ISEL \ DEETC

MEIM | MEIC

SIGM / CSI – Guia aula prática #04

Paulo Trigo Silva

1. Criar e destruir base de dados (com "`template`" usando *psql*)

a) Complete 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` (onde "`aug_real`" exprime *augmented reality* !).

2. Estender modelo relacional - novas estruturas (tipos)

Cenário A: considere um contexto onde se pretende caracterizar a velocidade e a aceleração de corpos (objectos) imersos num espaço 2D. Ambas, velocidade e aceleração, têm duas componentes: uma linear e outra angular. A componente linear é descrita por um vector de duas dimensões, `x` e `y`, que caracterizam a velocidade, ou aceleração, em cada eixo (`x` e `y`) da base 2D. A componente angular é descrita por um único valor (e.g., radianos por segundo) que representa a velocidade, ou aceleração, com que a orientação do corpo se altera durante o seu movimento.

Nesta alínea deve completar o *script*: `01_script_ESTENDER_MODELO_RELACIONAL.txt`**.**

a) Adicione, a `my_gis_aug_real`, um novo tipo (estrutura) de nome `t_vector` com duas componentes, `x` e `y` (nesta ordem) ambas do tipo `real`. Realize o teste indicado no *script*.

b) Adicione, a `my_gis_aug_real`, um novo tipo (estrutura) de nome `t_velocidade` com duas componentes, `linear` e `angular` (nesta ordem), onde `linear` é do tipo `t_vector` e `angular` é do tipo `real`. Realize o teste indicado no *script*.

c) Adicione, a `my_gis_aug_real`, 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`. Realize o teste indicado no *script*.

3. Estender modelo relacional - novo comportamento (funções)

Continue a considerar o ***Cenário A*** (acima descrito). Confirme que tem instalado o interpretador de Python (versão 3.x.y; i.e., superior a 2.7.x). Pode verificar em "`...\PostgreSQL\10\lib`" a existência da biblioteca "`plpython3.dll`" que suporta compatibilidade com a versão 3 do Python (a biblioteca PostgreSQL que suportava a versão 2.6.x era a "`plpython.dll`"). Mais info em: \ *Start* \ *All Programs* \ *PostgresSQL 10* \ *Installation Notes* \ *Procedural Languages* onde está indicado que a versão suportada na PostgreSQL 10 é a Python 3.4.

Nota: Na minha instalação, o *PostgreSQL 10 funcionou com Python 3.6.4*. É possível que funcione com versão posterior do Python mas a testada é a 3.6.4. Para instalar uma versão aceda a `https://www.python.org/downloads/windows/` (atenção à compatibilidade 32bit ou 64bit). **Após instalar garantir** que a **variável `PATH`** do seu ambiente Windows contém o caminho para o executável (`python.exe`) desta versão da linguagem Python.

Atenção: Caso tenha dificuldade em executar o código Python (ao longo desta alínea) prossiga implementando em "plgsql" (como indicado no script que suporta esta alínea). Usamos Python para ilustrar a capacidade de extensão do modelo relacional (no PostgreSQL) usando linguagem imperativa "standard". No entanto a dificuldade resultante de incompatibilidade entre a versão do

Python e do atual PostgreSQL não pode impedir que prossiga o seu trabalho. Em síntese, se deparar com dificuldades na execução do Python avance para a implementação com "plgsql".

Nesta alínea deve continuar a completar: `01_script_ESTENDER_MODELO_RELACIONAL.txt`.

a) Analise a função: `produto_vector_por_escalar(vec t_vector, v real)`. Considere também a informação adicional sobre PL/Python em `a01_postgresql-10-A4.pdf`, cap. 45.

b) Analise também as funções:

```
produto_vector_por_escalar_PLGSQL( vec t_vector, v real )
produto_vector_por_escalar_SQL( vec t_vector, v real )
```

Elimine a anterior definição (`python`) do operador `*` e defina-o à custa de uma destas funções. ***Nota***: no caso de ter dificuldade com a execução das funções Python então, a partir daqui, use exclusivamente funções "`plpgsql`" (e "`sql`"); cf., `postgresql-10-A4.pdf`, cap. 42.

c) Implemente 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. Valide a implementação realizando testes (e.g., os indicados no script).

d) Implemente: `normalizar( vec t_vector )` que normaliza o `t_vector` em parâmetro. Valide com testes. *Nota*: para normalizar dividir cada componente pela norma (do vector); a norma é a sua distância à origem (dimensão), dada por sqrt($x^2 + y^2$); em Python x^y é x**y; em plgsql x^y é dado por "power(x, y)".

4. Modelar noção de "cinemática" e seu "histórico"

Cenário B: considere que a noção de cinemática (caracterização do movimento) se descreve por um identificador (`id`) único, do tipo `integer`, (permitindo descrever vários movimentos diferentes), uma orientação (`orientacao`) do tipo `real` (direcção, em radianos, para onde se está virado) uma velocidade (`velocidade`) tipo `t_velocidade` e aceleração (`aceleracao`) tipo `t_aceleracao`. Adicionalmente, é essencial saber qual a coordenada geográfica (`g_posicao`) do tipo `POINT` onde se encontra o objecto. Pretende-se conhecer os trajectos (e.g., ao longo do tempo) dos objectos em `cinematica`. Para isso basta manter o histórico de todos os valores da "cinemática".

Nesta alínea deve completar: `02_script_CRIAR_MODELO_CINEMATICA.txt`.

a) Faça a tabela `cinematica(id, orientacao, velocidade, aceleracao, g_posicao)` com tipos e restrições de integridade adequados. Teste inserindo os tuplos indicados no *script*.

b) Construa a tabela `cinematica_hist(id_hist, <mesmos-atributos-de-cinematica>)` onde `id_hist` é a chave primária do tipo `SERIAL`. Atenção à restrição de integridade referencial (`id` para `cinematica`). Teste a tabela copiando os dados de cinemática (indicado no *script*).

c) Utilize o QuantumGIS para visualizar os objectos construídos.

SIGM / CSI – Guia aula prática #04

Paulo Trigo Silva

5. Criar o comportamento associado à "cinemática"

Considere a seguinte formulação para actualização da cinemática:

- g_posicao := g_posicao + velocidade.linear * tempo
- orientacao := orientacao + velocidade.angular * tempo
- velocidade.linear := velocidade.linear + aceleracao.linear * tempo
- velocidade.angular := velocidade.angular + aceleracao.linear * tempo

Nesta alínea deve completar: `03_script_CRIAR_COMPORTAMENTO_CINEMATICA.txt`.

a) Analise: `novo_posicao( g_posicao geometry, velocidade t_velocidade, tempo real )`. Valide com o teste do *script* e teste a função para diferentes instantes de `tempo`.

b) Implemente: `novo_orientacao( orientacao real, velocidade t_velocidade, tempo real )`. Valide com o teste do *script* e teste a função para diferentes instantes de `tempo`. Note que esta função está escrita em Python (`plpythonu`).

c) Implemente: `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`.

d) Execute a vista `v_novo_cinematica`. Altere a vista para analisar outro instante de tempo.

e) Utilize o QuantumGIS para visualizar as vistas construídas.

6. Simular trajectórias

Nesta alínea deve analisar: `04_script_SIMULAR_TRAJECTORIAS.txt`.

a) Analise o *script* e inicie os dados para executar uma trajectória dos objectos 1 e 2.

b) Analise o *script* e obtenha as trajectórias ilustradas em `x_fig_duasTrajectorias.bmp`.