* Adding a lot of data may be slow, depending on your implementation, so be patient. Consider turning off some of the configuration settings in Entity Framework, if you use it.
* All text can be random string with valid length depending on the field.

Problem 4. SQL Queries (10 points)

Write these two queries in native SQL:

1. Get the price, breed and birth year of the top 5 most expensive pets with birth year after 2012 (inclusive). Order them by price in descending order.
2. Get all species' name and how many products are available for each species, ordered ascending by the number of products.

Provide two .sql files with the two queries in them.

Problem 5. Code First (20 points)

In a **social network** different people can create **user profiles**. Each user profile has unique **username** (mandatory, from 4 to 20 symbols, inclusive), **first name**, **last name** (both optional but from 2 and 50 symbols, inclusive) and **registration date**. Each user has zero or more **images**, which have mandatory **image URL** and mandatory **file extension** (4 symbols maximum). Users can be tagged in **posts**. Each post has mandatory **content** with at least 10 symbols and **posting date**. Each user can be tagged in multiple posts and each post can tag one or more users. Users can create **friendships**. Each friendship has mandatory **first user** and **second user**, whether or not the friendship is **approved** and **date** pointing to when friendship is approved. The logic is: the first user sends friendship request to the second user. If the second user approves, they are now friends and the date and time of the approval is saved in the database. Users can send each other **chat messages**. Each message has **friendship**, **author** (user) of the message, mandatory **content** of the message, **date and time of sending** and **date and time of seeing** the message (such a stupid feature!).

Design a database schema "**SocialNetwork**" using Code First approach with Entity Framework.

The application using the database is expected to filter and sort a lot by the messages' sending date and whether or not friendships are approved so make sure you **optimize** it for these operations.

HUGE HINT 1:

When making the relationships between friendships and user profiles, make sure you put the foreign key properties to the two users in the friendship table, but **do not link from the user profiles to the friendships** with any properties (or do it, if you want to kill yourself with configuration nightmare).

HUGE HINT 2:

Add the following to your context class, if you encounter a cascade delete exception. Keep in mind this is not a good practice but it will make it easier for you:

```
using System.Data.Entity.ModelConfiguration.Conventions;

protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>();
    modelBuilder.Conventions.Remove<ManyToManyCascadeDeleteConvention>();

    base.OnModelCreating(modelBuilder);
}
```