



Universidade Federal do Ceará
Centro de Ciências/Departamento de Computação
Código da Disciplina: CK0084
Professor: Ismayle de Sousa Santos

Sistemas de Informações e Banco de Dados

Manipulação e uso de dados e Introdução ao SQL



IsmayleSantos



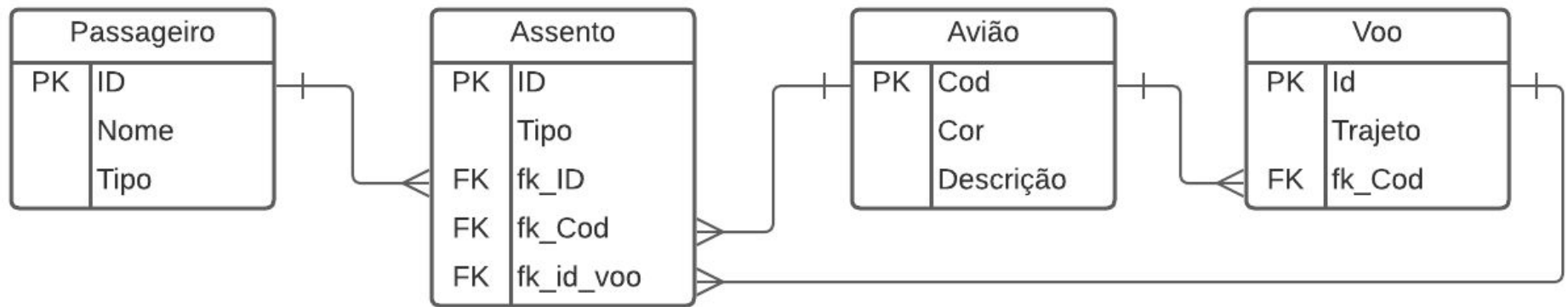
ismayle@ufc.br



@IsmayleSantos

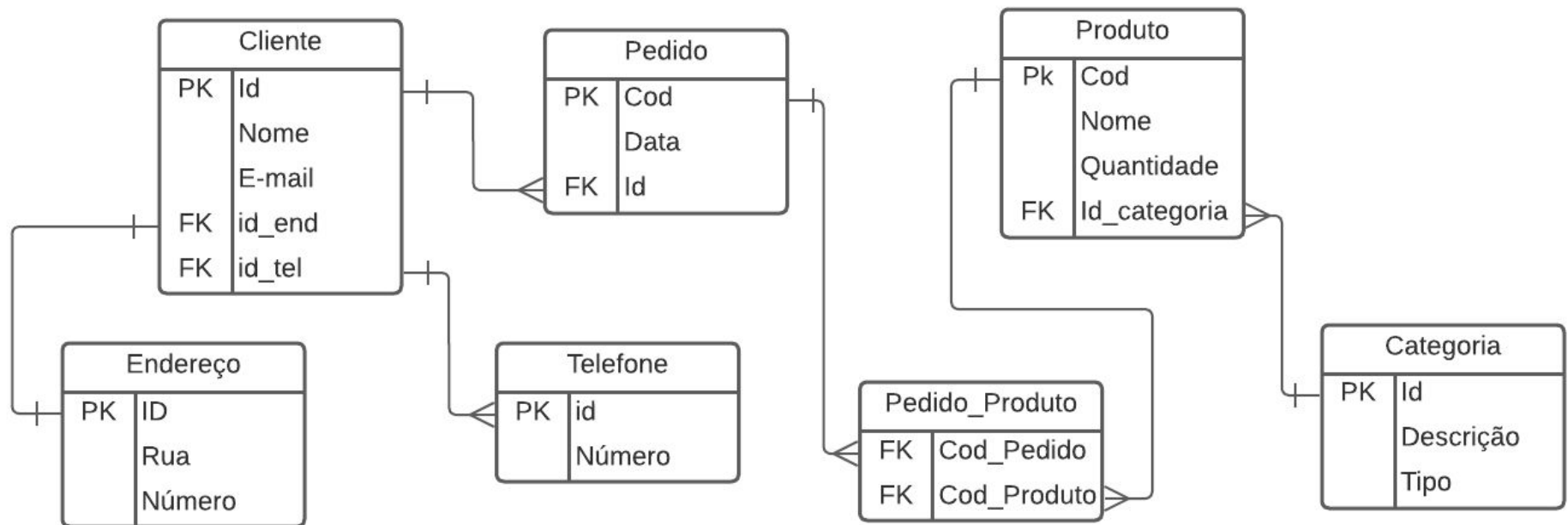
Resposta TP-2

- Resposta para atividade 1



Resposta TP-2

- Resposta para atividade 2



Relembrando alguns conceitos ...

- SGBD
 - Transação
 - É uma sequência de operações executadas como uma única unidade lógica de trabalho
 - E.g.: Transferência de dinheiro entre duas contas correntes
 - Backup x Rollback
 - Abstração de dados
 - Modelo Conceitual
 - Modelo Lógico
 - Modelo Físico
-

Modelo Físico (SGBD)

- No modelo físico fazemos a modelagem física do modelo de banco de dados
 - É feito por meio de alguma linguagem
 - PostgreSQL, MySQL, dentre outros
 - Deve-se criar o modelo físico sempre com base no modelo lógico
-

Modelo Físico (SGBD)

```
CREATE TABLE Usuario (  
    ID varchar (4) NOT NULL,  
    matricula varchar (11) UNIQUE,  
    nome varchar (22),  
    email varchar (18),  
    endereco varchar (20) ,  
    telefone varchar (8),  
    PRIMARY KEY (MATRICULA)  
);
```

Banco Manipulação e Uso de Dados

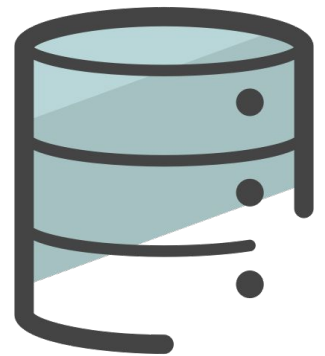
- Os níveis de abstração (níveis físico, conceitual e de visão ou externa) não se aplicam somente à definição ou estrutura de dados, mas também à sua manipulação
 - A manipulação de dados significa:
 - Busca da informação armazenada no BD
 - Inserção de novas informações nos BD
 - Eliminação de informações no BD
 - Modificação de dados armazenados no BD
 - No nível físico é que definimos algoritmos que permitem um acesso eficiente aos dados
-

Banco Manipulação e Uso de Dados

- O principal objetivo da manipulação de dados é alterar o relacionamento (lógico ou físico) que um item de dados possui com outro, não os dados em si
 - As operações comuns usadas para manipular dados incluem:
 - Filtragem de linhas e colunas, agregação, junção e concatenação, manipulação de cadeias, classificação, regressão e fórmulas matemáticas
-

Linguagens de Banco de Dados

Existem vários tipos de linguagens para atender as diferentes abstrações



Linguagem de Banco de Dados

- A linguagem do SGBD é dividida em subconjuntos de acordo com as operações que queremos efetuar sobre um banco de dados, tais como:
 - DML - Linguagem de Manipulação de Dados
 - DDL - Linguagem de Definição de Dados
 - SDL – Linguagem de Definição de Esquemas
 - VDL – Linguagem de Definição de Visualização
 - DCL - Linguagem de Controle de Dados
 - DTL - Linguagem de Transação de Dados
 - DQL - Linguagem de Consulta de Dados
-

Linguagens no SGBDs atuais

- Nos SGBDs atuais, esses tipos de linguagens normalmente não são considerados linguagens distintas
 - Um exemplo típico de linguagem de banco de dados abrangente é a linguagem de banco de dados relacional **SQL**, que representa uma combinação de DDL, VDL e DML, bem como as instruções para especificação de restrição, evolução de esquema e outros recursos
-

DML - Data Manipulation Language

- A DML faz a **manipulação dos dados**
 - Através de uma DML é possível recuperar e inserir as informações, alterar ou remover dados existentes no banco de dados
 - A DML é usada depois que o esquema é compilado e enxertado dados no banco de dados
 - Exemplo de comandos responsáveis pela manipulação dos dados: **SELECT**, **DELETE**, **UPDATE** e **INSERT**
-

DDL - Data Definition Language

- A DDL define o banco de dados
 - O resultado de comandos de uma DDL é um conjunto de tabelas que são armazenadas em um arquivo especial chamado dicionário (ou diretório) de dados
 - Exemplos de comandos para definição das tabelas:
 - **CREATE** - usado para criar objetos
 - **ALTER** - usado para alterar objetos
 - **DROP** - remove objetos
 - **TRUNCATE** - remove rapidamente todas as linhas de um conjunto de tabelas
-

SDL - Storage Definition Language

- Utiliza-se a linguagem SDL para a especificação do esquema/nível interno
 - Em um SGBD em que a separação entre os níveis conceitual e interno são bem claras, é utilizado uma SDL a para a especificação do esquema interno
 - A especificação do esquema conceitual fica por conta da DDL
-

VDL - Vision Definition Language

- Sistemas de Banco de Dados que utilizam a arquitetura três esquemas necessitam de uma linguagem para a definição de visões, a VDL
 - Nos SGBDs relacionais, a SQL é usada pela VDL para definir visões do usuário ou da aplicação como resultados de consultas predefinidas
-

DCL - Data Control Language

- A DCL controla os aspectos de **autorização de dados** e **licenças de usuários** para controlar quem tem acesso para ver ou manipular dados
 - Para usar comandos que ajudam na segurança do banco de dados deve-se usar:
 - **GRANT** - autoriza ao usuário executar ou setar operações
 - **REVOKE** - remove ou restringe a capacidade de um usuário de executar operações
-

DTL - Data Transaction Language

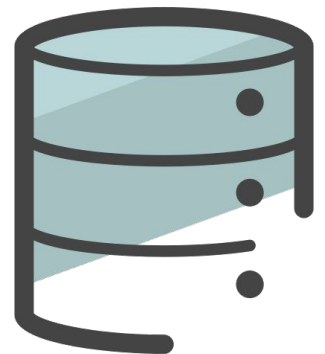
- Os comandos DTL são responsáveis por gerenciar diferentes transações ocorridas dentro de um banco de dados
 - Para usar comandos DTL basta usar
 - **BEGIN WORK** - (ou BEGIN TRANSACTION) - pode ser usado para marcar o começo de uma transação
 - **COMMIT** - finaliza uma transação dentro de um sistema de gerenciamento de banco de dados
 - **ROLLBACK** - faz com que as mudanças nos dados existentes desde o último COMMIT ou ROLLBACK sejam descartadas
-

DQL - Data Query Language

- É a linguagem de **consulta de dados**
 - A DQL possui apenas um comando, o SELECT
 - O **SELECT** é o principal comando usado em SQL para realizar consultas a dados pertencentes a uma tabela
 - O comando SELECT permite ao usuário especificar uma consulta ("query") como uma descrição do resultado desejado
 - Esse comando é composto de várias cláusulas e opções, possibilitando elaborar consultas das mais simples às mais elaboradas
-

Linguagem SQL

Agora iremos estudar os principais recursos do padrão SQL para SGBDs relacionais



O que é SQL?

- SQL é uma linguagem de banco de dados abrangente com instruções para definição de dados, consultas e atualizações
 - SQL - **Structured Query Language**, ou “Linguagem de Consulta Estruturada”, em português
 - SQL é uma **DDL** e uma **DML**
 - É uma linguagem de programação para lidar com banco de dados relacional (baseado em tabelas)
 - Foi criado para que vários desenvolvedores pudessem acessar e modificar dados de uma empresa simultaneamente, de maneira descomplicada e unificada
-

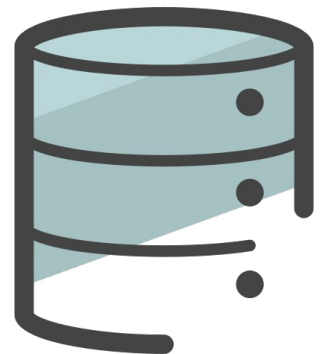
História do SQL

- A SQL foi criada e implementada na IBM Research como a interface para um sistema de banco de dados relacional experimental, chamado SYSTEM R. A SQL
 - Embora o SQL tenha sido originalmente criado pela IBM, rapidamente surgiram vários "dialetos" desenvolvidos por outros programadores
 - Embora a linguagem tenha sido padronizada pela ANSI e ISO, possui muitas variações e extensões produzidos pelos diferentes fabricantes
 - Hoje a SQL é a linguagem padrão para SGBDs relacionais comerciais
-

Porque usar SQL?

- A programação SQL pode ser usada para analisar ou executar tarefas em tabelas, principalmente através dos seguintes comandos: inserir (**insert**), pesquisar (**search**), atualizar (**update**) e excluir (**delete**)
 - A também linguagem SQL oferece uma interface de linguagem declarativa de nível mais alto, assim o usuário apenas especifica qual deve ser o resultado, deixando a otimização real e as decisões sobre como executar a consulta para o SGBD
-

Instâncias, Esquema e Catálogo em SQL



Instâncias e Esquemas

- Os bancos de dados mudam à medida que informações são inseridas ou apagadas
 - A coleção de informações armazenadas é chamada de **instâncias** do banco de dados (mudam com frequência)
 - O projeto geral do banco de dados é chamado **esquema do banco de dados** (não mudam com frequência)
 - É a descrição do banco de dados
-

SQL Schemas

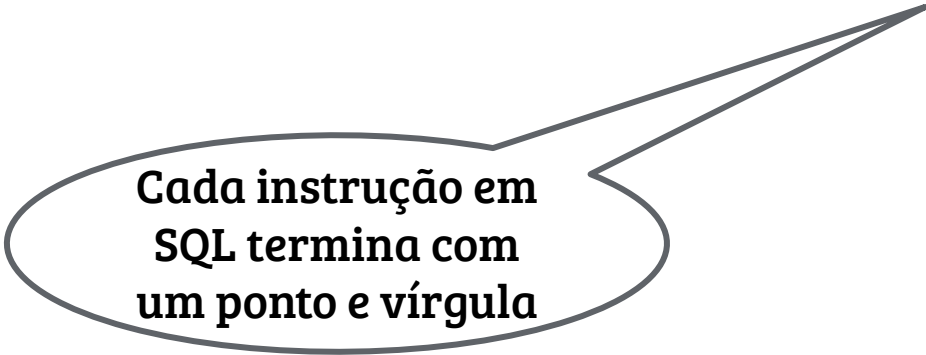
- São coleções de objetos dentro de um determinado banco de dados, que organizam vários aspectos e são importantes para segmentação da segurança, facilitando a administração dos objetos e dos dados
 - Um esquema SQL **visa agrupar tabelas** e outras construções que pertencem à mesma aplicação de banco de dados
-

SQL Schemas

- Um esquema SQL é identificado por um **nome**, e inclui um **identificador de autorização** para indicar o usuário ou conta proprietário do esquema, bem como **descritores** para cada elemento
 - Esses elementos incluem tabelas, restrições, views, domínios e outras construções (como concessões - *grants* - de autorização) que descrevem o esquema que é criado por meio da instrução CREATE SCHEMA
-

SQL Schemas

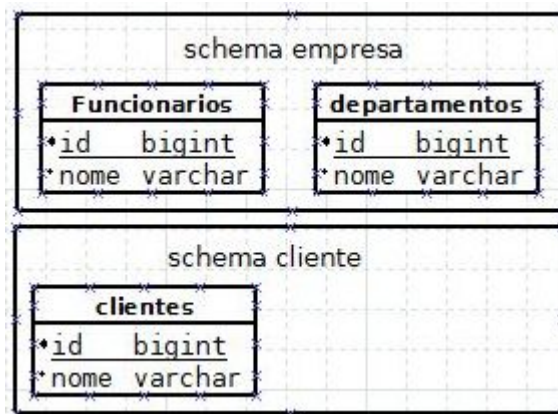
- Um esquema também pode receber só um nome e autorização e os elementos podem ser definidos mais tarde
- Vamos criar um esquema chamado EMPRESA pertencente ao usuário com identificador de autorização 'Jsilva'!
 - **CREATE SCHEMA EMPRESA AUTHORIZATION 'Jsilva';**



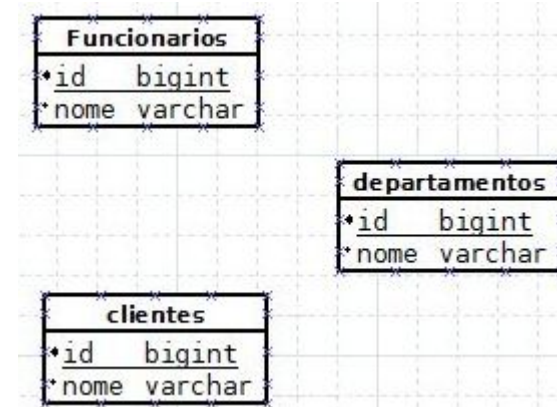
Cada instrução em SQL termina com um ponto e vírgula

SQL Schemas

- Os esquemas fornecem a oportunidade para simplificar a administração de segurança, backup/restauração e gestão de banco de dados, permitindo que os objetos de banco de dados, ou entidades, a ser logicamente agrupadas



Com Schema, o Cliente só “enxerga” os dados relativos a ele



Sem Schema, o Cliente teria acesso às tabelas funcionários e departamentos

Quais as vantagens de Schemas?

- A primeira vantagem é o permissionamento de usuários e grupos
 - Quando um usuário é autorizado em um Schema, ele pode ser autorizado em vários bancos de dados ao mesmo tempo cujos objetos estejam incluso dentro de um Schema
 - A segunda vantagem é o agrupamento físico dos dados
 - Alguns servidores, como as versões mais recentes do SQL Server, permitem que objetos de um mesmo Schema, sejam agrupados fisicamente para administração de backups e carga
-

Quando devo usar Schemas?

- **Em bases de dados com múltiplos bancos de dados** (que tenham vários sistemas, por exemplo) e seja necessário autorizar ou revogar usuários e grupos rapidamente
 - **Em bases de dados cujo permissionamento seja característica essencial** para a segurança dos dados (sistemas antigos cliente-servidor, por exemplo, que dependem muito da base de dados para implementação das regras de negócio)
 - **Em segmentação de dados por grupo de usuários.** Você pode ter no mesmo database duas tabelas com o mesmo nome, estando cada uma em um Schema diferente
-

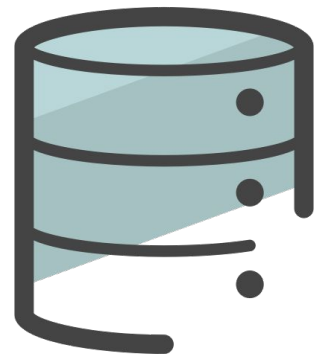
O que são Catálogos?

- São uma coleção nomeada de esquemas
 - O catálogo contém informações detalhadas sobre os vários objetos de interesse do próprio sistema
 - Um catálogo contém um esquema especial, chamado INFORMATION_SCHEMA
 - Oferece informações sobre todos os esquemas no catálogo e todos os seus descritores de elemento
 - Do ponto de vista SQL:
 - Um catálogo geralmente é sinônimo de banco de dados
-

Diferenciando as Nomenclaturas

- Tabela, linha e coluna correspondentes aos termos relação, tupla e atributo do modelo relacional
 - Um servidor de banco de dados é um cluster
 - Um cluster possui **catálogos**
 - Os catálogos têm **esquemas**
 - Os esquemas têm **tabelas**
 - As tabelas possuem **linhas**
 - As linhas têm valores definidos por **colunas**
 - A coluna define o tipo de dados dos valores (texto, data, número e assim por diante)
-

Comando CREATE TABLE e Tipos de Dados de Atributo



CREATE TABLE

- A SQL usa os termos tabela (relação), linha (tupla) e coluna (atributo) para os termos do modelo relacional
 - O principal comando SQL para a definição de dados é o **CREATE**, que pode ser usado para criar esquemas, tabelas (relações) e domínios de atributos
 - O comando **CREATE TABLE** é usado para especificar uma nova relação, dando-lhe um nome e especificando seus atributos e restrições iniciais
-

CREATE TABLE

- Os atributos são especificados primeiro, e cada um deles recebe:
 - Um **nome**, um **tipo de dado** para especificar seu domínio de valores e **restrições de atributo**, como NOT NULL

```
CREATE TABLE <nome_da_tabela>
( <nome_da_col1> <tipo_da_col1> NOT NULL,
  <nome_da_col2> <tipo_da_col2> NOT NULL,
  ...
  PRIMARY KEY <lista_de_nomes_de_col>,
  FOREIGN KEY <nomes_de_col>
  REFERENCES <nome_tab_ref>(<nome_da_col_ref>)
);
```

CREATE TABLE

- Em geral, o esquema SQL em que as relações são declaradas é especificado implicitamente no ambiente em que as instruções CREATE TABLE são executadas
 - Como alternativa, podemos conectar explicitamente o nome do esquema ao nome da relação, separados por um ponto
 - Criando a tabela no schema EMPRESA
 - CREATE TABLE EMPRESA.FUNCIONARIO ...
 - Em vez de
 - CREATE TABLE FUNCIONARIO ...
-

CREATE TABLE

- Exemplo:

- Para criar tabela com chave estrangeira

CREATE TABLE Empregado (...,

NOME DA RESTRIÇÃO → **CONSTRAINT** TrabalhaEm

NOME DA CHAVE → **FOREIGN KEY**(Cod_Dept)

ESTRANGEIRA

NOME DA OUTRA → **REFERENCES** Departamento(Dcod), ...);

TABELA E COLUNA

Tipos de Dados

- Os tipos de dados básicos disponíveis para atributos são numérico, cadeia ou sequência de caracteres, cadeia ou sequência de bits, booleano, data e hora
 - Os tipos de dados são usados na criação de tabelas
 - Serão usadas pelos bancos independentemente da sua arquitetura
 - Tipos de dados suportados pelo SQL
 - Tipos numéricos
 - Tipos caracteres
 - Tipos data e hora
-

Tipos Numéricos

- Os tipos de dados numérico incluem números inteiros de vários tamanhos (**INTEGER** ou **INT** e **SMALLINT**) e números de ponto flutuante (reais) de várias precisões (**FLOAT** ou **REAL** e **DOUBLE PRECISION**)
 - O formato dos números pode ser declarado usando:
 - **DECIMAL(i, j)** ou **DEC(i, j)** ou **NUMERIC(i, j)**
 - i, a precisão, é o número total de dígitos decimais
 - j, a escala, é o número de dígitos após o ponto decimal
-

Tipos Caracteres

- Tipos de dados de cadeia de caracteres de tamanho fixo — **CHAR**(n) ou **CHARACTER**(n), onde n é o número de caracteres
 - O máximo de caracteres de CHAR é 255, porém ele tem melhor desempenho que VARCHAR
 - Tipos de dados de caracteres de tamanho variável — **VARCHAR**(n) ou **CHAR VARYING**(n) ou **CHARACTER VARYING**(n), onde n é o número máximo de caracteres
 - Ao especificar um valor literal de cadeia de caracteres, ele é colocado entre aspas simples (apóstrofes), e é case sensitive (diferencia maiúsculas de minúsculas)
-

Tipos Caracteres

- Para cadeias de **caracteres de tamanho fixo**, uma cadeia mais curta é preenchida com caracteres em branco à direita
 - Exemplo:
 - Se o valor 'Silva' for para um atributo do tipo CHAR(10), ele é preenchido com cinco caracteres em branco para se tornar 'Silva ', se necessário
 - Outro tipo de dado de cadeia de caracteres de tamanho variável, chamado CHARACTER LARGE OBJECT ou CLOB, também está disponível para especificar colunas que possuem grandes valores de texto, como documentos
-

Tipos Data e Hora

- O tipo de dados DATE possui dez posições, e seus componentes são DAY (dia), MONTH (mês) e YEAR (ano) na forma DD-MM-YYYY
 - O tipo de dado TIME (tempo) tem pelo menos oito posições, com os componentes HOUR (hora), MINUTE (minuto) e SECOND (segundo) na forma HH:MM:SS
 - Um tipo de dados TIME WITH TIME ZONE inclui seis posições adicionais para especificar o deslocamento com base no fuso horário universal padrão, que está na faixa de +13:00 a -12:59 em unidades de HOURS:MINUTES
-

Tipos Data e Hora

- Datetime e datetime2
 - O tipo **datetime** é usado para armazenar valores do tipo data e hora
 - A diferença entre datetime e datetime2 está no intervalo de datas que eles suportam entre outras características
 - Intervalo suportado por datetime: 01/01/1753 a 31/12/9999
 - Intervalo suportado por datetime2: 01/01/0001 a 31/12/9999
-

Exemplo de Tipos de Dados

CREATE TABLE FUNCIONARIO

(Pnome	VARCHAR(15)	NOT NULL,
Minicial	CHAR,	
Unome	VARCHAR(15)	NOT NULL,
Cpf	CHAR(11),	NOT NULL,
Datanasc	DATE,	
Endereço	VARCHAR(30),	
Sexo	CHAR,	
Salario	DECIMAL(10,2),	
Cpf_supervisor	CHAR(11),	NOT NULL,
Dnr	INT	

PRIMARY KEY (Cpf),

FOREIGN KEY (Cpf_supervisor) **REFERENCES** FUNCIONARIO(Cpf),

FOREIGN KEY (Dnr) **REFERENCES** DEPARTAMENTO(Dnumero));

Visão Geral dos Tipos de Dados

SQL Padrão (ANSI)

CHAR(tamanho)

CHARACTER(tamanho)

INT

INTEGER

SMALLINT

NUMERIC(precisão, escala)

DECIMAL(precisão, escala)

DEC(precisão, escala)

FLOAT(precisão)

REAL

DOUBLE PRECISION

SQL 2 = Padrão +

VARCHAR(tamanho)

CHAR VARYING(tamanho)

CHARACTER VARYING(tamanho)

NCHAR(tamanho)

NATIONAL CHAR(tamanho)

NATIONAL CHARACTER(tamanho)

VARYING(tamanho)

BIT(tamanho)

BIT VARYING(tamanho)

DATETIME

TIME(precisão)

TIMESTAMP(precisão)

INTERVAL

Sugestões

<https://www.hackerrank.com/challenges/select-all-sql/problem>

<https://www.sqlite.org/index.html>

<https://dev.mysql.com/downloads/installer/>

sqlite

- **CREATE DATABASE**

- **sqlite3 test.db**

- Criar banco de dados "test.db"

```
C:\Users\ismaylesantos\Desktop\banco De Dados>sqlite3 teste5.db
SQLite version 3.36.0 2021-06-18 18:36:39
Enter ".help" for usage hints.
sqlite>
```

- **CREATE TABLE Test (Id int);**

- Cria a tabela Test e usando .schema Test você a instrução SQL da tabela

```
Enter ".help" for usage hints.
sqlite> CREATE TABLE Test( Id int, Nome varchar(80));
sqlite> .schema Test
CREATE TABLE Test( Id int, Nome varchar(80));
sqlite>
```

sqlite

- CREATE DATABASE
 - INSERT INTO Test VALUES (1,'João');
 - Inserir um valor

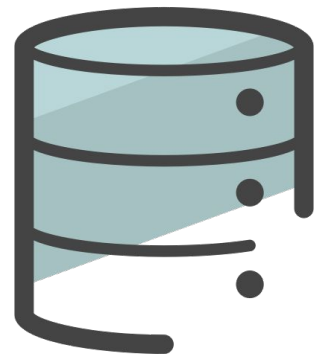
```
sqlite> INSERT INTO Test VALUES (2, 'Pedro');
```

- SELECT * FROM Teste
 - Visualizar Tabela

```
sqlite> SELECT * FROM TEST;  
1|Joao  
2|Pedro  
sqlite>
```

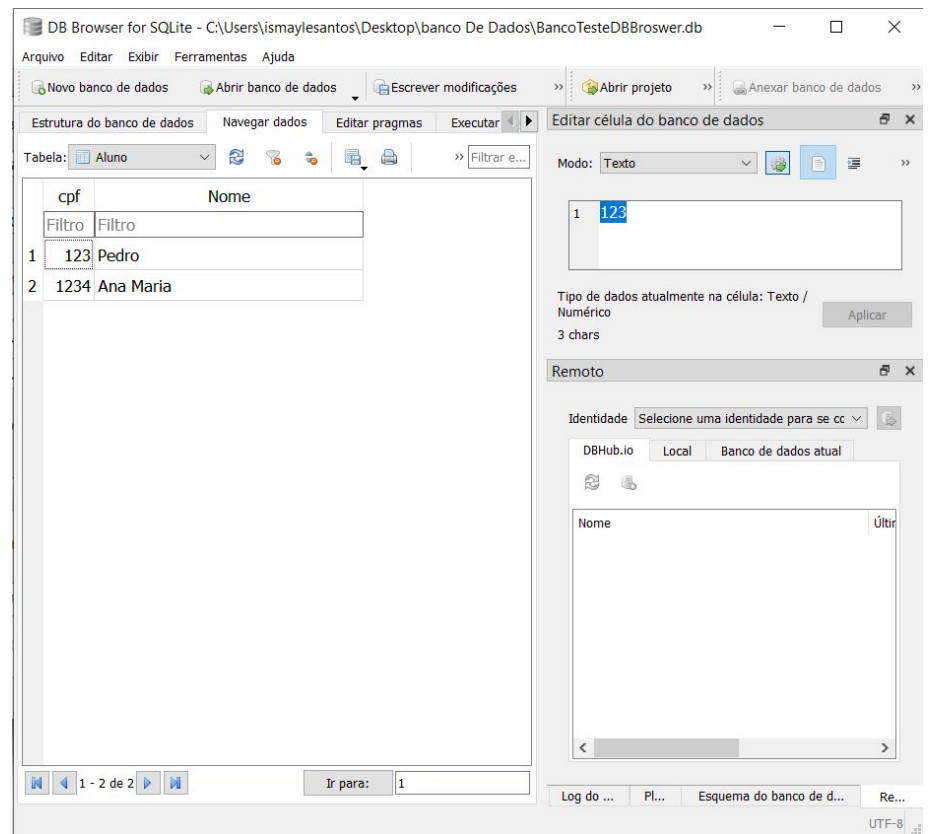
SQLITE 3 + DBBrowser

DROP TABLE e ALTER TABLE em SQL



DBBrowser

- Download:
 - <https://sqlitebrowser.org/>



Tipos de Dados SQLite3

- **INTEGER**
 - Valor inteiro
- **REAL**
 - Valor real
- **TEXT**
 - String de texto
- **BLOB (Binary Large Object)**
 - Coleção de Dados (serve para armazenar vídeos, imagens, sons)

Nota: Não tem no SQLITE o tipo **Boolean** (pode-se usar Integer) e nem os tipos **Date/Time** (mas o SQLITE tem funções que podem ajudar

Tipos de Dados SQLite3

- DATE e TIME

- **date()**

Retorna a data e hora no formato AAAA-MM-DD

- **time()**

Retorna a hora no formato HH:MM:SS: YYYY-MM-DD.

- **datetime()**

- Retorna a data e hora no formato AAAA-MM-DD
HH:MM:SS

- **strftime()**

- Retorna a data formatada de acordo com o formato enviado através do primeiro parâmetro

exemplo: date
('now')

Nota: Podem ser armazenadas como texto

DROP TABLE

- **DROP TABLE** elimina completamente a tabela (vazia ou não)
 - Remove as tuplas da tabela e sua definição do catálogo
 - Depois que uma tabela é removida não é possível recuperá-la
 - Sintaxe:
 - **DROP TABLE** <nome_da_tabela>;
 - Exemplo:
 - **DROP TABLE** Empregado;
-

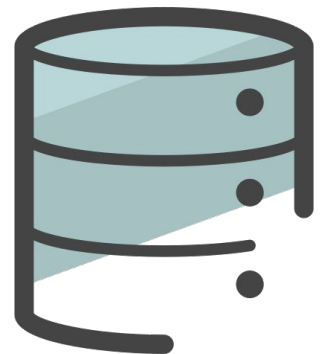
ALTER TABLE

- **ALTER TABLE** modifica um esquema
- Sintaxe:
 - **ALTER TABLE** <nome_da_tabela> <alteração>;
- Exemplo:
 - Para adicionar uma coluna:
 - **ALTER TABLE** Empregado
ADD Telefone **VARCHAR(30)**
ADD sexo **CHAR(1)**
DEFAULT 'F';

Valor padrão
definido para o
atributo sexo

Especificando restrições em SQL

Aqui descreveremos as restrições básicas que podem ser especificadas em SQL como parte da criação de tabela



Restrições e Defaults de Atributo

- Existem três restrições básicas em SQL
 - Restrições de chave e de integridade referencial
 - Restrições sobre domínios de atributos e NULLs
 - Restrições sobre tuplas individuais de uma relação
-

Restrições e Defaults de Atributo

- Como a SQL permite **NULLs** como valores de atributo, uma restrição **NOT NULL** pode ser especificada se o valor NULL não for permitido para determinado atributo
 - O atributo deve ser obrigatoriamente preenchido
 - NULL não é permitido para um determinado atributo
 - Isso sempre é especificado de maneira implícita para os atributos que fazem parte da **chave primária** de cada relação, mas pode ser especificado para quaisquer outros atributos cujos valores não podem ser NULL
-

Restrições e Defaults de Atributo

- Também é possível definir um valor padrão para um atributo anexando a cláusula **DEFAULT** <valor> a uma definição de atributo
 - Adiciona-se a cláusula **DEFAULT** <valor> logo após a restrição
- O valor padrão está incluído em qualquer nova tupla se um valor explícito não for fornecido para esse atributo

```
CREATE TABLE Empregado
```

```
( ...
```

```
Sexo CHAR(1) NOT NULL DEFAULT "F",
```

```
...
```

```
);
```

Restrição

Definição do
valor *default*

Restrições e Defaults de Atributo

- Outro tipo de restrição pode limitar valores de atributo ou domínio usando a cláusula **CHECK** (verificação) após uma definição de atributo ou domínio
 - **Exemplo:**
 - **Dnumero INT NOT NULL CHECK (Dnumero > 0 AND Dnumero < 21);**
-

Exemplos no DB Browser

- Not NULL, Default e Check

Empregado

▼ Avançado

Campos Restrições

Adicionar Remove Mover para o topo Mover para cima Mover para baixo Mover para o fundo

Nome	Tipo	NN	PK	AI	U	Default	Check
Ecod	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Enome	TEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
CPF	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
Salario	REAL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'0,00'	"Salario" > 0

Restrições de Chave e Integridade Referencial

- A cláusula **UNIQUE** especifica chaves alternativas (secundárias)
- Chave alternativa:
 - **UNIQUE KEY** (CPF)

Editar definição da tabela

Tabela

Empregado

▼ Avançado

Campos Restrições

Adicionar Remover Mover para o topo Mover para cima Mover para l

Nome	Tipo	NN	PK	AI	U	Default
Ecod	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
Enome	TEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
CPF	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
Salario	REAL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Restrições de Chave e Integridade Referencial

- Como chaves e restrições de integridade referencial são muito importantes, existem cláusulas especiais dentro da instrução CREATE TABLE para especificá-las
 - A cláusula **PRIMARY KEY** especifica um ou mais atributos que compõem a chave primária de uma relação
 - Se uma chave primária tiver um único atributo, a cláusula pode acompanhar o atributo diretamente
 - Chave primária:
 - PRIMARY KEY (<nomeColuna>),
-

Exemplo de Chave Primária

```
CREATE TABLE Departamento  
( Dcod    INTEGER          NOT NULL,  
  Dnome   VARCHAR(20)    NOT NULL,  
  Cidade  VARCHAR(20) ,  
  PRIMARY KEY (Dcod) ) ;
```

Exemplo de Chave Primária Composta

```
CREATE TABLE Empregado
(   Ecod          INTEGER          NOT NULL,
    Enome         VARCHAR(40)      NOT NULL,
    CPF           VARCHAR(15)      NOT NULL,
    Salario       DECIMAL(7,2) ,
    Cod_Dept      INTEGER          NOT NULL,
    PRIMARY KEY (Ecod, ENome) ) ;
```

Restrições de Chave e Integridade Referencial

- A integridade referencial é especificada por meio da cláusula **FOREIGN KEY** (chave estrangeira)
 - Implementa o conceito de chave estrangeira
 - Chave estrangeira:
 - **FOREIGN KEY (<nomeColuna1>) REFERENCES**
 <nomeTbReferencia>,
 - Uma restrição de integridade referencial pode ser violada quando tuplas são inseridas ou excluídas, ou quando um valor de atributo de chave estrangeira ou chave primária é modificado
-

Exemplo de Chave Estrangeira

Empregado

Ecod	Enome	CPF	Salario	Cod_Dept
------	-------	-----	---------	----------

```
CREATE TABLE Empregado
( ...,
  CONSTRAINT TrabalhaEm
  FOREIGN KEY (Cod_Dept) REFERENCES
  Departamento (Dcod),
  ...
);
```

Exemplo no DB Browser

- Primary Key e Foreign Key

Empregado

▼ Avançado

Campos Restrições

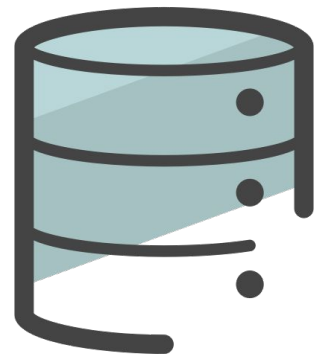
Adicionar Remover Mover para o topo Mover para cima Mover para baixo Mover para o fundo

		Tipo	NN	PK	AI
Nome	Foreign Key	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ecod		TEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Enome		TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CPF		REAL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Salario		INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
TrabalhaEm	"Departamento"("DCod")				

Se for o caso, você pode criar um campo só para ser a Primary Key e marcar a opção AutoIncremento

INSERT, DELETE e UPDATE em SQL

Em SQL, três comandos podem ser usados para modificar o banco de dados: INSERT , DELETE e UPDATE



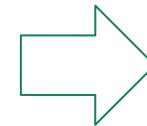
INSERT - 1º Forma

- Em sua forma mais simples, **INSERT** é usado para acrescentar uma única tupla a uma relação
 - Temos de especificar o **nome da relação** e uma **lista de valores** para a tupla
 - Os valores devem ser listados na **mesma ordem** em que os atributos correspondentes foram especificados no comando CREATE TABLE
-

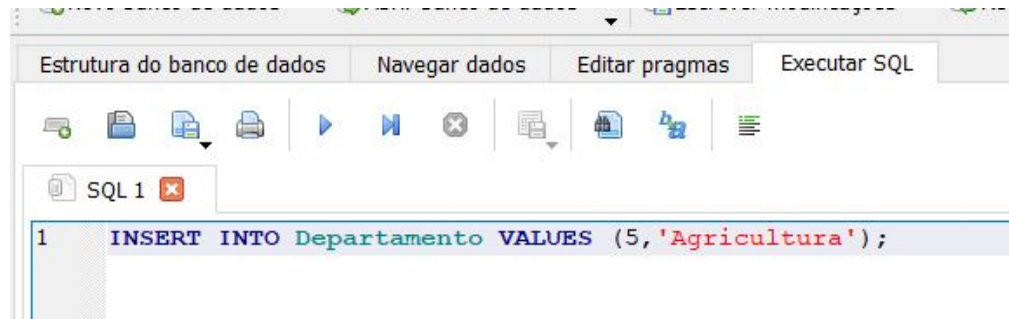
INSERT - 1º Forma

- Exemplo:
 - Acrescentar uma nova tupla à relação FUNCIONARIO

```
INSERT INTO FUNCIONARIO  
VALUES ( 'Ricardo', 'K', 'Marini',  
        '65329865388', '30-12-  
        1962', 'Rua Itapira, 44,  
        Santos, SP', 'M', 37.000,  
        '65329865388', 4 );
```



Valores na
mesma ordem
das colunas
criadas no
comando
CREATE TABLE



INSERT - 2º Forma

- Uma segunda forma da instrução **INSERT** permite que o usuário especifique nomes de atributo explícitos que correspondem aos valores fornecidos no comando **INSERT**
 - É útil se uma relação tiver muitos atributos, mas apenas alguns deles recebem valores em uma nova tupla
 - Porém, os valores precisam incluir todos os atributos com a especificação **NOT NULL** e nenhum valor padrão
-

INSERT - 2º Forma

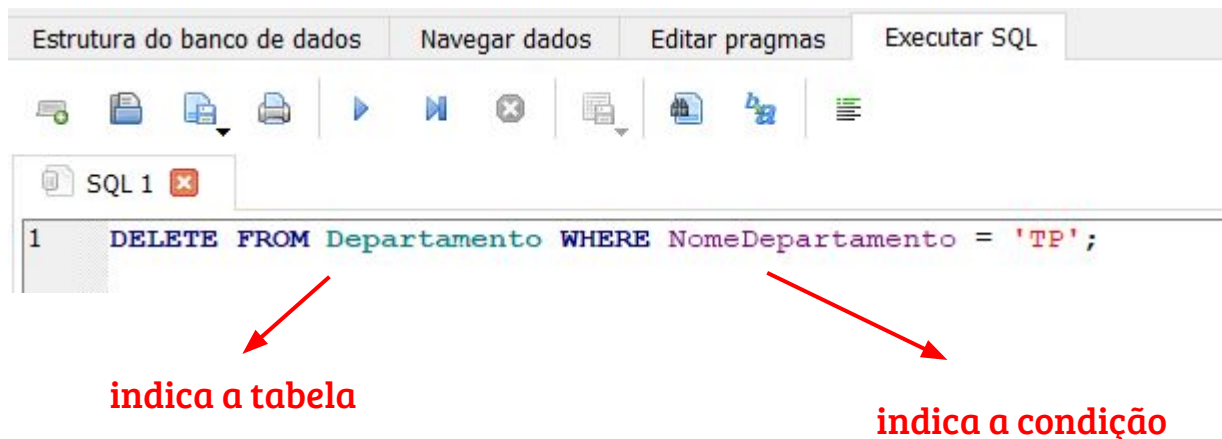
- Exemplo:
 - Inserir uma tupla para um novo FUNCIONARIO do qual conhecemos apenas os atributos Pnome, Unome, Dnr e Cpf

```
INSERT INTO  FUNCIONARIO (Pnome,  
Unome, Dnr, Cpf)  
VALUES      ('Ricardo', 'Marini', 4,  
             '65329865388');
```

- Os atributos não especificados são definidos como seu valor DEFAULT ou NULL , e os valores são listados na mesma ordem que os atributos são listados no próprio comando INSERT
-

DELETE

- O comando **DELETE** remove tuplas de uma relação
- Ele inclui uma cláusula **WHERE**, semelhante a que é usada em uma consulta SQL, para selecionar as tuplas a serem excluídas
 - As tuplas são explicitamente excluídas uma tabela por vez



DELETE

- Dependendo do número de tuplas selecionadas pela condição na cláusula **WHERE**, zero, uma ou várias tuplas podem ser excluídas por um único comando **DELETE**
 - Uma cláusula **WHERE** inexistente especifica que todas as tuplas na relação deverão ser excluídas; porém, a tabela permanece no banco de dados como uma tabela vazia
 - Temos de usar o comando **DROP TABLE** para remover a definição da tabela
-

UPDATE

- O comando **UPDATE** é usado para modificar valores de atributo de uma ou mais tuplas selecionadas
 - Assim como no comando DELETE , uma cláusula **WHERE** no comando UPDATE seleciona as tuplas a serem modificadas em uma única relação
-

UPDATE

- Uma cláusula **SET** adicional no comando UPDATE especifica os atributos a serem modificados e seus novos valores
- Por exemplo, para alterar o local e número de departamento que controla o número de projeto 10 para 'Santo André' e 5, respectivamente:

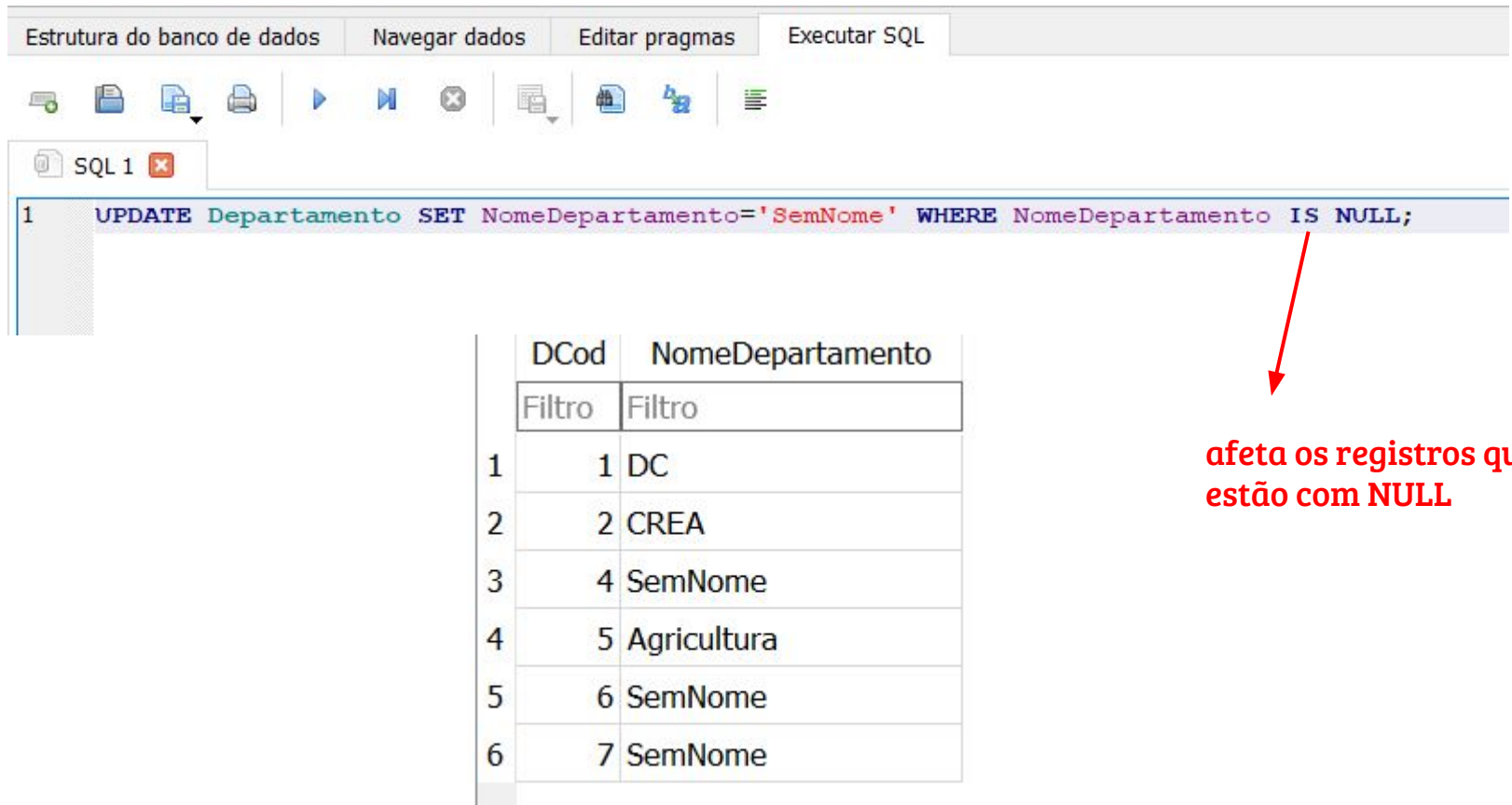
```
UPDATE PROJETO
```

```
SET Projlocal = 'Santo André', Dnum=5
```

```
WHERE Projnumero =10;
```

Exemplo no DBBrowser

- UPDATE



The screenshot shows the DBBrowser application interface. At the top, there are tabs for 'Estrutura do banco de dados', 'Navegar dados', 'Editar pragmas', and 'Executar SQL'. Below the tabs is a toolbar with various icons. A tab labeled 'SQL 1' is active, displaying the following SQL statement:

```
1 UPDATE Departamento SET NomeDepartamento='SemNome' WHERE NomeDepartamento IS NULL;
```

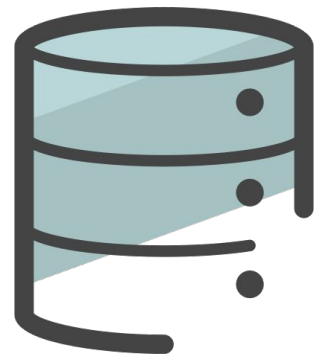
Below the SQL editor, a table of data is displayed. The table has two columns: 'DCod' and 'NomeDepartamento'. The data is as follows:

	DCod	NomeDepartamento
	Filtro	Filtro
1	1	DC
2	2	CREA
3	4	SemNome
4	5	Agricultura
5	6	SemNome
6	7	SemNome

A red arrow points from the text 'afeta os registros que estão com NULL' to the 'WHERE NomeDepartamento IS NULL;' part of the SQL statement.

afeta os registros que
estão com NULL

Consultas de Recuperação Básicas em SQL



Consultas **SELECT-FROM-WHERE**

- O comando **SELECT** permite recuperar os dados de um objeto do banco de dados, como uma tabela, view e, em alguns casos, uma stored procedure (alguns bancos de dados permitem a criação de procedimentos que retornam valor)
- A forma básica do comando **SELECT**, às vezes chamada de mapeamento ou bloco select-from-where, é composta pelas três cláusulas **SELECT**, **FROM** e **WHERE**

SELECT	<lista atributos>	seleciona
FROM	<lista tabelas>	a partir de
WHERE	<condição>;	onde

Consultas SELECT-FROM-WHERE

- **<lista atributos>** é uma lista de nomes de atributo cujos valores devem ser recuperados pela consulta
 - **<lista tabelas>** é uma lista dos nomes de relação exigidos para processar a consulta
 - **<condição>** é uma expressão condicional (booleana) que identifica as tuplas a serem recuperadas pela consulta
-

Operadores de Comparação Lógica

- Em SQL, os operadores básicos de comparação lógicos para comparar valores de atributo entre si e com constantes literais são: =, <, <=, >, >= e <>
- A principal diferença sintática das demais linguagens é o operador diferente
- Exemplo:
 - Recuperar a data de nascimento e o endereço do(s) funcionário(s) cujo nome seja 'João B. Silva'

```
SELECT  Datanasc, Endereco
FROM    FUNCIONARIO
WHERE   Pnome='João' AND Minicial='B' AND
        Unome='Silva';
```

Obrigado!

Por hoje é só pessoal...

Dúvidas?



IsmayleSantos



ismayle@.ufc.br



@IsmayleSantos
