

SIGM – AULA PRÁTICA 4 – GRUPO 2

1. Criar e destruir bases de dados (com “template” usando *psql*)

a) Foi completado o *script* \scripts\00_script_CRIAR_BD_GIS.txt de modo a eliminar e construir a base de dados de nome my_gis_aug_real.

```
-----  
-- Dados da Conexao  
-----  
\set userName postgres  
\set hostName localhost  
\set portNumber 5432  
  
-----  
-- Nome da BD  
-----  
\set dataBase my_gis_aug_real  
;
```

```
\echo  
\echo "Remover Base de Dados" :dataBase  
;  
DROP DATABASE IF EXISTS :dataBase  
;
```

```
-----  
\echo  
\echo "Criar Base de Dados" :dataBase  
;  
CREATE DATABASE :dataBase  
;  
  
\echo  
\echo "Estabelecer Conexao com a Base de Dados" :dataBase  
;  
\c :dataBase :userName :hostName :portNumber  
;  
  
\echo  
\echo "Aplicar o Extensor "postgis" @ Base de Dados" :dataBase  
;  
CREATE EXTENSION postgis  
;
```

2. Estender modelo relacional - novas estruturas (tipos)

a) Foi adicionado um novo tipo de nome `t_vector` com componentes `x` e `y` do tipo real.

```
CREATE TYPE t_vector AS (  
  x REAL,  
  y REAL  
);  
  
-- TESTE  
--  
SELECT cast( (3, 4) AS t_vector ) AS v;  
SELECT (3, 4)::t_vector AS v;  
  
SELECT cast( (3.33, 4.44) AS t_vector ) AS v;  
SELECT (3.33, 4.44)::t_vector AS v;
```

	v	
	t_vector	
1	(3,4)	

	v	
	t_vector	
1	(3.33,4.44)	

b) Foi adicionado um novo tipo de nome `t_velocidade` com duas componentes, linear e angular onde linear é do tipo `t_vector` e angular é do tipo real.

```
CREATE TYPE t_velocidade AS (  
  linear t_vector,  
  angular REAL  
);  
  
-- TESTE  
--  
SELECT cast( ( cast( (3, 4) AS t_vector ), 10.09 ) AS t_velocidade ) AS v;  
SELECT ( (3, 4)::t_vector, 10.09 )::t_velocidade AS v;
```

	v	
	t_velocidade	
1	("(3,4)",10.09)	

```
CREATE TYPE t_aceleracao AS (  
  linear t_vector,  
  angular REAL  
);  
  
-- TESTE  
--  
SELECT cast( ( cast( (3, 4) AS t_vector ), 10.09 ) AS t_aceleracao ) AS v;  
SELECT ( (3, 4)::t_vector, 10.09 )::t_aceleracao AS v;
```

c) Foi adicionado um novo tipo (estrutura) de nome `t_aceleracao` com duas componentes, linear e angular (nesta ordem), onde linear é do tipo `t_vector` e angular é do tipo real.

	v	
	t_aceleracao	
1	("(3,4)",10.09)	

3. Estender modelo relacional - novo comportamento (funções)

a) Foi analisada a função: produto_vector_por_escalar(vec t_vector, v real).

produto_vector_por_escalar: Multiplica um vetor (t_vector) por um escalar (valor real).

- **Parâmetros:**
 - **vec:** Um vetor do tipo t_vector.
 - **v:** Um número real (escalar).
- **Retorno:** Um novo vetor t_vector resultante da multiplicação de cada componente do vetor de entrada pelo escalar.
- **Implementação:** A função é escrita em Python (plpython3u) e realiza a multiplicação componente a componente do vetor pelo escalar.

```
--  
-- Produto de um vector por um escalar  
--  
CREATE OR REPLACE FUNCTION produto_vector_por_escalar( vec t_vector, v real )  
RETURNS t_vector  
AS $$  
new_x = vec[ "x" ] * float( v )  
new_y = vec[ "y" ] * float( v )  
return { "x": new_x, "y": new_y }  
$$ LANGUAGE plpython3u;
```

b) Foram analisadas também as funções:

produto_vector_por_escalar_PLGSQL(vec t_vector, v real)

produto_vector_por_escalar_SQL(vec t_vector, v real)

```
CREATE OR REPLACE FUNCTION produto_vector_por_escalar_PLGSQL( vec t_vector, v real )  
RETURNS t_vector  
AS $$  
DECLARE  
    new_x real;  
    new_y real;  
BEGIN  
    new_x := vec.x * v;  
    new_y := vec.y * v;  
    RETURN (new_x, new_y);  
END;  
$$ LANGUAGE plpgsql;
```

Função produto_vector_por_escalar_PLGSQL

- **Propósito:** Multiplica um vetor (t_vector) por um escalar (valor real) utilizando a linguagem PL/pgSQL.
- **Parâmetros:**
 1. **vec:** Um vetor do tipo t_vector, contendo as coordenadas x e y.
 2. **v:** Um número real representando o escalar.
- **Retorno:** Um novo vetor t_vector resultante da multiplicação componente a componente.
- **Implementação:**
 1. **Declaração de variáveis:** Declara duas variáveis new_x e new_y do tipo real para armazenar as novas coordenadas.
 2. **Cálculo:** Multiplica cada coordenada do vetor de entrada vec pelo escalar v e armazena os resultados nas variáveis new_x e new_y.
 3. **Retorno:** Retorna um novo vetor t_vector com as coordenadas calculadas.

```
CREATE OR REPLACE FUNCTION produto_vector_por_escalar_SQL( vec t_vector, v real )
RETURNS t_vector
AS $$
    SELECT ( ($1).x * $2, ($1).y * $2 )::t_vector;
    -- Ou, escrevendo o CAST de outro modo,
    -- SELECT CAST( ( ($1).x * $2, ($1).y * $2 ) AS t_vector );
$$ LANGUAGE 'sql';
```

Função produto_vector_por_escalar_SQL

- **Propósito:** Multiplica um vetor (t_vector) por um escalar (valor real) utilizando uma única instrução SQL.
- **Parâmetros:**
 - **vec:** Um vetor do tipo t_vector (primeiro parâmetro).
 - **v:** Um número real (segundo parâmetro).
- **Retorno:** Um novo vetor t_vector resultante da multiplicação componente a componente.
- **Implementação:**
 - **Cálculo inline:** Realiza a multiplicação das coordenadas diretamente na cláusula SELECT.
 - **Casting:** Converte o resultado da multiplicação em um vetor t_vector usando o operador de casting ::t_vector.

```
-- Definição do Operador
```

```
CREATE OPERATOR * (
    leftarg = t_vector,
    rightarg = real,
    procedure = produto_vector_por_escalar,
    commutator = *
);
```

c) Foi implementada a função: soma_vector_vector(vec_a t_vector, vec_b t_vector) que devolve um t_vector com a soma dos dois t_vector recebidos nos parâmetros formais.

	v
	t_vector
1	(6.5,8.9)

	v1 t_vector	v2 t_vector
1	(20.5,28.5)	(32.5,44.5)

```
--
-- Soma de dois vetores
--
CREATE OR REPLACE FUNCTION soma_vector_vector( vec_a t_vector, vec_b t_vector )
RETURNS t_vector
AS $$
new_x = vec_a["x"] + vec_b["x"]
new_y = vec_a["y"] + vec_b["y"]
return { "x": new_x, "y": new_y }
$$ LANGUAGE plpython3u;

SELECT soma_vector_vector((3, 4)::t_vector, (2, -1)::t_vector);
-- Resultado esperado: (5, 3)
```

```
-----
CREATE OPERATOR + (
leftarg = t_vector,
rightarg = t_vector,
procedure = soma_vector_vector,
commutator = +
);

-- TESTE
--
SELECT cast( (3, 4) AS t_vector ) + cast( (3.5, 4.9) AS t_vector ) AS v;
-- resultado: v = (6.5, 8.9)

SELECT (3, 4) + (3.5, 4.9) * 5 AS v1, ( cast( (3, 4) AS t_vector ) + cast( (3.5, 4.9) AS t_vector ) ) * 5 AS v2;
-- resultado: v1 = (20.5, 28.5); v2 = (32.5, 44.5)
```

d) A funcao normalizar(vec t_vector) normaliza o t_vector.

```
-----
normal: raíz cuadrada da soma dos quadrados de cada componente
CREATE OR REPLACE FUNCTION normalizar( vec t_vector )
RETURNS t_vector
AS $$
from math import sqrt

def normalizar(vec):
    norma = sqrt(vec["x"]**2 + vec["y"]**2)
    new_x = vec["x"] / norma
    new_y = vec["y"] / norma
    return { "x": new_x, "y": new_y }

return normalizar(vec)
$$ LANGUAGE plpython3u;

SELECT normalizar((3, 4)::t_vector);
-- Resultado esperado: aproximadamente (0.6, 0.8)
```

	normalizar t_vector
1	(0.6,0.8)

4. Modelar noção de “cinemática” e seu “histórico”

a) Foi feita a tabela cinemática(id, orientacao, velocidade, aceleracao, g_posicao) com tipos e restrições de integridade adequados.

```
CREATE TABLE cinemática(
    id SERIAL PRIMARY KEY,
    orientacao REAL,
    velocidade t_velocidade,
    aceleracao t_aceleracao
);
SELECT AddGeometryColumn( '', 'cinemática', 'g_posicao', 3763, 'POINT', 2 );
```

```
INSERT INTO cinemática( id, g_posicao, orientacao, velocidade, aceleracao ) VALUES(
1,
ST_GeomFromText( 'POINT( 5 6 )', 3763 ),
0.0,
ROW( ROW( 1, 1 ), 1.3 ),
ROW( ROW( 0.5, 3 ), 1.0 )
);
```

```
INSERT INTO cinemática( id, g_posicao, orientacao, velocidade, aceleracao ) VALUES(
2,
ST_GeomFromText( 'POINT( 2 3 )', 3763 ),
0.0,
ROW( ROW( 1, 1 ), 0.3 ),
ROW( ROW( 2, 0.5 ), 1.0 )
);
```

```
SELECT *
FROM cinemática;
```

	id [PK] integer	orientacao real	velocidade t_velocidade	aceleracao t_aceleracao	g_posicao geometry
1	1	0	("(1,1)",1.3)	("(0.5,3)",1)	0101000020B30E000000000000000000144000000000000018...
2	2	0	("(1,1)",0.3)	("(2,0.5)",1)	0101000020B30E0000000000000000004000000000000008...

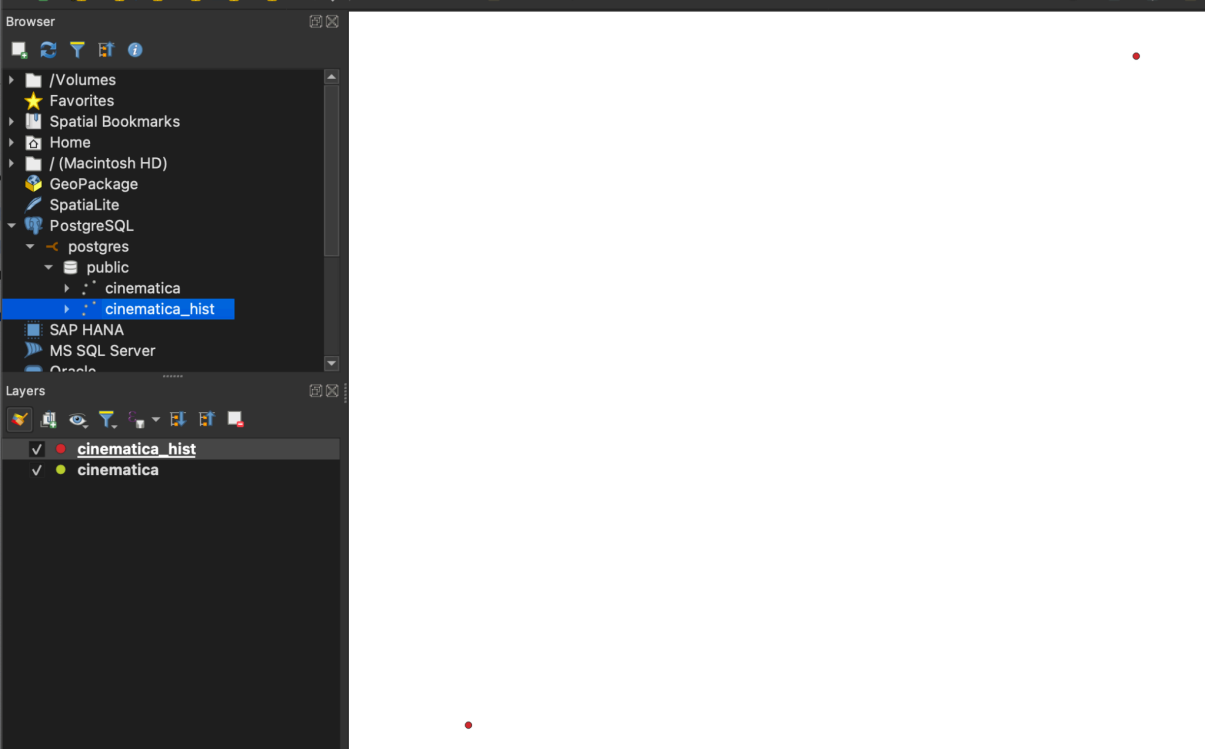
b) Foi construída a tabela cinemática_hist(id_hist, <mesmos-atributos-de-cinemática>) onde id_hist é a chave primária do tipo SERIAL.

```
INSERT INTO cinematica_hist
SELECT nextval('cinematica_hist_id_hist_seq'), id, NOW(), orientacao, velocidade , aceleracao, g_posicao
FROM cinematica;

SELECT *
FROM cinematica_hist;
```

	id_hist [PK] integer	id integer	data_hora timestamp with time zone	orientacao real	velocidade t_velocidade	aceleracao t_aceleracao	g_posicao geometry
1	1	1	2024-10-23 11:09:52.448844+01	0	((1,1),1,3)	((0.5,3),1)	0101000020B30E0000000000000000014400000000000000018...
2	2	2	2024-10-23 11:09:52.448844+01	0	((1,1),0,3)	((2,0.5),1)	0101000020B30E000000000000000004000000000000000008...

c) Foi utilizado o QuantumGIS para visualizar os objectos construídos.



5. Criar o comportamento associado à “cinemática”

a) Foi analisada a função novo_posicao(g_posicao geometry, velocidade t_velocidade, tempo real).

g_posicao := g_posicao + velocidade.linear * tempo (calcula a nova posição do objeto)

- **g_posicao:** Representa a posição atual do objeto no espaço.
- **velocidade.linear:** É a velocidade com que o objeto se move.
- **tempo:** É o intervalo de tempo que passou desde a última atualização.

Interpretação: A nova posição é obtida somando à posição antiga o deslocamento causado pela velocidade linear durante o intervalo de tempo.

orientacao := orientacao + velocidade.angular * tempo

- **orientacao:** Representa a direção para onde o objeto aponta.
- **velocidade.angular:** É a velocidade com que o objeto gira.
- **tempo:** É o intervalo de tempo que passou desde a última atualização.

Interpretação: A nova orientação é obtida somando à orientação antiga a rotação causada pela velocidade angular durante o intervalo de tempo.

velocidade.linear := velocidade.linear + aceleracao.linear * tempo

- **velocidade.linear:** Representa a velocidade linear atual do objeto.
- **aceleracao.linear:** É a taxa de variação da velocidade linear.
- **tempo:** É o intervalo de tempo que passou desde a última atualização.

Interpretação: A nova velocidade linear é obtida somando à velocidade linear antiga a variação da velocidade causada pela aceleração linear durante o intervalo de tempo.

velocidade.angular := velocidade.angular + aceleracao.angular * tempo

- **velocidade.angular:** Representa a velocidade angular atual do objeto.
- **aceleracao.angular:** É a taxa de variação da velocidade angular.
- **tempo:** É o intervalo de tempo que passou desde a última atualização.

Interpretação: A nova velocidade angular é obtida somando à velocidade angular antiga a variação da velocidade angular causada pela aceleração angular durante o intervalo de tempo.

```
CREATE OR REPLACE FUNCTION novo_posicao( g_posicao geometry, velocidade t_velocidade, tempo real )
RETURNS geometry
AS $$
SELECT
ST_Translate( $1,
               ((($2).linear * $3 ).x,
                (($2).linear * $3 ).y )
             )
$$ LANGUAGE 'sql';

-- TESTE
-- tempo=2
-- _____
SELECT ST_Astext( novo_posicao( g_posicao, velocidade, 2 ) )
FROM cinematica;
-- resultado:
-- POINT(7, 8)
-- POINT(4, 5)
```

	st_astext text
1	POINT(7 8)
2	POINT(4 5)

tempo=2

	st_astext text
1	POINT(6 7)
2	POINT(3 4)

tempo=1

	st_astext text
1	POINT(8 9)
2	POINT(5 6)

tempo=3

b) Foi implementada novo_orientacao(orientacao real, velocidade t_velocidade, tempo real).

```
CREATE OR REPLACE FUNCTION novo_orientacao( orientacao real, velocidade t_velocidade, tempo real )
RETURNS real
AS $$
    return orientacao + velocidade['angular'] * tempo
$$ LANGUAGE plpython3u;

-- TESTE
-- tempo=2
--
SELECT novo_orientacao( orientacao, velocidade, 2 )
FROM cinematica;
-- resultado:
-- 2.6
-- 0.6
```

	novo_orientacao	real
1		2.6
2		0.6

c) Foi implementada novo_velocidade(velocidade t_velocidade, aceleracao t_aceleracao, tempo real). Valide com o teste do script e teste a função para diferentes instantes de tempo.

```
CREATE OR REPLACE FUNCTION novo_velocidade(velocidade t_velocidade, aceleracao t_aceleracao, tempo real)
RETURNS t_velocidade AS $$
DECLARE
    nova_velocidade t_velocidade;
BEGIN
    nova_velocidade.linear := velocidade.linear + aceleracao.linear * tempo;
    nova_velocidade.angular := velocidade.angular + aceleracao.angular * tempo;
    RETURN nova_velocidade;
END;
$$ LANGUAGE plpgsql;

-- TESTE
-- tempo=2
--
SELECT novo_velocidade( velocidade, aceleracao, 2 )
FROM cinematica;
-- resultado:
-- ("(2, 7)", 3.3)
-- ("(5, 2)", 2.3)
```


	novo_velocidade	t_velocidade
1		("(2,7)",3.3)
2		("(5,2)",2.3)

d) Foi executada a vista v_novo_cinematica.

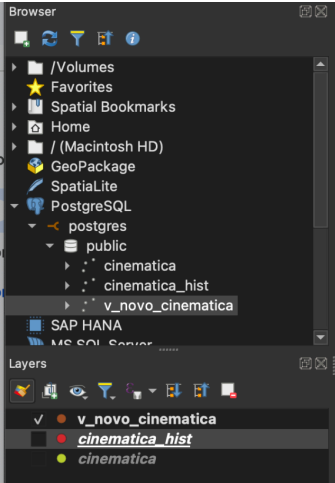
```

-----
-- Vista da 'cinematica' para um instante do tempo (e.g., 2)
-----

```

	id integer	orientacao real	velocidade t_velocidade	aceleracao t_aceleracao	posicao_texto text	 posicao geometry
1	1	2.6	((2,7),3,3)	((0.5,3),1)	POINT(7 8)	0101000020B30E000000000000000001C40000000000000020...
2	2	0.6	((5,2),2,3)	((2,0.5),1)	POINT(4 5)	0101000020B30E00000000000000000104000000000000014...

e) Utilize o QuantumGIS para visualizar as vistas construídas.



6. Simular trajetórias

a) Foi analisado o *script* para executar uma trajectória dos objectos 1 e 2.

```
DELETE 2
DELETE 2
INSERT 0 1
INSERT 0 1
```

id_hist	id	orientacao	velocidade	aceleracao	g_posicao
3	1	0	("(1,1)","1.3)	("(0.5,3)","1)	0101000020B30E000000000000000001440000000000001840
4	2	0	("(1,1)","0.3)	("(2,0.5)","1)	0101000020B30E00000000000000000040000000000000840
5	1	2.3	("(1.5,4)","2.3)	("(0.5,3)","1)	0101000020B30E0000000000000000001A40000000000002440
6	2	1.3	("(3,1.5)","1.3)	("(2,0.5)","1)	0101000020B30E0000000000000000001440000000000001240
7	1	5.6	("(2,7)","3.3)	("(0.5,3)","1)	0101000020B30E0000000000000000002140000000000003140
8	2	3.6	("(5,2)","2.3)	("(2,0.5)","1)	0101000020B30E0000000000000000002440000000000001A40
9	1	9.9	("(2.5,10)","4.3)	("(0.5,3)","1)	0101000020B30E0000000000000000002640000000000003B40
10	2	6.8999996	("(7,2.5)","3.3)	("(2,0.5)","1)	0101000020B30E0000000000000000003140000000000002240
11	1	15.2	("(3,13)","5.3)	("(0.5,3)","1)	0101000020B30E0000000000000000002C400000000000004440
12	2	11.2	("(9,3)","4.3)	("(2,0.5)","1)	0101000020B30E0000000000000000003A400000000000002840
13	1	21.5	("(3.5,16)","6.3)	("(0.5,3)","1)	0101000020B30E00000000000000000083140000000000004C40
14	2	16.5	("(11,3.5)","5.3)	("(2,0.5)","1)	0101000020B30E000000000000000000804240000000000002F40
15	1	28.8	("(4,19)","7.3)	("(0.5,3)","1)	0101000020B30E0000000000000000008354000000000000C05240
16	2	22.8	("(13,4)","6.3)	("(2,0.5)","1)	0101000020B30E00000000000000000049400000000000083340
17	1	37.1	("(4.5,22)","8.3)	("(0.5,3)","1)	0101000020B30E0000000000000000003A4000000000000405840
18	2	30.099998	("(15,4.5)","7.3)	("(2,0.5)","1)	0101000020B30E00000000000000000040504000000000003840
19	1	46.399998	("(5,25)","9.3)	("(0.5,3)","1)	0101000020B30E0000000000000000003F40000000000000805E40
20	2	38.399998	("(17,5)","8.3)	("(2,0.5)","1)	0101000020B30E000000000000000000805440000000000003040
21	1	56.699997	("(5.5,28)","10.3)	("(0.5,3)","1)	0101000020B30E0000000000000000004042400000000000C06240
22	2	47.699997	("(19,5.5)","9.3)	("(2,0.5)","1)	0101000020B30E0000000000000000004059400000000000404140

(20 rows)

b) Foram obtidas as trajectórias conforme ilustradas em x_fig_duasTrajectories.bmp.

