A *framework* `data-adapter` tem a capacidade de criar uma instância de uma implementação de `DataMapper` para uma determinada *entidade de domínio* (ED) tirando partido do serviço de reflexão (`java.lang.reflect`).

Para tal, pressupõe-se que cada ED tem uma tabela correspondente numa base de dados relacional.

Considere o seguinte exemplo de criação de um *data mapper* para a ED `Product` (com uma tabela correspondente `Products`) através da classe `Builder` integrante da *framework* `data-adapter`:

```
Builder b = new Builder(...);
DataMapper<Product> prodMapper = b.build(Product.class);
Iterable<Product> prods = prodMapper.getAll();
```

A *framework* `data-adapter` suporta o encadeamento de cláusulas de `where` sobre o resultado de um `getAll`. Para tal a interface `DataMapper` obedece à especificação da Figura 2.

```
public interface DataMapper<T>{
  SqlIterableImpl<T> getAll();
  void update(T val);
  void delete(T val);
  void insert(T val);
}
```

```
public interface SqlIterable<T> extends Iterable<T>, AutoCloseable{

    SqlIterable<T> where(String clause);
    int count();
}
```

Figura 1

Exemplo de utilização:

```
int nrOfProducts = prodMapper
     .getAll()
     .where("UnitPrice > 15.5")
     .where("UnitsinStock > 5")
     .count();
```

NOTA: o resultado do `where` é avaliado de forma **Lazy** modificando em *runtime* a *query* que será executada sobre a base de dados.

A cláusula especificada pode incluir parâmetros que podem ser ligados (*bind*) a diferentes argumentos em tempo de execução, conforme apresentado no exemplo seguinte.

```
SqlIterable<Product> res = prodMapper
                .getAll()
                .where("UnitPrice > ?")
                .where("UnitsInStock > ?");

int count = res.bind(20, 10).count();
assertEquals(30, count);

count = res.bind(30.8, 5).count();
assertEquals(21, count);
```

A *framework* `data-adapter` dá suporte para a associação (simples e múltipla) entre EDs, que estão relacionadas por *foreign key* na base de dados.

Uma propriedade/campo de uma ED pode ser do tipo de outra ED (associação **simples** 1-1), ou do tipo `Iterable<ED>` (associação **múltipla** 1-*).

Por exemplo: `Product` pode ter uma propriedade do tipo `Category` (associação simples 1-1) e `Supplier` pode ter uma propriedade do tipo `Iterable<Product>` (associação múltipla 1-*).

A associação simples entre EDs é *lazy* através da utilização de um *dynamic proxy* para o objecto associado.

A implementação da *framework* `data-adapter` deve obedece aos seguintes requisitos:

- A classe que define a ED deve estar anotada com a informação do nome da tabela correspondente.

- Por omissão, considera-se que o nome de cada propriedade pública da ED corresponde ao nome de uma coluna da respectiva tabela.

- Na instanciação de `Builder` o programador deve poder especificar o tipo de mapeamento pretendido entre a ED e as colunas da tabela: se baseado no nome dos campos da ED; se baseado no nome das propriedades da ED; ou outro mapeamento qualquer que seja implementado à posteriori, sem necessidade de alterar o código da *framework*.

- **A**dmite-se que o tipo do campo/propriedade da ED mapeado é compatível com o tipo do valor da coluna correspondente obtido via JDBC, **NÃO** sendo necessário implementar nenhum suporte de conversão entre tipos.

- Os campos/propriedades da ED que corresponderem a colunas de chave primária devem estar anotados com essa informação.

- Admite-se que a *framework* só suporta tabelas com PK do tipo *identity*, NÃO sendo necessário implementar suporte para outro tipo de chaves. OPCIONAL: suportar chaves não *identity* e compostas.

- Na instanciação de `Builder` o programador deve poder especificar a política de gestão de ligações (`SqlConnection`), ou seja, se é reutilizada a mesma ligação em diferentes execuções dos métodos do *data mapper*, ou se é criada uma nova ligação em cada execução de cada método, ou outra estratégia qualquer de gestão de ligações que seja implementada à posteriori.

- Política de gestão de ligações com suporte para iniciar e finalizar uma transacção através *rollback* ou *commit* explícito.

- O iterador retornado por `getAll` usa uma implementação **lazy**, que só faz *fetch* do `ResultSet` à medida que é iterado.

- Todos os recursos são fechados (*closed*/*disposed*) após terminada a iteração de todos os elementos, excepto numa política de *singleton connection* onde a ligação partilhada só deve ser fechada por ordem do programador/utilizador do `data-adapter`.