UVV

Melhor Universidade Particular do Brasil *(MEC)*

THE Times Higher Education

Pacto Nacional pela Primeira Infância

universidades empreendedoras

QS WORLD UNIVERSITY RANKINGS

AD Scientific Index

Prêmio Sebrae de Educação Empreendedora

CENTELHA

QS

THE IMPACT RANKINGS

DESIGN E DESENVOLVIMENTO DE BANCO DE DADOS I
UNIDADE V

Prof. Msc. Gustavo Nunes Rocha

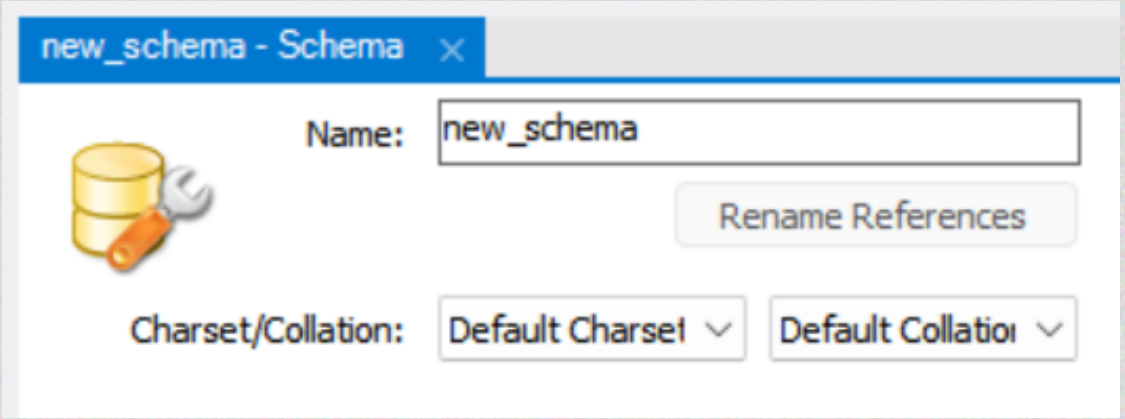
Definições Preliminares

- Na tela inicial do MySQL Workbench, clique em "New Connection" (Nova Conexão) para criar uma nova conexão com o servidor MySQL.
- Preencha os detalhes de conexão:
- Connection Name: Dê um nome para essa conexão (por exemplo, "MeuServidor").
- Hostname: Geralmente, você usará "localhost" para conexões locais.
- Port: Deixe como está (geralmente é 3306).
- Username: Insira o nome de usuário do MySQL (por exemplo, "root").
- Password: Insira a senha associada ao usuário.

Criando o schema

- Após se conectar ao servidor, clique com o botão direito em "Data Connections" (Conexões de Dados) na barra lateral esquerda e escolha "Create Schema" (Criar Esquema).
- Schema Name: Insira um nome para o banco de dados (por exemplo, "MinhaBaseDeDados").
- Default Collation: Se necessário, escolha uma colação padrão para o banco de dados. A colação define como a ordenação e comparação de strings serão realizadas. Você pode deixar como está se não tiver necessidade específica.

Criando o schema



Usando o assistente do Workbench

```
1    CREATE SCHEMA `exemplo` DEFAULT CHARACTER SET utf8mb4 ;
```

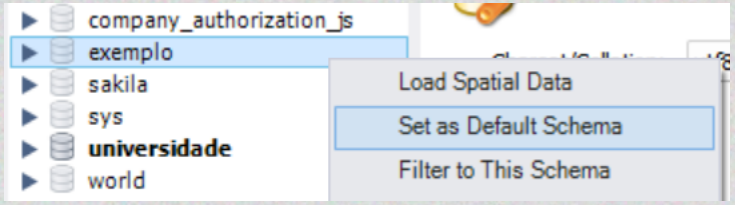
Usando o script em SQL

Definições Preliminares

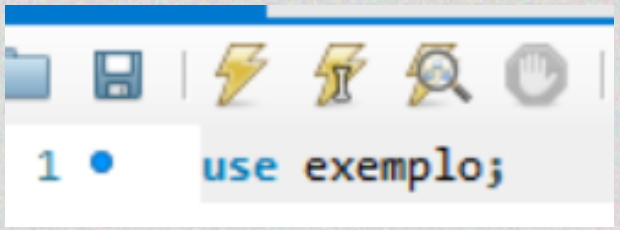
- A criação de tabelas é uma parte fundamental ao projetar e criar um banco de dados utilizando o MySQL. O comando CREATE TABLE é utilizado para definir a estrutura de uma nova tabela, especificando os nomes das colunas, seus tipos de dados, restrições e outros atributos.

Criando as tabelas

- Com o banco de dados criado, clique com o botão direito sobre ele na barra lateral e selecione "Set as Default Schema" (Definir como Esquema Padrão) para que todas as operações subsequentes ocorram no contexto desse banco de dados. Ou clique no ícone de relâmpago na parte superior para abrir um novo script.



Usando o assistente do Workbench



Usando o script em SQL

Criando as tabelas

- Você pode agora escrever e executar os comandos SQL diretamente na aba de consulta.Vamos usar o comando CREATE TABLE para criar uma tabela. Aqui está um exemplo de comando:
- CREATE TABLE clientes (
- id INT PRIMARY KEY,
- nome VARCHAR(50),
- idade INT,
- email VARCHAR(100) UNIQUE
-);
- Ou pode ser um pouco mais complexo que isso...

Criando as tabelas

```
CREATE TABLE (IF NOT EXISTS) nome_tabela (
    coluna1 tipo_dado (RESTRIÇÕES),
    coluna2 tipo_dado (RESTRIÇÕES),
    ...
    (RESTRIÇÕES),
    (ÍNDICES)
)
```


Clausulas de criação de tabelas

- Na criação de tabelas no MySQL, você tem várias opções e comandos disponíveis para definir a estrutura da tabela, suas colunas e as restrições. Aqui estão alguns dos comandos mais comuns que podem ser usados na criação de tabelas:
- CREATE TABLE: O comando principal para criar uma nova tabela.
- IF NOT EXISTS: Opcional. Evita que ocorra um erro caso a tabela já exista.
- nome_tabela: O nome da tabela que você está criando.

- CREATE TABLE (IF NOT EXISTS) nome_tabela (...

Clausulas de criação de tabelas

- coluna1 tipo_dado (RESTRIÇÕES),
- coluna2 tipo_dado (RESTRIÇÕES),
- ...
- coluna_nome tipo_dado: Define o nome da coluna e o tipo de dado que ela irá armazenar. Cada tipo de dado tem um propósito específico e é escolhido com base no tipo de informações que a coluna irá armazenar.
- (RESTRIÇÕES): As restrições são usadas para definir regras e limitações nas colunas de uma tabela, a fim de garantir a consistência e a integridade dos dados armazenados.

Tipos de dados numéricos

- INT: Número inteiro (4 bytes).
- TINYINT: Número inteiro pequeno (1 byte).
- SMALLINT: Número inteiro pequeno (2 bytes).
- MEDIUMINT: Número inteiro médio (3 bytes).
- BIGINT: Número inteiro grande (8 bytes).
- FLOAT: Número de ponto flutuante (precisão simples).
- DOUBLE: Número de ponto flutuante (dupla precisão).
- DECIMAL(p, s): Número decimal com precisão p e escala s.

Tipos de dados de texto

- CHAR(n): String de comprimento fixo de até n caracteres.
- VARCHAR(n): String de comprimento variável de até n caracteres.
- TINYTEXT: Texto pequeno (até 255 caracteres).
- TEXT: Texto normal (até 65,535 caracteres).
- MEDIUMTEXT: Texto médio (até 16,777,215 caracteres).
- LONGTEXT: Texto longo (até 4,294,967,295 caracteres).
- ENUM('valor1', 'valor2', ...): Enumeração de valores possíveis.
- SET('valor1', 'valor2', ...): Conjunto de valores possíveis.

Tipos de dados temporais

- DATE: Data (formato 'YYYY-MM-DD').
- TIME: Hora (formato 'HH:MM:SS').
- DATETIME: Data e hora (formato 'YYYY-MM-DD HH:MM:SS').
- TIMESTAMP: Data e hora com fuso horário.
- YEAR: Ano (formato 'YYYY').

Tipos de dados binários

- BINARY(n): Dados binários de comprimento fixo de até n bytes.
- VARBINARY(n): Dados binários de comprimento variável de até n bytes.
- TINYBLOB: Dados binários pequenos (até 255 bytes).
- BLOB: Dados binários normais (até 65,535 bytes).
- MEDIUMBLOB: Dados binários médios (até 16,777,215 bytes).
- LONGBLOB: Dados binários longos (até 4,294,967,295 bytes).

Outro Tipos de dados

- BOOL / BOOLEAN: Booleano (1 para verdadeiro, 0 para falso).
- BIT: Tipo de bit.
- JSON: Armazenamento de dados JSON.
- GEOMETRY: Dados geométricos.

Primary key (chave primária)

- A chave primária (primary key) é uma restrição usada em bancos de dados para identificar exclusivamente cada registro em uma tabela. Ela garante que não haja duplicatas e permite um acesso rápido e eficiente aos dados. No MySQL, a chave primária é definida durante a criação da tabela usando a cláusula PRIMARY KEY.

```
CREATE TABLE alunos (
    matricula INT PRIMARY KEY,
    nome VARCHAR(100),
    data_nascimento DATE
);
```


foreign key

Uma chave estrangeira (foreign key) é uma restrição em um banco de dados que estabelece uma relação entre duas tabelas, referenciando uma coluna (ou conjunto de colunas) em uma tabela como sendo a mesma que a chave primária em outra tabela. Isso cria um relacionamento entre os dados nas duas tabelas e ajuda a manter a integridade referencial dos dados.

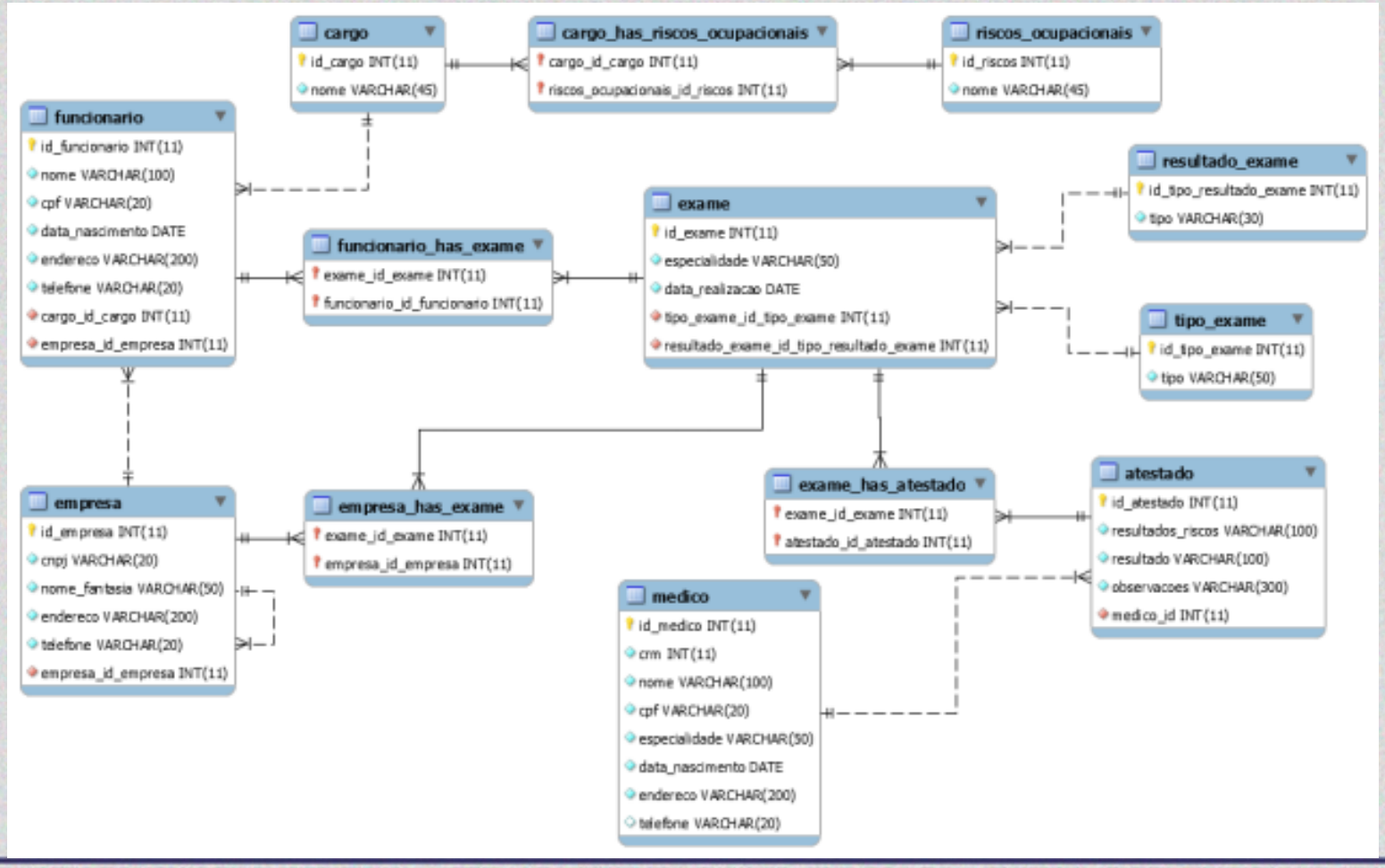
```
CREATE TABLE pedidos (
    id INT PRIMARY KEY,
    cliente_id INT,
    data_pedido DATE,
    FOREIGN KEY (cliente_id) REFERENCES clientes(id)
);
```


Not null

- A restrição NOT NULL é usada para garantir que uma coluna não contenha valores nulos (NULL). Isso significa que cada registro inserido na tabela deve ter um valor válido na coluna que possui a restrição NOT NULL. Essa restrição é especialmente útil para garantir a integridade dos dados e evitar que informações essenciais estejam ausentes.

```
CREATE TABLE produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    preco DECIMAL(10, 2) NOT NULL
);
```


Exercícios – crie os comandos sql para criação da tabela descrita abaixo:



unique

- A restrição UNIQUE é usada para garantir que os valores em uma coluna ou conjunto de colunas sejam únicos em relação aos valores em outras linhas da mesma tabela. Isso evita que dados duplicados sejam inseridos em uma tabela e ajuda a manter a integridade dos dados.

```
CREATE TABLE emails (
    id INT PRIMARY KEY,
    endereco VARCHAR(255) UNIQUE
);
```


default

- A cláusula DEFAULT é usada para definir um valor padrão para uma coluna em uma tabela. Isso significa que, se nenhum valor for especificado durante a inserção de um novo registro, a coluna receberá automaticamente o valor padrão especificado. Isso é útil quando você deseja fornecer um valor predefinido para uma coluna, caso o valor não seja explicitamente fornecido.

```
CREATE TABLE tarefas (
    id INT PRIMARY KEY,
    descricao VARCHAR(200),
    data_conclusao DATE DEFAULT '2023-12-31'
);
```


Auto increment

- A opção AUTO_INCREMENT é frequentemente usada no MySQL para atribuir automaticamente valores únicos e incrementais a uma coluna, normalmente usada como chave primária. Ela simplifica a inserção de registros, pois o valor é gerado automaticamente pelo sistema e evita a necessidade de especificar um valor para essa coluna durante as inserções.

```
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    email VARCHAR(255)
);
```


On delete / on update

- A cláusula ON DELETE é usada em conjunto com a criação de uma chave estrangeira (FOREIGN KEY) no MySQL para especificar o que acontece com os registros na tabela filho quando um registro correspondente na tabela pai é excluído. A cláusula ON DELETE oferece opções para controlar o comportamento de exclusão em cascata, nulo ou restrito.
- A cláusula ON UPDATE é usada em conjunto com a criação de uma chave estrangeira (FOREIGN KEY) no MySQL para especificar o que acontece com os registros na tabela filho quando um registro correspondente na tabela pai é atualizado. A cláusula ON UPDATE oferece opções para controlar o comportamento de atualização em cascata, nulo ou restrito.

On delete / on update

- CASCADE: Quando um registro na tabela pai é substituído ou excluído, os registros correspondentes na tabela filho também são automaticamente substituídos ou excluídos.
- SET NULL: Quando um registro na tabela pai é é substituído ou excluído, os valores das colunas correspondentes na tabela filho são definidos como NULL.
- SET DEFAULT: Similar ao SET NULL, mas os valores das colunas correspondentes na tabela filho são definidos como o valor padrão especificado.
- RESTRICT: Impede a exclusão ou atualização na tabela pai se houver registros correspondentes na tabela filho.

On delete / on update

- CASCADE: Quando um registro na tabela pai é substituído ou excluído, os registros correspondentes na tabela filho também são automaticamente substituídos ou excluídos.
- SET NULL: Quando um registro na tabela pai é é substituído ou excluído, os valores das colunas correspondentes na tabela filho são definidos como NULL.
- SET DEFAULT: Similar ao SET NULL, mas os valores das colunas correspondentes na tabela filho são definidos como o valor padrão especificado.
- RESTRICT: Impede a exclusão ou atualização na tabela pai se houver registros correspondentes na tabela filho.

On delete / on update

```
CREATE TABLE autores (
    id INT PRIMARY KEY,
    nome VARCHAR(100)
);
CREATE TABLE livros (
    id INT PRIMARY KEY,
    titulo VARCHAR(200),
    autor_id INT,
    FOREIGN KEY (autor_id) REFERENCES autores(id) ON DELETE CASCADE ON UPDATE SET NULL
);
```


Index

- Um índice no MySQL é uma estrutura de dados que melhora a velocidade de recuperação de registros de uma tabela. Ele atua como um mecanismo de pesquisa que permite ao banco de dados localizar registros rapidamente com base nos valores das colunas indexadas.

```
CREATE TABLE produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    preco DECIMAL(10, 2),
    categoria_id INT,
    INDEX idx_categoria (categoria_id));
```


Index

- Quando você cria um índice em uma coluna, o MySQL mantém um "mapa" que associa os valores dessa coluna aos locais físicos dos registros na tabela, facilitando as operações de busca. Mas quais são as vantagens disso:
- Melhora o desempenho das consultas SELECT.
- Acelera operações de JOIN, ORDER BY e GROUP BY.
- Permite que o banco de dados encontre registros com base em valores específicos de colunas de forma mais eficiente.

Índice b-tree

- O tipo de índice mais comum no MySQL.
- Eficiente para consultas de igualdade e intervalos (ex: coluna = valor, coluna BETWEEN valor1 AND valor2).
- Também suporta ordenação (ORDER BY).
- Pode ser usado em colunas individuais ou combinações de colunas.

Índice b-tree

```
CREATE TABLE produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    preco DECIMAL(10, 2),
    categoria_id INT,
    INDEX idx_categoria (categoria_id)
);
```

- Neste exemplo, estamos criando uma tabela de produtos e criando um índice B-Tree chamado idx_categoria na coluna categoria_id. Isso pode melhorar o desempenho de consultas que filtram ou agrupam registros com base na categoria.

Índice hash

- Apropriado para consultas de igualdade exata (coluna = valor).
- Mapeia valores para posições de armazenamento usando uma função de hash.
- Não suporta ordenação.
- Não é adequado para consultas que envolvem intervalos ou ordenação.

Índice hash

```
CREATE TABLE usuarios (
    id INT PRIMARY KEY,
    nome VARCHAR(50),
    email VARCHAR(100),
    INDEX idx_email (email) USING HASH
);
```

Neste exemplo, estamos criando uma tabela de usuários e criando um índice Hash chamado idx_email na coluna email. Isso é útil para consultas de igualdade exata em emails, mas não é adequado para ordenação.

Índice de texto completo

- Projetado para pesquisas de texto completo em colunas de texto (VARCHAR, TEXT).
- Pode executar pesquisas mais complexas, como palavras-chave, frases e proximidade.
- Normalmente usado para implementar funcionalidades de pesquisa avançada em aplicativos que lidam com texto.

Índice de texto completo

```
CREATE TABLE posts (
    id INT PRIMARY KEY,
    titulo VARCHAR(200),
    corpo TEXT,
    FULLTEXT idx_pesquisa (titulo, corpo));
```

Neste exemplo, estamos criando uma tabela de posts e criando um índice de Texto Completo chamado idx_pesquisa nas colunas titulo e corpo. Isso permitirá a realização de pesquisas avançadas em texto completo.

Índice espacial

- Usado para consultas baseadas em localização geográfica, como encontrar pontos próximos, calcular distâncias, etc.
- Suporta tipos de dados espaciais (ex: POINT, POLYGON) e funções de geometria.
- Útil em aplicativos que trabalham com dados de localização, como aplicativos de mapeamento ou geolocalização.

Índice espacial

```
CREATE TABLE locais (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    coordenadas POINT,
    SPATIAL INDEX idx_local (coordenadas));
```

Neste exemplo, estamos criando uma tabela de locais e criando um índice espacial chamado idx_local na coluna coordenadas. Isso permite a realização de consultas baseadas em localização geográfica.

collation

- No MySQL, uma colação (ou collation) define como os dados de texto são classificados e comparados. As colações são importantes porque determinam a ordem em que as strings são organizadas alfabeticamente e como as comparações de strings são feitas, incluindo a distinção entre maiúsculas e minúsculas. Existem vários tipos de colações disponíveis no MySQL, e cada uma delas tem um propósito específico. Podem também ser feitos na tabela:
- CREATE TABLE usuarios (
- id INT PRIMARY KEY,
- nome VARCHAR(50)
- CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
-);

Collation – mais usados

- utf8_general_ci (ou utf8mb4_general_ci): Essa colação é popular para bancos de dados que precisam suportar caracteres multilíngues e emojis. Ela é insensível a maiúsculas e minúsculas e considera caracteres acentuados e variantes como iguais. O utf8mb4 é uma versão mais recente que suporta um conjunto de caracteres mais amplo, incluindo emojis.
- utf8_unicode_ci (ou utf8mb4_unicode_ci): Semelhante à colação utf8_general_ci, essa colação também é insensível a maiúsculas e minúsculas, mas fornece um método mais preciso para ordenar caracteres em diferentes idiomas. Ela é especialmente útil quando se lida com idiomas que têm regras de ordenação específicas.

Collation – mais usados

- latin1_swedish_ci: Uma colação para conjuntos de caracteres latinos (como ISO 8859-1) que é insensível a maiúsculas e minúsculas. É adequada para idiomas que usam o alfabeto latino.
- utf8_bin (ou utf8mb4_bin): Essa colação considera as strings como sequências de bytes exatas, incluindo diferenças de maiúsculas e minúsculas. Ou seja, 'A' e 'a' são tratados de forma diferente.
- binary: Essa colação é semelhante ao utf8_bin, mas é usada para caracteres binários. Ela também considera as strings como sequências de bytes exatas.

Collation – mais usados

- utf8_general_cs (ou utf8mb4_general_cs): Essa colação é semelhante ao utf8_general_ci, mas é sensível a maiúsculas e minúsculas. Ela considera 'A' e 'a' como diferentes.
- utf8_spanish_ci (ou utf8mb4_spanish_ci): Uma colação otimizada para idioma espanhol que leva em consideração as regras de ordenação específicas do idioma.

engine

- A cláusula ENGINE é usada para definir o motor de armazenamento que será usado para uma tabela no MySQL. O motor de armazenamento determina como os dados serão armazenados e gerenciados internamente pela tabela. Cada motor de armazenamento possui características diferentes que se adequam a diferentes cenários e necessidades. Aqui estão alguns dos motores de armazenamento mais comuns no MySQL:
- InnoDB: O motor de armazenamento padrão do MySQL a partir da versão 5.5. Ele oferece suporte a transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), bloqueio de linha, chaves estrangeiras e recuperação de falhas. É adequado para a maioria dos cenários de aplicativos.

engine

- MyISAM: Um motor de armazenamento mais antigo que não oferece suporte a transações ACID nem a chaves estrangeiras. No entanto, é mais rápido para leitura e é adequado para cenários em que a recuperação de falhas não é uma preocupação crítica.
- MEMORY (ou HEAP): Armazena os dados em memória RAM, tornando as operações de leitura extremamente rápidas. No entanto, os dados são voláteis e serão perdidos quando o servidor for reiniciado.
- CSV: Armazena os dados em formato CSV (valores separados por vírgula), o que pode ser útil para importação/exportação de dados.

engine

- ARCHIVE: Projetado para armazenar grandes quantidades de dados de maneira eficiente em termos de espaço, mas não é adequado para operações de atualização frequente.
- BLACKHOLE: Aceita dados de gravação, mas não os armazena. Pode ser usado para replicação ou redirecionamento de dados.
- FEDERATED: Permite acessar dados de tabelas em servidores remotos como se estivessem locais.
- NDB (Cluster): Projetado para alta disponibilidade e escalabilidade em clusters de servidores MySQL.

Row format

A cláusula ROW FORMAT é usada para definir o formato de armazenamento das linhas em uma tabela, determinando como os dados são organizados e armazenados nas páginas de dados do banco de dados. Existem vários formatos de linha disponíveis no MySQL, e a escolha do formato pode afetar o desempenho e o consumo de espaço do banco de dados.

```
CREATE TABLE produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    descricao TEXT,
    preco DECIMAL(10, 2)
) ENGINE=InnoDB
ROW_FORMAT=DYNAMIC;
```


Row format

- DEFAULT: Este é o formato de linha padrão usado pelo mecanismo de armazenamento específico. Por exemplo, o InnoDB usará o formato de linha padrão do InnoDB, que é otimizado para desempenho e consumo de espaço.
- DYNAMIC: Este formato de linha é usado pelo mecanismo InnoDB. Ele armazena os dados das colunas fixas no cabeçalho da página e os dados das colunas variáveis no corpo da página. Isso permite um consumo de espaço mais eficiente para tabelas com muitas colunas variáveis.
- COMPRESSED: Também usado pelo mecanismo InnoDB, o formato de linha comprimido aplica compactação aos dados da linha para economizar espaço em disco. Isso pode melhorar significativamente o consumo de espaço, mas pode exigir mais recursos de CPU para compressão e descompressão.

Row format

- REDUNDANT: Um formato de linha usado pelo mecanismo MyISAM, que armazena os dados da linha em uma estrutura redundante que oferece suporte a recuperação de falhas e recuperação rápida.
- COMPACT: Outro formato de linha usado pelo mecanismo MyISAM, que armazena os dados de forma compacta para economizar espaço em disco.
- PAGE: Usado pelo mecanismo MEMORY, armazena cada linha em uma página de memória separada.

comment

A cláusula COMMENT é usada no MySQL para adicionar um comentário ou uma descrição a uma tabela, coluna ou até mesmo a um banco de dados. O comentário pode ser usado para fornecer informações adicionais sobre a finalidade, estrutura ou qualquer outra coisa relevante para o objeto em questão.

```
CREATE TABLE usuarios (
    id INT PRIMARY KEY,
    nome VARCHAR(50),
    email VARCHAR(100),
    data_criacao TIMESTAMP,
    COMMENT 'Tabela para armazenar informações dos usuários.');
```


comment

O uso de comentários pode ajudar a tornar seu esquema de banco de dados mais compreensível e facilitar a colaboração entre membros da equipe. Lembre-se de que os comentários não afetam o funcionamento do banco de dados, mas fornecem informações valiosas para aqueles que trabalham com o esquema de banco de dados.

```
CREATE TABLE produtos (
    COMMENT 'Este é um banco de dados de exemplo para demonstração.';
    id INT PRIMARY KEY,
    nome VARCHAR(100) COMMENT 'Nome do produto',
    preco DECIMAL(10, 2) COMMENT 'Preço do produto em reais.');
```

Partition by

A cláusula PARTITION BY é usada no MySQL para definir a partição de uma tabela em várias partes (partições) com base em um critério específico. A partição é uma técnica de gerenciamento de tabelas que divide grandes conjuntos de dados em partes menores, facilitando o gerenciamento, a manutenção e a consulta de dados.

```
CREATE TABLE vendas (
(...)
) PARTITION BY RANGE (YEAR(data_venda)) (
    PARTITION p2019 VALUES LESS THAN (2020),
    PARTITION p2020 VALUES LESS THAN (2021),
    PARTITION p2021 VALUES LESS THAN (2022)
);
```

Partition by

- Cada partição é tratada como uma "subtabela" separada com sua própria estrutura de armazenamento. Algumas razões para Particionar Tabelas:
- Melhorar o desempenho de consultas, pois apenas as partições relevantes precisam ser lidas.
- Facilitar a manutenção de grandes conjuntos de dados.
- Possibilitar a remoção ou arquivamento de partições antigas.
- O particionamento é uma estratégia avançada de gerenciamento de dados que pode melhorar significativamente o desempenho e a manutenção de grandes tabelas. No entanto, é importante planejar e testar cuidadosamente o particionamento, pois ele pode ter impactos significativos na estrutura da tabela e nas operações de consulta.

Partition by range ou intervalo

- As partições são definidas com base em intervalos de valores. Neste exemplo, a tabela vendas é particionada por intervalo com base no ano da coluna data_venda. Cada partição abrange um ano específico.

```
PARTITION BY RANGE (YEAR(data_venda)) (
    PARTITION p2019 VALUES LESS THAN (2020),
    PARTITION p2020 VALUES LESS THAN (2021),
    PARTITION p2021 VALUES LESS THAN (2022));
```


Partition by list ou lista

A cláusula PARTITION BY LIST é usada no MySQL para particionar uma tabela com base em uma lista predefinida de valores de coluna. Isso significa que cada partição é criada para armazenar um conjunto específico de valores da coluna especificada. O particionamento por lista é útil quando você deseja agrupar registros com valores específicos em partições separadas.

```
PARTITION BY LIST (departamento) (
    PARTITION p_engenharia VALUES IN ('Engenharia'),
    PARTITION p_vendas VALUES IN ('Vendas'),
    PARTITION p_marketing VALUES IN ('Marketing'),
    PARTITION p_outros VALUES IN (DEFAULT));
```

Partition by list ou lista

- Cada partição agrupa os funcionários de um departamento específico.
- p_engenharia: Partição para funcionários do departamento de Engenharia.
- p_vendas: Partição para funcionários do departamento de Vendas.
- p_marketing: Partição para funcionários do departamento de Marketing.
- p_outros: Partição para qualquer outro departamento que não esteja especificado.
- O particionamento por lista é útil quando você tem um conjunto discreto de valores de coluna pelos quais deseja particionar seus dados. Isso pode melhorar a eficiência das consultas ao acessar partições específicas diretamente, em vez de verificar toda a tabela. Certifique-se de planejar seu particionamento cuidadosamente e entender os padrões de acesso aos dados para aproveitar ao máximo essa técnica.

Partition by hash ou tabela hash

- A cláusula PARTITION BY HASH é usada no MySQL para particionar uma tabela usando uma função de hash aplicada aos valores das colunas selecionadas. O particionamento por hash é uma técnica que distribui os registros de forma uniforme entre várias partições com base em um valor calculado por uma função de hash. Isso pode ser útil quando você deseja distribuir os dados de maneira equilibrada em várias partições.
- CREATE TABLE ordens (
- id INT PRIMARY KEY,
- numero_pedido VARCHAR(20),
- data_pedido DATE
-) PARTITION BY HASH(id) PARTITIONS 4;

Partition by hash ou tabela hash

- Neste exemplo, estamos criando uma tabela ordens particionada por hash com base na coluna id. O número total de partições é definido como 4. Isso significa que o MySQL aplicará uma função de hash ao valor da coluna id e distribuirá os registros em 4 partições diferentes com base nos resultados da função de hash.
- O particionamento por hash é útil quando você deseja distribuir uniformemente os registros em várias partições, o que pode ser benéfico para balancear a carga em sistemas distribuídos ou para melhorar a eficiência das operações de leitura e gravação em tabelas grandes.
- No entanto, lembre-se de que o particionamento por hash pode fazer com que os dados relevantes para uma consulta específica estejam distribuídos em várias partições, o que pode afetar o desempenho das consultas. Portanto, é importante entender bem os padrões de acesso aos dados ao usar essa técnica.

Partition by key ou chave

A cláusula PARTITION BY KEY é usada no MySQL para particionar uma tabela com base em uma expressão de chave, que geralmente se refere à coluna de chave primária da tabela. O particionamento por chave é semelhante ao particionamento por hash, mas é otimizado para uso com chaves primárias ou únicas. Ele garante que as chaves primárias ou únicas sejam distribuídas uniformemente entre as partições, o que pode ajudar a melhorar o desempenho e a escalabilidade em ambientes de banco de dados distribuídos.

```
CREATE TABLE produtos (
    id INT PRIMARY KEY,
    nome VARCHAR(100),
    categoria_id INT
) PARTITION BY KEY();
```


Partition by key ou chave

- Neste exemplo, estamos criando uma tabela produtos particionada por chave, onde a chave primária id é usada como a expressão de particionamento. O MySQL distribuirá uniformemente os registros com base nas chaves primárias entre as partições.
- O particionamento por chave é particularmente útil em cenários de escalabilidade horizontal, onde os dados são distribuídos entre diferentes servidores de banco de dados ou nós. Ele pode ajudar a evitar gargalos de desempenho e melhorar a distribuição da carga entre as partições. No entanto, assim como com outras técnicas de particionamento, é importante entender bem os padrões de acesso aos dados e as características do seu aplicativo antes de implementar o particionamento por chave.

Fontes de conhecimento

- Documentação Oficial do MySQL: A fonte mais confiável e completa para informações sobre MySQL é a sua documentação oficial. Ela abrange uma variedade de tópicos, desde conceitos básicos até recursos avançados.
- Site: https://dev.mysql.com/doc/

- MySQL Tutorial no W3Schools: Um tutorial online abrangente que cobre os conceitos básicos do MySQL, incluindo consultas, inserções, atualizações e muito mais.
- Site: https://www.w3schools.com/mysql/

Fontes de conhecimento

- MySQL Developer Zone: Este é o portal de desenvolvedores do MySQL, oferecendo recursos, blogs e artigos relacionados ao desenvolvimento e administração do MySQL.
- Site: https://dev.mysql.com/

- MySQL Performance Blog:
- Um blog focado em otimização e desempenho do MySQL, com informações valiosas sobre como melhorar a velocidade e eficiência do banco de dados.
- Site: https://www.percona.com/blog/

Fontes de conhecimento

- MySQL Forums: Os fóruns oficiais do MySQL são um ótimo lugar para fazer perguntas, receber ajuda e aprender com a comunidade MySQL.
- Site: https://forums.mysql.com/

- Curso Online Gratuito da Oracle: MySQL for Beginners: Um curso online introdutório oferecido pela Oracle que abrange os conceitos básicos do MySQL.
- Site: https://education.oracle.com/mysql-beginners

UVV

universidadevilavelha

uvvoficial

universidadevilavelha