- Relatório Técnico -

Este é o terceiro relatório do projeto **TH-APP** e representa as atividades executadas durante o terceiro mês.

As atividades executadas foram:

- Definição de um conjunto de tabelas para armazenamento do grafo em banco de dados relacionais,
- Estudo e implementação de metodologias para busca particionada dos dados do grafo no banco de dados,
- Criação de um modelo de *cache* que permita o controle dos dados em memória.

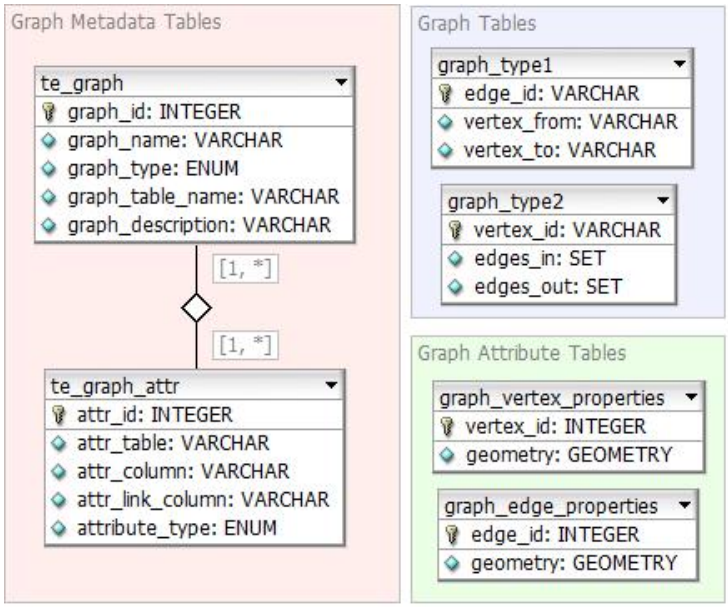
Introdução

O conceito de rede podem representar, por exemplo, as informações associadas a serviços de utilidade pública, como água, luz e telefone; e também a redes relativas a bacias hidrográficas e rodovias. Essas redes podem ser tão grandes e densas que tornam sua manipulação inviável, seja por falta de memória RAM disponível no computador ou por falta de ferramentas apropriadas. Esta etapa do projeto apresenta uma metodologia que permite o armazenamento e manipulação de grandes grafos em banco de dados geográficos através da definição de um conjunto de tabelas relacionais que descrevem os metadados do grafo e a definição de uma política de *cache* para otimizar o acesso aos dados.

Modelo de Persistência

Para que um grafo possa ser armazenado em um SGBD e posteriormente recuperado é necessário um conjunto de metadados que o descreva. Informações como características do grafo, atributos associados e como os dados estão armazenados, são de fundamental importância para sua manipulação.

A figura abaixo exemplifica o modelo adotado para armazenamento dos grafos em bancos de dados relacionais.



- *Graph Metadata Tables*: tabelas utilizadas para identificação dos grafos e atributos,
- *Graph Tables*: tabelas utilizadas para armazenamento dos dados do grafo,
- *Graph Attribute Tables*: tabelas utilizadas para associar atributos aos grafos.

Tabela de Metadado te_graph

Nome	Tipo	Descrição
graph_id	INTEGER	identificador único do grafo
graph_name	VARCHAR	nome associado ao grafo
graph_type	INTEGER	enumeral que define o tipo do grafo
Graph_table_name	VARCHAR	nome da tabela com os dados do grafo
graph_description	VARCHAR	descrição sobre o grafo

```
CREATE TABLE te_graph (
  graph_id serial NOT NULL,
  graph_name VARCHAR NULL,
  graph_type INTEGER NULL,
  graph_table_name VARCHAR NULL,
  graph_description VARCHAR NULL,
  PRIMARY KEY(graph_id)
);
```

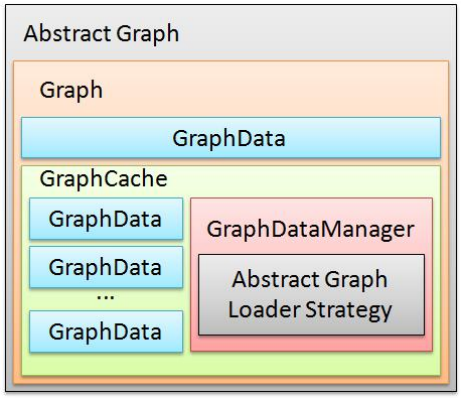
Tabela de Metadado te_graph_attr

Nome	Tipo	Descrição
attr_id	INTEGER	identificador único do atributo
attr_table	VARCHAR	nome da tabela que possui este atributo
attr_column	VARCHAR	nome da coluna que identifica o atributo
attr_link_column	VARCHAR	nome da coluna de ligação para associação ao objeto
attribute_type	INTEGER	enumeral que descreve o tipo do atributo

```
CREATE TABLE te_graph_attr (
  attr_id serial NOT NULL,
  attr_table VARCHAR NULL,
  attr_column VARCHAR NULL,
  attr_link_column VARCHAR NULL,
  attribute_type INTEGER NULL,
  PRIMARY KEY(attr_id)
);
```

Acesso a dados

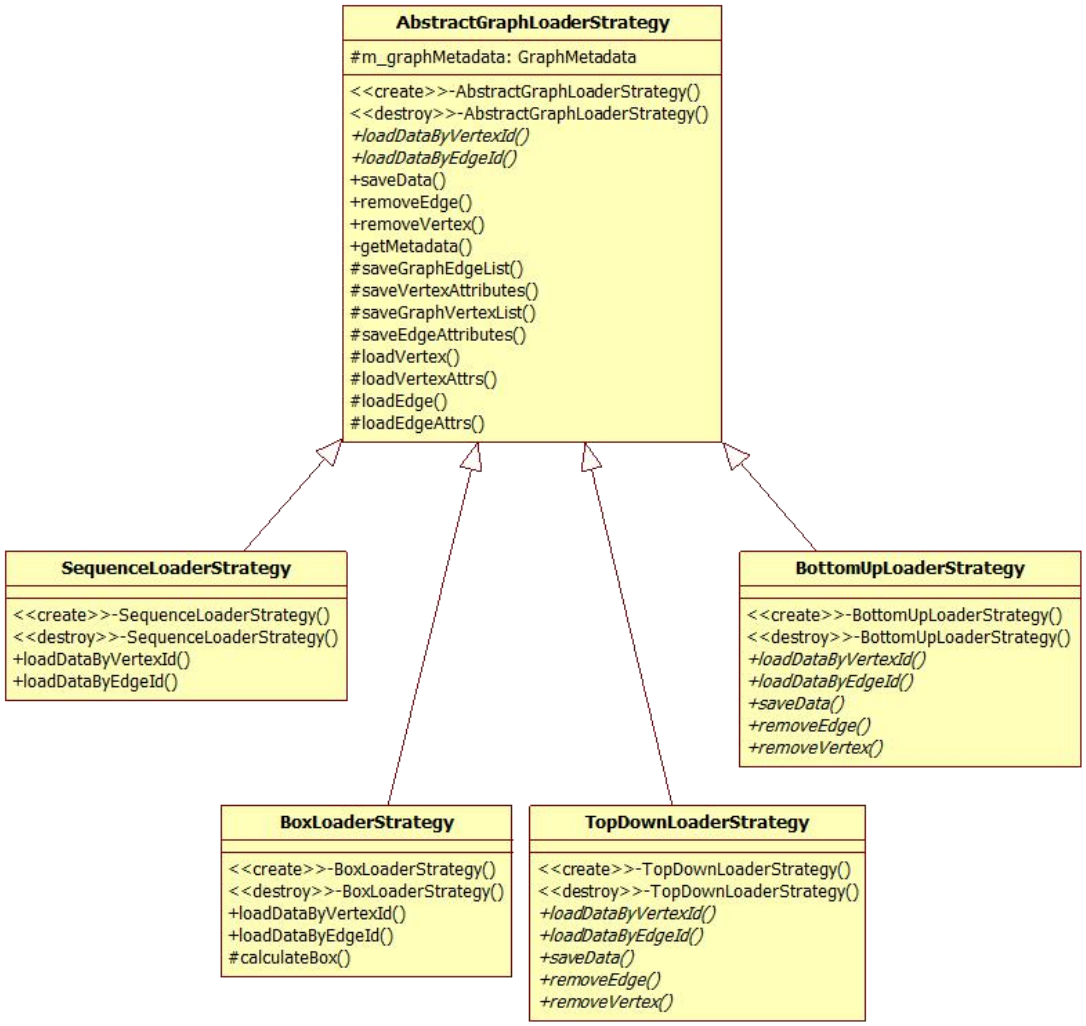
Quando um grafo esta sendo acessado pela primeira vez ou quando um elemento do grafo desejado ainda não se encontra em memória ou em *cache*, é necessário fazer a sua carga. Esse processo de carregar os dados para a memória não ocorre de elemento a elemento. Um conjunto de dados selecionados seguindo um padrão é recuperado e armazenados em **pacotes** que ficam em memória, a figura abaixo representa o encapsulamento das classes.



O padrão utilizado para carga dos dados é definido pelas estratégias de carga. As estratégias definidas são:

- *Box*: dado uma área de interesse é carregado um conjunto de objetos tendo como objeto central o elemento procurado,
- *Sequence*: dado um identificador inicial é carregado um conjunto de objetos de forma sequencial, tendo como objeto inicial o elemento procurado.

Essas estratégias são implementadas a partir de uma classe genérica, o que facilita a definição de novas estratégias, algumas outras possibilidades de carga seriam: *BottomUp* e *TopDown* que utilizariam a própria hierarquia do grafo para indicar os dados a serem carregados. A definição dessas classes são representadas na figura abaixo.

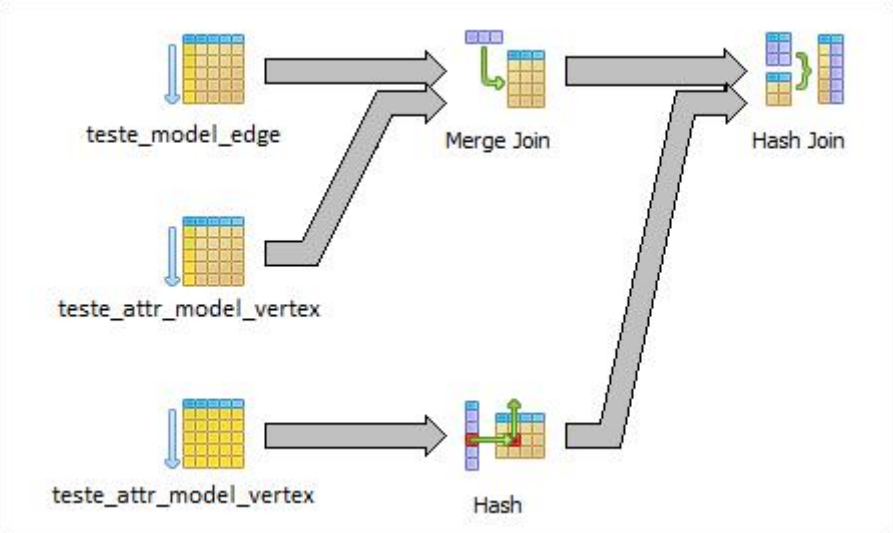


Para acessar os dados de um grafo estando armazenados em um banco de dados relacional, a recuperação é feita do seguinte modo:

Defini-se uma SQL que recupera em um único passo as informações das arestas e vértices.

```
SELECT * FROM teste_model_edge AS edges
JOIN teste_attr_model_vertex AS v1 ON (edges.vertex_from = v1.vertex_id)
JOIN teste_attr_model_vertex AS v2 ON (edges.vertex_to = v2.vertex_id)
```

A cláusula ***Where*** desta SQL é definida dependendo de qual estratégia se está utilizando, seja por *Box* ou *Sequence*. A forma de como o banco responde a esta pesquisa é mostrada na Figura abaixo:



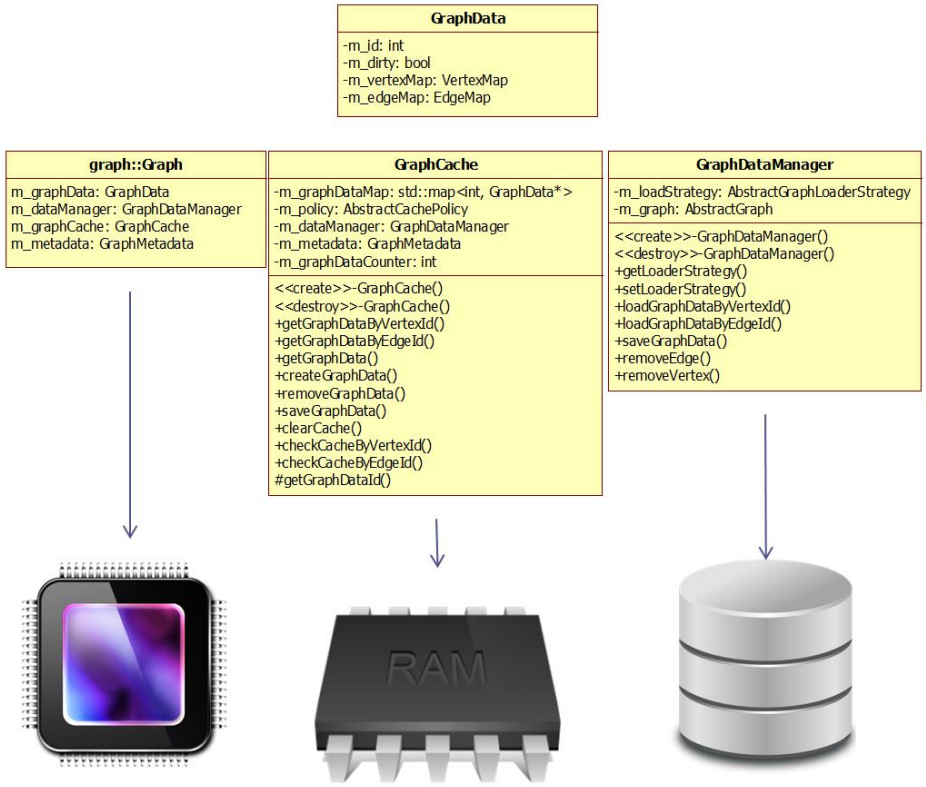
O resultado de uma pesquisa em um banco de dados relacional é uma outra tabela. Neste caso estamos fazendo uma pesquisa em duas tabelas distintas:

- teste_model_edge: contendo informações sobre as arestas e seus atributos,
- teste_attr_model_vertex: contendo informações sobre os vértices e seus atributos.

A tabela resultante irá possuir as informações das arestas e informações tanto dos vértices de origem quanto de destino. Isso é possível através de uma operação de junção entre as tabelas devido ao fato de os vértices serem representados por um identificador único.

Cache

Podemos entender o *cache* como sendo o histórico de tudo que já foi acessado. Se tivermos um bom tamanho desse histórico e quanto mais inteligente ele for ao descartar informações, melhor proveito ele terá. Fazendo uma simples analogia, podemos entender como o objeto *Graph* sendo o processador principal de um computador onde se tem acesso direto e muito rápido a um pequeno conjunto de elementos, o objeto *GraphCache* sendo a memória RAM tendo uma parte dos dados carregada e com um acesso não tão rápido e por último a classe *GraphDataManager* sendo nossa fonte de dados com um acesso lento.



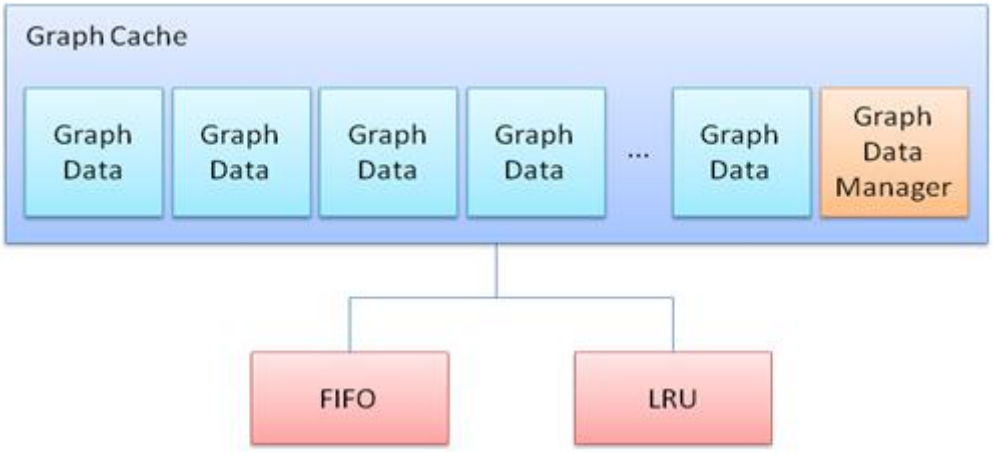
Duas observações importantes a fazer a respeito dessa estrutura de *cache* são:

- Tamanho do vetor: quanto mais pacotes o *cache* possuir, maiores são as chances de o elemento procurado estar carregado, porém em cada busca ele terá que pesquisar em mais pacotes,

- Política de *cache*: uma vez atingido o limite máximo de pacotes é necessário a remoção de pacotes da memória. Duas estratégias de policiamento são definidas:

- *FIFO (First In First Out)*: conceito de uma fila comum, o primeiro pacote a entrar será o primeiro pacote a sair,

- *LFU (Least Frequency Used)*: pode ser visto como uma fila também, só que a cada vez que um pacote é usado ele é movido para o começo da fila e as remoções são feitas no fim da fila.



Essa política passa a ser simples devido ao fato de os elementos estarem agrupados em pacotes. O controle é feito por pacotes e não por elementos individuais, evitando uma sobrecarga de checagens e controles.