Universidade Federal do ABC

**Bacharelado em Ciência da Computação**

**Banco de Dados (MC 3310)**

**PoGym -  ferramenta auxiliar para batalhas no jogo “Pokémon Go”**

Felipe Rigo Yoshimura RA:11032811

Lenin Cristi RA: 21030114

Victor Fernandes RA: 11043616

30/11/2016

1. **Objetivo**

O vigente projeto tem como objetivo a criação de um banco de dados que contenha informações sobre o jogo “Pokémon GO”, onde será possível fazer variadas consultas para auxiliar o usuário a tomar decisões estratégicas dentro do jogo, entre os objetivos principais estão:

* Permitir consultar a listagem de pokemons e seus respectivos ataques para inserção na ferramenta;
* Dado o número do pokemon e seus dois ataques, retornar dados sobre os mesmos;
* Permitir gerar as tabelas de multiplicadores de vantagem/desvantagem por tipo dados quaisquer dois tipos de um defensor;

1. **Modelo Conceitual Ajustado / DER ajustado**

Como pode ser observado nas figuras 1 e 2, foram realizadas alterações no modelo conceitual do projeto

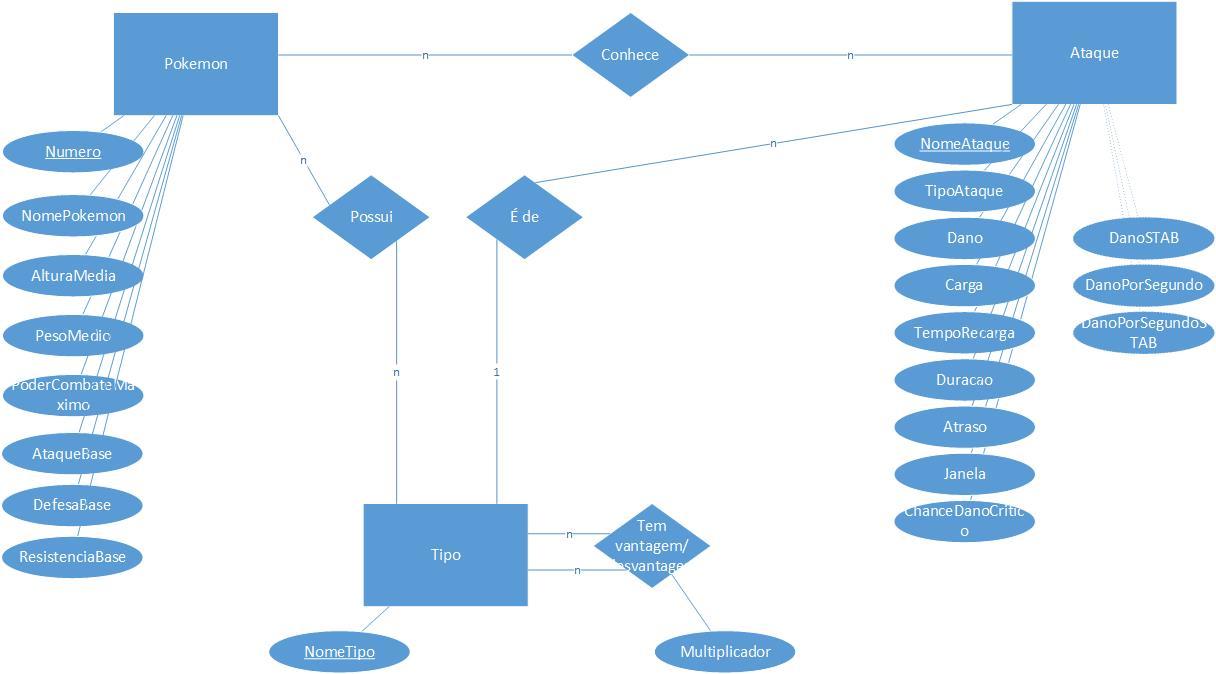


Figura 1: Diagrama original

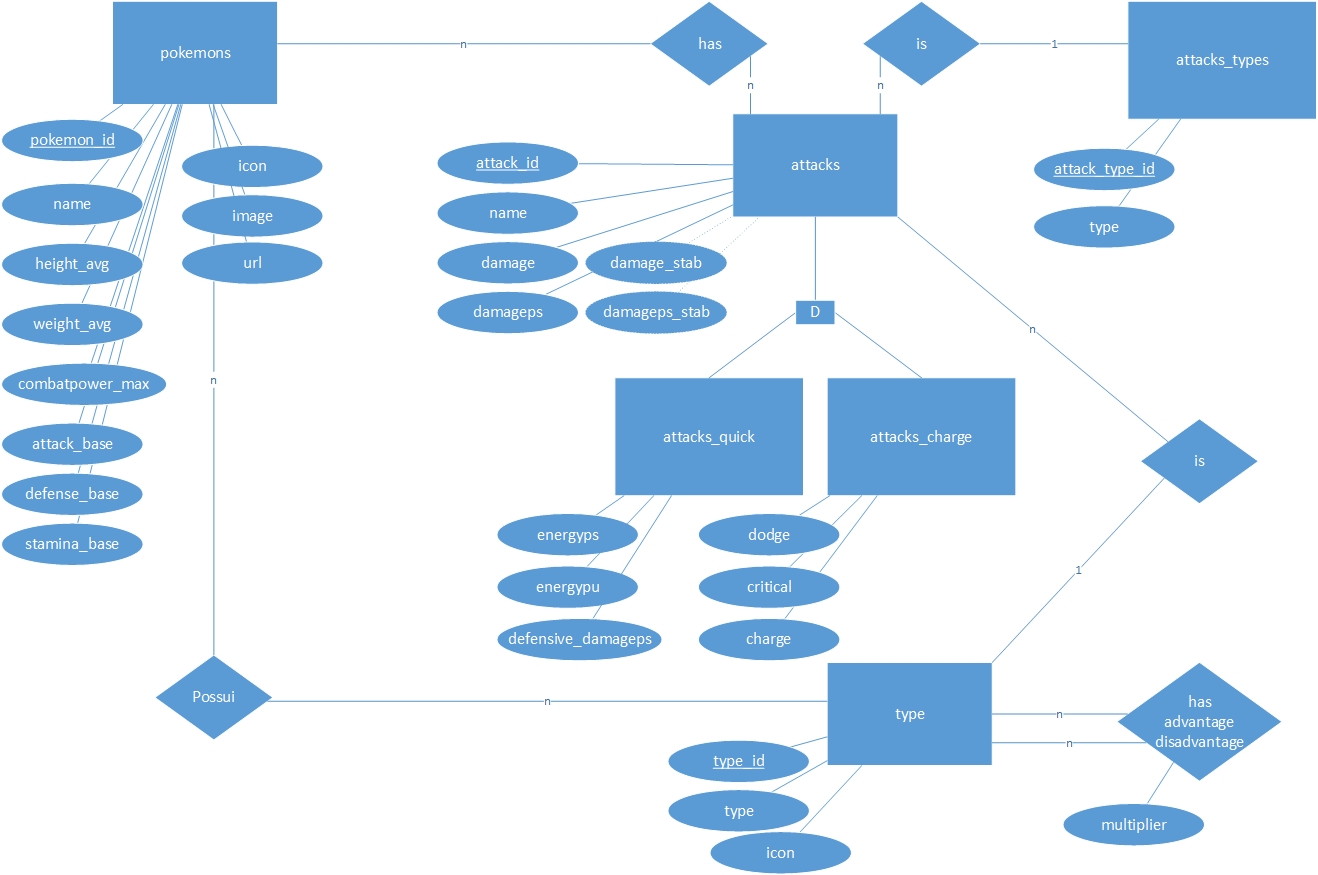


Figura 2: Diagrama alterado

1. **Modelo Relacional**

A partir do DER pôde-se construir o modelo relacional mostrado abaixo:

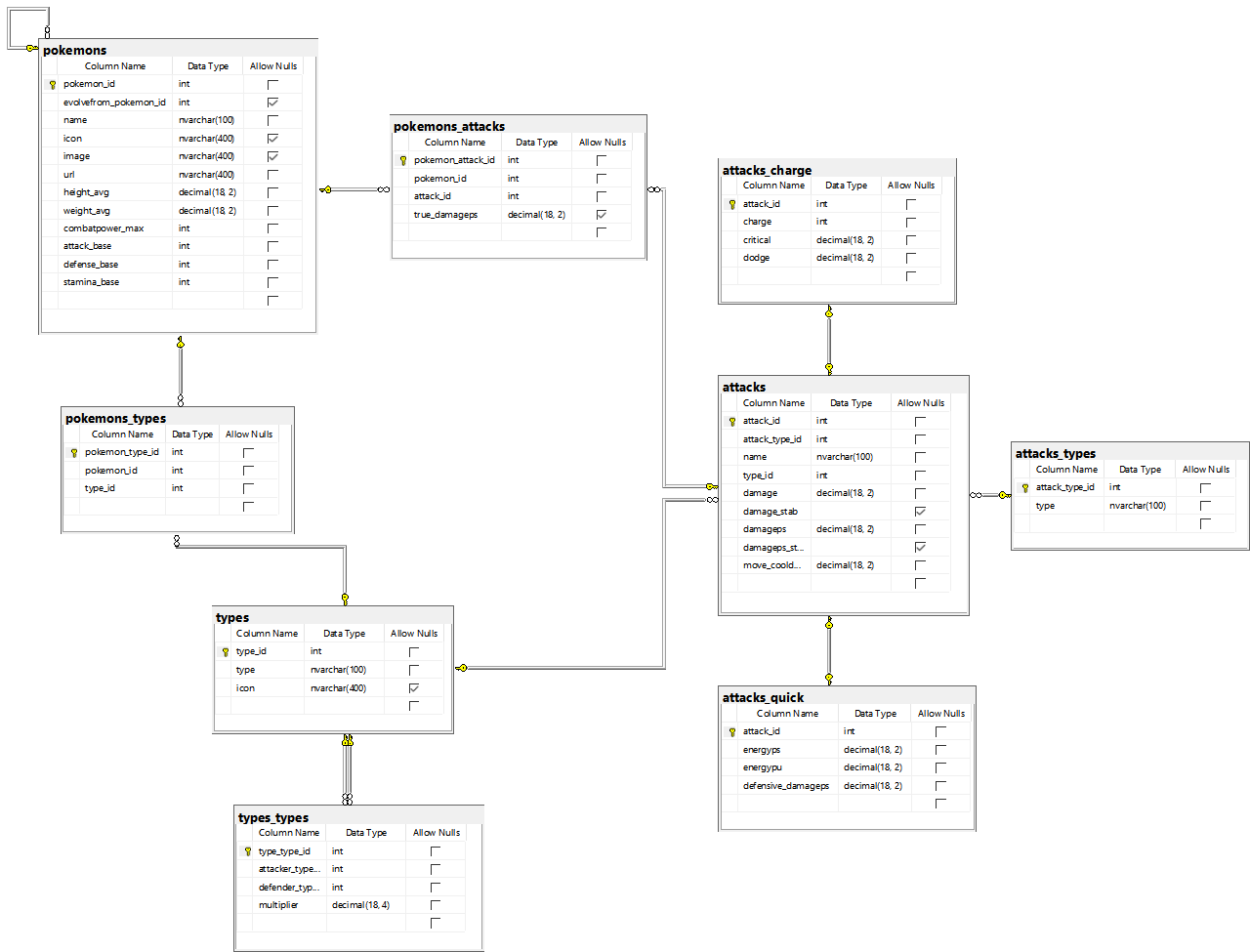


Figura 3: Modelo ER

1. **Normalização**

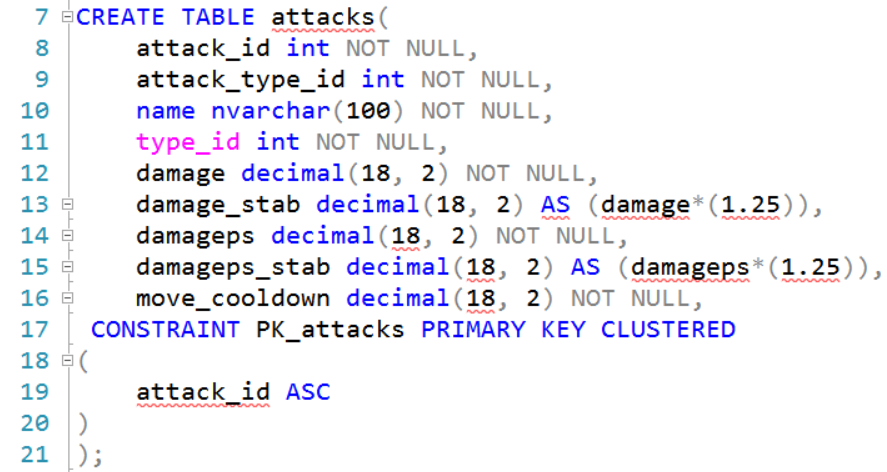
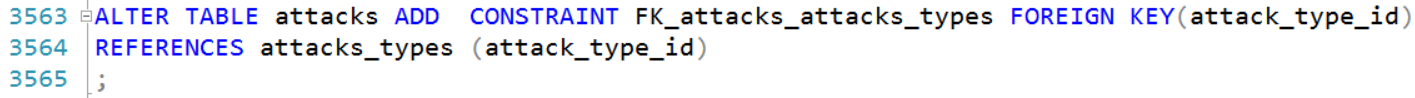
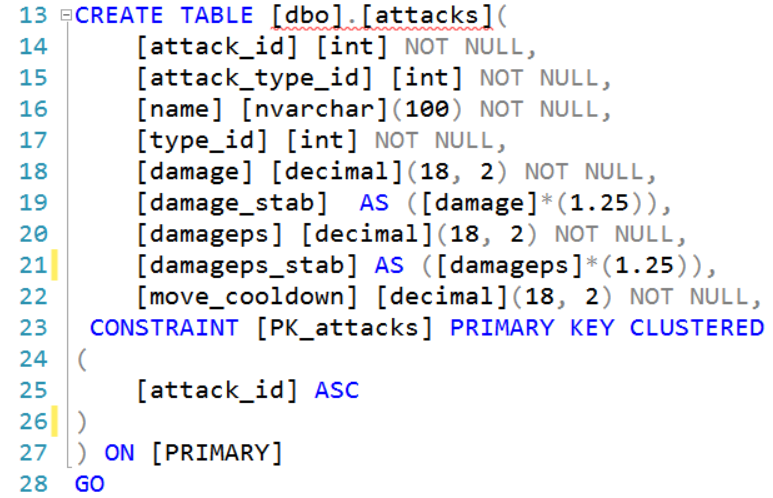
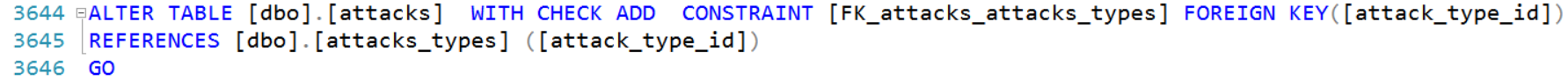
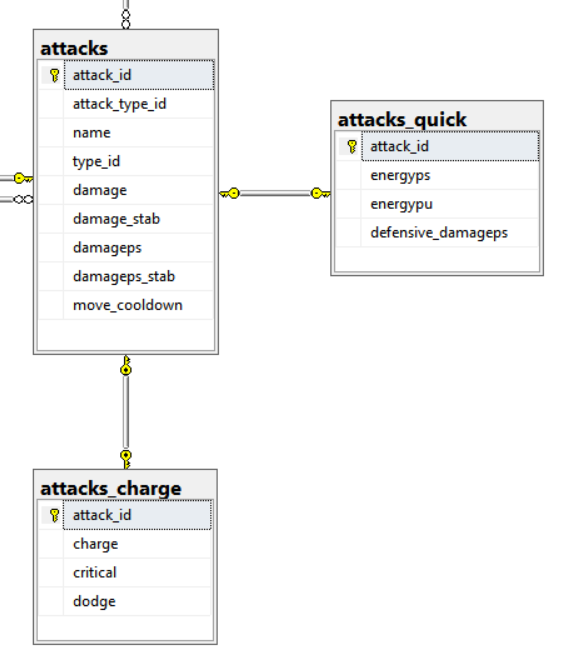
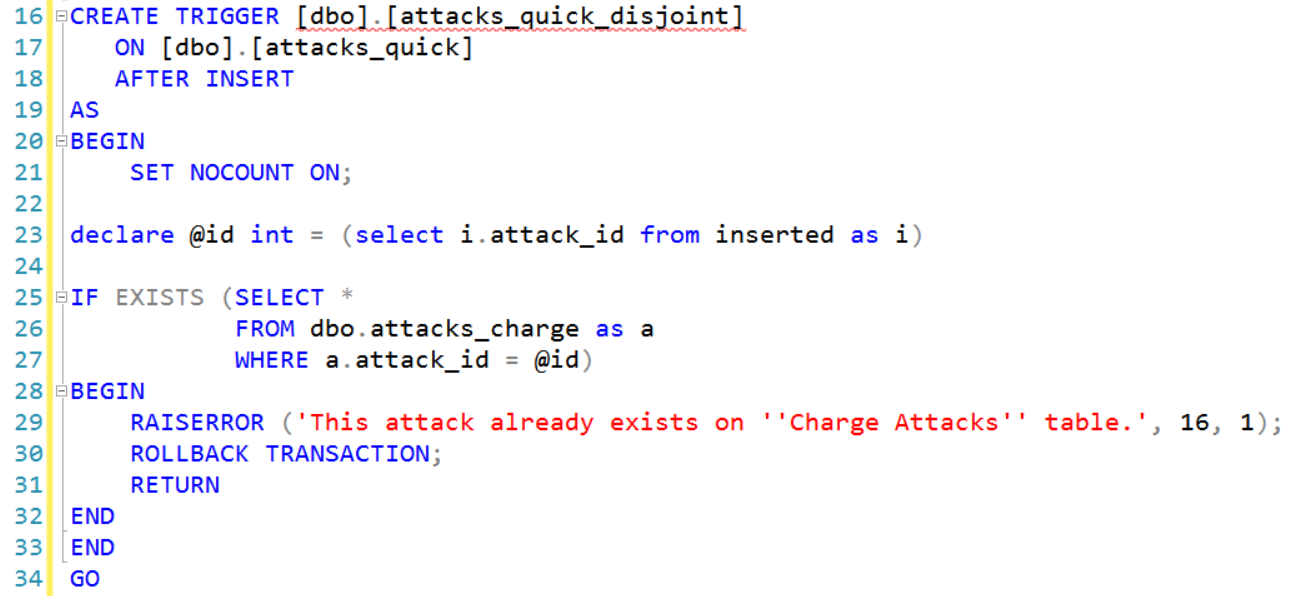
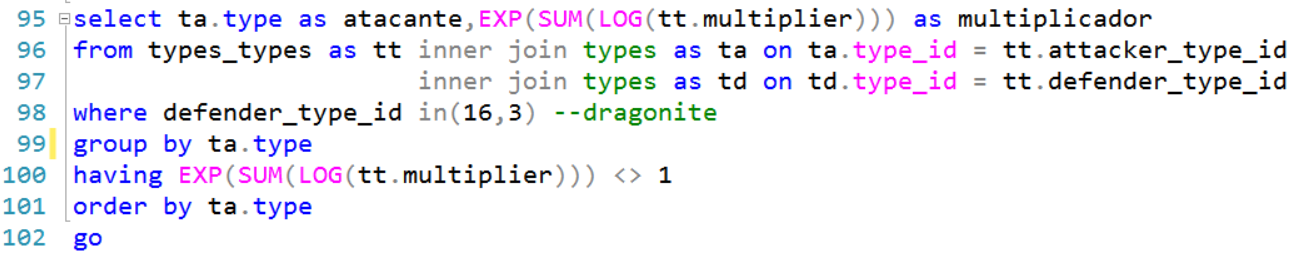
A revisão do desenho do DER permitiu eliminar boa parte do trabalho de normalização, vários passos como a verificação de dependência funcional com relação às chaves primárias foram adiantados, mas pode se fazer duas observações como exemplos de destaque na aplicação de regras de normalização:

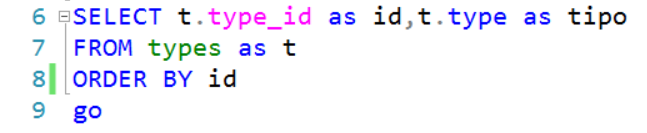
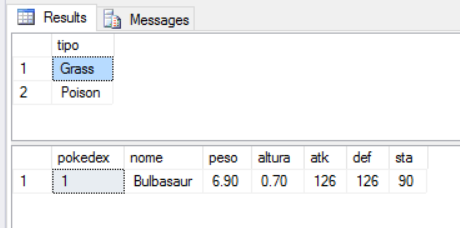
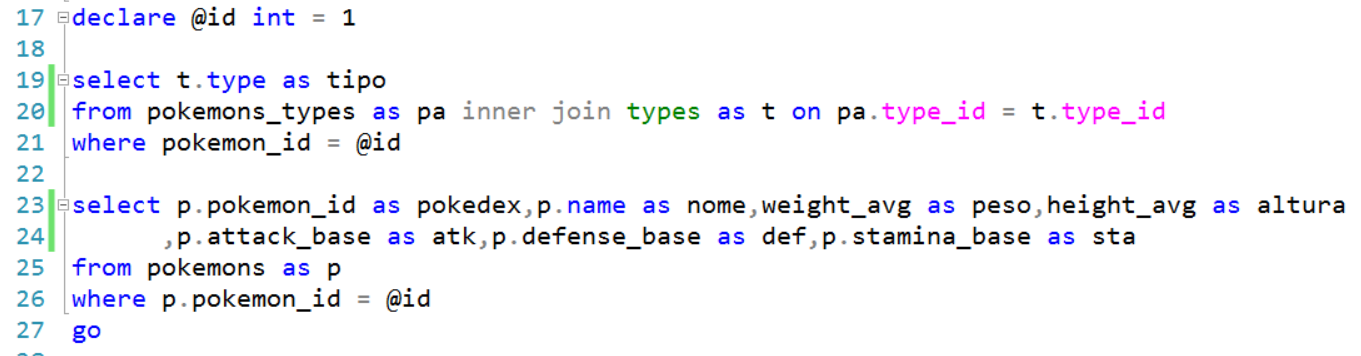
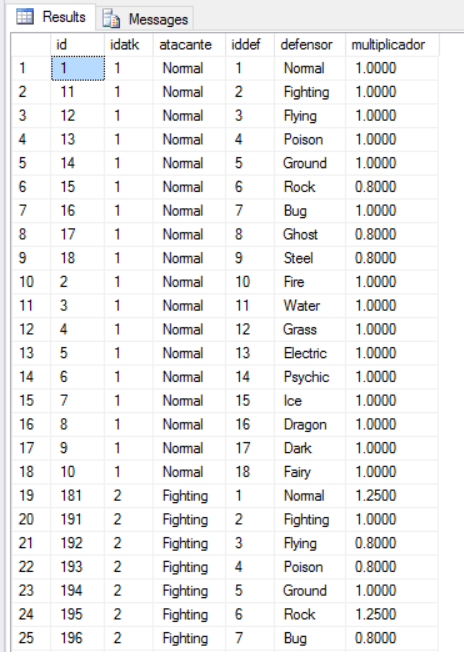
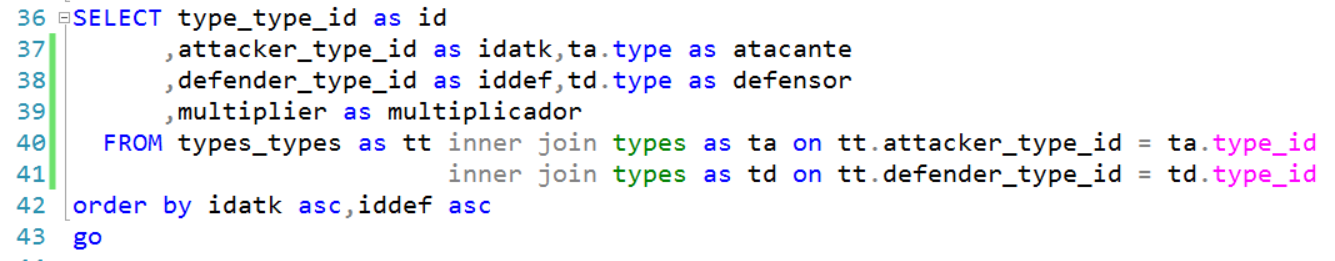
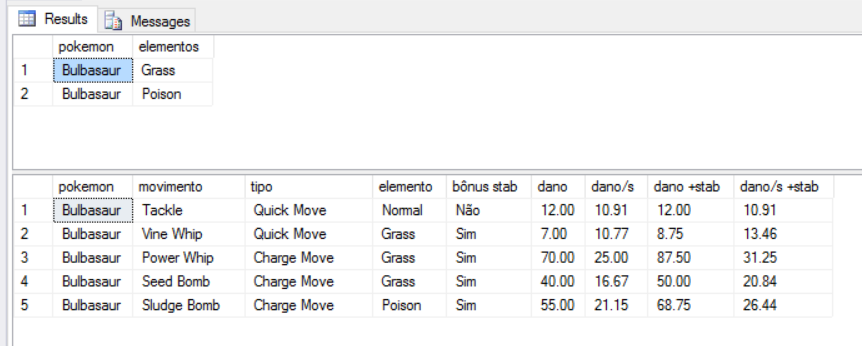
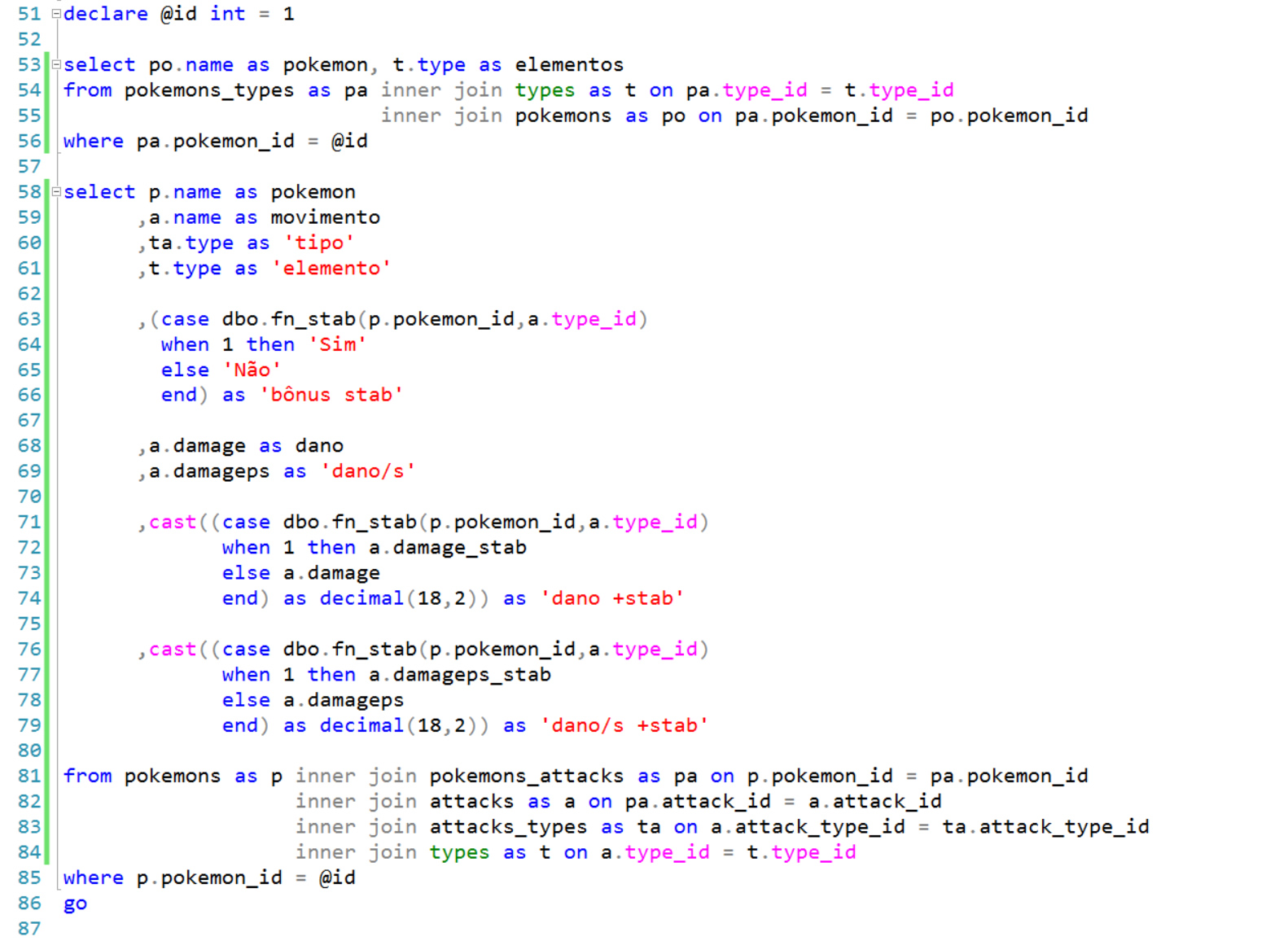
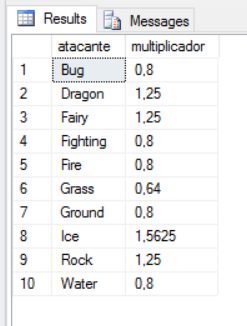
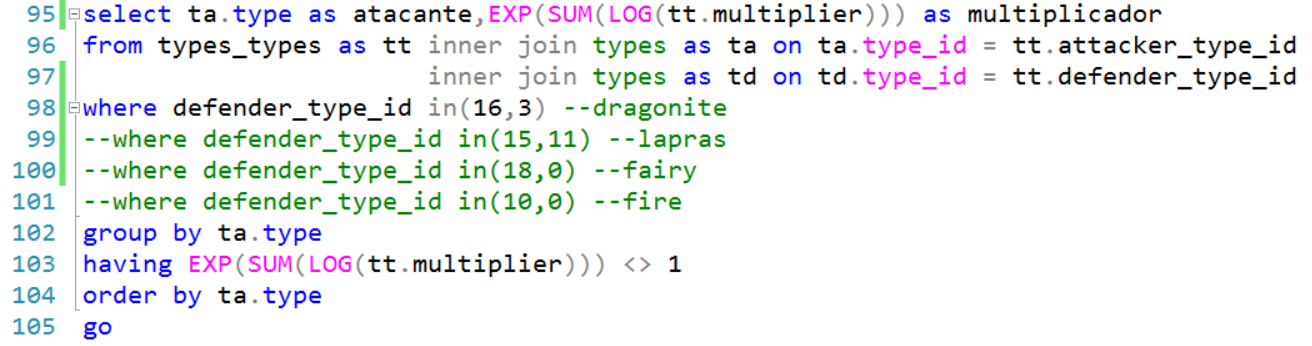
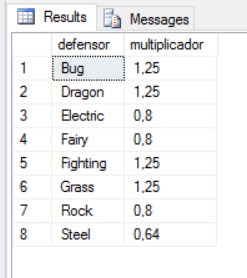
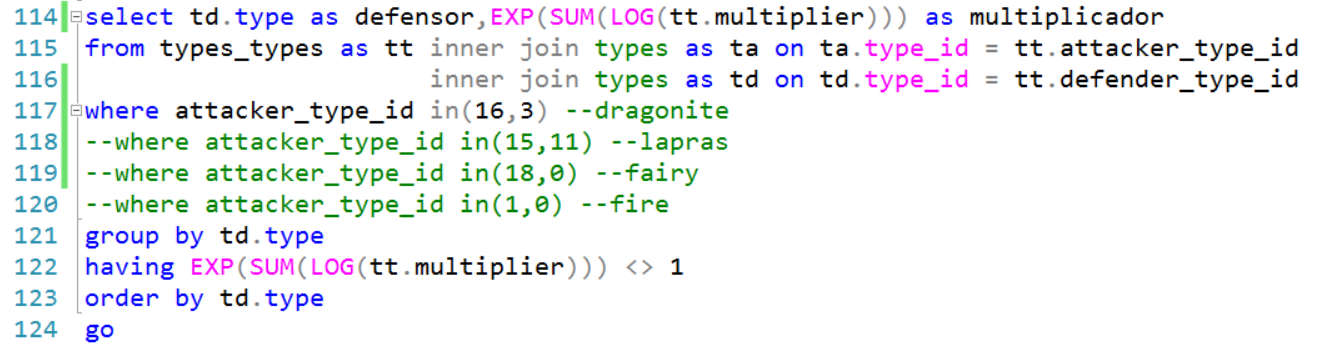
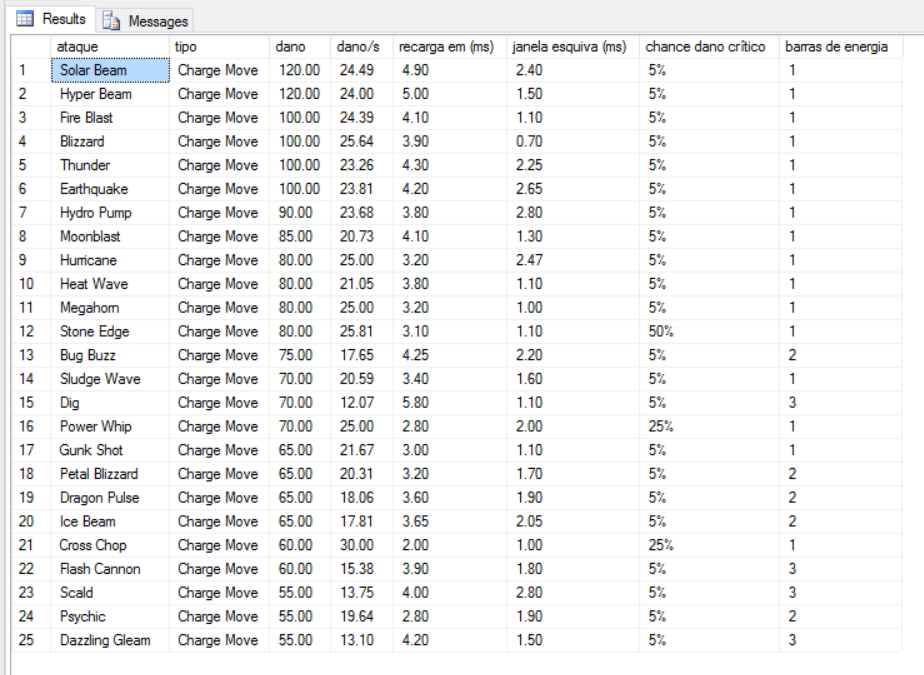
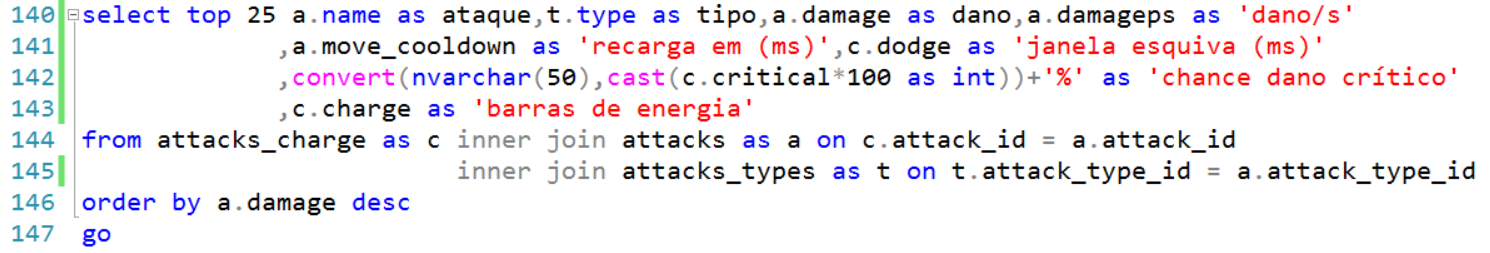
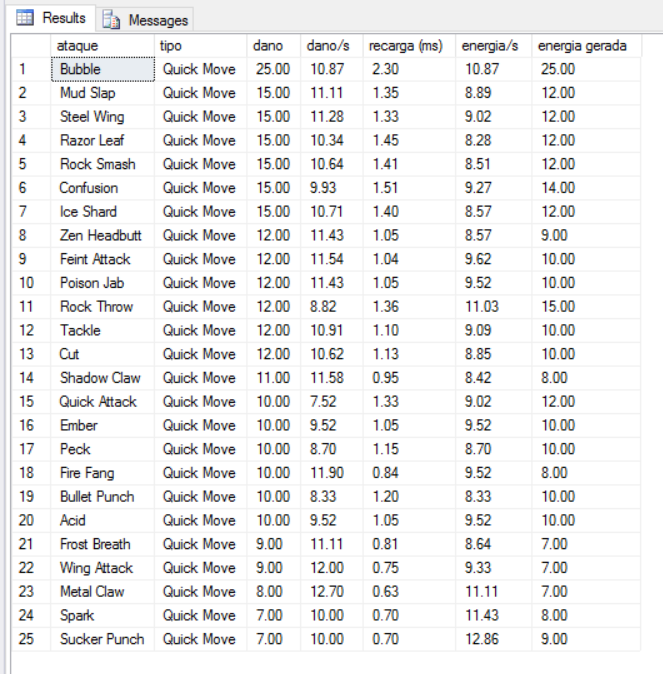
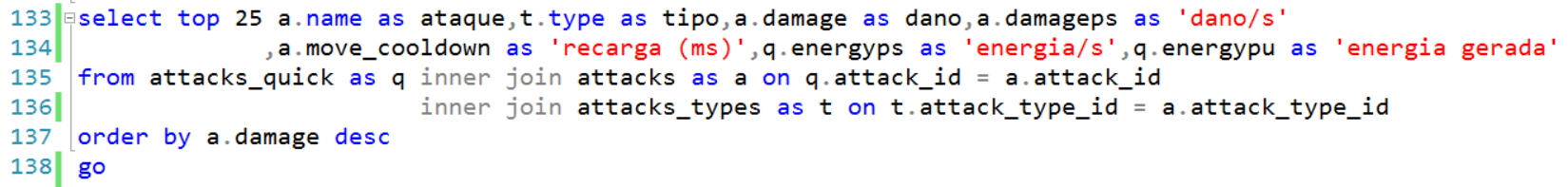
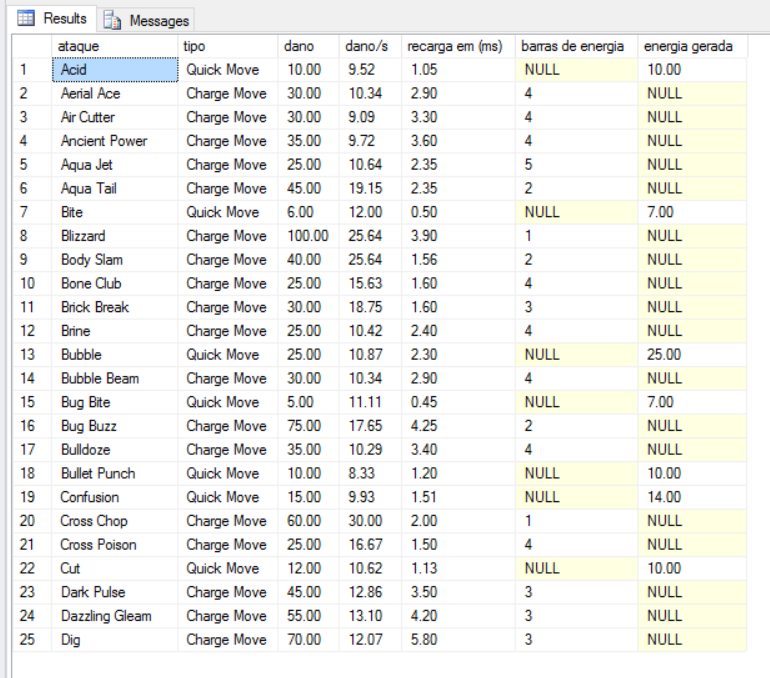
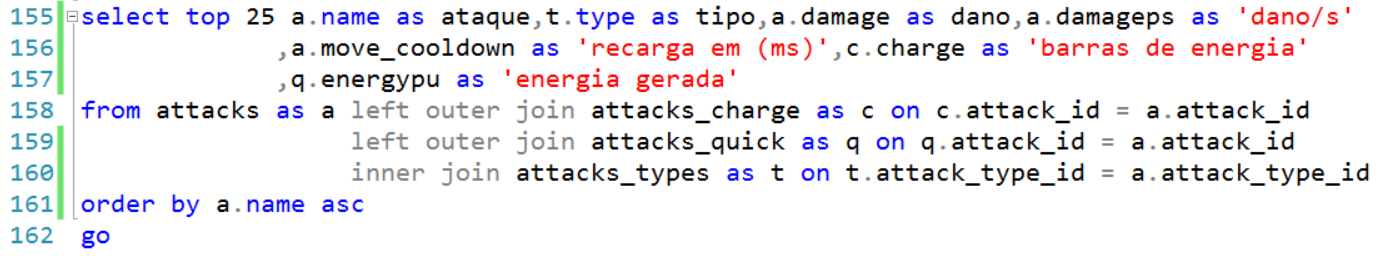
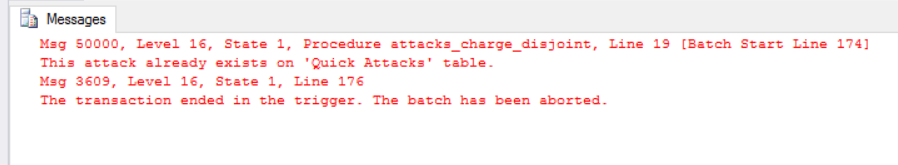
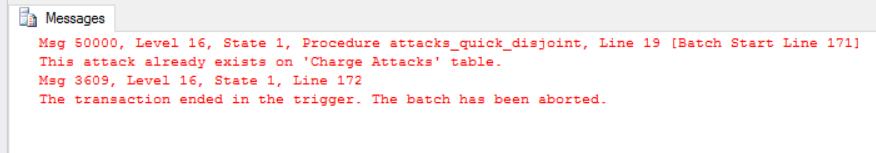
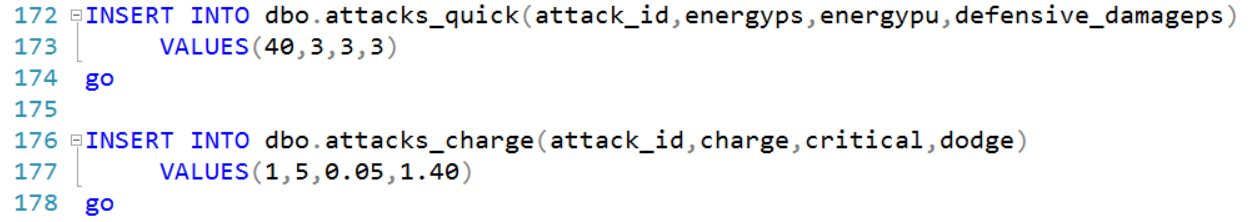
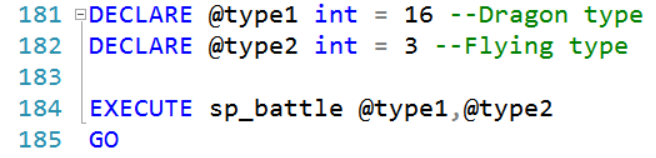
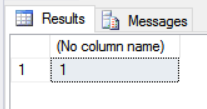
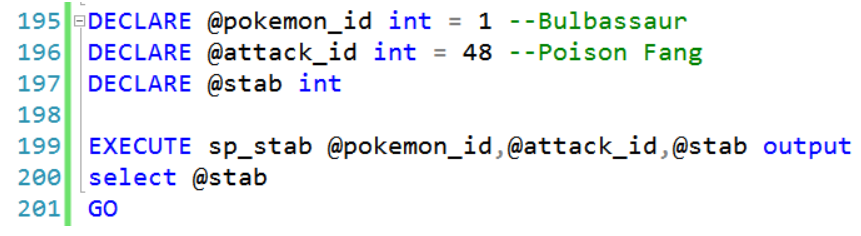
* A tabela “Ataque” possuía um atributo chamado “Tipo de ataque” que poderia assumir dois valores: “Ataque rápido” ou “Ataque carregado” no primeiro diagrama. No segundo, esse campo foi removido para uma entidade separada “Tipo de ataque”;
* A tabela “Ataque” possuía atributos utilizados somente em ataques rápidos, atributos utilizados somente em ataques carregados e atributos utilizados nos dois casos, um caso de uso da especialização, que gerou duas subclasses “Ataque rápido” e “Ataque carregado”;

1. **Modelo Físico**

Para a implementação do modelo físico utilizou-se o SGBD SQL Server, mas também foram gerados na pasta do projeto scripts adaptados para a criação das relações e carga de dados do banco em MySQL, exceto as duas procedures (sp\_battle e sp\_stab) e a função (fn\_stab) utilizadas no projeto.

Considerações importantes nos scripts SQL Server e MySQL:

* O MySQL chama o banco de “schema”, no SQL Server essa nomenclatura é um objeto utilizado para organizar objetos dentro do banco. Isso fez necessário remover os nomes de “schema” do script para utilização no MySQL;
* O MySQL pela questão do “schema” não utiliza chaves “[,]” para delimitar objetos;
* Os campos calculados têm criação pouco diferente:
  + SQL Server
  + MySQL  (Essa construção para o MySQL só funciona a partir da versão de banco 5.7)
* Os tipos de campos não mostraram problemas e foram reconhecidos nas duas plataformas;
* As diferenças na criação de chaves e relações também foram mínimas:
  + Relação “Ataques” no MySQL 
    - Sua Chave estrangeira com a relação “Ataques tipos” 
  + Relação “Ataques” no SQL Server 
    - Sua Chave estrangeira com a relação “Ataques tipos” 
* Foi no SQL Server implementado o “disjoint” entre as tabelas “Ataques rápidos” e “Ataques carregados” com trigger, existia a possibilidade de utilizar o campo chave de “Ataques” junto ao campo chave de “Ataques tipos” como chave composta na relação e adicionar uma restrição no segundo, mas optamos por economizar um atributo nas subclasses e implementar via trigger que checa a existência da chave de “Ataques” na outra relação do disjoint:
  + Modelo do disjoint  
     
  + Trigger na tabela “Ataque rápido”
  + O trigger na tabela “Ataque carregado” é idêntico, mas invertem-se as tabelas na lógica
* A consulta de batalha agrega os multiplicadores, mas como não tínhamos esta função para uso na linguagem SQL implementada nos bancos, tivemos de utilizar uma construção (EXP+SUM+LOG) para agregar os multiplicadores multiplicando-os  Esta construção funciona em ambas as plataformas

1. **Consultas**
   1. **Select simples para listagem de tipos por sua ordem natural no jogo**
   2. **Selects que listam o(s) tipo(s) e ficha de um dado pokemon** 
   3. **Select que mostra a relação da tabela tipos pra ela própria com o atributo de relação denominado 'multiplicador'**
   4. **Selects que listam o(s) tipo(s) e ataques possíveis de um dado pokemon**
   5. **Select que combina os dois tipos de um pokemon SE DEFENDENDO e gera a tabela de multiplicadores (tabela do site gamepress)**
   6. **Select que combina os dois tipos de um pokemon ATACANDO e gera a tabela de multiplicadores** 
   7. **Selects que listam a tabela de ataques e UMA de suas disjuntas com registros ordenados do maior ao menor dano**
   8. **Select que lista a tabela de ataques JUNTO a suas disjuntas com registros ordenados por nome** 
   9. **Inserção de registro de ataque nas tabelas attacks\_quick e attacks\_charge para demostrar o disjoint entre elas**
   10. **Execução da procedure de batalha para gerar a tabela de multiplicadores**
   11. **Execução da procedure de verificação de bônus STAB**
2. **Conclusão**

A ferramenta de consumo e apresentação dos dados foi feita utilizando a tecnologia ASP.NET MVC 5.2.3 da plataforma .NET 4.6.1 da microsoft, a linguagem utilizada foi C# , o acesso a dados utiliza tecnologia Entity Framework 6.1.3, um serializador/deserializador JSON Newtonsoft.JSON 9.0.1 e finalmente a apresentação utiliza saída HTML5/CSS3 pura com os frameworks Bootstrap 3.3.7 e jQuery 3.1.1. O Bootstrap a partir da versão 3 é mobile-first.

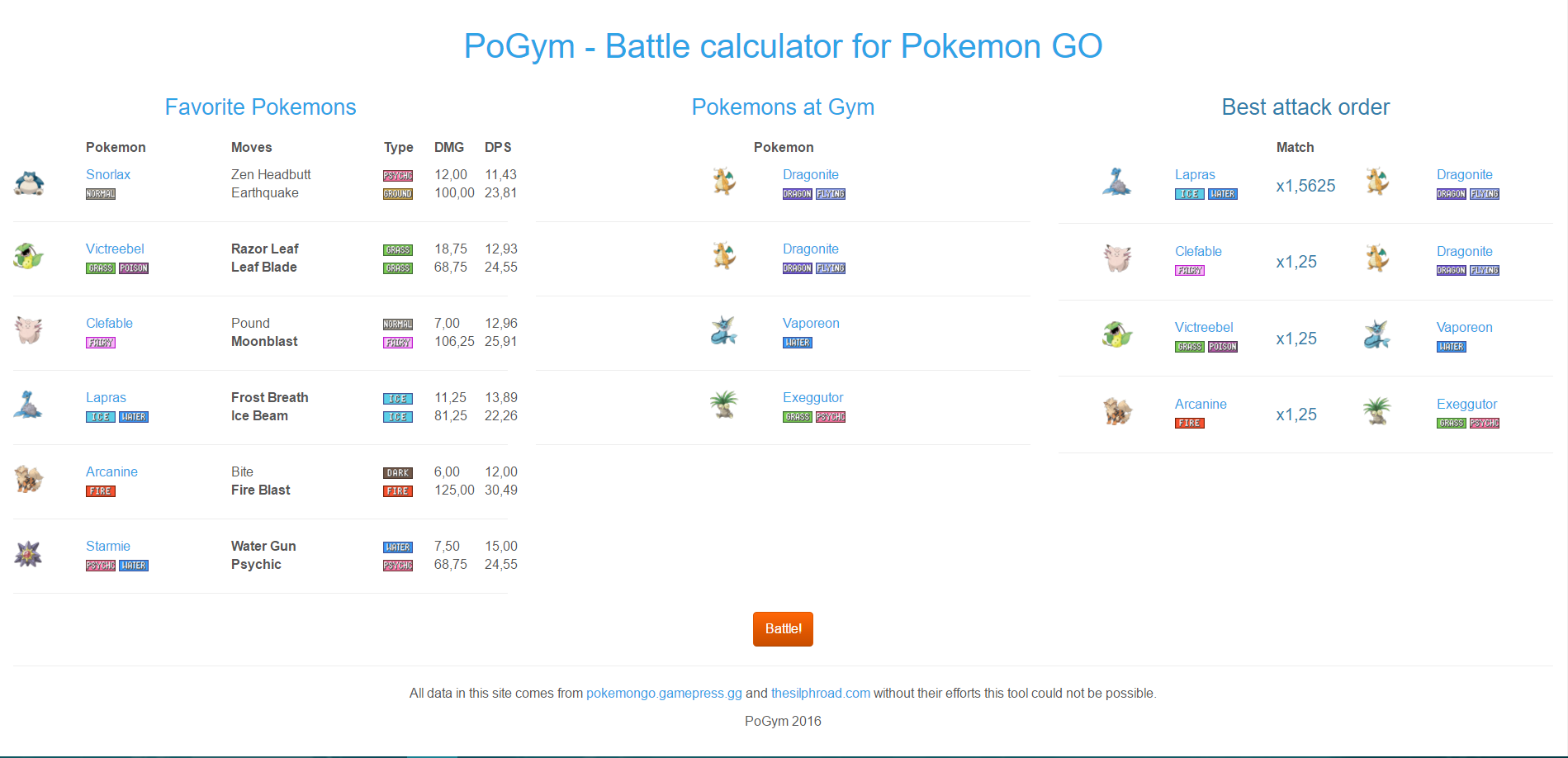


Figura 4: Ferramenta em execução