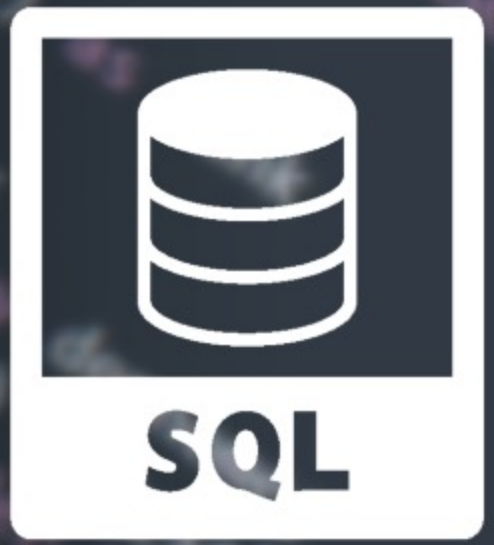
# Linguagem de Definição de Dados

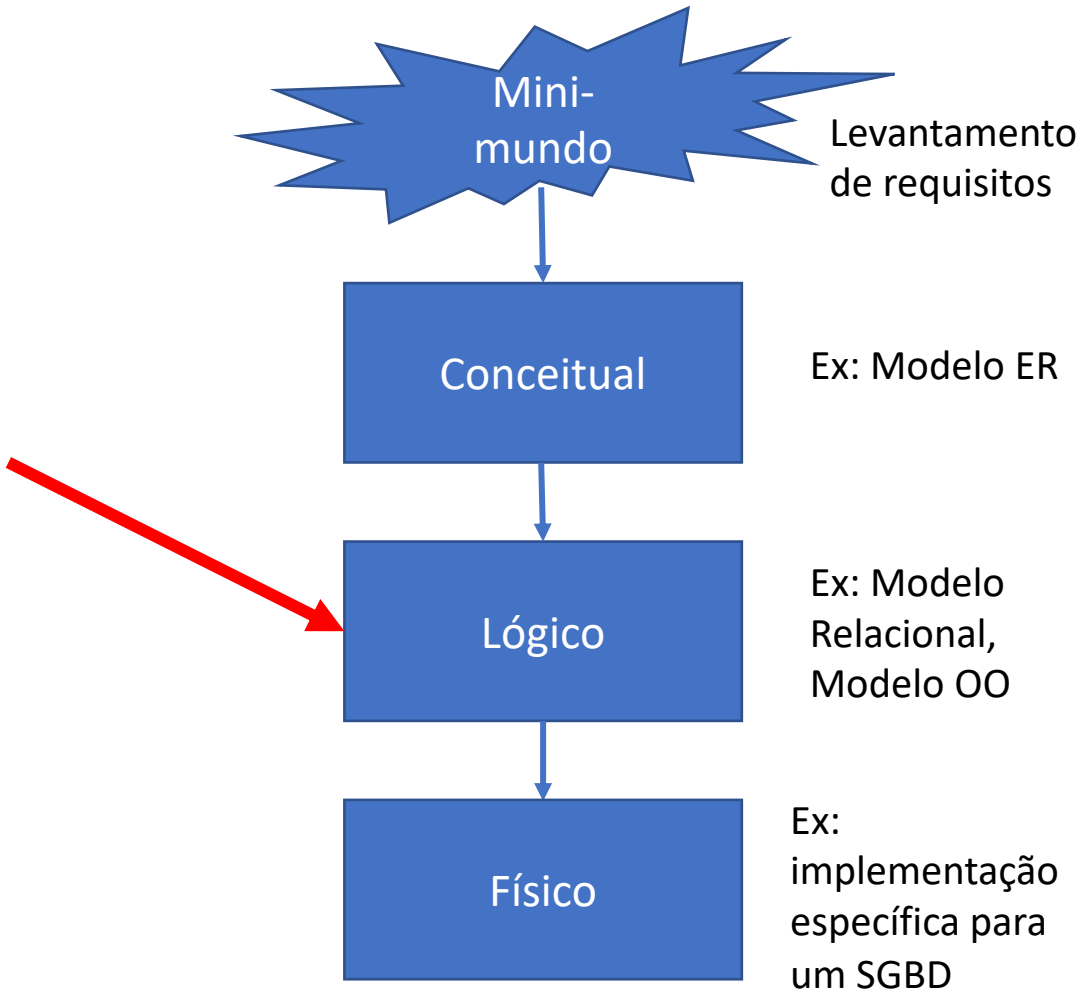
Vanessa Borges




## Banco de Dados

# Visão Geral

- Já criamos **abstração** do mundo real em um modelo conceitual de dados
- **Agora iremos descrever a estrutura de um banco de dados da forma como será manipulado pelo SGBD**
- Esse modelo é utilizado nos SGBDs comerciais tradicionais



# Modelo relacional - Modelo Físico

- Seguindo a hierarquia baseada no nível de abstração exigido por cada modelo, o modelo **físico é o que apresenta a menor abstração**.
- Ele **está relacionado ao SGBD escolhido** para a implementação do sistema objeto. Nesse modelo, as características mais específicas do SGBD devem estar presentes (**tipo de dados, funções, procedimentos, nome de atributos, gatilhos, visões, permissões e etc**).
- O grande sucesso comercial dos SGBDs relacionais se deve principalmente à linguagem SQL.

# Modelo relacional - Modelo Físico

**Exemplos de SGBD**

# SQL – Structured Query Language

- SQL – *Structured Query Language*
- Originalmente: SEQUEL – *Structured English QUEry Language*
- Criada pela IBM Research

  - ***1970*** *- SEQUEL do SYSTEM R nos laboratórios da IBM 1986 – Padrão ANSI chamado SQL1 ou SQL-86*
    ***1992*** *– SQL2 => introdução de chave estrangeira*
  - ***1999*** *– SQL3 => gatilhos; características objeto-relacional(tipo LOB-Large Object); consultas recursivas, etc.*
  - ***2003*** *– SQL-2003: mais recursos para definição de tipos (OO); SQL/XML; etc.*
  - ***2008*** *– SQL-2008: mais recursos para orientação a objetos*

# SQL – Structured Query Language

- É baseada em **cálculo relacional de tuplas** com alguns recursos da **álgebra relacional**

- SQL
  - Se tornou um padrão para banco de dados relacionais comerciais
  - Facilita a migração entre SGBDs
  - Tem instruções para definição de dados, consultas e atualizações (DDL e DML)

# Linguagens – DDL e DML

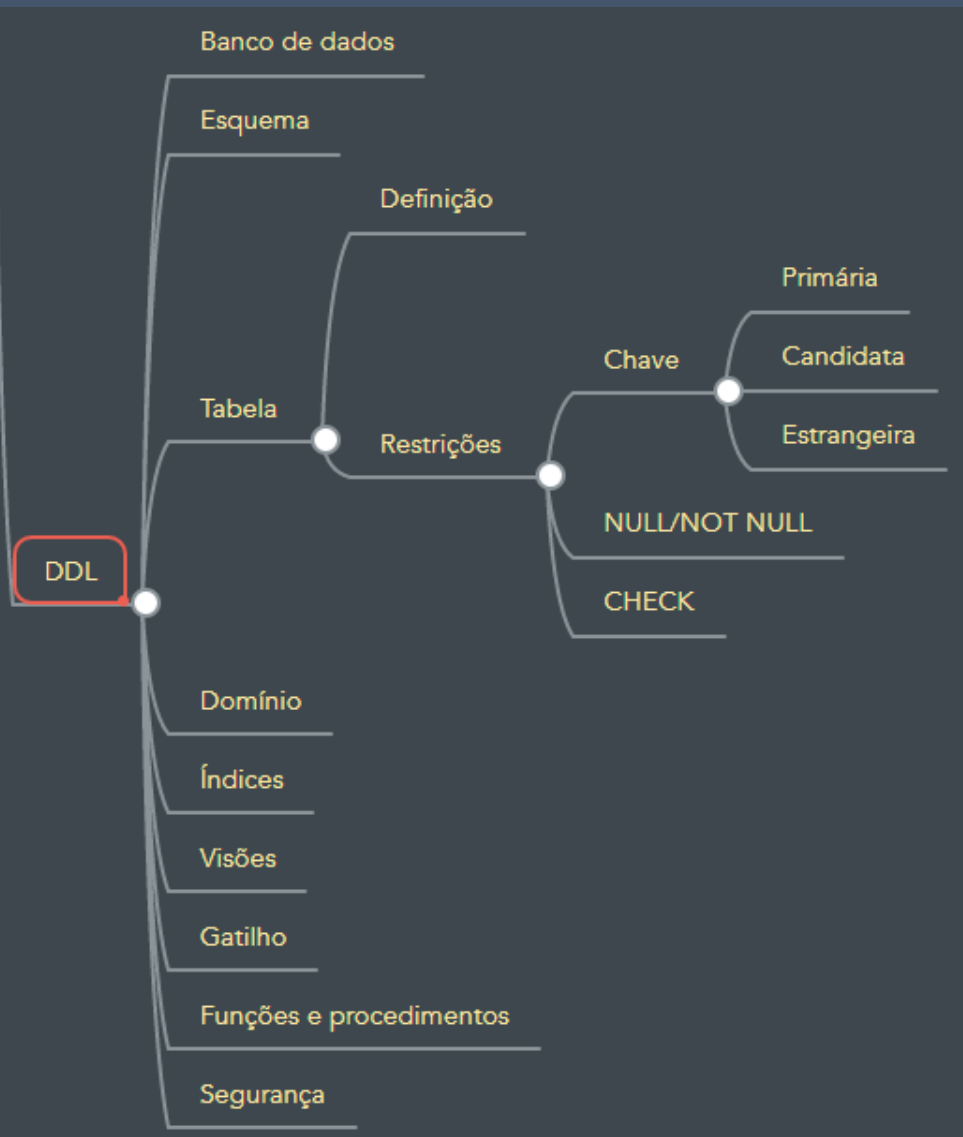
- Cada SGBD possui uma gramática para cada comando da linguagem SQL.
- Grupos de Linguagem SQL
  - **Linguagem de definição de dados (LDD / DDL)**
    - Create
    - Alter
    - Drop
    - Manipulam o esquema do SGBD.
  - **Linguagem de manipulação de dados (LMD / DML)**
    - Insert
    - Update
    - Delete
    - Manipulam as instâncias do SGBD.

# Linguagens – DDL e DML

- É utilizada para definir as **relações**, os **relacionamentos**, os **atributos**, os **tipos de atributos**, **visões**, **índices**, **triggers**, **procedimentos**, **funções** e outros “objetos” de um Banco de Dados.

SQL permite que uma tabela (relação) tenha duas ou mais tuplas idênticas em todos os seus valores de atributo. Assim, em geral, uma tablema SQL não é um conjunto de tuplas (não permite membros idênticos) mas sim um **multiconjunto (*bag*)** de tuplas.

# Abrangência da DDL



# Convenção – Sintaxe dos comandos demonstrados

| Convenção | Significado |
|---|---|
| UPPERCASE (maiúscula) | Palavras reservadas do SQL. |
| \| (barra vertical) | Separa elementos opcionais da sintaxe dentro de colchetes ou chaves. Somente um dos itens pode ser escolhido. |
| [] (colchetes) | Item de sintaxe opcional. Os colchetes não fazem parte do comando. |
| {} (chaves) | Item da sintaxe obrigatório. As chaves não fazem parte do comando. |
| [,...] | O item precedente pode ser repetido N vezes. A separação entre os itens é feita por uma vírgula. |
| [ ...] | O item precedente pode ser repetido N vezes. A separação entre os itens é feita por um espaço em branco. |
| <> | Identificadores ou constantes SQL informadas pelo usuário. |

# Definição de banco de dados
# CREATE DATABASE

- **Especifica um novo banco de dados relacional**

Conjunto de esquemas S = {R1, R2,..., Rn}
Conjunto de restrições de integridade RI

- **Sintaxe:**

```
CREATE DATABASE <nome>
[ WITH OWNER = <dono_do_banco_de_dados> ]
[ENCODING = <codificação>] ;
```

Normalmente, o criador se torna o dono do novo banco de dados

- **Exemplo:**

**CREATE DATABASE empresadb;**


# Definição de banco de dados
# ALTER DATABASE

O comando **ALTER DATABASE** altera os atributos de um banco de dados.

**ALTER DATABASE** <*nome*> **SET** <*parâmetro*> { TO | = } { *valor* | DEFAULT } ;

**ALTER DATABASE** <*nome*> **RESET** <*parâmetro*> ;

**ALTER DATABASE** <*nome*> **RENAME TO** <*novo_nome*> ;

**ALTER DATABASE** <*nome*> **OWNER TO** <*novo_dono*>;


# Definição de banco de dados
# DROP DATABASE

- Remover a definição de um  banco de dado

  **DROP DATABASE** [**IF EXISTS**] <nome_do_database>
  [**CASCADE** | **RESTRICT**];

- IF EXISTS
  - Não retorna erro se o banco de dados não existir
- CASCADE
  - Remove um banco de dados, incluindo todos os seus esquemas, tabelas e outros elementos
- RESTRICT
  - Remove um banco de dados somente se não existirem elementos definidos

# Esquema em SQL

- O conceito de um esquema SQL foi incorporado a partir do SQL2 com o objetivo de **agrupar tabelas e outros elementos referentes ao banco de dados de uma aplicação**
  - Um banco de dados pode conter várias aplicações sobre ele. Contudo, é possível criar um esquema  para cada aplicação sobre o mesmo banco de dados
  - **Os principais elementos de um esquema:**
    - Tabela
    - Restrições
    - Trigger
    - Visões
    - Procedimentos
    - Funções
    - Índices

# Esquema e Conceitos de Catálogo em SQL

- **Integração entre os esquemas:**
  - Restrições de **integridade referencial** podem ser definidas sobre duas tabelas **apenas se elas estão contidas em esquemas do mesmo catálogo**.
  - **Esquemas pertencentes ao mesmo catálogo** podem compartilhar elementos como **tabela**, **funções**, e **definições de domínio** dentre outros.


# Definição de esquema
# CREATE SCHEMA

**SCHEMA – coleção de objetos de banco de dados relacional.**

- O comando utilizado para **criar** um Esquema  possui a seguinte sintaxe:

```
CREATE SCHEMA [IF NOT EXISTS] <nome_do_esquema>
AUTHORIZATION [<nome_grupo|nome_usuário>];
```

- O comando a seguir **cria** um esquema denominado EMPRESA cujo **dono** é o usuário ‘Jsmith’.

```
CREATE SCHEMA EMPRESA AUTHORIZATION ‘Jsmith’;
```

# Definição de esquema
# CREATE SCHEMA

Em geral, os usuários não têm permissão (privilégios) para criar esquemas e elementos nos esquemas. O privilégio para criar esquemas, tabelas e outros elementos deve ser concedido explicitamente pelo administrador do banco de dados.


# Definição de esquema [ALTER | DROP] SCHEMA

- Sintaxe para alterar a definição de um esquema
  **ALTER SCHEMA** <nome_do_esquema> **RENAME TO** <novo_nome_do_esquema>;
  **ALTER SCHEMA** <nome_do_esquema> **OWNER TO** <novo_nome_grupo | novo_nome_usuário>;

- Sintaxe para remover a definição de um esquema
  **DROP SCHEMA** <nome_do_esquema> [**CASCADE** | **RESTRICT**];

  - **CASCADE**
    - Remove um esquema de BD, incluindo todas as suas tabelas e os seus outros elementos
  - **RESTRICT**
    - Remove um esquema de BD somente se não existirem elementos definidos para esse esquema

# Definição de tabela

- **Tabelas da base (relações da base)**
  - A relação e suas tuplas são realmente criadas e armazenadas como um arquivo pelo SGBD

- **Tabelas temporárias**
  - As tabelas temporárias são automaticamente removidas no final da sessão ou, opcionalmente, no final da transação corrente

- **Relações virtuais**
  - Criadas por meio da instrução CREATE VIEW

# Definição de tabela
# CREATE TABLE

**Comando utilizado para definir uma nova relação considerando seus atributos e restrições.**

**CREATE** [TEMPORARY | TEMP] **TABLE** **<nome da tabela>**

**( [..., <atributo> <tipo> [NULL|NOT NULL] [restrição do atributo],**

**[CONSTRAINT <nome da restrição de tabela> <tipo da restrição de tabela>, ...]);**

**Exemplo:**

```
CREATE TABLE cliente (
<declaração dos atributos, tipos e restrições>
nome        VARCHAR(20),
cpf         VARCHAR(14),
... ,
<declaração das restrições de tabela>
CONSTRAINT pk_cliente PRIMARY KEY (cliid),
CONSTRAINT unique_cpf UNIQUE (cpf),
...) ;
```

# Linguagens – DDL – Tipos de Dados

- **Um SGBD suporta vários tipos de dados.**
  - Tipo numérico:
    - INTEGER, INT, SMALLINT, BIGINT, FLOAT, REAL, DOUBLE, DECIMAL, NUMERIC, PRECISION
  - Cadeia de caractere:
    - CHAR, CHARACTER, VARCHAR, CHARACTER VARYING, CHARACTER LARGE OBJECT (CLOB)
  - Cadeia de bits:
    - BIT, BIT VARYING, BINARY LARGE OBJECT (BLOB)
  - Booleanos:
    - TRUE, FALSE, NULL, UNKNOWN
  - Data:
    - DATE, TIME, DATETIME
  - Outros
    - TIMESTAMP, INTERVAL, etc


# Definição de tabela
# CREATE TABLE

- É possível especificar um valor default para o atributo anexando a cláusula **DEFAULT <valor>** à definição do atributo.

```
CREATE TABLE funcionario (
    pnome varchar(255),
    minicial character(1),
    unome varchar(255),
    cpf varchar(11) PRIMARY KEY,
    datanasc date,
    endereco varchar(255),
    sexo character(1),
    salario numeric(10,2) NOT NULL,
    cpf_supervisor varchar(11),
    dnr integer
);
```


# Definição de tabela
# CREATE [ TEMPORARY | TEMP ] TABLE

- Uma tabela temporária (temporary table) é uma tabela como as demais, exceto pelo fato de que ela somente existe enquanto a sessão na qual ela foi criada estiver ativa, ou seja, assim que a conexão com o banco é fechada a tabela temporária é apagada do banco.
- No caso de tabelas temporárias é possível encontrar a tabela por meio de uma consulta

```
SELECT schemaname, tablename
FROM pg_tables
WHERE tablename = 'nome da tabela';
```

- **pg_tables** é uma view que fornece informações sobre as tabelas do banco de dados
  - Atributos: schemaname, tablename, tableowner, tablespace, etc

# Alteração de tabela
# ALTER TABLE

- Utilizada para:
  - Adicionar coluna
  - Remover coluna
  - Adicionar restrição
  - Remover restrição
  - Mudar valor padrão
  - Mudar tipo de dado de coluna
  - Mudar nome de coluna
  - Mudar nome de tabela


# Alteração de tabela
# ALTER TABLE

- **Adicionar coluna**
  - Inicialmente a nova coluna é preenchida com o valor padrão especificado, ou nulo se a cláusula DEFAULT não for especificada.

  - Adicionar a coluna idade na tabela funcionário

    **ALTER TABLE** funcionario **ADD COLUMN** idade **integer**;
  - Adicionar uma coluna com uma restrição

    **ALTER TABLE** funcionario **ADD COLUMN** idade **integer CHECK (**idade > 18**);**

- **Remover coluna**
  - Remover a coluna idade da tabela funcionário

    **ALTER TABLE** funcionario **DROP COLUMN** idade;
  - Remover uma coluna que é referenciada em outro item de banco de dados

    **ALTER TABLE** funcionario **DROP COLUMN** idade **CASCADE**;


# Alteração de tabela
# ALTER TABLE

- **Adicionar uma restrição**
  - A restrição será verificada imediatamente, portanto os dados da tabela devem satisfazer a restrição para esta poder ser adicionada

    **ALTER TABLE** funcionario **ADD CHECK (**sexo IN (‘M’, ‘F’**))**;

    **ALTER TABLE** funcionario **ADD CONSTRAINT** u1 **UNIQUE** (pnome, minicial, unome);

    **ALTER TABLE** funcionario **ADD FOREIGN KEY** (dnr) **REFERENCES** departamento(dnumero);
- **Remover uma restrição**
  - Para remover uma restrição é necessário conhecer seu nome.
    - Utilize o comando \d *nome_da_tabela* do psql para descobrir o nome da restrição.

    **ALTER TABLE** <nome_da_tabela> **DROP CONSTRAINT** <nome_da_restrição>;
  - Pra remover uma restrição que possui dependências com outros objetos, utilize **CASCADE**

    **ALTER TABLE** <nome_da_tabela> **DROP CONSTRAINT** <nome_da_restrição> **CASCADE**;


# Alteração de tabela
# ALTER TABLE

- **Muda valor padrão da coluna**
  - Define um novo valor padrão para a coluna preço da tabela produtos
    - Esse comando não afeta nenhuma coluna existente na tabela, apenas muda o valor padrão para os próximos comandos INSERT

    **ALTER TABLE** funcionario **ALTER COLUMN** salario **SET DEFAULT** 0**;**

- **Remover valor padrão da coluna**
  - É o mesmo que definir o valor NULL como sendo o valor padrão

    **ALTER TABLE** funcionario **ALTER COLUMN** salario **DROP DEFAULT;**


# Alteração de tabela
# ALTER TABLE

- **Muda tipo de dado de coluna**
  - Converte a coluna em um tipo de dado diferente
    - Esse comando somente será bem-sucedido somente se todas as entradas existentes na coluna puderem ser convertidas para o novo tipo por meio de uma conversão implícita

```
ALTER TABLE funcionario ALTER COLUMN salario TYPE numeric(10,2);
```


# Alteração de tabela
# ALTER TABLE

- **Muda nome de coluna**
  - Renomeia a coluna cod_prod da tabela produtos para cod_produto

```
ALTER TABLE produtos RENAME COLUMN cod_prod TO cod_produto;
```


# Alteração de tabela
# ALTER TABLE

- **Muda nome de tabela**
  - Renomeia a tabela produtos para equipamentos

```
ALTER TABLE funcionario RENAME TO empresa_funcionario;
```


# Remoção de tabela
# DROP TABLE

**Comando utilizado para remover uma tabela e seus “objetos”**

- Não pode ser excluída a tabela que possui alguma referência. Neste caso, deve-se primeiro excluir a tabela que possui algum campo que a está referenciando e depois excluir a tabela inicial.

- Sintaxe:

**DROP TABLE <nome da tabela>;**

- Exemplo:

```
-- Remove a tabela departamento
DROP TABLE Departamento;
```

# Definição de domínio

- **Domínio**
  - Nome usado com a especificação de atributo
  - Torna mais fácil mudar o tipo de dado para um domínio que é usado por diversos atributos
  - Melhora a legibilidade do esquema


# Definição de domínio
# [ CREATE | DROP ] DOMAIN

Cria um domínio para um tipo de dado.

- Sintaxe:

  [**CREATE | DROP] DOMAIN** <nome_do_domínio> **AS** <tipo_do_domínio>
  <expressão default> <restrições>;

```
CREATE DOMAIN tipo_cpf AS CHAR(11);
```

```
CREATE TABLE dependente (
    fcpf tipo_cpf NOT NULL,
    nome_dependente varchar(255) NOT NULL,
    sexo character(1),
    datanasc date,
    parentesco varchar(255),
    PRIMARY KEY (fcpf, nome_dependente)
);
```

```
CREATE TABLE departamento (
    dnome varchar(255) NOT NULL,
    dnumero integer PRIMARY KEY,
    cpf_gerente tipo_cpf,
    data_inicio_gerente date
);
```

LABORATÓRIO DE BANCO DE DADOS

# SQL - Restrições

PostgreSQL

Vanessa Borges – vanessa@facom.ufms.br

# Restrições de integridade

- **Garantem que as mudanças feitas no banco de dados por usuários autorizados não resultam em uma perda de consistência de dados**

- São definidas em expressões ou comandos que são escritos uma vez e guardados na BD e serão executados como resposta a certos eventos

- Em SQL-92 há maneira de especificar restrições de integridade como parte de um esquema de BD:
  - Restrições de chave;
  - Restrições de integridade referencial;
  - Restrições de domínio;

# Restrições de chave

- Restringem os valores permitidos em determinados atributos de relações
- São especificadas como parte da instrução **CREATE TABLE**
  - **Chaves candidatas** são especificadas usando **UNIQUE**
    - Podem existir várias declarações UNIQUE mas só uma chave primária
  - **Chaves primárias** são especificadas usando **PRIMARY KEY**
    - Especifica um ou mais atributos que compõem a chave primária da relação

CARRO

| Placa | Numero_chassi | Marca | Modelo | Ano |
|---|---|---|---|---|
| Itatiaia ABC-7039 | A6935207586 | Volkswagen | Gol | 02 |
| Itu TVP-3470 | B4369668697 | Chevrolet | Corsa | 05 |
| Santos MPO-2902 | X8355447376 | Fiat | Uno | 01 |
| Itanhaem TFY-6858 | C4374268458 | Chevrolet | Celta | 99 |
| Itatiba RSK-6279 | Y8293586758 | Renault | Clio | 04 |
| Atibaia RSK-6298 | U0283657858 | Volkswagen | Parati | 04 |

# Restrições de chave UNIQUE

Atributo UNIQUE

**UNIQUE (A1, A2, ... , An);**

- A especificação **UNIQUE** diz que o atributo A1, ... An formam uma **chave candidata**, ou seja, duas tuplas na relação **não** podem ser iguais em todos os atributos listados
- Atributos de chave candidata **podem ser NULL**, a menos que tenham sido declarados explicitamente como NOT NULL


# Restrições de chave PRIMARY KEY

- **Chaves primárias** são especificadas usando **PRIMARY KEY**
  - Especifica um ou mais atributos que compõem a chave primária da relação
  - PRIMARY KEY = UNIQUE + NOT NULL

```
CREATE TABLE departamento (
    dnome varchar(255) NOT NULL,
    dnumero integer PRIMARY KEY,
    cpf_gerente varchar(11),
    data_inicio_gerente date
);
```

```
CREATE TABLE departamento (
    dnome varchar(255) NOT NULL,
    dnumero integer,
    cpf_gerente varchar(11),
    data_inicio_gerente date,
    CONSTRAINT pkdnumero PRIMARY KEY(dnumero)
);
```


# Restrições de chave
# PRIMARY KEY – Chave composta

- **Chaves primárias** compostas de **mais de um atributo**

```
CREATE TABLE dependente (
    fcpf varchar(11),
    nome_dependente varchar(255),
    sexo character(1),
    datanasc date,
    parentesco varchar(255),
    PRIMARY KEY (fcpf, nome_dependente)
);
```

~~
```
CREATE TABLE dependente (
    fcpf varchar(11) PRIMARY KEY,
    nome_dependente varchar(255) PRIMARY KEY,
    sexo character(1),
    datanasc date,
    parentesco varchar(255)
);
```
~~

# Restrições baseadas em atributos

- Os valores permitidos **para os atributos** podem ser restringidos por meio de:
  - **Restrições expressas na sua definição**.
  - **Restrições expressas num domínio usado na sua definição**.

- Se uma restrição for violada a instrução SQL é desfeita (*rollback*) e há um erro em tempo de execução.


# Restrições baseadas em atributos Valor Nulo

Atributo
Valor Nulo

- SQL permite NULLs como valor de atributos
- Uma restrição NOT NULL pode ser especificada se o valor NULL não for permitido para determinado atributo
  - Exemplo: na relação aluno o nome não pode ser NULL
- Atributos que fazem parte da chave primária por padrão possuem a restrição de NULL

```
CREATE TABLE departamento (
    dnome varchar(255) NOT NULL,
    dnumero integer PRIMARY KEY,
    cpf_gerente varchar(11),
    data_inicio_gerente date
);
```

# Restrições baseadas em atributos CHECK

Atributo CHECK

- A restrição de check (verificação) especifica uma condição que precisa ser satisfeita por cada tupla da relação.
- O predicado de uma cláusula CKECK pode ser uma subconsulta SQL

```
CREATE TABLE departamento (
    dnome varchar(255) NOT NULL,
    dnumero integer PRIMARY KEY,
    cpf_gerente varchar(11),
    data_inicio_gerente date,
    CONSTRAINT chdnumero CHECK (dnumero>0 AND dnumero<20)
);
```

**Verifica se o atributo dnumero é maior que zero e menor que 20**

# Restrições baseadas em atributos CHECK em domínio

Atributo CHECK

Domínio refere-se a especificação de um tipo de dado

- Um domínio pode ser declarado e seu nome usado com a especificação de atributo.
- Na definição de um domínio utilizar a seguinte sintaxe:

```
CREATE DOMAIN <nome_do_dominio> AS <tipo_de_dado> [CHECK (<expressão>)];
```

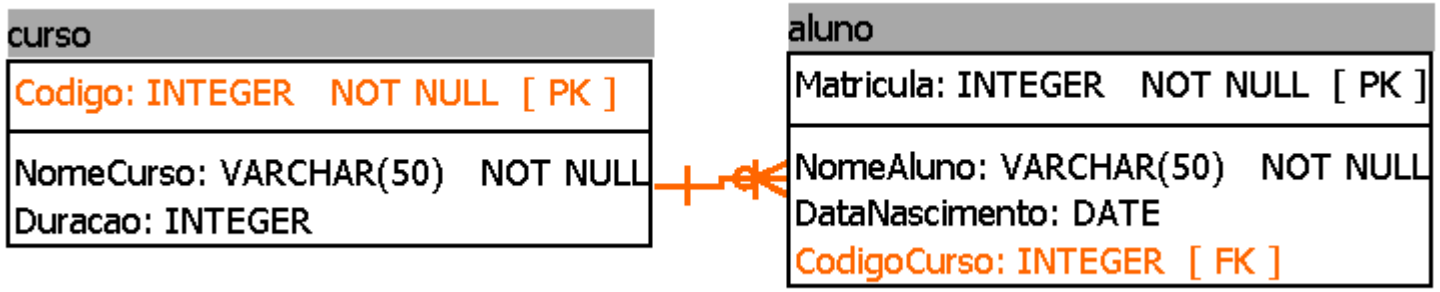
- Exemplo:

```
CREATE DOMAIN d_dnumero AS INTEGER
CHECK (dnr>0 AND dnr<21);
```

```
CREATE TABLE funcionario (
    pnome varchar(255),
    minicial character(1),
    unome varchar(255),
    cpf varchar(11) PRIMARY KEY,
    datanasc date,
    endereco varchar(255),
    sexo character(1),
    salario numeric(10,2) NOT NULL,
    cpf_supervisor varchar(11),
    dnr d_dnumero
);
```

# Restrição de integridade referencial

- **Integridade referencial** garante que um valor que aparece em uma relação para determinado conjunto de atributos também aparecerá para certo conjunto de atributos em outra relação.
- Para isso **chaves estrangeiras (FOREING KEY)** são especificadas como parte do comando **CREATE TABLE**



# Violação de integridade referencial

- A violação da integridade referencial pode ocorrer quando:
  - **Tuplas são inseridas ou excluídas**
  - **Quando atributos de chave estrangeira ou chave primária são alterado**

- A ação default da SQL é rejeitar a operação que causou a violação da integridade (RESTRICTED). Porém o projetista pode especificar ações alternativas.


# Restrição de integridade referencial CREATE TABLE ... PRIMARY KEY ... FOREIGN KE

Integridade referencial

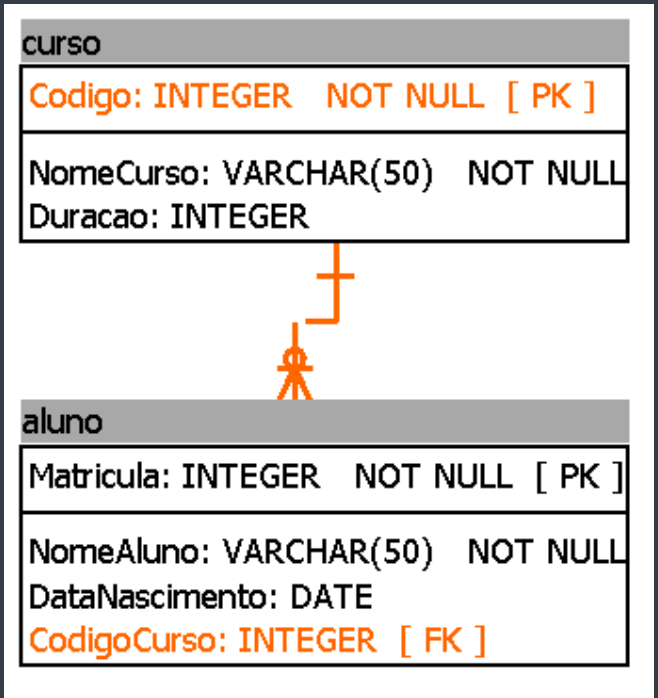
- Comando que cria uma tabela considerando suas restrições de chave estrangeira
- A integridade referencial é especificada por meio da cláusula **FOREIGN KEY**

**CREATE TABLE [<nome do esquema>.]<nome da tabela>...**
**FOREIGN KEY (<coluna$_1$> [,...,<coluna$_n$>])**
**REFERENCES <tabela_ref>([<coluna_ref>[,...,<coluna_ref]])**
**[ON DELETE <acao_ref>] [ON UPDATE <acao_ref>];**

**<acao_ref>**

- NO ACTION ou RESTRICTED → **impede** a ação na tabela mestre <tabela_ref>
- CASCADE → **propaga** a ação da tabela mestre
- SET NULL → valores de referencias alterados para nulo
- SET DEFAULT → valores de referencias alterados para default

# Restrição de integridade referencial

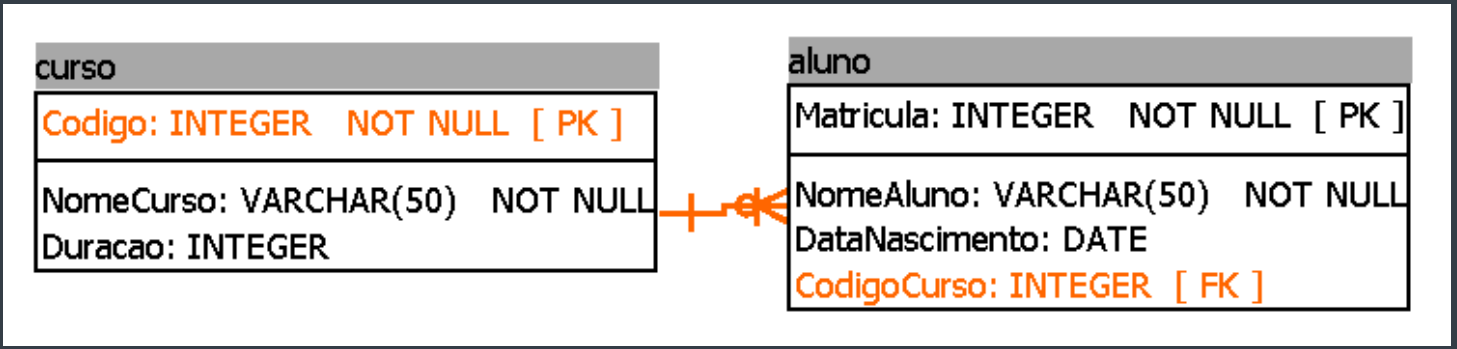



```
CREATE TABLE curso (
Codigo INTEGER PRIMARY KEY,
NomeCurso VARCHAR(50) NOT NULL,
Duracao INTEGER
);

CREATE TABLE aluno (
Matricula INTEGER PRIMARY KEY,
NomeAluno VARCHAR(50) NOT NULL,
DataNascimento DATE,
CodigoCurso INTEGER,
FOREIGN KEY (CodigoCurso)
REFERENCES curso(Codigo)
ON UPDATE NO ACTION  ON DELETE NO ACTION
);
```

- Para cada aluno que for cadastrado, poderá ter um código de curso cadastrado na tabela CURSO.
- O comando REFERENCES está referenciando o campo CodigoCurso da tabela ALUNO, que é chave estrangeira, que aponta para o campo Código da tabela CURSO, que é a chave primária de CURSO.
- A partir da criação desse relacionamento, ao tentarmos cadastrar um aluno com um código de curso que não esteja cadastrado na tabela CURSO, haverá restrição.

# Adicionando uma chave estrangeira




```sql
CREATE TABLE curso (
Codigo INTEGER PRIMARY KEY,
NomeCurso VARCHAR(50) NOT NULL,
Duracao INTEGER
);
```

```sql
CREATE TABLE aluno (
Matricula INTEGER PRIMARY KEY,
NomeAluno VARCHAR(50) NOT NULL,
DataNascimento DATE,
CodigoCurso INTEGER
);
```

-- Altera a tabela aluno para incluir a restrição de chave estrangeira referenciando a chave primária da tabela curso.

```sql
ALTER TABLE aluno ADD CONSTRAINT fk_codcurso
FOREIGN KEY (CodigoCurso) REFERENCES universidade.curso(Codigo)
ON UPDATE NO ACTION ON DELETE NO ACTION;
```