRIMARY KEY(CidadeId)
);


CREATE TABLE Raca (
  RacaId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  EspecieId INTEGER UNSIGNED NOT NULL,
  NomeRaca VARCHAR(30) NULL,
  PRIMARY KEY(RacaId),
  FOREIGN KEY(EspecieId) REFERENCES Especie(EspecieId)
);
```

```
CREATE TABLE Animal (
  AnimalId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  RacaId INTEGER UNSIGNED NOT NULL,
  NomeAnimal VARCHAR(30) NOT NULL,
  DataNascimento DATE NOT NULL,
  Sexo CHAR(1) NOT NULL,
  Cor VARCHAR(50),
  Peso DECIMAL(6,3),
  Altura DECIMAL(3,2),
  PRIMARY KEY(AnimalId),
  FOREIGN KEY(RacaId) REFERENCES Raca(RacaId)
);


CREATE TABLE Cliente (
  ClienteId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  CidadeId INTEGER UNSIGNED NOT NULL,
  NomeCliente VARCHAR(60) NOT NULL,
  Endereco VARCHAR(60),
  Bairro VARCHAR(50),
  Telefone VARCHAR(20) NOT NULL,
  PRIMARY KEY(ClienteId),
  FOREIGN KEY(CidadeId) REFERENCES Cidade(CidadeId)
);
```

Estudo de Caso: Agência Imobiliária




```
CREATE TABLE TipoPropriedade (
  TipoPropriedadeId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  NomeTipoPropriedade VARCHAR(30) NOT NULL,
  PRIMARY KEY(TipoPropriedadeId)
);

CREATE TABLE Cidade (
  CidadeId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  NomeCidade VARCHAR(50) NOT NULL,
  UF CHAR(2)  NOT NULL,
  PRIMARY KEY(CidadeId)
);

CREATE TABLE Proprietario (
  ProprietarioId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  CidadeId INTEGER UNSIGNED NOT NULL,
  TipoProprietario CHAR(1) NOT NULL,
  NomeProprietario VARCHAR(60) NOT NULL,
  Endereco VARCHAR(60) NOT NULL,
  Numero VARCHAR(10) NOT NULL,
  Bairro VARCHAR(50),
  Cep CHAR(8) NOT NULL,
  Email VARCHAR(100) NULL,
  Telefone VARCHAR(20),
  Celular VARCHAR(20) NOT NULL,
  PRIMARY KEY(ProprietarioId),
  FOREIGN KEY(CidadeId) REFERENCES Cidade(CidadeId)
```

```
);
CREATE TABLE Locatario (
  LocatarioId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  CidadeId INTEGER UNSIGNED NOT NULL,
  NomeLocatario VARCHAR(60) NOT NULL,
  Endereco VARCHAR(60),
  Numero VARCHAR(10),
  Bairro VARCHAR(50),
  Cep CHAR(8),
  Email VARCHAR(100),
  Telefone VARCHAR(20),
  Celular VARCHAR(20) NOT NULL,
  PRIMARY KEY(LocatarioId),
  FOREIGN KEY(CidadeId) REFERENCES Cidade(CidadeId)
);

CREATE TABLE Propriedade (
  PropriedadeId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  CidadeId INTEGER UNSIGNED NOT NULL,
  TipoPropriedadeId INTEGER UNSIGNED NOT NULL,
  ProprietarioId INTEGER UNSIGNED NOT NULL,
  Endereco VARCHAR(60) NOT NULL,
  Numero VARCHAR(10) NOT NULL,
  Bairro VARCHAR(50) NOT NULL,
  Cep CHAR(8) NOT NULL,
  PRIMARY KEY(PropriedadeId),
  FOREIGN KEY(ProprietarioId) REFERENCES Proprietario(ProprietarioId),
  FOREIGN KEY(CidadeId) REFERENCES Cidade(CidadeId),
  FOREIGN KEY(TipoPropriedadeId) REFERENCES TipoPropriedade(TipoPropriedadeId)
);


CREATE TABLE ContratoLocacao (
  ContratoLocacaoId INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  LocatarioId INTEGER UNSIGNED NOT NULL ,
  ProprietarioId INTEGER UNSIGNED NOT NULL,
  DataAssinatura DATE NOT NULL,
  DataInicioLocao DATE NOT NULL,
  DataTerminoLocacao INTEGER UNSIGNED NOT NULL,
  ValorLocacao DECIMAL(10.2) NOT NULL,
  PRIMARY KEY(ContratoLocacaoId),
  FOREIGN KEY(ProprietarioId) REFERENCES Proprietario(ProprietarioId),
  FOREIGN KEY(LocatarioId) REFERENCES Locatario(LocatarioId)
);
```