Trabalho Banco de

Dados

Feito por: Bryan Diego;

Everthon Luiz

Requisitos do software

Funcionalidade Principal:

O software deve permitir a usuários publicar e fazer download de arquivos

Processos ou Fluxos de Trabalho:

ao entrar no software, deve ser apresentado os arquivos com maiores quantidades de download e avaliações juntamente c seu tamanho e sua classificação, podendo ser um aplicativo, uma música ou filmes/vídeos, juntamente com a opção de upar um arquivo para o site.

Após escolhido o arquivo desejado a pessoa poderia escolher entre baixar via torrent ou via navegador, se escolhida a opção de baixar por torrent seria instalado um arquivo que após aberto iniciaria o download dentro do aplicativo do torrent, se escolhida a opção de baixar por navegador, seria iniciado o download direto do navegador que está sendo usado.

O software deve apresentar uma aba de pesquisa onde será possível adicionar filtros nos arquivos pesquisados.

Requisitos de Integração:

O software poderia se conectar a aplicativos de vídeo para que as pessoas que vão upar os arquivos possam apresentar uma prévia ou talvez algum tutorial de como utilizar o aplicativo.

Requisitos de Entrada/Saída:

O sistema deve processar do que se trata, seja ele é um apk ou mp4, processar o tamanho do arquivo e exibir o nome dele e se for um filme/vídeo informar a sua duração

Conversa com IA

**Pergunta feita:**

requisitos de um software de downloads, exemplo playstore só que com arquivos do tipo jogos filmes musicas publicado por pessoas

**O ChatGPT disse:**

Um software de downloads que permita a distribuição de jogos, filmes e músicas, similar à Play Store, mas com a diferença de que os arquivos seriam publicados por pessoas, deve atender a vários requisitos técnicos, legais e de segurança. Aqui estão alguns dos principais requisitos:

# 1. Interface de Usuário (UI)

* **Design Intuitivo:** A interface precisa ser simples e fácil de usar, com uma navegação clara e organizada para facilitar o acesso aos diferentes tipos de mídia (jogos, filmes, músicas).
* **Busca e Filtros:** Funcionalidade de pesquisa eficiente, permitindo aos usuários buscar por nome, categoria, gênero, popularidade, avaliações, etc.
* **Tela de Detalhes:** Cada item (jogo, filme, música) deve ter uma página com informações detalhadas, como descrições, avaliações de usuários, trailers (para filmes e jogos), imagens, etc.

# 2. Gerenciamento de Conteúdo

* **Sistema de Upload:** Permitir que os usuários publiquem seus próprios arquivos (jogos, filmes, músicas). Deve haver um processo de envio claro e seguro.
* **Metadados dos Arquivos:** Cada arquivo deve ser associado a metadados como título, descrição, categoria, gênero, e tags para facilitar a organização e pesquisa.
* **Formato e Compressão de Arquivos:** Suporte a diferentes tipos de arquivos, como APK para jogos, MP4 para filmes, MP3 para músicas, etc., com opções para compressão para otimizar o espaço.

# 3. Autenticação e Autorização

* **Cadastro e Login:** O usuário precisa se cadastrar e fazer login para enviar conteúdo ou baixar arquivos. A autenticação pode ser feita via e-mail, redes sociais, ou mesmo autenticação multifatorial.
* **Controle de Acesso:** Permitir que apenas usuários verificados possam publicar conteúdo ou interagir com funcionalidades avançadas.

# 4. Segurança e Proteção de Dados

* **Verificação de Malware:** Implementar uma ferramenta para verificar os arquivos antes de serem disponibilizados para download para garantir que não contenham vírus ou malware.
* **Criptografia:** Proteger a comunicação entre o usuário e o servidor utilizando HTTPS, e criptografar as informações sensíveis (como dados de pagamento, caso haja).
* **Backup de Arquivos:** Manter backups regulares dos arquivos carregados para evitar perda de dados.

# 5. Monetização e Pagamentos

* **Sistema de Pagamento:** Caso o modelo seja pago, deve haver um sistema de pagamento integrado, como cartões de crédito, PayPal, ou criptomoedas.
* **Modelos de Licenciamento:** Permitir diferentes formas de monetização, como compras dentro do aplicativo (in-app purchases), ou download pago com possibilidade de recompensa (como pontos, por exemplo).
* **Sistema de Receita para Criadores:** Um mecanismo de distribuição de receita para os usuários que publicam conteúdo, como um modelo de comissão para cada venda ou download.

# 6. Armazenamento e Rede

* **Armazenamento em Nuvem:** O conteúdo de jogos, filmes e músicas precisará ser armazenado em servidores robustos e seguros, com alta disponibilidade.
* **Redes de Distribuição de Conteúdo (CDNs):** Utilizar CDNs para garantir que os downloads sejam rápidos e eficientes, independentemente da localização do usuário.

# 7. Avaliações e Feedback dos Usuários

* **Sistema de Avaliações:** Permitir que os usuários possam avaliar e comentar sobre os conteúdos. Isso é essencial para a confiança entre usuários e criadores.
* **Relatórios de Problemas:** Implementar um sistema onde os usuários podem relatar problemas com o conteúdo, como erros, links quebrados ou conteúdos ofensivos.

# 8. Conformidade Legal

* **Direitos Autorais:** O software deve ter mecanismos para garantir que o conteúdo compartilhado não infrinja direitos autorais. Pode ser necessário um sistema de moderação para garantir que apenas conteúdos com permissão sejam distribuídos.
* **Termos de Serviço e Política de Privacidade:** Como uma plataforma que lida com conteúdo gerado por usuários, o software deve ter Termos de Serviço claros e uma Política de Privacidade para proteger dados dos usuários e garantir o cumprimento de regulamentações locais (como GDPR, LGPD).

# 9. Escalabilidade e Performance

* **Capacidade de Escala:** O sistema deve ser capaz de lidar com grandes volumes de usuários e conteúdo, especialmente se a plataforma crescer rapidamente.
* **Desempenho do Download:** O tempo de download deve ser otimizado para garantir uma boa experiência para o usuário.

# 10. Suporte Multiplataforma

* **Versão para Desktop e Mobile:** A plataforma deve ser acessível tanto por aplicativos móveis (Android/iOS) quanto por meio de uma versão web ou de desktop.
* **Compatibilidade com Diferentes Sistemas Operacionais:** Garantir que os arquivos sejam compatíveis com uma variedade de dispositivos e sistemas operacionais (Windows, macOS, Linux, Android, iOS).

# 11. Funções Sociais e Interativas

* **Redes Sociais Integradas:** Permitir que os usuários compartilhem suas descobertas e baixem conteúdos nas redes sociais.
* **Playlists e Favoritos:** Oferecer funcionalidades como listas de reprodução para músicas ou filmes, e "favoritar" para conteúdo como jogos.

# 12. Suporte e Ajuda ao Usuário

* **FAQ e Tutoriais:** Oferecer uma seção de perguntas frequentes (FAQ) e tutoriais sobre como usar a plataforma e como publicar conteúdo de forma correta.
* **Suporte ao Cliente:** Um sistema de suporte por chat ou e-mail para resolver problemas técnicos e dúvidas dos usuários.

Esse é um conjunto amplo de requisitos que ajudará a construir uma plataforma de downloads robusta e bem-sucedida, capaz de competir com soluções como a Play Store, mas com foco em conteúdo de usuários.

Conversa 2

**Pergunta feita:**

o que seriam os requisitos de um software

**O ChatGPT disse:**

Os **requisitos de um software** são as especificações detalhadas que definem as funcionalidades, comportamentos, e características de um sistema ou aplicação. Esses requisitos são fundamentais para garantir que o software atenda às necessidades e expectativas do usuário, do negócio e da equipe de desenvolvimento.

Os requisitos podem ser divididos em duas categorias principais: **requisitos funcionais** e **requisitos não funcionais**. Vamos ver o que cada um desses termos significa.

# 1. Requisitos Funcionais (RF)

Estes requisitos descrevem o comportamento do sistema e o que ele deve fazer, ou seja, as funcionalidades específicas que o software deve executar.

* **Funcionalidade Principal**: O software deve permitir que os usuários realizem determinadas tarefas. Por exemplo, em um software de downloads, um requisito funcional seria "O usuário deve poder buscar e baixar filmes".
* **Processos ou Fluxos de Trabalho**: Detalhamento de como o sistema deve responder em determinadas situações. Exemplo: "Quando o usuário clicar no botão de download, o arquivo deve ser iniciado imediatamente".
* **Requisitos de Integração**: Definem como o software deve interagir com outros sistemas ou serviços. Exemplo: "O sistema deve ser capaz de integrar com uma plataforma de pagamento para compras".
* **Requisitos de Entrada/Saída**: Detalham os dados que o sistema deve receber e produzir. Exemplo: "O sistema deve processar um arquivo MP4 e exibir informações como nome, duração e formato".
* **Requisitos de Relatório e Notificação**: Descrição de como o sistema deve gerar relatórios ou enviar notificações aos usuários. Exemplo: "O usuário deve receber uma notificação quando o download estiver concluído".

# 2. Requisitos Não Funcionais (RNF)

Estes requisitos descrevem como o sistema deve se comportar em termos de desempenho, segurança, usabilidade, etc. Eles não se referem a funcionalidades específicas, mas sim às qualidades do sistema.

* **Desempenho e Escalabilidade**: O software deve ser capaz de lidar com uma grande quantidade de usuários ou dados. Exemplo: "O sistema deve ser capaz de suportar 1.000 usuários simultâneos sem queda de performance".
* **Segurança**: Requisitos relacionados à proteção de dados e à privacidade dos usuários. Exemplo: "Os dados dos usuários devem ser criptografados durante a transmissão e armazenamento".
* **Usabilidade**: Refere-se à experiência do usuário, garantindo que o software seja fácil de usar. Exemplo: "A interface deve ser intuitiva e simples de navegar".
* **Compatibilidade**: Define em quais plataformas ou dispositivos o software deve funcionar. Exemplo: "O software deve ser compatível com os sistemas operacionais Windows e macOS".
* **Disponibilidade**: Define os tempos de operação e a confiabilidade do software. Exemplo: "O sistema deve estar disponível 99,9% do tempo, com períodos de manutenção agendados".
* **Manutenibilidade**: Define a facilidade de manutenção do software ao longo do tempo. Exemplo: "O código do software deve ser modular, facilitando a correção de bugs".
* **Portabilidade**: Define a capacidade de o software ser executado em diferentes ambientes ou plataformas. Exemplo: "O software deve ser capaz de rodar em diferentes dispositivos móveis (Android e iOS)".

# 3. Requisitos de Interface

Estes requisitos definem como o software deve se comunicar com o usuário, outros sistemas ou dispositivos. Exemplos incluem:

* **Interface de Usuário (UI)**: O design da interface, a estrutura de navegação, a acessibilidade.
* **Interfaces Externas**: Como o sistema interage com outros aplicativos ou hardware (APIs, sistemas de pagamento, etc.).

# 4. Requisitos de Conformidade Legal e Regulatória

Estes requisitos garantem que o software atenda a normas legais e regulamentações aplicáveis, como proteção de dados, leis de direitos autorais e regulamentações de acessibilidade. Exemplo: "O sistema deve estar em conformidade com a Lei Geral de Proteção de Dados (LGPD)".

**Exemplo Prático de Requisitos de Software:**

*Requisitos Funcionais:*

1. O software deve permitir ao usuário fazer login e criar uma conta.
2. O sistema deve permitir que o usuário pesquise e faça download de conteúdos, como filmes, músicas e jogos.
3. O software deve permitir que o usuário faça compras de conteúdos pagos através de um sistema de pagamento integrado.

*Requisitos Não Funcionais:*

1. O sistema deve suportar até 10.000 usuários simultâneos.
2. O tempo de resposta para qualquer busca de conteúdo não deve exceder 2 segundos.
3. O software deve ser compatível com sistemas operacionais Android, iOS, Windows e macOS.

*Requisitos de Interface:*

1. A interface de usuário deve ser limpa e fácil de usar, com menus claros e navegação intuitiva.
2. O software deve fornecer uma opção de tema escuro.

*Requisitos Legais:*

1. O sistema deve garantir que os dados dos usuários sejam armazenados de forma segura e criptografada, em conformidade com as leis de proteção de dados locais.
2. O sistema deve possuir um mecanismo de controle de copyright para garantir que os usuários não compartilhem conteúdo protegido sem autorização.

# Importância dos Requisitos de Software

Os requisitos de software são fundamentais para:

* **Alinhar expectativas**: Garantir que todos os envolvidos (clientes, desenvolvedores, stakeholders) tenham uma visão clara do que será entregue.
* **Guiar o desenvolvimento**: Servem como base para o planejamento e implementação do software, ajudando a definir o escopo do projeto.
* **Gerenciar mudanças**: Quando mudanças no projeto são solicitadas, os requisitos ajudam a entender o impacto e a viabilidade dessas mudanças.
* **Garantir a qualidade**: Requisitos bem definidos ajudam a garantir que o software seja desenvolvido de acordo com os padrões de qualidade desejados.

Esses requisitos ajudam no planejamento e no controle do projeto, além de fornecer uma base para a validação e verificação do sistema ao longo do seu ciclo de vida.

**MER e DER: Modelagem de Bancos de Dados**

A modelagem de diagramas de bancos de dados MER e DER é fundamental para facilitar a visualização e a padronização de diagramas de caso de uso. um dos primeiros passos a ser executado é o estudo e [levantamento dos requisitos necessários](https://www.devmedia.com.br/tecnicas-para-levantamento-de-requisitos/9151) para a construção do produto final. Durante essa análise, identifica-se as principais partes e objetos envolvidos, suas possíveis ações e responsabilidades, suas características e como elas interagem entre si.

A partir das informações obtidas, pode-se desenvolver um modelo conceitual que será utilizado para orientar o desenvolvimento propriamente dito, fornecendo informações sobre os aspectos relacionados ao domínio do projeto em questão.

Quando se inicia o desenvolvimento de um novo sistema, ou mesmo de uma nova funcionalidade para um sistema existente, um dos primeiros passos a ser executado é o estudo e [**levantamento dos requisitos necessários**](https://www.devmedia.com.br/tecnicas-para-levantamento-de-requisitos/9151) para a construção do produto final. Durante essa análise, identifica-se as principais partes e objetos envolvidos, suas possíveis ações e responsabilidades, suas características e como elas interagem entre si.

A partir das informações obtidas, pode-se desenvolver um modelo conceitual que será utilizado para orientar o desenvolvimento propriamente dito, fornecendo informações sobre os aspectos relacionados ao domínio do projeto em questão.

# Modelo Entidade Relacionamento

**O Modelo Entidade Relacionamento (também chamado Modelo ER, ou simplesmente MER)**, como o nome sugere, é um modelo conceitual utilizado na [**Engenharia de Software**](https://www.devmedia.com.br/principios-da-engenharia-de-software/29630) para descrever os objetos (entidades) envolvidos em um domínio de negócios, com suas características (atributos) e como elas se relacionam entre si (relacionamentos).

Em geral, este modelo representa de forma abstrata a estrutura que possuirá o [**banco de dados**](https://www.devmedia.com.br/cursos/banco-de-dados) da aplicação. Obviamente, o banco de dados poderá conter várias outras entidades, tais como chaves e tabelas intermediárias, que podem só fazer sentido no contexto de [**bases de dados relacionais**.](https://www.devmedia.com.br/curso/curso-modelagem-de-bancos-de-dados-relacionais/409)

**Observação**: Nem sempre criaremos modelos para um sistema completo, pois isso poderia resultar em um modelo muito extenso e difícil de interpretar. Dependendo da magnitude do que estaremos desenvolvendo, podemos criar modelos apenas para uma parte do sistema, um módulo, ou mesmo uma funcionalidade. Imagine, por exemplo, um sistema ERP de grande porte que contemple vendas, finanças, recursos humanos, etc. Várias entidades estão presentes em mais de uma parte do sistema, mas não seria muito interessante, e provavelmente nem mesmo necessário, criar um único modelo para todo o sistema, por isso pode-se dividir a modelagem em várias partes menores.

# Entidades

Os objetos ou partes envolvidas um domínio, também chamados de entidades, podem ser classificados como físicos ou lógicos, de acordo sua existência no mundo real. Entidades físicas: são aquelas realmente tangíveis, existentes e visíveis no mundo real, como um cliente (uma pessoa, uma empresa) ou um produto (um carro, um computador, uma roupa). Já as entidades lógicas são aquelas que existem geralmente em decorrência da interação entre ou com entidades físicas, que fazem sentido dentro de um certo domínio de negócios, mas que no mundo externo/real não são objetos físicos (que ocupam lugar no espaço). São exemplos disso uma venda ou uma classificação de um objeto (modelo, espécie, função de um usuário do sistema).

As entidades são nomeadas com substantivos concretos ou abstratos que representem de forma clara sua função dentro do domínio. Exemplos práticos de entidades comuns em vários sistemas são Cliente, Produto, Venda, Turma, Função, entre outros.

Podemos classificar as entidades segundo o motivo de sua existência:

·**Entidades fortes**: são aquelas cuja existência independe de outras entidades, ou seja, por si só elas já possuem total sentido de existir. Em um sistema de vendas, a entidade produto, por exemplo, independe de quaisquer outras para existir.

·**Entidades fracas**: ao contrário das entidades fortes, as fracas são aquelas que dependem de outras entidades para existirem, pois individualmente elas não fazem sentido. Mantendo o mesmo exemplo, a entidade venda depende da entidade produto, pois uma venda sem itens não tem sentido.

·**Entidades associativas**: esse tipo de entidade surge quando há a necessidade de associar uma entidade a um relacionamento existente. Na modelagem Entidade-Relacionamento não é possível que um relacionamento seja associado a uma entidade, então tornamos esse relacionamento uma entidade associativa, que a partir daí poderá se relacionar com outras entidades. Para melhor compreender esse conceito, tomemos como exemplo uma aplicação de vendas em que existem as entidades Produto e Venda, que se relacionam na forma muitos-para-muitos, uma vez que em uma venda pode haver vários produtos e um produto pode ser vendido várias vezes (no caso, unidades diferentes do mesmo produto). Em determinado momento, a empresa passou a entregar brindes para os clientes que comprassem um determinado produto. A entidade Brinde, então, está relacionada não apenas com a Venda, nem com o Produto, mas sim com o item da venda, ou seja, com o relacionamento entre as duas entidades citadas anteriormente. Como não podemos associar a entidade Brinde com um relacionamento, criamos então a entidade associativa "Item da Venda", que contém os atributos identificadores das entidades Venda e Produto, além de informações como quantidade e número de série, para casos específicos. A partir daí, podemos relacionar o Brinde com o Item da Venda, indicando que aquele prêmio foi dado ao cliente por comprar aquele produto especificamente.

Mais adiante veremos um exemplo prático onde poderemos observar a existência dessas entidades de forma mais clara.

# Relacionamentos

Uma vez que as entidades são identificadas, deve-se então definir como se dá o relacionamento entre elas. De acordo com a quantidade de objetos envolvidos em cada lado do relacionamento, podemos classifica-los de três formas:

* **Relacionamento 1..1 (um para um)**: cada uma das duas entidades envolvidas referenciam obrigatoriamente apenas uma unidade da outra. Por exemplo, em um banco de dados de currículos, cada usuário cadastrado pode possuir apenas um currículo na base, ao mesmo tempo em que cada currículo só pertence a um único usuário cadastrado.
* **Relacionamento 1..n ou 1..\* (um para muitos)**: uma das entidades envolvidas pode referenciar várias unidades da outra, porém, do outro lado cada uma das várias unidades referenciadas só pode estar ligada uma unidade da outra entidade. Por exemplo, em um sistema de plano de saúde, um usuário pode ter vários dependentes, mas cada dependente só pode estar ligado a um usuário principal. Note que temos apenas duas entidades envolvidas: usuário e dependente. O que muda é a quantidade de unidades/exemplares envolvidas de cada lado.
* **Relacionamento n..n ou \*..\* (muitos para muitos)**: neste tipo de relacionamento cada entidade, de ambos os lados, podem referenciar múltiplas unidades da outra. Por exemplo, em um sistema de biblioteca, um título pode ser escrito por vários autores, ao mesmo tempo em que um autor pode escrever vários títulos. Assim, um objeto do tipo autor pode referenciar múltiplos objetos do tipo título, e vice versa.

Os relacionamentos em geral são nomeados com verbos ou expressões que representam a forma como as entidades interagem, ou a ação que uma exerce sobre a outra. Essa nomenclatura pode variar de acordo com a direção em que se lê o relacionamento. Por exemplo: um autor escreve vários livros, enquanto um livro é escrito por vários autores.

# Atributos

Atributos são as características que descrevem cada entidade dentro do domínio.

Por exemplo, um cliente possui nome, endereço e telefone. Durante a análise de requisitos, são identificados os atributos relevantes de cada entidade naquele contexto, de forma a manter o modelo o mais simples possível e consequentemente armazenar apenas as informações que serão úteis futuramente. Uma pessoa possui atributos pessoais como cor dos olhos, altura e peso, mas para um sistema que funcionará em um supermercado, por exemplo, estas informações dificilmente serão relevantes.

Os atributos podem ser classificados quanto à sua função da seguinte forma:

* **Descritivos**: representam característica intrínsecas de uma entidade, tais como nome ou cor.
* **Nominativos**: além de serem também descritivos, estes têm a função de definir e identificar um objeto. Nome, código, número são exemplos de atributos nominativos.
* Referenciais: representam a ligação de uma entidade com outra em um relacionamento. Por exemplo, uma venda possui o CPF do cliente, que a relaciona com a entidade cliente.

Quanto à sua estrutura, podemos ainda classificá-los como:

* **Simples**: um único atributo define uma característica da entidade. Exemplos: nome, peso.
* **Compostos**: para definir uma informação da entidade, são usados vários atributos. Por exemplo, o endereço pode ser composto por rua, número, bairro, etc.

Alguns atributos representam valores únicos que identificam a entidade dentro do domínio e não podem se repetir. Em um cadastro de clientes, por exemplo, esse atributo poderia ser o CPF. A estes chamamos de Chave Primária.

Já os atributos referenciais são chamados de Chave Estrangeira e geralmente estão ligados à chave primária da outra entidade. Estes termos são bastante comuns no contexto de bancos de dados. Mantendo o exemplo anterior, a entidade cliente tem como chave primária seu CPF, assim, a venda possui também um campo “CPF do cliente” que se relaciona com o campo CPF da entidade cliente.

# Diagrama Entidade Relacionamento

Enquanto o MER é um modelo conceitual, o Diagrama Entidade Relacionamento (Diagrama ER ou ainda DER) é a sua representação gráfica e principal ferramenta. Em situações práticas, o diagrama é tido muitas vezes como sinônimo de modelo, uma vez que sem uma forma de visualizar as informações, o modelo pode ficar abstrato demais para auxiliar no desenvolvimento do sistema. Dessa forma, quando se está modelando um domínio, o mais comum é já criar sua representação gráfica, seguindo algumas regras.

O diagrama facilita ainda a comunicação entre os integrantes da equipe, pois oferece uma linguagem comum utilizada tanto pelo analista, responsável por levantar os requisitos, e os desenvolvedores, responsáveis por implementar aquilo que foi modelado.

Em sua notação original, proposta por Peter Chen (idealizador do modelo e do diagrama), as entidades deveriam ser representadas por retângulos, seus atributos por elipses e os relacionamentos por losangos, ligados às entidades por linhas, contendo também sua cardinalidade (1..1, 1..n ou n..n). Porém, notações mais modernas abandonaram o uso de elipses para atributos e passaram a utilizar o formato mais utilizado na UML, em que os atributos já aparecem listados na própria entidade. Essa forma torna o diagrama mais limpo e fácil de ser lido.

Observe na **Figura 1** um exemplo simples de um diagrama para um sistema de imobiliárias.

**Figura 1**. Diagrama Entidade Relacionamento de sistema de imobiliária

No domínio representado pelo diagrama acima temos as seguintes entidades e relacionamentos:

* Proprietário contata Corretor (um proprietário pode contatar vários corretores e um corretor pode ser contatado por vários proprietários).
* Corretor atende Inquilino (um corretor pode atender vários inquilinos e um inquilino pode ser atendido por vários corretores).
* Inquilino aluga Imóvel (um inquilino aluga um imóvel e um imóvel pode ser alugado por vários inquilinos).
* Proprietário possui Imóvel (um proprietário possui vários imóveis e um imóvel pertence a apenas um proprietário).

Uma variante da **Figura 1** pode ser vista na **Figura 2**, onde a cardinalidade do relacionamento é exibida junto do losango.

**Figura 2**. Diagrama de Entidade Relacionamento (variação)

Uma outra variação já mostra a cardinalidade de uma forma mais completa, deixando claro as possibilidades de números de objetos envolvidos em cada relacionamento. Nesse modelo, em cada lado do relacionamento os números aparecem no formato (X,Y) ao invés de um único número como vemos nas figuras anteriores. A **Figura 3** ilustra um exemplo desse tipo.

**Figura 3**. Diagrama Entidade Relacionamento (variação 2)

Neste diagrama, lemos os relacionamentos da seguinte forma:

* 1 ou 1 grupo possui 0 ou muitos produtos. Como de um lado temos “1 ou 1”, isso equivale a apenas “1”, pois não temos várias possibilidades. Já do lado do produto, indicamos que um grupo pode possuir nenhum produto, mas também pode possuir vários.
* 0 ou várias vendas contém 1 ou muitos produtos. Ou seja, um produto pode nunca ser vendido (0 vendas) como também pode ser vendido várias vezes (n vendas). Já uma venda deve conter 1 ou vários produtos, pois uma venda não pode estar vazia (0 produtos).

Os atributos, como já foi dito, podem aparecer no diagrama na forma de elipses ligadas às entidades. Essa foi a notação original proposta, mas como podemos ver na **Figura 4**, ela deixa o diagrama com muitos itens e pode atrapalhar um pouco a organização destes.

**Figura 4**. Atributos apresentados como elipses

Em uma notação mais atual, comumente utilizada na UML, os atributos aparecem listados dentro do próprio retângulo da entidade, enquanto o nome da entidade aparece no topo na forma de título. Na **Figura 5** temos um exemplo.

**Figura 5**. Diagrama com atributos nas entidades

# Ferramentas CASE

Do inglês Computer-Aided Software Engineering, as chamadas [**ferramentas CASE**](https://www.devmedia.com.br/ferramentas-case-e-qualidade-dos-dados-o-paradigma-da-boa-modelagem/6905) são aquelas baseadas em computadores (softwares) utilizadas na Engenharia de Software para auxílio nas atividades desde análise de requisitos até, **modelagem de dados**.

No contexto desse artigo, as **ferramentas CASE** permitem a criação de diagramas de forma simples em um ambiente de fácil utilização e com recursos para incluir as principais regras de composição dos diagramas. Exemplos comuns desse tipo de ferramenta são: [**Star UML**,](http://staruml.io/) [**Astah**](http://astah.net/editions/community) e [**ERwin Data Modeler**.](https://erwin.com/products/data-modeler/) Na **Figura 6** vemos um exemplo de diagrama sendo construído no Astah.

**Figura 6**. Diagrama no Astah Community

Além dessas ferramentas específicas, alguns IDEs (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado) como o Visual Studio e ferramentas de [**gerenciamento de bancos de dados**](https://www.devmedia.com.br/gerenciamento-de-banco-de-dados-analise-comparativa-de-sgbd-s/30788) como [**SQL Server**](https://www.devmedia.com.br/sql-server-management-studio/25504)

[**Management Studio**](https://www.devmedia.com.br/sql-server-management-studio/25504) possuem funcionalidades para criar diagramas facilmente e já gerar o código equivalente (SQL para criação das tabelas, chaves e relacionamentos, por exemplo).

# Exemplo prático

Para fixar tudo que foi visto ao longo deste artigo, vamos agora desenvolver um pequeno exemplo prático em que modelaremos um sistema de bibliotecas, focando especificamente no empréstimo de livros.

Primeiramente precisamos identificar as entidades envolvidas nesse contexto. Sabemos que as entidades físicas existentes são o Usuário da biblioteca e o Livro que será emprestado. Além disso, consideraremos aqui que o livro pertence a uma Sessão, que ajuda na organização das obras do acervo. Em um sistema real pode haver outras informações sobre o livro, mas para esse exemplo a sessão é o bastante. Por fim, temos a entidade lógica Empréstimo, que tanto está relacionada com o usuário, quanto com o livro.

Assim já podemos esboçar nosso primeiro diagrama, simples, contendo as principais entidades e o relacionamento entre elas (**Figura 7**).

**Figura 7**. Primeiro DER de um sistema para biblioteca

Neste primeiro diagrama podemos identificar alguns dos conceitos vistos:

* Entidades fortes: Usuário, Livro e Sessão;
* Entidades fracas: Empréstimo;
* Relacionamentos: um Usuário efetua vários Empréstimos, vários

Empréstimos contêm vários Livros, vários Livros pertencem a uma Sessão.

Agora que visualizamos o domínio no diagrama, podemos adicionar os atributos e outras entidades que se façam necessárias. Assim, passamos à **Figura 8**,

**Figura 8**. DER mais completo do sistema para bibliotecas

Neste ponto cabe fazer algumas observações importantes:

Especificamos os atributos de cada entidade e marcamos algumas elas com um asterisco, indicando que aquela é a chave primária da tabela, ou seja, um atributo único, que nunca poderá se repetir entre as entidades do mesmo tipo. Note que neste momento ainda não é necessário especificar o tipo de cada atributo (texto, número, data, etc.), isso só será necessário mais adiante, quando já estivermos planejando o banco de dados da aplicação.

Surgiu a entidade associativa Livro\_Empréstimo, que representa os livros contidos em um empréstimo (considerando um empréstimo contém vários livros e um livro pode estar contido em vários empréstimos). Esta entidade é composta pelas chaves das duas entidades principais. Se fosse necessário, nesta entidade também poderíamos adicionar informações complementares como quantidade (não se aplica neste caso, mas caberia em um sistema de vendas, por exemplo) e observações sobre o item.

Na entidade associativa, o relacionamento n..n foi dividido em dois relacionamentos do tipo 1..n, agora lidos da seguinte forma: um empréstimo contém vários itens, mas um item só pode estar contido em um único empréstimo (restrito pelas chaves primárias); um livro pode estar contido em vários itens de empréstimo (ser emprestado várias vezes), mas cada item refere-se a um único livro.

O **Modelo Entidade Relacionamento** (e principalmente o diagrama) é uma importante ferramenta durante o desenvolvimento de sistemas, principalmente aqueles mais complexos e difíceis de visualizar sem uma análise mais aprofundada.

A correta modelagem auxilia no correto **desenvolvimento da base de dados** e evita que várias alterações sejam necessárias para corrigir erros de concepção provenientes de falhas durante a análise, ou ainda por problemas de comunicação entre os membros da equipe.

**Documentação de Banco de Dados para Site de Downloads/Uploads**

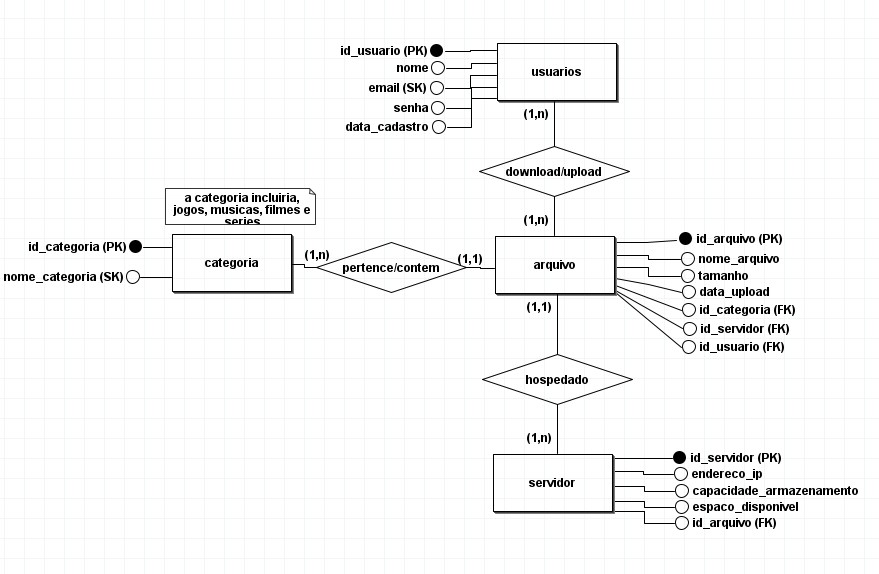
## Visão Geral do Sistema

Este banco de dados foi projetado para um site de compartilhamento de arquivos que permite uploads e downloads de diversos tipos de conteúdo como jogos, músicas, filmes, séries, fotos e outros. O sistema inclui funcionalidades para gerenciamento de usuários, categorização de arquivos e armazenamento em servidores.

## Diagrama Entidade-Relacionamento (DER)

O diagrama apresenta as seguintes entidades e relacionamentos:

1. **usuário** - armazena informações dos usuários do sistema
2. **categoria** - classifica os arquivos por tipo (jogos, músicas, filmes, etc.)
3. **arquivo** - registra todos os arquivos enviados ao sistema
4. **servidor** - mantém informações sobre os servidores de armazenamento



**Dicionário de Dados**

**Tabela: usuario**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Obrigatório** | **Chave** | **Descrição** |
| id\_usuario | INT | - | Sim | PK | Identificador único do usuário |
| nome | VARCHAR | 100 | Sim | - | Nome completo do usuário |
| email | VARCHAR | 100 | Sim | SK | E-mail do usuário (único) |
| senha | VARCHAR | 255 | Sim | - | Senha criptografada do usuário |
| data\_cada  stro | DATETIME | - | Sim | - | Data e hora do cadastro do usuário |

**Tabela: categoria**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Obrigatório** | **Chave** | **Descrição** |
| id\_categoria | INT | - | Sim | PK | Identificador único da categoria |
| nome\_cate  goria | VARCHAR | 50 | Sim | SK | Nome da categoria (ex: Jogos, Músicas) |

**Tabela: arquivo**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Obrigatório** | **Chave** | **Descrição** |
| id\_arquivo | INT | - | Sim | PK | Identificador único do arquivo |
| nome\_arq  uivo | VARCHAR | 255 | Sim | - | Nome original do arquivo |
| tamanho | BIGINT | - | Sim | - | Tamanho do arquivo em bytes |
| data\_uplo ad | DATETIME | - | Sim | - | Data e hora do upload |
| id\_categor  ia | INT | - | Sim | FK | Categoria do arquivo |
| id\_servido  r | INT | - | Sim | FK | Servidor onde o arquivo está armazenado |
| id\_usuario | INT | - | Sim | FK | Usuário que fez o upload |

**Tabela: servidor**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Campo** | **Tipo** | **Tamanho** | **Obrigatório** | **Chave** | **Descrição** |
| id\_servidor | INT | - | Sim | PK | Identificador único do servidor |
| endereco\_ip | VARCH  AR | 15 | Sim | - | Endereço IP do servidor |
| capacidade\_armaze namento | BIGINT | - | Sim | - | Capacidade total em bytes |
| espaco\_disponivel | BIGINT | - | Sim | - | Espaço disponível em bytes |

## Relacionamentos

### 1. usuario - arquivo (1:N)

1. Um usuário pode fazer upload de vários arquivos
2. Cada arquivo pertence a apenas um usuário

### 2. categoria - arquivo (1:N)

1. Uma categoria pode conter vários arquivos
2. Cada arquivo pertence a apenas uma categoria

### 3. servidor - arquivo (1:N)

1. Um servidor pode armazenar vários arquivos
2. Cada arquivo está armazenado em apenas um servidor

**Considerações Finais**

Este modelo de banco de dados permite:

* Cadastro e autenticação de usuários
* Organização de arquivos por categorias
* Controle de uploads e downloads
* Gerenciamento de capacidade dos servidores de armazenamento

Para melhorar o sistema, poderiam ser adicionados:

* Tabela para logs de downloads
* Sistema de permissões para usuários
* Histórico de alterações de arquivos
* Sistema de avaliação/comentários para arquivos