Banco de Dados

PROF.: JOÃO VICTOR PEREIRA

Introdução

Para que hajam sistemas funcionais em relação a cadastros, dados precisam ser armazenados e recuperados. E, para que isto aconteça, é preciso criar estruturas que permitam o armazenamento de forma íntegra, facilitando as consultas e manipulações dos dados.

SGBD

O Sistema Gerenciador de Banco de Dados (SGBD) é um conjunto de programas que permite ao usuário criar e manter um banco de dados. O SGBD possibilita a definição, construção, manipulação e compartilhamento de bancos de dados entre diversos usuários e aplicações.

Definição: trata a respeito de como os dados serão armazenados (forma, tipo de dados, estruturas e restrições).

Construção: armazenamento dos dados em algum meio controlado pelo SGBD.

Manipulação: funções de consulta a dados ou atualização nos dados.

Compartilhamento: permissão para que vários usuários e/ou aplicações acessem e mantenham simultaneamente esses dados.

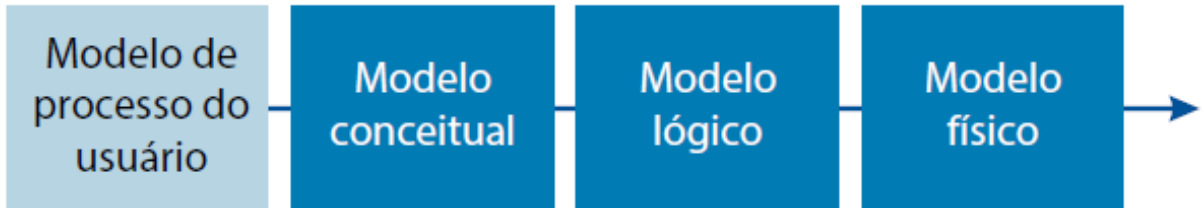
MODELAGEM DE DADOS

Para a criação adequada de um banco de dados, é necessário realizar a modelagem deste banco de dados, primeiramente. A modelagem é uma tarefa essencial para que o banco esteja alinhado com as necessidades dos usuários que vão utilizar o sistema conectado a este banco de dados.

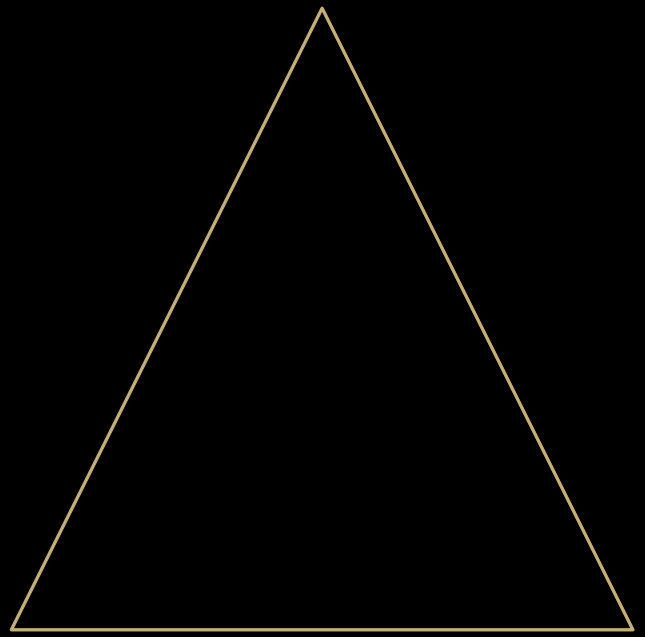
Um modelo de banco de dados é uma descrição dos tipos de informações que estão armazenadas em um banco de dados. Para a construção deste modelo, é usada uma linguagem de modelagem de dados, que podem ser classificadas de acordo com a forma de apresentar modelos, em linguagens textuais ou linguagens gráficas.

ARQUITETURA DE 3 ESQUEMAS

Esta arquitetura proporciona uma forma de entendimento do modelo de negócios que precisa ser modelado de forma clara e gradativa. A representação acontece da seguinte forma:



ARQUITETURA EM 3 ESQUEMAS

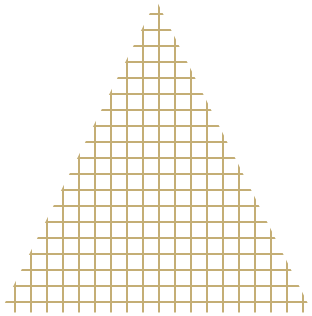


Modelo de Processo de Usuário: como as atividades, tarefas e processos do dia a dia dos usuários para os quais se pretende desenvolver uma solução que envolva banco de dados.

Modelo Conceitual: utiliza os conceitos de banco de dados, mas sem apego a tecnologias. Nesta etapa é modelado o domínio de negócio, delimitando o minimundo (cenário a ser atendido), sem detalhes de modelagem.

Modelo Lógico: modelo mais ligado à estrutura que o banco de dados terá quando for implementado. Detalhes mais específicos de regras e restrições são detalhados. Deve ser possível visualizar como todas as estruturas do banco de dados funcionarão.

Modelo Físico: detalhes de métodos de armazenamento e acesso aos dados. Reflete exatamente o banco a ser criado e manipulado.



MODELO CONCEITUAL

Descrição do banco de dados de forma independente da sua implementação em um SGBD. Descreve quais dados podem existir no banco, porém não registra como estes dados estão armazenados no SGBD físico.

Possui uma linguagem visual simples, permitindo a compreensão de usuários leigos para entendimento do modelo.

ABSTRAÇÃO DE DADOS: processo onde é necessária a compreensão de um problema do mundo real e traduzi-lo para um modelo de dados. Este processo tem origem no Levantamento de Requisitos, onde o desenvolvedor realiza entrevistas, questionários, reuniões e observações junto aos usuários que estão demandando o software (cliente).

Desta forma é possível compreender como funciona o processo a ser informatizado. Neste momento é preciso criar o minimundo (cenário do cliente), visando o prosseguimento de um foco determinado para o desenho da modelagem de dados do sistema. **Aquilo que não é essencial não deve ser modelado**.

MODELO CONCEITUAL

Imagine a situação abaixo:

É necessária uma modelagem de dados em relação à situação anterior, o que é possível identificar? O processo é um *checkout* de compras em um supermercado.

MODELO CONCEITUAL

Agora é necessário pensar em quais tipos de informações reais precisam ser armazenadas neste cenário. O primeiro conceito que vem à mente é do produto (um ou mais produtos irão compor uma compra). Outro conceito é o do pagamento, pois o operador informará ao Sistema a forma de pagamento. Neste cenário, o cliente e o operador também são relevantes para fins de registro. Desta forma é possível delimitar o minimundo a ser modelado.

MODELO ER

Entidade/ Relacionamento

Este modelo visa representar de forma conceitual o banco de dados, para o entendimento do usuário final, demonstrando os elementos do negócio, como: entidade, atributos e relacionamentos. O esquema utilizado é denominado de Diagrama Entidade Relacionamento (DER).

Entidades: representações de objetos do mundo real, dentro de um banco de dados. Podem representar objetos físicos, organizações, pessoas, lugares, processos.

Para um modelo de dados, o que importa são apenas os objetos que se deseja manter informações. Estes objetos são as entidades do BD. As entidades são representadas por um retângulo e, ao centro, o nome da entidade (ex.: veículo, imóvel, produto, venda, vendedor, pagamento, banco, agência).

MODELO ER

Entidade/ Relacionamento

Atributos: características que definem uma entidade ou um relacionamento. Na prática não são representados graficamente para não sobrecarregar o diagrama. Mas em modelos conceituais, aparecem para entendimento do escopo do modelo.

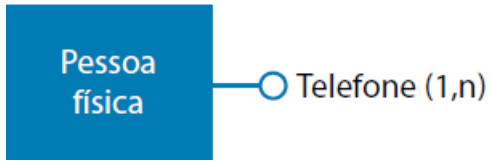
Atributo Identificador: representado por meio do círculo preenchido na extremidade do atributo. Identificam ou compõem a identificação única de uma ocorrência em uma entidade.

Atributo Não Identificador: representado por meio de um círculo vazio na extremidade do atributo. A grande maioria dos atributos de uma entidade serão atributos não identificadores.

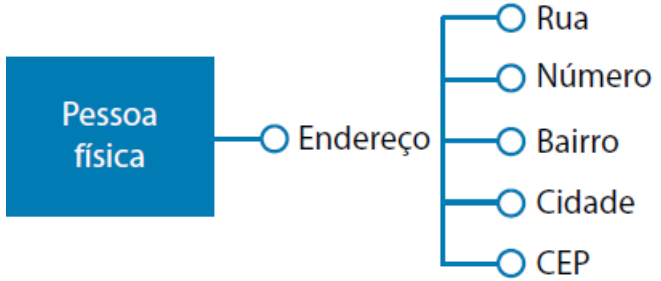
MODELO ER

Entidade/ Relacionamento

Atributos Multivalorados: representado pela sua cardinalidade na extremidade do atributo. O primeiro número representa o número mínimo de ocorrências e, o segundo, o número máximo. São usados para representar dados que podem ter mais de uma ocorrência para o mesmo registro. Ex.: telefone – geralmente uma pessoa possui mais de um telefone para cadastro.



Atributos Compostos: representados por meio de uma oval com vários nós na extremidade do atributo. Um atributo composto é usado quando uma informação contém várias partes, como um endereço. Composto por rua, número, bairro, cidade, estado, CEP, pode ser representado de forma conjunta.



RELACIONAMENTOS

Um relacionamento dentro de um banco é uma associação entre entidades. As entidades vinculadas a um relacionamento também são chamadas de entidades participantes.

O relacionamento deve ser identificado por um verbo, que ajudará na compreensão do relacionamento. Outra característica importante é que um relacionamento sempre opera em ambas as direções.

O relacionamento é representado por um losango, com o verbo que o representa contido dentro do losango.

São representadas duas entidades **Venda** e **Produto**. Entre elas, o relacionamento é representado pelo losango e seu verbo identificador. Próximo de cada entidade envolvida, é apresentada a sua **cardinalidade**.

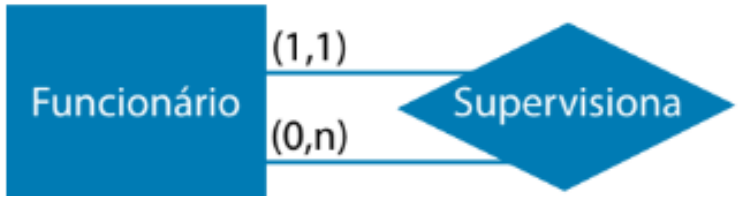
Neste caso, a leitura será: **uma venda contém um ou muitos produtos e um produto pode estar contido em nenhuma ou em muitas vendas**.

OBS.: LEITURA CRUZADA (CARDINALIDADE)

RELACIONAMENTOS

Ao realizar a leitura, a cardinalidade deve ser lida do lado mais próximo da entidade destino. Deve-se entender que existe uma cardinalidade mínima e uma cardinalidade máxima nos relacionamentos. Neste caso, de Venda para Produto, mínimo 1 e máximo n (muitos).

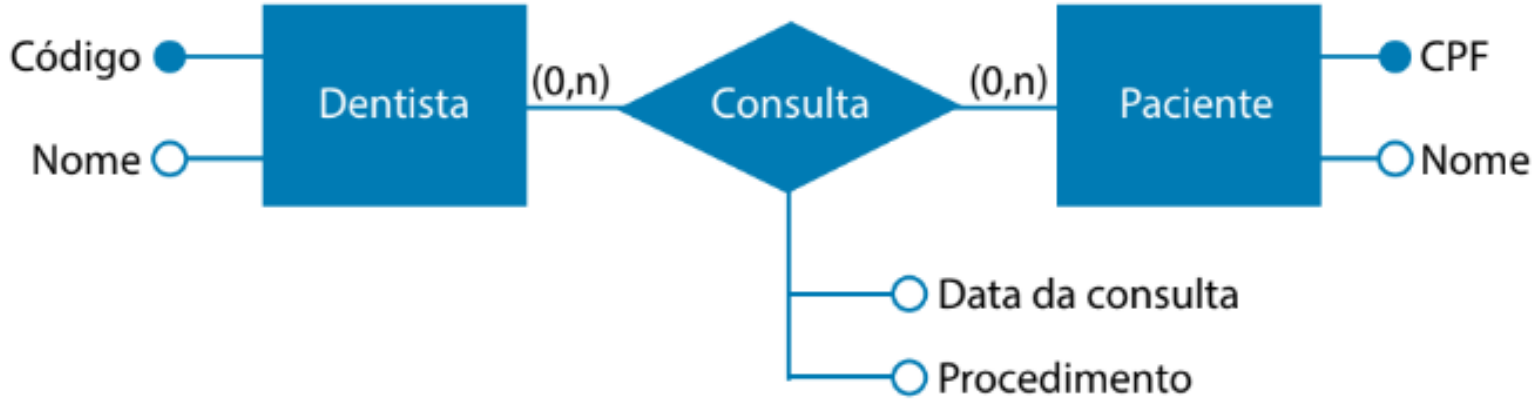
Na realização da modelagem de um banco de dados, é possível esbarrar-se com cenários onde uma entidade se relaciona com ela mesma. Por exemplo, um software de RH, onde são cadastrados os funcionários da empresa e o supervisor de uma equipe também é um funcionário. Nisto temos um auto relacionamento (ou um relacionamento recursivo).



Um funcionário supervisiona zero ou muitos funcionários e um funcionário é supervisionado por um e somente um funcionário.

RELACIONAMENTOS

Existem também casos onde haverão atributos definidos para o relacionamento como um todo (e não apenas nas entidades). Ex.: um consultório médico, onde o atendimento é realizado diariamente e é necessário registrar data, a hora e os procedimentos realizados.

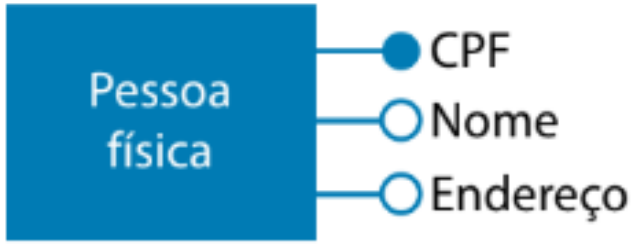


Um dentista pode consultar muitos pacientes. Um paciente pode ser consultado por muitos dentistas. Para cada consulta são registrados a data da consulta e o procedimento realizado.

IDENTIFICADORES DE ENTIDADES

Toda entidade precisa ter um identificador, que pode ser formado por um único atributo ou um conjunto deles. A utilidade do identificador é distinguir cada ocorrência de uma entidade das outras.

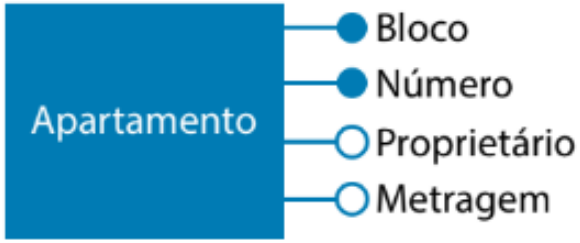
No modelo conceitual, os atributos identificadores são simbolizados pelo círculo preenchido.



Um conceito importante a ser reforçado: todo atributo utilizado como identificador precisa ser único. O atributo **nome** não poderia ter sido usado, pois podem existir centenas de **João da Silva** ou **Maria de Souza**, mas o CPF sim, pois é único para cada pessoa.

IDENTIFICADORES DE ENTIDADES

Exemplo de identificador composto:



Para realizar a identificação de um apartamento em um condomínio, são necessárias as informações de **bloco** e **número**. Um apartamento 701 pode existir em n blocos, assim como um bloco A contém dezenas de apartamentos, portanto um dado complementa o outro.

MODELO LÓGICO

A segunda etapa na construção de um BD é a definição de um modelo lógico. Este é desenvolvido a partir do modelo conceitual e traz uma visão mais voltada para a implementação no SGBD, ao invés de uma visão abstrata de negócio (como o modelo conceitual).

Neste ponto é necessário preocupar-se com nomenclaturas e restrições mais completas de consistência e integridade dos dados.

O modelo lógico também é chamado de **Modelo Relacional**.

Entre o modelo conceitual e o modelo lógico relacional existem algumas conversões de nomenclaturas:

Entidade → **Tabela ou relação**
Atributo → **Coluna**
Atributo identificador → **Chave Primária** *(PK – Primary Key)*
Atributo identificador em outra tabela → **Chave Estrangeira** *(FK – Foreign Key)*

CHAVE PRIMÁRIA, CANDIDATA E ESTRANGEIRA

O que são chamados de **atributos identificadores** no modelo conceitual, no Lógico transferem-se para conceitos de **chave primária** e **chave candidata**.

A chave candidata é composta por um ou mais atributos (colunas) que identificam unicamente uma tabela. Uma tabela pode ter uma ou mais chaves candidatas. Elas são chaves candidatas a chave primária, ou seja, uma delas será escolhida para ser a chave primária.

E como escolher qual será a chave primária? Verificando qual terá uma maior utilização pelos usuários. Por exemplo: cadastro de veículos – tanto a placa, como RENAVAM e Chassi são chaves candidatas, mas qual é a mais provável de um proprietário lembrar? Obviamente a placa do seu carro. **O atributo que identificar mais facilmente uma tabela, é o melhor a ser utilizado**.

CHAVE PRIMÁRIA, CANDIDATA E ESTRANGEIRA

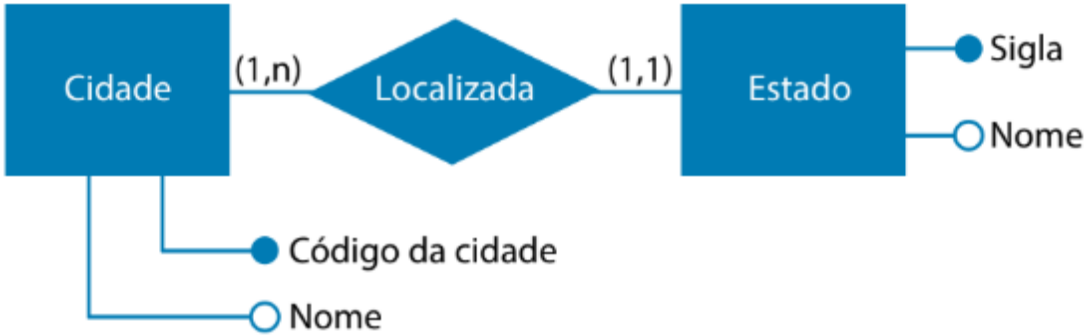
A chave primária *(PK – Primary Key)* é o atributo que identifica unicamente um registro em uma tabela. Também é usada para formalizar os relacionamentos entre as tabelas. Em uma tabela central em um sistema que se relaciona com outras 20 tabelas, a chave primária será transportada para outras 20 tabelas como **chave estrangeira**.

Portanto, dentre algumas características essenciais das chaves primárias estão:

- *ser unívoca, ou seja, ter uma valor único para cada registro;*
- *ser não nula (NOT NULL), nenhum dos atributos que compõem a chave primária podem estar vazios;*
- *ser não redundante, ou seja, não incluir mais atributos que o mínimo necessário para identificação daquela entidade.*

CHAVE PRIMÁRIA, CANDIDATA E ESTRANGEIRA

A chave estrangeira *(FK – Foreign Key)* é o meio de materialização do relacionamento entre as tabelas.



O exemplo acima está representado no modelo conceitual. Já no modelo lógico, isso se transformará em duas tabelas ligadas por uma chave estrangeira:

ESTADO

SIGLA	NOME
RJ	Rio de Janeiro
PR	Paraná
AC	Acre
AM	Amazonas
CE	Ceará

CIDADE

CÓDIGO	NOME	SIGLA_ESTADO
1	Búzios	RJ
2	Fortaleza	CE
3	Rio Branco	AC
4	Manaus	AM
5	Niterói	RJ
6	Pato Branco	PR

CHAVE PRIMÁRIA, CANDIDATA E ESTRANGEIRA

ESTADO	
SIGLA	**NOME**
RJ	Rio de Janeiro
PR	Paraná
AC	Acre
AM	Amazonas
CE	Ceará

CIDADE		
CÓDIGO	**NOME**	**SIGLA_ESTADO**
1	Búzios	RJ
2	Fortaleza	CE
3	Rio Branco	AC
4	Manaus	AM
5	Niterói	RJ
6	Pato Branco	PR

Considerando que um Estado pode conter mais de uma cidade, e que uma cidade pode estar localizada em apenas um Estado, a tabela Cidade recebe o atributo SIGLA_ESTADO, que é a chave primária da tabela Estado. A tabela Cidade, portanto, fica com três colunas: Código, Nome e Sigla_Estado. Sendo assim, a cada registro de uma cidade, deverá ser informado seu código, seu nome e a qual estado ela pertence.

OBS.: só podem ser informadas siglas de estados que estejam cadastrados na tabela Estado. Isso se chama **Integridade Referencial**.

INTEGRIDADE REFERENCIAL

É a propriedade que garante que os dados que compõem uma chave estrangeira estejam sempre íntegros. Significa que nenhum dado em uma tabela destino de uma chave estrangeira deixa de estar contido na tabela origem, e nenhum dado contido na tabela origem será alterado ou excluído enquanto estiver vinculado à tabela destino.

ESTADO	
SIGLA	NOME
RJ	Rio de Janeiro
PR	Paraná
AC	Acre
AM	Amazonas
CE	Ceará

CIDADE		
CÓDIGO	NOME	SIGLA_ESTADO
1	Búzios	RJ
2	Fortaleza	CE
3	Rio Branco	AC
4	Manaus	AM
5	Niterói	RJ
6	Pato Branco	PR
7	Tubarão	SC

No exemplo acima, é simulada a criação de uma nova cidade. Considerando que há uma chave estrangeira entre Cidade e Estado (SIGLA_ESTADO), esta inserção causaria uma violação da integridade referencial, uma vez que a sigla SC não é a sigla de nenhum dos estados contidos na tabela Estado. O SGBD não permitirá a operação e emitirá uma mensagem de erro.

INTEGRIDADE REFERENCIAL

ESTADO	
SIGLA	NOME
RJ	Rio de Janeiro
PR	Paraná
AC	Acre
AM	Amazonas
CE	Ceará

CIDADE		
CÓDIGO	NOME	SIGLA_ESTADO
1	Búzios	RJ
2	Fortaleza	CE
3	Rio Branco	AC
4	Manaus	AM
5	Niterói	RJ
6	Pato Branco	PR

No exemplo acima, será simulada a exclusão do estado AC – Acre. Porém, ao solicitar esta operação ao SGBD, ele verificará que há uma cidade (Rio Branco) vinculada ao estado AC. Este também é um caso de violação de integridade referencial, causando um erro emitido pelo SGBD, não permitindo a operação.

INTEGRIDADE REFERENCIAL

ESTADO	
SIGLA	NOME
RJ	Rio de Janeiro
PR	Paraná
AC	Acre
AM	Amazonas
CE	Ceará

CIDADE		
CÓDIGO	NOME	SIGLA_ESTADO
1	Búzios	RJ
2	Fortaleza	CE
3	Rio Branco	AC
4	Manaus	AM
5	Niterói	RJ
6	Campo Grande	RJ -> MS

No exemplo acima, será simulada a atualização da sigla do estado da cidade de Campo Grande, de RJ para MS. Porém MS não é a sigla de nenhum estado registrado na tabela de estados. Esta atualização infringe a regra de integridade referencial e o banco também retornará uma mensagem de erro.

Resumidamente, **a integridade referencial é um guardião da qualidade dos dados de um banco. As regras são checadas a todo momento que um dado vinculado a um relacionamento é modificado, na tabela origem ou na tabela destino.**

MODELO FÍSICO

O modelo físico de um banco de dados é uma descrição de um banco no nível de abstração visto pelo usuário do SGBD. Neste modelo serão detalhados os componentes da estrutura física do banco, como tabelas, campos, tipos de valores, índices. Momento em que o banco em si é criado, por isso chama-se **modelo físico**.

TIPOS DE DADOS

O tipo de dado – *também chamado de* ***domínio de um atributo***, define o que será permitido informar naquela coluna (atributo). Os tipos mais comuns são: numéricos, alfanuméricos e de datas. Dentro dessas grandes divisões existem subtipos específicos.

Imagine um atributo que armazene a data de nascimento de uma pessoa. Se for um campo livre, alguns usuários podem informar no formato adequado: 05/08/1996. Outros poderão informar por extenso: cinco de Agosto de 1996. Outros ainda poderão usar 05 ago. 1996. Como seria possível organizar os dados já que eles não seguem um padrão? Por isso há a necessidade de criar restrições de tipos de dados para o atributo.

TIPOS DE DADOS

Principais tipos de dados utilizados nas tabelas:

TIPO DE DADO	APLICAÇÃO
TINYINT	Um inteiro muito pequeno. De -128 a 127
SMALLINT	Um inteiro pequeno. De -32768 a 32767
INTEGER ou INT	Um inteiro de tamanho normal. De -2147483648 a 2147483647
DECIMAL(N, P)	Um número decimal, onde N é número de dígitos e P o número de casas após a vírgula.
DATE	Suporta uma data. No formato YYYY-MM-DD
DATETIME	Uma combinação de data e hora. No formato: YYYY-MM-DD HH:MM:SS
TIME	Suporta uma hora. No formato HH:MM:SS
CHAR(T)	Cadeia de caracteres. Em T deve ser determinado o tamanho da coluna.
VARCHAR(T)	Cadeia de caracteres. Em T deve ser determinado o tamanho da coluna.

TIPOS DE DADOS

CHAR(T)	Cadeia de caracteres. Em T deve ser determinado o tamanho da coluna.
VARCHAR(T)	Cadeia de caracteres. Em T deve ser determinado o tamanho da coluna.

A diferença dos tipos CHAR e VARCHAR está na relação à alocação de espaço. Atributos do tipo CHAR já tem espaço reservado no banco para armazenar todo o tamanho definido para o campo. No caso de um atributo VARCHAR, o SGBD fará a alocação dinamicamente, de acordo com o tamanho do dado armazenado.

Mas por que não pode-se utilizar o VARCHAR sempre?

Cada tipo de dado tem sua aplicação de uso. Para atributos onde o tamanho de todos os registros será o mesmo, é indicado usar o CHAR. Assim, o SGBD não precisará realizar um processamento na inserção para validar o tamanho informado no campo e realizar a alocação de espaço, por exemplo: CPF – sempre terá 11 dígitos, portanto o uso indicado é o CHAR(11).

No caso de um atributo que guarde o nome de uma rua, não é possível determinar que todos os registros terão o mesmo tamanho. A indicação, neste caso, é o uso do tipo VARCHAR com tamanho adequado, ex.: VARCHAR(150), um tamanho mais do que suficiente para armazenar o nome de uma rua.

LINGUAGEM SQL

A linguagem SQL foi criada nos laboratórios da IBM nos anos 1970 e é um padrão mundial para bancos de dados relacionais desde a década de 1980.

SQL – *do inglês Structured Query Language* *(Linguagem de Consulta Estruturada)* é uma linguagem usada para a criação, manipulação (inserção, atualização, exclusão) e seleção de dados.

Características:

- é independente do fabricante (ou seja, para os mais diversos fabricantes de banco de dados, a linguagem é a mesma: SQL);
- é CASE INSENSITIVE, ou seja, não faz diferenciação entre letras maiúsculas e minúsculas;
- utiliza um inglês de alto nível de fácil entendimento;
- uma linguagem em que se determina o resultado esperado, e não quais os caminhos que se deve percorrer para chegar até ele. Pouca programação resolve problemas complexos.

DIVISÕES DA LINGUAGEM SQL

A linguagem SQL é dividida em 5 conjuntos de comandos:

a) DDL (Data Definition Language): criação e alteração da estrutura dos bancos de dados;

b) DML (Data Manipulation Language): criação, alteração e manipulação dos dados contidos em um banco de dados;

c) DQL (Data Query Language): consulta (recuperação) de dados inseridos no banco;

d) DCT (Data Control Language): controle do acesso ao banco e aos dados;

e) DTL (Data Transaction Language): gerenciamento das transações ocorridas em um banco de dados.

DDL:

DATA DEFINITION LANGUAGE

A linguagem DDL é a área do SQL responsável pela criação e manutenção das estruturas dos bancos de dados. Com ela pode-se criar e alterar tabelas e seus atributos, criar relacionamentos entre os atributos, definir restrições, criar triggers (gatilhos), procedures (procedimentos executados de uma vez) e views (visão conjunta resultado de uma consulta armazenada sobre os dados).

COMANDOS:

CREATE DATABASE: permite a criação de um novo banco de dados no servidor. Geralmente é criado um banco de dados para cada sistema (assunto) que será trabalhado. A sintaxe funciona da seguinte forma:

```
CREATE DATABASE nomebasededados
```

DDL:

DATA DEFINITION LANGUAGE

A criação de um banco não o seleciona para uso, para isso é necessário especificar qual base deseja utilizar, através do comando USE.

```
USE nomebadededados
```

Após a criação e a seleção do banco de dados, é possível criar as demais estruturas.

CREATE TABLE: comando mais usado da DDL, pois através dele as tabelas são criadas com suas colunas e toda a sua estrutura.

```
CREATE TABLE nomedatabela (
    nomedoatributo1 tipodedado obrigatoriedade
    nomedoatributo2 tipodedado obrigatoriedade
    nomedoatributo3 tipodedado obrigatoriedade,
restrições de integridade);
```

DDL:

DATA DEFINITION LANGUAGE

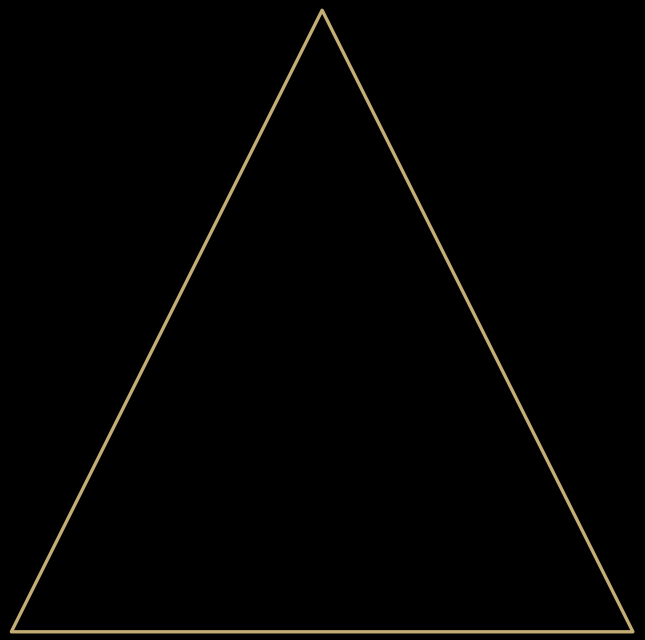
Ao criar uma tabela, é preciso definir, para cada atributo, seu nome, tipo, tamanho (se for o caso), sua obrigatoriedade e as restrições de integridade.

Exemplo prático:

```
CREATE TABLE produto (
   CODPRODUTO INTEGER NOT NULL PRIMARY KEY,
   DESCPRODUTO VARCHAR(100) NOT NULL,
   PRECOVENDA DECIMAL(10,2) NOT NULL,
   OBSERVAÇÃO VARCHAR(500) NULL
   );
```

DDL:

DATA DEFINITION LANGUAGE



ALTER TABLE: comando com o objetivo de alterar a estrutura de uma tabela. Pode ser a adição de uma nova coluna, a alteração de uma coluna já existente, ou até mesmo a retirada de uma coluna. Também podem ser alteradas as restrições de integridade da tabela, como a chave primária e a criação de chaves estrangeiras.

```
ALTER TABLE nomedatabela
```

Seguido da modificação que se pretende fazer, como adicionar uma nova coluna em uma tabela já existente:

```
ALTER TABLE nomedatabela ADD COLUMN nomecoluna
tipo tamanho obrigatoriedade
```

Se a necessidade for a adição de uma chave estrangeira, além da sintaxe básica, deve-se adicionar a criação da foreign key:

```
ALTER TABLE nomedatabela ADD CONSTRAINT
FK_nomecolunadestino FOREIGN KEY (nomecolunadestino)
REFERENCES nometabelaorigem (colunaorigem)
```

DDL:

DATA DEFINITION LANGUAGE

Exemplo prático:

```
ALTER TABLE CURSO1 ADD FOREIGN KEY (CURSOPAI)
REFERENCES CURSOS(CDCURSO)
```

Principais comandos para modificações em tabelas:

ALTERAÇÃO	SINTAXE
Alterar a definição de uma coluna	ALTER TABLE <nometabela> MODIFY COLUMN <nomecoluna> <tipo tamanho> <obrigatoriedade>
Modificar o nome de uma coluna	ALTER TABLE <nometabela> RENAME COLUMN <nomeantigo> TO <novonome>
Modificar o nome da tabela	ALTER TABLE <nometabela> RENAME TO <novonometabela>;
Eliminar uma coluna	ALTER TABLE <nometabela> DROP COLUMN <nomecoluna>

DDL:

DATA DEFINITION LANGUAGE

DROP TABLE: comando com o objetivo de excluir uma tabela do banco de dados. Importante lembrar que ao excluir uma tabela do BD, o usuário estará excluindo sua estrutura e os dados contidos nela. Mesmo que sem uso, as tabelas geralmente não são excluídas de um banco de dados, pois podem conter informações estratégicas das empresas. Portanto o comando DROP TABLE deve ser utilizado apenas em casos específicos. **LEMBRAR DA INTEGRIDADE REFERENCIAL**.

```
DROP TABLE nomedatabela
```

Exemplo prático:

```
DROP TABLE ALUNO
```

DQL:

DATA QUERY LANGUAGE

A linguagem DQL é responsável pela recuperação dos dados armazenados em um banco de dados. É centralizada numa única estrutura: o comando SELECT, que possui uma variedade grande de composições.

Sintaxe Básica:

```
SELECT coluna
FROM nometabela
WHERE condições(opcional)
```

Em um mesmo comando SELECT, pode-se obter dados de diversas colunas e de diversas tabelas. As condições (WHERE) são filtros aplicados nos dados selecionados a fim de apresentar apenas os dados que importam. Ex.: SELECT contendo apenas uma tabela, onde pode-se apresentar TODOS os dados da tabela aluno.

```
SELECT * FROM ALUNO
```

O asterisco (*) simboliza que todas as colunas das tabelas envolvidas na cláusula FROM sejam apresentadas. A ausência da cláusula WHERE significa que todos os registros devem ser apresentados, sem nenhum tipo de restrição.

DQL:

DATA QUERY LANGUAGE

Ex.: caso seja necessário apresentar apenas a matrícula e o nome de todos os alunos, o comando poderá ser escrito da seguinte forma:

```
SELECT idMatricula, nome FROM aluno
```

O nome das colunas (atributos) devem ser definidos exatamente como foram criados e deve-se usar a vírgula como separador.

OPERADORES DE COMPARAÇÃO: são utilizados em conjunto com a cláusula WHERE.

OPERADOR	FUNCIONAMENTO	EXEMPLO
=	Igual a	nota = 10
>	Maior que	preco > 50
>=	Maior ou igual a que	nota >= 7
<	Menor que	nota < 7
<=	Menor ou igual a que	preco <= 2
<>	Diferente de	bairro <> 'Centro'

DQL:

DATA QUERY LANGUAGE

SQL JOINS

Em SQL, um JOIN é uma operação que combina registros de duas ou mais tabelas com base em uma determinada condição a ser relacionada, possibilitando a obtenção de dados de várias tabelas em uma única consulta.

Características:

1. **Combinação de Dados:** o JOIN é usado para combinar dados de diferentes tabelas com base em uma relação definida entre elas.

2. **Chaves de Junção:** são colunas comuns entre as tabelas que são usadas para combinar os registros. As condições de junção podem ser estabelecidas usando operadores de comparação, como **=, <, >, etc.**

3. **Utilidade:** são amplamente utilizados em consultas SQL para extração de dados relacionados de várias tabelas em um único resultado. São essenciais em bancos de dados relacionais para consultar dados de forma eficiente e precisa.

DQL:

DATA QUERY LANGUAGE

TIPOS DE JOIN

- **INNER JOIN:** retorna apenas os registros que têm correspondências em ambas as tabelas. Combina registros das duas tabelas com base em uma condição de junção especificada.

```
SELECT *
FROM tabela1
INNER JOIN tabela2 ON tabela1.id = tabela2.id
```

- **LEFT (OUTER) JOIN (inclusive):** retorna todos os registros da tabela à esquerda e os registros correspondentes da tabela à direita. Se não houver correspondência, os valores NULL são retornados para as colunas da tabela à direita.

```
SELECT *
FROM tabela1
LEFT JOIN tabela2 ON tabela1.id = tabela2.id
```

DQL:

DATA QUERY LANGUAGE

TIPOS DE JOIN

- **LEFT (OUTER) JOIN (exclusive):** menos comum, retorna apenas os registros da tabela à esquerda que não possuem correspondências na tabela à direita.

```
SELECT *
FROM tabela1
LEFT JOIN tabela2 ON tabela1.id = tabela2.id
WHERE tabela2.id IS NULL
```

DQL:

DATA QUERY LANGUAGE

TIPOS DE JOIN

- **RIGHT (OUTER) JOIN (inclusive):** retorna todos os registros da tabela à direita e os registros correspondentes da tabela à esquerda.

```
SELECT *
FROM tabela1
RIGHT JOIN tabela2 ON tabela1.id = tabela2.id
```

- **RIGHT (OUTER) JOIN (exclusive):** assim como o LEFT OUTER JOIN exclusivo, este é menos comum. Retorna apenas os registros da tabela à direita que não têm correspondências na tabela à esquerda.

```
SELECT *
FROM tabela1
RIGHT JOIN tabela2 ON tabela1.id = tabela2.id
WHERE tabela1.id IS NULL
```

DQL:

DATA QUERY LANGUAGE

TIPOS DE JOIN

- **FULL OUTER JOIN (inclusive):** retorna todos os registros quando há uma correspondência em uma das tabelas. Se não houver correspondência, os valores NULL são retornados para as colunas da outra tabela.

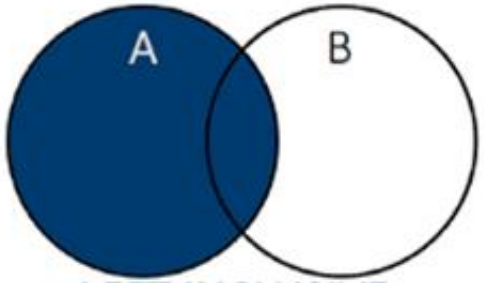
```
SELECT *
FROM tabela1
FULL OUTER JOIN tabela2 ON tabela1.id = tabela2.id
```

- **FULL OUTER JOIN (exclusive):** menos comum, retorna apenas os registros que não têm correspondências em ambas as tabelas.

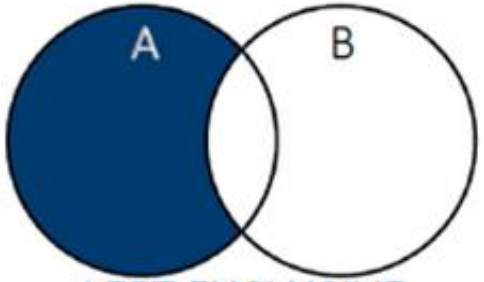
```
SELECT *
FROM tabela1
FULL OUTER JOIN tabela2 ON tabela1.id = tabela2.id
WHERE tabela1.id IS NULL OR tabela2.id IS NULL
```

DQL:

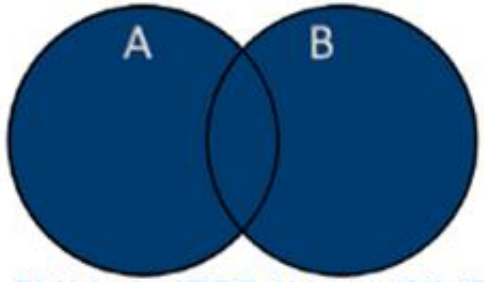
DATA QUERY LANGUAGE



LEFT INCLUSIVE



LEFT EXCLUSIVE



FULL OUTER INCLUSIVE

SQL JOINS

LEFT INCLUSIVE

```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
```

RIGHT INCLUSIVE

```
SELECT [Select List]
FROM TableA A
RIGHT OUTER JOIN TableB B
ON A.Key= B.Key
```

LEFT EXCLUSIVE

```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
WHERE B.Key IS NULL
```

RIGHT EXCLUSIVE

```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
WHERE A.Key IS NULL
```

FULL OUTER INCLUSIVE

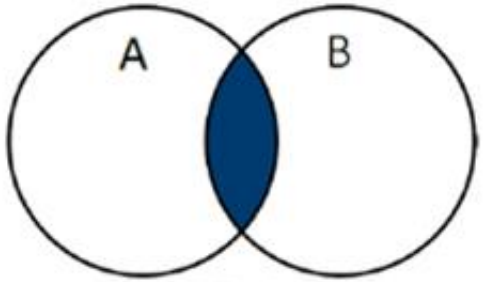
```
SELECT [Select List]
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

FULL OUTER EXCLUSIVE

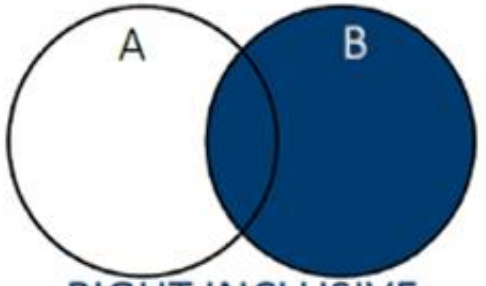
```
SELECT [Select List]
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL
```

INNER JOIN

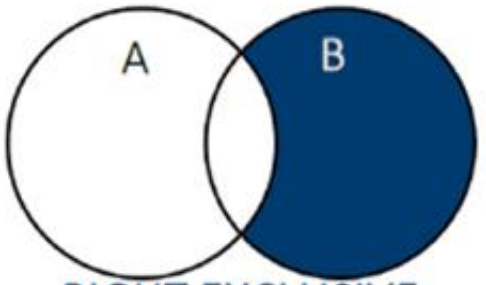
```
SELECT [Select List]
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



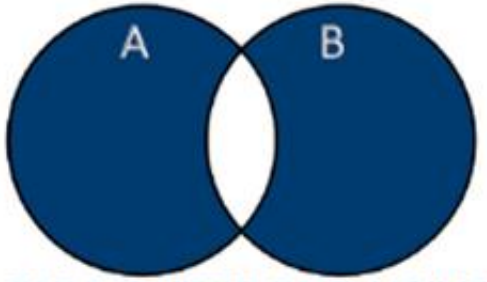
INNER JOIN



RIGHT INCLUSIVE



RIGHT EXCLUSIVE

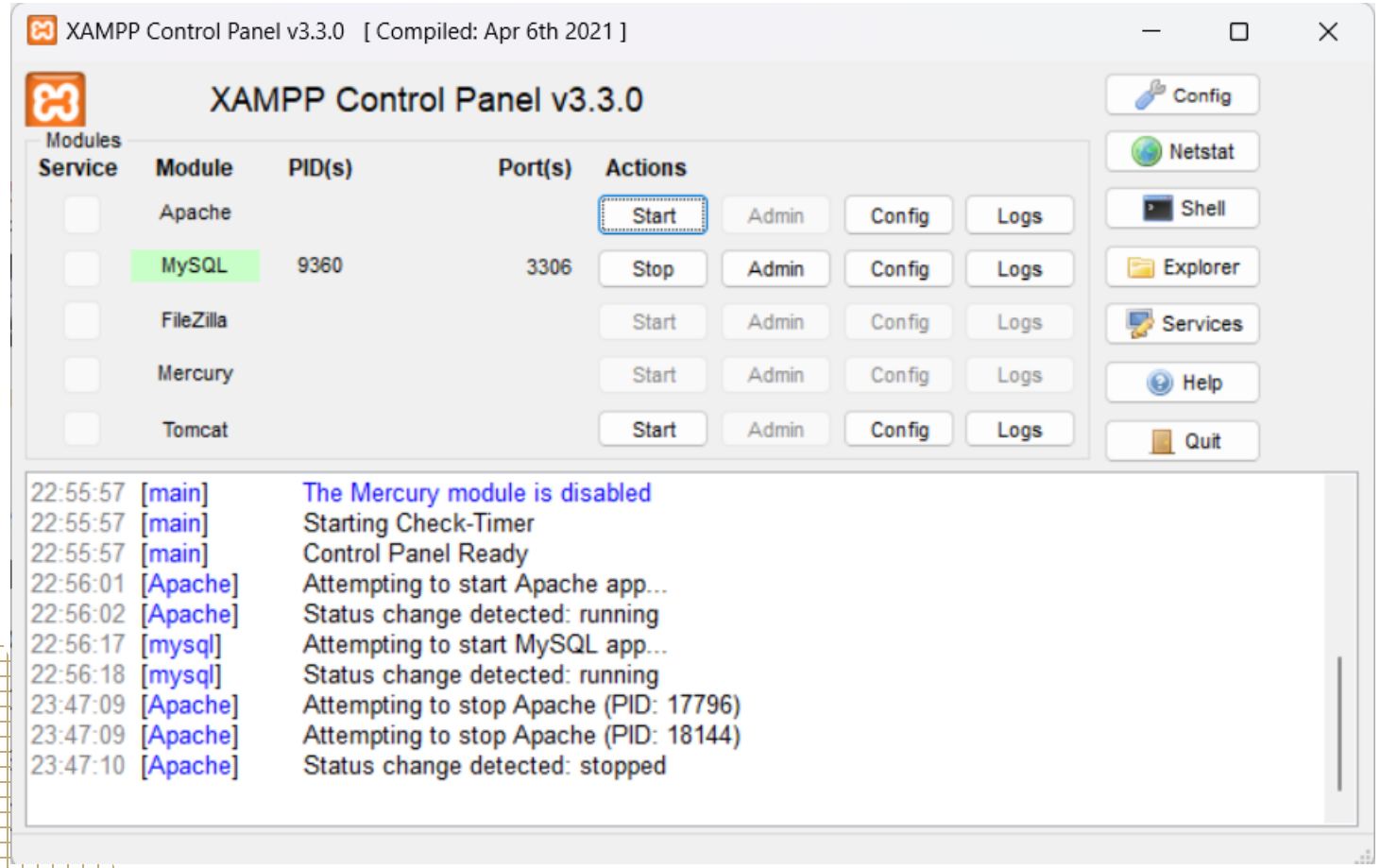


FULL OUTER EXCLUSIVE

NA PRÁTICA...

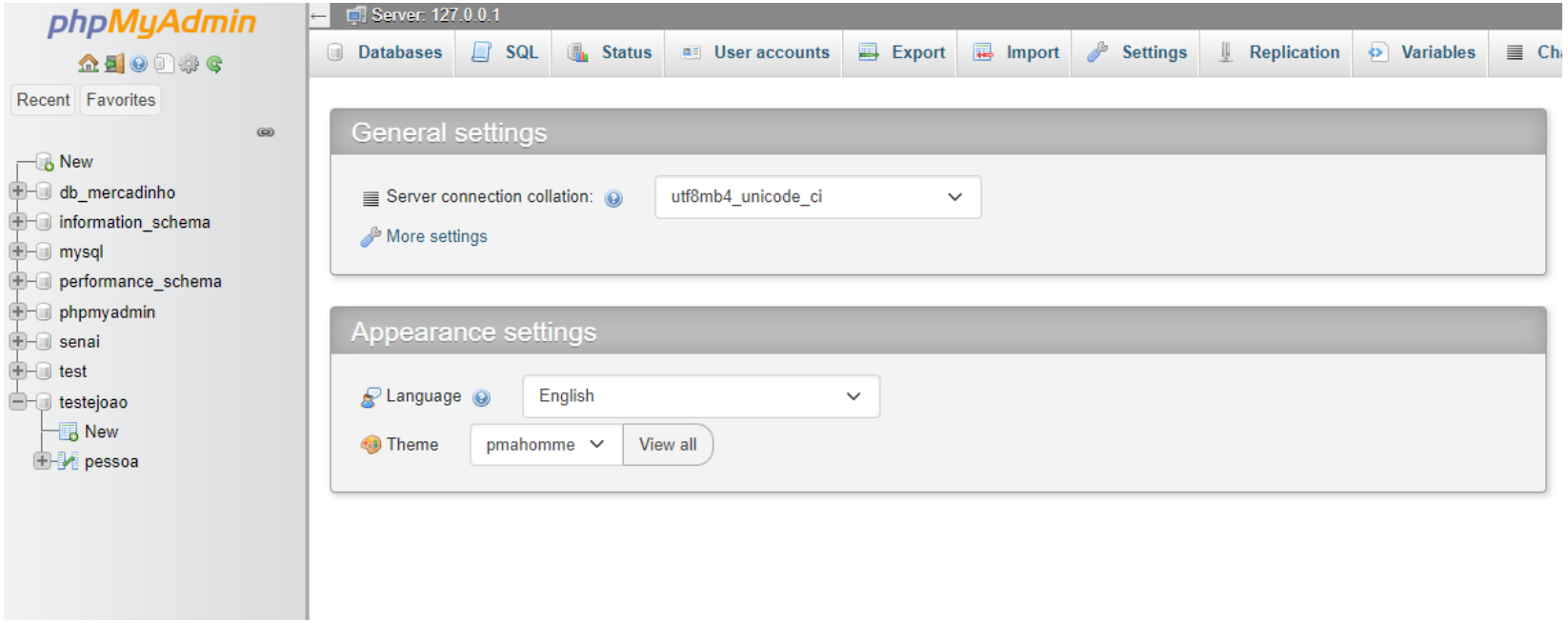
Para as atividades em aula, será utilizado o XAMPP, que é um pacote com os principais serviços de código aberto mundiais, incluindo o banco de dados MySQL (que será utilizado).

Faça o download aqui e instale o software. Após instalação, inicie os serviços Apache e MySQL clicando no botão Start.



NA PRÁTICA...

Ao clicar no botão **Admin**, o usuário será redirecionado ao PHPMyAdmin, que é uma ferramenta de suporte ao desenvolvimento e acesso facilitado a bancos de dados de sistemas.



Na aba **SQL** é possível realizar, via script SQL, a criação do banco, tabelas, atributos e a manipulação de dados.

NA PRÁTICA...

Para das continuidade às aulas e conectar o seu programa em **Python** ao banco de dados, é necessário seguir os pré-requisitos abaixo:

- possuir o Python (3.10 ou superior) instalado;
- ter o PyCharm instalado e configurado com a v3.10 ou v3.11 do Python;
- ter instalada a biblioteca MySQL Connector (caso não possua, executar o comando abaixo no terminal)

pip install mysql-conector-python

NA PRÁTICA...

Estrutura de Código Python + SQL

```
import mysql.connector #importação
```

a linha acima realiza a importação do módulo MySQL Connector

```
conexao = mysql.connector.connect(host="localhost", user="root",
password="", database="testejoao")
```

*a linha acima possui uma variável chamada **SQL** que recebe o comando de conexão ao banco de dados especificado no parâmetro. Banco de dados **testejoao**, usuário padrão (**root**) e hospedado do host local. Não possui senha para acesso.*

```
if conexao.is_connected():
        print("Conexão realizada com sucesso")
```

*este if realiza um teste de conexão com o comando **is_connected( )**. Caso seja constatado que a conexão foi realizada com sucesso, emite mensagem de OK.*

```
else:
        print("Falha na conexão")
```

NA PRÁTICA...

Comando cursor( )

O cursor é um objeto usado para realizar a conexão visando executar comandos em SQL. Ele funciona como uma ponte entre a conexão do banco de dados e a própria consulta SQL descrita no código.

```
cursor = conexao.cursor()
```

*lembrando que **conexao** é a variável que recebe os parâmetros para conexão ao banco de dados, definido anteriormente.*

Comandos execute( ) e commit ( )

O comando execute() é de extrema importância para o código, ele tem a função de pegar a consulta em SQL e executá-la no servidor.

O comando commit() garantirá que os comandos especificados na query SQL sejam implementados no banco de dados.

NA PRÁTICA...

Declare uma variável para receber a query de inserção de valores para o seu banco.

```
sql = "INSERT INTO pessoa (cpf, nome, endereco, telefone, sexo) values
(‘19876543210', 'Joao', 'Av. Passos', ‘21999680001', 'M’)”
```

Em seguida, programe os comandos execute() e commit().

```
cursor.execute(sql)
conexao.commit()
```

Agora você deve encerrar sua conexão com o comando close(). É recomendado que sempre se feche uma conexão quando não for mais utilizá-la, devido a performance e possíveis problemas que podem ser causados caso sua conexão fique aberta a longo prazo. Quanto menor o tempo de abertura de uma conexão, melhor será sua performance.

Outro ponto é a realização de diversas requisições ao mesmo tempo. Alguns SGBDs podem limitar a quantidade de conexões ao mesmo tempo (por exemplo, 100 conexões), então pode-se obter um grande problema em um sistema muito utilizado por várias pessoas dentro de uma empresa ao mesmo tempo, por exemplo.

NA PRÁTICA...

Após a execução do seu código, você poderá ver os valores inseridos refletidos no seu banco de dados.

```python
import mysql.connector

conexao = mysql.connector.connect(host="localhost", user="root", password="", database="testejoao")

if conexao.is_connected():
    print("Conexão realizada com sucesso")

else:
    print("Falha na conexão")

cursor = conexao.cursor()

sql = "INSERT INTO pessoa (cpf, nome, endereco, telefone, sexo) values ('19876543210', 'Joao', 'Av. Passos', " \
      "'21999680001', 'M')"

cursor.execute(sql)
conexao.commit()

print(cursor.rowcount, "registro inserido")

conexao.close()
```

NA PRÁTICA...

Após a execução do seu código, você poderá ver os valores inseridos refletidos no seu banco de dados.