MBA EM **ENGENHARIA DE SOFTWARE**

Bancos de Dados Relacionais II

Profª. Drª. Kelly Rosa Braghetto

MBA USP ESALQ

A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.

MBA USP ESALQ

Linguagem SQL
Definição de Esquemas

Slides baseados no livro *Sistemas de Bancos de Dados* (7ª edição),
de Elmasri e Navathe. Pearson, 2010 – Capítulos 6

SQL – Structured Query Language

- Foi criada em 1976
- Possui comandos para:
  - Definição do esquema do BD
  - Consultas ao BD
  - Modificação do BD (inserção, remoção, alteração)
  - Definição de visões

Padrões da SQL

- ANSI 1 SQL (1986)
- SQL-89 – inclusão de restrições de integridade
- SQL-92 (ou SQL2) – grande atualização da versão anterior
- SQL-99 (inicialmente chamada de SQL3) – inclusão de
  - expressões regulares, consultas recursivas, gatilhos, comandos
  - para controle de fluxo,
  - funcionalidades relacionadas à orientação a objetos, etc.
- SQL-2003, SQL-2006, SQL-2008, SQL-2016, SQL-2019
  - XML, JSON
  - BDs temporais, vetores multidimensionais
  - Reconhecimento de padrões em linhas, etc.

Os SGBDs implementam a ANSI SQL, (partes de) outras versões da SQL e extensões próprias também.

Declarações Simples de Tabelas

```
CREATE TABLE nome_tabela
        [({nome_coluna   tipo_dados   [{restricao_coluna}]}
          [{restricao_tabela }]
        )]
```

Exemplo:

```
CREATE TABLE FUNCIONARIO (
   Cpf              CHAR(11) PRIMARY KEY,
   Nome             VARCHAR(50) NOT NULL,
   Data_nascimento  DATE,
   Sexo             CHAR(1),
   Salario          DECIMAL(10,2) CHECK (Salario > 1045)
);
```

Tipos de Dados de Atributos

- **Números inteiros**
  - **INTEGER** ou **INT** (geralmente ocupa 4 bytes)
  - **SMALLINT** – ocupa geralmente a metade da quantidade de bytes usada por um INTEGER)
- **Números reais**
  - **FLOAT** ou **REAL** (geralmente ocupa 4 bytes)
  - **DOUBLE PRECISION** (geralmente ocupa 8 bytes)
  - **DECIMAL(i,j)** ou **DEC(i,j)** ou **NUMERIC(i,j)** – onde i é a precisão e indica o total de dígitos decimais, e j é a escala e indica o número de dígitos após o ponto decimal.

    ⇒ O valor padrão para j é 0, mas para i depende do SGBD

Tipos de Dados de Atributos

- **Caracteres**
  - **CHARACTER(n)** ou **CHAR(n)** – onde n é o número de caracteres, que define o tamanho fixo da cadeia
  - **CHAR VARYING(n)** ou **VARCHAR(n)** – onde n é o número máximo de caracteres da cadeia
  - **CHARACTER LARGE OBJECT** ou **CLOB** – para grandes cadeias de caracteres de tamanho variável (como documentos)
- **Observações**
  - Cadeias de caracteres literais devem ser delimitadas por aspas simples (apóstrofos), como em 'MBA USP/Esalq'
  - Caracteres em SQL são case sensitive. Portanto, 'USP' = 'Usp'.
  - Mas as palavras reservadas da SQL são case insensitive, ou seja, podemos usar CREATE TABLE ou cREAte TAblE de forma indistinta.

Tipos de Dados de Atributos

- **Datas e horários**
  - **DATE** – exemplo: '2004-10-23' (formato que é sempre válido: YYYY-MM-DD)
  - **TIME** – exemplo: '22:45:17' (formato HH:MM:SS)
  - **TIMESTAMP** – incluem os campos DATE e TIME e mais posições para frações decimais de segundos. Exemplo: '2014-08-20 15:43:34.827022'
- **Observações**
  - Os tipos TIME e TIMESTAMP podem ter também um qualificador **WITH TIME ZONE**.
  - Ex. de valor para um atributo do tipo TIMESTAMP WITH TIME ZONE:

    '2014-08-20 15:43:34.827022-03'

Tipos de Dados de Atributos

- **Datas e horários (continuação)**
  - **INTERVAL** – especifica um valor usado para incrementar ou decrementar o valor absoluto de uma data, hora ou timestamp.

    Um intervalo é qualificado para ser **YEAR-MONTH**, **DAY-TIME** ou uma mistura dos dois. Exemplos:
    - INTERVAL '1-2' – intervalo de 1 ano e 2 meses
    - INTERVAL '3 4:05:06' – intervalo de 3 dias, 4 horas, 5 minutos e 6 segundos
    - INTERVAL '1-2 3 4:05:06' – os dois acima juntos
- **Observações**
  - Podemos considerar que os tipos para data e hora em SQL são essencialmente cadeias de caracteres com um formato especial.

Tipos de Dados de Atributos

- **Booleano**
  - **BOOLEAN** – admite os valores **TRUE**, **FALSE** ou **UNKNOWN** (para o espanto do George Boole!)
- **Observações**
  - O valor UNKNOWN pode resultar de operações de comparação envolvendo o valor NULL; voltaremos a falar disso mais adiante.

Tipos de Dados de Atributos

- **Cadeia de bits**
  - **BIT(n)** – cadeia de bits de tamanho fixo n
  - **BIT VARYING(n)** – cadeia de bits com tamanho máximo n
  - **BINARY LARGE OBJECT** ou **BLOB** – para grandes cadeias de bits de tamanho variável (como imagens)

Criação de Domínios

- É possível declarar um novo domínio e usar o seu nome como especificação para um atributo.

Exemplo:

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

CREATE TABLE FUNCIONARIO (
    Cpf                 TIPO_CPF PRIMARY KEY,
    Nome                VARCHAR(50) NOT NULL,
    Data_nascimento DATE,
    Sexo                CHAR(1),
    Salario             DECIMAL(10,2) CHECK (Salario > 1045)
);
```

Restrições e Valores Padrão para Colunas

- Restrição contra valores nulos – cláusula **NOT NULL**
  - Define que uma coluna não pode receber o valor NULL
  - Colunas da chave primária são implicitamente NOT NULL
- Valor padrão – cláusula **DEFAULT**
  - Define um valor que será atribuído à coluna em uma nova tupla sempre que o valor para essa coluna não for fornecido
  - Se uma coluna não for NOT NULL e não tiver valor padrão definido para ela, o NULL será usado como padrão
- Restrição de verificação – cláusula **CHECK**
  - Restringe os valores que uma coluna pode assumir

Restrições e Valores Padrão para Colunas

Exemplo:

```
CREATE TABLE FUNCIONARIO (
    Cpf                 TIPO_CPF PRIMARY KEY,
    Nome                VARCHAR(50) NOT NULL,
    Idade               INT CHECK (Idade >= 18 AND Idade <= 120),
    Sexo                CHAR(1),
    Salario             DECIMAL(10,2) NOT NULL CHECK
                          (Salario >= 1045 AND Salario <= 50000),
    Casado              BOOLEAN NOT NULL DEFAULT FALSE,
    Cpf_supervisor      TIPO_CPF DEFAULT '12345678901'
);
```

Restrições para Tabelas

- O **CHECK** pode ser de tabela e envolver várias colunas

O último CHECK implementa a restrição de que todo funcionário com 65 ou mais anos deve receber um salário maior ou igual a 10000.

Exemplo:

```
CREATE TABLE FUNCIONARIO (
    Cpf                 TIPO_CPF PRIMARY KEY,
    Nome                VARCHAR(50) NOT NULL,
    Idade               INT CHECK (Idade >= 18 AND Idade <= 120),
    Sexo                CHAR(1),
    Salario             DECIMAL(10,2) NOT NULL CHECK
                          (Salario >= 1045 AND Salario <= 50000),
    Casado              BOOLEAN NOT NULL DEFAULT FALSE,
    Cpf_supervisor      TIPO_CPF DEFAULT '12345678901',
    CHECK (idade < 65 OR Salario >= 10000)
 );
```

Restrições de Chave

- Restrição de chave primária – cláusula **PRIMARY KEY**
  - Especifica uma ou mais colunas que compõem a chave primária da tabela
  - Se a chave primária tiver uma única coluna, a cláusula pode aparecer como uma restrição de coluna na definição da tabela
  - Para chaves com uma ou mais colunas, usa-se uma cláusula de restrição de tabela
- Observação
  - Sobre uma chave primária, sempre é imposta uma restrição de NOT NULL (independentemente de haver ou não uma restrição explícita de NOT NULL definida sobre os atributos que compõem a chave primária).

Restrições de Chave Primária

Exemplo:

```
CREATE TABLE DEPARTAMENTO (
    Dnumero     INT PRIMARY KEY,
    Dnome       VARCHAR(30) NOT NULL
 );

CREATE TABLE LOCALIZACAO_DEP (
    Dnumero     INT NOT NULL,
    Dlocal      VARCHAR(15) NOT NULL,
    PRIMARY KEY(Dnumero,Dlocal)
 );
```

Restrições de Chave

- Restrição de chave única – cláusula **UNIQUE**
  - Especifica uma ou mais colunas que compõem uma chave única da tabela
  - Se a chave única tiver uma única coluna, a cláusula pode aparecer como uma restrição de coluna na definição da tabela
  - Para chaves únicas com uma ou mais colunas, usa-se uma cláusula de restrição de tabela
- Observação
  - Diferentemente do que ocorre com a chave primária, uma coluna da chave única pode receber valores NULL.

Restrições de Chave Única

Exemplo:

```
CREATE TABLE DEPARTAMENTO (
   Dnumero     INT PRIMARY KEY,
   Dnome       VARCHAR(30) NOT NULL UNIQUE
 );
```

ou (usando restrições de tabela):

```
CREATE TABLE DEPARTAMENTO (
   Dnumero     INT,
   Dnome       VARCHAR(30) NOT NULL,
   PRIMARY KEY(DNumero),
   UNIQUE(Dnome)
 );
```

Restrições de Integridade Referencial

- Chave estrangeira - Cláusula **FOREIGN KEY**
  - Pode ser definida como restrição de coluna ou de tabela

```
CREATE TABLE FUNCIONARIO (
   Cpf           CHAR(11) PRIMARY KEY,
   Nome          VARCHAR(50) NOT NULL,
   Salario       DECIMAL(10,2) );

CREATE TABLE DEPARTAMENTO (
   Dnumero         INT PRIMARY KEY,
   Dnome           VARCHAR(30) NOT NULL UNIQUE,
   Cpf_gerente     CHAR(11) NOT NULL REFERENCES FUNCIONARIO(Cpf) );

CREATE TABLE LOCALIZACAO_DEP (
   Dnumero     INT NOT NULL,
   Dlocal      VARCHAR(15) NOT NULL,
   PRIMARY KEY(Dnumero,Dlocal),
   FOREIGN KEY(Dnumero) REFERENCES DEPARTAMENTO(Dnumero) );
```

Restrições de Integridade Referencial

- Opções para o tratamento das violações de integridade referencial causadas por operações de inserção, remoção e alteração nas tabelas referenciadas:
  - **RESTRICT** (padrão) – rejeita a operação de atualização que causará uma violação
  - **SET NULL** – atribuirá NULL à chave estrangeira que ficar sem sua "referência"
  - **SET DEFAULT** – atribuirá um valor padrão à chave estrangeira que ficar sem sua "referência"
  - **CASCADE** – propaga a alteração feita na chave referenciada para as linhas que a referenciam

- As ações acima devem ser qualificadas com as cláusulas **ON DELETE** ou **ON UPDATE**

Restrições de Integridade Referencial + Opções de Tratamento

Exemplo:

```
CREATE TABLE FUNCIONARIO (
   Cpf           CHAR(11) PRIMARY KEY,
   Nome          VARCHAR(50) NOT NULL,
   Salario       DECIMAL(10,2) );

CREATE TABLE DEPARTAMENTO (
   Dnumero          INT PRIMARY KEY,
   Dnome            VARCHAR(30) NOT NULL UNIQUE,
   Cpf_gerente      CHAR(11) NOT NULL DEFAULT '12345678912'
                    REFERENCES FUNCIONARIO(Cpf)
                    ON DELETE SET DEFAULT ON UPDATE CASCADE );

CREATE TABLE LOCALIZACAO_DEP (
   Dnumero     INT NOT NULL,
   Dlocal      VARCHAR(15) NOT NULL,
   PRIMARY KEY(Dnumero,Dlocal),
   FOREIGN KEY(Dnumero) REFERENCES DEPARTAMENTO(Dnumero)
      ON DELETE CASCADE ON UPDATE CASCADE );
```

Nomeando Restrições

- Com a cláusula CONSTRAINT, é possível atribuir nomes às restrições.

Exemplo:

```
CREATE TABLE FUNCIONARIO (
   Nome          VARCHAR(50) NOT NULL,
   Cpf           CHAR(11),
   Salario       DECIMAL(10,2),
   CONSTRAINT ChP_Func PRIMARY KEY(Cpf) );

CREATE TABLE DEPARTAMENTO (
   Dnumero         INT,
   Dnome           VARCHAR(30) NOT NULL,
   Cpf_gerente     CHAR(11) NOT NULL DEFAULT '12345678901',
   CONSTRAINT ChP_DepNum PRIMARY KEY(Dnumero),
   CONSTRAINT ChS_DepNome UNIQUE(Dnome),
   CONSTRAINT ChE_DepGegerente
      FOREIGN KEY(Cpf_gerente) REFERENCES FUNCIONARIO(Cpf)
        ON DELETE SET DEFAULT  ON UPDATE CASCADE  );
```

Remoção de Tabelas

DROP TABLE nome_tabela [CASCADE | RESTRICT];

- Com a opção **CASCADE** – remove também os objetos que dependem da tabela removida (ex.: views, restrições de chave estrangeira, índices).
- Com a opção **RESTRICT** (padrão) – impede que a tabela seja removida caso haja no BD objetos que dependam dela.

Exemplo:

```
DROP TABLE FUNCIONARIO;
```

Inserção de Dados – Comando Básico

INSERT INTO nome_tabela [(coluna1, coluna2, …)] VALUES (valor1,valor2,...);

- Insere uma linha na tabela nome_tabela
- Quando a ordem das colunas não é especificada no comando, os valores são atribuídos de acordo com a ordem em que as colunas foram criadas na tabela
- Se os valores passados para o comando não satisfazem as restrições definidas sobre a tabela, a linha não é inserida no BD

Inserção de Dados – Comando Básico

Exemplo:

```
INSERT INTO FUNCIONARIO(Nome, Cpf, salario)
                    VALUES ('Fernando Pessoa', '12345678901', 4532.99);
INSERT INTO FUNCIONARIO(Nome, Cpf, salario)
                    VALUES ('Clarice Lispector', '12782392989', 1238.23);
INSERT INTO FUNCIONARIO(Nome, Cpf, salario)
                    VALUES ('Carlos Drummond', '11728237928', 23919.00);

INSERT INTO DEPARTAMENTO(Dnumero, Dnome, Cpf_gerente)
                    VALUES (1, 'Contabilidade', '11728237928');
INSERT INTO DEPARTAMENTO(Dnumero, Dnome)
                    VALUES (2, 'Recursos Humanos');

INSERT INTO LOCALIZACAO_DEP VALUES (1,'Rua do Matao');
INSERT INTO LOCALIZACAO_DEP VALUES (1,'Reitoria');
```

Alteração de Esquemas – Modificando Colunas

ALTER TABLE nome_tabela ADD [COLUMN] nome_coluna tipo_dado [restrições];

- Adiciona uma nova coluna em uma tabela

Exemplo:

```
ALTER TABLE Funcionario ADD COLUMN Sexo CHAR(1) DEFAULT 'F';
```

ALTER TABLE nome_tabela DROP [COLUMN] nome_coluna [RESTRICT | CASCADE];

- Remove uma coluna em uma tabela. Se a opção CASCADE for usada, todas as restrições e *views* que referenciam a coluna serão removidas.

Exemplo:

```
ALTER TABLE Departamento DROP COLUMN Cpf_gerente CASCADE;
```

Alteração de Esquemas – Modificando a Cláusula DEFAULT

ALTER TABLE nome_tabela ALTER [COLUMN] nome_coluna DROP DEFAULT;

- Remove a definição de valor padrão de uma coluna

Exemplo:

```
ALTER TABLE Funcionario ALTER COLUMN Sexo DROP DEFAULT;
```

ALTER TABLE nome_tabela ALTER [COLUMN] nome_coluna SET DEFAULT valor;

- Define um novo valor padrão para uma coluna

Exemplo:

```
ALTER TABLE Funcionario ALTER COLUMN Sexo SET DEFAULT 'M';
```

Linguagem SQL
Consultas

Os slides sobre Consultas em SQL são uma adaptação dos *slides* do prof. Jeffrey Ullman, da *Stanford University*

http://infolab.stanford.edu/~ullman/dscb/gslides.html

Esquema do BD Relacional para as Consultas de Exemplo

- As consultas em SQL que veremos a seguir serão baseadas no seguinte esquema relacional de BD:

Refrigerante(<u>nome</u>, fabricante)

Lanchonete(<u>nome</u>, endereco, cnpj)

Cliente(<u>nome</u>, endereco, telefone)

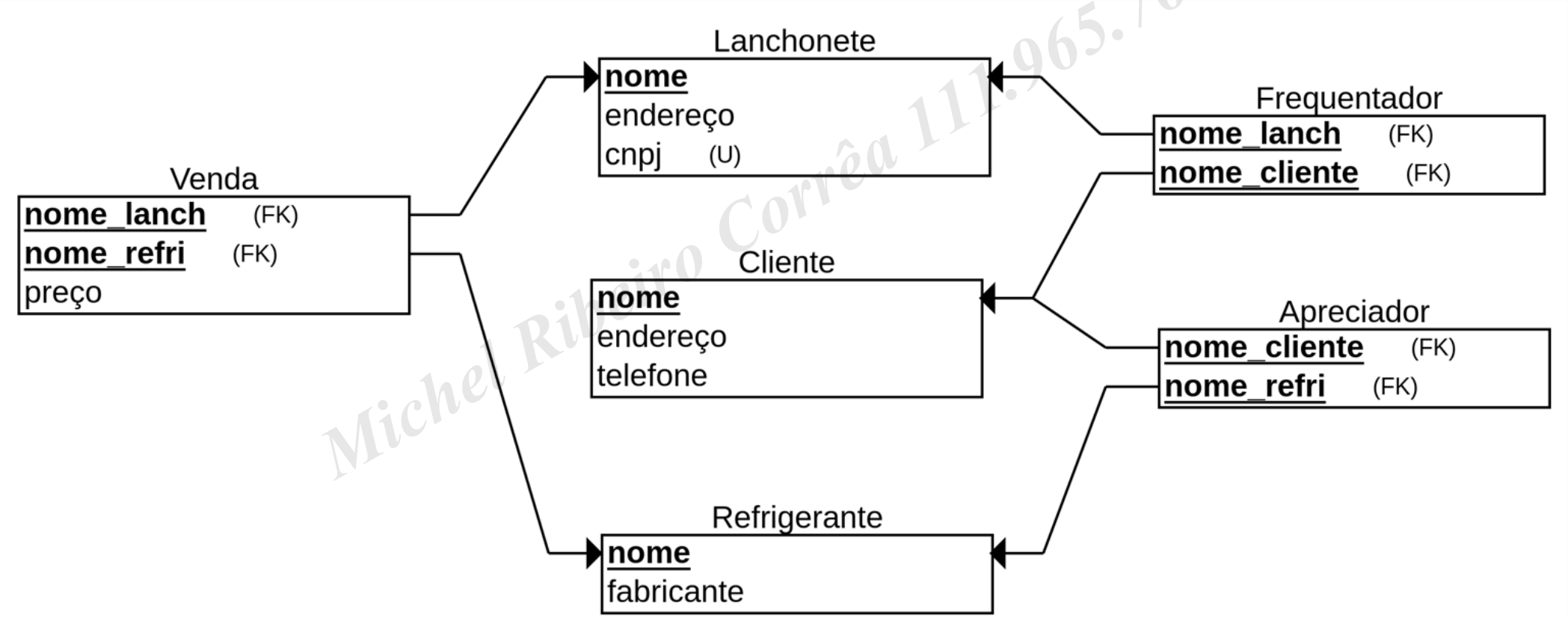
Apreciador(<u>nome_cliente</u>, <u>nome_refri</u>)

Venda(<u>nome_lanch</u>, <u>nome_refri</u>, preco)

Frequentador(<u>nome_cliente</u>, <u>nome_lanch</u>)

Esquema do BD Relacional para as Consultas de Exemplo

- As consultas em SQL que veremos a seguir serão baseadas no seguinte esquema relacional de BD:



Formato de uma Consulta em SQL

SELECT <lista de atributos ou expressões>

FROM <lista de relações>

WHERE <condição>

- **Exemplo:**

  Usando Refrigerante(nome, fabricante), quais são os refrigerantes feitos pelo fabricante *Cola-Coca*?

```
SELECT nome
FROM Refrigerante
WHERE fabricante = 'Cola-Coca';
```

Resultado da Consulta

- **Exemplo:**

  Usando Refrigerante(nome, fabricante), quais são os refrigerantes feitos pelo fabricante *Cola-Coca*?

```
SELECT nome
FROM Refrigerante
WHERE fabricante = 'Cola-Coca';
```

O resultado é uma relação com um único atributo, **nome**, e tuplas com o nome de cada refrigerante produzido pela Cola-Coca

nome
Fanfa
Kuaif
Sprife
...

Semântica Operacional

Processamento de uma consulta:

- Para cada tupla *t* na relação na cláusula FROM, repita:
  - Se o valor atual de *t* satisfaz a cláusula WHERE, então:
    - Compute os atributos ou expressões da cláusula SELECT usando os componentes de *t*
    - Inclua os valores computados como uma nova tupla na resposta da consulta

Semântica Operacional

```
SELECT nome
FROM Refrigerante
WHERE fabricante = 'Cola-
Coca';
```

A variável-tupla ***t*** percorre cada tupla da relação, uma por vez

Refrigerante

nome	fabricante
Fanfa	Cola-Coca
Uva	Lolly
Kuaif	Cola-Coca
Sprife	Cola-Coca

Para cada tupla t, verifica se ***t.fabricante*** é Cola-Coca

Se sim, inclui ***t.nome*** na relação de resultado da consulta

nome
Fanfa
Kuaif
Sprife

O * em Cláusulas SELECT

- Quando há apenas uma relação na cláusula FROM, um * na cláusula SELECT equivale a **"todos os atributos dessa relação"**

- **Exemplo:**

Usando Refrigerante(nome, fabricante)

```
SELECT  *
FROM Refrigerante
WHERE fabricante = 'Cola-Coca';
```

Refrigerante

nome	fabricante
Fanfa	Cola-Coca
Uva	Lolly
Kuaif	Cola-Coca
Sprife	Cola-Coca

Resultado

nome	fabricante
Fanfa	Cola-Coca
Kuaif	Cola-Coca
Sprife	Cola-Coca

Renomeando Atributos

- Para modificar os nomes dos atributos no resultado, use "**AS <novo nome>**" para renomear um atributo

- **Exemplo:**

Usando Refrigerante(nome, fabricante)

```
SELECT nome AS refri, fabricante
FROM Refrigerante
WHERE fabricante = 'Cola-Coca'
```

Refrigerante

nome	fabricante
Fanfa	Cola-Coca
Uva	Lolly
Kuaif	Cola-Coca
Sprife	Cola-Coca

Resultado

refri	fabricante
Fanfa	Cola-Coca
Kuaif	Cola-Coca
Sprife	Cola-Coca

Expressões em cláusulas SELECT

- Constantes, expressões matemáticas e funções podem aparecer como elementos na cláusula SELECT

- **Exemplo:**

Usando Venda(nome_lanch, nome_refri, preco)

```
SELECT nome_lanch, nome_refri,
       preco as preco_reais,
       preco * 29.43 AS preco_iene
FROM Venda
```

nome_lanch	nome_refri	preco_reais	preco_iene
Burgão	Fanfa	6.00	176.58
KiSabor	Sprife	5.00	147.15
...	...	...	...

Condições Complexas para a Cláusula WHERE

- Operadores booleanos AND, OR, NOT
- Comparações =, <>, <, >, <=, >=
  - E outros operadores que produzem valores booleanos como resultado

- **Exemplo:**

Usando Venda(nome_lanch, nome_refri, preco), encontre o preço cobrado na lanchonete Burgão pelo refrigerante Fanfa:

```
SELECT preco
FROM Venda
WHERE nome_lanch = 'Burgão' AND nome_refri = 'Fanfa';
```

Padrões em Cadeias de Caracteres (*Strings*)

- Uma condição pode comparar uma *string* com um padrão (~ expressão regular) usando:

  <Atributo> **LIKE** <padrão>

  ou

  <Atributo> **NOT LIKE** <padrão>

- *Padrão* é uma *string* contendo caracteres especiais:

  % (porcentagem) – "casa" com qualquer *string*

  _ (*underline*) – "casa" com **um** caractere qualquer

LIKE - Exemplo 1

- Usando Cliente(nome, endereço, telefone), encontre os clientes com DDD de São Paulo.

SELECT nome FROM Cliente

WHERE telefone **LIKE '(11)%'**

'(11) 91234-5678' LIKE '(11)%' → TRUE

'(15) 5432-5432' LIKE '(11)%' → FALSE

'(11) 3333-4567' LIKE '(11)%' → TRUE

'11 3333-4567' LIKE '(11)%' → FALSE

'(11)33334567' LIKE '(11)%' → TRUE

'(11)' LIKE '(11)%' → TRUE

LIKE - Exemplo 2

- Usando Cliente(nome, endereço, telefone), encontre os clientes cujo primeiro nome tem 3 letras:

```
SELECT nome
FROM Cliente
WHERE nome LIKE '___ %';
```

```
'Joe Singer' LIKE '___ %' → TRUE
'Anne Lee Jones' LIKE '___ %' → FALSE
'Tom'LIKE '___ %' → FALSE
```

Ordenação do Resultado de uma Consulta

- É possível ordenar as tuplas da relação resultante de uma consulta por meio da cláusula

ORDER BY <lista de atributos> [ASC | DESC]

- A ordenação ascendente (ASC) é a padrão

- **Exemplos:**

1) SELECT * FROM Cliente ORDER BY nome,telefone;

2) SELECT * FROM Cliente ORDER BY nome DESC;

Um Exemplo Interessante

- Considere a relação Venda a seguir:

nome_lanch	nome_refri	preco_reais
Burgão	Fanfa	NULL

Qual é o resultado da consulta abaixo?

```
SELECT nome_lanch FROM Venda
WHERE preco < 2.00 OR preco >= 2.00;
```

Comparando NULL com Outros Valores

- A lógica das condições em SQL é uma lógica ternária: **TRUE, FALSE, UNKNOWN**
- Comparar qualquer valor (incluindo o próprio NULL) com NULL resulta em **UNKNOWN**
- Uma tupla é incluída no conjunto resposta de uma consulta se e somente se a cláusula WHERE é **TRUE**
  - Se a cláusula é FALSE ou UNKNOWN, então a tupla **NÃO** entra na resposta

Um Exemplo Interessante

- Considere a relação Venda a seguir:

nome_lanch	nome_refri	preco_reais
Burgão	Fanfa	NULL


```
SELECT nome_lanch FROM Venda
WHERE preco < 2.00 OR preco >= 2.00;
```

UNKNOWN UNKNOWN

UNKNOWN

Resultado: nenhuma tupla é selecionada!

Junção de Relações

- Consultas interessantes frequentemente combinam dados de múltiplas relações
- **Exemplo:** Usando as relações Lanchonete(nome, endereco, cnpj) e Venda(nome_lanch, nome_refri, preco), mostre o nome e endereço das lanchonetes que vendem Sprife.

```
SELECT nome, endereco
FROM Lanchonete, Venda
WHERE nome = nome_lanch AND
      nome_refri = 'Sprife';
```

Junção de Relações

- **Exemplo:** Usando as relações Lanchonete(nome, endereco, cnpj) e Venda(nome_lanch, nome_refri, preco), mostre o nome e endereço das lanchonetes que vendem Sprife.

```
SELECT nome, endereco

FROM Lanchonete, Venda

WHERE nome = nome_lanch AND
      nome_refri = 'Sprife';
```

produto cartesiano das duas relações

seleciona os pares de tuplas do produto cartesiano que se "casam" de acordo com essa condição

Junção de Relações

- **Exemplo:** Usando a relação Apreciador(nome_cliente, nome_refri) e Frequentador(nome_cliente, nome_lanch), encontre os refrigerantes apreciados por pelo menos uma pessoa que frequenta a lanchonete Burgão.

```
SELECT nome_refri
FROM Apreciador, Frequentador
WHERE nome_lanch = 'Burgão' AND
      Frequentador.nome_cliente = Apreciador.nome_cliente;
```

Para distinguir atributos de relações diferentes que possuem o mesmo nome: <relação>.<atributo>

Semântica Operacional - Junção

- Considere que há uma variável-tupla para cada relação na cláusula FROM
  - Essas variáveis visitam cada combinação possível de tuplas, uma de cada relação
- Se as variáveis-tuplas apontam para tuplas que satisfazem a cláusula WHERE, envie essas tuplas para a cláusula SELECT

Semântica Operacional - Junção

```
SELECT nome_refri
FROM Apreciador, Frequentador
WHERE nome_lanch = 'Burgão' AND
      Frequentador.nome_cliente =
      Apreciador.nome_cliente;
```

Verifica se é Burgão

A variável ***t2*** percorre cada tupla da relação 2

Apreciador

nome_cliente	nome_refri
John	Cola-Coca
George	Fanfa
Paul	Sprife
Ringo	Cola-Coca

Frequentador

nome_cliente	nome_lanch
John	Burgão
Paul	Burgão
John	KiSabor

A variável ***t1*** percorre cada tupla da relação 1

Verifica se t1.nome_cliente = t2.nome_cliente

Se igual, inclui na resposta

Resposta

nome
Cola-Coca
Sprife

Autojunção com Variáveis-Tuplas Explícitas

- Às vezes, uma consulta precisa usar duas cópias de uma mesma relação
  - Para diferenciar as cópias, acrescente o nome de uma variável-tupla na frente do nome da relação na cláusula FROM
- Exemplo: A partir de Refrigerante(nome_refri, fabricante), encontre todos os pares de refri feitos por um mesmo fabricante, de modo a
  - não produzir pares como (Fanfa, Fanfa)
  - Produzir pares em ordem alfabética, p.ex., (Fanfa, Sprife), mas não (Sprife, Fanfa)

```
SELECT r1.nome, r2.nome
FROM Refrigerante AS r1, Refrigerante AS r2
WHERE r1.fabricante = r2.fabricante AND r1.nome < r2.nome;
```

Obs.: o AS é opcional no comando.

União, Intersecção e Diferença

- União, intersecção e diferença de relações são expressas nas seguintes formas, todas envolvendo subconsultas:

  - **(<subconsulta>) UNION (<subconsulta>)**
  - **(<subconsulta>) INTERSECT (<subconsulta>)**
  - **(<subconsulta>) EXCEPT (<subconsulta>)**

INTERSECT - Exemplo

- Usando Apreciador(nome_cliente, nome_refri), Venda(nome_lanch, nome_refri, preco) e Frequentador(nome_cliente, nome_lanch), encontre os clientes e refrigerantes tais que:
  1. O cliente aprecia o refrigerante e
  2. O cliente frequenta pelo menos uma lanchonete que vende o refrigerante

```
(SELECT * FROM Apreciador)
  INTERSECT
(SELECT nome_cliente, nome_refri
 FROM Venda, Frequentador
 WHERE Frequentador.nome_lanch = Venda.nome_lanch);
```

Refris vendidos nas lanchonetes que o cliente frequenta

Semântica de Multiconjunto

- Os comandos SELECT-FROM-WHERE usam **semântica de multiconjunto**
  - A resposta deles pode conter tuplas repetidas
- Já para as operações de união, intersecção e diferença, o padrão é a **semântica de conjunto**
  - As **duplicações de tuplas são eliminadas quando a operação é aplicada**

Controlando a Eliminação de Duplicações

- Para forçar que o resultado seja um multiconjunto (ou seja, que as duplicações não sejam eliminadas) use a cláusula **ALL**, como em:

```
(SELECT nome_cliente FROM Frequentador)
EXCEPT ALL
(SELECT nome_cliente FROM Apreciador);
```

Frequentador

nome_cliente	nome_lanch
John	Burgão
Paul	Burgão
John	KiSabor
John	Delice
Ringo	KiSabor

Apreciador

nome_cliente	nome_refri
John	Cola-Coca
George	Fanfa
Paul	Sprife
George	Cola-Coca

Resposta

nome_cliente
John
John
Ringo

Controlando a Eliminação de Duplicações

- Com a cláusula **DISTINCT**, é possível forçar que o resultado de uma consulta seja um conjunto de tuplas (e não um muticonjunto), ou seja, forçar a remoção das tuplas repetidas na resposta

- Exemplo: A partir de Venda(nome_lanch, nome_refri, preco), encontre todos os diferentes preços cobrados por refrigerantes:

```
SELECT DISTINCT preco FROM Venda;
```

- Sem o DISTINCT, cada preço seria listado tantas vezes quanto o número de pares (nome_lanch,nome_refri) associados a esse preço na tabela Venda.

Subconsultas

- Um comando SELECT-FROM-WHERE parentizado (*subconsulta*) pode ser usado como um valor em diferentes lugares de uma consulta, incluindo nas cláusulas FROM e WHERE.

- Exemplo: no lugar de uma relação na cláusula FROM, nós podemos usar uma subconsulta e então consultar o seu resultado.
  - Para isso, faz-se necessário o uso de uma variável-tupla para nomear as tuplas do resultado.

Exemplo: Subconsulta no FROM

- Frequentador(nome_cliente, nome_lanch) Apreciador(nome_cliente, nome_refri):
- Encontre os refris apreciados por pelo menos uma pessoa que frequenta o Burgão.

```
SELECT nome_refri
FROM Apreciador, (SELECT nome_cliente
                  FROM Frequentador WHERE
                  nome_lanch = 'Burgão') CS
WHERE Apreciador.nome_cliente = CS.nome_cliente;
```

Subconsulta com os clientes que frequentam a lanchonete Burgão

Subconsultas que Devolvem uma Tupla

- Se uma subconsulta garantidamente produz uma única tupla, então a subconsulta pode ser usada como um valor.
  - Geralmente, a tupla tem um único componente.
  - Um erro é gerado em tempo de execução se não há nenhuma tupla no resultado ou se o resultado contém mais do que uma tupla.

Subconsultas que Devolvem uma Tupla - Exemplo

- Usando Venda(nome_lanch, nome_refri, preco), encontre as lanchonetes que vendem Fanfa pelo preço que a lanchonete Burgão cobra pela Sprife.
- A combinação de duas consultas resolve a questão:
  - Encontre o preço da Sprife na lanchonete Burgão.
  - Encontre as lanchonetes que vendem Fanfa por esse preço.

```
SELECT nome_lanch FROM Venda
WHERE nome_refri = 'Fanfa' AND
preco = (SELECT preco FROM Venda
         WHERE nome_lanch = 'Burgão'
         AND nome_refri = 'Sprife');
```

Subconsulta que recupera o preço cobrado pela Sprife pela lanch. Burgão

O Operador IN

- A expressão
- **<tupla> IN (<subconsulta>)**
- é verdadeira se e somente se a tupla é membro da relação produzida pela subconsulta.

  - Oposto:
  - **<tupla> NOT IN (<subconsulta>)**

- Expressões com IN podem aparecer na cláusula WHERE.

O Operador IN - Exemplo

- Usando Refrigerante(nome, fabricante) e Apreciador(nome_cliente, nome_refri), encontre o nome e o fabricante de cada refri que o Fred gosta.

```
SELECT *
FROM Refrigerante
WHERE nome IN
        (SELECT nome_refri
        FROM Apreciador
        WHERE nome_cliente = 'Fred');
```

Subconsulta que recupera os nomes dos refris que o Fred gosta.

O Operador EXISTS

- A expressão EXISTS(<subconsulta>) é verdadeira se e somente se o resultado da subconsulta não é vazio.

- Exemplo: A partir de Refrigerante(nome, fabricante), encontre os refris que são os únicos fabricados por seus fabricantes.

```
SELECT nome  FROM Refrigerante r1
WHERE NOT EXISTS
        (SELECT * FROM Refrigerante
        WHERE fabricante =
r1.fabricante
        AND nome <> r1.nome);
```

fabricante se refere à relação na cláusula FROM mais próxima que tenha o atributo

Subconsulta que recupera todo refri com o mesmo fabricante de r1, mas que não seja o refri r1.

Agregações

- As funções
- **SUM**, **AVG**, **COUNT**, **MIN** e **MAX**
- podem ser aplicadas a um atributo na cláusula SELECT para produzir a agregação dos valores do referido atributo.
- Além disso, **COUNT(*)** conta o número de tuplas.

Agregações - Exemplos

- A partir de Venda(nome_lanch, nome_refri, preço), encontre o preço médio do refri Fanfa:

```
SELECT AVG(preço)
FROM Venda WHERE nome_refri = 'Fanfa';
```

- Encontre o número de preços *diferentes* cobrados pela Fanfa:

```
SELECT COUNT(DISTINCT preço)
FROM Venda WHERE nome_refri = 'Fanfa';
```

Efeito dos Valores NULL nas Agregações

- Um valor NULL nunca contribui para uma soma, média ou contagem.
- O valor NULL também não será nem o mínimo, nem o máximo de um atributo se ele possuir outros valores.
- Mas se não existirem valores não nulos em um atributo, então o resultado da agregação é NULL.
  - Exceção: COUNT de um conjunto vazio é 0.

Efeito dos Valores NULL nas Agregações

```
SELECT count(*)
FROM Venda WHERE
nome_refri = 'Fanfa';
```

O número de lanchonetes que vendem Fanfa.

```
SELECT count(preço)
FROM Venda WHERE
nome_refri = 'Fanfa';
```

O número de lanchonetes que vendem Fanfa a um preço conhecido (ou seja, diferente de NULL).

Agrupamento

- Depois de uma expressão SELECT-FROM-WHERE, podemos adicionar **GROUP BY** e uma lista de atributos.
- A relação resultante do SELECT-FROM-WHERE com GROUP BY é agrupada de acordo com os valores de todos os referidos atributos
  - Quando há agrupamento, as agregações são aplicadas sobre cada grupo.

Agrupamento – Exemplo 1

- Exemplo: A partir de Venda(nome_lanch, nome_refri, preço), encontre o preço médio de cada refri:

```
SELECT nome_refri, AVG(preco) as preco_medio
FROM Venda GROUP BY nome_refri;
```

Venda

nome_lanch	nome_refri	preco
Burgão	Fanfa	6.00
Burgão	Cola-Coca	7.00
Burgão	Sprife	6.00
KiSabor	Cola-Coca	7.50
KiSabor	Sprife	7.50

Resultado

nome_refri	preco_medio
Fanfa	6.00
Cola-Coca	7.25
Sprife	6.75

Agrupamento – Exemplo 2

- A partir de Venda(nome_lanch, nome_refri, preço) e Frequentador(nome_cliente, nome_lanch), encontre, para cada cliente, o preço médio da Fanfa nas lanchonetes que ele frequenta:

```
SELECT nome_cliente, AVG(preço)
FROM Frequentador, Venda
WHERE nome_refri = 'Fanfa' AND
      Frequentador.nome_lanch = Venda.nome_lanch
GROUP BY nome_cliente;
```

Agrupamento - Restrição

- Se um agrupamento é usado, então cada elemento da lista do SELECT precisa ser **uma agregação** ou **um atributo da lista do GROUP BY**.

Exemplo de consulta incorreta:

```
SELECT nome_cliente, nome_lanch, AVG(preço)
FROM Frequentador, Venda
WHERE nome_refri = 'Fanfa' AND
    Frequentador.nome_lanch = Venda.nome_lanch
GROUP BY nome_cliente;
```

Agrupamento - Exemplo

- Encontrar a lanchonete que vende Fanfa ao menor preço
  - Forma incorreta:

```
SELECT nome_lanch, MIN(preço)
FROM Venda WHERE nome_refri = 'Fanfa';
```

  - Forma correta de fazer a consulta:

```
SELECT nome_lanch, preço FROM Venda
WHERE nome_refri = 'Fanfa' AND
            preço =  (SELECT MIN(preço) FROM Venda
                        WHERE nome_refri = 'Fanfa');
```

Filtrando Grupos com a Cláusula HAVING

- A cláusula **HAVING <condição>** pode aparecer depois da cláusula GROUP BY.
  - A condição do HAVING é aplicada sobre cada grupo de tuplas.
  - Grupos que não satisfazem a condição são eliminados da resposta da consulta.

- A partir de Venda(nome_lanch, nome_refri, preço) encontre os nomes dos refris que são vendidos a um preço médio menor do que R$ 6,00.

```
SELECT nome_refri, AVG(preço)
FROM Venda
GROUP BY nome_refri HAVING AVG(preço) <= 6
```

Requisitos para a Condição do HAVING

- Nas condições do HAVING, pode-se ter qualquer tipo de subconsultas;
- Fora de subconsultas, o HAVING pode referenciar um elemento somente se ele for:

1) Um atributo agrupador, ou
2) Uma agregação

(essa é a mesma regra usada para cláusulas SELECT com agregação).

Junção Interna (INNER JOIN)

- Forma da junção interna:

R INNER JOIN S ON <condição de junção>

- Equivale a um produto cartesiano de R por S, seguido de uma seleção de tuplas
- A **<condição de junção>** é usada para a seleção das tuplas
  - Todo par de tuplas de R X S que satisfaz a condição de junção é selecionado
- A palavra-chave "INNER" não precisa ser escrita
  - Esse é o tipo padrão de JOIN

Junção Interna - Exemplo

- Usando

Cliente(nome, endereço) e

Frequentador(nome_cliente, nome_lanch)

queremos encontrar todas duplas **(*c*, *l*)** tais que o cliente ***c*** mora em São Paulo e frequenta a lanchonete ***l***.

```
SELECT nome, nome_lanch
FROM Cliente JOIN Frequentador
ON nome = nome_cliente
WHERE endereço = 'São Paulo';
```

Junção Natural (Natural JOIN)

- Forma da junção natural:

R INNER NATURAL JOIN S

- Equivale a uma junção teta, mas tem uma condição de junção pré-definida.
- A condição de junção é sempre a igualdade sobre os pares de atributos das duas relações que possuem o mesmo nome.
- Elimina da resposta as duplicações de atributos.
- A palavra-chave “INNER” não precisa ser escrita
  - Esse é o tipo padrão de JOIN

Junção Natural - Exemplo

- Usando

Vendedor(**nome_lanch**, nome_refri) e Frequentador(nome_cliente, **nome_lanch**),

Queremos encontrar todos as triplas ***(l, r, c)*** tais que a lanchonete ***l*** vende o refrigerante ***r*** e é frequentada pelo cliente ***c***.

SELECT *

FROM Vendedor NATURAL JOIN Frequentador;

Obs.: O atributo nome_lanch aparecerá uma só vez na resposta.

Junção Externa (OUTER JOIN)

- Forma da junção externa:

R OUTER JOIN S ON <condição de junção>

- Inclui na resposta tuplas soltas, ou seja, tuplas que não "casam" com nenhuma outra da outra relação da junção.
- Ela pode conter também as seguintes cláusulas:

1) **LEFT**, **RIGHT**, ou **FULL** antes de OUTER
   - LEFT = inclui apenas as tuplas soltas de R
   - RIGHT = inclui apenas as tuplas soltas de S
   - FULL = inclui as tuplas soltas de ambas; esta é a opção padrão
2) **NATURAL** antes de tudo **ou** **ON** <condição> depois de JOIN

- A palavra-chave "OUTER" é opcional na expressão.

Junção Externa - Exemplo

- Usando Cliente(nome, endereço) e

Frequentador(nome_cliente, nome_lanch)

```
SELECT  C.*, F.nome_lanch
FROM Cliente as C LEFT OUTER JOIN Frequentador as F
ON nome = nome_cliente;
```

- Essa consulta nos dá uma lista de tuplas, onde cada tupla associa os dados de um cliente ao nome de uma lanchonete que ele frequenta.
- Se um cliente não frequenta nenhuma lanchonete, seus dados aparecerão na lista associados ao valor NULL (para o nome de lanchonete).

Junção Externa à Esquerda – Exemplo

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

D	E	F
2	3	10
2	3	11
6	7	12

Relação S

```
SELECT *
FROM R LEFT OUTER JOIN S
ON (B=D AND C=E);
```

A	B	C	D	E	F
1	2	3	2	3	10
1	2	3	2	3	11
4	5	6	NULL	NULL	NULL
7	8	9	NULL	NULL	NULL

Resultado de $R \ltimes_{B=D, C=E} S$

Junção Externa à Direita – Exemplo

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

D	E	F
2	3	10
2	3	11
6	7	12

Relação S

```
SELECT *
FROM R RIGHT OUTER JOIN S
ON (B=D AND C=E);
```

A	B	C	D	E	F
1	2	3	2	3	10
1	2	3	2	3	11
NULL	NULL	NULL	6	7	12

Resultado de $R \rtimes_{B=D, C=E} S$

Junção Externa Completa – Exemplo

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

D	E	F
2	3	10
2	3	11
6	7	12

Relação S

```
SELECT *
FROM R FULL OUTER JOIN S
ON (B=D AND C=E);
```

A	B	C	D	E	F
1	2	3	2	3	10
1	2	3	2	3	11
4	5	6	NULL	NULL	NULL
7	8	9	NULL	NULL	NULL
NULL	NULL	NULL	6	7	12

Resultado de $R \bowtie_{B=D, C=E} S$

Junção Natural Externa à Esquerda – Exemplo

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

B	C	D
2	3	10
2	3	11
6	7	12

Relação S

```
SELECT *
FROM R NATURAL LEFT OUTER JOIN S
ON (B=D AND C=E);
```

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	NULL
7	8	9	NULL

Resultado da junção natural externa à esquerda

Junção Natural Externa à Esquerda – Exemplo

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

B	C	D
2	3	10
2	3	11
6	7	12

Relação S

```
SELECT *
FROM R NATURAL RIGHT OUTER JOIN S
ON (B=D AND C=E);
```

A	B	C	D
1	2	3	10
1	2	3	11
NULL	6	7	12

Resultado da junção natural externa à direita

Junção Natural Externa à Esquerda – Exemplo

A	B	C
1	2	3
4	5	6
7	8	9

Relação R

B	C	D
2	3	10
2	3	11
6	7	12

Relação S

```
SELECT *
FROM R NATURAL FULL OUTER JOIN S
ON (B=D AND C=E);
```

A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	NULL
7	8	9	NULL
NULL	6	7	12

Resultado da junção natural externa completa

Linguagem SQL
Comandos para a Modificação de Dados

Os slides sobre Consultas em SQL são uma adaptação dos *slides* do prof. Jeffrey Ullman, da *Stanford University*

http://infolab.stanford.edu/~ullman/dscb/gslides.html

Modificações no Banco de Dados

- Um comando de modificação não devolve um (multi)conjunto de tuplas como resultado (como uma consulta faz); ele modifica o estado do BD de alguma forma
- Existem 3 tipos de modificações:
  - **Inserção** de tuplas
  - **Remoção** de tuplas
  - **Modificação** dos valores dos componentes de tuplas existentes

Inserção

- Para inserir uma única tupla:

INSERT INTO <relação>
VALUES (<lista de valores>);

- Exemplo: adicione a Apreciador(nome_cliente, nome_refri) o fato de que Ana gosta de Fanfa.

```
INSERT INTO Apreciador
VALUES('Ana', 'Fanfa');
```

Especificando atributos no INSERT

- Nós podemos especificar depois do nome da relação uma lista de atributos.
- Há duas razões para se fazer isso:
  - Quando esquecemos a ordem de criação dos atributos na relação ou
  - Quando não temos valores para todos os atributos e queremos que o SGBD preencha os componentes faltantes com NULL ou um valor default.

- Ex.: Outra forma de adicionar o fato de que Ana gosta de Fanfa a Apreciador(nome_cliente, nome_refri):

```
INSERT INTO Apreciador(nome_refri, nome_cliente)
VALUES('Fanfa', 'Ana');
```

INSERT com Valores Padrão

```
CREATE TABLE Cliente (
  nome CHAR(30) PRIMARY KEY,
  endereco CHAR(50) DEFAULT 'Av. Paulista, 123',
  telefone CHAR(16)
);

INSERT INTO Cliente(nome) VALUES('Ana');
```


- Tupla resultante:

Cliente

nome	endereço	telefone
John Lennon	Av. Paulista 123	NULL

Inserção de Várias Tuplas

- Podemos inserir o resultado todo de uma consulta em uma relação usando a forma: INSERT INTO <relação> (<subconsulta>);
- Ex.: Usando Frequentador(nome_cliente, nome_refri), insira em uma nova relação Colegas(nome) todos os colegas “em potencial” da John Lennon, i.e., os clientes que frequentam pelo menos uma lanchonete frequentada pelo John:

```
INSERT INTO Colegas
    (SELECT f2.nome_cliente
     FROM Frequentador f1, Frequentador f2
     WHERE f1.nome_cliente = 'John Lennon' AND
           f2.nome_cliente <> 'John Lennon' AND
           f1.nome_lanch = f2.nome_lanch );
```

Remoção

- Para remover tuplas que satisfazem uma condição em uma relação:

DELETE FROM <relação> WHERE <condição>;

- Exemplo: Remova de Apreciador(nome_cliente, nome_refri) o fato de que John Lennon gosta de Fanfa.

```
DELETE FROM Apreciador
WHERE nome_cliente = 'John Lennon'
      AND nome_refri = 'Fanfa';
```

Remoção – Outros Exemplos

- Exemplo: Remova de Refrigerante(nome, fabricante) todos os refris que não são vendidos por nenhuma lanchonete.

```
DELETE FROM Refrigerante
WHERE nome NOT IN
        (SELECT nome_refri FROM Venda);
```

- Exemplo: Esvazie a relação Apreciador.

```
DELETE FROM Apreciador;
```

Alteração

- Para mudar o valor de atributos em tuplas que já existem em uma relação:

UPDATE <relação>
SET <lista de atribuições a atributos>
WHERE <condição de seleção das tuplas a serem modificadas>;

Assim como no comando DELETE, a cláusula WHERE é opcional.

- Exemplo: Mude o número do telefone do cliente Paul para (11)555-1212 e endereço para ‘Av. Paulista, 123’.

```
UPDATE Cliente
SET telefone = '(11)555-1212',
    endereco = 'Av. Paulista, 123'
WHERE nome = 'Paul';
```

Transações em SQL

Slides baseados no material do prof. Jeffrey Ullman,
da Universidade de Stanford

Por que Transações?

- Sistemas de BD geralmente são acessados por muitos usuários ou processos ao mesmo tempo
  - Esses acessos podem ser de consulta ou modificação de dados
- Diferentemente dos sistemas operacionais, que suportam a interação entre processos, um SGBD Relacional precisa evitar os problemas causados pela interação entre os processos

Exemplo: Interação ruim

- Um casal saca R$100 de uma conta corrente conjunta em diferentes caixas eletrônicos ao mesmo tempo
  - O SGBD relacional deve garantir que nenhum dos débitos na conta sejam perdidos

- Comparação: Um sistema operacional permite que duas pessoas editem um mesmo documento ao mesmo tempo. Se ambas gravarem as alterações, as mudanças de uma das duas serão perdidas

Transações

- ***Transação*** = processo envolvendo consultas e/ou modificações no BD
- Geralmente, possui algumas propriedades rígidas relacionadas a controle de concorrência
- Em SQL, é formada por comandos simples ou pelo controle explícito do programador

Modelo Transacional ACID

***Transações ACID garantem*:**

- **Atomicidade**: Ou todas as operações da transação são corretamente refletidas no banco de dados, ou nenhuma delas é.
- **Consistência**: A execução de uma transação de forma isolada preserva a consistência do banco de dados.
- **Isolamento**: Embora múltiplas transações possam ser executadas concorrentemente, cada transação deve ser alheia às outras transações executadas simultaneamente. Resultados intermediários de uma transação devem ser ocultados das demais transações concorrentes.
- **Durabilidade**: Após uma transação ser concluída com sucesso, as alterações que ela realizou no banco de dados persistem, mesmo na ocorrência de falhas do sistema.

Modelo Transacional ACID

***Transações ACID garantem (em termos mais simples)*:**

- **Atomicidade**: ou a transação completa é feita, ou nada é feito
- **Consistência**: as restrições do BD devem ser garantidas
- **Isolamento**: para o usuário, deve parecer que há somente a sua transação em execução num dado momento
- **Durabilidade**: os efeitos da transação devem "sobreviver" a uma falha

Em alguns sistemas, formas mais "fracas" de transações também são implementadas.

Transações na SQL

- Não existe no padrão SQL um comando do tipo **BEGIN TRANSACTION** para começar uma transação de um modo explícito
  - O início de uma transação é implícito, quando instruções SQL começam a ser executadas
  - Mas muitos “dialetos” de SQL ou SGBDs implementam o **BEGIN TRANSACTION** (ou algo parecido)
- Toda transação precisa ter uma instrução de fim explícita (ou seja, um **COMMIT** ou um **ROLLBACK**)

COMMIT

- O comando **COMMIT** da SQL causa o encerramento da transação
  - Depois do COMMIT, as modificações feitas no BD na transação se tornam permanentes

ROLLBACK

- A instrução **ROLLBACK** da SQL também causa o encerramento da transação, mas ***abortando-a***
  - Efeito: nenhuma modificação é feita no BD de fato
  - Falhas como divisão por 0 ou uma violação de restrição também podem causar um *rollback*, mesmo que o programador não o tenha solicitado

Exemplo: Processos Interativos

- Considere a relação Vendas(nome_lanche,nome_refri,preco) e suponha que a lanchonete Burgão vende apenas Fanfa por R$2.50 e Sprife por R$3.00
- Considere que o banco de dados está sendo acessado ao mesmo tempo por dois usuários diferentes: a cliente Sara e o gerente do Burgão.
- Sara está consultando Vendas, para saber o maior e o menor preço cobrado por um refrigerante no Burgão
- O gerente do Burgão decide parar de vender Fanfa e Sprife, e passar a vender só Cola-Coca por R$3.50

Exemplo: Processos Interativos

- Sara executa os dois comandos SQL a seguir, com rótulos **(max)** e **(min)** para nos ajudar a lembrar o que eles fazem:

(max)

```
SELECT MAX(preco) FROM Vendas
WHERE nome_lanch = ’Burgão’;
```

(min)

```
SELECT MIN(preco) FROM Vendas
WHERE nome_lanch = ’Burgão’;
```

Exemplo: Processos Interativos

- Ao mesmo tempo, o gerente executa os seguintes comandos rotulados com **(del)** e **(ins)**:

(del)

```
DELETE FROM Vendas
WHERE nome_lanch = ’Burgão’;
```

(ins)

```
INSERT INTO Vendas
VALUES(’Burgão’, ’Cola-Coca’, 3.50);
```

Entrelaçamentos dos Comandos

Comandos da Sara

(max)
SELECT MAX(preco) FROM Vendas
WHERE nome_lanch = 'Burgão';

(min)
SELECT MIN(preco) FROM Vendas
WHERE nome_lanch = 'Burgão';

Comandos do Gerente

(del)
DELETE FROM Vendas
WHERE nome_lanch = 'Burgão';

(ins)
INSERT INTO Vendas
VALUES('Burgão', 'Cola-Coca', 3.50);

- Embora **(max)** deva vir antes de **(min)**, e **(del)** deva vir antes de **(ins)**, não há nenhuma outra restrição com relação à ordem desses comandos (a menos que agrupemos os comandos da Sara em uma transação e o comandos do gerente em outra)

Exemplo: Entrelaçamento Estranho

- Suponha que os passos sejam executados na seguinte ordem:

(max)(del)(ins)(min)

Preços do Burgão	{2.50, 3.00}	{2.50, 3.00}		{3.50}
Comando	**(max)**	**(del)**	**(ins)**	**(min)**
Resultado	3.00			3.50

Linha do Tempo →

- Sara verá MAX < MIN!

Corrigindo o Problema Usando Transações

- Se encapsulássemos os comandos **(max)(min)** da Sara em uma transação, então não teríamos essa inconsistência
- Sara veria os preços do Burgão em um momento fixo do tempo
  - Ou antes, ou depois, ou no meio da mudança de preços feita pelo gerente do Burgão, mas o MAX e MIN seriam computados a partir do mesmo conjunto de preços


```
BEGIN TRANSACTION
SELECT MAX(preco) FROM Vendas
WHERE nome_lanch = 'Burgão';
SELECT MIN(preco) FROM Vendas
WHERE nome_lanch = 'Burgão';
COMMIT
```

Outro Problema: *Rollback*

- Suponha que o gerente do Burgão execute (del)(ins) não como uma transação, mas depois de executar esses comandos, pense melhor sobre eles e execute uma instrução de ROLLBACK
- Se Sara executar seus comandos depois do (ins) mas antes do *rollback*, ela verá o valor 3.50, que nunca existiu de fato no banco de dados

Exemplo: Problema *Rollback*

- Suponha que os passos sejam executados na seguinte ordem:

(del)(ins)(max)(min)(rollback)

Preços do Burgão	{2.50, 3.00}		{3.50}	{3.50}	{3.50}	{2.50, 3.00}
Comando	**(del)**	**(ins)**	**(max)**	**(min)**	**(rollback)**	
Resultado			3.50	3.50		

Linha do Tempo →

- Sará leu um valor de MAX e MIN que posteriormente se tornou inválido, porque não foi confirmado no BD → problema da **leitura suja**

Solução

- Se o gerente executar (del)(ins) como uma transação, seu efeito não pode ser visto por outros antes que a transação execute um **COMMIT**
- Se a transação executar ROLLBACK em vez de COMMIT, seus efeitos nunca serão vistos

Transação do Gerente

```
BEGIN TRANSACTION
DELETE FROM Vendas
WHERE nome_lanch = 'Burgão';
INSERT INTO Vendas
VALUES('Burgão', 'Cola-Coca', 3.50);
COMMIT
```

Propriedades das Transações na SQL

- Na SQL, é possível atribuir propriedades a uma transação por meio do comando

  SET TRANSACTION

- Essas propriedades podem ser de dois tipos:
  - **Modo de acesso** (READ WRITE ou READ ONLY)
  - **Nível de isolamento** (SERIALIZABLE, READ COMMITTED, REPEATABLE READ ou READ UNCOMMITED)

Transações do Tipo “READ WRITE”

- Tipo de transação “padrão”
  - Lê dados, faz algum processamento e então grava dados no BD
- Esse tipo de transação está mais sujeito a problemas de seriação
- Para indicar que uma transação faz leituras e escritas, podemos usar o comando a seguir (mas isso já é o padrão):

SET TRANSACTION READ WRITE;

Transações do Tipo “READ WRITE”

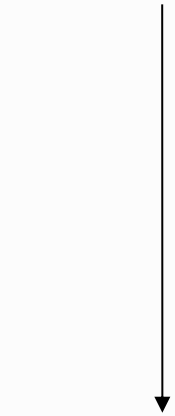
- Exemplo: reserva de assentos em um avião (envolve 2 etapas)

```
(op1)
EXEC SQL SELECT ocupado INTO :ocup
FROM Voos WHERE NumVoo = :voo AND
        DtVoo = :data AND PoltronaVoo = :poltr;

If (!ocup)
(op2)
EXEC SQL  UPDATE Voo SET ocupado = TRUE
                WHERE NumVoo = :voo AND
                        DtVoo = :data AND PoltronaVoo = :poltr;
```

Transações do Tipo “READ WRITE”

- Exemplo: reserva de assentos em um avião
- Vamos considerar que há dois clientes tentando reservar a mesma poltrona (22A) num dado voo:



Transação 1 (t1)	Transação 2 (t2)
Cliente 1 encontra a poltrona 22A desocupada (t1-op1)	
	Cliente 2 encontra a poltrona 22A desocupada (t2-op1)
Cliente 1 ocupa a poltrona 22A (t1-op2)	
	Cliente 2 ocupa a poltrona 22A (t2-op2)

Transações do Tipo “READ WRITE”

- A menos que se indique explicitamente o contrário, uma transação geralmente é considerada pelo SGBD como sendo READ WRITE
  - **Esse tipo de transação é sempre executado de forma seriada**
  - A execução seriada do exemplo do slide anterior forçaria um dos seguintes tipos de execução: T1 seguido de T2, ou T2 seguido de T1.

Transações do Tipo “READ WRITE”

- Exemplo: reserva de assentos em um avião

Vamos considerar que há dois clientes tentando reservar a mesma poltrona (22A) num dado voo

Execução seriada das duas transações: t1, t2

Linha do Tempo

Transação 1 (t1)	Transação 2 (t2)
Cliente 1 encontra a poltrona 22A desocupada (t1-op1)	
Cliente 1 ocupa a poltrona 22A (t1-op2)	
	Cliente 2 encontra a poltrona 22A **ocupada** (t2-op1)
	Cliente 2 cancela a tentativa de reserva

Transações do Tipo “READ WRITE”

- Exemplo: reserva de assentos em um avião

Vamos considerar que há dois clientes tentando reservar a mesma poltrona (22A) num dado voo

Execução seriada das duas transações: t2, t1

Linha do Tempo

Transação 1 (t1)	Transação 2 (t2)
	Cliente 2 encontra a poltrona 22A desocupada (t2-op1)
	Cliente 2 ocupa a poltrona 22A (t2-op2)
Cliente 1 encontra a poltrona 22A **ocupada** (t1-op1)	
Cliente 1 cancela a tentativa de reserva	

Transações do Tipo “READ ONLY”

- São transações que apenas leem dados do BD
- Esse tipo de transação geralmente pode ser executado paralelamente a outras transações “READ ONLY” ou que não escrevem nos dados de interesse
- Para indicar que uma transação é “READ ONLY”, usar o comando a seguir no início da transação:

SET TRANSACTION READ ONLY;

Níveis de Isolamento de Transações

- A SQL-92 define quatro ***níveis de isolamento***:
  - SERIALIZABLE, REPEATABLE READ, READ COMMITTED ou READ UNCOMMITTED
- Eles determinam quais interações são permitidas entre transações que executam ao mesmo tempo
- O nível de isolamento de uma transação é especificado por meio do comando:

SET TRANSACTION ISOLATION LEVEL <nível de isolamento>;

Exemplo: Transações do tipo “SERIALIZABLE”

- Considere que

  Sara = **(max)(min)** e

  Gerente = **(del)(ins)**

  são transações

- Se a transação da Sara rodar com nível de isolamento “SERIALIZABLE”, então Sara verá o BD **ou antes ou depois da execução completa da transação do gerente**, mas não no meio da execução
- “SERIALIZABLE” é o nível de isolamento padrão para transações

O Nível de Isolamento é uma Escolha “Pessoal”

- A escolha de um nível para uma transação afeta somente a forma como **essa** transação vê o BD, e não como os outros o vêem
- Exemplo: Se a transação do gerente executar no modo “serializable”, mas a da Sara não, então a Sara **poderá** não ver preço algum para o Burgão
  - Ou seja, para a Sara, pode ocorrer a situação em que parecerá que ela está executando no meio da transação do gerente

Transações “READ COMMITTED”

- Se a transação da Sara executa com nível de isolamento “READ COMMITTED”, então ela pode ver somente dados permanentes (= *committed)*, mas não necessariamente os mesmos dados a cada vez
- Exemplo: Sob “READ COMMITTED”, o entrelaçamento **(max)(del)(ins)(min)**

  é permitido, desde que o gerente faça um **COMMIT** depois de **(ins)**

  - Nesse caso, Sara verá MAX < MIN

Preços do Burgão	{2.50, 3.00}	{2.50, 3.00}			{3.50}
Comando	**(max)**	**(del)**	**(ins)**	**(commit)**	**(min)**
Resultado	3.00				3.50

Linha do Tempo →

Transações “READ COMMITTED”

- Para indicar que uma transação é “READ COMMITTED”:

  SET TRANSACTION READ ONLY ISOLATION LEVEL READ COMMITTED;

- Ou

  SET TRANSACTION READ WRITE

  ISOLATION LEVEL READ COMMITTED;

Transações “REPEATABLE READ”

- Em transações com “REPEATABLE READ”, os requisitos são semelhantes aos do “READ COMMITTED”, com a adição de: se o dado é lido novamente, então tudo visto na primeira vez será visto na segunda também
  - Mas a segunda vez (e as leituras subsequentes) poderão ver tuplas a mais (ou seja, que não apareceram na primeira leitura)

- Para indicar que uma transação é “REPEATABLE READ”:

  SET TRANSACTION READ ONLY ISOLATION LEVEL REPEATABLE READ;

  Ou

  SET TRANSACTION READ WRITE ISOLATION LEVEL REPEATABLE READ;

Exemplo: “REPEATABLE READ”

- Suponha que Sara execute sob REPEATABLE READ, e a ordem da execução é **(max)(del)(ins)(min)**
- **(max)** vê os preços 2.50 e 3.00
- **(min)** pode ver 3.50, mas precisa também ver 2.50 e 3.00, porque eles foram vistos na leitura anterior feita por **(max)**

Preços do Burgão	{2.50, 3.00}	{2.50, 3.00}		{2.50, 3.00, 3.50}
Comando	**(max)**	**(del)**	**(ins)**	**(min)**
Resultado	3.00			2.50

Linha do Tempo →

Transações “READ UNCOMMITTED”

- Uma transação executando sob “READ-UNCOMMITTED” pode ver dados no BD mesmo se eles tiverem sido escritos por uma transação que ainda não sofreu COMMIT (e que pode nunca sofrer!)
- Exemplo: Se Sara executar sob “READ UNCOMMITTED”, ela poderá ver o preço 3.50 mesmo se o gerente aborte sua transação depois

Transações “READ UNCOMMITTED”

- Para indicar que uma transação é “READ UNCOMMITTED”:

SET TRANSACTION READ ONLY
ISOLATION LEVEL READ UNCOMMITTED;

Ou

SET TRANSACTION READ WRITE
ISOLATION LEVEL READ UNCOMMITTED;

- Para “READ UNCOMMITTED”, o padrão é “READ ONLY”, e não “READ WRITE” como nos outros casos

Violações que Podem Ocorrer em Isolamentos “Inferiores” a SERIALIZABLE

- **Leitura de dados sujos**
  - T1 pode ler uma atualização de T2 que ainda não foi confirmada. Se T2 for abortada, T1 terá lido um valor sujo (incorreto).
- **Leitura não repetível**
  - T1 pode ler um dado valor de uma tabela. Se depois T2 atualizar esse valor e T1 lê-lo novamente, T1 verá um valor diferente.
- **Fantasmas**
  - T1 pode ler um conjunto de linhas de uma tabela a partir de uma dada consulta. Suponha que depois T2 inseriu novas tuplas na tabela que também entrariam na resposta da consulta executada por T1. Se T1 for repetida, então ela verá “fantasmas”, ou seja, tuplas que não existiam anteriormente.
- Obs.: T1 e T2 usadas nos exemplos acima são transações.

Possíveis Violações por Nível de Isolamento

	Tipo de violação		
Nível	**Leitura "suja"**	**Leitura não repetível**	**Fantasma**
READ UNCOMMITTED	Sim (mas no PG não)	Sim	Sim
READ COMMITTED (Padrão no PG)	Não	Sim	Sim
REPEATABLE READ	Não	Não	Sim (mas no PG não)
SERIALIZABLE (Padrão na SQL)	Não	Não	Não

PG = PostgreSQL
https://www.postgresql.org/docs/14/sql-set-transaction.html
https://www.postgresql.org/docs/current/transaction-iso.html

Um Parênteses sobre a Leitura de “Dados Sujos”

- **Dados sujos** → dados escritos por uma transação que não foi confirmada (“*committed*”) ainda
- A leitura de dados sujos pode:
  - Não ser um problema
  - Ser um problema grave
  - Ser um problema leve, que justifica o risco de se ter dados ocasionalmente sujos em troca de melhor desempenho no BD
- Evitar a leitura de dados sujos custa bastante tempo a um SGBD e diminui o paralelismo na execução das transações

Exemplo: Problema Causado pela Leitura de Dados Sujos

- Considere um programa P que faz uma transferência entre duas contas bancárias:
  - Passo 1: Adiciona o dinheiro na conta 2
  - Passo 2: Verifica se a conta 1 tem saldo suficiente
    - Se não tiver, remove o dinheiro da conta 2 e termina
    - Se tiver, subtrai o dinheiro da conta 1 e termina
- Se P é executado de forma seriada, não há problema algum em colocarmos dinheiro temporariamente na conta 2 (ninguém verá esse dinheiro caso a transferência não seja concluída).
- Mas se a leitura de dados sujos for possível durante a execução de P, então o resultado pode ser incorreto.

Implementação dos Níveis de Isolamento pelo SGBD

- **Bloqueios (*locking*)**
  - Bloqueio do banco de dados inteiro vs. bloqueio em itens de dados
  - Bloqueios compartilhados e bloqueios exclusivos
- **Marcas de tempo (*Timestamps*)** - A cada transação é atribuída uma marca de tempo, por exemplo, no momento em que a transação se inicia.
  - Os itens de dados armazenam duas marcas de tempo: tempo de leitura e tempo de escrita
  - As marcas de tempo são usadas para detectar acessos fora de ordem
- **Múltiplas versões de cada item de dado**
  - Permite que transações leiam a partir de um “instantâneo” (*snapshot*) do banco de dados

Referências Bibliográficas

- ***Database Systems – A Complete Book*** (2ª edição), Garcia-Molina, Ullman e Widom, 2002.
  - Capítulo 6 - SQL (incluindo Transações)
  - Capítulo 7 - Restrições em SQL (Chaves e Chaves Estrangeiras)
- **Sistemas de Bancos de Dados** (7ª edição), Elmasri e Navathe.
  - Capítulos 6 - SQL Básico
  - Capítulo 7 - SQL Avançado
  - Capítulo 20 - Transações
- *Transaction Isolation in PostgreSQL:*

  https://www.postgresql.org/docs/current/transaction-iso.html

Obrigada!

Profª Kelly Rosa Braghetto | linkedin.com/in/kelly-rosa-braghetto/

MBA USP ESALQ