INICIAÇÃO EM BANCO DE DADOS

Índice

Capítulo 01 - Instalação e configuração Banco de Dados

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Onde baixar o banco de dados MySQL
- Instalação do MySQL
- Instalação do SGBD

Onde baixar o banco de dados MySQL

Neste treinamento utilizaremos o banco de dados MariaDB, que é a versão open source do banco de dados MySQL.

O MariaDB versão 10.1.22 pode ser baixado no seguinte endereço:

https://mirror1.cl.netactuate.com/mariadb//mariadb-10.5.8/winx64-packages/mariadb-10.5.8-winx64.zip

Instalação do MySQL

O link de download nos retorna um arquivo do tipo zip que já contém o banco de dados pronto para uso, para isso precisamos apenas descompactar o arquivo zip.
Após descompactar o arquivo devemos acessar a pasta bin e executar o arquivo mysqld, esse aplicativo é o aplicativo responsável por subir o servidor do banco de dados e torná-lo disponível para uso. Por padrão o mysql roda na porta 3306.

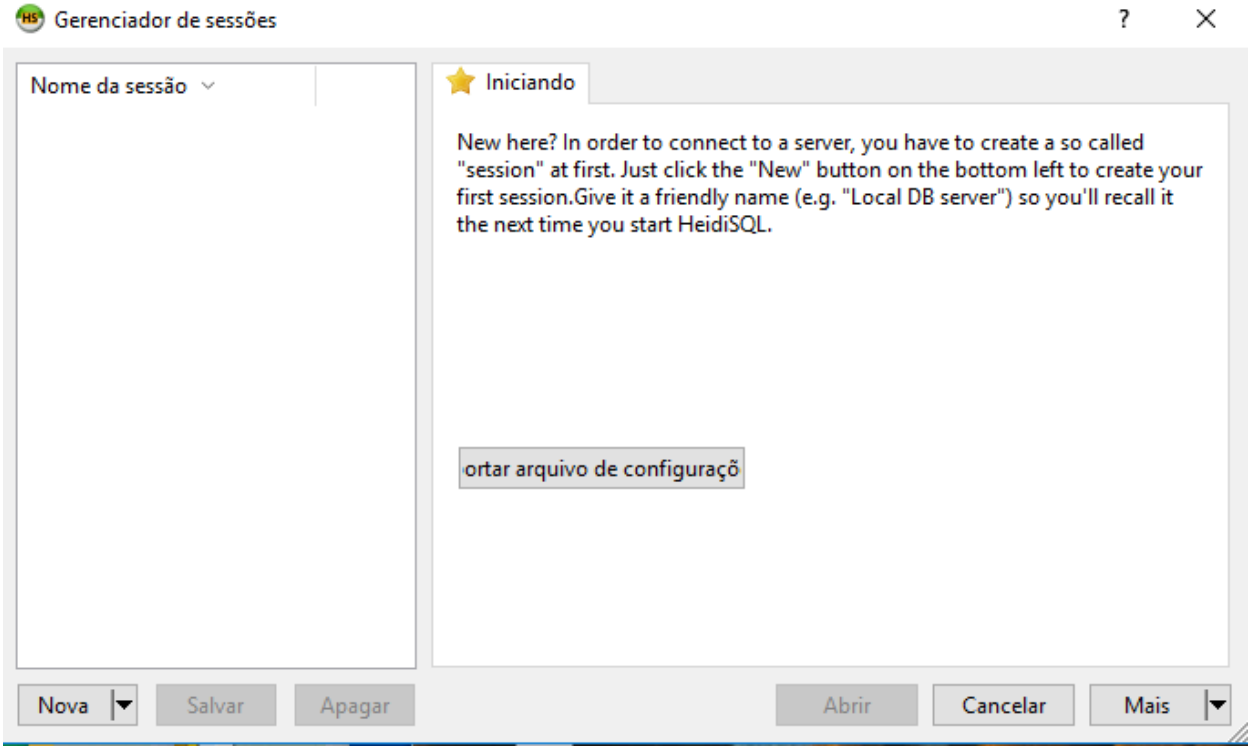
Instalação do SGBD

O SGBD utilizado neste treinamento será o HeidiSQL, ele é uma interface gráfica também open source que podemos utilizar para conectarmos ao banco e executar os comandos.
O HeidiSQL pode ser baixado no seguinte endereço:

https://www.heidisql.com/downloads/releases/HeidiSQL_9.4_Portable.zip

Após o download do arquivo basta descompactar o zip e executar o arquivo \heidisql.exe

A tela inicial deverá ser como a da imagem abaixo.

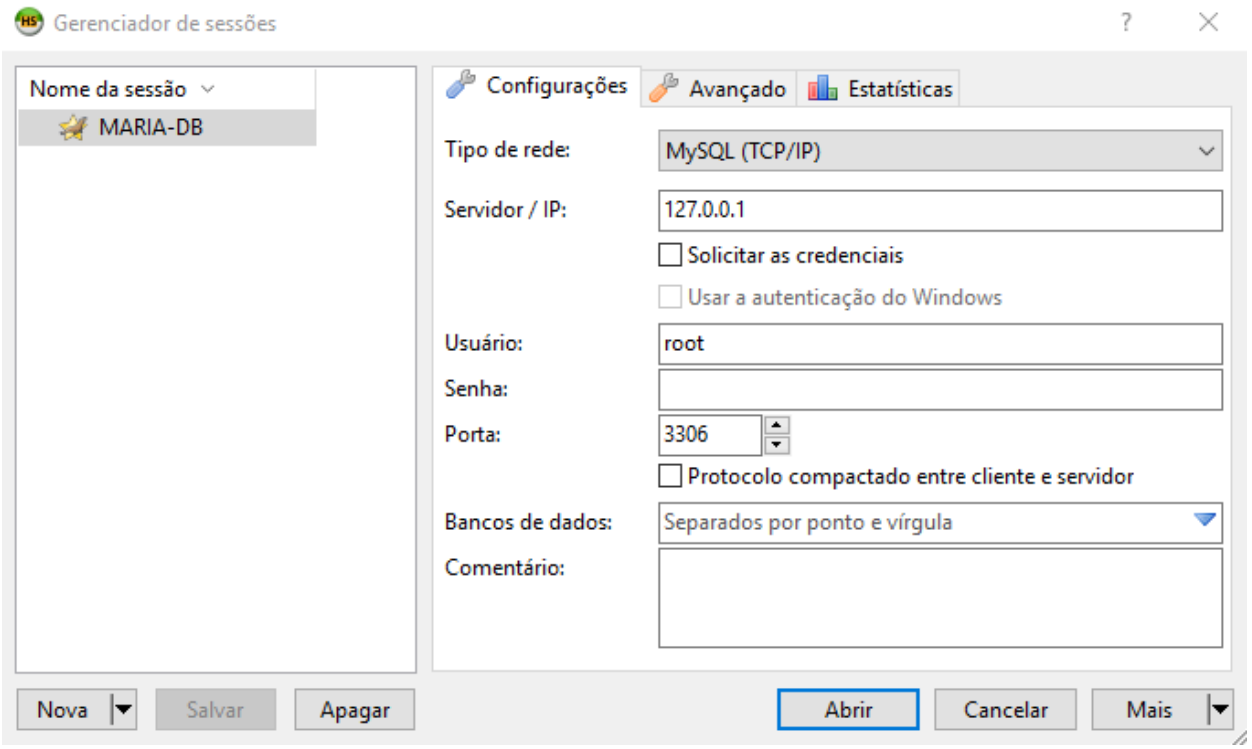


Para adicionarmos uma conexão com o banco de dados precisamos acessar o botão Nova e em seguida informar o tipo de rece, o ip do servidor e os dados do usuário, senha e rede. Neste treinamento utilizaremos as seguintes configurações:

Tipo de Rede: MySQL(TCP/IP)
Servidor/IP: 127.0.0.1
Usuário: root
Senha:””
Porta: 3306

No painel mais à esquerda devemos informar o um nome para a nossa conexão.

A imagem abaixo mostra um exemplo depois das configurações aplicadas.



Capítulo 02 - Conceitos básicos sobre banco de dados

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Conceito de dados
- Conceito de informação
- O que é um Banco de dados
- O que é um SGBD
- Tipos de banco de dados

Conceito de dados

Em um sistema computacional pode-se dizer que dados é um conjunto alfanumérico ou até mesmo de imagens e que não está agregado a algum conhecimento específico, necessitando de um contexto para que possa ser interpretado.

Conceito de informação

É a classificação de um grupo de dados através de um conhecimento específico, permitindo que os dados possam ser interpretados de acordo com um determinado contexto.

Dado + informação = Conhecimento

O que é um Banco de dados

Banco de dados é um conjunto de dados relacionados utilizado para a manipulação de informações.
Um banco de dados pode ser armazenado em arquivos, papel, ou qualquer outro meio físico de armazenamento.

O que é um SGBD

Ferramenta utilizada para a manipulação do banco de dados, usualmente conhecida como a interface de comunicação entre o usuário e o banco de dados.

Tipos de banco de dados

O tipo de banco de dados está relacionado a forma como os dados são armazenados, atualmente os bancos de dados mais utilizados, são os bancos de dados relacionais, que armazenam as informações na forma de tabelas e registros, onde cada tabela é formada por um conjunto de colunas, esse tipo de banco de dados recebe o nome relacional devido ao fato de podermos relacionar (unir) as informações entre tabelas através da definição de chaves de ligação (colunas de relacionamento).
Além dos bancos de dados relacionais existem ainda outros tipos atualmente denominados NOSQL, estes tipos utilizam outras formas de armazenamento de dados, podendo ser no formato de documentos, ligação chave valor, colunas ou objetos.

Este treinamento abordará a utilização de bancos de dados relacionais que utilizam a linguagem SQL (Structured Query Language).

Exercícios

1. Descreva com suas palavras o significado de dados.
2. Descreva com suas palavras o significado de informação.
3. Descreva com suas palavras o significado de conhecimento.
4. O que é um SGBD?
5. Qual o tipo de banco de dados mais utilizado atualmente?

Capítulo 03 - Comandos para manipulação de base dados

Neste capítulo veremos os seguintes tópicos:

- Como criar uma base de dados
- Como selecionar uma base de dados
- Como excluir uma base de dados

Como criar uma base de dados

Para que possamos trabalhar com um banco de dados devemos inicialmente criar uma base de dados onde nossas informações possam ser armazenadas, uma base de dados um armazenamento lógico onde dispomos informações relacionadas.

Para criarmos uma base de dados devemos executar o seguinte comando:

```
CREATE DATABASE NOME_DA_BASE;
```

onde:
CREATE DATABASE: comando para criação de uma nova base de dados
NOME_DA_BASE: nome que será atribuído a base de dados

Exemplo

```
CREATE DATABASE FINANCEIRO;
```

Como selecionar uma base de dados

No Mysql podemos selecionar uma base de dados através do seguinte comando:

```
USE NOME_DA_BASE
```

onde:

USE: comando utilizado para a seleção de uma base de dados
NOME_DA_BASE: nome da base de dados que queremos utilizar.

Como excluir uma base de dados

Para a remoção de uma base de dados devemos utilizar o seguinte comando:

```
DROP DATABASE NOME_DA_BASE
```

onde:

DROP DATABASE: comando para exclusão da base de dados
NOME_DA_BASE: nome da base de dados que queremos remover

Exercícios:

1. Crie uma base de dados com o nome Financeiro.
2. Crie uma base de dados com o nome ERP.
3. Seleciona para a utilização da base de dados Financeiro.
4. Selecione para a utilização a base de dados ERP.
5. Remova a base de dados Financeiro.

Capítulo 04 - Comandos para manipulação de tabelas

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Tipos de dados
- Como criar uma tabela
- Como alterar uma tabela
- Como excluir uma tabela
- Como incluir registros em uma tabela
- Como selecionar registros
- Como atualizar registros
- Como remover registros

Tipos de dados

O Mysql oferece o armazenamento para os seguintes tipos de dados:

Tipos para armazenamento de texto

CHAR(tamanho) – Armazena um número fixo de caracteres alfanuméricos com um limite de 255 posições
VARCHAR(tamanho) – De forma semelhante ao char armazena um número fixo de caracteres porém com suporte maior do que o char
TEXT: Armazena um texto de até 65.535 caracteres
BLOB: Armazena informação no formato binário (bytes), é normalmente utilizado para armazenar arquivos e imagens, possui uma capacidade de 65.535bytes

Tipos para armazenamento numérico

TINYINT: guarda números do tipo inteiro. Suporta de -128 até 127 caracteres.
SMALLINT: guarda números do tipo inteiro. Suporta de -32768 até 32767 caracteres.
MEDIUMINT: guarda números do tipo inteiro. Suporta de -8388608 até 8388607 caracteres.
INT: guarda números inteiros. Suporta de -2147483648 até 2147483647 caracteres. O número máximo de caracteres pode ser especificado entre parênteses.

BIGINT: guarda números do tipo inteiro. Suporta de -9223372036854775808 até 9223372036854775807 caracteres.
FLOAT(tamanho,decimal): guarda números REAIS. O número máximo de caracteres pode ser especificado entre parênteses. Deve-se especificar o tamanho inteiro e o tamanho numérico da coluna.
DOUBLE(tamanho,decimal): guarda números REAIS. O número máximo de caracteres pode ser especificado entre parênteses. Deve-se especificar o tamanho inteiro e o tamanho numérico da coluna. Esse tipo armazena uma quantidade maior de número do que os campos do tipo FLOAT.

Tipos para armazenamento de data

DATE: tipo de campo que vai armazenar datas no: YYYY-MM-DD, onde Y refere-se ao ano, M ao mês e D ao dia;
DATETIME: a combinação de data e tempo, no formato YYYY-MMDD HH:MI:SS;
TIME: armazena horas, minutos e segundos no formato HH:MI:SS.

Como criar uma tabela

Para a criação de uma tabela no banco de dados devemos informar o nome da tabela juntamente com as informações de definições das colunas pertencentes a tabela. No MySQL o comando para a criação de tabelas seguinte sintaxe:

```
CREATE TABLE NOME_DA_TABELA(
NOME_COLUNA TIPO_DE_DADOS RESTRIÇÕES,
NOME_COLUNA TIPO_DE_DADOS RESTRIÇÕES,
...
NOME_COLUNA TIPO_DE_DADOS RESTRIÇÕES,
)
```

Onde:
NOME_DA_TABELA: Identificador que será atribuido a tabela, esta informação é obrigatória. Exemplo clientes.
NOME_COLUNA: Identificador da coluna dentro da tabela, esta informação é obrigatória. Exemplo ID.
TIPO_DE_DADOS: definição de qual tipo de dado será armazenado na coluna, esta informação é obrigatória. Exemplo INT.
RESTRIÇÕES: Tipo de restrições que a coluna terá, esta informação é opcional.

Exemplo:

```
CREATE TABLE PESSOAS(
ID INT primary key,
NOME varchar(255)
```

```
)
```

Como alterar uma tabela

Muitas vezes após a criação de uma tabela se faz necessário realizarmos alterações para a inclusão, alteração ou remoção de alguma coluna ou restrição. Para realizarmos essas operações devemos utilizar um dos comando abaixo:

Adicionar uma nova coluna

```
ALTER TABLE NOME_DA_TABELA ADD COLUMN NOME_COLUNA TIPO_DE_DADOS RESTRIÇÕES;
```

Alterar uma coluna existente

```
ALTER TABLE NOME_DA_TABELA CHANGE NOME_COLUNA NOVO_NOME_COLUNA TIPO_DE_DADOS RESTRIÇÕES;
```

Remover uma coluna existente

```
ALTER TABLE NOME_DA_TABELA DROP NOME_COLUNA;
```

Onde:
NOME_DA_TABELA: Identificador da tabela que será alterada.
NOME_COLUNA: Identificador da coluna dentro da tabela.
NOVO_NOME_COLUNA: Novo Identificador que será atribuido à coluna dentro da tabela
TIPO_DE_DADOS: definição de qual tipo de dado será armazenado na coluna

Exemplos:

```
ALTER TABLE PESSOAS ADD COLUMN DATA_NASC DATETIME;
ALTER TABLE PESSOAS CHANGE DATA_NASC DATA_NASCIMENTO DATETIME;
ALTER TABLE PESSOAS DROP DATA_NASCIMENTO;
```

Como excluir uma tabela

Para removermos uma tabela da base de dados utilizamos o seguinte comando:

```
DROP TABLE NOME_DA_TABELA;
```

Onde:
NOME_DA_TABELA: Identificador da tabela que será removida.

Como incluir registros em uma tabela

Para incluirmos registros em uma tabela utilizamos o comando INSERT com a seguinte sintaxe:

```
INSERT INTO NOME_DA_TABELA(LISTA_DE_COLUNAS)
VALUES(LISTA_DE_VALORES)
```

NOME_DA_TABELA: Identificador da tabela.
LISTA_DE_COLUNAS: Identificadores das colunas que serão populadas separadas por vírgula.
LISTA_DE_VALORES: Valores que serão atribuídos para as colunas informadas, estes valores também são separados por vírgula.

Exemplos:

Inserir um único registro

```
INSERT INTO PESSOAS(ID,NOME) VALUES (1,'JOÃO');
```

Inserir vários registros:

```
INSERT INTO PESSOAS(ID,NOME)
VALUES(2,'MARIA'),(3,'ADÃO'),(4,'EVA');
```

Como selecionar registros

Para realizarmos a seleção dos registro de uma tabela utilizamos o comando SELECT com a seguinte definição:

```
SELECT LISTA_DE_COLUNAS FROM NOME_DA_TABELA;
```

Onde:
NOME_DA_TABELA: Identificador da tabela.
LISTA_DE_COLUNAS: identificadores das colunas a serem utilizadas na consulta, caso queiramos buscar todas as colunas podemos utilizar '*'

Exemplos:

```
SELECT ID, NOME FROM PESSOAS;
SELECT * FROM PESSOAS;
```

Muitas vezes não desejamos consultar todos os registros, mas apenas alguns registros conforme alguma determinada regra de restrição. Para aplicarmos essas regras de restrições utilizamos a cláusula WHERE.

Exemplo:

```
SELECT * FROM PESSOAS WHERE ID = 1;
```

Como atualizar registros

Para a atualização de um ou mais registros utilizamos o comando UPDATE conforme sintaxe abaixo:

```
UPDATE NOME_DA_TABELA SET NOME_COLUNA_1=NOVO_VALOR_1,
NOME_COLUNA_2=NOVO_VALOR_2,...,NOME_COLUNA_N=NOVO_VALOR_N WHERE
RESTRIÇÕES DE SELEÇÃO.
```

Onde:
NOME_DA_TABELA: Identificador da tabela.
NOME_COLUNA: identificador da coluna a ser alterada.
NOVO_VALOR: Valor a ser atribuído para a coluna.
RESTRIÇÕES DE SELEÇÃO: restrições para identificação dos registros a serem atualizados.

ATENÇÃO.
Caso não seja utilizado a cláusula WHERE, todos os registro da tabela serão atualizados com os valores informados.

Exemplo

```
UPDATE PESSOAS SET NOME='JOÃO DA SILVA BRASIL' WHERE ID=1;
```

Como remover registros

Para removermos um registro da tabela devemos utilizar o comando DELETE conforme abaixo:

```
DELETE FROM NOME_DA_TABELA WHERE RESTRIÇÕES DE SELEÇÃO
```

Onde:
NOME_DA_TABELA: identificador da tabela;
RESTRIÇÕES DE SELEÇÃO: restrições para identificação dos registros a serem removidos.

ATENÇÃO.
Caso não seja utilizada a cláusula WHERE, todos os registros da tabela serão removidos.

Exemplo:

```
DELETE FROM PESSOAS WHERE NOME = 'ADÃO'
```

Caso queiramos realizar uma espécie de reset na tabela apagando todas as suas informações podemos utilizar o comando TRUNCATE.

```
TRUNCATE TABLE NOME_DA_TABELA
```

Onde:
NOME_DA_TABELA: identificador da tabela;

Exemplo:

```
TRUNCATE TABLE Pessoas;
```

Capítulo 05 - Operadores

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Operadores Matemáticos
- Operadores de comparação
- Operadores lógicos

Operadores matemáticos

Com o uso dos operadores matemáticos podemos realizar operações dentro de um select. Estas operações serão realizadas em todos os registros selecionados.

Operador de adição

Para realizarmos a adição entre o valor de duas ou mais colunas utilizamos o operador '+';

Exemplo:

```
select valor_pedito + valor_multa from Pedido;
```

Operador de subtração

Para realizarmos a subtração entre o valor de duas ou mais colunas utilizamos o operador '-';

Exemplo:

```
select valor_pedito - valor_desconto from Pedido;
```

Operador de multiplicação

Para realizarmos a multiplicação entre o valor de duas ou mais colunas utilizamos o operador '*';

Exemplo:

```
select valor_produto * quantidade from Pedido_Item;
```

Operador de divisão

Para realizarmos a divisão entre o valor de duas ou mais colunas utilizamos o operador '/';

Exemplo:

```
select valor_total / qt_parcelas from Pedido;
```

Operadores de comparação

Para filtrar os dados da nossa busca podemos utilizar um ou mais operadores de comparação, estes operadores realizam a comparação entre dois valores e retornam um resultado lógico verdadeiro ou falso.

Operador Maior Que '>'

O operador maior que irá retornar verdadeiro sempre que o valor a esquerda for maior do que o valor a direita.

Exemplo

```
Select * from Pessoas where ID > 10; --retorna todas as pessoas com id --maior que 10. neste caso a pessoa com id 10 não constará no resultado
```

Operador Maior ou Igual '>='

O operador maior ou igual irá retornar verdadeiro sempre que o valor a esquerda for maior ou igual ao valor a direita.

Exemplo

```
Select * from Pessoas where ID >= 5; --retorna todas as pessoas com id maior ou igual a 5. neste caso a pessoa com id 5 constará no resultado
```

Operador Menor Que '<'

O operador menor que irá retornar verdadeiro sempre que o valor a esquerda for menor do que o valor a direita.

Exemplo

```
Select * from Pessoas where ID < 10; --retorna todas as pessoas com id menor que 10. neste caso a pessoa com id 10 ou superior não constarão no resultado.
```

Operador Menor ou Igual '<='

O operador menor ou igual irá retornar verdadeiro sempre que o valor a esquerda for menor ou igual ao valor a direita.

Exemplo

```
Select * from Pessoas where ID <= 5; --retorna todas as pessoas com id menor ou igual a 5. neste caso a pessoa com id 5 constará no resultado
```

Operador Diferente de '<>'

O operador diferente irá retornar verdadeiro sempre que o valor a esquerda for diferente do valor a direita.

Exemplo

```
Select * from Pessoas where ID <> 5; --retorna todas as pessoas exceto a de id 5.
```

Operador Igual '='

O operador igual irá retornar verdadeiro sempre que o valor a esquerda for o mesmo valor a direita.

Exemplo

```
Select * from Pessoas where ID = 5; --retorna somente a pessoa com id 5.
```

Operadores Lógicos

Os operadores lógicos são utilizados para unirmos duas ou mais comparações. Estes operadores irão comparar o valor lógico retornado pelas comparações e irão retornar um novo valor lógico.

Operador 'AND'

O operador AND irá retornar verdadeiro sempre que as duas condições utilizadas na comparação sejam verdadeiros.
Exemplo

```
Select * from pessoas where id >= 5 AND id <=10; --retorna todas as pessoas que possuam id entre 5 e 10
```

Operador 'OR'

O operador OR irá retornar verdadeiro sempre que ao menos uma das condições utilizadas na comparação sejam verdadeiras.
Exemplo

```
Select * from pessoas where id <= 5 OR id >=10; --retorna todas as pessoas exceto as que possuem id entre 6 e 9
```

Operador 'NOT'

O operador NOT irá retornar o valor inverso ao valor lógico contido na expressão posterior.
Exemplo

```
Select * from pessoas where not (id <= 5 OR id >=10); --retorna todas as pessoas que possuem id entre 6 e 9
```

OBS. Quando queremos comprarar um valor não nulo utilizamos a expressão IS NOT NULL.

Exemplo

```
select * from Pedidos where data_finalizacao is not null -- retorna todos os pedidos finalizados (possuem data de finalização).
```

Capítulo 06 - Funções de manipulação de texto

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Função substr
- Função length
- Função lower
- Função Ucase
- Função concat
- Função concat_ws

Função substr

Muitas vezes queremos recuperar apenas parte de um determinado texto em uma coluna, para realizarmos essa operação podemos utilizar a função substr.

Exemplo

Data a tabela pessoas conforme abaixo:

```
id| nome |sobrenome |endereco
1 |João  |Silva     |Rua Amazonas 1000, bairro Garcia, Blumenau
```

A seguinte consulta irá trazer apenas as 10 primeiras letras do endereço

```
select 8 nome, substr(endereco,1,10) from pessoas.
```

Função length

Retorna a quantidade de caracteres de um texto.

Exemplo

```
select nome, length(nome) as letras from Pessoas. -- retorna o nome e a quantidade de letras do nome nome das pessoas

select * from pessoas where length(nome) > 3 -- retorna as pessoas que possuam mais de 3 letras no nome
```

Função lower

Retorna o texto em caixa baixa

Exemplo:

```
select lower(nome) from pessoas; -- retorna o nome de todas as pessoas em caixa baixa.

select * from pessoas where lower(nome) = 'joao' -- retorna as pessoas que possuem o nome 'joao'
```

Função Ucase

Retorna um texto no formato de caixa alta.

```
select ucase(nome) from pessoas; -- retorna o nome de todas as pessoas em caixa alta.

select * from pessoas where ucase(nome) = 'JOAO' -- retorna as pessoas que possuem o nome 'JOAO'
```

Função concat

Concatena dois ou mais textos.

Exemplo

```
select concat(nome, ' ', sobrenome) from pessoas; -- retorna o nome completo de todas as pessoas.
```

Função concat_ws

Concatena duas ou mais colunas assim como em concat, a diferença é que precisamos informar como primeiro parâmetro qual será o caracter separador.

Exemplo

```
select concat_ws(' ',nome, sobrenome) from pessoas; -- retorna o nome completo de todas as pessoas.
```

Capítulo 07 - Funções Básicas

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Função count
- Função max
- Função min
- Função avg
- Função sum
- Comando distinct
- Comando order by
- Comando limit

Função count

Retorna a quantidade total de registros.

Exemplo

```
Select count(*) from pessoas -- retorna a quantidade de registros da tabela pessoas
```

Função max

Retorna o maior valor de uma determinada coluna.

Exemplo

```
Select max(valor) from produtos --retorna o produto de maior valor
```

Função min

Retorna o menor valor de uma determinada coluna.

Exemplo

```
Select min(valor) from produtos --retorna o produto de menor valor
```

Função avg

Retorna o valor da média simples de todos os registros de uma determinada coluna.
Exemplo

```
select avg(valor) from produtos -- seleciona média de preços dos produtos cadastrados
```

Função sum

Retorna o valor da soma de todos os registros de uma determinada coluna

Exemplo

```
Select sum(valor) from produtos -- retorna o valor da soma de todos os produtos
```

Comando distinct

Para realizarmos uma consulta e ignorarmos os valores repetidos podemos utilizar a cláusula distinct;

Exemplo

```
select distinct nomes from pessoas -- irá retornar o nome das pessoas sem repetição
```

Comando order by

Para realizar a ordenação dos dados podemos utilizar o comando order by. Este comando recebe o nome das colunas que serão levadas em consideração para a ordem e se a ordenação deve ser de forma ascendente ou descendente.

Exemplo

```
select * from produtos order by valor desc -- irá trazer todos os produtos trazendo primeiramente os de maior valor;
```

Comando limit

Para limitar a quantidade de registros em uma consulta podemos utilizar o comando limit

Exemplo

```
select * from pessoas limit 10; --busca as pessoas com um limite de 10 registros
```

Capítulo 08 - Funções de manipulação de data

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Função curdate
- Função now
- Função sysdate
- Função curtime
- Função dayname
- Função dayofmonth
- Função last_day
- Função date_format
- Função str_to_date

Função curdate

Retorna a data atual

Exemplo

```
select curdate(); -- retorna a data atual
```

Função now

Retorna a data e hora atual.

Exemplo

```
select now() --retorna data e hora atual
```

Função sysdate

Assim como now, sysdate retorna a data e hora atual.

Exemplo

```
select sysdate() --retorna data e hora atual
```

Função curtime

Retorna apenas a hora atual.

Exemplo

```
select curtime() --retorna a hora atual
```

Função dayname

Retorna no nome do dia referente a uma data.

Exemplo

```
select dayname(curdate()) - retorna o nome do dia atual
```

Função dayofmonth

Retorna apenas o dia de do mês em uma data

Exemplo

```
select dayofmonth(curdate())
```

Função last_day

Retorna a última data do mês com base em uma data.

Exemplo

```
select last_day(curdate())
```

Função date_format

Realiza a formatação de uma data de acordo com uma formatação.

Exemplo

```
select date_format(curdate(),get_format(date,'EUR')) --realiza a formatação de acordo com o padrão Europeu
```

Função str_to_date

Converte um texto para o formato data de acordo com uma formatação:

Exemplo

```
select str_to_date('05.10.2017',get_format(date,'EUR'))
```

Capítulo 09 - Normalização de dados

Objetivos

Neste capítulo veremos os seguintes tópicos:

- O que é normalização
- Primeira forma normal
- Segunda forma normal
- Terceira forma normal

O que é normalização

Normalização consiste na forma de organização dos dados dentro das tabelas do banco de modo, que esta estrutura tenha uma melhor performance e consistência. Para dizermos que um banco está normalizado, devemos verificar algumas regras, estas regras são denominadas formas normais. Aqui neste curso iremos tratar das três primeiras formas.

Primeira forma normal

Para dizermos que um banco de dados encontra-se na primeira forma normal devemos seguir duas regras:
1 - As informações devem estar divididas em tabelas formadas por linhas e colunas e cada célula deve possuir apenas um valor
2 - cada coluna deve armazenar valores únicos ou valores chaves para outras tabelas.

Segunda forma normal

A segunda forma normal impoe a regra de que a base de dados deve atender a primeira forma normal e ainda cada atributo não chave na tabela deve depender exclusivamente de todas as chaves da tabela

Terceira forma normal

A terceira forma normal diz que a base de dados deve atender a segunda forma normal e que os atributos não chave dependam exclusivamente da chave primária da tabela.

Capítulo 10 - Relacionamento entre tabelas

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Chave primária
- Chave estrangeira
- Relacionamento 1 x 1
- Relacionamento 1 x N
- Relacionamento N x N

Chave Primária

Uma prática importante no momento da criação de uma tabela, é a definição de qual coluna representará a sua chave primária. A chave primária é utilizada para definir um identificador único para um registro em meio ao conjunto de registros definidos na tabela, o valor deste identificador não pode ser repetido em nenhum outro registro da mesma tabela. A chave primária pode ser de dois tipos, sintética ou natural. Chamamos uma chave primária de sintética, quanto o campo é adicionado ao registro exclusivamente para criarmos uma chave primária, o exemplo mais simples é a inclusão de um campo id em um registro da tabela. Quanto utilizamos um campo que naturalmente faz parte do registro, dizemos que esta chave primária é uma chave natural, para exemplificarmos esse tipo de chave podemos tomar como exemplo uma tabela de chamada PessoaFísica onde temos os atributos , nome e cpf, naturalmente o atributo cpf é um valor único para cada registro pois não existem duas pessoas com o mesmo número de cpf, neste caso podemos utilizar o campo cpf como chave primária para a tabela PessoaFísica.

Para identificarmos qual campo será definido como chave primária utilizamos o identificador primary key. Existem três formas básicas de definirmos a chave primária de uma tabela:

1º) - No momento da definição do campo da tabela, jutamente com suas restrições.

Exemplo

```
create table Pessoas(
id int not null primary key,
nome varchar(150)
)
```

2º – No momento da definição dos campos da tabela, mais após a declaração de todos os campos.

Exemplo

```
create table Pessoas(
id int not null,
nome varchar(150),
primary key(id)
)
```

3º – Após a criação da tabela juntamente com o comando alter table.

Exemplo

```
create table Pessoas(
id int not null,
nome varchar(150)
);
alter table pessoas add constraint pk_pessoas primary key(id)
```

Chave estrangeira

Um banco de dados relacional possui esse nome devido a capacidade de relacionarmos informações entre tabelas através da inclusão de colunas de relacionamento, essas colunas de relacionamento são chamadas de chave estrangeira.

Quando definimos uma chave estrangeira em uma determinada tabela, estamos instruindo o banco que os valores a serem adicionados na respectiva coluna, só podem ser um dos valores contidos na coluna de referência da tabela que queremos ligar.

Podemos tomar como exemplo as tabelas Produtos e Fabricantes, onde queremos realizar a ligação entre um produto e seu fabricante. Para realizarmos esta ligação podemos realizar o mapeamento das tabelas conforme o exemplo abaixo:

Exemplo

```
create table Fabricantes(
id int not null primary key,
nome varchar(200) not null
)

create table produtos(
id int not null primary key,
```

```
id_fabricante int not null,
constraint fk_fabricantes foreign key(id_fabricante) references
fabricantes(id),
nome varchar(200)
);
```

Através do comando foreign key(id_fabricante) references fabricantes(id), definimos que na tabela produtos os valores possíveis para o campo id_fabricante serão apenas os valores contidos da coluna id da tabela fabricantes.

Relacionamento 1 x 1

Quanto um registro de uma tabela se relaciona com apenas um único registro de outra tabela, dizemos que a relação entre as tabelas é de 1 para 1.

Exemplo

Em um sistema onde um usuário pode conter apenas um e-mail poderíamos realizar o mapeamento das tabelas Usuarios e Emails da seguinte forma:

```
create table emails(
id int not null primary key,
e-mail varchar(200) not null
)

create table usuários(
id int not null primary key,
id_email int not null,
nome varchar(200) not null
constraint fk_email foreign key(id_email) references email(id)
)
```

Na prática esse tipo de relacionamento não é tão utilizado, pois o e-mail poderia ser um campo específico da tabela usuários.

Relacionamento 1 x N

Quanto um registro de uma tabela se relaciona com um ou mais registros de outra tabela, dizemos que a relação entre as tabelas é de 1 para N.

Para fins de exemplificarmos esse tipo de relacionamento podemos tomar como base um sistema de vendas, onde possuímos as tabelas pedidos, que referencia um pedido de compra, e pedidosItem, que realiza a ligação de um produto com em um determinado pedido. Neste tipo de

ligação podemos identificar que em um único pedido teremos vários itens adicionados.

```
Create table Pedido(
id int not null primary key,
codigo varchar(10) not null,
data_criacao date
)

create table pedidoItem(
id int not null primary key,
id_produto not null,
id_pedido not null,
quantidade not null,
constraint fk_produto foreign key(id_produto) references produto(id),
constraint fk_pedido foreign key(id_pedido) references pedido(id),
)
```

Relacionamento N x N

Quanto vários registros de uma tabela se relacionam com vários registros de outra tabela, dizemos que a relação entre as tabelas é de N para N. Para representar o relacionamento N x N criamos uma terceira tabela chamada de tabela de ligação, esta tabela irá conter um relacionamento de 1 x N entre as tabelas 1 e 2.

Exemplo

Imagine que precisamos modelar um sistema para livraria onde para um determinado livro podem ser atribuídas várias tags, e cada tag pode estar associada a mais de um livro, neste caso podemos modelar o sistema da seguinte forma:

```
create table livros(
id int not null primary key,
titulo varchar(200) not null
)
create table tag(
id int not null primary key,
nome varchar(200) not null
)
```

Por fim podemos criar uma tabela de ligação entre livros e tags:

```
create table livro_tag(
id int not null primary key auto_increment,
```

```
id_livro int not null ,
id_tag int not null
);
alter table livro_tag
add constraint fk_livro foreign key(id_livro)
references livro(id);
alter table livro_tag
add constraint fk_tag foreign key(id_tag)
references tag(id);
```

Desta forma para sabermos quais são as tags de um determinado livro precisaremos consultar a tabela livro tag.

Capítulo 11 - Estudo de caso - definindo as tabelas do sistema

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Definição de um sistema para livraria

Definição de um sistema para livraria

Para dar continuidade aos nossos estudos vamos realizar a definição de um sistema para livraria, para esse sistema foram levantados os seguintes requisitos:

R1 – Um livro pode possuir mais de um autor
R2 – Um livro deve possuir uma categoria e podem existir vários livros com a mesma categoria
R3 – Podem ser atribuídas várias tags para um determinado livro e uma tag pode estar associada a vários livros.
R4 – A livraria deve possuir um cadastro de seus associados.
R5 – Cada associado pode alugar um ou mais livros durante uma única locação
R6 – O valor de cada locação será definido pela quantidade de livros retirados, e todos os livros possuem o mesmo valor de locação, entretanto o valor de colação de um livro pode mudar em períodos distintos.
R7 – Livros de mesmo isbn devem possuir um código sequencial para diferenciá-los.

Após um estudo dos requisitos chegamos ao seguinte mapeamento:

Definição da tabela Autor:

```
CREATE TABLE autor(
id int not null primary key auto_increment,
nome varchar(50) not null,
sobrenome varchar(100) not null,
data_nascimento date not null
);
```

Definição da tabela Categoria:

```
create table categoria(
```

```
id int not null primary key auto_increment,
nome varchar(100) not null
);
```

Definição da tabela Tag:

```
create table tag(
id int not null primary key auto_increment,
nome varchar(100) not null
);
```

Definição da tabela Livro:

```
create table livro(
id int not null primary key auto_increment,
titulo varchar(200) not null,
data_publicacao date not null,
edicao int not null,
codigo_sequencial int not null,
id_categoria int not null
);
alter table livro
add constraint fk_categoria foreign key(id_categoria)
references categoria(id);
```

Definição da tabela Livro_Autor

```
create table livro_autor(
id int not null primary key auto_increment,
id_livro int not null ,
id_autor int not null
);

alter table livro_autor
add constraint fk_livro foreign key(id_livro)
references livro(id);
alter table livro_autor
add constraint fk_autor foreign key(id_autor)
references autor(id);
```

Definição da tabela livro_tag:

```
create table livro_tag(
id int not null primary key auto_increment,
id_livro int not null ,
id_tag int not null
```

```
);
alter table livro_tag
add constraint fk_livro foreign key(id_livro)
references livro(id);
alter table livro_tag
add constraint fk_tag foreign key(id_tag)
references tag(id);
```

Definição da tabela sócio

```
CREATE TABLE socio(
id int not null primary key auto_increment,
nome varchar(50) not null,
sobrenome varchar(100) not null,
data_nascimento date not null,
profissao varchar(200),
sexo varchar(1) not null
);
```

Definição da tabela locação

```
create table locacao(
id int not null primary key auto_increment,
data_retirada date not null,
data_previsao_devolucao date not null,
data_devolucao date,
id_socio int not null
);
alter table locacao
add constraint fk_socio foreign key(id_socio)
references socio(id);
```

Definição da tabela locacao_livro:

```
create table locacao_livro(
id int not null primary key auto_increment,
id_livro int not null ,
id_locacao int not null
);

alter table locacao_livro
add constraint fk_livro_locacao_livro foreign key(id_livro)
references livro(id);
alter table locacao_livro
add constraint fk_locacao_locacao_livro foreign key(id_locacao)
references locacao(id);
```

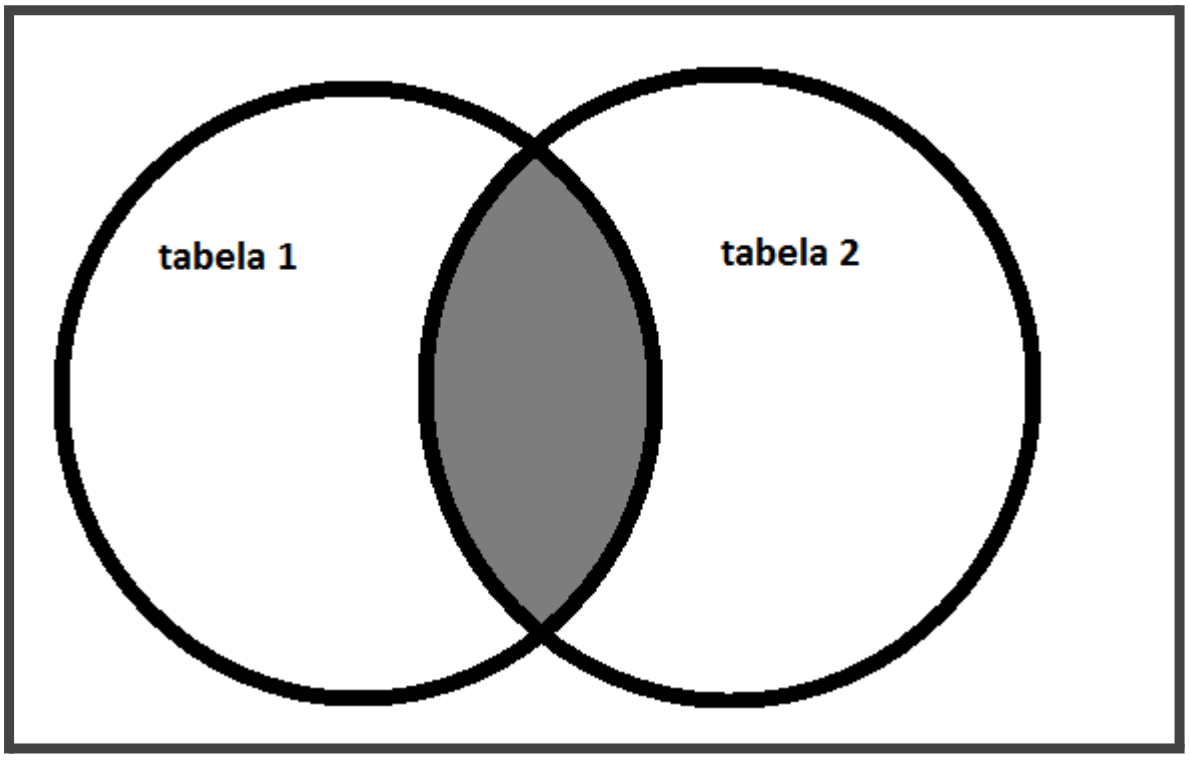
Capítulo 12 - Unindo tabelas para realização de consultas

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Comando inner join
- Command left join
- Comando right join
- Comando union

Comando inner Join



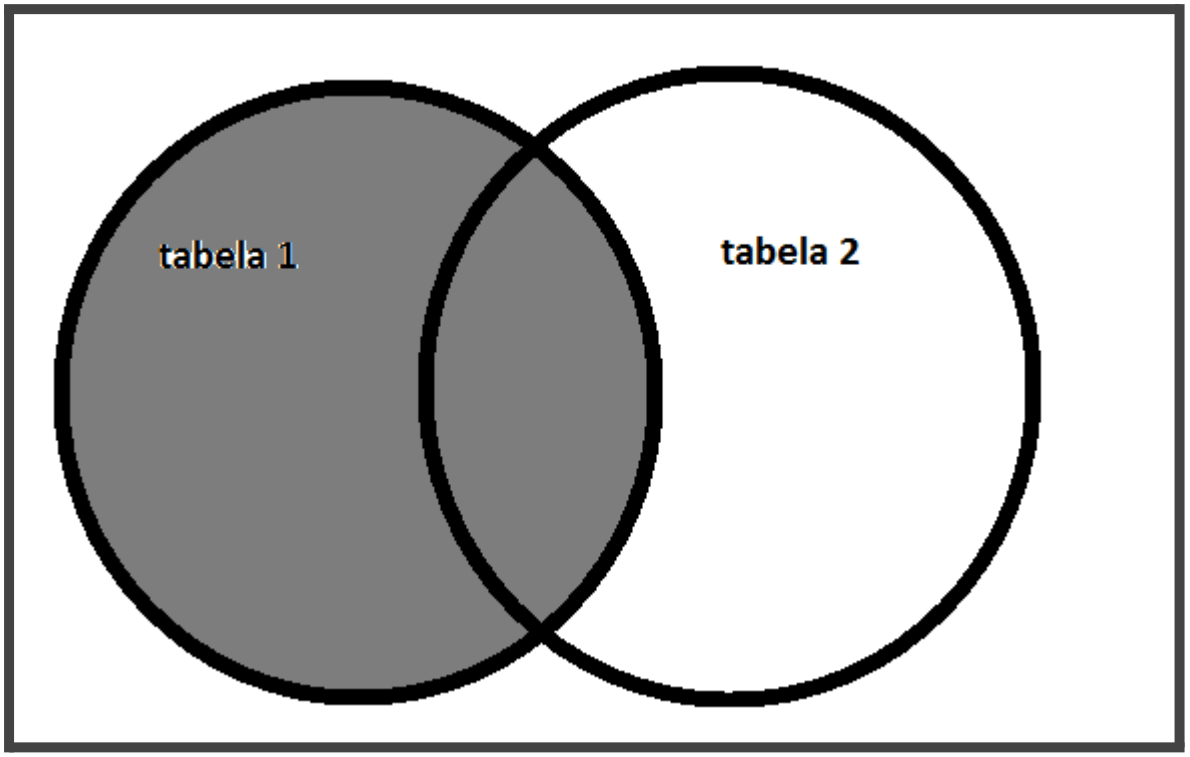
O comando inner join serve para unirmos as informações entre duas ou mais tabelas e retorna somente as informações que possuem ligação entre as tabelas relacionadas.

Por exemplo para consultarmos as o título de um livro juntamente com o nome de sua categoria podemos utilizar o seguinte comando:

```
Select livro.titulo,
```

```
categoria.nome
from livro
inner join categoria on categoria.id = livro.id_categoria
```

Comando left join



O comando left join serve para buscarmos as informações relacionadas entre duas ou mais tabelas, ele irá trazer todas as informações da tabela da esquerda acompanhado de suas relações na tabela da direita, caso não haja relação entre algum item da tabela da esquerda com a da direita, o valor referente a tabela da direita será nulo.

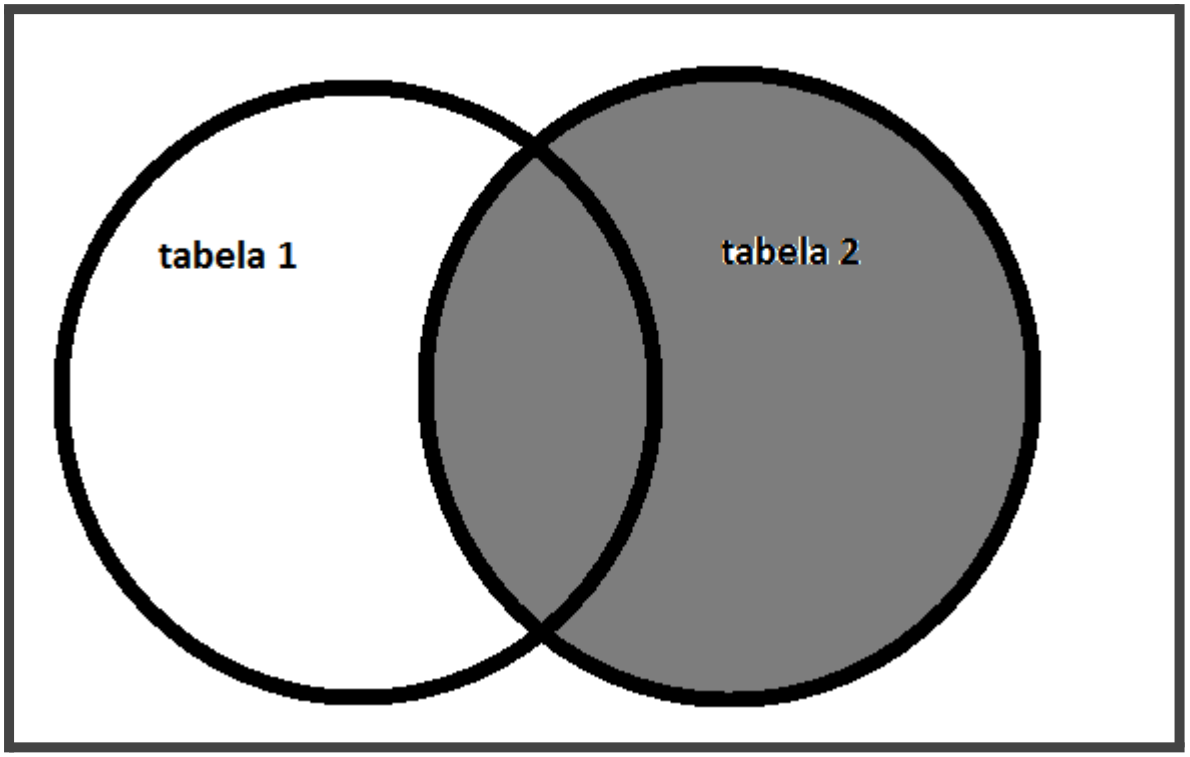
Exemplo

Podemos consultar os livros com suas tags, mas caso um livro não possua tag queremos relacioná-lo também em nossa consulta.
Para isso podemos utilizar o seguinte comando:

```
select livro.titulo,
tag.nome
from livro
left join livro_tag on livro_tag.id_livro = livro.id
left join tag on tag.id = livro_tag.id_tag
```

Percebam que para esta busca precisamos incluir em nossa consulta a tabela de ligação livro_tag.

Comando right join



O comando right join serve para buscarmos as informações relacionadas entre duas ou mais tabelas, ele irá trazer todas as informações da tabela da direita acompanhado de suas relações na tabela da esquerda, caso não haja relação entre algum item da tabela da direita com a da esquerda, o valor referente a tabela da esquerda será nulo.

Comando Union

O comando union é utilizado quando queremos unir o resultado entre dois comandos de consultas.

Exemplo

Para trazermos o nome de todas as tags acompanhado do nome de todas as categorias podemos utilizar o seguinte comando:

```
select tag.nome from tag
```

```
union
select categoria.nome from categoria
```

Capítulo 13 - Funções Intermediárias

Neste capítulo veremos os seguintes tópicos:

- Cláusula group by
- Cláusula having

Cláusula group by

Quando utilizamos funções para realizarmos operações sobre os dados selecionados, normalmente precisamos que estas funções executem suas operações dentro de um determinado conjunto de dados, para que isso seja possível utilizamos a cláusula group by para agrupar os dados corretamente.

Exemplo

Imagine que queremos saber quantas tags estão relacionadas a um determinado livro, para isso, se simplesmente utilizássemos a função count sem um agrupador, ela retornaria a soma de todas as tags, mas como desejamos que esta contagem seja agrupada por livros, necessitados da cláusula group by.

```
select livro.titulo,
count(tag.nome)
from livro
inner join livro_tag on livro_tag.id_livro = livro.id
inner join tag on tag.id = livro_tag.id_tag
group by livro.nome
```

caso removêssemos a cláusula group by, o retorno da consulta estaria incorreto.

Cláusula having

A cláusula having é necessária quando desejamos utilizar o resultado de uma função para realizar o filtro de dados.

Exemplo

Na consulta do exemplo anterior, se quiséssemos trazer somente os livros que possuem duas ou mais tags associadas precisaríamos adicionar a cláusula having.

```
select livro.titulo,
count(tag.nome)
from livro
inner join livro_tag on livro_tag.id_livro = livro.id
inner join tag on tag.id = livro_tag.id_tag
group by livro.titulo
having count(tag.nome) > 1
```

Capítulo 14 - Criação de Views

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Como criar uma view
- Como selecionar dados de uma view
- Como alterar uma view
- Como excluir uma view

Como criar uma view

Muitas vezes realizamos as mesmas consultas várias vezes, seja porque necessitados apenas das informações de alguns poucos registros de uma determinada tabela, ou seja porque necessitamos das informações de um conjunto de tabelas. Nessas situações uma boa prática é a criação de views. Uma view é o armazenamento do resultado de uma consulta em um tipo de tabela virtual, após criarmos a view podemos realizar várias consultas como se ela fosse uma tabela do sistema.

Para criar uma view utilizamos o comando create view seguido do select responsável pela busca dos dados.

Exemplo

Podemos criar uma view que irá armazenar o resultado da consulta dos livros juntamente com o nome de sua respectiva categoria.

```
create or replace view livros_com_categoria as
select livro.titulo,
categoria.nome
from livro
inner join categoria on categoria.id = livro.id_categoria
```

Como selecionar dados de uma view

Para selecionar os dados de uma view podemos utilizar o comando select da mesma forma que utilizamos em uma tabela normal.

Exemplo

```
select * from livros_com_categoria where categoria = 'programação'
```

Como alterar as informações de uma view

Para que possamos utilizar o comandos insert, update ou delete em uma view, o comando de criação da view não pode conter joins ou funções agregadoras como o group by. Se seguirmos estas regras podemos utilizar os comandos de insert, update ou delete como se estivéssemos utilizando uma tabela normal.

Como excluir uma view

Para excluir uma view utilizamos o comando drop view.

Exemplo

```
drop view livros_com_categoria
```

Capítulo 15 - Indexação de tabelas

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Como criar um índice
- Como remover um índice

Como criar um índice

Muitas vezes nossas tabelas possuem muitos dados e a performance das consultas começa a ser penalizada, para melhorarmos a performance podemos dizer ao banco de dados que ele deve iniciar a consulta através das informações de algumas colunas específicas, esse tipo de configuração é chamado de indexação.
Para adicionarmos um novo índice em uma tabela utilizamos o comando alter table tabela add index nome do indice (colunas_do_indice).

Exemplo

```
alter table locacao add index idx_socio(id_socio)
```

Para melhorar nossas consultas devemos no momento de criar um índice, optar por campos que normalmente sejam utilizados em uma pesquisa, ordenação, agrupamento ou filtro.

Como remover um índice

Para remover um índice utilizamos o comando alter table em conjunto o comando drop index nome_do_indice.

Exemplo

```
alter table locacao drop index idx_socio
```

Capítulo 16 - Como realizar o backup dos dados

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Como realizar o backup

Como realizar o backup do banco de dados

Para a realização do backup o mysql possui um aplicativo de linha de comando chamado mysqldump esse aplicativo fica dentro da pasta bin localizada dentro do diretório de instalação do mysql.

Para realizarmos o backup precisamos executar o comando

```
mysqldump -u root -p curso_db >c:/bkp_tabelas.sql
```

onde:
-u root indica o nome do usuário
-p instrui o mysql a solicitar a senha do usuários
curso_db é o nome da base de dados que se deseja realizar o backup
c:/bkp_tabelas.sql é o caminho completo do arquivo onde se deseja armazenar o backup.

Capítulo 17 - Como restaurar as informações da base através do backup

Objetivos

Neste capítulo veremos os seguintes tópicos:

- Como restaurar a base de dados

Como restaurar um backup

Para realizar a restauração de um backup o mysql possui um aplicativo de linha de comando chamado mysql, esse aplicativo fica dentro da pasta bin localizada dentro do diretório de instalação do mysql.

Para restaurar a base de dados devemos acessar o prompt de comando e acessar a pasta bin dentro do diretório de instalação do mysql. Dentro da pasta bin devemos digitar o seguinte comando:

```
mysql -h localhost -u root -p curso_db < c:/bkp_tabelas.sql
```

onde:
-h localhost instrui o mysql a acessar o servidor instalado na própria máquina
-u root indica que deve ser utilizado o usuário root
-p indica que deve ser solicitada a senha do usuário
curso_db = base de dados onde deve ser restaurado o backup
c:/bkp_tabelas.sql = caminho completo do arquivo de backup.