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**

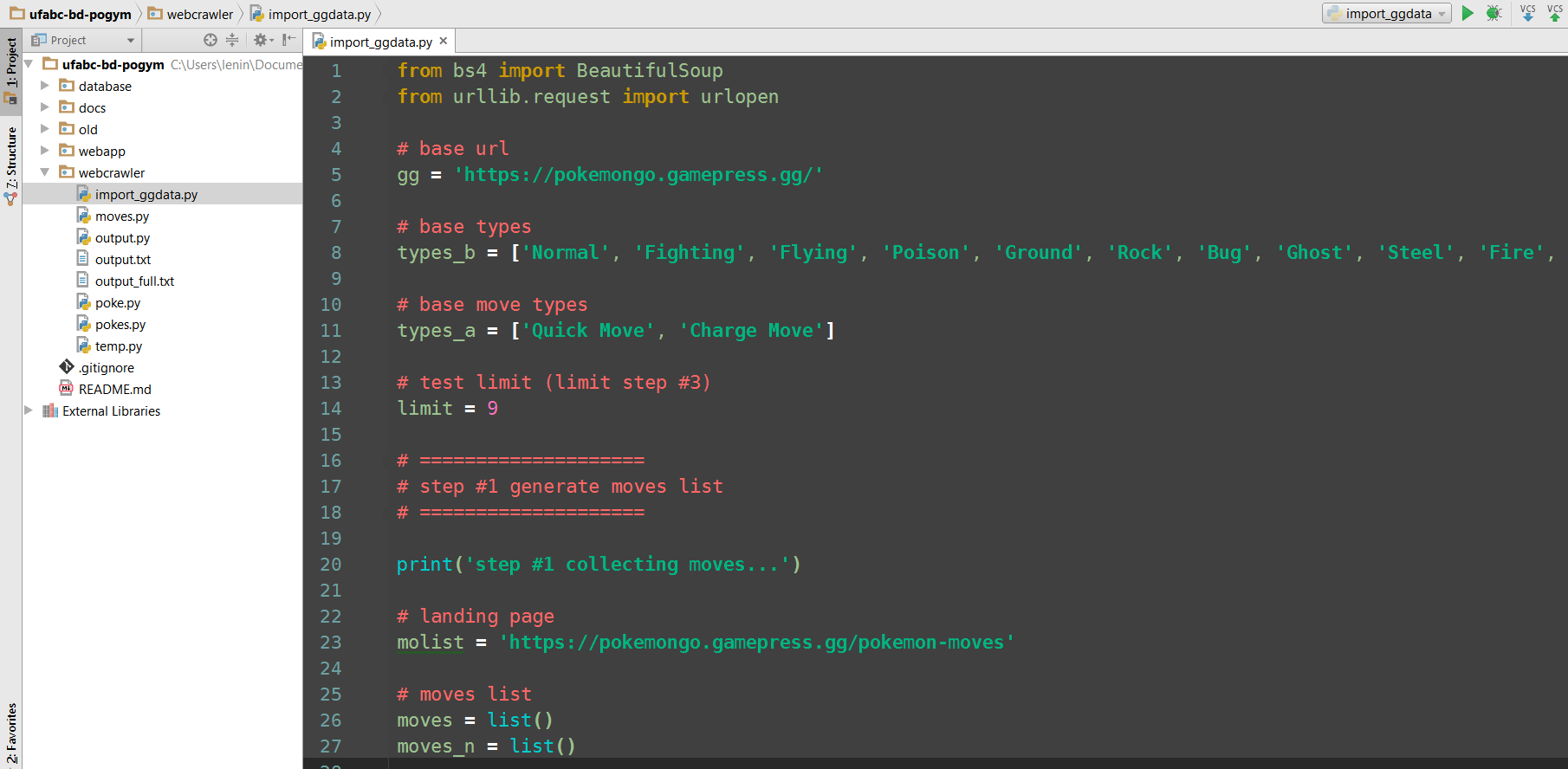
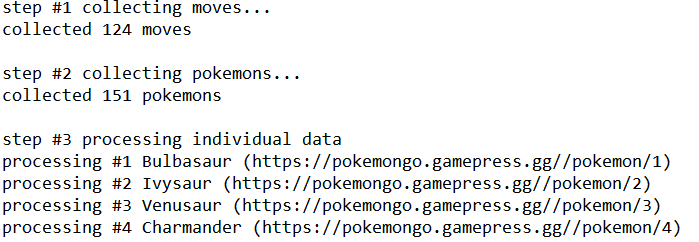
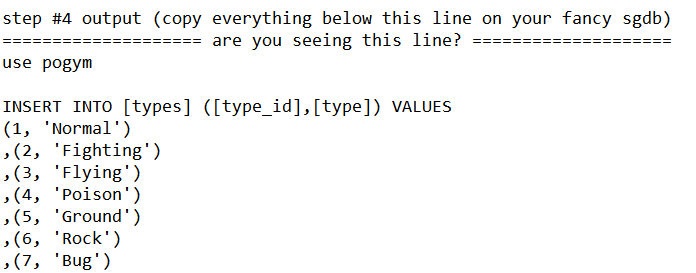
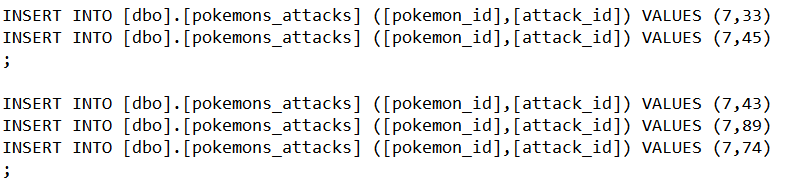
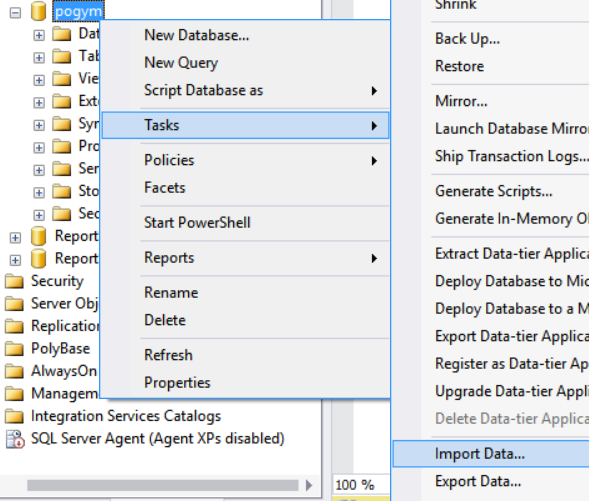
O vigente projeto tem como objetivo a criação de um banco de dados que contenha informações sobre o jogo “Pokémon GO” onde será possível fazer variadas consultas desde estatísticas dos personagens e seus movimentos até cálculos de multiplicadores de vantagem baseada nos tipos dos personagens. Esse projeto também conta com uma aplicação web simples para facilitar seu consumo e as principais consultas do projeto serão:

* Permitir consultar a listagem de pokémons e seus possíveis ataques para inserção nas listas da ferramenta;
* Dado um pokemon e seus dois respectivos ataques, retornar dados sobre o mesmo;
* Permitir gerar as tabelas de multiplicadores de vantagem/desvantagem por tipo dados quaisquer dois tipos de um defensor e o tipo do atacante;

**Carga de dados**

As consultas do banco notadamente a consulta que agrega multiplicadores, talvez a mais importante delas, necessitam dos dados do jogo já inseridos para terem resultado satisfatório, seriam necessários os dados da tabela de pokemons, movimentos (ou ataques), tipos, e de uma tabela que relacione o tipo atacando, o tipo defendendo e o multiplicador entre eles.

Como fonte dos dados utilizamos dois sites, o GamePress e o TheSilphRoad, o primeiro conta com uma comunidade ativa que gera dados de varias fontes, desde o código do jogo até distribuições que os usuários coletam e geram modelos estatísticos, o segundo é mantido por um usuário do Reddit de mesmo nome que faz também esse mesmo trabalho, os dois foram escolhidos pelo critério com que tratam estes dados. O que foi extraído de cada fonte:

* Gamepress <http://pokemongo.gamepress.gg/>
  + Foi usado como fonte primária. Um script python de extração de dados foi construído para recuperar seus dados, ele basicamente lê o código HTML das páginas, procura nas tags HTML os dados e gera uma saída em texto puro que na verdade é uma série de inserts em linguagem SQL92 para serem executadas no banco
    - Script python de extração de dados 
    - Saída do script em execução  
      
    - Fim da saída com comandos SQL de inserção  
      (trecho 1)  
      (trecho 2)  
      
* The SilphRoad <https://thesilphroad.com/research>
  + Checagem da precisão dos dados como movimentos, dano crítico, janela, etc.
  + Fonte da planilha usada para gerar a tabela de multiplicadores de tipos, e a relação entre eles (tabelas “types” e “types\_types”) <https://docs.google.com/spreadsheets/d/1xZ1daunztOgvOZxO7ABfJPF41-2mqMDzY1Qs_0OWwq8/edit#gid=1151218844>
    - A carga destes dados foi feita utilizando a importação de dados do SQL Server, no assistente apontamos a planilha baixada e com somente a aba de multiplicadores e pedimos para descarregar numa nova tabela criada automaticamente  
       
    - Em seguida estes dados foram tratados e migrados para a tabela “types\_types” com os scripts “createmultipliersdata.sql” e “importmultipliersdata.sql” presentes na pasta “database” do repositório;

Este processo de carga teve de ser feito já com a modelagem do banco adiantada, no que foram úteis os diagramas da primeira versão.

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

O diagrama DER do banco a primeira vista parece ter sofrido alterações profundas, mas uma análise mais detalhada mostra que sua estrutura principal foi mantida, principais alterações:

* Os nomes de atributos foram utilizados no seu original em inglês para facilitar o trabalho de mapeamento dos atributos da tabela com o script que lê os dados diretamente da página da gamepress, esta alteração facilitou também o trabalho de checagem dos dados no pós importação;
* Os nomes de tabelas servem como prefixos das relacionadas, para facilitar a compreensão. Por exemplo “Pokemons (pokemons)” e a relacionada “Tipos dos Pokemons (pokemons\_types)”;
* Foi adotada uma nova formatação de nome de atributo:
  + Todo em minúsculas;
  + Com nomes sem abreviatura divididos por “\_” para fins de tornar mais claro seu uso;
  + Chaves estrangeiras mantém o nome de atributo original da chave primária de origem para facilitar a localização;
* Foi feita a especialização da tabela de ataques em uma superclasse “ataque” (attack) e duas subclasses uma para “ataques rápidos” (attacks\_quick) e outra para “ataques especiais” (attacks\_charge);
* Atributos adicionais incluídos no site posteriormente foram adicionados com base em sua utilidade, por exemplo:
  + Atributos “charge” e “energypu” nas sub-classes de “attack”;
  + Atributos que guardam informação de ícone, imagem e url em entidades como “pokemons” e “types”;

Seguem os dois modelos antes e depois das alterações realizadas:

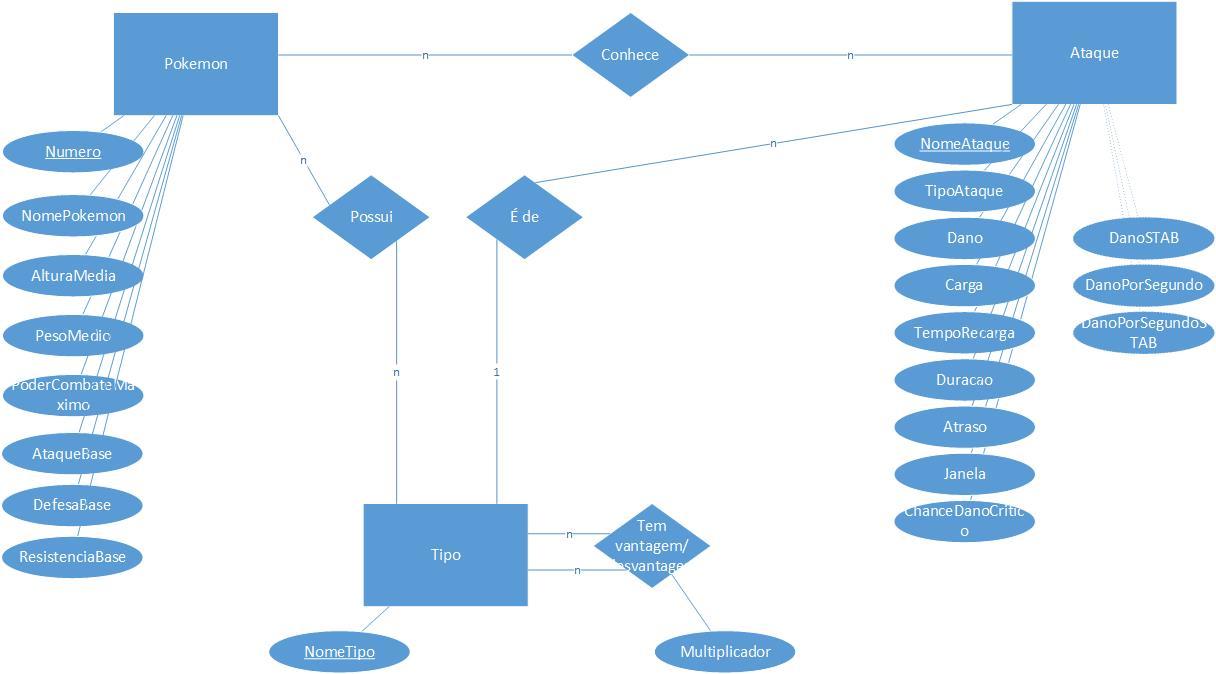


Figura 1: Diagrama original.



Figura 2: Diagrama alterado.

1. **Modelo Relacional**

A partir do DER pôde-se construir o modelo relacional do banco, que é o que segue:

**attacks**(attack\_id,attacks\_types.attack\_type\_id,name,types.type\_id,damage,damage\_stab,damageps,damageps\_stab,move\_cooldown);

**attacks\_charge**(attacks.attack\_id,charge,critical,dodge)

**attacks\_quick**(attacks.attack\_id,energyps,energypu,defensive\_damageps)

**attacks\_types**(attack\_type\_id,type)

**pokemons**(pokemon\_id,pokemons.evolvefrom\_pokemon\_id,name,icon,image,url,height\_avg,weight\_avg,combatpower\_max,attack\_base,defense\_base,stamina\_base)

**pokemons\_attacks**(pokemon\_attack\_id,pokemons.pokemon\_id,attacks.attack\_id,true\_damageps)

**pokemons\_types**(pokemon\_type\_id,pokemons.pokemon\_id,types.type\_id)

**types**(type\_id,type,icon)

**types\_types**(type\_type\_id,types.attacker\_type\_id,types.defender\_type\_id,multiplier)

1. **Normalização**

A revisão do modelo DER, e seu processo de amadurecimento a nosso ver permitiram eliminar boa parte do trabalho posterior de normalização.

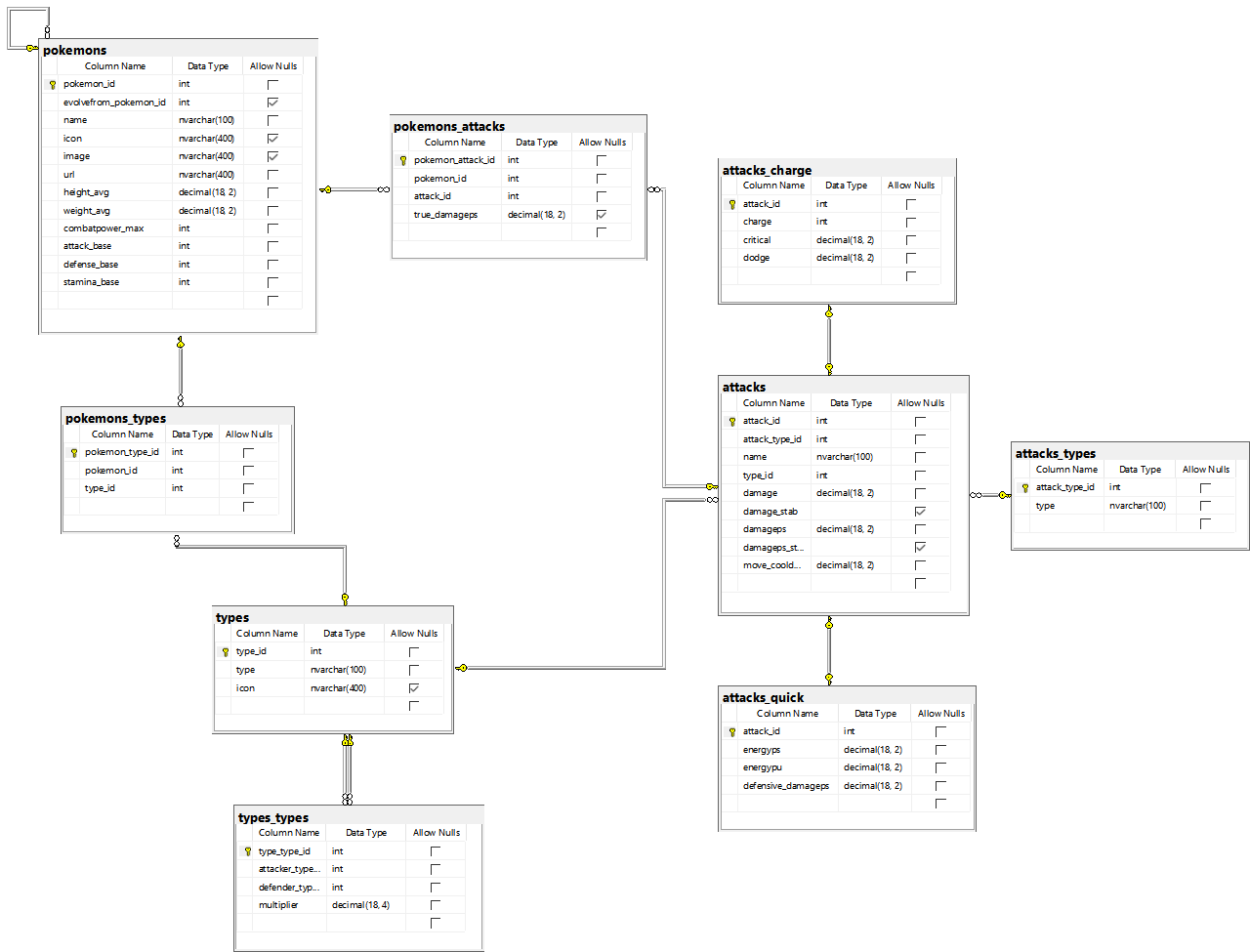
Ao analisar as entidades com relação a dependência funcional por exemplo, entendemos que os atributos nas tabelas não só eram funcionalmente dependentes da sua chave como no nosso entendimento não estavam presentes atributos funcionalmente dependentes de outros atributos não presentes nas chaves. Mesmo redundâncias e campos multivalorados presentes no primeiro DER foram eliminados no segundo.

Podemos fazer duas observações como exemplos de destaque na normalização:

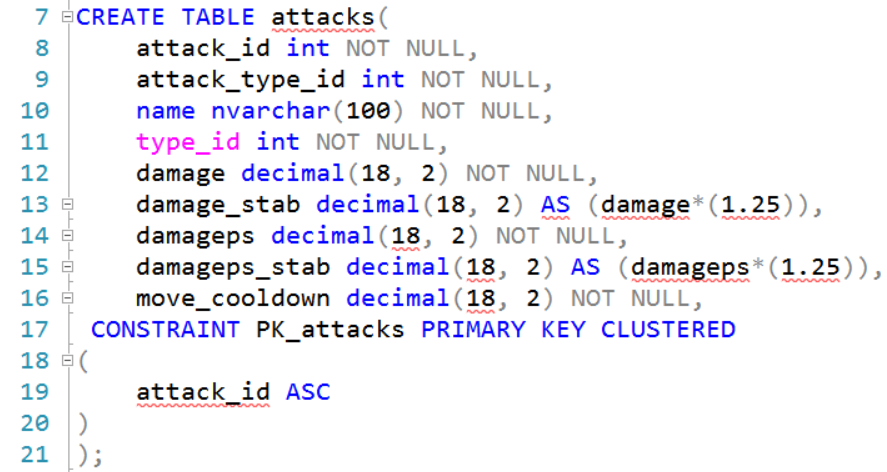
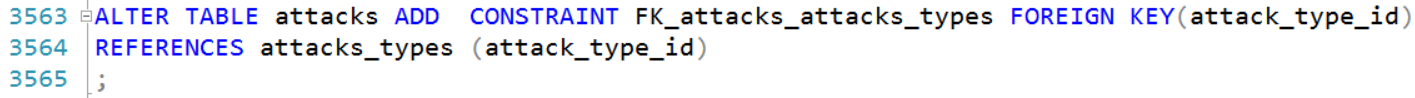
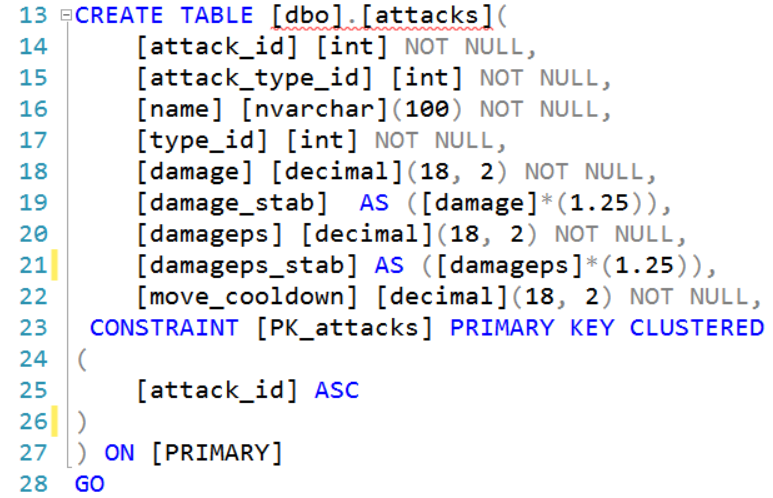
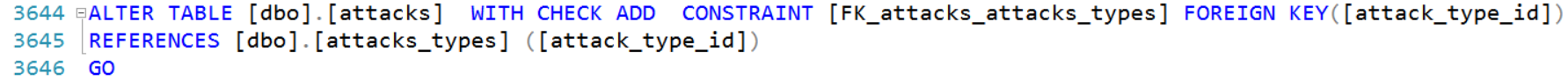
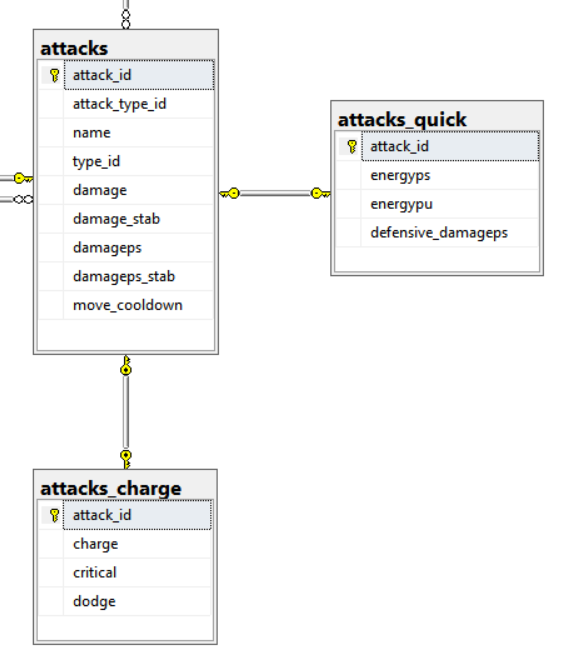
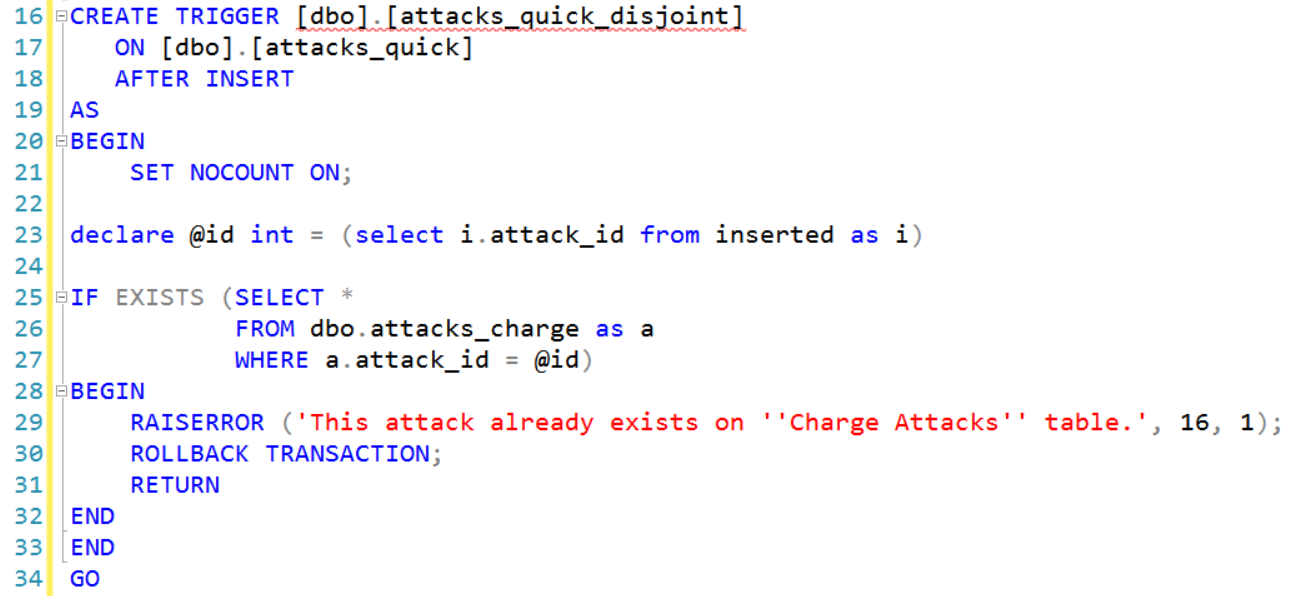
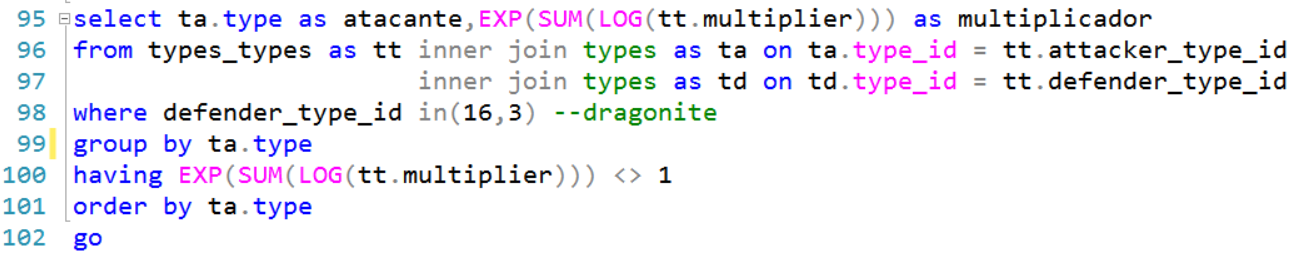
* A tabela “Ataque (attack)” possuía um atributo chamado “Tipo de ataque” que poderia assumir dois valores: “Ataque rápido” ou “Ataque carregado” no primeiro diagrama. No segundo, essa redundância foi resolvida com uma nova entidade separada “Tipo de ataque (attacks\_types)” e o atributo se tornou a chave estrangeira “attack\_type\_id”;
* A tabela “Ataque (attack)” possuía atributos utilizados somente em ataques rápidos, atributos utilizados somente em ataques carregados e atributos utilizados nos dois casos, um caso típico na nossa avaliação no qual poderíamos empregar a especialização, geramos assim duas subclasses da entidade “Ataque (attack)”: “Ataque rápido (attack\_quick)” e “Ataque carregado (attack\_charge)”;

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, com exceção para as duas procedures (sp\_battle e sp\_stab), a função (fn\_stab) e os dois triggers (attacks\_charge\_disjoint e attacks\_quick\_disjoint) utilizadas no projeto.

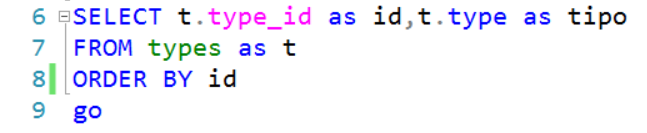
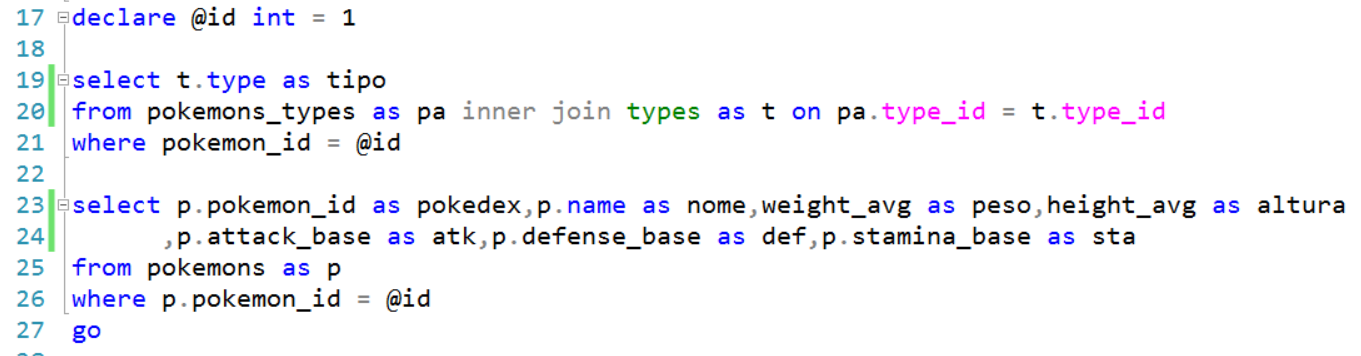
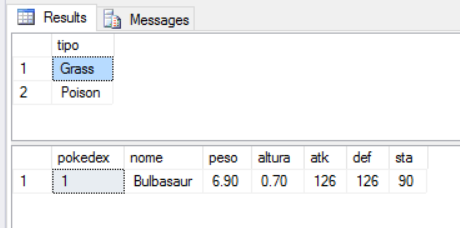
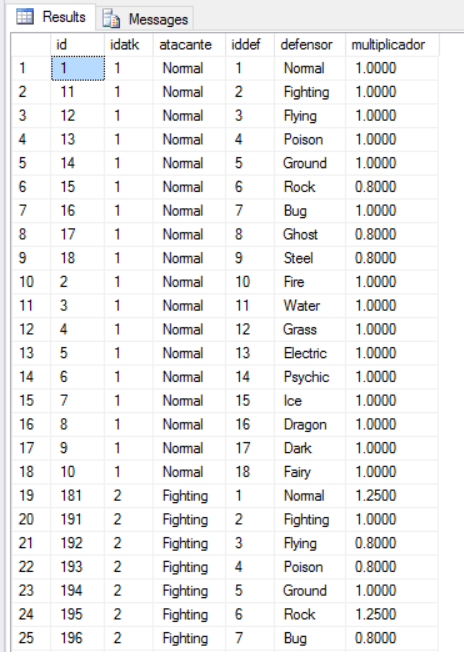
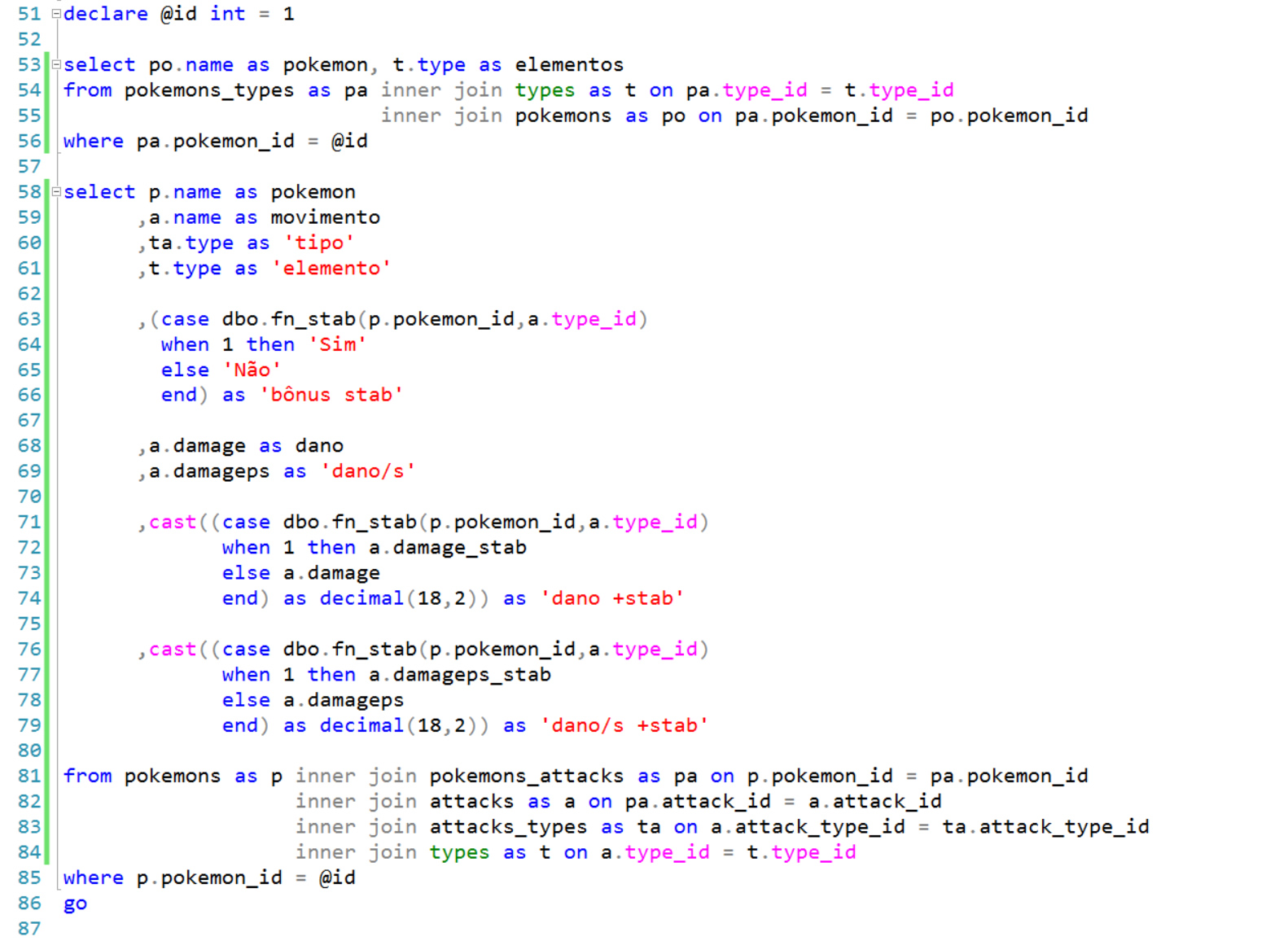
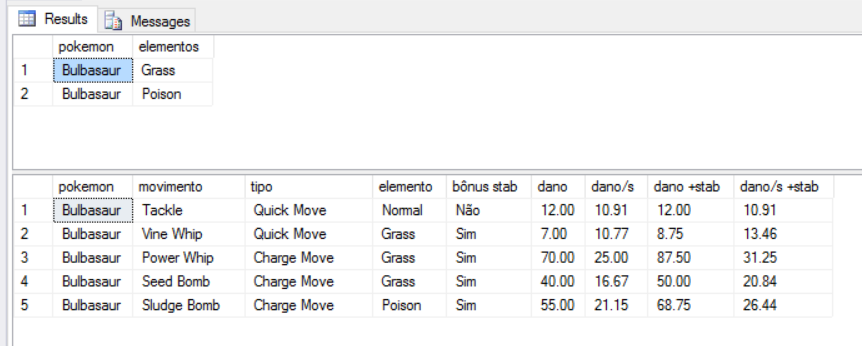
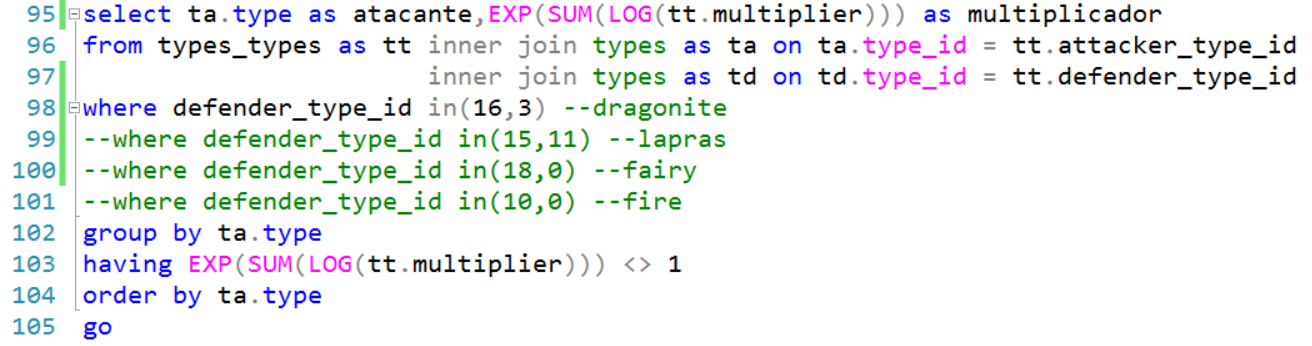
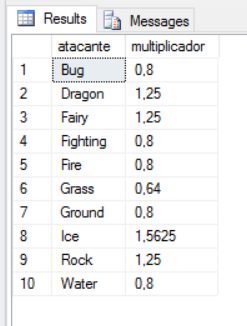
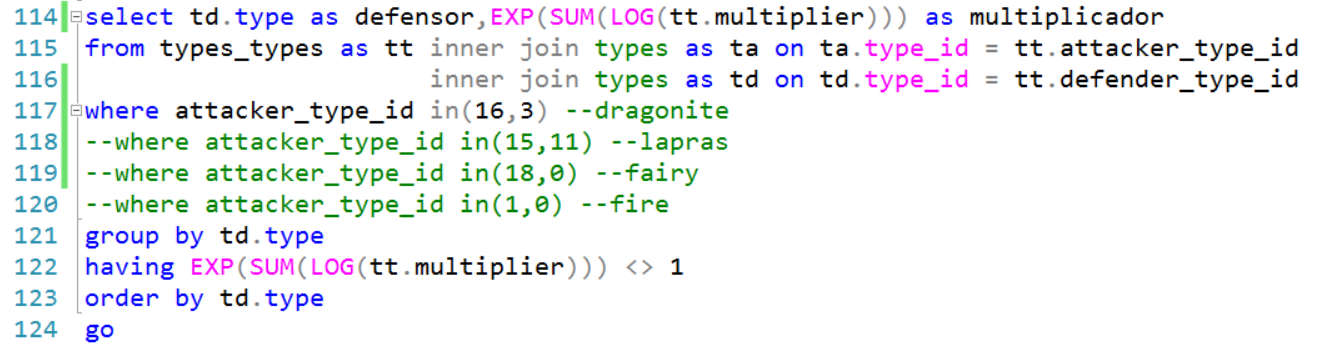
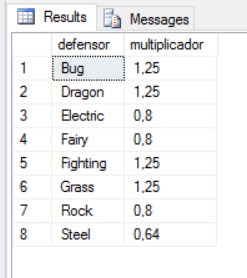
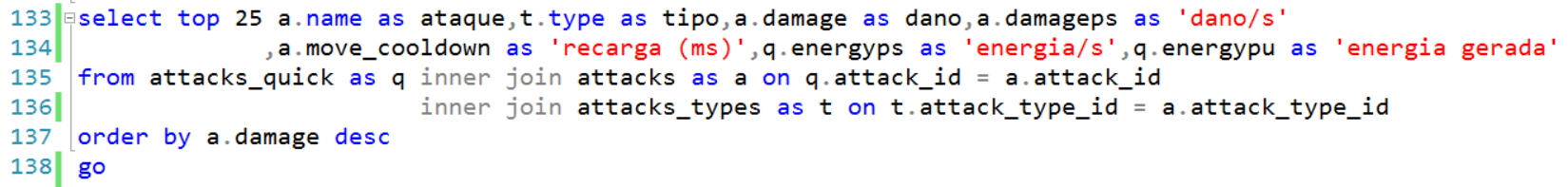
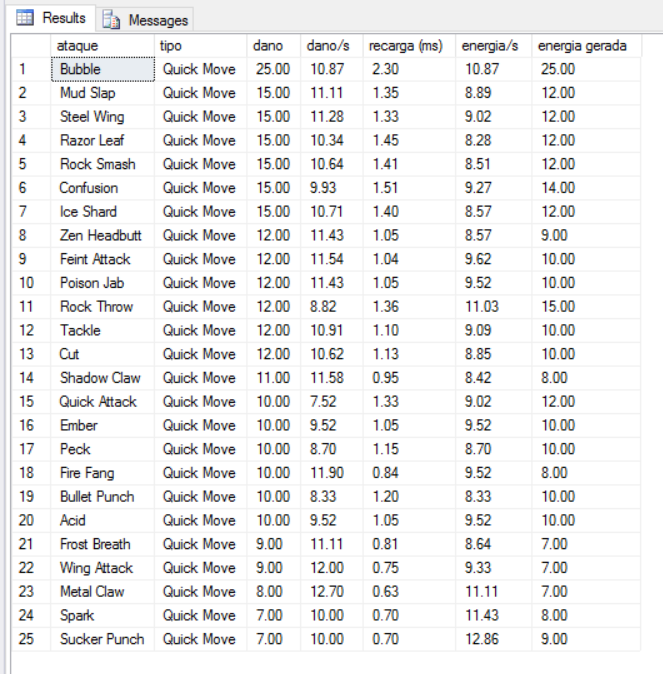
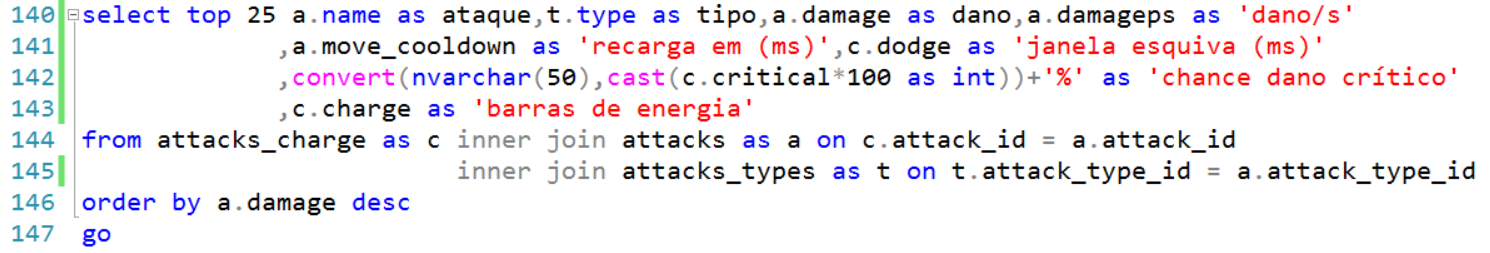
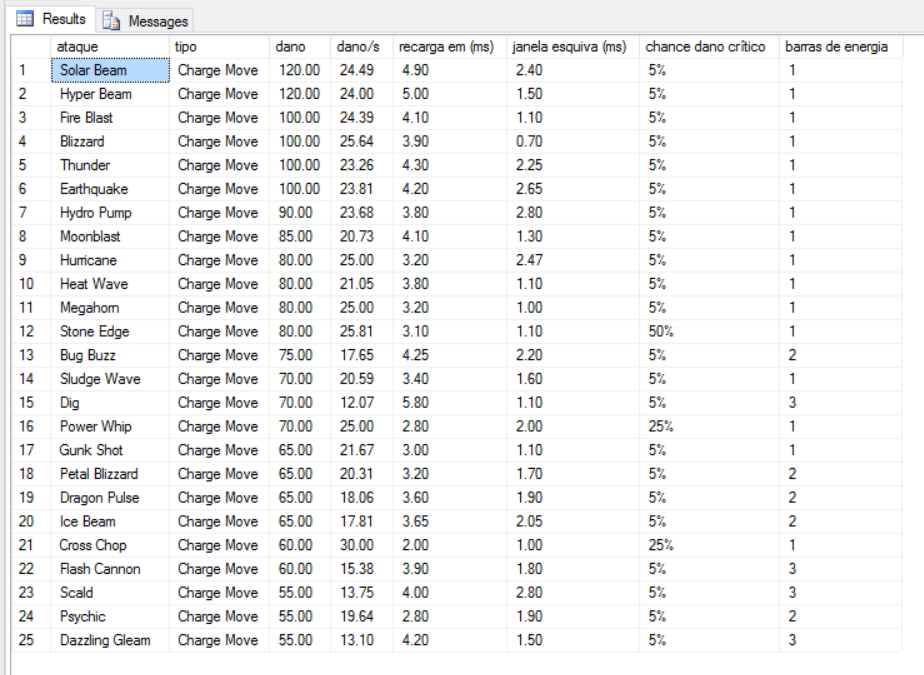
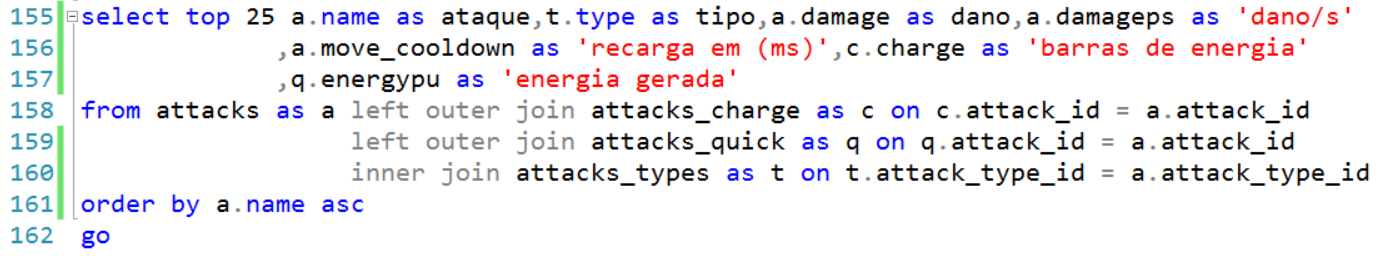
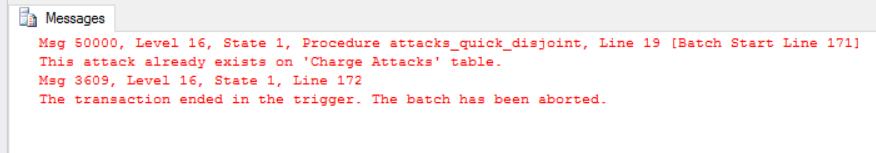
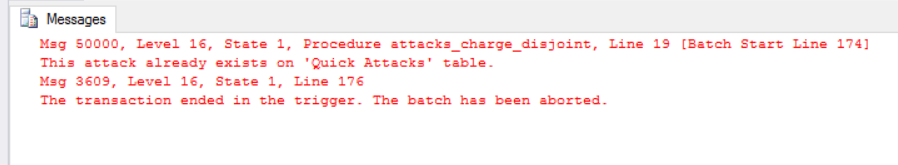
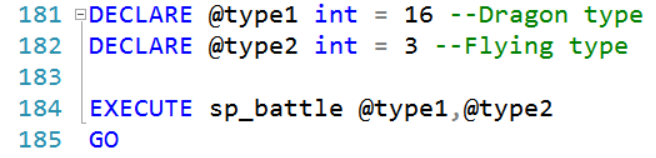
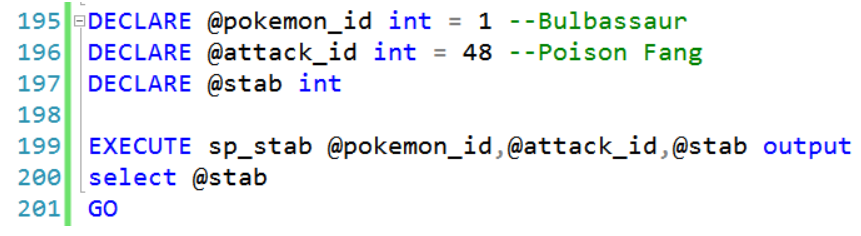
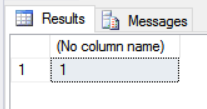
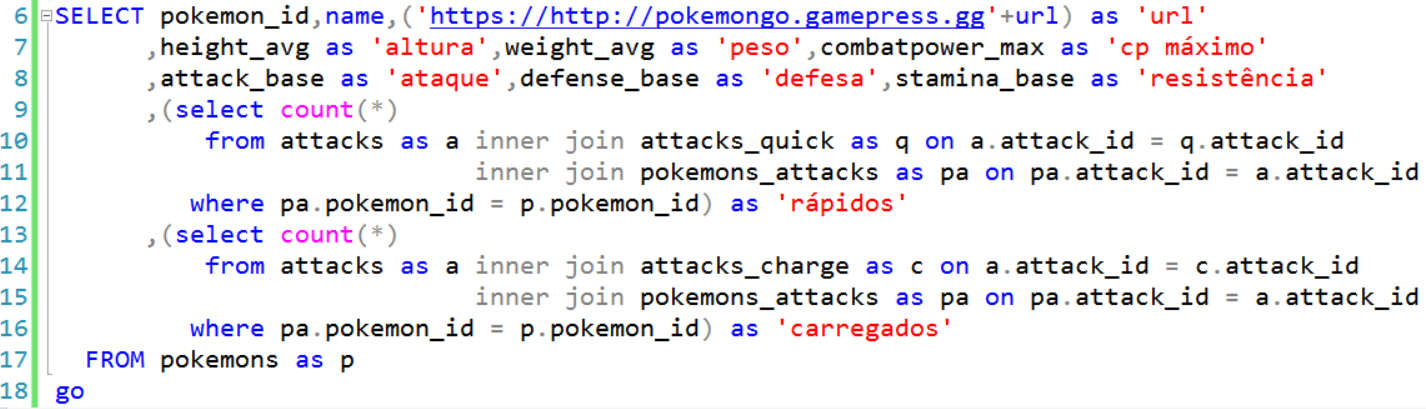
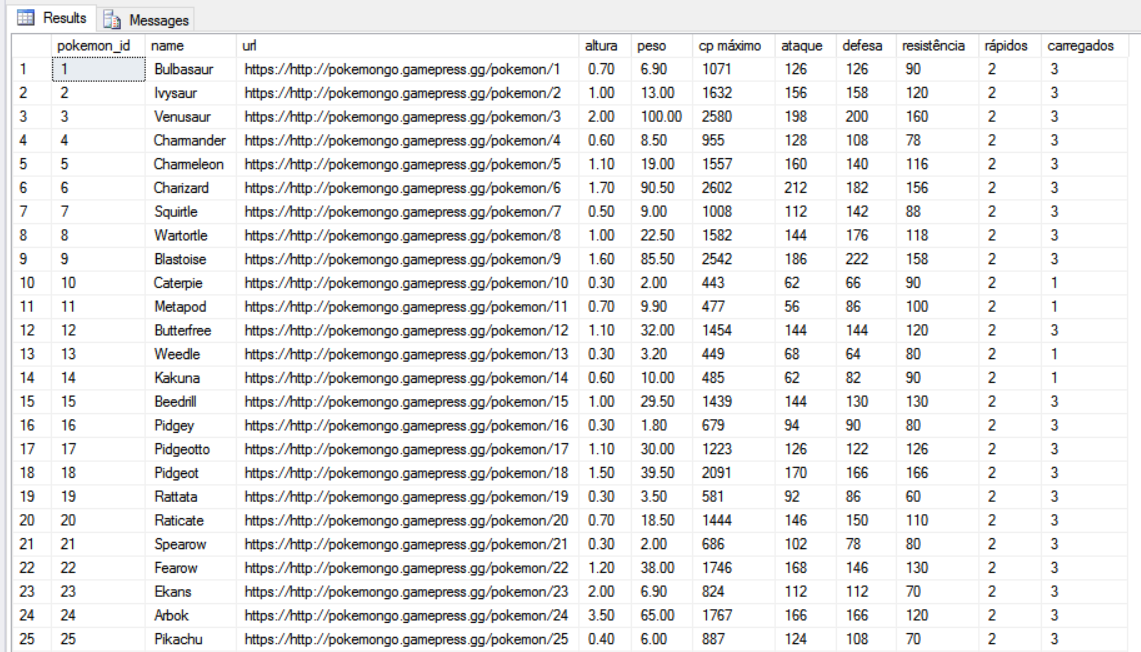
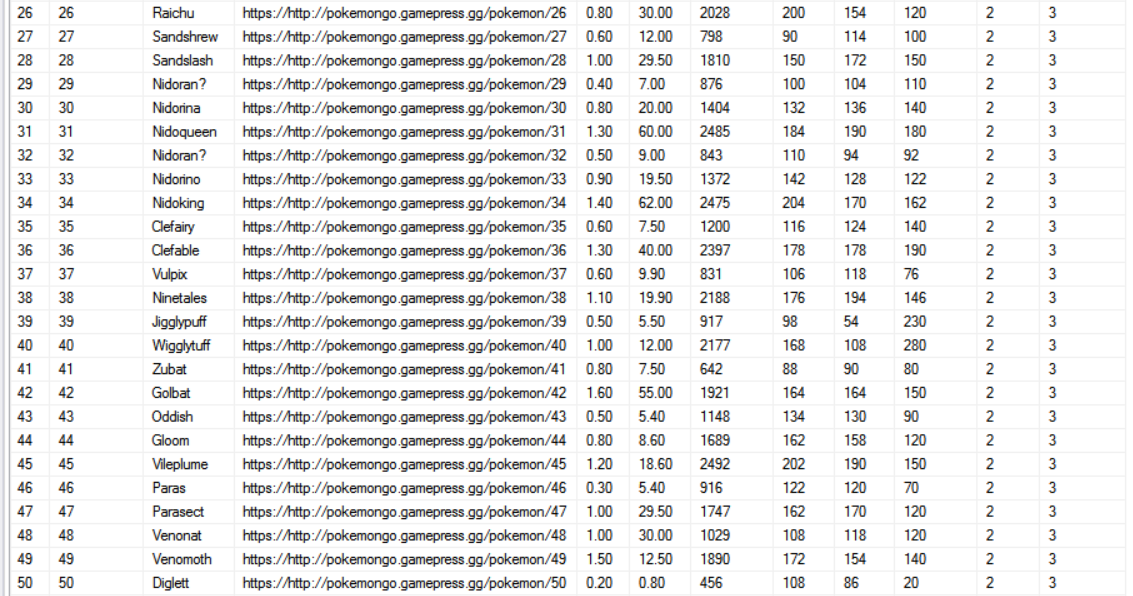
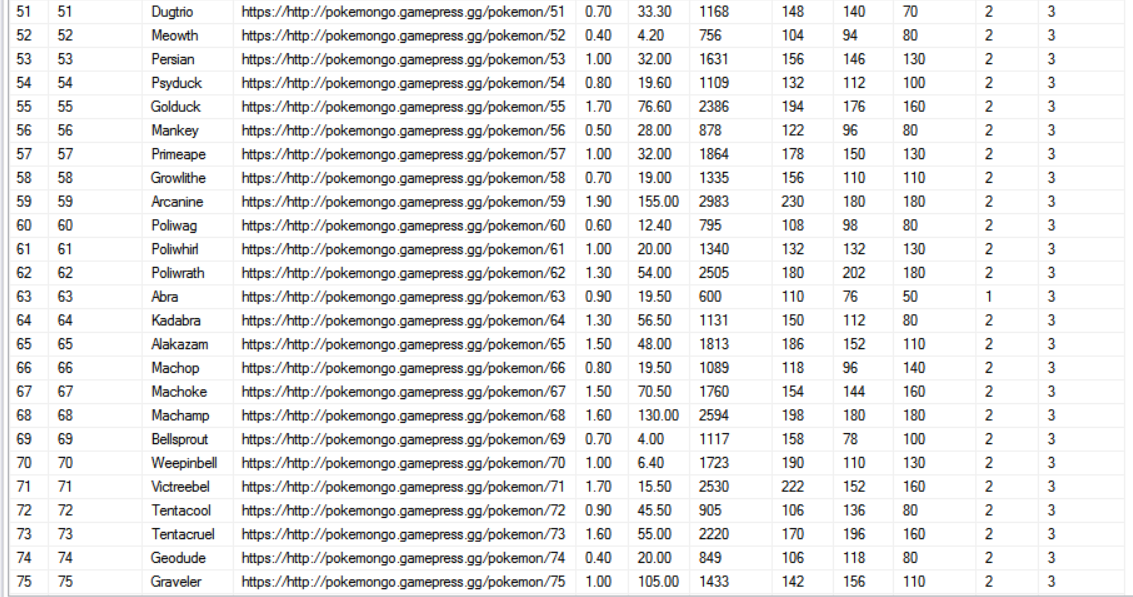
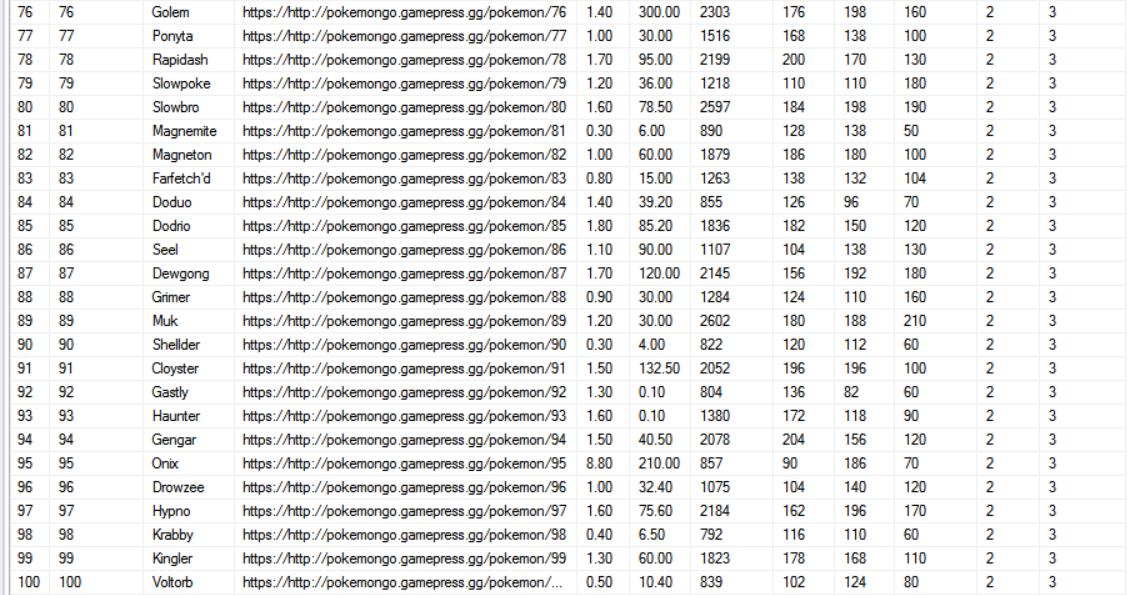
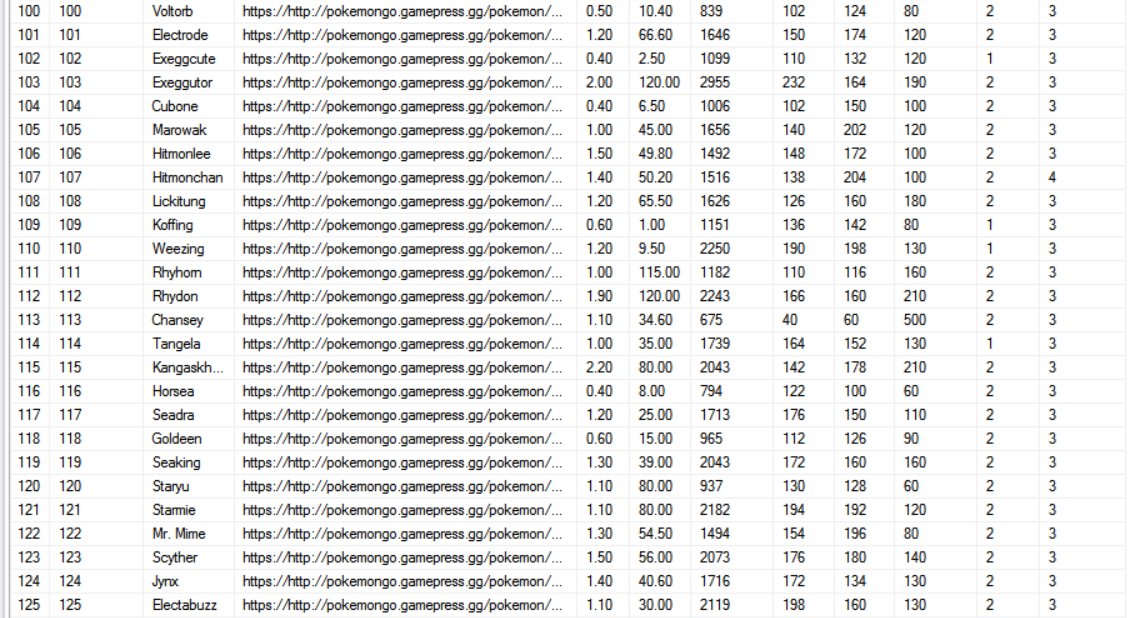
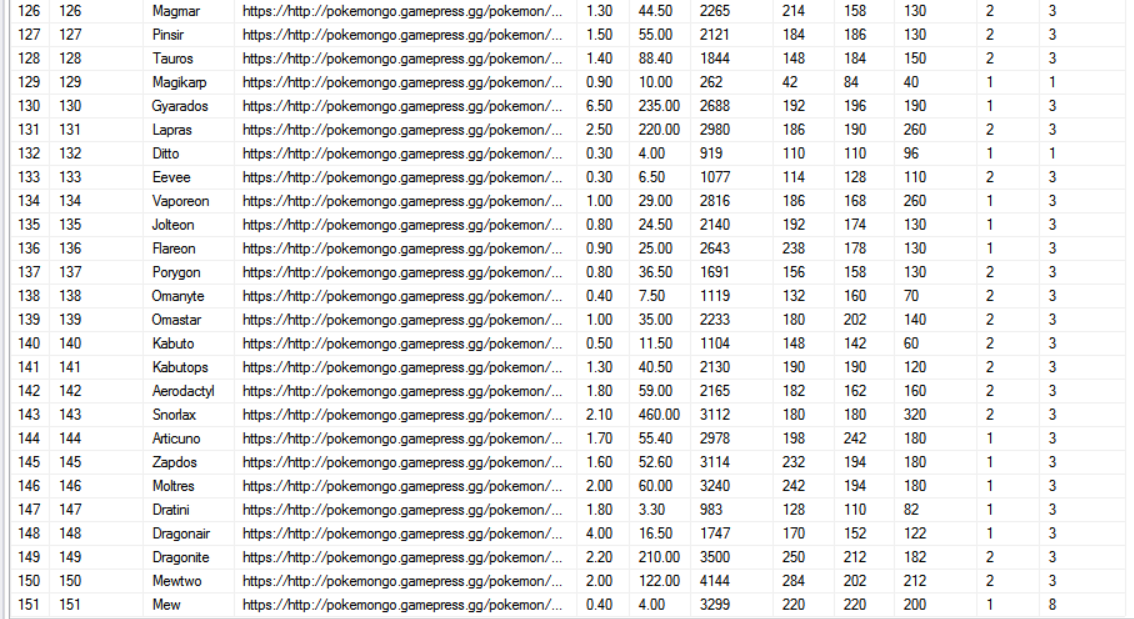
Figura 3: Diagrama do banco no SQL Server

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 implementado no SQL Server o “disjoint” entre as tabelas “Ataques rápidos (attacks\_quick)” e “Ataques carregados (attacks\_charge)” com triggers. Existia a possibilidade de utilizar o campo chave de “Ataques (attacks)” junto ao campo chave de “Ataques tipos (attacks\_types)” 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 (attacks)” na outra relação do disjoint:
  + Modelo do disjoint  
     
  + Trigger na tabela “Ataque rápido (attacks\_quick)”
  + O trigger na tabela “Ataque carregado (attacks\_charge)” é idêntico, mas invertem-se as tabelas na lógica do trigger;
* 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”: 
  + OBS: Esta construção funciona em ambas as plataformas;

1. **Consultas**

As consultas mais importantes no banco e para consumo no aplicativo web s

* Select simples para listagem de tipos por sua ordem natural no jogo  
  
  + Resultado
  + Álgebra Relacional desta consulta  
    t←types  
    R(id,tipo)←𝚷type\_id,type(t)  
    𝝉 id asc (R)
* Selects que listam o(s) tipo(s) e ficha de um dado pokemon 
  + Resultado
  + Álgebra Relacional:  
    @id←1  
    pa ←pokemons\_types  
    t←types  
    tipo←𝚷type(t)  
    σpokemon\_id=@id(pa ⋈ pa.type\_id = t.type\_id t)  
    p←pokemons  
    R(pokedex, nome, peso, altura, atk, def, sta)←𝚷pokemon\_id, name, weight\_avg, height\_avg, attack\_base, defense\_base, stamina\_base(p)  
    σp.pokemon\_id=@id(R)
* Select que mostra a relação da tabela tipos para ela própria com o atributo de relação denominado “multiplicador”
  + Resultado  
    
* Selects que listam o(s) tipo(s) e ataques possíveis de um dado pokemon
  + Resultado
* Select que combina os dois tipos de um pokémon SE DEFENDENDO e gera a tabela de multiplicadores (tabela do site gamepress)
  + Resultado  
    
* Select que combina os dois tipos de um pokémon ATACANDO e gera a tabela de multiplicadores 
  + Resultado  
    
* Selects que listam a tabela de ataques e UMA de suas disjuntas com registros ordenados do maior ao menor dan
  + Select 1  
    
  + Resultados  
    
  + Select 2
  + Resultados
* Select que lista a tabela de ataques JUNTO a suas disjuntas com registros ordenados por nome 
  + Resultados
* Inserção de registro de ataque nas tabelas “attacks\_quick” e “attacks\_charge” para demonstrar o “disjoint” entre elas
  + Resultado trigger tabela “attacks\_quick” 
  + Resultado trigger tabela “attacks\_charge”  
    
* Execução da procedure de batalha para gerar a tabela de multiplicadores
  + Resultados  
    
* Execução da procedure de verificação de bônus “STAB
  + Resultados  
    
* Execução de select que lista todos os pokémons presentes no jogo, junto a sua url, alguns dados e a quantidade de ataques “rápidos” e “especiais” disponíveis para ele  
  
  + Resultados 1-25  
    
  + Resultados 26-50  
    
  + Resultados 51-75  
    
  + Resultados 76-100  
    
  + Resultados 101-125  
    
  + Resultados 126-151  
    

1. **Conclusão**

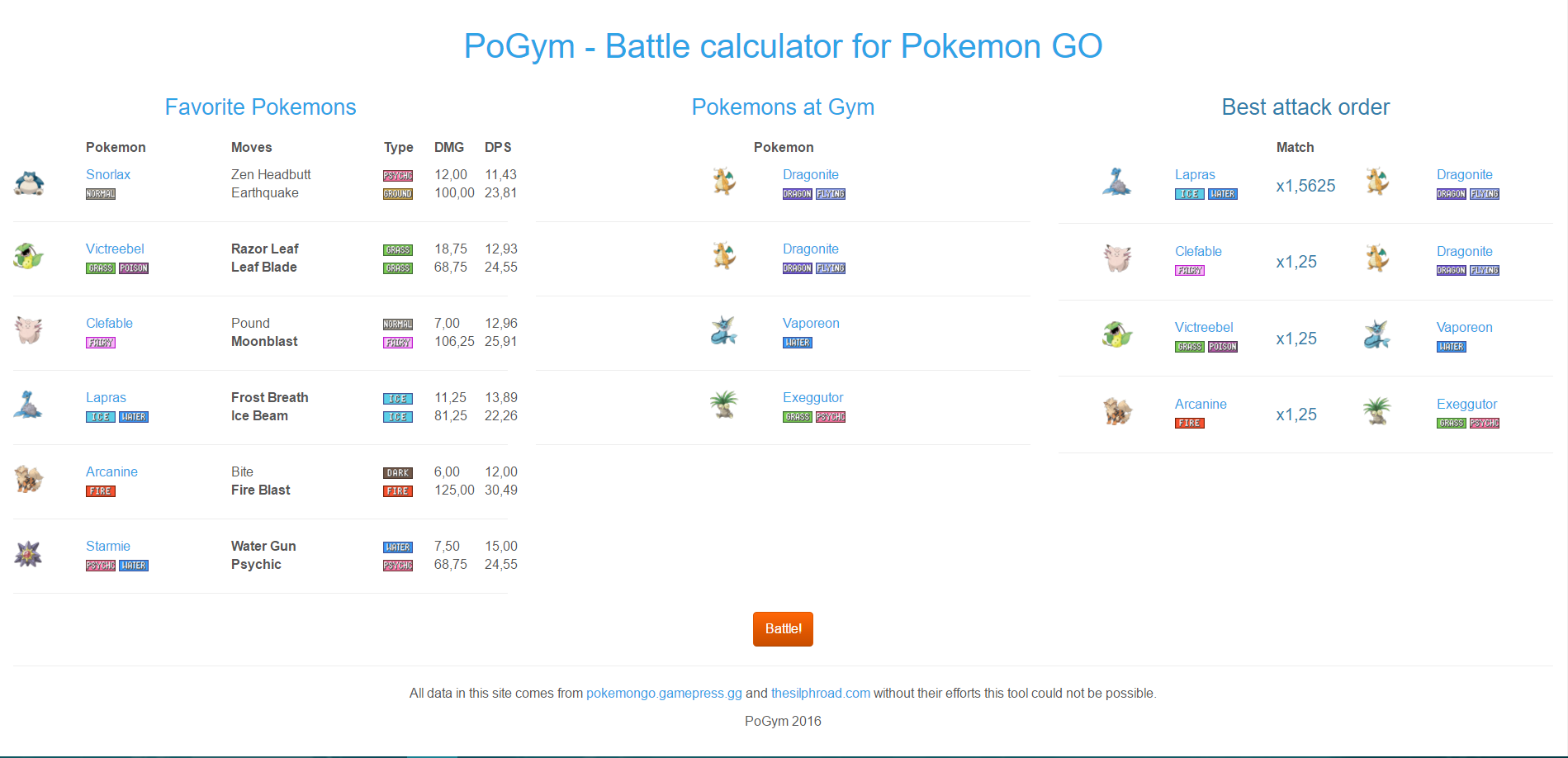
A ferramenta de consumo e apresentação dos dados foi feita utilizando a tecnologia ASP.NET MVC 5.2.3 da plataforma .NET 4.6.1 da microsoft, a linguagem utilizada foi C# , o acesso a dados utiliza tecnologia Entity Framework 6.1.3, um serializador/deserializador JSON Newtonsoft.JSON 9.0.1 e finalmente 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

**Anexos**

Repositório GitHub do projeto: <https://github.com/lnncrs/PoGym>