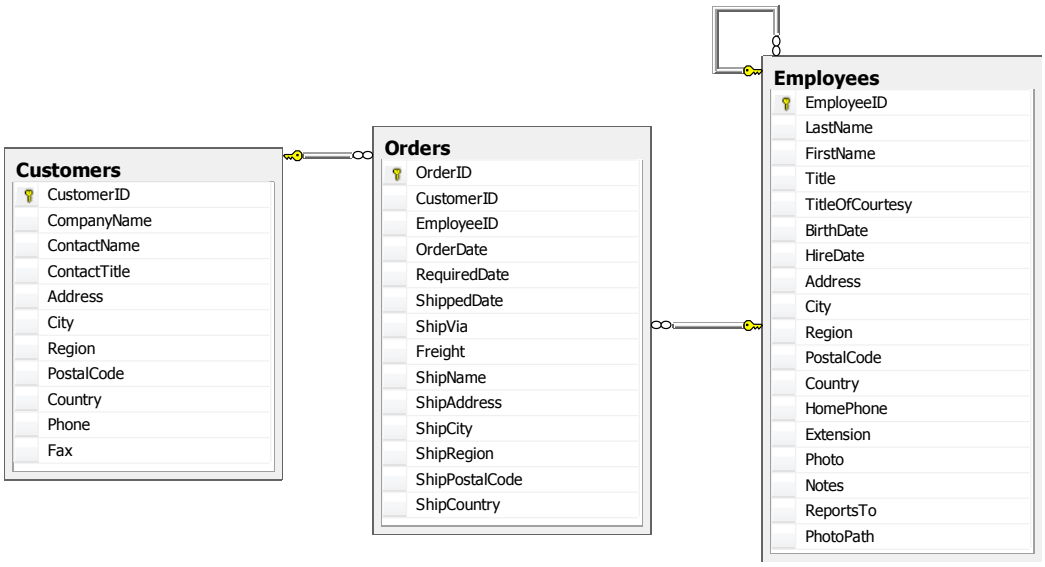
O objectivo deste trabalho é desenvolver uma *framework* `sqlmapper` capaz de criar uma instância de uma implementação de `DataMapper` para uma determinada *entidade de domínio* (ED) tirando partido do serviço de reflexão (`java.lang.reflect`).

Para tal, pressupõe-se que cada ED tem uma tabela correspondente numa base de dados relacional.
A implementação do trabalho deve ser validada no **MÍNIMO** com testes unitários para as EDs correspondentes às seguintes tabelas (contudo a *framework* suportará a geração de *data mappers* para qualquer tabela além destas).



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* `sqlmapper`:

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

A interface `DataMapper` obedece à seguinte definição, onde `T` representa o tipo da ED.

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

Figura 1

A *framework* `sqlmapper` deve ser implementada em **três partes, incrementalmente**.

- Leia ATENTAMENTE TODOS os requisitos incluindo os TRANSVERSSAIS às 3 partes
- NÃO implemente SIMULTANEAMENTE 2 alíneas
- Passe à alínea seguinte APENAS quando tiver CONCLUÍDA a alínea anterior
- A implementação de cada alínea deve ser validada com testes unitários que confirmem a sua correcção.
- Exige-se a resolução das alíneas opcionais para ter avaliação no trabalho acima de 17 valores.
- **DATA LIMITE DE ENTREGA**: 7 de Julho de 2014, incluindo relatório.

1ª parte: sem associação entre EDs.

1. Implemente a *framework* de acordo com a especificação da interface `DataMapper` da Figura 1.

2. Altere a *framework* `sqlmapper` de modo a suportar o encadeamento de cláusulas de `where` sobre o resultado de um `getAll`. Para tal a interface `DataMapper` passa a obedecer à nova 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 2

Exemplo de utilização:

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

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

2ª parte: associação entre EDs

Adicione 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-*).

3ª Parte: *binding*

Altere a implementação do método `where` de `ISqlEnumerable` de modo a que cláusula especificada possa 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);
```

4ª Parte: associação simples *lazy* através de *dynamic proxy* (Opcional).

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

Requisitos Transversais a todas as Partes do Trabalho

A implementação da *framework* `sqlmapper` deve obedecer 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*.
- **Pressuposto**: admite-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.
- Deve ser ainda implementada/disponibilizada uma política de gestão de ligações com suporte para iniciar e finalizar uma transacção através *rollback* ou *commit* explícito.
- O iterador retornado por `getAll` deve usar uma implementação **lazy**, que só faz *fetch* do `ResultSet` à medida que é iterado.
- Deve-se garantir que 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 `sqlmapper`.
- Como tecnologia de ligação à base de dados poderá ser usado **apenas JDBC** e o package `java.sql`. **NÃO pode ser usado o package `javax.sql` ou qualquer outro para ligação à base de dados.**
- **São valorizadas as soluções que tenham cuidados de eficiência da solução. Por exemplo, entre outros aspectos, deve:**
    - **maximizar o uso de reflexão na construção (*build*) do *data mapper* e minimizando (a zero se possível) o uso de reflexão na execução dos métodos do *data mapper*.**
    - **obter os dados de forma *lazy*, sempre que possível.**