Bases de Dados

Universidade do Porto

Faculdade de Engenharia

FEUP

- INTRODUÇÃO
- MODELOS CONCEPTUAIS
  - Diagrama de Classes UML
  - Modelo Entidade-Associação (E-A)
- MODELO RELACIONAL
- **LINGUAGEM DE DEFINIÇÃO DE DADOS**
- INTERROGAÇÃO DE DADOS
  - Álgebra relacional
  - Linguagem de Manipulação de Dados (LMD)

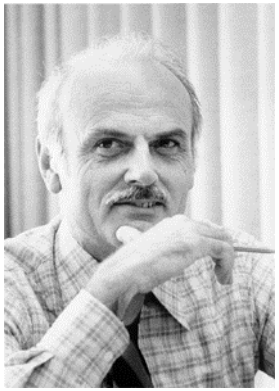
Observação: baseado em slides desenvolvidos pelo Prof. Jeffrey D. Ullman

Índice

- História da SQL.
- Linguagem de definição de dados
- CREATE TABLE
- Restrições
  - Chaves
  - Chaves externas
  - Restrições de valor
  - Restrições entre atributos
  - Asserções
- Modelação UML das restrições
- Gatilhos
- Instruções da LDD SQL

História da SQL

1. Publicação do artigo "A Relational Model of Data for Large Shared Data Banks“ por Edgar F. Codd na revista *Communications of the ACM*, em Junho de 1970.



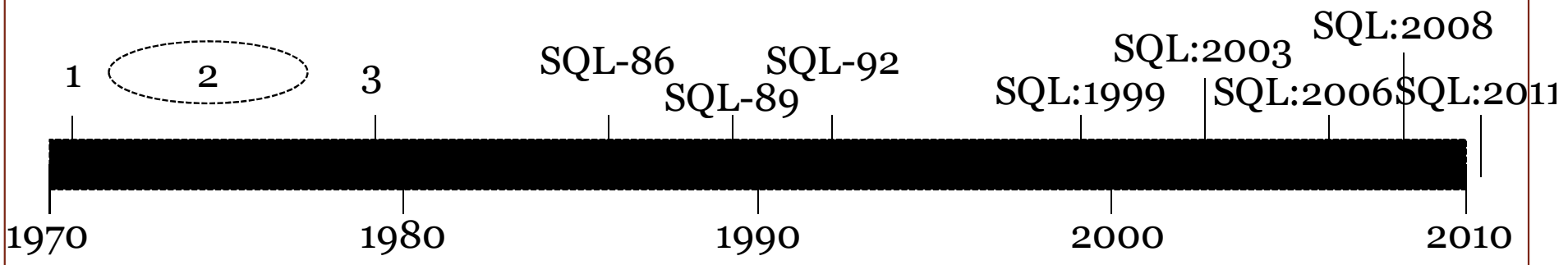
Edgar F. Codd

2. Desenvolvimento da linguagem SEQUEL, mais tarde designada SQL, na IBM, por Donald D. Chamberlin e Raymond F. Boyce.

3. Aparecimento da primeira versão comercial por parte da *Relational Software* (agora Oracle), em 1979.

Donald D. Chamberlin

Raymond F. Boyce



Linguagem de Definição de Dados

- A linguagem SQL (*Structured Query Language*) tem duas componentes:
  - Linguagem de Definição de Dados (LDD);
  - Linguagem de Manipulação de Dados (LMD).
- A Linguagem de Definição de Dados (LDD) é uma linguagem de programação para definição de estruturas de dados.
- A Linguagem de Manipulação de Dados (LMD) é uma linguagem de programação para manipulação de dados.
- Vamos estudar agora a LDD SQL.
- Mais tarde estudaremos a LMD SQL.

Linguagem de Definição de Dados

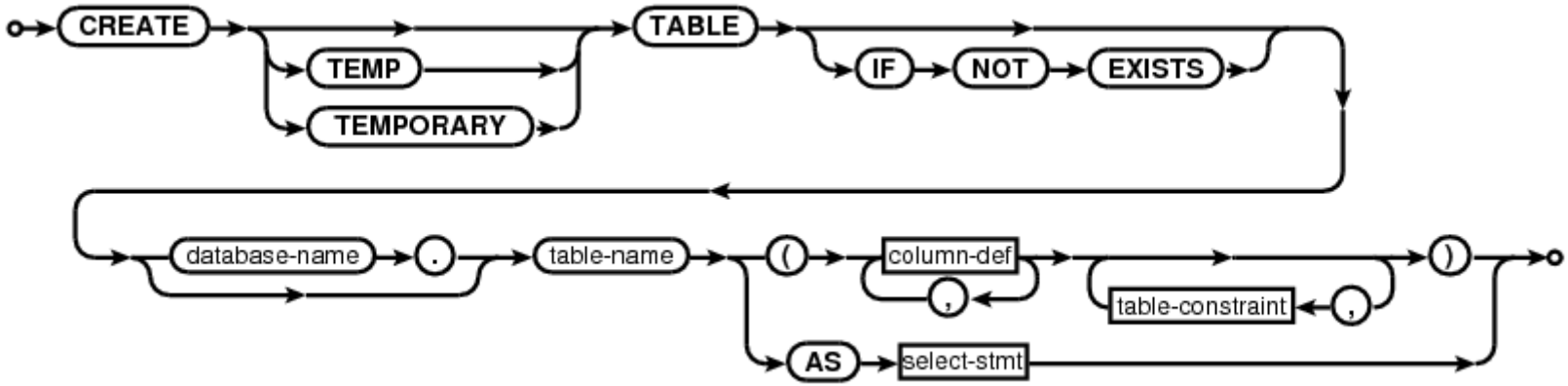
A LDD permite:

- Criar, alterar e remover estruturas de dados da base de dados
- Definir a política de controlo de acessos
- Optimizar a base de dados com vistas, índices, …
- Desenvolver funções e procedimentos através de uma linguagem procedimental própria (PSM: Persistent Stored Modules)
- Definir a política de cópias de degurança e de recuperação
- Definir opções de administração a um nível mais baixo (armazenamento em disco, …)
- Auditar a base de dados
- E mais algumas coisas …

CREATE TABLE

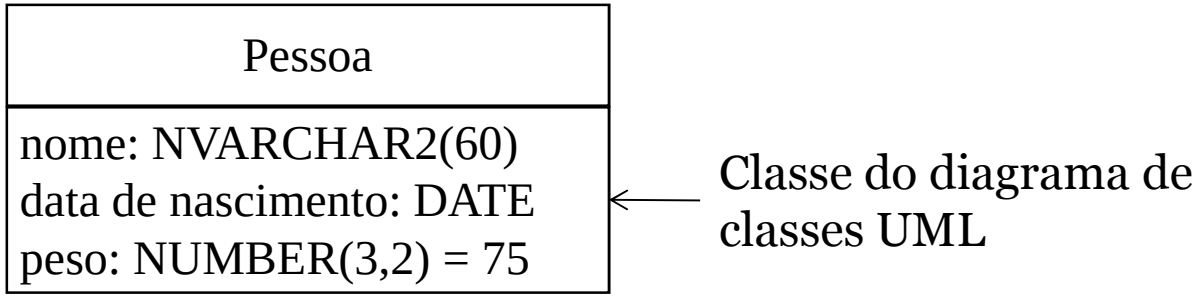
Create relational tables using SQLite:

http://www.sqlite.org/lang_createtable.html



CREATE TABLE

Exemplo




```
CREATE TABLE Pessoa (
    idPessoa          NUMBER PRIMARY KEY,
    nome              NVARCHAR2(20),
    dataNascimento    DATE,
    peso              NUMBER DEFAULT 75);
```

CREATE TABLE

Tipos de dados mais comuns (ORACLE)

NVARCHAR2(*size [BYTE | CHAR]): Variable-length national character string having maximum length size* bytes or characters. Maximum *size is 4000 bytes or characters,* and minimum is 1 byte or 1 character.

CHAR [(*size [BYTE | CHAR])*]: *Fixed-length character data of length size bytes or characters.* Maximum *size is 2000 bytes or characters. Default and* minimum *size is 1 byte.*

NUMBER[(*p [, s])*]: *Number having precision p and scale s. The precision p can* range from 1 to 38. The scale *s can range from -84 to 127. Both* precision and scale are in decimal digits. A NUMBER value requires from 1 to 22 bytes.

DATE: Valid date range from January 1, 4712 BC, to December 31, 9999 AD. The size is fixed at 7 bytes. [...] This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

TIMESTAMP [(fractional_seconds_precision)]: Year, month, and day values of date, as well as hour, minute, and second values of time, where fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values of fractional_seconds_precision are 0 to 9. [...] This data type contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. [...]

RAW(size): Raw binary data of length size bytes. Maximum size is 2000 bytes. [...]

LONG RAW: Raw binary data of variable length up to 2 gigabytes.

BLOB: A binary large object. Maximum size is (4 gigabytes - 1) *(database block size).

NOTA: Estes são alguns dos tipos de dados ORACLE mais comuns. Mas há mais:
http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/datatype.htm

CREATE TABLE

Tipos de dados (SQLite)

- O SQLite é compatível com os tipos de dados usados pelos Sistemas Gestores de Bases de Dados mais comuns. Ex.: Oracle, MySQL, etc.
- No entanto o SQLite usa o conceito de **Afinidade (Affinity)**.
- Converte os tipos de dados mais comuns numa das cinco afinidades existentes de acordo com o quadro ao lado e pela ordem indicada.

Example Typenames From The CREATE TABLE Statement or CAST Expression	Resulting Affinity	Rule Used To Determine Affinity
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	1
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	2
BLOB *no datatype specified*	NONE	3
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	4
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	5

Restrições e gatilhos

- Uma *restrição* é uma condição que é imposta aos dados.
  - Exemplo: chaves.
- *Gatilhos define uma acção a executar* quando ocorre um determinado tipo de evento.
  - Exemplo: actualizar o número de pontos que um piloto tem no campeonato de F1 depois de inserir a classificação numa corrida.
  - São mais fáceis de implementar do que restrições complexas.

Restrições

Tipos de restrições

- *Chaves*.
- *Chaves-externas*, ou integridade referencial.
- Restrições de *valor*.
  - Limites ao valor de um certo atributo (ex.: preço > 0).
  - NOT NULL
- Restrições *entre atributos*.
  - Associações entre diferentes atributos (ex.: dataFim > dataInicio).
- *Asserções*: qualquer expressão SQL booleana.

Chaves

Chaves de um só atributo

- Pôr PRIMARY KEY ou UNIQUE depois do tipo de dados na declaração do atributo.
  - Só pode haver uma chave primária (primary key) por tabela, mas podem haver várias chaves (unique).
  - Os atributos da chave primária não podem ter valores NULL, mas alguns dos atributos da restrição UNIQUE podem ser NULL desde que a combinação desses atributos seja única.
- Exemplo:

```
CREATE TABLE Cervejas (
        nome CHAR(20) PRIMARY KEY,
        empr CHAR(20) );
```

Chaves

Chaves multi-atributo

- Bar e cerveja juntos são a chave primária de Vende:

```
CREATE TABLE Vende (
        bar         CHAR(20),
        cerveja     NVARCHAR(20),
        preco       NUMBER,
PRIMARY KEY (bar, cerveja)
);
```

Chaves externas

- Valores que aparecem em atributos de uma relação devem também aparecer em certos atributos de outra relação.
- Exemplo: em Vende(bar, cerveja, preço), espera-se que o valor de cerveja também apareça em Cervejas.nome.

Chaves externas

- Utilizar a palavra chave REFERENCES em qualquer uma das seguintes situações:
  1. Depois de um atributo (a seguir a uma chave externa com um só atributo).
  2. Como elemento do esquema:

  FOREIGN KEY (<lista de atributos>)

  REFERENCES <relação> (<atributos>)
- Atributos referenciados devem ser declarados como PRIMARY KEY ou UNIQUE.

Chaves externas

Exemplos

```
CREATE TABLE Cervejas (
  nome      CHAR(20) PRIMARY KEY,
  empr      CHAR(20) );

CREATE TABLE Vende (
  bar  CHAR(20),
  cerveja CHAR(20) REFERENCES Cervejas(nome),
  preço     NUMBER);
```

Chaves externas

Exemplos

```
CREATE TABLE Cervejas (
  nome    CHAR(20) PRIMARY KEY,
  empr    CHAR(20) );
CREATE TABLE Vende (
  bar     CHAR(20),
  cerveja CHAR(20),
  preço   NUMBER,
  FOREIGN KEY(cerveja) REFERENCES
  Cervejas(nome));
```

Chaves externas

Restrições de chave externa

- Se houver uma restrição de chave-externa da relação *R* para a relação *S*, podem acontecer 2 violações:
  1. Um INSERT ou UPDATE de *R* introduz valores que não existem em *S*.
  2. Um DELETE ou UPDATE de S faz com que alguns registos de *R* "fiquem pendurados".

Chaves externas

Restrições de chave externa

- Exemplo: vamos supor que R = Vende, S = Cervejas.
- Um INSERT ou UPDATE de Vende que introduza uma cerveja não existente deve ser rejeitado.
- Um DELETE ou UPDATE de Cervejas que remove uma cerveja existente nos registos de Vende pode ser resolvido de 3 maneiras diferentes (ver slide seguinte).

Chaves externas

Restrições de chave externa

Se ao apagar ou alterar o atributo nome da tabela Cervejas exstirem registos na tabela Vende com referência aos registos que se quer apagar/alterar:

1. *Por defeito* : Rejeita a modificação.
2. *Em cascata*: Faz as mesmas alterações em Vende.
   - Cerveja removida: apagar registo(s) de Vende.
   - Cerveja actualizada: alterar valor de Vende.
3. *Atribuir NULL* : Alterar a cerveja para NULL.

Chaves externas

Restrições de chave externa: em cascata

- Apaga o registo Abadia da tabela Cervejas:
  - Depois apaga todos os registos de Vende com cerveja = 'Abadia'.
- Altera o registo Abadia da tabela Cervejas substituindo 'Abadia' por 'Super Bock Abadia':
  - Depois altera os registos de Vende com cerveja = 'Abadia' para cerveja = 'Super Bock Abadia'.

Chaves externas

Restrições de chave externa: atribuir NULL

- Apagar o registo Abadia de Cervejas:
  - Mudar todos os registos de Vende que tenham cerveja = 'Abadia' para cerveja = NULL.
- Alterar o registo Abadia de Cervejas substituindo 'Abadia' por 'Super Bock Abadia':
  - A mesma alteração que para o apagar.

Chaves externas

Restrições de chave externa: escolha de política

- Quando declaramos uma chave externa, podemos escolher como políticas SET NULL ou CASCADE independentemente para DELETEs e UPDATEs.
- A seguir à declaração de chave externa pôr:

ON [UPDATE, DELETE][SET NULL | CASCADE]

- Podem ser usadas estas duas cláusulas.
- Caso contrário, é usada a acção de rejeição por defeito.

Chaves externas

Restrições de chave externa: escolha de política

Exemplo :

```
CREATE TABLE Vende(
  bar      CHAR(20),
  cerveja  CHAR(20),
  preco    NUMBER,
  FOREIGN KEY(cerveja)
      REFERENCES Cervejas(nome)
      ON DELETE SET NULL
      ON UPDATE CASCADE
);
```

Restrições de valor

- Restrições ao valor de um dado atributo.
- Acrescentar CHECK(<condição>) à declaração do atributo.
- A condição pode usar o nome do atributo, mas o nome de qualquer outra relação ou atributo de outra relação tem de estar numa subconsulta (*subquery*).

Restrições de valor

Exemplo

```
CREATE TABLE Vende (
  bar   CHAR(20) NOT NULL,
  cerveja CHAR(20)
      CHECK (cerveja IN
            (SELECT nome FROM Cerveja)),
  preco   NUMBER CHECK (preco <= 5.00));
```

Sub-consultas em restrições do tipo CHECK não são possíveis em SQLite.

Restrições de valor

Momento de realização dos testes

- Os testes aos atributos só são realizados quando o valor do atributo é inserido ou actualizado.
  - Exemplo: `CHECK (preco <= 5.00)` testa todos os preços novos e rejeita a modificação (para esse registo) se o preço for superior a 5 €.
  - Exemplo: `CHECK (cerveja IN (SELECT nome FROM Cervejas))` não é testado se a cerveja for apagada de Cervejas (ao contrário das chaves-externas).

Restrições entre atributos

- CHECK (<condição>) pode ser acrescentado.
- A condição pode-se referir a qualquer atributo da relação.
  - Mas quaisquer outros atributos ou relações requerem uma subconsulta.
- Os testes só são efectuados aquando das inserções ou actualizações.

Restrições entre atributos

Exemplo

- Só o bar Mercedes vende cerveja por mais de 5 €:

```
CREATE TABLE Vende(
    bar      CHAR(20),
    cerveja  CHAR(20),
    preco    NUMBER,
    CHECK (bar = 'Mercedes' OR
                 preco <= 5.00));
```

Asserções

- São elementos das bases de dados, da mesma forma que as relações o são.
- Define-se da seguinte forma:

  CREATE ASSERTION <nome>

  CHECK (<condição>);
- A condição pode-se referir a qualquer relação ou atributo da base de dados.
- No entanto a grande maioria dos Sistemas Gestores de Bases de Dados comerciais não têm esta funcionalidade implementada. O SQLite também não tem.

Asserções

Exemplo

- Em Vende(bar, cerveja, preco), nenhum bar pode cobrar em média mais de 5 €.

```
CREATE ASSERTION NoBaresCaros CHECK (
   NOT EXISTS (
        SELECT bar FROM Sells
        GROUP BY bar
        HAVING AVG(preco) > 5
   ));
```

Bares com preços médios acima de 5 €

Asserções

Exemplo

- Em Clientes(nome, morada, telefone) e Bares(nome, morada, licenca), não pode haver mais bares do que clientes.

```
CREATE ASSERTION PoucosBares CHECK (
  (SELECT COUNT(*) FROM Bares) <=
  (SELECT COUNT(*) FROM Clientes)
);
```

Asserções

Momento de realização dos testes

- Em princípio, temos de testar todas as asserções depois de qualquer modificação a uma relação da base de dados.
- Um sistema inteligente deve ter em conta que só algumas mudanças podem originar violações das asserções.
  - Exemplo: Nenhuma alteração a Cervejas afecta PoucosBares. Nem nehuma inserção a Clientes.

Relembrar

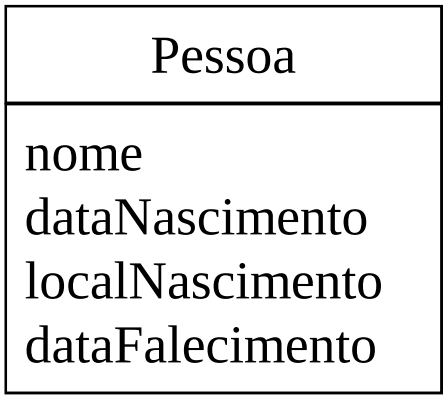
UML: Diagrama de Classes

Restrições

- Uma restrição especifica uma condição que tem de se verificar no estado do sistema (objectos e ligações)
- Uma restrição é indicada por uma expressão ou texto entre chavetas ou por uma nota posicionada junto aos elementos a que diz respeito, ou a eles ligada por linhas a traço interrompido (sem setas, para não confundir com relação de dependência)
- Podem ser formalizadas em UML com a OCL - "Object Constraint Language"
- Também podem ser formalizadas (como invariantes) numa linguagem de especificação formal como VDM++

Modelação UML das restrições

Classes




```
CREATE TABLE Pessoa (
idPessoa         NUMBER PRIMARY KEY,
nome             NVARCHAR2(20),
dataNascimento   DATE,
localNascimento NUMBER,
dataFalecimento DATE,
UNIQUE (nome, dataNascimento, localNascimento),
CHECK (dataFalecimento > dataNascimento)
);
```

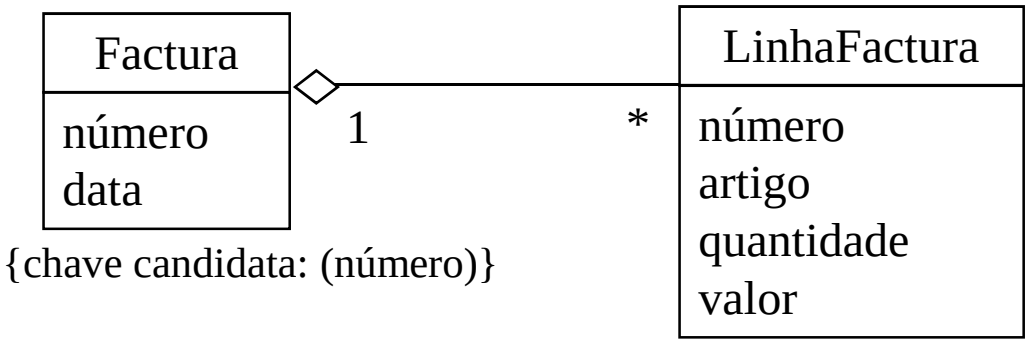
{chave candidata: (nome, dataNascimento, localNascimento)}

{dataFalecimento > dataNascimento}

Modelação UML das restrições

Chaves candidatas e chaves externas

```
CREATE TABLE Factura (
numero NUMBER PRIMARY KEY,
data     DATE);
```



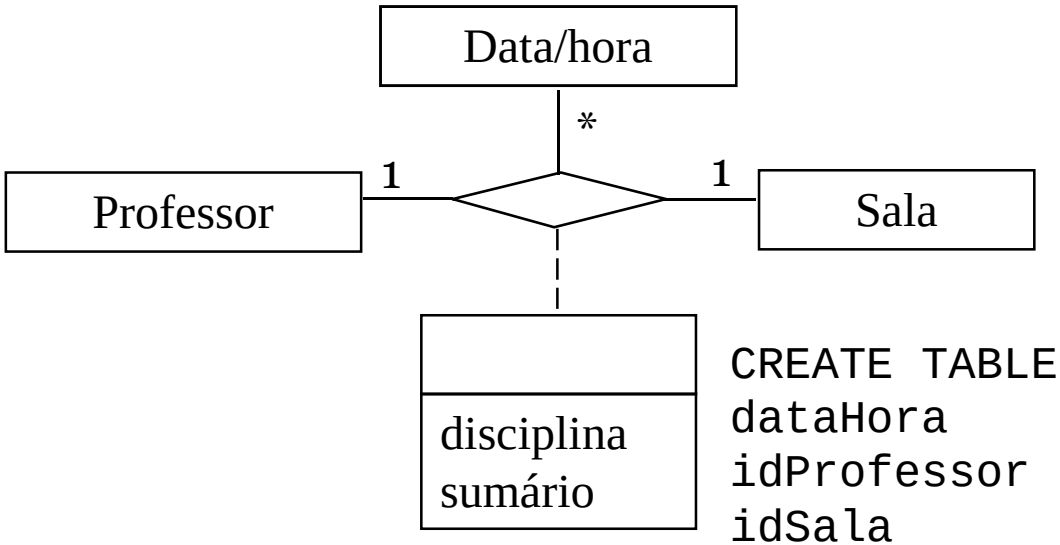
{chave candidata: (número)}

{chave candidata: (factura, número)}

```
CREATE TABLE LinhaFactura (
idLinhaFactura NUMBER PRIMARY KEY,
numero  NUMBER,
artigo  NVARCHAR2(20),
quantidade NUMBER,
valor NUMBER,
numFactura NUMBER REFERENCES Factura(numero),
UNIQUE (numFactura, numero));
```

Modelação UML das restrições

Associações ternárias




```
CREATE TABLE aula (
dataHora       TIMESTAMP,
idProfessor    NUMBER,
idSala         NUMBER,
disciplina     NVARCHAR2(20),
sumario        NVARCHAR2(100),
PRIMARY KEY (dataHora, idProfessor),
UNIQUE (dataHora, idSala)
);
```

Gatilhos

Motivação

- As asserções são poderosas, mas os SGBD não costumam ser capazes de dizer quando é que o teste deve ser feito.
- Testes a atributos e a tuplos são efetuados em momentos conhecidos, mas não são poderosos.
- Os gatilhos deixam o responsável pela BD decidir quando é que deve ser efetuado o teste a uma dada condição.

Gatilhos

Regras acontecimento-condição-acção

- Um gatilho pode ser visto como uma regra *acontecimento-condição-acção*.
- *Acontecimento* : são tipos de modificações que acontecem numa BD, ex.: inserir em Vende.
- *Condição* : Qualquer expressão SQL cujo resultado seja do tipo Booleano.
- *Acção* : Qualquer comando SQL.

Gatilhos

Exemplo

- Em vez de usar uma restrição de chave-externa e rejeitar inserções em Vende(bar, cerveja, preco) com cervejas desconhecidas, um gatilho pode adicionar a cerveja a Cervejas, atribuindo o valor NULL a empresa.

Gatilhos

Exemplo

Este exemplo não funciona em SQLite.

```
CREATE TRIGGER GatCerveja
    AFTER INSERT ON Vende
    REFERENCING NEW ROW AS NovoTuplo
    FOR EACH ROW
    WHEN (NovoTuplo.cerveja NOT IN
          (SELECT nome FROM Cervejas))
    INSERT INTO Cervejas(nome)
          VALUES(NovoTuplo.cerveja);
```

O acontecimento

A condição

A acção

Gatilhos

CREATE TRIGGER

- CREATE TRIGGER <nome>
- Ou:

CREATE OR REPLACE TRIGGER <nome>

  - Útil no caso de existir um gatilho com esse nome e se queira modificá-lo.

Gatilhos

o acontecimento

- Em vez de AFTER pode ser BEFORE.
  - Ou, INSTEAD OF, se a relação for uma vista.
    - Uma forma inteligente de executar alterações a vistas: substitui a ação por modificações adequadas nas tabelas que participam na vista.
- INSERT pode ser DELETE ou UPDATE.
  - E UPDATE pode ser UPDATE OF . . . ON um dado atributo.

Gatilhos

FOR EACH ROW

- Os gatilhos podem ser definidos ao "nível-de-linha" ou ao "nível-de-instrução".
- FOR EACH ROW indica nível-de-linha; a sua omissão indica nível-de-instrução.
- *Gatilhos ao nível-de-linha*: executa uma vez por cada linha modificada.
- *Gatilhos ao nível-de-instrução*: executa uma vez para uma instrução SQL, independentemente do número de linhas modificadas.

Gatilhos

REFERENCING

- As instruções INSERT implicam um novo registo (para o nível-de-linha) ou uma nova "tabela" (para o nível-de-instrução).
  - A "tabela" é o conjunto de novas linhas.
- DELETE implica um registo ou "tabela" velhos.
- UPDATE implica ambos (novo e velho).
- A referência é feita da seguinte forma:

NEW ou OLD antes do repectivo atributo.

Gatilhos

A condição

- Qualquer expressão SQL com resultado Booleano.
- Avaliação efectuada na base de dados de acordo com o seu estado antes ou depois do evento do gatilho, dependendo da utilização de BEFORE ou AFTER.
  - Mas sempre antes das alterações serem efectuadas.
- Acede ao novo/velho registo/tabela através dos nomes NEW e OLD.

Gatilhos

A acção

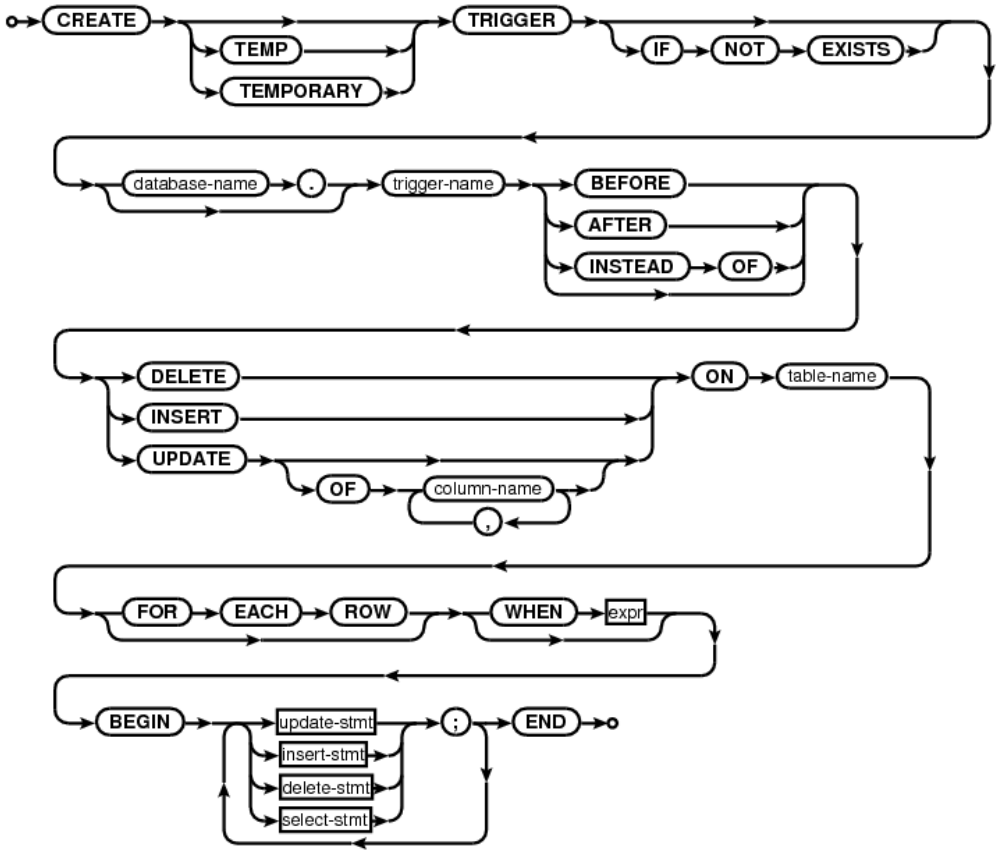
- Pode existir mais do que uma instrução SQL statement na acção.
  - Limitada por BEGIN . . . END caso existam várias.
- Mas como as consultas (SELECT) não fazem sentido numa acção, estamos limitados a INSERT, UPDATE, e/ou DELETE.

Gatilhos

Usando SQLite

A sintaxe dos TRIGGERS em SQLite tem algumas diferenças em relação à sintaxe dos standars.

Observe essas diferenças no exemplo que se segue.



Gatilhos

Exemplo

- Usando Vende(bar, cerveja, preco) e a relação unária BaresCaros(bar), manter a lista dos bares que aumentam o preço de alguma cerveja em mais de 1 €.

Gatilhos

Exemplo

```
CREATE TRIGGER GatilhoPreco
    AFTER UPDATE OF preco ON Vende
    FOR EACH ROW
    WHEN(NEW.preco > OLD.preco + 1.00)
    BEGIN
        INSERT INTO BaresCaros
            VALUES(NEW.bar);
    END;
```

O evento: apenas mudanças de preço

Temos de considerar amudança de cada preço

Condição: acréscimo de preço>1€

Quando a mudança de preço é suficientemente grande, acrescentar o bar a BaresCaros

Instruções da LDD SQL

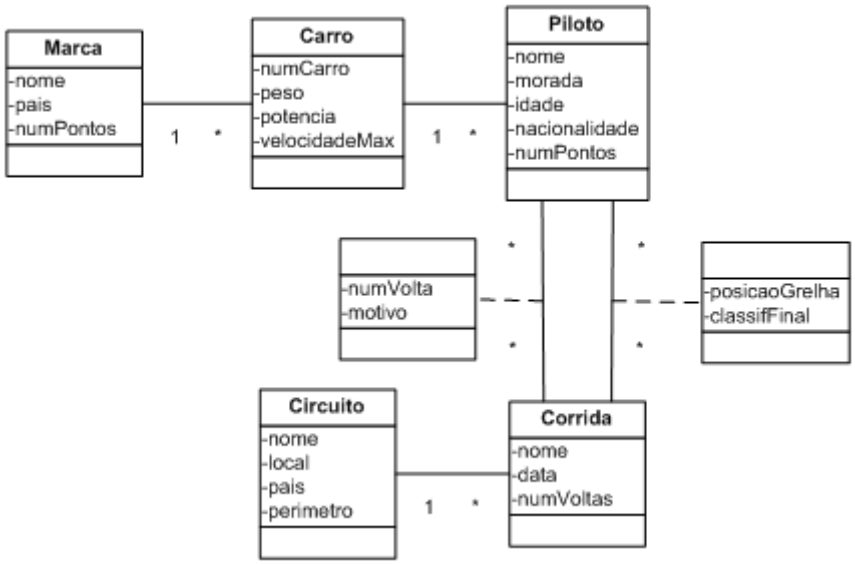
Criação da BD

```
CREATE TABLE IF NOT EXISTS Cervejas (
          idCerveja INTEGER AUTOINCREMENT,
          nome      CHAR(20),
          empr      CHAR(20),
CONSTRAINT pk_Cervejas PRIMARY KEY (idCerveja));
CREATE TABLE IF NOT EXISTS Vende (
          idVenda   INTEGER AUTOINCREMENT,
          bar       CHAR(20),
          idCerveja NUMBER,
          preco     NUMBER,
CONSTRAINT pk_Vende PRIMARY KEY (idVenda),
CONSTRAINT k1_Vende UNIQUE (bar,idCerveja),
CONSTRAINT fk_Vende_cerveja FOREIGN KEY (idCerveja) REFERENCES Cervejas (idCerveja));
```

Inserção dos dados

```
INSERT INTO Cervejas (nome, empr)
          VALUES ('Abadia', 'UNICER');
INSERT INTO Vende (bar, idCerveja, preco)
          VALUES ('Pipa velha', 1, 2.5);
```

Exercício


Marca
-nome
-pais
-numPontos
1
*
Carro
-numCarro
-peso
-potencia
-velocidadeMax
1
*
Piloto
-nome
-morada
-idade
-nacionalidade
-numPontos
*
*
-numVolta
-motivo
*
*
-posicaoGrelha
-classifFinal
Circuito
-nome
-local
-pais
-perimetro
1
*
Corrida
-nome
-data
-numVoltas