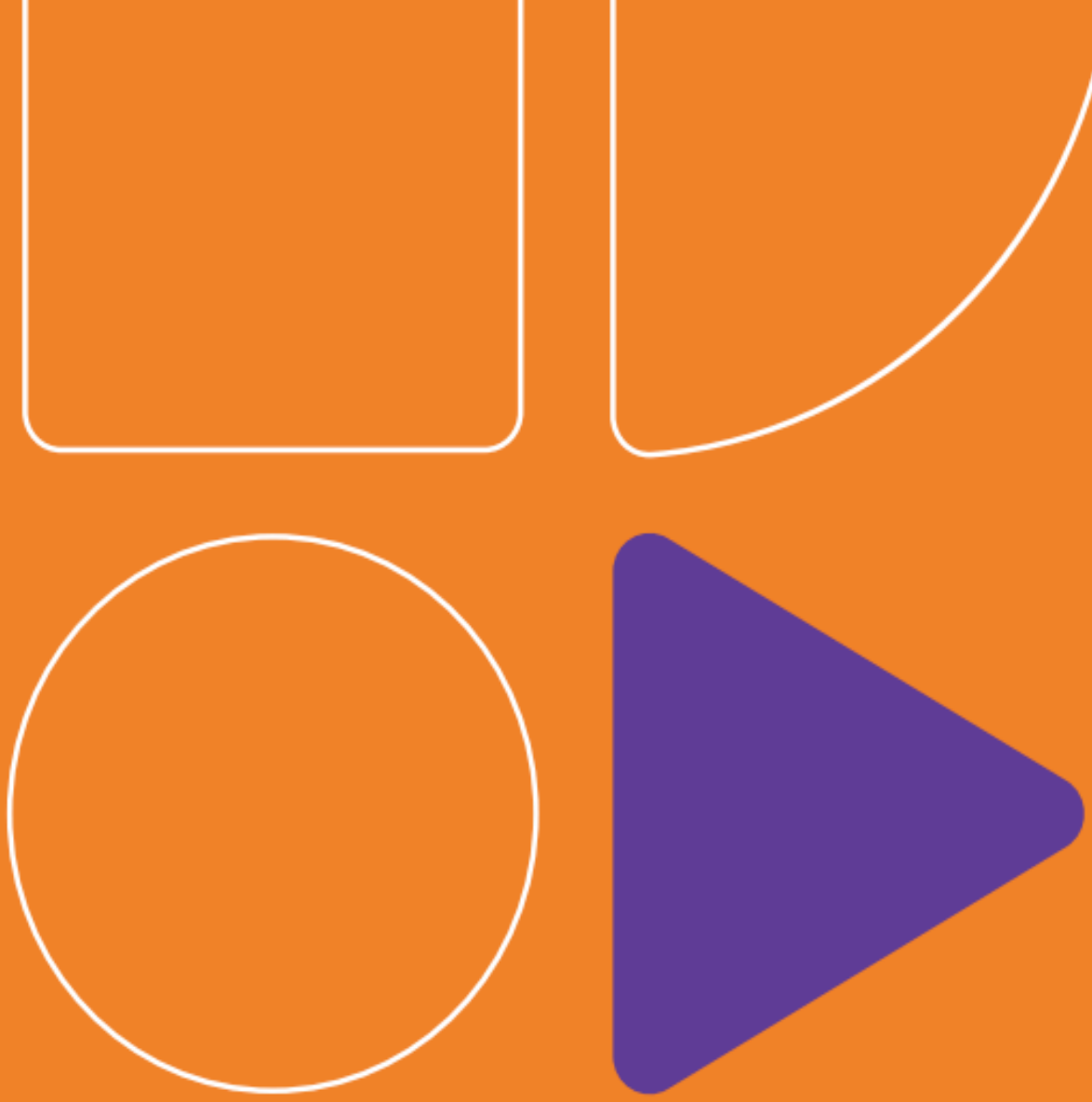


Pratique, aprenda, conquiste.

Proz
Viva sua profissão!

Disciplina | Bancos de Dados

Téc. Desenvolvimento de Sistemas

Aqui começa a sua jornada

Vamos nessa?

Disciplina | Bancos de Dados

Téc. Desenvolvimento de Sistemas

SUMÁRIO

BANCO DE DADOS **5**
INTRODUÇÃO **5**
TEMA 01 ..... **6**
Introdução À Banco De Dados 6
TEMA 02 ..... **19**
Modelagem De Bancos De Dados 19
TEMA 03 ..... **34**
Regras de Normalização 34
TEMA 04 ..... **39**
Introdução À Linguagem Sql - (Ddl) 39
TEMA 05 ..... **50**
DML , manipulando dados com SQL 50
TEMA 06 ..... **58**
Introdução À Linguagem Sql - (Ddl) 58
TEMA 07 ..... **68**
Conectividade Em Python 68

BANCO DE DADOS

INTRODUÇÃO

Um banco de dados é uma coleção organizada de informações geralmente armazenadas eletronicamente em um computador. Projetado para armazenar, gerenciar e recuperar dados de forma eficiente e segura.

Na década de 70, o conceito de banco de dados foi revolucionário e trouxe uma abordagem mais estruturada para o armazenamento e gerenciamento de informações. O modelo predominante na época era o modelo relacional, proposto por Edgar F. Codd em 1970. O modelo relacional baseia-se em tabelas, onde os dados são organizados em linhas (registros) e colunas (campos). Esse modelo permitiu uma forma padronizada de armazenar e recuperar dados, além de fornecer mecanismos para garantir a integridade e consistência dos dados.

Atualmente, o conceito de banco de dados evoluiu e se tornou mais abrangente, com o avanço da tecnologia e o crescimento exponencial dos dados, surgiram sistemas de banco de dados mais escaláveis e distribuídos, como bancos de dados NoSQL.

O objetivo desta disciplina é fornecer competência e capacidade ao aluno de desenvolver modelo de banco de dados.com eficiência, em sua atuação no mercado.

TEMA 01

Introdução À Banco De Dados

habilidades

- Conhecer os conceitos e tipos de banco de dados atualmente no mercado.
- Conhecer e começar a compreender com maiores detalhes a estrutura lógica do SGBD.(Schema,Tabela,Campo e Registro).
- Lidar com gerenciador de alto-nível de banco de dados (Workbenchs).
- Conhecer os motivos técnicos e as vantagens no uso do SGBD.

Iremos trabalhar com banco de dados que obedecem ao modelo relacional e utilizam a linguagem SQL, este padrão foi criado por **Edgar Frank Codd** , entre 1960 - 70, ao propor o modelo relacional ele realizou a separação entre estrutura lógica manipulada pelos programadores e usuários, da forma de armazenamento físico que ficou a cargo dos sistemas criados,

De maneira geral, podemos concluir que, quando criamos variáveis, em geral fazemos acesso direto a memória do computador, e a necessidade de manter estes dados (Persistência), e principalmente destes dados serem confiáveis vem crescendo desde a época de Edgar.

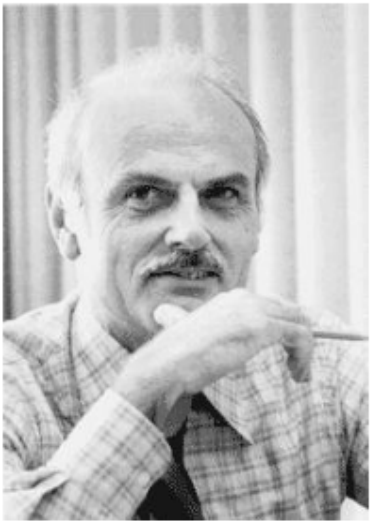


Figura 1: Edgar Frank Codd,
o pai do modelo relacional.

Disponível em <en.wikipedia.org/wiki/Edgar_F._Codd>. Acesso em 20 jul. 2023.

A linguagem SQL surgiu então da necessidade de agilizar comandos e padronizar o acesso para facilitar aos programadores o acesso ao SGBD.

Com a expansão da internet na década de 90, ocorreu a natural evolução de vários sistemas,

sendo que um que se destacou e que é usado devido a sua flexibilidade e gratuidade é o Mysql, por ser compacto e geralmente suportado por vários provedores na internet, o Mysql e alguns outros sistemas Open-source ganharam espaço.

Em algumas situações, devido ao tamanho(escala) da massa de dados, sempre é estudado a eficiência do banco de dados e empresas como Microsoft e Oracle figuram como fornecedores de versões mais robustas de SGBD´s cada um com sua pecularied ade. Lembrando que no dia 16 de Janeiro de 2008, a MySQL AB, desenvolvedora do MySQL foi adquirida pela Sun Microsystems, por US$ 1 bilhão, um preço jamais visto no setor de licenças livres. No dia 20 de Abril de 2009, foi anunciado que a Oracle compraria a Sun Microsystem. Quando a companhia gigantesca como a ORACLE comprou o MYSQL como tática e mercado ainda mantêm versões gratuitas para o uso dos clientes, porém o código do Mysql não é mais livre, diante disto surgiu uma versão baseada em Mysql chamada MariaDB que está em 8º no ranking.

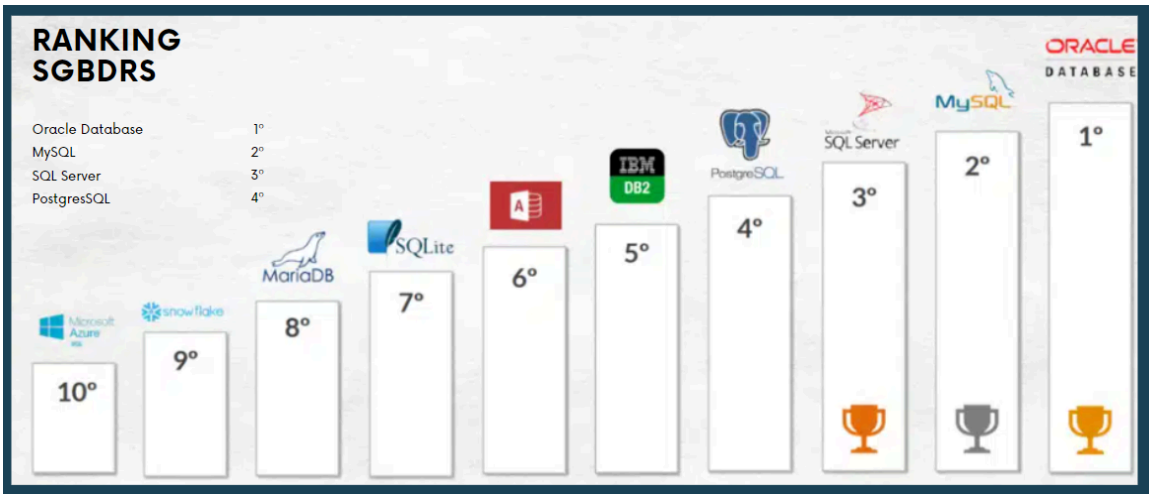


Figura 2: Disponível em <Ranking SGBDR´s https://db-engines.com/en/ranking/relational+dbms>. Acesso em 20 jul. 2023.

Portanto, obviamente verificamos que atualmente a ORACLE® detém a liderança no mercado de Banco de Dados Relacionais, agora vamos a prática iremos usar o MySQL.

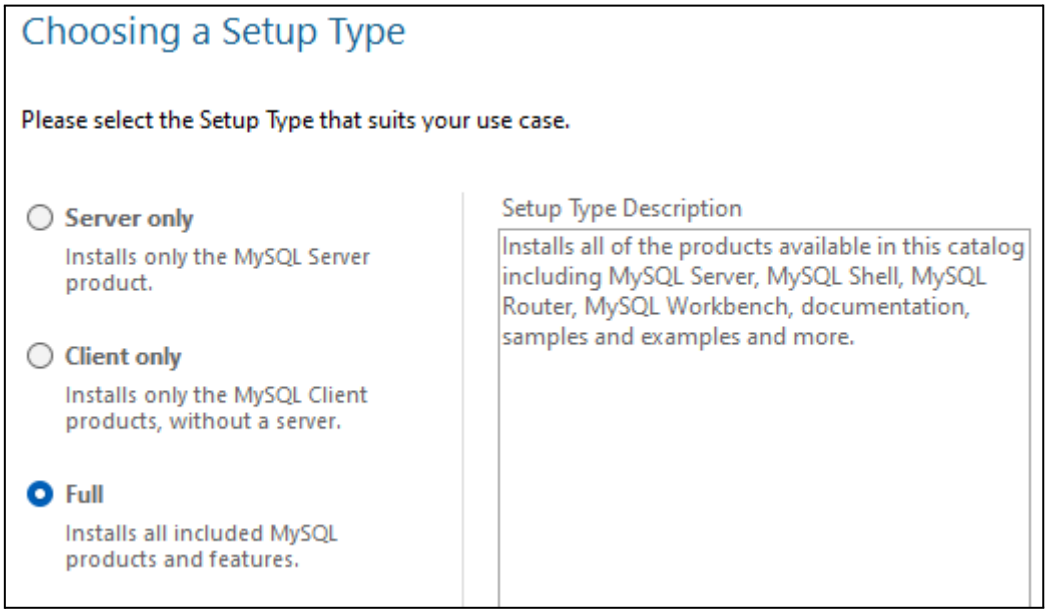
Preparando o ambiente para uso do Mysql no Windows.

Sempre é importante dominarmos os detalhes da ferramenta que estamos usando, portanto, mesmo se o computador que você vai usar já possuir o Mysql instalado, ou, em alguns casos podemos acessar um servidor externo(Nuvem-Cloud) do Mysql, consideramos muito importante que o aluno saiba a sequência de instalação e o comportamento de um SERVIDOR DE BANCO DE DADOS.

Tratando-se de Mysql, vamos trabalhar inicialmente com a ferramenta Mysql Workbench, a versão Workbench, como a própria tradução da palavra do inglês denuncia é a nossa ferramenta de bancada, porém antes de mais nada temos que instalar o servidor Mysql.

Recomendamos para instalação pelo instalador : https://dev.mysql.com/downloads/installer/

O instalador permite que você verifique o que está instalado em sua máquina e você pode escolher o que você deseja instalar. Ao clicar no botão ADD, na tela a seguir escolhemos a opção **FULL,** para garantir a instalação do Servidor e Workbench além de outras ferramentas e exemplos.



Fonte: Elaborado pelo autor <execução de instalação> (2023).

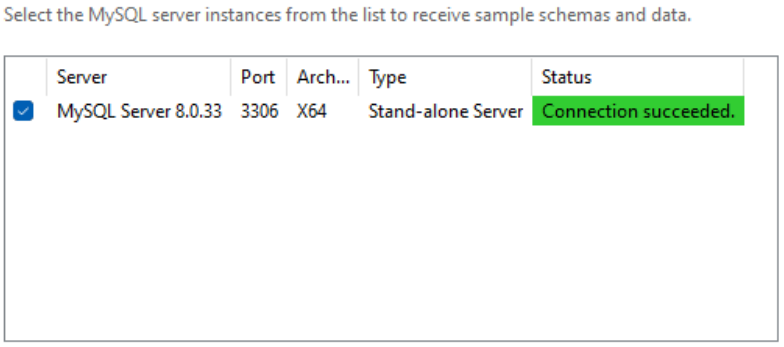
Recomendamos que use sempre o instalador para instalar e remover os componentes de forma mais assertiva, muitas vezes você acaba instalando o workbench e ele pode não encontrar os arquivos de servidor, além disso usando o instalador você pode atualizar componentes caso for necessário e ainda recebe arquivos de exemplos que podem ser úteis aos seus estudos.

Senha, por questões de praticidade podemos usar a senha : 1234 como senha padrão para acessar o Servidor Mysql ela será solicitada pelo instalador.

Ao final da instalação você poderá testar se o seu servidor está funcional.

Lembre-se de utilizar uma senha de fácil memorização caso decida mudar a senha.

Repare que ele informa que o serviço de banco de dados (servidor) estará disponível através da porta 3306, caso houver problemas com antivírus ou acesso a portas, devemos lembrar de qual porta precisa estar liberada.

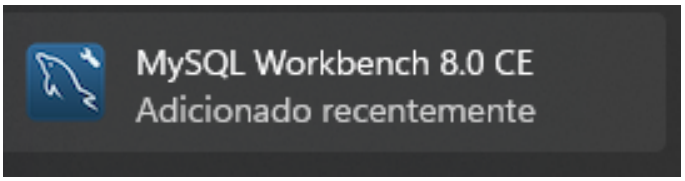


Fonte: Elaborado pelo autor <execução de instalação> (2023).

Utilizando o Mysql Workbench.

Vamos partir para um exemplo bem prático, abrindo o Workbench e seguindo os passos:

1. Execute o workbench.;



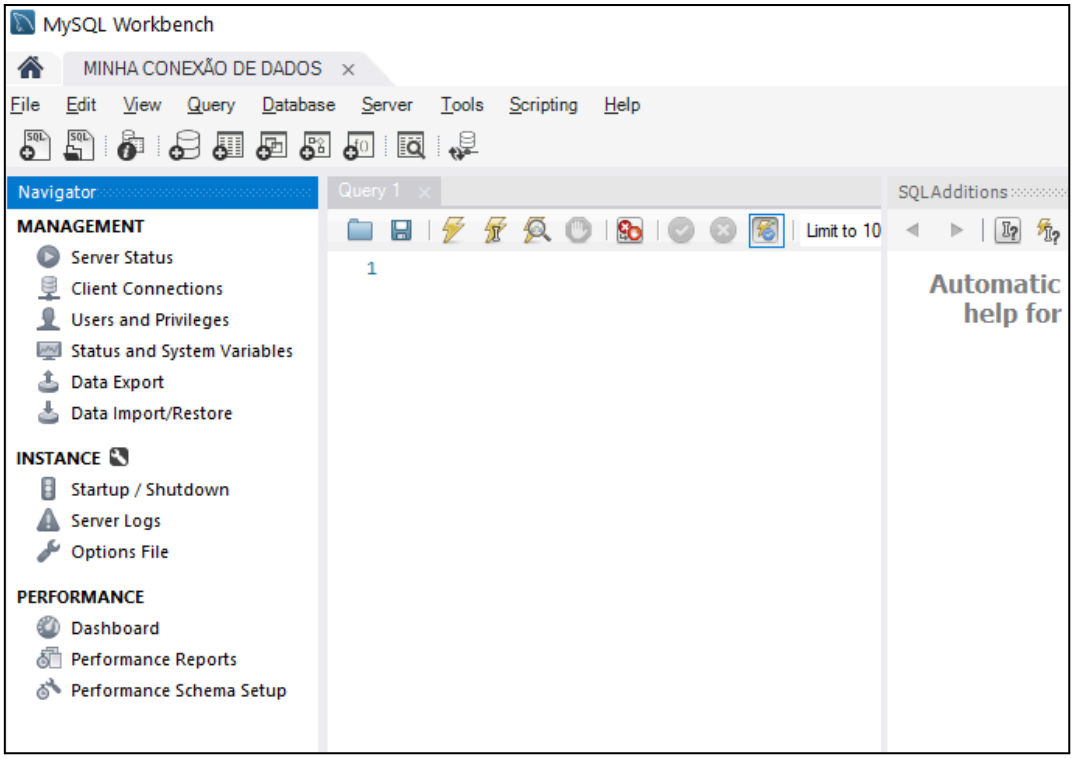
Fonte: Elaborado pelo autor <execução do programa> (2023).

2. Na tela de boas vindas ele mostra quais conexões de banco de dados ele esta reconhecendo, se você acabou de instalar não haverá nenhuma, criamos nossa primeira conexão clicando no botão +, ao lado de **MySQL**
Connections

Fonte: Elaborado pelo autor <execução do programa> (2023).

3. Em connection name colocamos: MINHA CONEXÃO DE DADOS, o restante das informações de referem ao servidor que iremos nos conectar, permitindo inclusive testar a conexão.

4. Agora basta dar um duplo clique no ícone representando a sua conexão que teremos acesso ao painel do Workbench.



Fonte: Elaborado pelo autor <execução do programa> (2023).

Não se preocupem em dominar de imediato todos os comandos do Workbench, agora que temos um servidor e conseguimos conectar a ele, podemos agora continuar a entender algumas regras importantes sobre banco de dados antes de começar a manipular.

Por conceito, antes de criar um banco de dados temos que lembrar que ele deve ser definido como um conjunto de dados organizados e relacionados. A definição de dados compreende como o uso de “fatos conhecidos” que podem ser armazenados e que possuem um significado importante, portanto os mesmos devem ser definidos e coletados de forma organizada.

Compreendendo os campos

Qualquer informação sobre um determinado assunto em um banco de dados é organizado em forma de campos, campos são informações importantes a serem armazenadas.

Um conjunto de campos forma um registro.

Um conjunto de registros forma uma tabela.

Exemplo: Vamos pensar em organizar os meus livros em minha estante, percebi que tenho mais ou menos 40 livros que estão bem desorganizados, pois bem, não é difícil imaginarmos que teríamos uma estante para colocar os livros, e para ficar visualmente agradável seriam separados por tema. Pois bem a “estante” seria analogicamente a minha TABELA, e cada informação importante sobre o livro seriam os campos, sendo que cada livro cadastrado com seus dados na minha tabela

seria um REGISTRO.

Para organizar melhor chamarei esta minha tarefa de SCHEMA: LIVRARIA.

SCHEMA: LIVRARIA

TABELA LIVROS

- NOME LIVRO
- AUTOR
- EDITORA
- ISBN
- TEMA
- VALOR
- DISPONÍVEL PARA TROCA(S/N)

Nesta primeira organização vemos que cada CAMPO terá um tipo, ou seja, qual tipo de informação será armazenada em cada um deles.

NOME LIVRO :- TEXTO DE ATÉ 50 CARACTERES

AUTOR : TEXTO DE ATÉ 40 CARACTERES

EDITORA : TEXTO DE ATÉ 50 CARACTERES

ISBN : CÓDIGO ÚNICO A SER ESTUDADO.

TEMA : TEXTO DE 20 CARACTERES.

VALOR :NÚMERO EXPRESSANDO MOEDA.

DISPONÍVEL (S/N) : TIPO BOOLEAN - tipo lógico que aceita apenas dois estágios VERDADEIRO ou FALSO.

O Banco de Dados irá lhe oferecer tipos de dados, sendo sua obrigação escolher o mais adequado.

Podemos realizar os passos de criar o SCHEMA, CRIAR A TABELA e CRIAR OS CAMPOS DESEJADOS.

No momento de criação é o momento de criar a estrutura, em seguida podemos fazer testes de preenchimento.

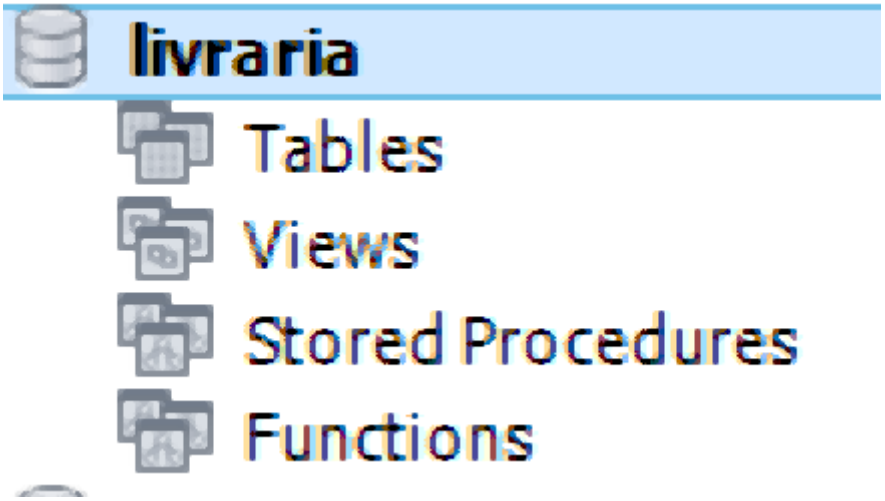
Execute os passos:

1. Conecte-se a conexão criada no Workbench;

2. Identifique o botão  Create a new schema, e dê o nome de LIVRARIA, procure o botão de Apply na parte inferior ele irá mostrar um comando e executar.

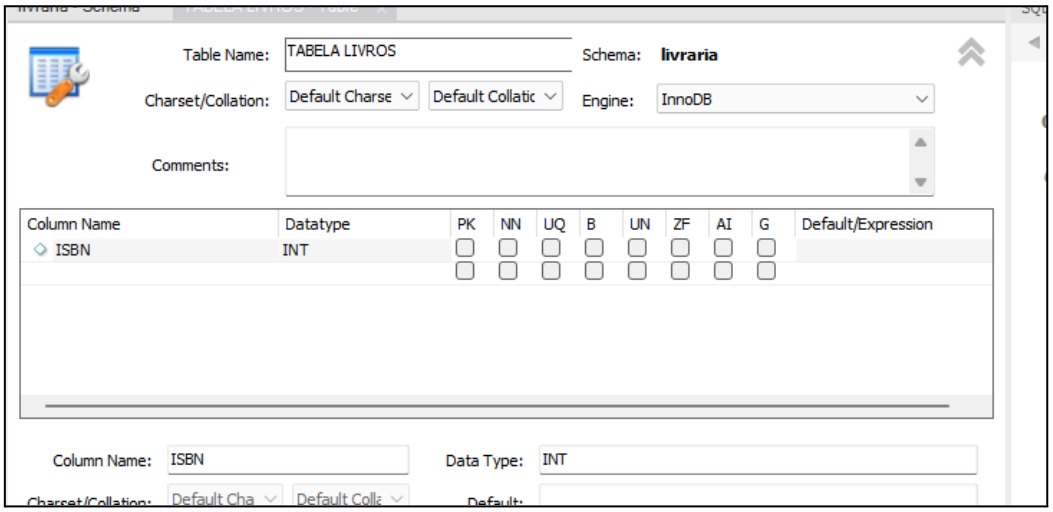
3. Após criar o Schema, encontre a aba SCHEMA e acesse com duplo-clique, caso não aparecer use o botão atualizar.

4. Quando acessar o item SCHEMA repare que ele contêm grupos dentro dele:



Fonte: Elaborado pelo autor <execução do programa> (2023).

5. Clique com o botão direito sobre Tables e escolha Create Table, em seguida se abrirá uma janela que irá permitir a criação de cada campo com seu respectivo tipo, como table name digite: TABELA LIVROS, e em seguida no quadro abaixo crie um campo a cada linha colocando seu NOME e escolhendo seu tipo e características.



Fonte: Elaborado pelo autor <execução do programa> (2023).

6. Preencha em Colum name o nome do campo desejado e escolha o Datatype para cada campo, a princípio não se preocupe com os outros detalhes. Quando estiver criado todos os campos clique no botão APPLY e o Workbench irá mostrar o comando SQL que irá realizar, em seguida irá realizar e criar a tabela.

Não se preocupe com as inúmeras opções que visualizamos nesta ferramenta, aqui nesta introdução estamos nos atendo ao básico, para finalizar uma pergunta importante.

Qual é o campo dos escolhidos para os nossos livros que é único e nunca se repete ?

a resposta seria o ISBN.

O que é o #ISBN?

De acordo com a Câmara Brasileira do Livro, “o ISBN (International Standard Book Number/

Padrão Internacional de Numeração de Livro) é um padrão numérico criado com o objetivo de fornecer uma espécie de "RG" para publicações monográficas, como livros, artigos e apostilas. A difusão global do ISBN e a facilidade com que é lido por redes de varejo, bibliotecas e sistemas gerais de catalogação, tornou-o imprescindível para qualquer publicação.

A sequência é criada a partir de um sistema de registro utilizado pelo mercado editorial e livreiro em todo o mundo. A **estrutura do ISBN** é composta de 13 números que indicam o título, o autor, o país, a editora e a edição de uma obra. (Fonte: https://cbl.org.br)

Disponível em <https://www.cblservicos.org.br/isbn/estrutura/>. Acesso em 20 jul. 2023.

No MySQL, você pode usar o tipo de campo VARCHAR para armazenar a numeração ISBN de um livro. O ISBN (International Standard Book Number) é uma sequência de dígitos e pode incluir hífens. O tipo VARCHAR permite armazenar cadeias de caracteres de comprimento variável, o que é adequado para armazenar o número ISBN. Você pode definir um tamanho máximo para o campo VARCHAR, dependendo do formato do ISBN que você está usando. Por exemplo, o ISBN-13 tem 13 dígitos, então você pode definir o campo VARCHAR com um tamanho máximo de 13 caracteres.

CONCEITO CHAVE-PRIMÁRIA

O ISBN será a chave-primária, esta chave é um campo especial que nunca poderá ter dois registros com o mesmo ISBN, o uso da chave primária impede a duplicação de registros.

O nosso CPF é uma chave-primária a nível federal, não existe o mesmo número usado por duas pessoas, ao contrário do que acontece com o RG quem têm a sua validação a nível estadual e pode gerar duplicidade. Apesar deste primeira vivência prática não estarmos dando ênfase a NORMALIZAÇÃO(veremos normalização mais adiante) é interessante sempre pensar em como o banco ficará acessível e qual será a principal chave de pesquisa.

Fonte Vídeo: https://tinyurl.com/5fm87cup

Não deixe de rever a playlist de reforço, sendo que o segundo vídeo serve para aqueles que ainda têm muito receio em trabalhar com banco de dados de forma geral, encare o segundo vídeo como um desafio para quem já conhece alguma linguagem de programação, em nosso caso específico exatamente o javascript.

Vantagens e desvantagens do uso de um SGBDR

Em termos gerais, o programador não pode apenas se basear em variáveis e matrizes para manter as informações, e o usuário comum de um escritório padrão, de forma intuitiva, mantém os seus dados de forma tabular em planilhas como : Excel ou no Sheet do Google por exemplo.

Então, quais as vantagens de se utilizar um SGBDR ao invés de continuar controlando os dados de forma semi-estruturada em planilha principalmente.

Podemos destacar várias vantagens técnicas formais que são:

Controle de Redundância

Quando se trata do sistema de armazenamento e processamento de arquivos, sem utilizar um SGBD, os dados são responsabilidade dos usuários que os utilizam. Neste caso, as redundâncias são inevitáveis e surgem as seguintes dificuldades:

- quando é necessária a atualização de um arquivo específico de usuário ou grupo, todos os dados devem ser atualizados para manter a sua integridade no ambiente, como um todo;
- dados redundantes levam a um armazenamento excessivo de informações, tomando espaço que poderia estar sendo utilizado com novas informações.

O controle de redundância de um SGBDR começa pelo uso da CHAVE-PRIMÁRIA e passa por outras características técnicas que veremos adiante, o controle de redundância é muito mais possível e efetivo usando o SGBDR.

Compartilhamento de Dados

Dependendo da situação, o compartilhamento para multiusuários de uma planilha ou qualquer outro tipo de dado é uma situação delicada no universo de TI. Um SGBDR multiusuário permite a conexão de muitos usuários acessem o banco de dados ao mesmo tempo. O que é fundamental essencial para múltiplos acessos das aplicações integradas ao banco de dados.

Restrição a Acesso não Autorizado, e segurança na conexão

O SGBDR implementa a segurança no acesso às contas de usuários, com as devidas restrições de acordo com os perfis de cada conta, sendo aplicado tanto para o acesso, como às aplicações que são gerenciadas por ele, cada usuário só terá acesso aquilo que lhe for designado pelo Administrador do Banco de Dados (DBA).

Representação de Relacionamentos Complexos entre Dados

Um banco de dados tem em si um conjunto de dados relacionados de muitas formas, por exemplo: Uma tabela de estoque tem relacionamento direto com a tabela de vendas, porque ao se vender algo deve se dar "baixa" no estoque. O SGBDR gerencia todos os relacionamentos entre as tabelas de dados, assim como, guardar, recuperar e atualizar de forma dinâmica e eficaz.

Tolerância a Falhas

Um SGBD precisa gerenciar e oferecer formas de recuperação a falhas, sejam físicas ou lógicas.

Há situações em que não se deve utilizar um SGBDR?

Não existem desvantagem, mas sim, quando o uso de um SGBD representar um alto custo em comparação aos sistemas de processamento tradicional de arquivos:

- seja um alto custo na compra do software ou infraestrutura;generalidade que um SGBD fornece na definição e processamento de dados;
- sobrecarga no fornecimento do controle da segurança, concorrência, recuperação e integração de funções.
- maior conhecimento por parte do usuário que estava acostumado com um sistema menos seguro (. Pode parecer ridículo mas o ser humano muitas vezes é avesso ao novo).

Problemas adicionais podem surgir caso os projetistas de banco de dados ou os administradores de banco de dados não elaborem os projetos corretamente ou se as aplicações não são implementadas de forma inapropriada.

Se o Administrador do banco de dados (DBA – Database Administrator) não administrar o

banco de dados de forma adequada, tanto a segurança quanto a integridade dos sistemas podem ser comprometidas. Algumas questões justificam a utilização uma abordagem processamento tradicional de arquivos:

- quando um banco de dados e suas aplicações são simples, bem definidas e não se espera mudanças no projeto;
- não há a necessidade de processamento em tempo real das aplicações, podendo ter prejuízo caso sejam sobrecarregadas por um SGBDR;
- quando não houver múltiplos acessos ao banco de dados, ao mesmo tempo.

Reafirmando o nosso exemplo da LIVRARIA, um banco de dados é uma coletânea lógica e coesa de dados, que serão utilizados dentro de um certo DOMÍNIO.

Quando usamos a palavra DOMÍNIO, para o programador representa a situação prática, quais serão as situações e operações de acesso a esta base de dados.

Devemos observar que apenas controlar uma coleção de livros a nível particular é totalmente diferente de colocar estes livros à venda ou para empréstimo em um sistema WEB por exemplo.

Podemos controlar uma lista de livros em uma planilha simples, mas quando formos disponibilizar estes dados no SITE temos que imaginar e preparar o banco de dados para as TRANSAÇÕES.

Seguindo estas premissas reforçamos que a informação em um banco está organizada em forma de registros, cada registro contém toda a informação sobre uma pessoa ou um elemento do banco (no caso da livraria cada registro representa um livro), os dados sobre este livro denominamos como campos.

O nome de um campo geralmente identifica a informação armazenada no campo. Por exemplo, os campos podem se chamar Nome, Endereço ou Número Telefônico. Cada campo tem um tipo que identifica a classe de informação que pode ser armazenada: números, datas, caracteres alfanuméricos e outros.

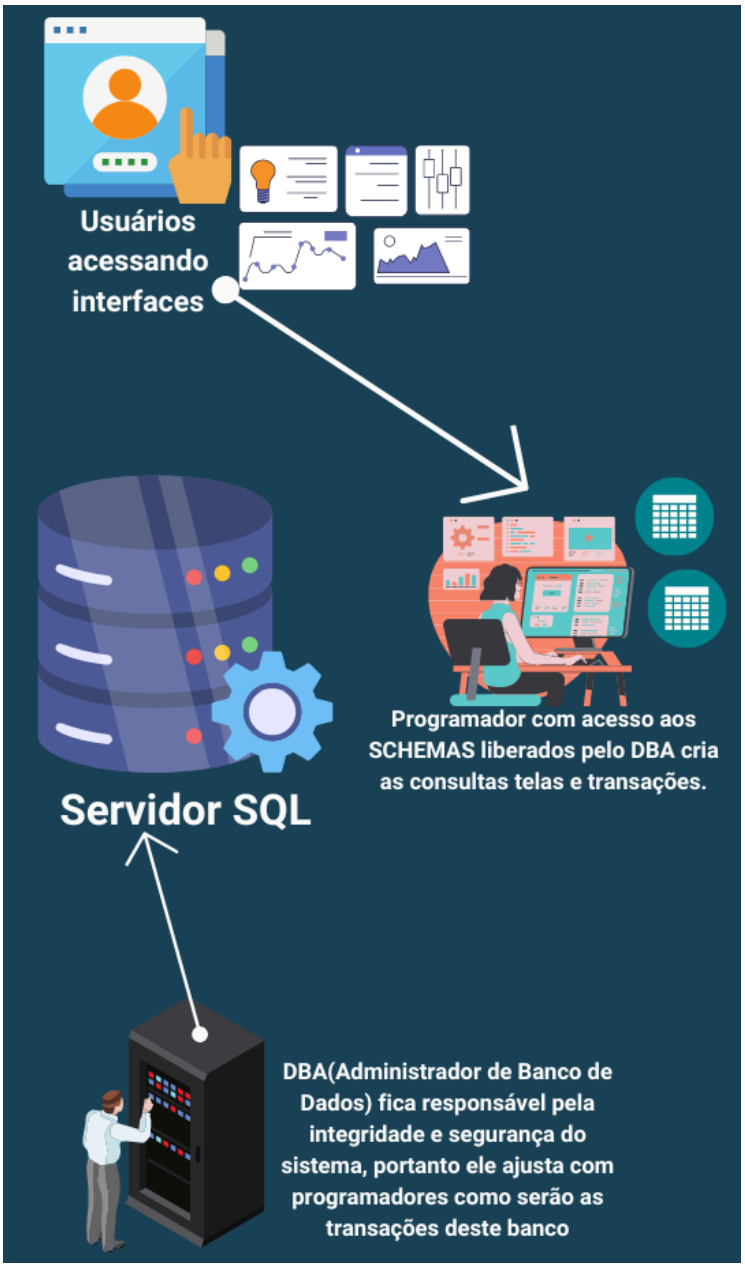
Como cada campo contém um tipo específico de dados, você pode realizar cálculos e outras operações com a informação guardada neles. Por exemplo, pode-se somar os números dos campos. Pode comparar a data de um campo com a de outro. Pode mostrar o nome de uma pessoa (armazenado em um campo) depois de seu sobrenome (armazenado em outro campo) para construir a primeira linha de uma etiqueta de correio. O conjunto de registros que utilizam os mesmos campos forma uma tabela. Cada banco de dados pode ter muitas tabelas. A imagem a seguir mostra como se relacionam estes conceitos.

Usuários comuns: Estes acessam os dados através de interfaces sejam elas página de internet, relatório e ou planilhas que o sistema permite acessar. Por exemplo: Quando estamos navegando em um e-commerce a lista de produtos está sendo recuperada de uma consulta ao SGBDR.

Programadores: Criam as interfaces ou a forma de acesso que os usuários irão interagir, portanto ele precisa estar sempre em contato com o **DBA,** não fica obrigatoriamente responsável pela manutenção ele acessa os SCHEMAS e TABELAS que o DBA deixar como padrão. Evidentemente que o programador deve conhecer a linguagem SQL e estar alinhado ao que o DBA exige como rotinas ao SGBDR.

DBA(Administrador de Banco de Dados): Este profissional sempre fica mais próximo a parte estrutural e acompanha a capacidade e o dia a dia da base de dados, geralmente em conjunto com a

equipe de desenvolvimento ele estabelece os SCHEMAS e as TRANSAÇÕES que irão ou não ser permitidas.



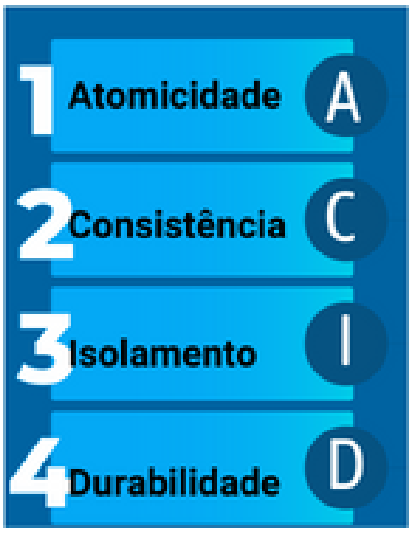
PROPRIEDADES ACID.

"No contexto de banco de dados, transação é uma sequência de operações de banco de dados que satisfaz as propriedades ACID e, portanto, pode ser percebida como uma operação lógica única sobre os dados. Por exemplo, uma transferência de fundos de uma conta bancária para outra, mesmo envolvendo múltiplas mudanças, como debitar uma conta e creditar outra, é uma transação única." - fonte wiki: (ACID – Wikipédia, a enciclopédia livre).

Na prática as condições que um banco deve ter para atender a premissa ACID,são:

Atomicidade: Qualquer operação realizada no banco de dados deve ser feita de forma ATÔMICA(Indivisível), ou seja, se uma transação iniciar e por algum motivo não for finalizada, então

ela não será finalizada e o Banco de dados deve voltar ao seu estado anterior. O exemplo quando estamos enviando um PIX a operação começa em checar o saldo do remetente, enviar a ordem para o destinatário bloqueando instantâneamente o valor e em seguida será creditado na chave fornecida, a transação deve ocorrer por completo, caso contrário uma mensagem de erro deve ser administrada. Isto garante que toda a transação possa ter seu Rollback;



Fonte: Elaborado pelo autor (2023).

Consistência: Toda transação, ou seja mudança efetuada no banco de dados deve levar ele de um estágio consistente a outro estado consistente, isto quer dizer que deve-se respeitar as regras de integridade dos dados (como unicidade de chaves, restrições de integridade lógica, etc.).

Isolamento: A propriedade de isolamento é muito vista em sistema de acesso de vários usuários, onde possivelmente os dados podem sofrer uma tentiva de acesso simultâneo, o banco pode "lockar" ou seja travar determinado registro assim que ele está sendo usado para evitar alterações incoerentes. Exemplo Clássico: Quando eu acesso um produto e coloco ele no meu carrinho, poderíamos "travar-lockar" aquele registro do estoque e enquanto o usuário não finalizar a compra se estiver havendo um acesso simultâneo dar preferência ao primeiro acesso. Imagine em um atendimento de CALL-CENTER ao abrir um atendimento a um usuário de determinado CPF, caso outra tentativa de acessar o mesmo CPF deveria resultar em uma mensagem:"Usuário em atendimento", e não permitir abrir atendimento, já que o sistema indica que ele está sendo atendido naquele momento.

Durabilidade: Esta propriedade se aproxima mais da parte física dos dados, as alterações após uma transação em caso de sucesso (chamamos de commit), devem persistir no banco de dados mesmo em casos de quedas de energia, travamentos ou erros. Deve-se garantir que os dados estarão disponíveis em definitivo. Em um banco de dados relacional, por exemplo, quando um grupo de instruções SQL é executado, os resultados precisam ser armazenados permanentemente (mesmo que o banco de dados falhe imediatamente depois). Para se defender contra a perda de energia, as transações (ou seus efeitos) devem ser registradas em uma memória não volátil.

O **DBA** deve se atentar a estas demandas e o **programador** também se acostuma às mesmas, veja no link do wikipedia quem foi o criador desta regra e desde quando a usamos em SGBDRs.

Em nossa playlist de reforço, que está nas páginas anteriores, separamos um vídeo sobre o

assunto ACID que é bem ilustrativo.

Lembrando que muitas vezes por falta de estrutura ou profissionais qualificados alguns programadores acabam em algumas situações realizando o papel de **Programador e DBA.**

ATIVIDADE DE FIXAÇÃO

1. O que são banco de dados?

2. Explique o conceito de generalidade e alto grau de abstração de banco de dados.

3. Comente por que é importante a eficiência dos sistemas atualmente?

4. Como é possível atingir usos muito mais sofisticados dos recursos de dados?

5. Como eram as primeiras arquiteturas de banco de dados?

6. O que é um SGBDR?

7. O que é um SGBD ou Sistema Gerenciador de Bancos de Dados e qual é a sua melhoria em relação ao armazenamento de dados em arquivos?

8. Quais os benefícios de usar um sistema de banco de dados para uma empresa?

9. Se o Administrador do banco de dados (DBA – Database Administrator) não administrar o banco de dados de forma adequada, o que será comprometido?

10. Algumas empresas ainda usam abordagem de processamento tradicional de arquivos, quais motivos disso ainda acontecer?

TEMA 02

Modelagem De Bancos De Dados

habilidades

- Entender o que é Modelagem de Banco de Dados.
- Conhecer os tipos de modelagem usados.
- Compreender Entidades, Atributos, Relacionamentos e Restrições.
- Praticar e enxergar na prática a modelagem realizada no BD.

Fonte Vídeo: https://youtu.be/SdrTMuY3aTA

Disponível em <Imagem-macrovector-https://tinyurl.com/yxa8ukauhttps://tinyurl.com/mcb37akw>. Acesso em 20 jul. 2023.

Um modelo de dados é uma abstração de um ambiente de dados real e complexo. Os projetistas de banco de dados utilizam os modelos de dados para se comunicar com programadores e usuários de aplicações. Os componentes básicos de modelagem de dados são as entidades, os atributos, os relacionamentos e as restrições.

As regras de negócio são utilizadas para identificar e definir os componentes básicos de modelagens em um ambiente específico real. Em resumo: modelo de dados é a técnica que evita os comandos mais técnicos, mas demonstra através de "desenhos-modelos", como funcionará nosso Banco de Dados no dia a dia da empresa, sendo a regras de negócio as dinâmicas das informações dentro destes bancos de dados.

Projeto de Banco de Dados

O projeto de banco de dados foca em como a estrutura do banco de dados será utilizada para armazenar e gerenciar dados do usuário final.

A modelagem de dados deve ser específica para um determinado problema de domínio. Esse problema de domínio é uma área claramente definida no ambiente real, com escopo e fronteiras bem definidos, que deve ser tratada de forma sistemática, e quanto mais fiel a modelagem for ao ambiente do problema em domínio, maior é a chance de o projeto ter um bom resultado, permitindo assim a criação de um banco de dados mais aderente à realidade, possibilitando de forma mais eficiente o desenvolvimento da aplicação.

Seguindo esta abordagem temos várias formas de modelar, "pensa", "desenhar" um Banco de Dados.

O objetivo da modelagem de dados é garantir que todos os objetos de dados existentes em determinado contexto e requeridos pela aplicação estejam representados com precisão dentro do Banco de Dados.

Pode-se definir modelagem de dados como sendo um conjunto de conceitos que devem ser usados para descrever a estrutura de uma base de dados. Um modelo de dados é uma representação relativamente simples, normalmente gráfica, de estruturas de dados reais mais complexas.

Em termos gerais, modelo é uma abstração de um objeto ou evento real de maior complexidade do ambiente real.

MODELO CONCEITUAL

O Modelo Conceitual é uma abstração da realidade, em que os aspectos do mundo real são descritos de forma natural, bem como seus atributos e relacionamentos. Esse modelo é utilizado para entendimento, transmissão, validação de conceitos e mapeamento do ambiente, possibilitando um melhor diálogo entre desenvolvedores e usuários.

O Modelo Conceitual não está relacionado diretamente com o modelo de banco de dados, forma de acesso ou armazenamento dos dados. Ele está focado em uma representação gráfica de uma realidade existente em um contexto de negócio, conforme está ilustrado na figura. Essa modelagem é feita utilizando o modelo entidade-relacionamento. Tem como características:

- Visão Geral do negócio;

- Facilita o entendimento entre usuários e desenvolvedores;
- Possui somente as entidades e atributos principais;
- Pode conter relacionamentos muitos para muitos.

O modelo conceitual representa uma visão global do banco de dados inteiro conforme visto pela organização como um todo. Ou seja, o modelo integra todas as visões externas (entidades, relacionamentos, restrições e processos) em uma única visão global de todos os dados da empresa. Também conhecido como esquema conceitual, constitui a base para a identificação e descrição de alto nível dos principais objetos de dados (evitando quaisquer detalhes específicos do modelo de banco de dados).

O modelo conceitual mais utilizado é o ER. Lembre-se que o modelo ER é ilustrado com a ajuda do DER que, na prática, constitui a planta básica do banco. Este é utilizado para representar graficamente o esquema conceitual.

O modelo conceitual produz algumas vantagens importantes. Em primeiro lugar, fornece uma visão de cima (nível macro) compreendida de modo relativamente fácil sobre o ambiente de dados.

Em segundo lugar, o modelo conceitual é independente em relação tanto a software como a hardware.

A independência de software significa que o modelo não é dependente de software SGBD utilizado para implantá-lo. A independência de hardware significa que o modelo não depende do hardware utilizado em sua implantação. Portanto, alterações de hardware ou software do SGBD não terão efeito sobre o projeto de banco de dados no nível conceitual. Em geral, o termo projeto lógico é utilizado para se referir às tarefas de criação de modelo de dados conceitual que possa ser implantado em qualquer SGBD. Iremos focar neste Modelo com a ajuda do MYSQL WORKBENCH.

MODELO LÓGICO

Leva em conta limites impostos por algum tipo de tecnologia de banco de dados. (banco de dados hierárquico, banco de dados relacional, entre outros. Suas características são:

- Deriva do modelo conceitual e via a representação do negócio
- Possui entidades associativas em lugar de relacionamentos muitos para muitos
- Define as chaves primárias das entidades
- Normalização até a 3ª forma normal
- Adequação ao padrão de nomenclatura
- Entidades e atributos documentados

MODELO INTERNO

Uma vez selecionado o SGBD específico, o modelo interno mapeia o modelo conceitual para o SGBD. O modelo interno é a representação do banco de dados conforme “visto” pelo SGBD. Em outras palavras, o modelo interno exige que o projetista relacione as características e restrições do modelo conceitual com as do modelo selecionado para implementação. O esquema interno constitui a representação específica de um modelo interno, utilizando estruturas de banco de dados suportadas pelo banco escolhido.

Portanto, o esquema interno deve mapear o modelo conceitual para as estruturas do modelo

relacional, de modo similar, como selecionamos um banco de dados relacional, o esquema interno é expresso utilizando SQL, linguagem padrão para este banco.

Como o modelo interno depende do software específico do banco de dados, diz-se que ele é dependente de software. Portanto, uma alteração no software de SGBD exige que o modelo interno seja alterado para adequar-se às características e exigências de implementação do modelo de banco de dados. Quando é possível alterar o modelo interno sem afetar o modelo conceitual, tem-se independência lógica. No entanto, o modelo interno ainda é independente de hardware, pois não é afetado pela escolha do computador em que o software é instalado. Portanto, uma alteração nos dispositivos de armazenamento ou mesmo nos sistemas operacionais não afetará o modelo interno.

MODELO FÍSICO

Leva em consideração limites impostos pelo SGBD (Sistema Gerenciador de Banco de Dados) e pelos requisitos não funcionais dos programas que acessam os dados. Características:

Elaborado a partir do modelo lógico

- Pode variar segundo o SGBD
- Pode ter tabelas físicas (log)
- Pode ter colunas físicas (replicação)

O modelo físico opera nos níveis mais baixos de abstração, descrevendo o modo como os dados são salvos em meios de armazenamento como discos e fitas. O modelo físico exige a definição tanto dos dispositivos de armazenamento físico como dos métodos de acesso (físico) necessários para se chegar aos dados nesses dispositivos de armazenamento, o que torna dependente tanto de software quanto de hardware. As estruturas de armazenamento utilizados são dependentes do software SGBD e sistema operacional, e dos tipos de dispositivos de armazenamento com que o computador pode trabalhar. A precisão necessária na definição do modelo físico exige que o projetista que trabalha nesse nível tenha conhecimento detalhado do hardware e do software utilizado para implementar o projeto de banco de dados.

Modelos de dados anteriores exigiam que os projetistas levassem em conta os detalhes das necessidades de armazenamento de dados do modelo físico. No entanto, o modelo relacional atualmente dominante é direcionado amplamente para o nível lógico, não para o físico, portanto, não exige os detalhes desse segundo nível como seus antecessores.

Embora o modelo relacional não demande que o projetista se preocupe com as características de armazenamento físico dos dados, a implementação de um modelo relacional pode exigir sintonização refinada em nível físico para melhorar o desempenho. Essa sintonização refinada é especialmente importante quando os bancos de dados muito grandes são instalados em um ambiente mainframe. Mesmo assim, essa sintonização não exige conhecimento das características de armazenamento físico.

O modelo físico é dependente do SGBD, dos métodos de acesso aos arquivos e dos tipos de dispositivos de armazenamento suportados pelo sistema operacional. Quando é possível alterar o modelo físico sem afetar o modelo interno, tem-se independência física. Portanto uma alteração nos dispositivos ou métodos de armazenamento ou mesmo sistema operacional não afetará o modelo interno.

MODELO	GRAU DE ABSTRAÇÃO	FOCO	INDEPENDÊNCIA DE:
Externo	Alto	Visões dos usuários finais	Hardware e software
Conceitual		Visão global dos dados (independente do modelo de banco de dados	Hardware e software
Interno		Modelo específico de banco de dados	Hardware
Físico	Baixo	Métodos de armazenamento e acesso	Nem hardware, nem software.

Fonte: Elaborado pelo autor (2023).

MODELO DE DADOS RELACIONAL.

O Modelo de Dados Relacional foi introduzido por Codd (1970). Entre os modelos de dados de implementação, o modelo relacional é o mais simples, com estrutura de dados uniforme, e o mais formal. O modelo de dados relacional representa os dados da base de dados como uma coleção de relações. Informalmente, cada relação pode ser entendida como uma tabela ou um simples arquivo de registros.

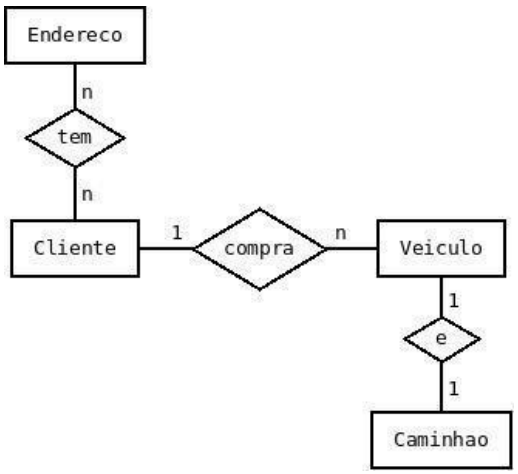


Figura 3: Representação com um Diagrama Entidade-Relacionamento/ Fonte: Elaborado pelo autor (2023).

Conceitos Do Modelo Entidade-Relacionamento

O objeto básico que o MER representa é a entidade. Os blocos básicos de construção de todos os modelos de dados são as entidades, os atributos, os relacionamentos e as restrições.

Entidade

Uma entidade é algo (uma pessoa, um local, um objeto, um evento) sobre o qual sejam coletados e armazenados dados. Ela representa um tipo particular de objeto do mundo real. Por isso, as entidades são distinguíveis, ou seja, cada ocorrência de entidade é única e distinta. As entidades podem ser objetos físicos, como clientes e produtos, mas também abstrações como rotas de voo ou apresentações musicais. Pode ser definida como qualquer coisa do mundo real, abstrata ou concreta, onde se deseja guardar informações. Exemplos de entidades: Cliente, Veículo, Produto, Venda. As entidades estão muitas vezes relacionadas às Tabelas do Banco de Dados sendo elas tabelas persistentes ou tabelas de apoio temporário.

Atributo

Um atributo é uma característica de uma entidade. É tudo o que se pode relacionar como propriedade da entidade (coluna, campo). Chama-se Domínio o conjunto de valores possíveis do atributo, em outras palavras menos técnicas são os campos que compunham a entidade, sendo que este atributo pode ter suas características mais especiais ou não para a coerência da Entidade/Tabela.

Por exemplo, uma entidade chamada cliente seria descrita por atributos como sobrenome, nome, telefone, endereço e limite de crédito de clientes.

Alguns atributos são obrigatórios, outros são opcionais. Algumas definições:

- Atributo obrigatório – é aquele que para uma entidade ou relacionamento deve possuir um valor.
- Atributo opcional - É aquele que para uma instância da entidade ou relacionamento pode possuir um valor ou não.

Classificação:

- **Atributo Identificador** - capaz de identificar exclusivamente cada ocorrência de uma entidade. Também conhecido como **chave Primária** ou Primary Key (PK). O símbolo # é usado para representar a chave primária em algumas notações)

Ex.: Código do Cliente, Código do Produto.. Lembra-se do exemplo da livraria seria o nosso **ISBN**

- **Chave Candidata -** Atributo ou grupamento de atributos que têm a propriedade de identificar unicamente uma ocorrência da entidade. Pode vir a ser uma chave primária. A chave candidata que não é chave primária também se chama chave Alternativa.

Características de uma Chave Primária:

- **NÃO PODE** haver duas ocorrências de uma mesma entidade com o mesmo conteúdo na Chave Primária
- **A chave primária não pode ser composta por atributo opcional**, ou seja, atributo que aceite nulo.
- Os atributos identificadores devem ser o conjunto mínimo que pode identificar cada instância de uma entidade. chamamos de registro cada instância ou seja cada lista de atributos preenchidos e armazenados na entidade.
- **Não devem ser usadas chaves externas ou atributos sobre os quais você não tem controle.**

- **Não deve conter informação volátil.**

Ao criar modelos geralmente temos diversas entidades, cada uma com diversos atributos que podem se relacionar entre si. Vamos definir como podem ser estes relacionamentos.

Importante: Devido a estes Relacionamentos que o Modelo Relacional se destaca em termos de integridade dos dados.

Relacionamento

Um relacionamento pode ser entendido como uma associação entre instâncias de Entidades de acordo com as **regras de negócio, caro aluno entende regra de negócios como enxergamos as formas que os dados serão transformados e usados no sistema**. Normalmente ocorre entre instâncias de duas ou mais Entidades, podendo ocorrer entre instâncias da mesma Entidade (auto relacionamento).

Quando o relacionamento é necessário? Quando existem várias possibilidades de relacionamento entre o par das entidades e se deseja representar apenas um, se ocorrer mais de um relacionamento entre o par de entidades, a fim de evitar ambiguidade, quando houver auto relacionamento, para definir o número de ocorrências de uma entidade usamos o conceito de Cardinalidade.

Podendo então pela Norma termos 03 tipos de relacionamento que são:

- Relacionamento um para muitos (1:M ou 1..*).

Por exemplo, um pintor faz várias pinturas, mas cada uma é criada por apenas um artista. Assim o pintor (uma entidade) relaciona-se com as pinturas (várias entidades).

Portanto, os projetistas de banco de dados identificam o relacionamento PINTOR pinta PINTURA como 1:M ou 1:N.. Particularmente prefiro um para N pois me lembra a matemática.

- Relacionamento de muitos para muitos (M: N ou *..*).

Um funcionário pode aprender várias habilidades profissionais e cada habilidade profissional pode ser aprendida por vários funcionários. Os projetistas de banco de dados identificam o relacionamento FUNCIONÁRIO aprende HABILIDADE como M: N. Pense no caso de um Aluno e as Disciplinas possíveis de cursar, se eu tiver uma Tabela de Alunos e uma Tabela Disciplinas o relacionamento entre estas duas tabelas provavelmente será de N:N.

- Relacionamento um para um (1:1 ou 1..1)

A estrutura de gerenciamento de uma empresa de varejo pode exigir que cada uma de suas lojas seja gerenciada por um único funcionário. Por sua vez, cada gerente de loja, que é um funcionário, gerencia uma loja apenas. Portanto o relacionamento FUNCIONÁRIO gerencia LOJA é identificado como 1:1. A discussão precedente identificou cada relacionamento em duas direções, ou seja, os relacionamentos são bidirecionais:

- Um CLIENTE pode gerar várias FATURAS.
- Cada uma das várias FATURAS é gerada apenas por um CLIENTE. Uma restrição é uma limitação imposta aos dados, que torna confiável alguns tipos de relação, por exemplo, não têm como cadastrar uma fatura sem identificar um cliente que exista no Banco de dados.

As restrições são importantes, pois ajudam a assegurar a integridade dos dados. Elas normalmente são expressas na forma de regras.

Regras de negócio.

Quando os projetistas de banco de dados cuidam da seleção ou determinação das entidades, atributos e relacionamentos utilizados para construir um modelo de dados, podem começar obtendo uma compreensão completa de quais tipos de dados existem em uma organização, como são utilizados e em que período são utilizados. Mas esses dados e informações não produzem por si

Regras de negócio escritas adequadamente são utilizadas para definir entidades, atributos, relacionamentos e restrições. O processo de identificar e documentação de regras de negócio é essencial para o projeto de banco de dados, pois:

- Auxiliam a estandardizar a visualização dos dados de uma empresa;
- Podem constituir uma ferramenta de comunicação entre os usuários e os projetistas;
- Permitem que o projetista compreenda a natureza, o papel e o escopo dos dados;
- Permitem que o projetista compreenda os processos comerciais;
- Permitem que o projetista desenvolva regras e restrições adequadas de participações em relacionamentos e crie um modelo de dados preciso.

DIAGRAMA ENTIDADE-RELACIONAMENTO (DER)

O MER é um modelo conceitual, já o Diagrama Entidade Relacionamento (Diagrama ER - DER) é sua forma de representar graficamente, sua ferramenta de representação. O diagrama permite a visualização das informações, pois o modelo pode ficar muito abstrato sem estar representado, os modelos são muito próximos hoje em dia temos a tendência de já desenhar o DER para termos visualização da regra de negócios mais podemos também visualizar detalhes do atributo. Para criar a representação gráfica, existem algumas regras, conhecidas como notações do DER.

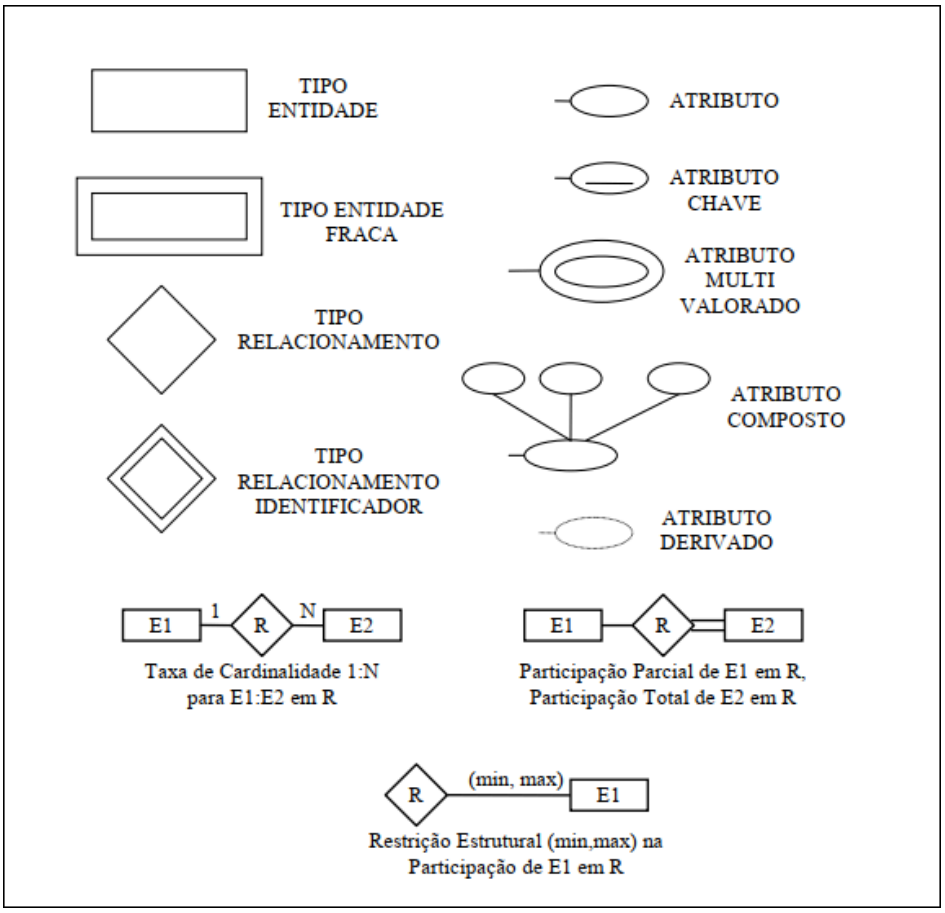


Figura 5: Notações do DER /Fonte: Elaborado pelo autor (2023).

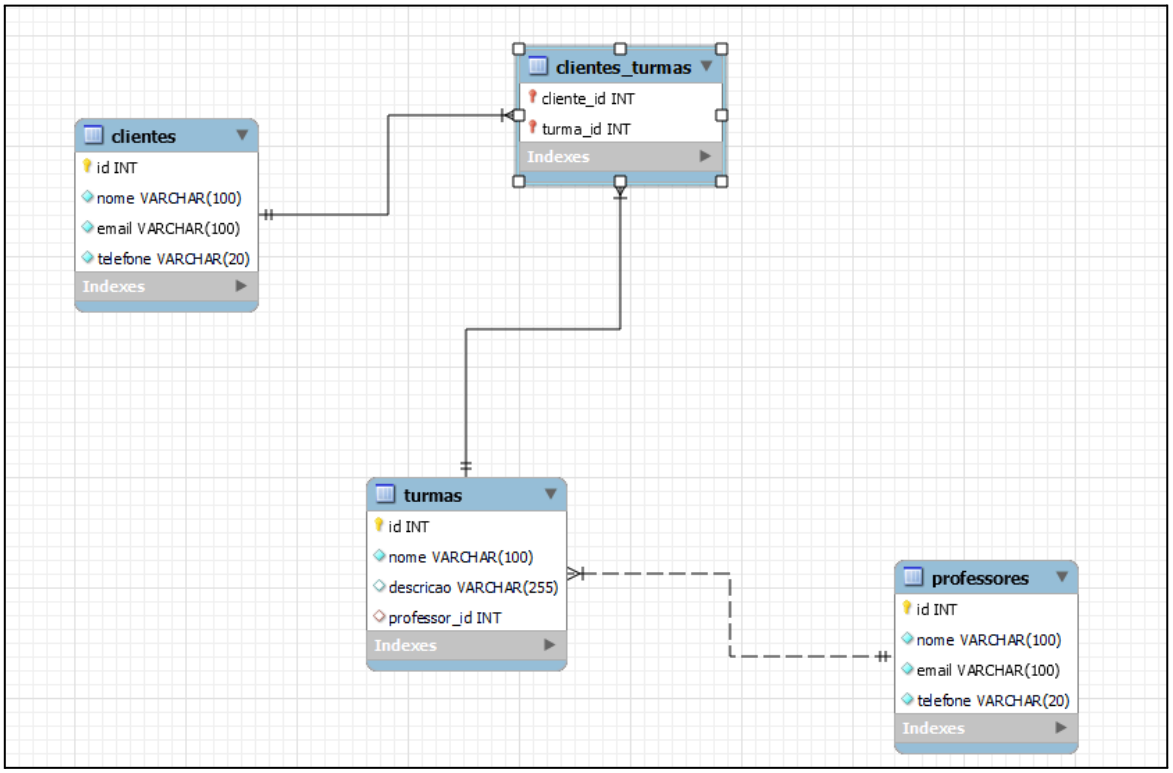
Exemplo de implementação de um sistema, representado em um DER.

O DER abaixo ilustra o esquema da base de dados ESCOLA_INGLES que foi criado através de um SCRIPT SQL. Veja no vídeo do QRCODE:

Fonte do vídeo: https://youtu.be/tv0bnRwaau8

Ao lado vemos a tela do Próprio Mysql Workbench mostrando ao Diagrama criado, através do

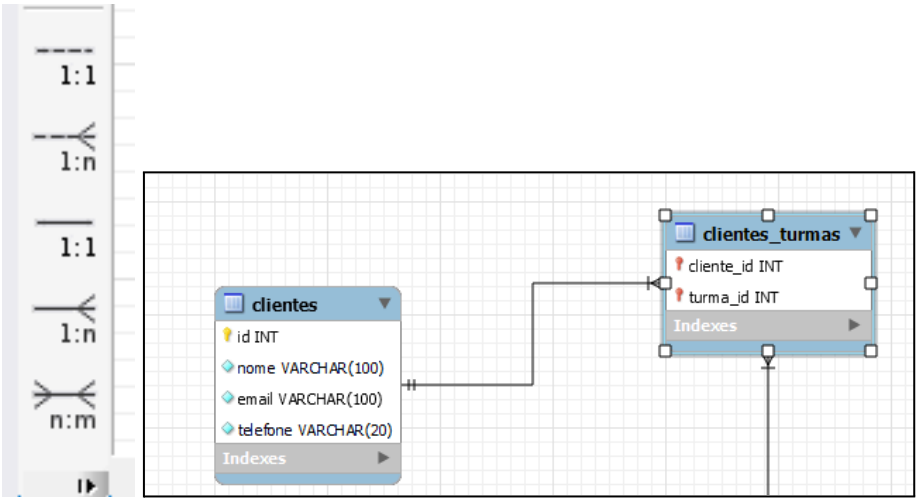
comando menu Database → Reverse → Engineer , como foi demonstrado no vídeo do QR-CODE.



Fonte: Elaborado pelo autor <execução do programa> (2023).

Os tipos de entidades Clientes, Turmas e Professores são mostrados em retângulos, vemos dentro delas os atributos os atributos marcados 🔑(Chave Amarela) são as Chaves-Primárias de cada entidade que se interligam com as chaves estrangeiras de outras entidades.

Simbologia usada pelo My Sql Workbench :



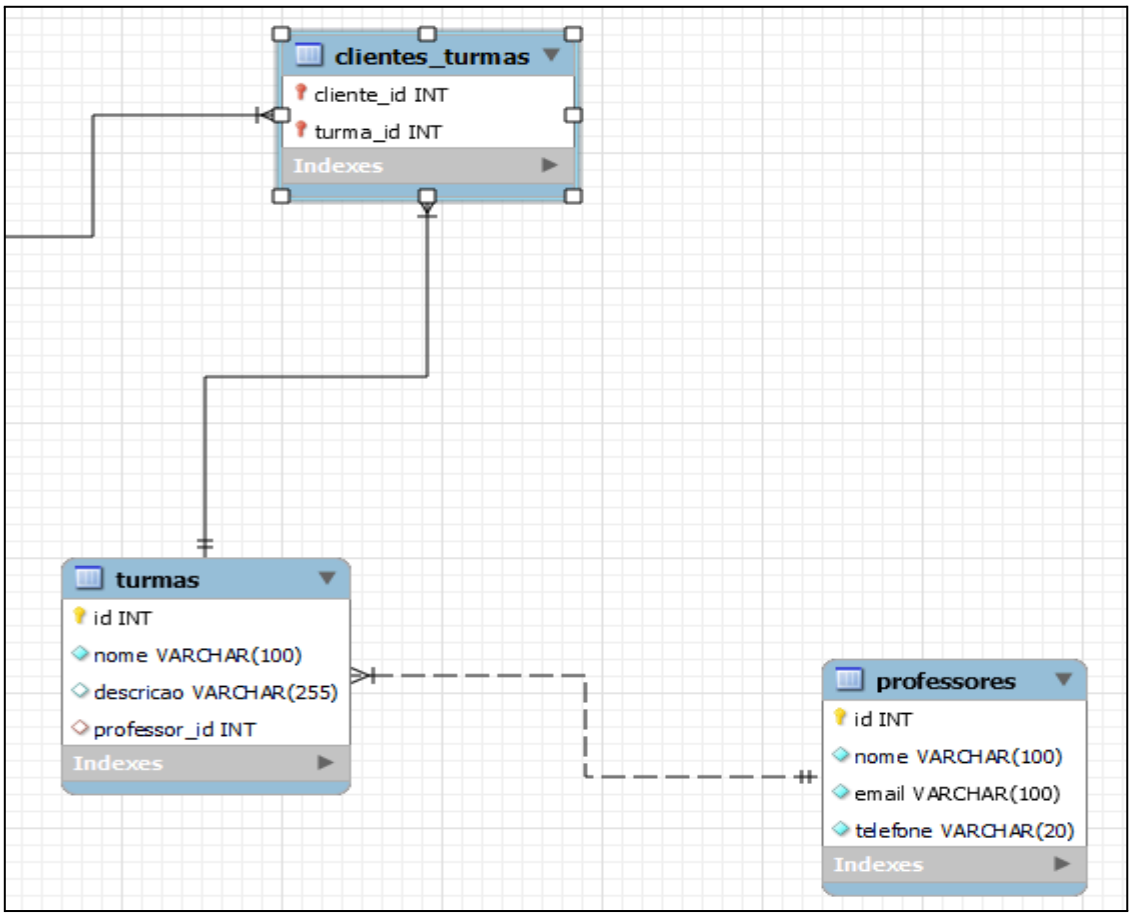
Fonte: Elaborado pelo autor <execução do programa> (2023).

Neste detalhe vemos que para cada cliente existente na tabela **clientes** podem ser geradas

várias ocorrências na tabela **Clientes_turmas**, pois a chave da Clientes_turmas e a combinação **Cliente_id e Turma_id.** Bom, é exatamente isto que acontece em nossa escola de inglês, onde cada aluno cadastrado é um cliente e ele pode se matricular em várias **Clientes_turmas**. O nome Clientes_Turmas poderia ser melhorado para ficar mais claro, mas é muito importante que você aprenda a interpretar o Diagrama.

Observando o detalhe do relacionamento entre as outras Tabelas:

Para cada registro em **Clientes_turmas** ele só aponta para 1 registro em **turmas,** se analisarmos pelo outro lado cada registro em **turmas** pode apontar para várias matrículas em **Clientes_Turmas**.

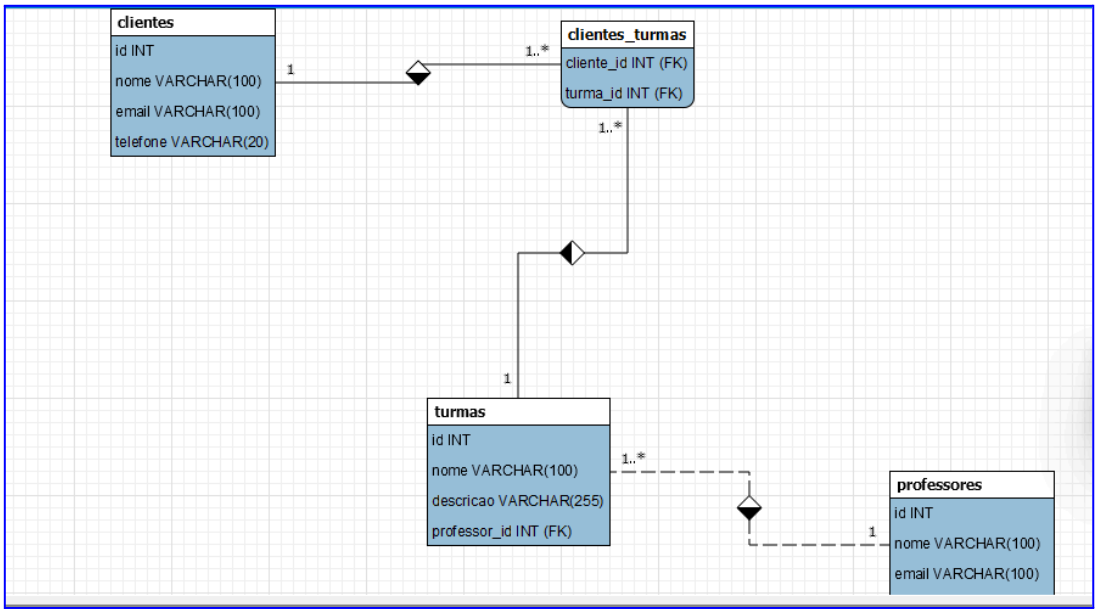


Fonte: Elaborado pelo autor <execução do programa> (2023).

Agora observe que o relacionamento entre **Turmas e Professores** esta desenhado no diagrama com uma linha tracejada, isto indica que apesar de cada registro em **Turmas** apontar para um registro em **Professores** este relacionamento não é obrigatório, ou seja eu posso cadastrar turmas sem obrigatoriamente apontar para o professor, isto me dá uma liberdade de cadastrar outro professor a qualquer momento, ou deixar para indicar professor mais tarde. Na realidade o Diagrama apenas destaca o que nós planejamos com as nossas chaves primárias, estrangeiras e atributos de obrigatoriedade dos campos.

Estamos mostrando o tipo de anotação padrão do WORKBENCH, você encontra na própria ferramenta visualizações diferentes que podem mostrar a CARDINALIDADE dos RELACIONAMENTOS.

Veja esta abaixo:



Fonte: Elaborado pelo autor <execução do programa> (2023).

Neste caso não existe o certo e errado existe o que a equipe consegue enxergar melhor e qual anotação a empresa prefere incluir em sua documentação.

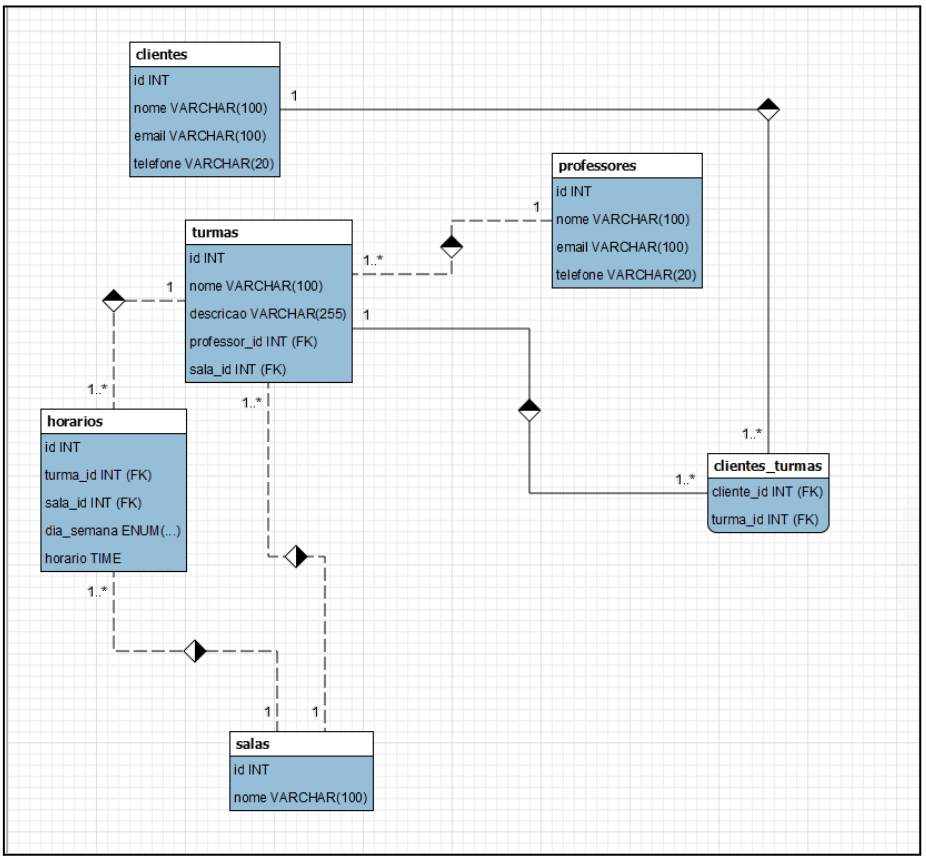
Vamos executar este Script SQL para construir um novo schema, iremos analisar a estrutura criada e criar um Diagrama ER para representá-lo:

```
CREATE DATABASE escola_ingles002;
USE escola_ingles002;
CREATE TABLE Clientes (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nome VARCHAR(100) NOT NULL,
  email VARCHAR(100) NOT NULL,
  telefone VARCHAR(20) NOT NULL
);
CREATE TABLE Professores (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nome VARCHAR(100) NOT NULL,
  email VARCHAR(100) NOT NULL,
  telefone VARCHAR(20) NOT NULL
);
CREATE TABLE Salas (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nome VARCHAR(100) NOT NULL
);
CREATE TABLE Turmas (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nome VARCHAR(100) NOT NULL,
  descricao VARCHAR(255),
  professor_id INT,
```

```
  sala_id INT,
  FOREIGN KEY (professor_id) REFERENCES Professores(id),
  FOREIGN KEY (sala_id) REFERENCES Salas(id)
);
CREATE TABLE Horarios (
  id INT PRIMARY KEY AUTO_INCREMENT,
  turma_id INT,
  sala_id INT,
  dia_semana ENUM('Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta'),
  horario TIME,
  FOREIGN KEY (turma_id) REFERENCES Turmas(id),
  FOREIGN KEY (sala_id) REFERENCES Salas(id)
);
CREATE TABLE Clientes_Turmas (
  cliente_id INT,
  turma_id INT,
  PRIMARY KEY (cliente_id, turma_id),
  FOREIGN KEY (cliente_id) REFERENCES Clientes(id),
  FOREIGN KEY (turma_id) REFERENCES Turmas(id)
);
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Uma das representações possíveis, quanto mais claro ficar a ligação e cardinalidade melhor será para a interpretação de como se dará o fluxo e armazenamento dos dados.



Fonte: Elaborado pelo autor <execução do programa> (2023).

Nota importante: Algumas anotações mais antigas ocupam muito espaço no Diagrama ER, por isto a tendência é a de usar Formas mais parecidas com as vistas no Próprio Workbench, conforme você tiver mais experiência você poderá criar as tabelas e relacionamentos diretos pelo Diagrama ou por Script SQL, iremos nos aprofundar em SQL mais a frente em nosso curso.

ATIVIDADE DE FIXAÇÃO

Peça aos alunos para identificarem as possíveis entidades envolvidas em uma clínica dentária. Por exemplo, eles podem listar entidades como "Paciente", "Dentista", "Consulta", "Procedimento", etc.

1. Atividade: Determinar as Entidades

Nesta atividade o aluno tem que pensar em quais as tabelas de informação e quantas serão necessárias para atender o bom andamento da clínica. Não existe um número certo mais a idéia e listar e identificar o porquê das existência de cada tabela/entidade.

2. Atividade: Atributos das Entidades:

Para cada entidade identificada na atividade anterior, os alunos devem listar os possíveis atributos que descrevem cada entidade. Por exemplo, para a entidade "Paciente", podem ser considerados atributos como "Nome", "Idade", "Gênero", "Telefone", etc. Desenhe todas em papel.

3. Atividade: Cardinalidade dos Relacionamentos:

peça aos alunos que baseado no desenho de Entidades tabelas conjuntos de entidades da clínica dentária e peça para determinarem a cardinalidade do relacionamento entre eles. Por exemplo, a Cardinalidade entre "Paciente" e "Consulta" pode ser um para muitos (um paciente pode ter várias consultas, mas uma consulta pertence a apenas um paciente). Faça uma discussão com os alunos.

4. Atividade: Identificação de Relacionamentos

Dê aos alunos um contexto de clínica dentária e peça para identificarem os possíveis relacionamentos entre as entidades previamente listadas. Por exemplo, um relacionamento "Realiza" pode existir entre "Dentista" e "Procedimento" (um dentista realiza vários procedimentos e um procedimento é realizado por um dentista).

5. Atividade: Diagrama ER Simplificado

Utilizando as informações das atividades anteriores, os alunos devem criar um diagrama ER simplificado representando as entidades, atributos e relacionamentos identificados até o momento.

6. Atividade: Especificação dos Atributos

Para cada atributo listado anteriormente, os alunos devem fornecer uma especificação

detalhada, incluindo o tipo de dado, tamanho e outras propriedades relevantes.

7. Atividade: Refinamento do Diagrama ER

Peça aos alunos para revisarem o diagrama ER criado na atividade 5 e fazerem ajustes ou adições com base nas especificações dos atributos. Eles devem garantir que todas as informações relevantes estejam representadas no diagrama.

8. Atividade: Cardinalidade Específica

Dê aos alunos um novo cenário e peça para determinarem a cardinalidade específica de um relacionamento específico. Por exemplo, no contexto de uma clínica dentária, eles podem analisar a cardinalidade entre "Dentista" e "Consulta" (um dentista pode ter várias consultas, mas uma consulta é realizada por apenas um dentista).

9. Atividade: Diagrama ER Completo

Com base nas informações das atividades anteriores, os alunos devem criar um diagrama ER completo, incluindo todas as entidades, atributos, relacionamentos e cardinalidades.

10. Atividade: Verificação de Consistência

Os alunos devem revisar o diagrama ER completo e verificar se todas as regras de consistência estão sendo atendidas, como a correta definição das chaves primárias e a coerência entre as cardinalidades dos relacionamentos.

Essas atividades formam uma sequência consistente de checagem e modelagem antes da criação do próprio banco de dados.

TEMA 03

Regras de Normalização

habilidades

- Entender o que é Normalização.
- Conhecer e aplicar a técnica de pelo menos 03 Formas Normais.
- Praticar e enxergar qual ganho do uso destas práticas.

Regras de Normalização são guiadas por Formas Normais

A normalização de tabelas tem como objetivo resolver problemas de atualização de bases de dados, minimizando redundâncias. É um processo em que são eliminados esquemas de relações (tabelas) não satisfatórios, decompondo-os, através da separação de seus atributos em esquemas de relações menos complexas, mas que satisfaçam as propriedades desejadas.

Um banco de dados dentro dos padrões de normalização reduz o trabalho de manutenção e ajuda a evitar o desperdício do espaço de armazenamento.

O processo de normalização conduz um esquema de relação através de vários testes para garantir o relacionamento analisado está na **1ª, 2ª e 3ª Formas Normais.**

Cada etapa ou teste corresponde a uma determinada forma normal, que representa um progressivo refinamento na estrutura das tabelas. Assim, uma tabela que se encontra na Terceira Forma Normal é considerada mais normalizada (mais "enxuta", pode-se dizer) que uma tabela que se encontra apenas na Segunda Forma Normal.

A normalização possui caráter organizativo e pode ocorrer durante a concepção do modelo conceitual, durante a derivação do modelo lógico para o relacional, ou após a derivação do modelo lógico.

As principais características de uma base de dados normalizada são:

- Geração de aplicações mais estáveis.
- Aumento do número de tabelas utilizadas.
- Diminuição dos tamanhos médios das tabelas.

Benefícios da normalização:

- Estabilidade do modelo lógico: capacidade de um modelo manter-se inalterado face às mudanças que venham a ser percebidas ou introduzidas no ambiente que tenha sido modelado.
- Flexibilidade: capacidade de adaptação a demandas diferenciadas, a expansão e redução, a omissão ou presença etc.
- Integridade: diz respeito à qualidade do dado. Um dado mapeado em mais de um local de modo diferente, com valores instanciados de modo diferentes, pode ser indício de que não há integridade entre eles.
- Economia: no espaço de armazenamento em relação ao custo de manipulação de dados (que representa todo e qualquer esforço, tempo, ou valor agregado ao fato de manipularmos

volumes de dados maiores do que os efetivamente necessários); custo causado pelo atraso do fornecimento da informação desejada.

- Fidelidade ao ambiente observado: ajuda a definir elementos que foram despercebidos durante o processo de modelagem.

Primeira Forma Normal – 1FN

Uma relação se encontra na primeira forma normal se todos os domínios de atributos possuem apenas valores atômicos (simples e indivisíveis), e que os valores de cada atributo na tupla seja um valor simples, ou seja, a 1FN não permite que haja em um atributo um conjunto de valores ou uma tupla de valores, em outras palavras não são aceitos *relações* dentro de *relações*. Assim sendo, todos os atributos compostos e multivalorados devem ser divididos em atributos atômicos.

Passos para obtenção da 1FN:

- Remova o(s) atributo(s) que viola(m) a primeira forma normal e coloque-o em uma relação em separado (Autores_Livros), juntamente com a chave primária da relação original (IdLivro). A chave primária da nova relação é composta da chave primária da relação original mais uma chave candidata dos atributos da nova relação.
- Poderão ser utilizados tantos atributos quantos forem necessários à formação da chave da nova relação, mesmo que tenhamos que chegar a concatenação de todos os atributos de uma linha. Na prática, o grande número de colunas formando a chave pode causar transtornos, por isso, nesse caso, pode-se criar uma coluna-artificial do tipo contador.

Exemplo de 1FN: Livros

Tabela de Livros: Estrutura original

IdLivro	Título	Assunto	Autor1	Autor2	Autor3
21237	Os Sertões	Ficção	E. Cunha		
33455	Eletricidade básica	Física	A. Silva	B. Santos	
12312	Atlas do Brasil	Geografia	IBGE		

Estrutura normalizada na 1FN:

Tabela de Livros

IdLivro	Título	Assunto
21237	Os Sertões	Ficção
33455	Eletricidade básica	Física
12312	Atlas do Brasil	Geografia

Tabela Autores_Livros

IdLivro	Autor
21237	E. Cunha
33455	A. Silva
33455	B. Santos
12312	IBGE

Fonte: Elaborado pelo autor <execução do programa> (2023).

Exemplo de 1FN: Clientes

Tabela de Clientes: Estrutura original

Codigo	Nome	Telefone	Tipo_tel	Rua	No	Cidade
00001	Maria	3441 8566	Residencial	Contorno	2316	Belo Horizonte
00001	Maria	3215 8751	Serviço	Contorno	2316	Belo Horizonte
00001	Maria	9158 3239	Celular	Contorno	2316	Belo Horizonte
00002	Antônio	8874 5698	Celular	Afonso Pena	5693	Belo Horizonte

Estrutura normalizada na 1FN:

Tabela: Clientes

Codigo	Nome	Rua	No	Cidade
00001	Maria	Contorno	2316	Belo Horizonte
00002	Antônio	Afonso Pena	5693	Belo Horizonte

Tabela: Telefone_Clientes

Codigo	Telefone	Tipo_tel
00001	3441 8566	Residencial
00001	3215 8751	Serviço
00001	9158 3239	Celular
00002	8874 5698	Celular

Fonte: Elaborado pelo autor (2023).execução do programa> (2023).

Segunda Forma Normal – 2FN

Uma relação encontra-se na segunda forma normal quando estiver na primeira forma normal e todos os atributos que não participam da chave primária são dependentes desta.

Assim, devemos verificar se todos os atributos são dependentes da chave primária e retirar-se da relação todos os atributos de um grupo não dependente que dará origem a uma nova relação, que conterá esse atributo como não chave. Desta maneira, na segunda forma normal evita inconsistências devido a duplicidade.

Exemplo de 2FN: Empregados trabalhando em projetos

Tabela de Empregado_Projeto: Estrutura original

Num_emp	Num_proj	Horas	Nome_emp	Nome_proj	Local_proj
00001	001	8	Maria	Versão Evolutiva 3.22	João Monlevade
00002	001	18	José	Versão Evolutiva 3.22	João Monlevade
00003	002	12	Samara	Versão Corretiva 3.21	Belo Horizonte

Estrutura normalizada - 2FN

Tabela: projetos

Num_proj	Nome_proj	Local_proj
001	Versão Evolutiva 3.22	João Monlevade
002	Versão Corretiva 3.21	Belo Horizonte

Tabela: Empregado_projeto

Num_emp	Num_proj	Horas
00001	001	8
00002	001	18
00003	002	12

Tabela: Empregado

Num_emp	Nome_emp
00001	Maria
00002	José
00003	Samara

Fonte: Elaborado pelo autor <execução do programa> (2023).

Terceira Forma Normal – 3FN

Para estar na terceira forma normal a tabela não pode ter atributos não-chave se referindo a outros atributos não-chave. Assim devemos verificar se existe um atributo que não depende

diretamente da chave, retirá-lo criando uma relação que conterá esse grupo de atributos, e defina com a chave, os atributos dos quais esse grupo depende diretamente. O processo de normalização deve ser aplicado em uma relação por vez, pois durante o processo de normalização vamos obtendo quebras, e, por conseguinte, novas relações. No momento em que o sistema estiver satisfatório, do ponto de vista do analista, este processo iterativo é interrompido.

Exemplo de 3FN: Empregados trabalhando em departamentos

Tabela de Empregado_Depto: Estrutura original

Num_emp	Nome	Data_nasc	Num_Depto	Nome_Depto	Emp_Ger_Depto
00001	Maria	06/03/1977	001	Homologação	018
00002	José	27/05/1973	002	Homologação	018
00003	Samara	24/08/1984	003	Desenvolvimento	005

Tabela: Empregado_Depto

Num_emp	Nome	Data_nasc	Num_Depto
00001	Maria	06/03/1977	001
00002	José	27/05/1973	002
00003	Samara	24/08/1984	003

Tabela: Departamento

Num_Depto	Nome_Depto	Emp_Ger_Depto
001	Homologação	018
002	Homologação	018
003	Desenvolvimento	005

Fonte: Elaborado pelo autor <execução do programa> (2023).

Resumo da normalização

O quadro abaixo descreve como realizar teste para identificação da desnormalização da relação e procedimento para normalização.

Formal Normal	Teste	Solução (normalização)
1FN	A relação não deve ter qualquer atributo não-atômico nem relações agrupadas.	Forme novas relações para cada atributo não- atômico ou relação aninhada.
2FN	Para relações nas quais a chave primária contém múltiplos atributos, nenhum atributo não- chave deve ser funcionalmente dependente de uma parte da chave primária.	Decomponha e monte uma rela para cada chave parcial com se atributo(s) dependente(s). Certifique-se manter uma relação com a chave prima original e quaisquer atributos que se completamente dependentes dela em termos funcionais.

3FN	A relação não deve ter um atributo não-chave funcionalmente determinado por um outro atributo não-chave (ou por um conjunto de atributos não-chave). Ou seja, não deve haver dependência transitiva de um atributo não chave na chave primária.	Decomponha e monte uma relação que inclua o(s) atributo(s) não-chave que funcionalmente determine(m) outros atributos não-chave.

Fonte: Elaborado pelo autor (2023).

ATIVIDADE DE FIXAÇÃO

1. O que é normalização?

2. Para que é usada a técnica de normalização?

3. Há alguma vantagem para um banco de dados normalizado?

4. Quais as características do banco de dados normalizado?

5. O que é a 1ª forma normal?

6. O que é a 2ª forma normal?

7. O que é a 2ª forma normal?

8. Quando termina o processo de normalização?

9. Qual ação deve ser tomada quando há relações dentro de relações?

10. O que fazer quando há dependência transitiva de um atributo não chave na chave primária?

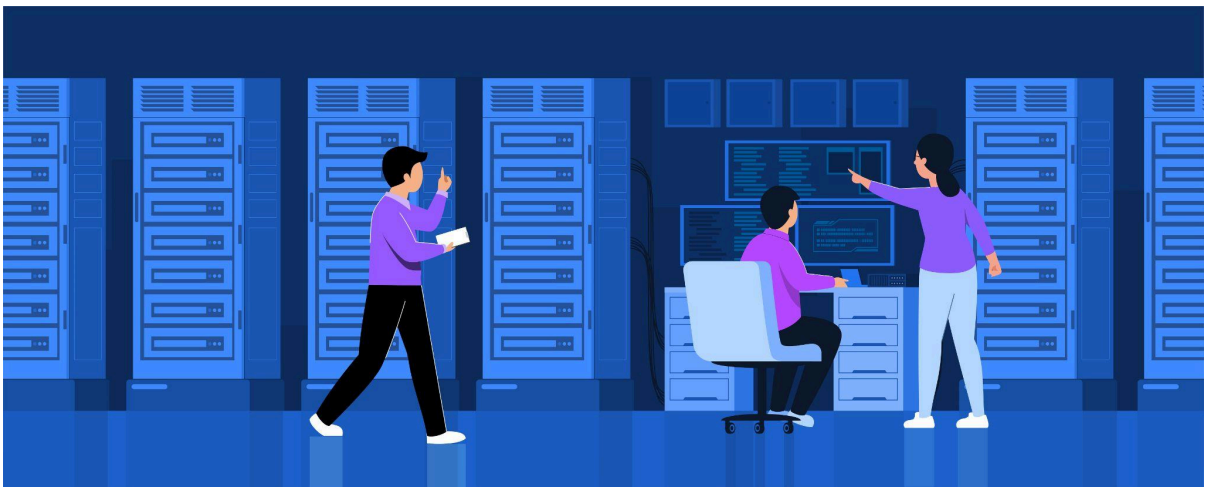
TEMA 04

Introdução À Linguagem Sql - (Ddl)

Habilidades

- Entender porque foi criado a Linguagem SQL
- Conhecer o painel de execução de Scripts
- Conhecer as principais categorias de comando da Linguagem SQL.
- Começar a praticar os comandos da categoria .
- Desenvolver raciocínio e entender o quanto ganhamos em velocidade com o e produtividade com a Linguagem SQL.
- Entender comandos de DML.
- Entender a importância das views e do uso correto do DML.
- Praticar Mysql Workbench criando SCRIPTS e se familiarizar com a linguagem SQL.

Fonte vídeo: https://youtu.be/PnySDkF9kRE



Disponível em <Imagem de jcomp-https://tinyurl.com/chw8f5p2>. Acesso em 20 jul. 2023.

A "Linguagem de Consulta Estruturada", comumente referida como SQL, é empregada para estabelecer comunicação com um Sistema de Gerenciamento de Banco de Dados (SGBD) e desempenhar diversas funções, tais como a criação de bases de dados, tabelas, campos e valores.

Além disso, é utilizada para a inserção e modificação de registros, a formação de objetos dentro do banco de dados, a administração de usuários e a busca por informações, bem como o controle de operações. Todas as transações realizadas no banco de dados podem ser requisitadas ao SGBD por meio da linguagem SQL.

A programação SQL se presta a analisar e efetuar operações em tabelas, além de habilitar a execução de tarefas. O SQL também capacita a realização de consultas e análises mais avançadas, permitindo a formulação de consultas com múltiplos critérios.

Exatamente por poder interagir de várias formas com seu sistema de banco de dados a linguagem é dividida em grupos:

- DDL
- DQL
- DML
- DCL e TCL.

A criação da Linguagem SQL, começou na Década de 70, dentro da IBM, veja histórico em: https://pt.wikipedia.org/wiki/SQL

Você verá neste artigo da Wikipédia que ela sofreu várias alterações, melhorias, mas a sua estrutura básica é usada em praticamente todos os Banco de Dados Relacionais da Atualidade.

Vamos começar pelo Começo:

GRUPO DDL (Data Definition Language - Linguagem de Definição de Dados).

Estes comandos permitem ao utilizador definir as tabelas novas(entidades) e elementos associados, foi um SCRIPT em formato DDL que usamos nos temas anteriores para criar nossas tabelas, executar estas scripts é muitas vezes mais rápido do que criarmos a tabela usando comandos da interface gráfica.

Como iremos criar estrutura que irão conter dados, precisamos antes de mais nada saber quais são os tipos básicos permitidos em nosso banco de dados.

Tipos de dados básicos

Dentro do contexto de bases de dados relacionais, em cada tabela é possível encontrar uma variedade de colunas. Estas colunas, por sua vez, representam os atributos ou campos específicos; para cada um deles é associado um tipo de dado correspondente. Durante o processo de criação da tabela, são estabelecidos os formatos dos tipos de dados. Os tipos primordiais de dados simples que são especificados são:

Tipos de Dados	Descrição
CHARACTER	Caractere de tamanho fixo – usualmente conhecido como CHAR
CHARACTER VARYING	Caractere de tamanho variante – usualmente conhecido como VARCHAR
CHARACTER LARGE OBJECT	Caractere longo – usualmente conhecido como CLOB
BINARY LARGE OBJECT	String binária para objetos longos – usualmente conhecido como BLOB
NUMERIC	Numérico exato
DECIMAL	Numérico exato
SMALLINT	Numérico exato
INTERGER	Numérico exato
BIGINT	Numérico exato
FLOAT	Numérico aproximado
REAL	Numérico aproximado
DOUBLE PRECISION	Numérico aproximado
BOOLEAN	Booleano
DATE	Data com informações de dia, mês e ano
TIME	Hora com informações de hora, minuto e segundo
TIMESTAMP	Determina um momento, com informações de ano, mês, dia, hora, minuto e segundo

Figura: Tipos de dados/ Fonte: Elaborado pelo autor (2023).

Para estudar mais a fundo as várias possibilidades de campos que o MYSQL pode armazenar consulte:

https://dev.mysql.com/doc/refman/8.0/en/data-types.html

PRINCIPAIS COMANDOS DDL

CREATE DATABASE

A indicação "create database", conforme seu próprio título indica, tem como propósito a geração da estrutura de dados primordial onde as tabelas serão concebidas, utilizando uma forma de escrita descomplicada.

CREATE DATABASE - Cria um novo banco de dados MySQL.

Para efetuar a criação de um banco de dados, é necessário possuir privilégios especiais de superusuário ou a capacidade de CREATEDB.

Por padrão, a configuração inicial do banco de dados ocorrerá com base no modelo de sistema conhecido como template1. Contudo, é viável especificar um modelo distinto ao mencionar o nome do TEMPLATE. Especificamente, ao utilizar a nomenclatura TEMPLATE0, é possível gerar um banco de dados vazio que inclui somente os elementos predefinidos padrão correspondentes à versão específica do MySQL. Tal abordagem é útil caso se queira evitar a replicação de objetos locais do sistema que possam ter sido incorporados ao template1.

```
EFETUAR CRIAÇÃO DO BANCO DE DADOS
[ [ COM ] [ PROPRIETÁRIO [=] nome_de_usuário ] [ MODELO [=] modelo ]
[ CODIFICAÇÃO [=] codificação ]
[ LC_COLLATE [=] lc_collate ] [ LC_CTYPE [=] lc_ctype ]
[ ESPAÇO_DE_TABELA [=] espaco_de_tabela ]
[ LIMITE_DE_CONEXÕES [=] limite_de_conexoes ] ]
```

Parâmetros

nome_de_usuário: Refere-se ao nome da função de usuário que será o detentor do novo banco de dados. Caso necessário, pode-se selecionar o valor DEFAULT para empregar as configurações padrão, ou seja, o usuário que está executando o comando. Caso deseje criar um banco de dados vinculado a outra função, é requerido que você seja membro direto ou indireto dessa função, ou então, que possua status de superusuário.

modelo: Denomina o template utilizado como base para a criação do novo banco de dados. Alternativamente, é possível optar pelo valor DEFAULT para adotar o template padrão (template1).

codificação: Especifica a codificação do conjunto de caracteres empregada no novo banco de dados. Pode ser fornecida uma constante de cadeia (como 'SQL_ASCII'), um número de codificação inteiro, ou DEFAULT para fazer uso da codificação padrão, correspondente à do modelo do banco de dados.

lc_collate: Diz respeito à ordem de agrupamento (LC_COLLATE) aplicada ao novo banco de dados. Isso influencia a ordenação de strings em operação.

CREATE TABLE

Após a configuração da base de dados, torna-se necessário estabelecer as tabelas. Para esta finalidade, emprega-se o comando de criação de tabela, que viabiliza a formação e a definição do formato de uma tabela. No âmbito desse comando, é viável também estipular os campos (denominados colunas), bem como as suas correspondentes restrições, juntamente com as chaves primárias e estrangeiras.

A instrução CREATE TABLE delimita uma nova tabela, originalmente desprovida de quaisquer entradas, no esquema do banco de dados corrente. A tabela recém-gerada está sob a responsabilidade do usuário que executa a instrução (ou seja, o usuário conectado).

Existe a possibilidade de designar um esquema distinto do esquema em uso para a geração da tabela, mediante a associação do esquema ao nome escolhido para a tabela (separado por um ponto.), utilizando uma sintaxe tal como esquema nome.nome_tabela.

O esquema a ser seguido na elaboração de uma tabela é o seguinte:

Formatação fundamental:

```
CREATE TABLE IF NOT EXISTS nome_tabela (
    nome_coluna1 tipo_de_dado1 [COLLATE colação1] restrição1,
    nome_coluna2 tipo_de_dado2 restrição2,
    nome_coluna3 tipo_de_dado3 restrição3,

     [CHAVE ESTRANGEIRA chave_externa FAZ REFERÊNCIA À coluna] [NA AÇÃO DELETAR
ação] [NA AÇÃO ATUALIZAR ação]
);
```

Se a opção COLLATE não for utilizada, o padrão de colação será empregado. Os elementos contidos entre colchetes
[ ] são opcionais durante a elaboração da tabela. A cláusula "SE NÃO EXISTIR" avalia se a tabela com o nome especificado já existe antes de iniciar a criação.

As restrições possíveis para as colunas abarcam:

- NOT NULL
- NULL (configuração padrão)
- VERIFICAÇÃO (CHECK)
- PREDEFINIÇÃO (DEFAULT)
- ÚNICO (UNIQUE)
- CHAVE PRIMÁRIA (PRIMARY KEY)
- FAZ REFERÊNCIA À (REFERENCES)

Quanto às restrições específicas para tabelas, tem-se:

- VERIFICAÇÃO (CHECK)
- ÚNICO (UNIQUE)
- CHAVE PRIMÁRIA (PRIMARY KEY)
- EXCLUSÃO (extensão específica do MySQL)
- CHAVE ESTRANGEIRA (FOREIGN KEY)

Na realidade, existem duas maneiras distintas de estabelecer limitações. É viável definir restrições tanto a nível de colunas quanto em relação à tabela como um todo.

Uma limitação referente a uma coluna é estipulada como parte integrante da descrição da coluna em si. Em contrapartida, uma restrição relacionada à tabela não se encontra associada a uma coluna específica, podendo ser aplicada simultaneamente a múltiplas colunas.

A cláusula de verificação é empregada para impor restrições ao domínio. Por exemplo, ao criar uma tabela de projetos, é possível inserir uma restrição que assegura que a data de início do projeto seja anterior à data prevista de conclusão. A cláusula de verificação também tem a capacidade de comparar um atributo com um valor absoluto, expandindo assim sua aplicação para além da simples comparação entre atributos.

Exemplo de ilustração sobre como estabelecer estruturas tabulares no MySQL:

Exemplo 1

A demonstração a seguir envolve a formação de uma tabela denominada "autores_tabela". Essa tabela será empregue para armazenar informações como COD_Autor, Nome_Autor, Sobrenome_Autor e Data_Nasc:

Ao efetuarmos o comando abaixo, a tabela_autores será criada, com a inclusão dos seguintes elementos: codigo_Autor (um número inteiro de até quatro dígitos, exclusivo e aumentado automaticamente), nome_Autor (um texto de até 30 caracteres e que não pode ser deixado em branco), sobrenome_Autor (um texto de até 50 caracteres) e data_Nasc (uma data específica). A definição de PRIMARY KEY fica associada ao campo codigo_Autor.

```
CREATE TABLE tabela_autores (
    codigo_Autor INT UNIQUE AUTO_INCREMENT,
    nome_Autor VARCHAR(30) NOT NULL,
    sobrenome_Autor VARCHAR(50),
    data_Nasc DATE,
    PRIMARY KEY (codigo_Autor)
);
```

Fonte: Elaborado pelo autor (2024)

Exemplo 2

Prosseguindo, apresentamos a geração da tabela de nome "editoras_tabela", compreendendo os campos COD_Editora e Nome_Editora:

O código subsequente originará a tabela_editoras, que abraçará os campos cod_Editora (um número inteiro) e Nome_Editora (um texto de até 35 caracteres, único e obrigatório). A definição de PRIMARY KEY está vinculada ao campo cod_Editora.

```
CREATE TABLE tabela_editoras (
    cod_Editora INT,
    Nome_Editora VARCHAR(35) UNIQUE NOT NULL,
    PRIMARY KEY (cod_Editora)
);
```

Fonte: Elaborado pelo autor (2024)

Demonstração de como criar tabelas no MySQL:

Exemplo 3

Vamos exemplificar também a construção de uma tabela destinada a guardar informações sobre gêneros literários:

```
CREATE TABLE tabela_generos (
    COD_Genero INT,
    Nome_Genero VARCHAR(40) UNIQUE NOT NULL,
    PRIMARY KEY (COD_Genero)
);
```

Fonte: Elaborado pelo autor (2024)

Exemplo 4

Por fim, iremos estabelecer a criação da tabela de livros, estabelecendo vínculos com outras tabelas por meio do uso de chaves estrangeiras:

```
CREATE TABLE tabela_livros (
    COD_Livro INT UNIQUE NOT NULL,
    Nome_Livro VARCHAR(50) NOT NULL,
    Autor INT NOT NULL,
    Editora INT NOT NULL,
    Data_Pub DATE,
    Genero INT NOT NULL,
    Preco_Livro DECIMAL,
    FOREIGN KEY (Autor) REFERENCES tabela_autores (codigo_Autor) ON DELETE CASCADE
);
```

Fonte: Elaborado pelo autor (2024)

O termo SERIAL na definição da coluna COD_Livro indica a aplicação de incremento automático nessa coluna. Isso implica que os códigos dos livros serão gerados automaticamente pelo MySQL, dispensando a necessidade de inserção desses dados ao cadastrar os livros, o que será explanado no próximo artigo. Adicionalmente, não é preciso indicar o tipo de dado dessa coluna, visto que o incremento automático ocorre sempre com números inteiros (integer).

Outros exemplos de criação de tabela:

1. Exemplo de criação de tabela com restrição UNIQUE:

```
CREATE TABLE PessoaUnica (
    ID int NOT NULL,
    Sobrenome varchar(255) NOT NULL,
    Nome varchar(255),
    Idade int,
    CONSTRAINT UQ_ID UNIQUE (ID)
);
```

Fonte: Elaborado pelo autor (2024)

2. Exemplo de criação de tabela com chave primária:

```
CREATE TABLE Pessoa (
    ID int NOT NULL,
    Sobrenome varchar(255) NOT NULL,
    Nome varchar(255),
    Idade int,
    PRIMARY KEY (ID)
);
```

Fonte: Elaborado pelo autor (2024)

3. Exemplo de criação de tabela com chave estrangeira:

```
CREATE TABLE Pedidos (
    PedidoID int NOT NULL,
    PedidoNumr int NOT NULL,
    PessoaID int,
    PRIMARY KEY (PedidoID),
    FOREIGN KEY (PessoaID) REFERENCES Pessoa(ID)
);
```

Fonte: Elaborado pelo autor (2024)

4. Exemplo de criação de tabela com valor padrão (DEFAULT):

```
CREATE TABLE Pessoa (
    ID int NOT NULL,
    Sobrenome varchar(255) NOT NULL,
    Nome varchar(255),
    Idade int,
    Cidade varchar(255) DEFAULT 'Sandnes'
);
```

Fonte: Elaborado pelo autor (2024)

5. Exemplo de criação de tabela com restrição CHECK:

```
CREATE TABLE Pessoa (
    ID int NOT NULL,
    Sobrenome varchar(255) NOT NULL,
    Nome varchar(255),
    Idade int,
    CONSTRAINT CK_Idade CHECK (Idade >= 18)
);
```

Fonte: Elaborado pelo autor (2024)

O COMANDO INSERT RESPONSÁVEL POR INSERIR DADOS NA TABELA NÃO É DA CATEGORIA DE DDL, ELE É UM COMANDO DE MANIPULAÇÃO(DML) PORÉM IREMOS USÁ-LO PARA TESTAR A "POPULAÇÃO" DAS TABELAS/ENTIDADES.

INSERINDO DADOS NAS TABELAS CRIADAS NO MYSQL:

O comando INSERT permite incluir novos dados dentro de uma tabela. Sintaxe:

`INSERT INTO nome_tabela VALUES (valores);`

ou I

`INSERT INTO nome_tabela (campos) VALUES (valores);`

Exemplo

```
Usando o formato VALUES para inserir todos os valores:

INSERT INTO tabela_autores VALUES (1, 'Carlos', 'Drumond de Andrade', '1902-10-31');

Ou especificando AS colunas que serão inseridas:

INSERT INTO tabela_autores (cod_Autor, nome_Autor, sobrenome_Autor, data_Nasc)
```

Fonte: Elaborado pelo autor (2024)

Detalhes Comando ALTER TABLE, comandos DROP.

O comando alter table

O comando alter table permite alterar a estrutura de uma tabela adicionando, alterando, retirando e alterando nomes, formatos das colunas e a integridade referencial definidas em uma determinada tabela.

- **nome-tabela** representa o nome da tabela que será atualizada.
- **nome-coluna** representa o nome da coluna que será criada.
- **tipo-do-dado** a cláusula que define o tipo e tamanho dos campos definidos para a tabela.
- **DROP nome-coluna** realiza a retirada da coluna especificada na estrutura da tabela. **ADD nome-coluna tipo-do-dado** realiza a inclusão da coluna especificada na estrutura da tabela. Na coluna correspondente a este campo nos registros já existentes será preenchido o valor NULL (Nulo). As definições NOT NULL e NOT NULL WITH DEFAULT são semelhantes à do comando CREATE TABLE.
- **MODIFY nome-coluna tipo-do-dado** permite a alteração na característica da coluna especificada.

Apagando uma coluna de uma tabela

Imagine que você deseja, por alguma razão, apagar a coluna que armazena o nome Nome_Genero dos livros da tabela tabela_generos.

Exemplo

```
ALTER TABLE tabela_generos DROP Nome_Genero;
```

Modificando uma coluna de uma tabela

Se precisássemos mudar as características de uma coluna da tabela após a sua criação, usaríamos o comando modify. Como exemplo, imagine que desejamos aceitar valores nulos no atributo PedidoNumr da tabela Pedidos.

```
CREATE TABLE Pedidos (PedidoID int NOT NULL, PedidoNumr int NOT NULL, PessoaID int, PRIMARY KEY (PedidoID),
FOREIGN KEY (PessoaID) REFERENCES Pessoa(PessoaID)
);
```

Fonte: Elaborado pelo autor (2024)

Exemplo

```
ALTER TABLE Pedidos
MODIFY PedidoNumr INTEGER NULL;
```

O COMANDO DROP TABLE

O comando drop table serve para destruirmos uma tabela. Se, por exemplo, precisássemos destruir a tabela Pedidos ,

Exemplo

```
DROP TABLE Pedidos;
```

```
ALTER TABLE
```

```
ALTER TABLE table_name
ADD column_name datatype;

ALTER TABLE table_name
ALTER COLUMN column_name datatype;

ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

Create table as

```
CREATE TABLE new_table_name AS
    SELECT column1,                          column2,...
    FROM existing_table_name
    WHERE ....;
```

Drop table

```
DROP TABLE table_name;
```

Figura: Tipos de dados/ Fonte: Elaborado pelo autor (2023).

SELECT

O operador SELECT é unário; isto é, ele é aplicado somente a uma relação. Assim, o SELECT não pode ser usado para selecionar tuplas de mais de uma relação. Observe também que a operação de seleção é aplicada individualmente para cada tupla. Assim, as condições de seleção não podem ser aplicadas a mais que uma tupla. O grau da relação resultante é a mesma que a relação original.

O número de tuplas da relação resultante é sempre menor ou igual ao número de tuplas da relação original. O comando SELECT permite recuperar os dados de um objeto do banco de dados, como uma tabela, view e, em alguns casos, uma stored procedure (alguns bancos de dados permitem a criação de procedimentos que retornam valor).

A sintaxe mais básica do comando é:

```
SELECT <lista_de_campos>
FROM <nome_da_tabela></nome_da_tabela></lista_de_campos>;
```

Exemplo:

```
SELECT CODIGO, NOME FROM CLIENTES;

SELECT * FROM CLIENTES;
```

Fonte: Elaborado pelo autor (2023).

É possível selecionar o conteúdo de uma tabela parcial colocando restrições para as linhas a serem incluídas no resultado. Isto é feito com a utilização da cláusula WHERE para adicionar restrições condicionais ao comando SELECT. A sintaxe a seguir permite especificar quais linhas serão selecionadas:

```
SELECT <lista_de_campos>
FROM <nome_da_tabela></nome_da_tabela></lista_de_campos>
WHERE  <lista de condições>;
```

Fonte: Elaborado pelo autor (2023).

O comando SELECT recupera todas as linhas que atendam às condições especificadas – também conhecidas como critérios condicionais – na cláusula where. A lista de condições em where é uma representação por uma ou mais expressões condicionais, separadas por operadores lógicos.

A cláusula where é opcional. Se nenhuma linha atender aos critérios especificados nesta cláusula, o usuário verá uma tela em branco ou uma mensagem dizendo que nenhuma linha foi retornada.

Operadores SQL

Os operadores aritméticos desempenham operações matemáticas em duas expressões de um ou mais dos tipos de dados da categoria de tipo de dados numéricos.

OperadorSignificado

+ (Somar)Adição

- (Subtrair) Subtração

* (Multiplicar) Multiplicação

/ (dividir) Divisão

% (módulo) Retorna o resto inteiro de uma divisão. Por exemplo, 12% 5 = 2 porque o resto de 12 dividido por 5 é 2.

= (Operador de atribuição)

O sinal de igual (=) é o único operador de atribuição Transact-SQL.

O operador de atribuição também pode ser usado para estabelecer a relação entre um título de coluna e a expressão que define os valores para a coluna.

SELECT FirstColumnHeading = ‘xyz', SecondColumnHeading = ProductID FROM Production.Product;

Operadores compostos

Os operadores compostos executam alguma operação e definem um valor original para o resultado da operação.

Operador	Significado
ALL	TRUE se tudo em um conjunto de comparações for TRUE.
AND	TRUE se as duas expressões boolianas forem TRUE.
QUALQUER	TRUE se qualquer conjunto de comparações for TRUE.
BETWEEN	TRUE se o operando estiver dentro de um intervalo.
EXISTS	TRUE se uma subconsulta tiver qualquer linha.
IN	TRUE se o operando for igual a um de uma lista de expressões.
LIKE	TRUE se o operando corresponder a um padrão.
NOT	Inverte o valor de qualquer outro operador booliano.
OR	TRUE se qualquer expressão booliana for TRUE.
SOME	TRUE se algum conjunto de comparações for TRUE.

Operadores lógicos UPDATE

O comando UPDATE é responsável por alterar um ou mais registros de uma tabela, dependendo de suas condições e é claro respeitando as restrições da tabela, sintaxe:

```
UPDATE <nome_tabela>
SET <nome_coluna> = <novo_valor>
    [, <nome_coluna1> = <novo_valor1>]
[WHERE <condição>]
```

Fonte: Elaborado pelo autor (2023).

DELETE

O comando DELETE nos permite remover um ou mais registros de uma tabela, sintaxe:

```
DELETE [FROM] <nome_tabela>
[WHERE <condição>]
```

Fonte: Elaborado pelo autor (2023).

Excluindo dados de uma tabela

O comando delete é utilizado para excluir linhas de uma tabela. No exemplo vamos apagar os pedidos que tenham seu número acima de 1000.

Exemplo

DELETE FROM Pedidos WHERE PedidoNumr>=1000

CONDIÇÕES DE FILTRAGEM EM BUSCAS SQL WHERE

A cláusula WHERE permite aplicar uma condição para que seja realizado o comando SQL. O objetivo dele é fazer a filtragem dos dados determinados pelos comandos SELECT, UPDATE e DELETE.A condição é construída através de uma comparação entre dois valores, utilizando os operadores relacionais.

Operadores Relacionais

Para aplicar a condição (ou filtro) podem ser utilizados os operadores de comparação: =, >, <, >=,

Símbolo	Operação
=	Igualdade
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
<>	Diferente

Fonte: Elaborado pelo autor (2023).

<= e <>

(diferente). A tabela mostra os operadores de comparação que são usados:

Exemplos: para saber se um determinado valor está abaixo de 100, podemos utilizar a comparação: valor < 100. Podemos testar se o código é igual

GROUP BY

Os dados resultantes de uma seleção podem ser agrupados de acordo com um critério específico. Este procedimento é realizado usando a cláusula group by.

Exemplo

SELECT COUNT(PedidoNumr) FROM Pedidos GROUP BY PedidoNumr

Junções (join)

Quando precisamos realizar consultas que envolvam mais de uma tabela, uma das soluções seria a utilização de junções. As junções permitem que acessemos mais de uma tabela utilizando apenas um SELECT.

Junção interna (inner Join)

A junção interna entre tabelas é a modalidade de junção que faz com que somente participem da relação resultante as linhas das tabelas de origem que atenderem à cláusula de junção.

SELECT * FROM tabela_Livros INNER JOIN tabela_autores

ON tabela_Livros.ID_Autor = tabela_autores.ID_Autor;

Junções externas (outer join)

Na junção externa, os registros que participam do resultado da junção não obedecem obrigatoriamente à condição de junção, ou seja, a não inexistência de valores correspondentes não limita a participação de linhas no resultado de uma consulta.

Sintaxe:

```
SELECT coluna
FROM tabela_esq
RIGHT (OUTER) JOIN tabela_dir
ON tabela_esq.coluna=tabela_dir.coluna WHERE tabela_esq.coluna IS NULL;
Exemplo:
SELECT * FROM tabela_Livros RIGHT JOIN tabela_editoras
ON tabela_Livros.ID_editora = tabela_editoras.ID_editora WHERE tabela_Livro.ID_editora IS NULL;
```

COMANDO: SELECT...INTO

Esse comando é usado para armazenar o resultado de uma consulta em uma variável.

SELECT Nome, Sobrenome INTO Relacao_Autores FROM tabela_Autores

where ID = ID_Autor

O resultado da consulta deve ter sempre como retorno somente uma linha, caso o resultado tenha mais de uma linha, deve ter o mesmo número de variáveis para receber esses valores.

Funções agregadas

Muitas vezes, precisamos de informações que resultado de alguma operação aritmética ou de conjunto sobre os dados contidos nas tabelas de um banco de dados. Para isso, utilizamos as funções agregadas. A seguir apresentaremos algumas delas.

Função count( )

A função count, como o próprio nome sugere, conta a quantidade de linhas de uma tabela que satisfazem uma determinada condição.

Para contar o número de pedidos que existe na tabela pedidos, pode-se usar o count.

Exemplo

```
SELECT COUNT(PedidoNumr) FROM Pedidos
```

Função avg( )

A função avg é responsável por extrair a média aritmética dos valores de uma coluna. Para calcular a média de pedidos da tabela pedidos, pode-se usar o avg.

Exemplo

```
SELECT AVG(PedidoNumr) FROM Pedidos
```

Função sum( )

A função sum é responsável por realizar a soma dos valores de uma coluna.

Para somar o total de pedidos abaixo de 1000 da tabela pedidos, pode-se usar o sum.

Exemplo

```
SELECT SUM(PedidoNumr) FROM Pedidos WHERE PedidoNumr<1000
```

COMANDO OU OPÇÃO	DESCRIÇÃO
Create schema authorization	Cria um esquema de banco de dados
Create table	Cria uma nova tabela no esquema de banco de dados do usuário
Not null	Assegura que uma coluna não contenha valores nulos
Unique	Assegura que uma coluna não contenha valores duplicados
Primary key	Define a chave primária de uma tabela
Foreign Key	Define a chave estrangeria de uma tabela
Default	Define o valor padrão de uma coluna(quando nenhum valor é fornecido)
Check	Valida os dados de um atributo
Create index	Cria um índice para uma tabela
Create view	Cria um subconjunto dinâmico de linhas/colunas a partir de uma ou mais tabelas
Alter table	Modifica uma definição de tabelas(adiciona, modifica ou exclui atributos ou restrições)
Create table as	Cria nova tabela com base em uma consulta no esquema de banco de dados do usuário
Drop table	Exclui uma tabela(e seus dados) de forma permanente
Drop index	Exclui um índice de forma permanente
Drop view	Exclui uma visuilização de forma permanente

Fonte: Elaborado pelo autor (2023).

CRIAÇÃO DE ÍNDICE

No banco de dados MySQL os índices podem ser criados com considerável facilidade, tanto no momento da concepção da tabela quanto em uma tabela já existente.

CREATE INDEX

CREATE UNIQUE INDEX index_name ON table_name (column1, column2, ...);

Neste exemplo é criada uma tabela chamada CLIENTES com dois campos: Codigo, do tipo inteiro e Nome do tipo texto.

```
CREATE TABLE CLIENTES(
Codigo INT,
Nome     VARCHAR(50),
INDEX (Codigo)
);
```

O índice é criado com o uso da palavra reservada INDEX, seguida do nome da(s) coluna(s) a ser(em) indexada(s). Porém, nem sempre sabemos onde vamos precisar de um índice e muitas vezes é preciso criá-los quando a tabela já existe e inclusive quando já possui registros. Isso pode ser feito com uma instrução DDL (Data Definition Language), como veremos a seguir. Inicialmente criamos a tabela sem índice algum, em seguida adicionamos o índice à coluna “Codigo”.

```
CREATE TABLE CLIENTES(
Codigo INT,
Nome        VARCHAR(50
);
```

Criando o índice separadamente

```
CREATE INDEX idx_CLIENTES_CODIGO ON CLIENTES(Codigo);
```

Nesse caso precisamos definir um nome para o índice (por questão de padronização, alguns profissionais optam por iniciar o nome do índice com um prefixo que indique que ele é um índice, como “id” ou “idx” de “index”, em inglês). Após o nome do índice adicionamos a palavra reservada “ON” que indica em que tabela e coluna o índice será criado, dados que vêm logo em seguida, como vemos na listagem.

Nem sempre o uso de índice trará um bom desempenho, pois a escolha incorreta de um índice pode causar um desempenho insatisfatório. Portanto, a tarefa do otimizador de consulta é selecionar um índice ou uma combinação de índices apenas quando isso gerar melhoria de desempenho e evitar a recuperação indexada quando isso atrapalhar o desempenho.

VIEW

Uma View é um objeto que pertence a um banco de dados, definida baseada em declarações SELECT´s, retornando uma determinada visualização de dados de uma ou mais tabelas.Esses objetos são chamados por vezes de “virtual tables“, formada a partir de outras tabelas que por sua vez são chamadas de “based tables” ou ainda outras Views.

CRIANDO UMA VIEW

```
Create view
CREATE VIEW view_name AS SELECT column1, column2, ... FROM table_name
WHERE condition; ou
CREATE VIEW nome_da_view AS SELECT * FROM nome_tabela;
```

Verificar se a View foi criada SHOW TABLES;

Caso exista uma nova tabela chamada “nome_da_view” foi o nome que definimos para essa nova view; A criação da view foi executada com sucesso.

Alterando uma View

```
ALTER VIEW nome_da_view AS SELECT * FROM nome_outra_tabela;
```

ATIVIDADE DE FIXAÇÃO

1.Qual é o propósito da Linguagem SQL e como ela é utilizada em relação a Sistemas de Gerenciamento de Banco de Dados (SGBD)?

2.Explique as principais categorias de comandos na Linguagem SQL e forneça exemplos de cada uma delas.

3.O que são os comandos DDL, DML, DCL e TCL na Linguagem SQL, e como cada um deles contribui para a manipulação e administração de um banco de dados?

4.Como é realizada a criação de uma tabela no MySQL utilizando o comando CREATE TABLE? Dê um exemplo prático.

5.Explique o papel das restrições em uma tabela MySQL. Quais são alguns exemplos de restrições que podem ser aplicadas a colunas?

6.Como é feita a criação de um banco de dados no MySQL? Quais são os parâmetros que podem ser especificados durante esse processo?

7.Qual a importância das junções (joins) em consultas SQL? Explique a diferença entre junção interna e junções externas.

8.O que são e para que servem as funções agregadas em SQL? Dê exemplos de funções como COUNT, AVG e SUM.

9.Como criar e utilizar índices em MySQL para otimizar o desempenho de consultas?

10.Explique o conceito de Views em SQL. Como criar uma View e como ela pode ser útil em operações de consulta?

PRÁTICA OBRIGATÓRIA: Lembre-se de solicitar ao seu instrutor que execute todos estes exercícios com vocês dentro do MYSQL WORKBENCH. Criem outras situações para usar os comandos vistos até aqui.

TEMA 05

Introdução À Linguagem Sql - (Ddl)

habilidades

- Entender a importância da segurança dos dados.
- Conhecer os comandos de DCL
- Implementar, ver e testar a segurança implantada.
- Aprender a logar no BD com usuários diferentes
- Desenvolver raciocínio e entender o quanto ganhamos com o nível de segurança que o BD coloca em suas informações.

Esquema de banco de dados:

Fonte: Elaborado pelo autor <execução do programa> (2023).

Quando se utiliza um SGBDR empresarial, antes de começar a criar tabelas é necessário ser autenticado por esse sistema. A autenticação é o processo por meio do qual um SGBD garante que somente usuários registrados possam acessar o banco de dados. Para ser autenticado, deve-se fazer login no SGBDR utilizando um ID de usuário e uma senha criada pelo administrador de banco de dados. Em um SGBDR empresarial todo ID de usuário está associado a um esquema de banco de dados.No ambiente de SQL esquema é um grupo de objetos de banco de dados como tabelas e índices relacionados entre si. Geralmente o esquema pertence a um único usuário ou aplicação. Um único banco de dados pode manter vários esquemas que pertençam a vários usuários ou aplicações. Pense no esquema como um agrupamento lógico de objetos de banco de dados, como tabelas, índices e views. Eles são úteis, pois agrupam as tabelas por proprietários (ou função) e aplicam o primeiro nível de segurança permitindo que cada usuário veja apenas as tabelas que lhe pertencem.

Os padrões de SQL ANSI definem um comando para a criação de um esquema de banco de dados:

CREATE SCHEMA AUTHORIZATION {CRIADOR}

Após a criação do esquema de banco de dados, estamos prontos para definir as estruturas das tabelas no banco de dados.

Especificações

Quando for indispensável haver dados disponíveis, a especificação NOT NULL não permitirá que o usuário deixe o atributo vazio (sem nenhuma entrada de dados).

UNIQUE

Restrições são regras que o Mecanismo de Banco de Dados do SQL Server impõe a você. Por exemplo, você pode usar as restrições UNIQUE para garantir que não há valores duplicados inseridos em colunas específicas que não participam de uma chave primária. Embora a restrição UNIQUE e a restrição PRIMARY KEY impõe exclusividade, use a restrição UNIQUE em vez da restrição PRIMARY KEY quando for impor a exclusividade de uma coluna, ou uma combinação de colunas, que não seja uma chave primária.

Diferente das restrições PRIMARY KEY, as restrições UNIQUE permitem o valor NULL. Porém, como com qualquer valor que participa de uma restrição UNIQUE, só um valor nulo é permitido por coluna. Uma restrição UNIQUE pode ser referenciada por uma restrição FOREIGN KEY.

Quando uma nova restrição UNIQUE é adicionada a uma coluna ou colunas existentes em uma tabela, o Banco de Dados, por padrão, examina os dados existentes nas colunas para certificar-se de que todos os valores são únicos. Se uma restrição UNIQUE for adicionada a uma coluna que tem valores duplicados, o mecanismo de Banco de Dados retornará um erro e não adicionará a restrição.

A especificação UNIQUE cria um índice exclusivo no respectivo atributo. Utilize-o para evitar a duplicidade de valores em uma coluna.

SYNTAX:

```
MySQL> create table tempdate (udatetime datetime, utimestamp
timestamp);
Query OK, 0 rows affected (0.32 sec)
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Além dos formatos apresentados na tabela, a SQL dá suporte a muitos outros tipos de dados, incluindo TIME(Hora), TIMESTAMP (registro de data e hora), REAL (número real), DOUBLE (duplo), FLOAT (ponto flutuante), e intervalos como INTERVAL, DAY TO HOUR. Muitos SGBDR´s também expandiram a lista para incluir outros tipos de dados como LOGICAL (lógico), CURRENCY (moeda), AUTONUMBER (numeração automática) e sequência.

Permissões dos usuários

O MySQL é um sistema gerenciador de banco de dados de código aberto que ajuda os usuários a armazenar, organizar e posteriormente recuperar dados.

Ele possui diversas opções para garantir a usuários específicos permissões dentro de tabelas

e bancos de dados.

Quando um usuário é criado, não possui permissão para fazer nada com os bancos de dados. Na verdade, mesmo se tentar fazer login (com a senha password), ele não será capaz de chegar ao shell do MySQL.

CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';

Portanto, a primeira coisa a ser feita é fornecer ao usuário o acesso às informações que eles irão precisar.

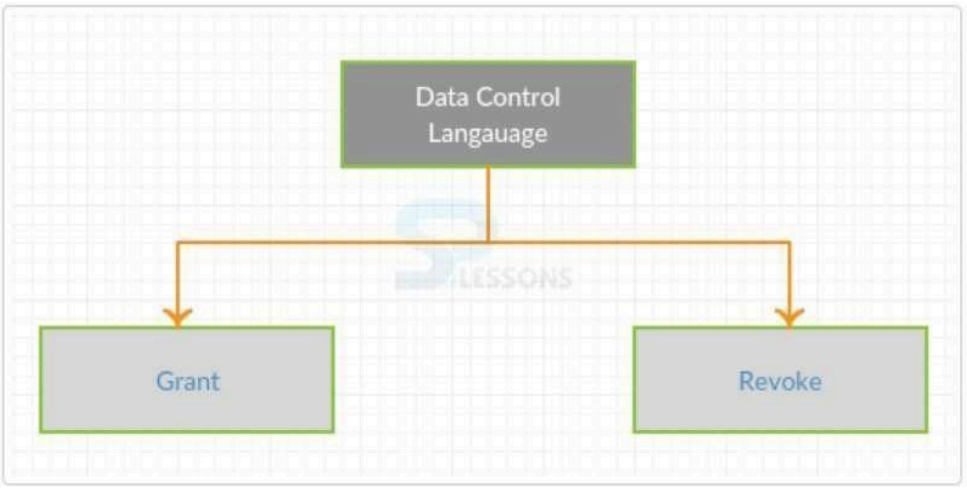
GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';

Os asteriscos do comando referem-se ao banco de dados e tabela que ele pode acessar . Este comando permite que o usuário leia, edite, execute e realize todas as tarefas em todos os bancos de dados e tabelas. Observe que, nesse exemplo, estamos concedendo ao usuário acesso root completo a tudo em nosso banco de dados.

Outras permissões dos usuários

- ALL PRIVILEGES — como vimos anteriormente, isso garante ao usuário do MySQL acesso completo a um banco de dados (ou, se nenhum banco de dados for selecionado, acesso global a todo o sistema)
- CREATE — permite criar tabelas ou bancos de dados
- DROP — permite deletar tabelas ou bancos de dados
- DELETE — permite excluir linhas de tabelas
- INSERT — permite inserir linhas em tabelas
- SELECT - permite usar o comando SELECT para ler os bancos de dados
- UPDATE — permite atualizar linhas de tabelas
- GRANT OPTION — permite conceder ou remover privilégios de outros usuários

Mas vamos ver em detalhes a atuação do DCL.



Grant

Grant é usado para conceder permissões aos clientes. No banco de dados MySQL, ele oferece

ao servidor e ao cliente uma grande quantidade de privilégios de controle. No lado do servidor do procedimento, ele incorpora a possibilidade de o servidor controlar determinados benefícios do cliente sobre o banco de dados MySQL e reduzir suas permissões de conexão do banco de dados ou conceder autorizações limitadas para uma tabela específica.

Concessões em objetos do banco de dados

Esta funcionalidade do comando GRANT concede privilégios específicos sobre um objeto do banco de dados para um ou mais papéis. Estes privilégios são adicionados aos já existentes, caso haja algum.

A palavra-chave PUBLIC indica que os privilégios devem ser concedidos para todos os papéis, inclusive aos que vierem a ser criados posteriormente. PUBLIC pode ser considerado como um grupo definido implicitamente que sempre inclui todos os papéis. Um determinado papel possui a soma dos privilégios concedidos diretamente para ele, mais os privilégios concedidos para todos os papéis que este seja membro, mais os privilégios concedidos para PUBLIC.

Se for especificado WITH GRANT OPTION quem receber o privilégio poderá, por sua vez, conceder o privilégio a terceiros. Sem a opção de concessão, quem recebe não pode conceder o privilégio. A opção de concessão não pode ser concedida para PUBLIC.

Não é necessário conceder privilégios para o dono do objeto (geralmente o usuário que o criou), porque o dono possui todos os privilégios por padrão (Entretanto, o dono pode decidir revogar alguns de seus próprios privilégios por motivo de segurança). O direito de remover um objeto, ou de alterar a sua definição de alguma forma, não é descrito por um privilégio que possa ser concedido; é inerente ao dono e não pode ser concedido ou revogado. O dono possui também, implicitamente, todas as opções de concessão para o objeto.

Dependendo do tipo do objeto, os privilégios padrão iniciais podem incluir a concessão de alguns privilégios para PUBLIC. O padrão é: não permitir o acesso público às tabelas, esquemas e espaços de tabelas; para os bancos de dados conceder o privilégio CONNECT e o privilégio de criação de tabela TEMP; para as funções conceder o privilégio EXECUTE; e para as linguagens conceder o privilégio USAGE. O dono do objeto poderá, é claro, revogar estes privilégios (para a máxima segurança o comando REVOKE deverá ser executado na mesma transação que criar o objeto; dessa forma não haverá espaço de tempo para outro usuário utilizar o objeto).

Os privilégios possíveis são:

SELECT

Permite consultar (SELECT) qualquer coluna da tabela, visão ou sequência especificada. Também permite utilizar o comando COPY TO. Para as sequências, este privilégio também permite o uso da função currval.

INSERT

Permite inserir (INSERT) novas linhas na tabela especificada. Também permite utilizar o comando COPY FROM.

UPDATE

Permite modificar (UPDATE) os dados de qualquer coluna da tabela especificada.

Os comandos SELECT ... FOR UPDATE e SELECT ... FOR SHARE também requerem este privilégio (além do privilégio SELECT). Para as sequências, este privilégio permite o uso das funções nextval e setval.

DELETE

Permite excluir (DELETE) linhas da tabela especificada. REFERENCES

Para criar uma restrição de chave estrangeira é necessário possuir este privilégio, tanto na tabela que faz referência quanto na tabela que é referenciada.

TRIGGER

Permite criar gatilhos na tabela especificada (Consulte o comando CREATE TRIGGER).

CREATE

Para bancos de dados, permite a criação de novos esquemas no banco de dados.

Para esquemas, permite a criação de novos objetos no esquema. Para mudar o nome de um objeto existente é necessário ser o dono do objeto e possuir este privilégio no esquema que o contém. Para espaços de tabelas, permite a criação de tabelas e índices no espaço de tabelas, e permite a criação de bancos de dados possuindo este espaço de tabelas como seu espaço de tabelas padrão (Deve ser observado que revogar este privilégio não altera a colocação dos objetos existentes).

CONNECT

Permite ao usuário se conectar ao banco de dados especificado. Este privilégio é verificado no estabelecimento da conexão (além de serem verificadas as restrições impostas por pg_hba.conf).

TEMPORARY - TEMP

Permite a criação de tabelas temporárias ao usar o banco de dados. EXECUTE

Permite utilizar a função especificada e qualquer operador implementado utilizando a função. Este é o único tipo de privilégio aplicável às funções (Esta sintaxe funciona para as funções de agregação também).

USAGE

Para as linguagens procedurais, permite o uso da linguagem especificada para criar funções nesta linguagem. Este é o único tipo de privilégio aplicável às linguagens procedurais.Para os esquemas, permite acessar os objetos contidos no esquema especificado (assumindo que os privilégios requeridos para os próprios objetos estejam atendidos). Essencialmente, concede a quem recebe o direito de "procurar" por objetos dentro do esquema. Sem esta permissão ainda é possível ver os nomes dos objetos, por exemplo consultando as tabelas do sistema. Além disso, após esta permissão ter sido revogada os servidores existentes poderão conter comandos que realizaram anteriormente esta procura, portanto esta não é uma forma inteiramente segura de impedir o acesso aos objetos.

ALL PRIVILEGES

Concede todos os privilégios disponíveis de uma só vez. A palavra chave PRIVILEGES é requerida pelo SQL estrito. Os privilégios requeridos por outros comandos estão listados nas páginas de referência dos respectivos comandos.

Sintaxe

```
GRANT {ALL | statement [ ,…n ] }
```

Exemplo

Visualizando o exemplo abaixo, o conceito de comando grant pode ser facilmente entendido.

```
mysql>grant select on sample.* to reader@localhost identified by 'secret';
Query OK,0 rows affected
mysql>exit;
Bye
D:\MySQL\bin>mysql -u reader -p
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| employee           |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| world              |
+--------------------+
7 rows in set (0.00 sec)
mysql> select * from employee;
+--------+----------+-------+
| emp_id | emp_name |salary |
+--------+----------+-------+
|   1001 | mike     | 12000 |
|   1002 | maze     | 13000 |
|   1003 | jack     | 14000 |
+--------+----------+-------+
3 rows in set (0.04 sec)
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Descrição: O comando revoke cancela todas as permissões do usuário. Sintaxe:

REVOKE

O comando REVOKE revoga, de um ou mais papéis, privilégios concedidos anteriormente. A palavra- chave PUBLIC se refere ao grupo contendo todos os usuários, definido implicitamente. O significado dos tipos de privilégio deve ser visto na descrição do comando GRANT.

Deve ser observado que um determinado papel possui a soma dos privilégios concedidos diretamente para o próprio papel, mais os privilégios concedidos para os papéis dos quais o papel é membro no momento, mais os privilégios concedidos para PUBLIC.

Daí, por exemplo, revogar o privilégio SELECT de PUBLIC não significa, necessariamente, que todos os papéis perderão o privilégio SELECT para o objeto: os papéis que receberam o privilégio diretamente, ou através de outro papel, ainda terão o privilégio.

Se for especificado GRANT OPTION FOR somente a opção de concessão do privilégio é revogada, e não o próprio privilégio. Caso contrário, tanto o privilégio quanto a opção de concessão serão revogados.

Se o usuário possui um privilégio com opção de concessão, e concedeu este privilégio para outros usuários, então os privilégios que estes outros usuários possuem são chamados de privilégios dependentes. Se o privilégio ou a opção de concessão que o primeiro usuário possui for revogada, e existirem privilégios dependentes, estes privilégios dependentes também serão revogados se for especificado CASCADE, senão a ação de revogar falhará.

Esta revogação recursiva somente afeta os privilégios que foram concedidos através de uma cadeia de usuários começando pelo usuário objeto deste comando REVOKE. Portanto, os usuários afetados poderão manter o privilégio, se o privilégio também tiver sido concedido por outros usuários.

Ao revogar o privilégio de membro de um papel, GRANT OPTION passa a se chamar ADMIN OPTION, mas o comportamento é semelhante.

Deve ser observado, também, que esta forma do comando não inclui a palavra GROUP.

<priv_type> [<column_list>]

[priv_type [<column_list>]] … ON [object_type] priv_level FROM user [user] …

REVOKE ALL PRIVILEGES, GRANT OPTION

FROM user [user] …

CREATE USER

Usada para criar um usuário no sistema (sem privilégios) Sintaxe:

CREATE USER usuário@host IDENTIFIED BY 'senha';

host é o nome do host a partir de onde o usuário pode se conectar ao banco de dados; geralmente usamos localhost para a máquina local. Se não for especificado um host, o MySQL acrescentará automaticamente o símbolo % como nome do host, o que significa que o usuário poderá se conectar de qualquer lugar. É possível também usar o endereço IP de um host (por exemplo, 127.0.0.1 para o host local).

Após a criação do usuário, ele não terá nenhum privilégio em nenhum banco de dados. Os privilégios podem ser atribuídos por meio da declaração **GRANT**, que estudaremos na próxima lição.

Exemplos:

1. Criando um usuário de nome “fabio” com senha “1234” no MySQL, com acesso a partir do host local:

CREATE USER fabio@localhost IDENTIFIED BY '1234';

Verificando se o usuário foi criado como especificado:

SELECT User, Host FROM mysql.user;

```
mysql> CREATE USER fabio@localhost IDENTIFIED BY '1234';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT User, Host FROM mysql.user;
+------------------+-----------+
| User             | Host      |
+------------------+-----------+
| root             | 127.0.0.1 |
| root             | ::1       |
| root             | debian    |
| debian-sys-maint | localhost |
| fabio            | localhost |
| root             | localhost |
+------------------+-----------+
6 rows in set (0.00 sec)
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

2. Criando um usuário de nome “ana” com acesso a partir de qualquer local:

CREATE USER ana IDENTIFIED BY "1234";

Fonte: Elaborado pelo autor <execução do programa> (2023).

Note que na coluna Host aparece o símbolo **%** para a usuária ana, significando que ela pode acessar o SGBD de qualquer local.

3. Criando um usuário de nome marcos sem senha definida no momento:

CREATE USER marcos@localhost;

Para alterar ou configurar uma senha para esse usuário posteriormente, use o comando **SET PASSWORD**: **SET PASSWORD FOR 'marcos'@'localhost' = PASSWORD('1234');**

Alteramos a senha do usuário **marcos** para **1234** com essa declaração.

Nas versões mais recentes do MySQL, a declaração SET PASSWORD foi deprecada, e não será mais utilizada nas próximas versões.

RENAME USER

Usada para renomear um usuário do MySQL. Se o usuário possuir privilégios configurados, eles são mantidos para o novo nome de usuário.

Sintaxe:

RENAME USER nome_atual TO novo_nome;

Exemplo:

1. Vamos renomear a usuária ana para monica:

RENAME USER ana TO monica;

DROP USER

Usada para excluir um usuário do MySQL. Esta declaração elimina o usuário e seus privilégios do sistema.

Sintaxe: **DROP USER nome_usuário;**

Exemplo:

Vamos remover a usuária monica: **DROP USER monica;**

Visualizando o exemplo abaixo, o conceito de comando revogar pode ser facilmente entendido.

```
mysql> SHOW GRANTS FOR 'david'@'localhost';
+-----------------------------------------------------------------------+
| Grants for david@localhost                                            |
+-----------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'david'@'localhost' WITH GRANT OPTION |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)mysql> REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'david'@
Query OK, 0 rows affected (0.00 sec)
mysql> show revoke for 'david'@'localhost';
ERROR 1064 (42000): there is no localhost user
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

ATIVIDADE DE FIXAÇÃO

1. O que é DCL?
2. Quais são os comandos DCL?
3. O que é o comando grant?
4. Explique a funcionalidade do comando revoke grant option.
5. Apresente a sintaxe e um exemplo do comando revoke.
6. Explique o que é a possibilidade de ALL PRIVILEGES.
7. Crie um comando DCL utilizando a funcionalidade all privileges.
8. Crie um comando Create user.
9. Crie um comando drop
10. Explique o comando rename

TEMA 06

Conectividade Em Python

habilidades

- Criar uma compreensão básica do Python
- Entender o papel dos conectores.
- Perceber que os comandos ao Banco de Dados podem vir de outra Linguagem, dependendo do nível de integração dos Conectores.

Fonte do vídeo: https://youtu.be/cYre7NNyTkg

Você necessita de um banco de dados para salvar informações em seus programas do Python, os principais que podem ser acessados por ele são: MySQL, SQLite, PostgreSQL e Interbase/Firebird. Falaremos sobre o MySQL. Primeiro temos que construir conexão com o MySQL. O seguinte irá se conectar ao banco de dados MySQL no programa Python.

```
import pymysql

#database connection
connection =
pymysql.connect(host="localhost",user="root",passwd="",database="databaseName" )
cursor = connection.cursor()
# some other statements  with the help of cursor
connection.close()
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Primeiro nós importamos o pyMySQL, então estabelecemos uma conexão. O pyMySQL.connect () leva quatro argumentos. O primeiro é o nome do host, ou seja, o localhost, e os três restantes são como são declarados. Usando esta conexão, criamos um cursor que será usado para diferentes consultas.

Para criar uma conexão com o banco de dados, use o nome de usuário e a senha do seu banco de dados MySQL:

import mysql.connector

mydb = mysql.connector.connect(host="localhost", user="yourusername", password="yourpassword"

)

print(mydb)

Criando um banco de dados

Para criar um banco de dados no MySQL, use uma instrução "CREATE DATABASE": Exemplo

Crie um banco de dados chamado "mydatabase": import mysql.connector

mydb = mysql.connector.connect(

host="localhost", user="yourusername",

password="yourpassword"

)

mycursor = mydb.cursor() mycursor.execute("CREATE DATABASE mydatabase")

Verifique se existe banco de dados

Você pode verificar se existe um banco de dados listando todos os bancos de dados em seu sistema usando uma instrução "SHOW DATABASES":

Exemplo

Retorne uma lista dos bancos de dados do seu sistema:

import mysql.connector

mydb = mysql.connector.connect(host="localhost", user="*yourusername*", password="*yourpassword*"

)

mycursor = mydb.cursor() mycursor.execute("SHOW DATABASES")

for x in mycursor: print(x)

Ou você pode tentar acessar o banco de dados ao fazer uma conexão:

Exemplo

Tente se conectar ao banco de dados "meu banco de dados": import mysql.connector

mydb = mysql.connector.connect(host="localhost",user="yourusername", password="yourpassword", **database=**"mydatabase"

)

Python MySQL creating table

Vamos agora criar uma tabela chamada Artist com colunas - name, id e track

```
import pymysql

#database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="databaseName")
cursor = connection.cursor()
# Query for creating table
ArtistTableSql = """CREATE TABLE Artists(
ID INT(20) PRIMARY KEY AUTO_INCREMENT,
NAME  CHAR(20) NOT NULL,
TRACK CHAR(10))"""

cursor.execute(ArtistTableSql)
connection.close()
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Python MySQL insert

Agora nosso interesse é inserir algumas entidades de linha na tabela. Primeiro você tem que escrever as consultas para inserir dados diferentes e, em seguida, executá-lo com a ajuda do cursor.

```
import pymysql

#database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="databaseName")
cursor = connection.cursor()

# queries for inserting values
insert1 = "INSERT INTO Artists(NAME, TRACK) VALUES('Towang', 'Jazz' );"
insert2 = "INSERT INTO Artists(NAME, TRACK) VALUES('Sadduz', 'Rock' );"

#executing the quires
cursor.execute(insert1)
cursor.execute(insert2)


#commiting the connection then closing it.
connection.commit()
connection.close()
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Python MySQL select

Nós inserimos duas linhas no código acima. Agora queremos recuperá-los. Para fazer isso, dê uma olhada no seguinte exemplo:

```
import pymysql

#database connection
connection = pymysql.connect(host="localhost", user="root", passwd="",
database="databaseName")
cursor = connection.cursor()

# queries for retrievint all rows
retrive = "Select * from Artists;"

#executing the quires
cursor.execute(retrive)
rows = cursor.fetchall()
for row in rows:
   print(row)


#commiting the connection then closing it.
connection.commit()
connection.close()
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

```
D:\Software\Python\Python36-32\python.exe D:/T_Code/Pythongenerator/package2/mySql.py
(1, 'Towang', 'Jazz')
(2, 'Sadduz', 'Rock')

Process finished with exit code 0
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Selecione com um filtro

Ao selecionar registros de uma tabela, você pode filtrar a seleção usando a instrução "WHERE": Exemplo

Selecione o (s) registro (s) em que o endereço seja "Park Lane 38": resultado:

import mysql.connector

mydb = mysql.connector.connect(host="localhost", user="yourusername",

password="yourpassword", database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address ='Park Lane 38'" mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:

print(x)

Caracteres curinga

Você também pode selecionar os registros que constam, incluindo ou terminam com uma determinada letra ou frase.

Use o % para representar caracteres curinga:

Exemplo

Selecione os registros em que o endereço contenha a palavra "caminho": import mysql.connector

mydb = mysql.connector.connect(host="localhost", user="yourusername", password="yourpassword", database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address LIKE '%way%'" mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:

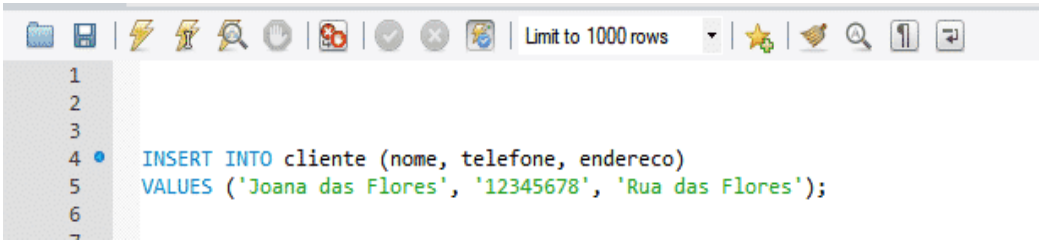
print(x)

CRUD - (Create, Read, Update e Delete)

Pode ser traduzido como: criar, ler, atualizar e excluir, engloba os principais comandos da linguagem SQL (Structured Query Language) para a manipulação de dados, que são: INSERT (inserir), READ (ler), UPDATE (alterar) e DELETE (remover). As interfaces CRUD permitem cadastrar (create), visualizar (read), editar (update) e excluir (delete) registros de um sistema.

INSERT – CREATE

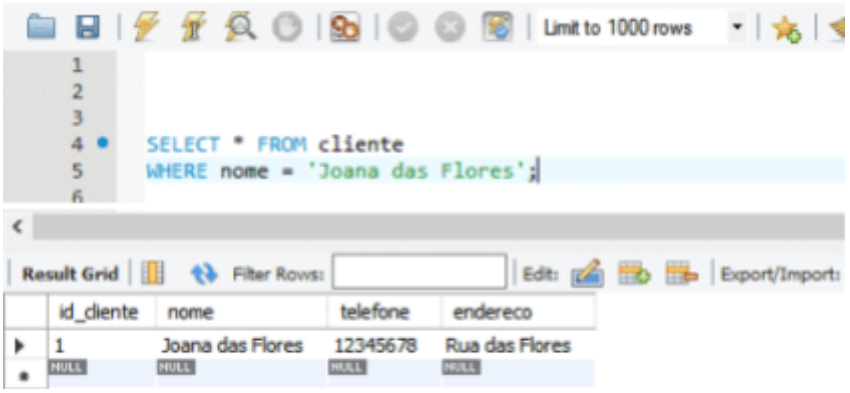
Esse comando é responsável por inserir dados na tabela. Assim, toda vez que você quiser adicionar algo novo, você precisa usar o comando INSERT, seguido do campo e do valor que você quer adicionar.



Fonte: Elaborado pelo autor <execução do programa> (2023).

SELECT – READ

O SELECT é de longe o comando mais utilizado do CRUD. Justamente porque é ele quem traz todo o resultado da busca que você quer saber. Assim, ele traz para você exatamente o que você quer saber quando pesquisa algo no banco de dados.



Fonte: Elaborado pelo autor <execução do programa> (2023).

UPDATE

O comando UPDATE tem por finalidade a atualização dos dados que você inseriu na tabela. Se por algum motivo você quer modificar um campo, atualizar o valor ou acrescentar novos dados, é o UPDATE que você vai utilizar.

Python MySQL Update

```
updateSql = "UPDATE  Artists SET NAME= 'Tauwang'  WHERE ID = '1' ;"
cursor.execute(updateSql  )
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Exemplo

UPDATE cliente SET endereco = ‘Rua da Flor’ WHERE nome = ‘Joana das Flores’;

Para atualizar um campo, basta especificar qual campo você vai modificar, com o valor antigo pelo valor que você quer substituir.

DELETE

Por fim temos o comando DELETE que simplesmente exclui o dado especificado. Muito cuidado, pois, uma vez excluído não tem Ctrl + z para voltar no tempo e desfazer o erro.

Python MySQL Delete

```
deleteSql = "DELETE FROM Artists WHERE ID = '1'; "
cursor.execute(deleteSql )
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Exemplo

DELETE from cliente WHERE nome = ‘Joana das Flores’;

Python MySQL – Drop Table

```
dropSql = "DROP TABLE IF EXISTS Artists;"
cursor.execute(dropSql)
```

Fonte: Elaborado pelo autor <execução do programa> (2023).

Impedir injeção de SQL

Quando os valores da consulta são fornecidos pelo usuário, você deve escapar dos valores.

Isso evita injeções de SQL, que é uma técnica comum de hacking para destruir ou usar indevidamente seu banco de dados.

O módulo mysql.connector tem métodos para escapar dos valores da consulta: Exemplo

Escape dos valores da consulta usando o %s método de placholder : import mysql.connector

mydb = mysql.connector.connect(host="localhost", user="yourusername", password="yourpassword", database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address = %s" adr = ("Yellow Garden 2",)

mycursor.execute(sql, adr) myresult = mycursor.fetchall()

for x in myresult: print(x)

Limite o resultado

Você pode limitar o número de registros retornados da consulta, usando uma instrução "LIMIT":

Exemplo

Selecione os 5 primeiros registros na tabela "clientes": import mysql.connector

mydb = mysql.connector.connect(host="localhost", user="yourusername", password="yourpassword", database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5") myresult = mycursor.fetchall()

for x in myresult:

print(x)

Comece de outra posição

Se você deseja retornar cinco registros, a partir do terceiro registro, você pode usar uma palavra-chave "OFFSET":

Exemplo

Comece na posição 3 e retorne 5 registros:

```
import mysql.connector
mydb = mysql.connector.connect( host="localhost", user="yourusername",
password="yourpassword", database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers LIMIT 5 OFFSET 2") myresult =
mycursor.fetchall()
for x in myresult:
print(x)
```

ATIVIDADE DE FIXAÇÃO

1. O que é o Python?

2. Por que é uma ferramenta interessante de se usar com MySQL?

3. Crie um comando para inserir uma linha usando Python e MySQL.

4. Crie um comando para selecionar os dados inseridos usando Python e MySQL.

5. Crie um comando para atualização de dados usando Python e MySQL.

6. Crie um comando para deletar um objeto de banco de dados usando Python e MySQL.

7. O que é o CRUD em Python e MySQL.

8. Apresente um exemplo de CRUD.

9. Explique como criar um banco de dados MySQL e Python.

10. Como limitar buscas?


Proz
Viva sua profissão!