

SCC-244 – Mineração a partir de Grandes Bases de Dados

Tipos de dados não atômicos em SQL

Grupo de Bases de Dados e Imagens
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - São Carlos

3 de outubro de 2023
São Carlos, SP - Brasil

Apresentação sobre dados não atômicos: tipos dimensionais (**ARRAYS**) em SQL.

Roteiro

- 1 Tipo de dados Não-atômicos e o Modelo Orientado a Objetos
- 2 Tipos de dados Array
- 3 Tarefa para entrega até a próxima aula

Tipo de dados Não-atômicos

- O modelo relacional impõe que cada atributo seja **Atômico**, ou seja:
 - **Indivisível** - não tem partes
 - **Monovalorado** - tem apenas um valor
- No entanto, as **implementações** do modelo relacional – os **SGBDR** – usualmente violam diversos preceitos do modelo, tais como:
 - **Tuplas repetidas** (**DISTINCT**);
 - **Valores nulos** (**NULL**); etc.
- Portanto, é um caminho natural que **Modelos pós-relacionais** [👉 especialmente os **objeto-relacionais** 👈] também admitam que atributos possam violar a restrição de serem **Atômicos**.

Modelo Orientado a Objetos

- O conceito de um **Modelo de Dados Orientado a Objetos** foi desenvolvido a partir de **1985**, como um paralelo das Linguagens Orientadas a Objetos sendo desenvolvidas na época:

👉 **Smalltalk**, **Java**, **C→C++**, até as mais recentes **Python**, **Rust**, etc.

- Alguns **SGBD-OO** foram desenvolvidos nas duas décadas seguintes, alguns com grande impacto conceitual:



👉 **Orion**, **Exodus**, **O2**, etc. e os mais recentes **Versant DB4O**, **ObjectStore** e **InterSystems Caché e IRIS**, etc.

- Os conceitos desenvolvidos foram incorporados aos **SGBDs Relacionais**, com os conceitos padronizados em **SQL**, o que gerou os chamados **SGBD Relacionais-Objeto**.

★ Hoje, todos os grandes SGBDs seguem um modelo **Relacional-Objeto**, mas usualmente são chamados apenas **SGBDs Relacionais**, porque os SGBD-OO “puros” se tornaram irrelevantes.



Tipos de dados Não-atômicos em SQL

- Os recursos incorporados à linguagem SQL oriundos dos conceitos de orientação a objetos se refletem na possibilidade de:
 - Criação de tipos de dados definidos pelo usuário (`CREATE TYPE`);
 - Definição de atributos multivalorados.
- Diversos tipos de dados não atômicos são disponibilizados em SQL (e em particular pelo , entre eles :
 - Tipos compostos (tuplas:** `CREATE TYPE`)
 - Tipos JSON e XML  Estudaremos depois.
 - ARRAYS**

Tipos de dados Não-atômicos em SQL

Usuários podem definir seus próprios tipos de dados.

Tipo de dados definido pelo usuário (*User-Defined data Type*: UDT)

É um tipo derivado dos tipos nativos providos por uma ferramenta computacional e que os estendem com novas propriedades e ou restrições.

Tipos definidos pelo usuário

Existem diversas necessidades que tornam interessante definir um novo tipo de dados, entre elas definir um:

Tipo estruturado (*composite type*), ou tupla:

Especifica uma lista de nomes de (sub-)atributos com seus respectivos *tipos de dados*;

Exemplo: **Endereços, Nomes de pessoas**

Tipo enumerador: Especifica uma lista de valores de um *tipo de dado* básico;

Exemplo: **Cores**

Tipo de arranjo: Especifica a existência de um arranjo multidimensional de valores de um *tipo de dado* básico;

Exemplo: **Data das prestações**


Tipo base: Especifica um novo tipo de dados básico, usualmente estruturado com suas próprias leis de formação e manipulação.

Exemplo: **Números complexos**

Tipos definidos pelo usuário

Definição de tipos em



Em , um Tipo definido pelo usuário é um novo **objeto de tipo Type**.

Ele pode ser definido por um comando:

Criar um novo tipo de dados

```
CREATE TYPE name [AS [ENUM | RANGE ] (<propriedades>)];
```

Exemplos: **Tipo enumerador:** (Em , um tipo básico tem que ser **SQL**)


```
CREATE TYPE Cor AS
    ENUM ('Branco', 'Cinza', 'Vermelho');

CREATE TABLE Carro (
    Placa TEXT,
    Pintura Cor);
```


Tipos definidos pelo usuário

Definição de tipos em



Em , um Tipo definido pelo usuário é um novo **objeto de tipo Type**.

Ele pode ser definido por um comando:

Criar um novo tipo de dados

```
CREATE TYPE name [AS [ENUM | RANGE ] (<propriedades>)];
```


Exemplos: **Tipo enumerador**: Propriedades:

- Os valores são sensíveis à caixa das letras e a <brancos>:
'Vermelho' != 'vermelho.'
- A ordem de comparação e a mesma da sequência de valores:
'Branco' < 'Cinza' < 'Vermelho'

Tipos definidos pelo usuário

Definição de tipos em



Em , um Tipo definido pelo usuário é um novo **objeto de tipo Type**.

Ele pode ser definido por um comando:

Criar um novo tipo de dados

```
CREATE TYPE name [AS [ENUM | RANGE ] (<propriedades>)];
```


Exemplos: **Tipo estruturado:** (ou **Tipo composto:**)

```
CREATE TYPE Endereco AS  
    (Rua TEXT, Numero INT, Cidade TEXT);  
CREATE TYPE NomePF AS  
    (Prenome TEXT, NomeMeio TEXT, Sobrenome TEXT);  
CREATE TABLE Moradia AS  
    (Quem NomePF, Onde Endereço);  
INSERT INTO Moradia VALUES  
    (('Tom', 'Zé', 'Silva'), ('R. Pio X', 11, 'Jau'));
```

Tipos definidos pelo usuário

Definição de tipos em



Em , um Tipo definido pelo usuário é um novo **objeto de tipo Type**.

Ele pode ser definido por um comando:

Criar um novo tipo de dados

```
CREATE TYPE name [AS [ENUM | RANGE ] (<propriedades>)];
```


Exemplos: **Tipo estruturado:** (ou **Tipo composto:**)

Um tipo composto é o próprio tipo **ROW** de uma tabela, mas um comando **CREATE TYPE** permite definir o tipo sem criar uma tabela associada.

Tipos definidos pelo usuário

Definição de tipos em




Em , um Tipo definido pelo usuário é um novo **objeto de tipo Type**.

Ele pode ser definido por um comando:


Criar um novo tipo de dados

```
CREATE TYPE name [AS [ENUM | RANGE ] (<propriedades>)];
```

Exemplos: **Tipo de arranjo:**

Em , sempre que um novo tipo é criado, cria-se automaticamente o tipo *Array* correspondente, que em [SQL](#) é referenciado pelo nome do tipo básico seguindo por `[]`.

Tipos de dados Array

-  PostgreSQL aceita atributos definidos como matrizes multidimensionais de dimensão variável: o tipo *Array*.
- Ele pode ser de qualquer tipo de dados: Números, cadeias, *time*, tuplas, UDT, etc.
- Tipos de dados não-atômicos permitem desnormalizar as relações.
- O tipo `ARRAY` pode ter diversos valores, acessíveis por um índice.

Tipos de dados Array

- Por exemplo: suponha que a relação de **Notas dos alunos** tem:
 - Os atributos usuais, como sua chave (**NUSP**, **Sigla**) e outros atributos escalares, como quem é o **Avaliador**, mais
 - um **atributo multivalorado unidimensional** com as notas das diversas **Provas**,
 - um **atributo multivalorado bidimensional** com as de diversos **Trabalhos** (um trabalho por linha), onde cada trabalho tem as notas de diversos exercícios (em colunas):

```
CREATE Table Notas(
  NUSP NUMERIC(10) NOT NULL,
  Sigla CHAR(8) NOT NULL,
  PRIMARY KEY (NUSP, Sigla),
  Avaliador NomePF,
  Provas NUMERIC(3.1)[ ],
  TrabsExerc NUMERIC(3.1)[ ][ ]
);
```

Tipos de dados Array

Definição de um **ARRAY**

- Um **ARRAY** pode ser definido
 - Com **dimensão variável**:
 - Provas **NUMERIC(3.1) []**
 - TrabsExerc **NUMERIC(3.1) [] []**
 - Pesos **INTEGER ARRAY**
 - Conceitos **TEXT ARRAY**
 - Ou com **dimensão fixa**:
 - Trancamentos **DATE[2]**
- **ARRAYs** podem ser uni-dimensionais, bi-dimensionais (matrizes), ou qualquer dimensão...
- Cada sub-dimensão tem que ter a mesma quantidade de valores (uma matriz tem que ter todas as linhas com o mesmo tamanho)
- A contagem dos índices começa em '1'.

Tipos de dados Array

Definir valores para **ARRAY**

Existem duas representações sintáticas para valores de **ARRAYs**:
como uma “**constante literal**” (texto comum):

- '{ val1, val2, ...}'
- '{ {val11, val12, ...} {val21, val22, ...} }'

como um “**construtor de ARRAY**”:

- ARRAY[val1, val2, ...]
- ARRAY[[val11, val12, ...] [val21, val22, ...]]

Tipos de dados Array

Definir valores para **ARRAY**

EXEMPLO:

```
INSERT INTO Notas (NUSP, Sigla, Provas, TrabsExerc, Conceitos) VALUES
  (1234, 'SCC-001', '{5,8, 10}', '{{5,5,5},{6,7,8}}', '{"regular","bom","ótimo"}'),
  (2345, 'SCC-001', '{9,9,10}', '{{8,7,8},{0,3,2}}', '{"ótimo", "ótimo", "ótimo"}'),
  (1234, 'SCC-002', ARRAY[7,7], ARRAY[[1,2,3,4],[5,6,7,8]], ARRAY['bom', 'bom']);
```

NUSP	Sigla	Provas	TrabsExercs	Conceitos
1234	'SCC-001 '	{5.0, 8.0, 10.0}	{{5.0, 5.0, 5.0},{6.0, 7.0, 8.0}}	'{regular,bom,ótimo}'
2345	'SCC-001 '	{9.0, 9.0, 10.0}	{{8.0, 7.0, 8.0},{0.0, 3.0, 2.0}}	'{ótimo,ótimo,ótimo}'
1234	'SCC-002 '	{7.0, 7.0}	{{1.0, 2.0, 3.0, 4.0},{5.0, 6.0, 7.0, 8.0}}	'{bom,bom}'

Tipos de dados Array

Acessando **ARRAY**s

- Para retornar todos os valores, basta indicar o atributo:

```
SELECT NUSP, Sigla, Provas FROM Notas;
```

- Retornar todos os valores de uma “linha”:

```
SELECT NUSP, Sigla, Provas[1] FROM Notas;
```

- Retornar um elemento específico de uma matriz:

```
SELECT NUSP, Sigla, TrabsExerc  
FROM Notas  
WHERE TrabsExerc[2][3]<5;
```

Tipos de dados Array

Acessando **ARRAY**s

Pode-se indicar “faixas de índices” (*slices*) com `[<inicio>:<fim>]`

- Se `<inicio>` for omitido, vale desde o início;
- Se `<fim>` for omitido, vale até o fim;

Exemplos:

- Retornar as notas de todos os exercícios do trabalho 1

```
SELECT NUSP, Sigla, TrabsExerc[1][:]  
FROM Notas;
```

- Retornar as notas dos exercícios 1 e 2 do trabalho 2

```
SELECT NUSP, Sigla, TrabsExerc[2:2][1:2]  
FROM Notas;
```

(se um dos índices tiver especificação de faixa, todos os índices são tratados como faixa.)

Tipos de dados Array

Procurando valores em **ARRAY**s

- Localizar os alunos que tiveram nota 10 em alguma prova

```
SELECT NUSP, Sigla, Provas  
FROM Notas  
WHERE 10 = ANY (Provas);
```

- Localizar os alunos que tiveram pelo menos 7 em todas as provas

```
SELECT NUSP, Sigla, Provas  
FROM Notas  
WHERE 7 <= ALL (Provas);
```

Tipos de dados Array

Convertendo **ARRAY** para tuplas

É possível converter **ARRAYs** para tuplas.

Para isso existe a **Função UNNEST**:

Sintaxe da função Unnest

```
Unnest (anyarray) → <Conjunto de anyelement>
```

- Cada **anyelement** produz uma tupla com os valores dos demais atributos repetidos.
- Se houver mais de uma função **UNNEST** no comando, ambas emparelham valores na mesma tupla.

Tipos de dados Array

Convertendo **ARRAY** para tuplas

Uma função auxiliar interessante é a **Função Generate_Subscripts**:

Sintaxe da função Generate_Subscripts

```
Generate_Subscripts (anyarray, dim[, reverse]) →  
    <Conjunto de anyelement>  
onde <reverse>:={true ou false}
```

- Ele gera o subscrito correspondente àquela tupla.

Tipos de dados Array

Convertendo **ARRAY** para tuplas

EXEMPLOS:

- ```
SELECT NUSP, Sigla, Generate_Subscripts(Provas, 1), Unnest(Provas)
FROM Notas;
```

- ```
SELECT NUSP, Sigla, generate_subscripts(Provas, 1),
       Unnest(Provas), Unnest(Conceito)
FROM Notas;
```

- ```
SELECT NUSP, Sigla, generate_subscripts(Provas, 1),
 Unnest(Provas), Unnest(Conceito),
 Unnest(ARRAY['um', 'dois', 'tres', 'quatro']);
FROM Notas;
```



Usualmente a chave do resultado é a chave já existente concatenada à função **Generate\_Subscripts**.

Neste exemplo é **NUSP, Sigla, Generate\_Subscripts(Provas, 1)**

# Tipos de dados Array

Convertendo tuplas para **ARRAY**

A operação inversa é possível também: converter tuplas para **ARRAYs**.  
Para isso existe a **Função `ARRAY_AGG()`**:

## Sintaxe da função `ARRAY_AGG`

```
ARRAY_AGG(<expressão>
 [ORDER BY [<expressão> {ASC|DESC}]], [...])
```

Agrega o valor de <expressão> em todas as tuplas para um **ARRAY**.



# Tipos de dados Array

Convertendo tuplas para **ARRAY**

**Exemplo:** Suponha que existem as relações:

**Alunos**=(Nome, NUSP, Idade, Cidade)    **Matric**=(NUSP, CodigoTurma, Nota)

- Podemos gerar uma nova tabela com a junção das duas, onde cada tupla tem a nota de um aluno em uma turma... (formato *tradicional*...)

```
SELECT A.NUSP, M.CodigoTurma, M.Nota
FROM Aluno A JOIN Matricula M ON A.NUSP=M.NUSP;
```

- ...ou podemos colocar as turmas e respectivas notas junto ao próprio aluno:

```
SELECT A.NUSP,
 Array_Agg(M.CodigoTurma ORDER BY CodigoTurma),
 Array_Agg(M.Nota ORDER BY CodigoTurma)
FROM Aluno A JOIN Matricula M ON A.NUSP=M.NUSP
GROUP BY A.NUSP;
```

# Tipos de dados Array

Convertendo tuplas para **ARRAY**

## Exemplo:

No entanto, por usar a cláusula **GROUP BY** agrupando pelo **NUSP**, não se pode associar outros atributos que não estejam em funções de agregação

- por exemplo, não se pode obter o nome dos alunos!

👉 Isso pode ser resolvido usando a função **Array\_Agg** como uma *window function*:

# Tipos de dados Array

Convertendo tuplas para **ARRAY** Exemplo:

```
SELECT DISTINCT A.*,
 Array_Agg(M.CodigoTurma) OVER(
 PARTITION BY A.NUSP ORDER BY CodigoTurma
 RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) CodigoTurma,
 Array_Agg(M.Nota) OVER(
 PARTITION BY A.NUSP ORDER BY CodigoTurma
 RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) Nota
FROM Aluno A JOIN Matricula M ON A.NUSP=M.NUSP;
```

| Nome     | NUSP | Idade | Nome       | CodigoTurma       | Nota       |
|----------|------|-------|------------|-------------------|------------|
| Catarina | 5678 | 23    | Sao Carlos | {102,104}         | {7,8}      |
| Cibele   | 6789 | 21    | Araraquara | {101,104}         | {6,5}      |
| Celia    | 9012 | 20    | Rio Claro  | {100,101,104}     | {6,9,9}    |
| Cicero   | 3456 | 22    | Araraquara | {100,101,102,104} | {7,9,9,10} |
| Celina   | 8901 | 27    | Sao Carlos | {100,101,102,104} | {4,7,9,8}  |
| Celso    | 2345 | 22    | Sao Carlos | {100,101,102,104} | {9,7,7,7}  |
| Corina   | 7890 | 25    | Rio Claro  | {101,104}         | {10,9}     |
| Carlos   | 1234 | 21    | Sao Carlos | {100,101,104}     | {8,9,4}    |
| Cesar    | 9123 | 21    | Araraquara | {100,102,104}     | {9,9,7}    |
| Carlitos | 4567 | 21    | Ibitinga   | {100,101,102,104} | {7,5,10,4} |

# Tipos de dados Array

Convertendo tuplas para **cadeia de caracteres**

- Quando o tipo básico de dados a ser agregado é cadeia de caracteres (*string*), o resultado será um **ARRAY** de cadeias.
- Como exemplo, suponha que queremos colocar os alunos que se matricularam em cada disciplina junto com a disciplina:

```
SELECT DISTINCT M.CodigoTurma,
 Array_Agg(A.NUSP) OVER(PARTITION BY M.CodigoTurma ORDER BY A.NUSP
 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) NUSPMatric,
 Array_Agg(A.Nome) OVER(PARTITION BY M.CodigoTurma ORDER BY A.NUSP
 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) NomeMatric,
 Array_Agg(M.Nota) OVER(PARTITION BY M.CodigoTurma ORDER BY A.NUSP
 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) NotaMatric
FROM Aluno A JOIN Matricula M ON A.NUSP=M.NUSP
```

| CodTurma | NUSPMatric                                          | NomeMatric                                                                 | NotaMatric             |
|----------|-----------------------------------------------------|----------------------------------------------------------------------------|------------------------|
| 100      | {1234,2345,3456,4567,8901,9012,9123}                | '{Carlos,Celso,Cicero,Carlitos,Celina,Celia,Cesar}'                        | {8,9,7,7,4,6,9}        |
| 101      | {1234,2345,3456,4567,6789,7890,8901,9012}           | '{Carlos,Celso,Cicero,Carlitos,Cibele,Corina,Celina,Celia}'                | {9,7,9,5,6,10,7,9}     |
| 102      | {2345,3456,4567,5678,8901,9123}                     | '{Celso,Cicero,Carlitos,Catarina,Celina,Cesar}'                            | {7,9,10,7,9,9}         |
| 104      | {1234,2345,3456,4567,5678,6789,7890,8901,9012,9123} | '{Carlos,Celso,Cicero,Carlitos,Catarina,Cibele,Corina,Celina,Celia,Cesar}' | {4,7,10,4,8,5,9,8,9,7} |

# Tipos de dados Array

## Convertendo tuplas para cadeia de caracteres

- No caso específico do tipo básico de dados a ser agregado ser cadeia de caracteres, as vezes é interessante ter como resultado agregado uma única cadeia, usando algum separador indicado.
- Isso pode ser feito com a Função `STRING_AGG()`:

### Sintaxe da função `STRING_AGG()`

```
STRING_AGG(<expressão>, <separador>
[ORDER BY [<expressão> {ASC|DESC}]], [...])
```

**Agrega** Concatena o valor de <expressão> em todas as tuplas para uma cadeia, separando cada valor pelo caracter <sep>.

- A diferença entre `ARRAY` e `String` é que um `ARRAY` tem os elementos indexáveis (primeiro, segundo, etc.) e uma `String` é um valor só, usando o <separador> para poder separar as sub-cadeias usando funções de manipulação de texto.

# Tipos de dados Array

Comparando **ARRAY** com cadeia de caracteres





- Comparando **ARRAY** com **String** :

```
SELECT DISTINCT M.CodigoTurma,
String_Agg(A.Nome, ',') OVER(
 PARTITION BY M.CodigoTurma
 ORDER BY A.NUSP
 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AsString,
Array_Agg(A.Nome) OVER(
 PARTITION BY M.CodigoTurma
 ORDER BY A.NUSP ROWS
 BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AsArray
FROM Aluno A JOIN Matricula M ON A.NUSP=M.NUSP;
```

| CodigoTurma | AsString                                                               | AsArray                                                                    |
|-------------|------------------------------------------------------------------------|----------------------------------------------------------------------------|
| 102         | Celso,Cicero,Carlitos,Catarina,Celina,Cesar                            | { Celso,Cicero,Carlitos,Catarina,Celina,Cesar }                            |
| 104         | Carlos,Celso,Cicero,Carlitos,Catarina,Cibele,Corina,Celina,Celia,Cesar | { Carlos,Celso,Cicero,Carlitos,Catarina,Cibele,Corina,Celina,Celia,Cesar } |
| 101         | Carlos,Celso,Cicero,Carlitos,Cibele,Corina,Celina,Celia                | { Carlos,Celso,Cicero,Carlitos,Cibele,Corina,Celina,Celia }                |
| 100         | Carlos,Celso,Cicero,Carlitos,Celina,Celia,Cesar                        | { Carlos,Celso,Cicero,Carlitos,Celina,Celia,Cesar }                        |

# Tipos de dados Array

Comparando **ARRAY** com cadeia de caracteres

- A atualização **SQL:2016** definiu a *window function* **LISTAGG**, que corresponde ao comportamento da **String\_Agg** em  PostgreSQL.
- Mas  PostgreSQL não implementa as condições de tratamento de **OVERFLOW** que foram definidas no padrão.
- O que acontece quando a *string* resultante excede o tamanho definido para o resultado?  
  PostgreSQL apenas retorna erro. O padrão permite indicar que se trunque o resultado.

# Tarefa para entrega até a próxima aula

Para entrega via **Tidia.Repositório** até às 16h:00 de 10 de outubro

O exercício se refere ao conceito de *arrays* em SQL.

- Crie uma tabela que tenha, para cada **CD\_Município** com **Pacientes**,
  - um atributo de tipo **TEXT**: com o nome do município,
  - outro atributo de tipo **Array** de **INTeiros**: com uma coluna para cada mês desde de [Janeiro de 2019 até Junho de 2021], que conta quantos **Desfechos** (pela **DT\_Atendimento**) ocorreram naquele mês naquela cidade.

Atenção:

A dificuldade para resolver este exercício é fazer com que todos os meses tenham uma contagem (“uma coluna para cada mês”), que deve ser **zero** quando não houver desfechos naquele mês naquela cidade.



# Tarefa para entrega até a próxima aula

## Lembretes:

- Uma sequência de valores pode ser criada com a função:  
`Generate_Series(Inicio, Final, Intervalo)`.  
Por exemplo:  
`Generate_Series(DATE('2022-01-01'), DATE('2022-31-12'),  
INTERVAL '1 Month') AS MesesDe2022`
- A função `COALESCE(<lista de valores>)` retorna o primeiro valor não nulo da lista.

## Sugestão:

- 1 Crie uma tabela com as respectivas contagens de desfechos em cada cidade em cada mês;
- 2 Gere um produto cartesiano com todos os meses necessários por todas as cidades, com contagens zero.
- 3 Execute a junção completa dos dois resultados escolhendo os valores adequadamente: isso garante que todos os meses tenham valores (nem que seja zero) em todas as cidades.

# SCC-244 – Mineração a partir de Grandes Bases de Dados

## Tipos de dados não atômicos em SQL

Grupo de Bases de Dados e Imagens  
Instituto de Ciências Matemáticas e de Computação  
Universidade de São Paulo - São Carlos

3 de outubro de 2023  
São Carlos, SP - Brasil



Grupo de  
Bases de Dados  
e Imagens

Dados  
Multidimensionais  
**FIM**