COMANDOS MYSQL WORKBENCH

Criar Banco de Dados

Para criar um novo banco de dados:
CREATE DATABASE [IF NOT EXISTS] nome_BD;
O elemento IF NOT EXISTS é opcional. Ele previne o erro de tentar criar um banco de dados que já existe no servidor. Não é possível ter dois bancos de dados com o mesmo nome.

Exemplo:
CREATE DATABASE db_MeusLivros;

Podemos verificar os bancos de dados existentes com o comando SHOW DATABASES;

Comando USE

O comando USE instrui o SGBDR a utilizar o banco de dados especificado para rodar os comandos. Sintaxe:
USE nome_banco_de_dados;
Para visualizar o banco de dados selecionado no momento use o comando:
SELECT DATABASE();

Excluir Banco de Dados

Podemos excluir um banco de dados existente com o comando DROP DATABASE, seguido do nome do banco:
DROP DATABASE [IF EXISTS] nome_BD;

Comentários

Para inserir um comentário em seu código SQL, use um dos estilos a seguir:
Comentário até o final da linha /* Bloco de comentários que ocupa várias linhas */

Selecionar Banco

Alternativamente, clique com o botão direito sobre o nome do banco de dados desejado no painel SCHEMA e escolha a opção Set as Default Schema

Tipos de Dados:

Tipo de Dado é um conjunto finito de valores, nomeado, ou seja, todos os valores possíveis de uma variedade específica. Por exemplo, todos os números inteiros

possíveis, ou todas as combinações de caracteres. Todo valor pertence a algum e apenas um tipo. Toda variável, atributo e parâmetro são declarados como sendo de um tipo.

Tipos em MySQL

O MySQL suporta as seguintes categorias de tipos de dados:

- Lógico
- Inteiro
- Ponto flutuante
- Ponto Fixo
- Caractere
- Data e Hora
- Binários (string de bits)

Tipos inteiros

TINYINT- números inteiros de -128 a 127
SMALLINT Números inteiros de -32768 a 32767
MEDIUMINT Números inteiros de -8.388.608 a 8.388.607
INT Números inteiros entre -2.147.483.648 e 2.147.483.647
BIGINT Números inteiros entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807

Tipos de Ponto Flutuante

FLOAT(P) (ocupa 4 bytes) Ponto flutuante com precisão P. o padrão é 10,2; P vai de 0 a 23. Valores de 1.175494351E-38 a 3.402823466E+38 (sem sinal); ou de -3.402823466E+38 a -1.175494351E-38 (com sinal).
DOUBLE(P) (precisão dupla; ocupa 8 bytes) Valores de 2.2250738585072014E-308 a 1.7976931348623157E+308 (sem sinal); ou de - 1.7976931348623157E+308 a -2.2250738585072014E-308 (com sinal).

Tipos de Ponto Fixo (Precisão Exata)

DECIMAL(M,D) DECIMAL(M) Ponto fixo com M dígitos no total (precisão) e D casas decimais (escala); o padrão de M é 10; M vai até 65 e D até 30.
NUMERIC Implementado como DECIMAL no MySQL (são iguais)
Usados para preservar precisão exata, por exemplo para dados monetários.

Tipos de Caractere

CHAR(M) Cadeia de caracteres que ocupa um tamanho fixo entre 0 e 255 caracteres, indicados pelo valor de M
VARCHAR(M) String (cadeia de caracteres) de tamanho variável, com até 65.535 caracteres, indicados pelo valor de M.
Obs.: O tamanho máximo de uma linha (registro) em uma tabela é de 65.535 caracteres. Se este tamanho for excedido ao somar todas as colunas, deve-se alterar campos VARCHAR para BLOB ou TEXT.

Tipos BIT e Booleano

BOOL / BOOLEAN Valores binários 0 ou 1; Não é um tipo realmente; na verdade, é um alias para o tipo TINYINT(1). Zero é considerado “falso” e outro valor qualquer, “verdadeiro”.
BIT(n) Armazenar uma sequência de bits (valor binário) com até n valores. O valor n pode variar entre 1 e 64, e o padrão é 1 bit, se nenhum valor for especificado.

Tipos de Data e Hora

DATE Data de 01/01/1000 a 31/12/9999, no formato YYYY-MM-DD
DATETIME Combinação de data e hora de 01/01/1000 00:00:00 a 31/12/9999 23:59:59, no formato YYYY-MM-DD HH:MM:SS
TIME Hora apenas, no formato HH:MM:SS
YEAR(M) Ano nos formatos de 2 ou 4 dígitos; Se forem 2 (YEAR(2)), ano vai de 1970 a 2069; para 4 (YEAR(4)), vai de 1901 a 2155. O padrão é 4.
TIMESTAMP Valores são convertidos do fuso horário local para UTC ao armazenar o dado, e convertidos de volta de UTC para horário local ao ler o dado.

Tipos BLOB

TINYBLOB Tamanho máximo de 255 bytes de dados
BLOB Tamanho máximo de 65.535 bytes de dados
MEDIUMBLOB Tamanho máximo de 16.777.215 bytes de dados
LONGBLOB Tamanho máximo de 4.294.967.295 bytes de dados

Tipos TEXT

TINYTEXT Tamanho máximo de 255 caracteres (255 bytes de texto)
TEXT Tamanho máximo de 65.535 caracteres (64 KB de texto)
MEDIUMTEXT Tamanho máximo de 16.777.215 caracteres (16 MB de texto)
LONGTEXT Tamanho máximo de 4.294.967.295 caracteres (4 GB de texto)

Tipos de Conjuntos de Valore

ENUM Tipo no qual um valor é escolhido a partir de uma lista de valores fornecidos quando da criação da tabela. Por exemplo, lista com os nomes dos meses do ano.
SET Objeto que possui zero ou mais valores separados por vírgula (no máximo 64), também escolhidos a partir de uma lista de valores fornecidos durante a criação da tabela.

Outros Tipos

- BINARY
- VARBINARY
- Tipos Espaciais
- GEOMETRY
- POLYGON
- LINESTRING
- MULTIPOINT
- JSON

Chaves

● Uma chave consiste em uma ou mais colunas de uma tabela cujos valores são usados para identificar de forma exclusiva uma linha ou conjunto de linhas.
● Pode ser única (identifica uma só linha) ou não-única (identifica um conjunto de linhas).

Classificação das Chaves

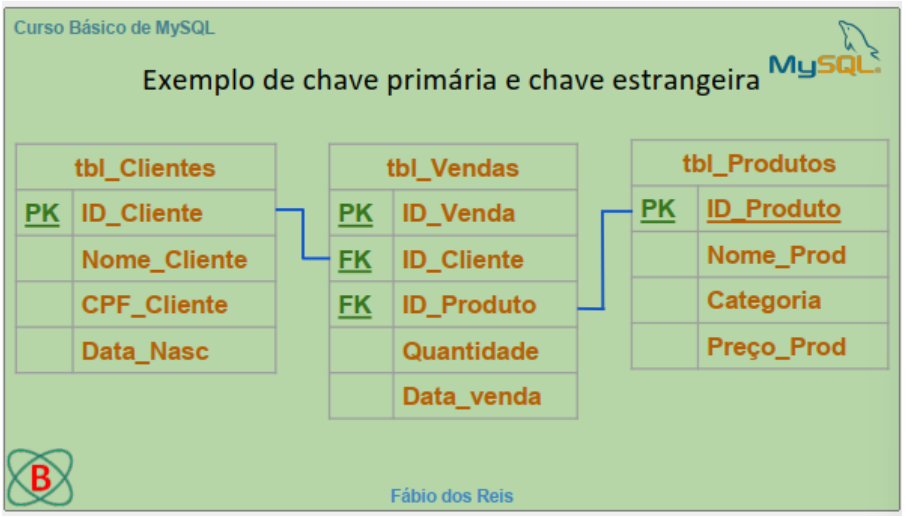
Classificamos as chaves em dois tipos:
● Chaves Únicas:
●Candidata
●Composta
●Primária
●Surrogada
● Chave Não-Única: Estrangeira

Chave Primária

● Primary Key / PK
● Chave candidata escolhida para ser a chave principal na relação.
● Identifica de forma exclusiva os registros em uma tabela, não sendo permitida a repetição de valores nem tampouco o valor nulo.
● Cada tabela pode ter uma única chave primária.

Chave Estrangeira

● Coluna de uma tabela que estabelece um Relacionamento com a Chave Primária (PK) ou coluna unique de outra tabela.
● É a partir da chave estrangeira (Foreign key / FK) que sabemos com qual registro em outra tabela um registro está relacionado.



Chave Composta

● Chave que é constituída por duas ou mais colunas.
● Geralmente empregada quando não é possível utilizar uma única coluna de uma tabela para identificar de forma exclusiva seus registros.

Chave Surrogada / Substituta

● Valor numérico, único, adicionado a uma tabela para servir como chave primária.
● Não possui significado para usuários e geralmente fica escondida nas aplicações.
● Chaves substitutas são frequentemente usadas no lugar de uma chave primária composta.

Constraints (Restrições)

● Restrições são regras aplicadas nas colunas de uma tabela.
● Usadas para limitar os tipos de dados que são inseridos.
● Podem ser especificadas no momento de criação da tabela (CREATE) ou após a tabela ter sido criada (ALTER)

Principais Constraints

● NOT NULL
● UNIQUE
● PRIMARY KEY
● FOREIGN KEY
● DEFAULT
● CHECK

NOT NULL

● A constraint NOT NULL impõe a uma coluna a NÃO aceitar valores NULL (nulos).
● Obriga um campo a sempre possuir um valor.
● Deste modo, não é possível inserir um registro (ou atualizar) sem entrar com algum valor neste campo.

UNIQUE

● A restrição UNIQUE identifica de forma única cada registro em uma tabela.
● UNIQUE e PRIMARY KEY garantem a unicidade em uma coluna ou conjunto de colunas.
● Uma constraint PRIMARY KEY automaticamente possui uma restrição UNIQUE definida.
● Você pode ter várias constraints UNIQUE em uma tabela, mas apenas uma PK.

PRIMARY KEY

● A chave primária identifica de forma única cada registro em uma tabela de banco de dados.
● Chaves Primárias devem conter valores únicos.
● Uma coluna de chave primária não pode conter valores NULL
● Cada tabela deve ter uma chave primária e apenas uma chave primária.

FOREIGN KEY

● Coluna de uma tabela que estabelece um Relacionamento com a Chave Primária (PK) ou coluna unique de outra tabela.
● É a partir da chave estrangeira (Foreign key / FK) que sabemos com qual registro em outra tabela um registro está relacionado.

DEFAULT

● A restrição DEFAULT é usada para inserir um valor padrão em uma coluna.
● O valor padrão será adicionado a todos os novos registros caso nenhum outro valor seja especificado.

CHECK

● A restrição CHECK garante que os valores em uma coluna satisfaçam a um determinado critério.
● Um novo registro só é aceito se o valor inserido na coluna com a restrição CHECK satisfizer ao critério estabelecido.
● Por exemplo, podemos impedir a inserção de números negativos em uma coluna de preços de itens.

Auto Incremento

● O auto incremento permite que um número único seja gerado quando um novo registro é inserido na tabela.
● Assim, não é necessário especificar valor para uma coluna de auto incremento ao inserir novos dados - o valor é gerado e inserido automaticamente.
● Só é permitido usar uma coluna de auto incremento por tabela

Banco de Dados Relacional

● Um Banco de dados relacional é uma coleção de relações, que são tabelas bidimensionais, onde os dados são armazenados.
● Como exemplo, podemos querer armazenar dados sobre os clientes de uma loja. Para isso, criamos tabelas para guardar diferentes conjuntos de dados relacionados a esses clientes, como dados pessoais, dados de compras, crédito, e outras.

Criar tabelas

Para criar tabelas, usamos o comando CREATE TABLE:

CREATE TABLE [IF NOT EXISTS] nome_tabela (
coluna1 tipo1 restrições, coluna2 tipo2 restrições, colunaN tipoN restrições,
restrições extras);

Criar tabela de Autores

CREATE TABLE tbl_Autores (
IdAutor SMALLINT AUTO_INCREMENT, NomeAutor VARCHAR(50) NOT NULL, SobrenomeAutor VARCHAR(60) NOT NULL, CONSTRAINT pk_id_autor PRIMARY KEY (IdAutor));

Verificar Campos da Tabela

Também podemos verificar a estrutura da tabela criada (nomes dos campos, tipos de dados, chaves, etc) rodando o comando DESCRIBE:

DESCRIBE tbl_autores;

Outro comando equivalente é o SHOW COLUMNS FROM:

SHOW COLUMNS FROM tbl_autores;

Criar tabela de Editoras

CREATE TABLE tbl_Editoras (
IdEditora SMALLINT PRIMARY KEY AUTO_INCREMENT,
NomeEditora VARCHAR(50) NOT NULL);

Criar tabela de Assuntos

CREATE TABLE tbl_Assuntos (IdAssunto Tinyint AUTO_INCREMENT, Assunto VARCHAR(25) NOT NULL, CONSTRAINT pk_id_assunto PRIMARY KEY (IdAssunto));

Criar tabela de Livros

CREATE TABLE IF NOT EXISTS tbl_Livros (
IDLivro SMALLINT NOT NULL AUTO_INCREMENT,
NomeLivro VARCHAR(70) NOT NULL, ISBN13 VARCHAR(13) NOT NULL, DataPub DATE, PrecoLivro DECIMAL(10,2) NOT NULL, NumeroPaginas SMALLINT NOT NULL, IdEditora SMALLINT NOT NULL, IdAssunto Tinyint NOT NULL, CONSTRAINT pk_id_livro PRIMARY KEY (IDLivro), CONSTRAINT fk_id_editora FOREIGN KEY (IdEditora) REFERENCES tbl_Editoras (IdEditora) ON DELETE

CASCADE, CONSTRAINT fk_id_assunto FOREIGN KEY (IdAssunto) REFERENCES tbl_Assuntos (IdAssunto) ON DELETE CASCADE
);

Criar tabela LivrosAutores
CREATE TABLE tbl_LivrosAutores (IdLivro SMALLINT NOT NULL, IdAutor SMALLINT NOT NULL, CONSTRAINT pk_id_livro_autor PRIMARY KEY (IdLivro, IdAutor), CONSTRAINT fk_id_livros FOREIGN KEY (IdLivro) REFERENCES tbl_Livros (IdLivro), CONSTRAINT fk_id_autores FOREIGN KEY (IdAutor) REFERENCES tbl_Autores (IdAutor)
);

Verificar tabelas

Podemos ver as tabelas criadas no banco de dados rodando o comando SHOW TABLES:
SHOW TABLES;

Renomear Tabelas

Para renomear uma tabela já criada no MySQL usamos o comando RENAME TABLE:
RENAME TABLE tabela1 TO tabela2;
Este comando também permite renomear views no banco.

Exemplo: Vamos criar uma nova tabela em nosso banco de dados, de nome Emprestimo:
CREATE TABLE Emprestimo (
IdEmprestimo TINYINT PRIMARY KEY, NomeLivro VARCHAR (60) NOT NULL, NomeEmprestador VARCHAR (50) NOT NULL, DataEmprestimo DATE NOT NULL, Duracao TINYINT NOT NULL);

Renomear Tabelas

A tabela foi criada com o nome errado. Está sem nosso prefixo padronizado tbl_. Podemos ver os nomes das tabelas do banco com o comando SHOW TABLES:
SHOW TABLES;
Então, vamos renomeá-la para tbl_Emprestimos:
RENAME TABLE Emprestimo TO tbl_Emprestimos;
E conferimos o resultado com SHOW TABLES:
SHOW TABLES;

Alterar Tabelas

● É possível alterar a estrutura de uma tabela após ter sido criada, acrescentando, excluindo ou modificando colunas (campos)
● Usamos para isso o comando ALTER TABLE

Adicionar Colunas - ADD

Adicionar colunas a uma tabela existente - Sintaxe:
ALTER TABLE tabela
ADD coluna tipo_dados constraints;
Exemplo:
ALTER TABLE tbl_Livros ADD Edicao TINYINT;

Excluir Colunas - DROP

Sintaxe para excluir colunas:
ALTER TABLE tabela DROP COLUMN coluna;
Exemplo:
ALTER TABLE tbl_Livros DROP COLUMN Edicao;

Adicionar Chave Primária

ALTER TABLE tabela ADD PRIMARY KEY (coluna)
Obs. A coluna deve existir antes de ser transformada em chave primária.
Essa coluna receberá a Constraint (restrição) PRIMARY KEY, e passará a ser a chave primária da tabela.

Adicionar Chave Estrangeira

ALTER TABLE tbl_Livros
ADD CONSTRAINT fk_IdAutor FOREIGN KEY (IdAutor) REFERENCES tbl_Autores (IdAutor);

Alterar Colunas Existentes

Podemos alterar colunas, por exemplo mudando o tipo de dados:
ALTER TABLE tbl_Emprestimos
MODIFY COLUMN IdEmprestimo SMALLINT;
E verificar a alteração com o comando DESCRIBE:
DESCRIBE tbl_Emprestimos;

Configurar Auto Incremento

Podemos configurar o auto incremento em uma tabela já criada, iniciando em um valor escolhido, com a seguinte sintaxe:

ALTER TABLE tbl_Livros AUTO_INCREMENT=100;

Execute este comando para ajustar o auto incremento na tabela de livros, conforme especifica a regra de negócio.

DROP TABLE – Excluir tabela

Podemos excluir tabelas de um banco de dados com o comando DROP TABLE. Sintaxe:

DROP TABLE IF EXISTS tabela1, tabela2, ...;

Exemplo:

DROP TABLE tbl_Emprestimos;

Inserir Dados - INSERT INTO

Inserimos novos registros em uma tabela com a declaração INSERT INTO. Sintaxe:

INSERT INTO tabela (coluna1, coluna2,...) VALUES (valor1, valor2,...);

Inserir Dados – Tabela de Autores

INSERT INTO tbl_autores (NomeAutor, SobrenomeAutor) VALUES ('Daniel', 'Barret'), ('Gerald', 'Carter'), ('Mark', 'Sobell'), ('William', 'Stanek'), ('Richard', 'Blum'), ('Christine', 'Bresnahan'), ('Richard', 'Silverman'), ('Robert', 'Byrnes'), ('Jay', 'Ts'), ('Robert', 'Eckstein'), ('Paul', 'Horowitz'), ('Winfield', 'Hill'), ('Joel', 'Murach'), ('Paul', 'Scherz'), ('Simon', 'Monk');

Inserir Dados – Tabela de Editoras

INSERT INTO tbl_editoras (NomeEditora) VALUES ('Prentice Hall'), ('O´Reilly'), ('Microsoft Press'), ('Wiley'), ('Mc Graw Hill'), ('Bookman'), ('Novatec'), ('Apress'), ('Sybex'), ('Mike Murach and Associates');

Inserir Dados – Tabela de Assuntos

INSERT INTO tbl_Assuntos (Assunto) VALUES ('Ficção'), ('Botânica'), ('Eletrônica'), ('Matemática'), ('Aventura'), ('Literatura'), ('Informática'), ('Suspense');

Inserir Dados – Tabela de Livros

INSERT INTO tbl_Livros (NomeLivro, ISBN13, DataPub, PrecoLivro, NumeroPaginas, IdAssunto, IdEditora)

VALUES
('Linux Command Line and Shell Scripting','9781118983843', '20150109', 165.55, 816, 7, 4), ('SSH, the Secure Shell','9780596008956', '20050517', 295.41, 672, 7, 2),
('Using Samba','9780596002565', '20031221', 158.76, 449, 7, 2),
('A Arte da Eletrônica', '9788582604342', '20170308', 176.71, 1160, 7, 6),
('Windows Server 2012 Inside Out','9780735666313', '20130125', 179.51, 1584, 7, 3), ('Murach´s MySQL','9781943872367', '20190501', 227.64, 650, 7, 10),
('Practical Electronics for Inventors', '9781259587542', '20160711', 119.58, 1056, 3, 5);

Inserir Dados – Tabela LivrosAutores
INSERT INTO tbl_LivrosAutores (IdLivro, IdAutor) VALUES (100,5), (100,6), (101,1), (101,7), (101,8), (102,2), (102,9), (102,10), (103,11), (103,12), (104,4), (105,13), (106,14), (106,15);

Verificar Dados Inseridos
SELECT * FROM tbl_Autores; SELECT * FROM tbl_Editoras; SELECT * FROM tbl_Assuntos; SELECT * FROM tbl_Livros; SELECT * FROM tbl_LivrosAutores;

Verificar Relacionamentos
SELECT L.NomeLivro AS Livro,
CONCAT(A.NomeAutor, ' ', A.SobrenomeAutor) AS Autor, L.PrecoLivro As 'Preço do Livro'
FROM tbl_livrosautores LA INNER JOIN tbl_livros L ON L.IdLivro = LA.IdLivro INNER JOIN tbl_Autores A ON A.IdAutor = LA.IdAutor;

Atualizar Registros
Cláusula UPDATE
UPDATE tabela SET coluna = novo_valor WHERE coluna = valor_filtro;
Obs.: Caso não seja usada a cláusula WHERE para filtrar os registros, todos os dados da coluna serão alterados.

Atualizar Registros - Exemplo
UPDATE tbl_Livros SET NomeLivro = 'SSH, o Shell Seguro' WHERE IdLivro = 101;
NomeLivro = Nome da coluna Apenas as linhas cujo IdLivro for igual a 101 terão o valor alterado.

Exemplo 02
UPDATE tbl_Livros SET PrecoLivro = PrecoLivro + 10 WHERE IdLivro = 105;

Exemplo 03
UPDATE tbl_Assuntos SET Assunto = 'Biologia' WHERE IdAssunto = 2;

Excluir Registros
Sintaxe:
DELETE FROM tabela WHERE coluna = valor;
Obs.: Sempre use a cláusula WHERE para evitar a perda de dados da tabela!

Excluir Registros - Exemplo
Vamos inserir uma nova editora na tabela de editoras:
INSERT INTO tbl_Editoras (NomeEditora) VALUES ('Bóson Editorial');
Verificamos se o registro foi incluído:
SELECT * FROM tbl_Editoras;
E então o excluímos com DELETE:
DELETE FROM tbl_Editoras WHERE IdEditora = 11;

TRUNCATE TABLE
● Remove todas as linhas de uma tabela sem registrar as exclusões de linhas individuais.
● Funciona quase como a instrução DELETE sem usar a cláusula WHERE.
● Entretanto, TRUNCATE TABLE é mais rápida e utiliza menos recursos de sistema e log de transações.

TRUNCATE TABLE - Exemplo
Exemplo: Vamos recriar a tabela de nome tbl_Emprestimos, com apenas três campos:
CREATE TABLE tbl_Emprestimos (IdEmprestimo TINYINT PRIMARY KEY, NomeLivro VARCHAR (60) NOT NULL, NomeEmprestador VARCHAR (50) NOT NULL
);

TRUNCATE TABLE - Exemplo
Inserimos alguns registros na tabela e verificamos seu conteúdo:
INSERT INTO tbl_Emprestimos (IdEmprestimo, NomeLivro, NomeEmprestador)

VALUES (20,'O Cortiço','João da Silva'), (21,'A Moreninha','Marcos de Souza'), (22,'Macunaíma','Ana Abreu');
SELECT * FROM tbl_Emprestimos;

TRUNCATE TABLE - Exemplo
E agora excluímos todos os registros da tabela de uma vez, e verificamos seu conteúdo:
TRUNCATE TABLE tbl_Emprestimos; SELECT * FROM tbl_Emprestimos;
E excluímos novamente a tabela:
DROP TABLE tbl_Emprestimos;

Consultas Simples
Executamos consultas em SQL usando a cláusula SELECT, conforme a seguinte sintaxe:
SELECT Colunas FROM Tabela;

Consultas Simples - Exemplos
SELECT NomeAutor FROM tbl_Autores; SELECT * FROM tbl_Autores; SELECT NomeLivro FROM tbl_Livros; SELECT Assunto FROM tbl_Assuntos;

Especificando Colunas
SELECT NomeLivro, PrecoLivro FROM tbl_Livros;
SELECT NomeAutor, SobrenomeAutor FROM tbl_Autores;
SELECT NomeLivro, ISBN13, DataPub FROM tbl_Livros;

Consulta com Ordenação
A cláusula ORDER BY é usada para ordenar o conjuntoresultado de registros em uma consulta. Sintaxe:
SELECT colunas FROM tabela ORDER BY colunas [ASC | DESC];
Parâmetros: ● ASC – Ordem ascendente ● DESC – Ordem descendente (inversa)

ORDER BY - Exemplos
SELECT * FROM tbl_Livros ORDER BY NomeLivro ASC;
SELECT NomeLivro, IdEditora FROM tbl_Livros ORDER BY IdEditora; SELECT NomeLivro, PrecoLivro FROM tbl_Livros ORDER BY PrecoLivro DESC; SELECT NomeLivro, DataPub, idAssunto FROM tbl_Livros ORDER BY idAssunto, NomeLivro;

LIMIT

A cláusula LIMIT é usada para especificar o número de registros a retornar em uma consulta. ùtil para quando queremos apenas um subconjunto de uma consulta grande.

Sintaxe:

```
SELECT colunas FROM tabela ORDER BY coluna
LIMIT [offset], contagem;
```

LIMIT - Exemplos

Retornar os dois livros mais baratos da tabela de livros:

```
SELECT NomeLivro, PrecoLivro FROM tbl_Livros ORDER BY PrecoLivro LIMIT 2;
```

LIMIT - Exemplos

Retornar os três livros mais caros da tabela de livros:

```
SELECT NomeLivro, PrecoLivro FROM tbl_Livros ORDER BY PrecoLivro DESC LIMIT 3;
```

LIMIT - Exemplos

Retornar o 3º, 4º e 5º livros com maior número de páginas da tabela de livros:

```
SELECT NomeLivro, NumeroPaginas FROM tbl_Livros ORDER BY NumeroPaginas DESC LIMIT 2, 3;
```

Aqui, 2 significa que a contagem se inicia no 3º item (contado a partir de 0), e serão retornados os próximos 3 itens.

Cláusula WHERE

A cláusula WHERE permite filtrar registros nos resultados de uma consulta, de acordo com um critério de valor especificado.

Sintaxe:

```
SELECT colunas FROM tabela WHERE coluna = valor;
```

WHERE - Exemplos

```
SELECT NomeLivro, DataPub FROM tbl_Livros WHERE IdEditora = 2;
SELECT IdAutor, NomeAutor FROM tbl_Autores WHERE SobrenomeAutor = 'Stanek';
SELECT IdAssunto FROM tbl_Assuntos WHERE Assunto = 'Terror';
```

Índices

● Índices são empregados em consultas para ajudar a encontrar registros com um valor específico em uma coluna de forma rápida - ou seja, aumentar o desempenho na execução de consultas.

● Com índices, o MySQL vai direto a uma linha em vez de buscar toda a tabela até encontrar os registros que importam.

Por padrão, o MySQL cria índices automaticamente para campos de:
● Chave Primária
● Chave Estrangeira
● Constraint UNIQUE
Além disso, podemos criar índices para outras colunas usadas com frequência em buscas ou junções.

Índice Clusterizado

● Índices clusterizados alteram a forma como os dados são armazenados em um banco de dados, pois ele classifica as linhas de acordo com a coluna que possui o índice.
● Uma tabela só pode ter um índice clusterizado. Geralmente está na coluna que é chave primária da tabela ou, em sua ausência, em uma coluna UNIQUE.
● Se uma tabela não possuir índice clusterizado, suas linhas são armazenadas em uma estrutura não-ordenada chamada de heap.

Índice Não-Clusterizado

● Em um índice não-clusterizado a forma como os dados são armazenados não é alterada, e um objeto separado é criado na tabela, apontando para as linhas da tabela original após a busca.
● Baseia-se em valores-chave
● Uma tabela pode ter vários índices nãoclusterizados.

Criar Índices

Declaração CREATE INDEX em uma tabela já existente: Sintaxe:
CREATE [UNIQUE] INDEX nome_índice ON nome_tabela (coluna1 [ASC | DESC], [coluna2 [ASC | DESC]]...
);

Criar Índices
Declaração ALTER TABLE / ADD INDEX em uma tabela já existente:
Sintaxe:
ALTER TABLE nome_tabela ADD INDEX nome_índice (colunas)

Podemos criar um índice não-clusterizado durante a criação da tabela também, como neste exemplo:
CREATE TABLE tbl_Editoras (IdEditora SMALLINT PRIMARY KEY AUTO_INCREMENT, NomeEditora VARCHAR(40) NOT NULL, INDEX (NomeEditora));

Visualizar Índices

Podemos visualizar os índices associados com uma tabela por meio do comando SHOW INDEX, como segue:
SHOW INDEX FROM tabela;

Testar Índices

Vamos testar a eficiência dos índices. Com o comando EXPLAIN podemos ver como o MySQL realiza uma consulta internamente:
EXPLAIN SELECT * FROM tbl_Editoras WHERE NomeEditora = 'Springer';
Testar Índices
Acrescentamos um índice à coluna NomeEditora da tabela tbl_Editoras:
CREATE INDEX idx_editora ON tbl_Editoras(NomeEditora);
O visualizamos:
SHOW INDEX FROM tbl_Editoras;

Testar Índices
Repetimos o comando EXPLAIN para ver como o MySQL irá se comportar após a criação do índice:
EXPLAIN SELECT * FROM tbl_Editoras WHERE NomeEditora = 'Springer';

Excluir Índices

Excluímos um índice com o comando DROP INDEX: Sintaxe:
DROP INDEX nome_índice ON tabela;
Exemplo:
DROP INDEX idx_editora ON tbl_Editoras;

Operadores Lógicos

● AND, OR e NOT
● Também chamados de Operadores Booleanos
● Permitem trabalhar com múltiplas condições relacionais na mesma expressão
● Retornam valores lógicos (True ou False).

AND, OR e NOT
Usados para filtrar registros baseados em mais de uma condição.
● O operador AND mostra um registro se ambas as condições forem verdadeiras.
● O operador OR mostra um registro se pelo menos uma das condições for verdadeira.
● O operador NOT é a negação de uma expressão.

Operador Lógico AND

Condição A	Condição B	Resultado
False	False	False
True	False	False
False	True	False
True	True	True

O operador lógico AND somente retorna True se todas as condições de entrada forem verdadeiras.
A <- False B <- True C <- A AND B
C é igual a False

Operador Lógico OR

Condição A	Condição B	Resultado
False	False	False
True	False	True
False	True	True
True	True	True

O operador lógico OR somente retorna False se todas as condições de entrada forem falsas.
A <- False B <- True C <- A OR B
C é igual a True

Operador Lógico NOT

Entrada	Saída
False	True
True	False

O operador lógico NOT inverte a condição de entrada: True se torna False, e False se torna True.
A <- True B <- NOT A
B é igual a False.

Exemplos – AND, OR e NOT
SELECT IdLivro, NomeLivro, IdEditora FROM tbl_Livros WHERE IdLivro > 102 AND IdEditora < 4; SELECT IdLivro, NomeLivro, IdEditora FROM tbl_Livros WHERE IdLivro > 102 OR IdEditora < 4; SELECT IdLivro, NomeLivro, IdEditora FROM tbl_Livros WHERE NOT IdEditora = 10; SELECT IdLivro, NomeLivro, IdEditora FROM tbl_Livros WHERE IdLivro > 102 AND NOT IdEditora <= 4;

Criar Alias com AS
Pode-se dar um nome diferente a uma coluna ou tabela no retorno de uma consulta usando um Alias.
Sintaxe:
SELECT coluna1 AS alias_coluna1, coluna2 AS alias_coluna2, colunaN alias_colunaN
FROM tabela AS alias_tabela;

SQL Alias - Exemplos
SELECT NomeLivro AS Livro FROM tbl_Livros
WHERE IdLivro > 102; SELECT NomeAutor AS Nome, SobrenomeAutor Sobrenome FROM tbl_Autores AS Autores;
SELECT NomeLivro AS Livro, PrecoLivro AS 'Preço do Livro'
FROM tbl_Livros AS Livros ORDER BY 'Preço do Livro' DESC;

Funções de Agregação
Funções de agregação são funções SQL que permitem executar uma operação aritmética nos valores de uma coluna em todos os registros de uma tabela.
Retornam um valor único baseado em um conjunto de valores.

Funções de Agregação
Sintaxe básica:
SELECT Função (ALL | DISTINCT expressão)
● ALL - avalia todos os registros ao agregar o valor da função; é o comportamento padrão.
● DISTINCT - Usa apenas valores distintos (sem repetição) ao
avaliar a função.
● As funções de agregação desconsideram valores NULL
(exceto a função COUNT(*))
● Cláusulas WHERE não podem conter funções agregadas.

Funções de Agregação
● MIN = Valor Mínimo
● MAX = Valor Máximo
● AVG = Média Aritmética
● SUM = Totalizar (Soma)
● COUNT = Contar quantidade de itens

Exemplos
SELECT COUNT(*) AS Total FROM tbl_Autores;
SELECT COUNT(DISTINCT idEditora) FROM tbl_Livros; SELECT MAX(PrecoLivro) AS 'Mais caro' FROM tbl_Livros; SELECT MIN(PrecoLivro) FROM tbl_Livros;
SELECT AVG(PrecoLivro) AS Média FROM tbl_Livros; SELECT SUM(PrecoLivro) AS 'Preço Total' FROM tbl_Livros; SELECT SUM(PrecoLivro) / COUNT(*) AS 'Preço Médio' FROM tbl_Livros;
SELECT COUNT(*) AS 'Quant Livros', SUM(NumeroPaginas) AS 'Páginas Totais', AVG(NumeroPaginas) 'Média de Págs.'
FROM tbl_Livros; Fábio dos Rei

BETWEEN - Seleção de Intervalos
A palavra BETWEEN permite estabelecer um intervalo de filtragem em uma cláusula WHERE. Sintaxe:
SELECT colunas FROM tabela WHERE coluna BETWEEN valor1 AND valor2;

BETWEEN - Exemplos
SELECT * FROM tbl_Livros WHERE DataPub BETWEEN '20040517' AND '20110517'; SELECT NomeLivro AS Livro, PrecoLivro AS Preço FROM tbl_Livros WHERE PrecoLivro BETWEEN 150.00 AND 200.00; SELECT NomeLivro, PrecoLivro FROM tbl_Livros WHERE PrecoLivro BETWEEN 170.00 AND 180.00 OR PrecoLivro BETWEEN 220.00 AND 300.00;

LIKE e NOT LIKE
● A cláusula LIKE determina se uma cadeia de caracteres corresponde a um padrão especificado. Um padrão pode incluir caracteres normais e coringas.
● NOT LIKE inverte a comparação, verificando se a cadeia de caracteres NÃO corresponde ao padrão especificado.

LIKE – Padrões específicos
Metacaracteres '%' Qualquer cadeia de 0 ou mais caracteres '_' Sublinhado: qualquer caractere único

Usando o LIKE
SELECT * FROM tbl_Livros WHERE NomeLivro LIKE 'A%'; SELECT * FROM tbl_Livros WHERE NomeLivro NOT LIKE 'S%'; SELECT NomeLivro FROM tbl_Livros WHERE NomeLivro LIKE '_i%';
SELECT NomeLivro AS Livro, PrecoLivro AS Valor FROM tbl_Livros
WHERE NomeLivro NOT LIKE 'A%' AND PrecoLivro <= 190.00;

Declaração GROUP BY

Usamos a declaração GROUP BY para agrupar registros em subgrupos baseados em colunas ou valores retornados por uma expressão.
A cláusula WHERE, quando empregada, é sempre avaliada antes do agrupamento com GROUP BY.

GROUP BY
Sintaxe básica:
SELECT colunas, função_agregação()
FROM tabela WHERE filtro GROUP BY colunas ORDER BY colunas
GROUP BY - Exemplos
SELECT IdAssunto, SUM(NumeroPaginas) FROM tbl_Livros GROUP BY IdAssunto;
SELECT IdEditora, SUM(PrecoLivro) FROM tbl_livros GROUP BY IdEditora;
SELECT IdEditora, AVG(NumeroPaginas), IdAssunto FROM tbl_livros GROUP BY IdEditora;
SELECT IdEditora, SUM(PrecoLivro) FROM tbl_livros
WHERE NumeroPaginas >= 1000
GROUP BY IdEditora ORDER BY NumeroPaginas;

WITH ROLLUP

Podemos acrescentar uma linha de sumário a um agrupamento feito com GROUP BY usando a declaração WITH ROLLUP
SELECT colunas, função_agregação() FROM tabela
WHERE filtro
GROUP BY coluna WITH ROLLUP
Não é possível usar ORDER BY ao usar WITH ROLLUP.

GROUP BY WITH ROLLUP - Exemplo
SELECT IdEditora, SUM(NumeroPaginas) AS SomaPaginas FROM tbl_livros WHERE IdAssunto > 3 GROUP BY IdEditora WITH ROLLUP HAVING SomaPaginas >= 900;
Temos aqui a soma de páginas totalizada também, além de agrupada por editora.

HAVING

Usada para aplicar filtros em grupos de registros ou agregações.
Frequentemente usada com a cláusula GROUP BY para filtrar as colunas agrupadas.
Sempre avaliada após o agrupamento com GROUP BY. Somente pode se referir a uma coluna listada na cláusula SELECT, ao contrário de filtros com WHERE.

HAVING
Sintaxe:
SELECT colunas, função_agregação() FROM tabela
WHERE filtro
GROUP BY colunas HAVING filtro_agrupamento ORDER BY coluna;

HAVING - Exemplo 01:
Consulta retornando IDs de editoras cuja soma de preços de livros seja maior que 200 reais
SELECT IdEditora, SUM(PrecoLivro) AS Soma FROM tbl_livros GROUP BY IdEditora HAVING Soma > 200;
HAVING - Exemplo 02:
Consulta retornando IDs de editoras, médias de números de páginas por editora e IDs de categorias, cuja média de nº de páginas seja igual ou maior a 1000 por editora.
SELECT IdEditora, AVG(NumeroPaginas), IdAssunto FROM tbl_livros GROUP BY IdEditora HAVING AVG(NumeroPaginas) >= 1000;

HAVING - Exemplo 02:
Retornar IDs de editoras e soma de páginas por editora, com ID de assunto > 3 e soma de páginas >= a 900 por editora.
SELECT IdEditora, SUM(NumeroPaginas) AS SomaPaginas FROM tbl_livros
WHERE IdAssunto > 3
GROUP BY IdEditora HAVING SomaPaginas >= 900 ORDER BY IdEditora;

JOINS

Cláusulas JOIN são usadas para combinar dados provenientes de duas ou mais tabelas em um único conjunto de resultados, baseado em condições de join especificadas. Há duas sintaxes possíveis para codificar joins: sintaxe explícita (SQL-92; recomendada) e sintaxe implícita (condição de join na cláusula WHERE).

JOINS - Categorias
● INNER JOIN: Retorna linhas quando houver pelo menos uma correspondência em ambas as tabelas. ● OUTER JOIN: Retorna linhas mesmo quando não houver pelo menos uma correspondência em uma das tabelas (ou ambas). O OUTER JOIN divide-se em LEFT JOIN, RIGHT JOIN e CROSS JOIN.

Cláusula ON

● A cláusula ON determina a condição de join, que indica como as tabelas devem ser comparadas.
● No geral, a comparação ocorre por meio de um relacionamento entre chave primária na primeira tabela e chave estrangeira na segunda tabela.

Condição de Join

Uma condição de join nomeia uma coluna em cada tabela envolvida no join e indica como as colunas devem ser comparadas.
No geral, usamos o operador = para obter linhas com colunas correspondentes. É comum usar o relacionamento de PK de uma tabela com FK de outra tabela.

Nomes de Coluna Qualificados

Um nome de coluna qualificado possui o nome da coluna precedido pelo nome da tabela à qual pertence, separados por um ponto.
Exemplo:
tbl_Livros.IdEditora
Indica a coluna IdEditora especificamente da tabela tbl_Livros.

Usamos os nomes de colunas qualificados em JOINS para identificar exatamente a qual tabela cada campo envolvido pertence.
Isso evita erro de ambiguidade caso uma coluna tenha o mesmo nome em duas tabelas diferentes.
ON tbl_Livros.IdEditora = tbl_Editoras.IdEditora

INNER JOIN Sintaxe

SELECT colunas FROM tabela1 [INNER] JOIN tabela2 ON tabela1.coluna=tabela2.coluna; [INNER] JOIN tabelaN ON tabela1.coluna=tabelaN.coluna;

INNER JOIN - Exemplos

SELECT * FROM tbl_Livros INNER JOIN tbl_Editoras ON tbl_Livros.IdEditora = tbl_Editoras.IdEditora;
SELECT tbl_Livros.NomeLivro, tbl_Livros.ISBN13, tbl_Assuntos.Assunto FROM tbl_Livros JOIN tbl_Assuntos ON tbl_Livros.IdAssunto = tbl_Assuntos.IdAssunto;

INNER JOIN – Mais exemplos
Usando Aliases e cláusulas WHERE e LIKE:
SELECT L.NomeLivro AS Livros, E.NomeEditora AS Editoras FROM tbl_Livros AS L JOIN tbl_Editoras AS E ON L.IdEditora = E.IdEditora WHERE E.NomeEditora LIKE 'M%';

INNER JOIN - com três tabelas
SELECT L.NomeLivro Livro, C.Assunto Assunto, E.NomeEditora Editora FROM tbl_Livros L JOIN tbl_Assuntos C ON L.IdAssunto = C.IdAssunto JOIN tbl_Editoras E ON L.IdEditora = E.IdEditora;

INNER JOIN - outro com três tabelas
SELECT L.NomeLivro Livro, CONCAT(A.NomeAutor, ' ', A.SobrenomeAutor) Autor, L.PrecoLivro 'Preço do Livro' FROM tbl_livrosautores LA JOIN tbl_livros L ON L.IdLivro = LA.IdLivro JOIN tbl_Autores A ON A.IdAutor = LA.IdAutor;

Joins com USING
Normalmente, as colunas comparadas em uma cláusula ON possuem o mesmo nome. Quando essa comparação ocorre por meio de um equijoin (sinal de =), é possível empregar a palavra USING para simplificar a consulta.

Joins com USING
Sintaxe:
SELECT colunas FROM tabela1
JOIN tabela2 USING (coluna_em_comum) JOIN tabelaN USING (coluna_em_comum);

Joins com USING
SELECT L.NomeLivro AS Livro,
CONCAT(A.NomeAutor, ' ', A.SobrenomeAutor) AS Autor,
L.PrecoLivro As 'Preço do Livro' FROM tbl_livrosautores LA
JOIN tbl_Livros L USING(IdLivro) JOIN tbl_Autores A USING(IdAutor)
WHERE L.PrecoLivro BETWEEN 160.00 AND 200.00
ORDER BY Livro;

NATURAL Joins
Em uma natural join, não especificamos a coluna usada para conectar as tabelas. O BD automaticamente as une baseado nas colunas das duas tabelas que possuem o mesmo nome. Este tipo de join só funciona corretamente se o banco for projetado

de forma específica. Evite o uso de joins naturais, a não ser em tabelas simples relacionadas com um único campo em comum.
NATURAL Joins
Sintaxe:
SELECT colunas FROM tabela1
NATURAL JOIN tabela2 [NATURAL JOIN tabelaN]...;

Joins com NATURAL
SELECT L.NomeLivro AS Livro, CONCAT(A.NomeAutor, ' ', A.SobrenomeAutor) AS Autor, L.PrecoLivro As 'Preço do Livro' FROM tbl_livrosautores LA NATURAL JOIN tbl_Livros L NATURAL JOIN tbl_Autores A WHERE L.PrecoLivro BETWEEN 160.00 AND 200.00 ORDER BY Livro;

INNER JOIN Implícito
Forma antiga de se realizar joins. Importante conhecer para dar manutenção em bancos legados.
Nesta forma de join, as condições de join são codificadas na cláusula WHERE.
Recomenda-se NÃO usar joins implícitas em queries.

INNER JOIN Implícito
Sintaxe:
SELECT colunas
FROM tabela1, tabela2 [, tabelaN]...
WHERE tabela1.coluna operador tabela2.coluna [AND tabela2.coluna operador tabela3.coluna}...

INNER JOIN Implícito
SELECT L.NomeLivro AS Livro,
CONCAT(A.NomeAutor, ' ', A.SobrenomeAutor) AS Autor, L.PrecoLivro As 'Preço do Livro' FROM tbl_livrosautores LA, tbl_Livros L, tbl_Autores A WHERE L.IdLivro = LA.IdLivro AND LA.IdAutor = A.IdAutor AND L.PrecoLivro BETWEEN 160.00 AND 200.00 ORDER BY Livro;

OUTER JOINS
● LEFT JOIN: Retorna todas as linhas da tabela à
esquerda, mesmo se não houver nenhuma correspondência na tabela à direita.
● RIGHT JOIN: Retorna todas as linhas da tabela à direita, mesmo se não houver nenhuma correspondência na tabela à esquerda.
● CROSS JOIN: Retorna o produto cartesiano das linhas das tabelas.

LEFT JOIN - Sintaxe
SELECT colunas FROM tabela_esq LEFT (OUTER) JOIN tabela_dir ON tabela_esq.coluna=tabela_dir.coluna;

LEFT JOIN - Exemplo
SELECT * FROM tbl_Assuntos LEFT JOIN tbl_Livros ON tbl_Livros.IdAssunto = tbl_Assuntos.IdAssunto;

LEFT JOIN - excluir correspondências
Sintaxe:
SELECT coluna FROM tabela_esq LEFT (OUTER) JOIN tabela_dir ON tabela_esq.coluna=tabela_dir.coluna WHERE tabela_dir.coluna IS NULL;

LEFT JOIN - excluir correspondências
Exemplo - Só os assuntos sem livros cadastrados ainda na tabela de livros:
SELECT * FROM tbl_Assuntos LEFT JOIN tbl_Livros ON tbl_Livros.IdAssunto = tbl_Assuntos.IdAssunto WHERE tbl_Livros.IdAssunto IS NULL;

RIGHT JOIN
Sintaxe:
SELECT colunas FROM tabela_esq RIGHT (OUTER) JOIN tabela_dir ON tabela_esq.coluna=tabela_dir.coluna;

RIGHT JOIN - Exemplo
SELECT * FROM tbl_Livros AS Li RIGHT JOIN tbl_Editoras AS Ed ON Li.IdEditora = Ed.IdEditora;

RIGHT JOIN - excluir correspondências
Sintaxe:
SELECT coluna FROM tabela_esq RIGHT (OUTER) JOIN tabela_dir ON tabela_esq.coluna=tabela_dir.coluna WHERE tabela_esq.coluna IS NULL;

RIGHT JOIN - excluir correspondências
Exemplo:
SELECT * FROM tbl_Livros RIGHT JOIN tbl_Editoras ON tbl_Livros.IdEditora = tbl_Editoras.IdEditora WHERE tbl_Livros.IdEditora IS NULL;

CROSS JOIN

Retorna um produto cartesiano entre as tabelas, mostrando todas as combinações possíveis entre os registros.

Sintaxe:

SELECT colunas FROM tabela1 CROSS JOIN tabela2;

CROSS JOIN - Exemplo

SELECT * FROM tbl_Livros CROSS JOIN tbl_livrosautores;

FULL OUTER JOIN

Ou simplesmente FULL JOIN, é uma junção que retorna todos os registros tanto da tabela da esquerda quanto da tabela da direita. Ou seja, retorna todos os registros de ambas as tabelas!

Não há suporte a FULL OUTER JOIN em MySQL. Veremos como simular essa junção usando UNION na próxima aula.

Operador UNION

Combina dados provenientes de duas ou mais consultas.

Uma UNION combina as linhas de dois ou mais conjuntos de resultados.

Cada declaração SELECT deve ter o mesmo número de colunas, tipos de dados e ordem das colunas.

Operador UNION

Sintaxe:

SELECT declaração1 UNION [ALL] SELECT declaração2 UNION [ALL] SELECT declaração3 ... [ORDER BY colunas];

Exemplo 01

Retornar nomes de livros e preços dos livros. Caso o preço do livro seja igual ou superior a R$ 150,00, mostrar a mensagem "Livro Caro" em uma coluna à direita no resultado da consulta.

Caso contrário, mostrar a mensagem "Preço Razoável" Ordenar por preço, do mais barato para o mais caro.

Exemplo 01 - Resolução

SELECT NomeLivro Livro, PrecoLivro Preço, 'Livro Caro' Resultado FROM tbl_Livros WHERE PrecoLivro >= 150.00 UNION SELECT NomeLivro Livro, PrecoLivro Preço, 'Preço Razoável' Resultado FROM tbl_Livros WHERE PrecoLivro < 150.00 ORDER BY Preço;

Exemplo 02
Retornar nomes de livros, preços e assuntos dos livros. Caso o assunto seja Eletrônica, mostrar o preço acrescido de 15% em seu valor.
Caso o livro custe mais de 200 reais, descontar 10% em seu valor.
Ordenar por preço ajustado, do mais caro para o mais barato.

Exemplo 02 - Resolução

```
SELECT L.NomeLivro Livro, L.PrecoLivro 'Preço Normal', L.PrecoLivro * 0.90 'Preço Ajustado', A.Assunto
FROM tbl_Livros L INNER JOIN tbl_Assuntos A
ON L.IdAssunto = A.IdAssunto WHERE L.PrecoLivro > 200.00 UNION
SELECT L.NomeLivro Livro, L.PrecoLivro 'Preço Normal', L.PrecoLivro * 1.15 'Preço Ajustado', A.Assunto
FROM tbl_Livros L INNER JOIN tbl_Assuntos A
ON L.IdAssunto = A.IdAssunto WHERE A.Assunto = 'Eletrônica' ORDER BY 'Preço Ajustado' DESC;
```

Exemplo 03
Vamos simular um FULL OUTER JOIN (sem suporte em MySQL) usando UNION:

```
SELECT * FROM tbl_Assuntos LEFT JOIN tbl_Livros
ON tbl_Livros.IdAssunto = tbl_Assuntos.IdAssunto UNION
SELECT * FROM tbl_Assuntos RIGHT JOIN tbl_Livros
ON tbl_Livros.IdAssunto = tbl_Assuntos.IdAssunto;
```

Operações aritméticas

É possível realizar operações matemáticas simples nos valores de uma coluna e retornar resultados em uma coluna calculada. Para isso usamos os operadores matemáticos comuns:

+ Soma
- Subtração
/ Divisão
* Multiplicação
% ou MOD Módulo (resto da divisão inteira) DIV Divisão inteira

Operações aritméticas– Exemplos:

```
SELECT 3 * 9; SELECT NomeLivro, PrecoLivro * 5 AS 'Preço de 5 Unidades' FROM tbl_Livros; SELECT 2 * 9 / 3; SELECT NomeLivro, PrecoLivro / 2 AS 'Preço com 50% de desconto' FROM tbl_Livros; SELECT 10 MOD 3;
```

Funções Matemáticas

É possível também utilizar funções matemáticas nos valores de uma coluna e retornar resultados em uma coluna calculada. No slide seguinte temos algumas das funções matemáticas mais comumente empregadas em consultas no MySQL

Funções Matemáticas

CEILING(x) Arredonda para cima
FLOOR(x) Arredonda para baixo
ROUND(n,x) Arredonda x casas decimais
HEX(d) Retorna a representação hexadecimal de um valor decimal d.
SIN(x) Retorna o seno de um número dado em radianos
PI() Retorna o valor de Pi
POW(x,y) Retorna x elevado a y
SQRT(x) Raiz quadrada de x

Exemplos de Funções Matemáticas

```
SELECT PI(); SELECT POW(2,4); SELECT SQRT(81); SELECT SIN(PI());
SELECT HEX(1200);
SELECT NomeLivro, ROUND(PrecoLivro * 5, 2) AS 'Preço Arredondado' FROM tbl_Livros;
```