Genoom Simpsons Family Tree

# Database design

This is the **key topic** to analyze for this exercise. The schema of the data seems simple at first sight but digging into it, we face the following problems:

* Each family member (node in a graph) has 3 levels for navigation:
  + Up 🡪 Parents
  + Down 🡪 Children
  + Side 🡪
    - Partner
    - Sibling
* Each relation between family members, is **symmetric**, that means that automatically in a standard RDBMs would **duplicate the amount of entries needed**.
* There is the transitive property between the family members. But in the exercise, should be taken in account for getting the tree.
  + We should use **recursive calls** to navigate through the parents tree, while keeping the code needed low.

## SQL Server (or any other standard RDBMS)

* Pros:
  + Pretty standard, very easy to work with because is ubiquitous.
  + Well supported in Azure.
  + Good and stable tools and documentation.
  + ACID compliant.
* Cons:
  + We generate a lot of entries in the table pretty quick due to the symmetric of nature of the data.
  + We need a good set of indexes to not income with serious performance problems soon.
  + We need to set data partitions to manage the large amount of data.
  + We should avoid large transactions for updating the data at all (should not be a big problem because the Create/Update operation is adding a new children member to an existing one directly without needing to traverse large trees).

## MongoDb (NoSQL document based)

* Pros:
  + The cost of accessing a family member and its direct tree levels is 1. As the document should contain this info.
  + The returning data format is JSON (technically BSON) so no extra serialization is needed.
* Cons:
  + Not as good support in Azure as the SQL Server.
  + The documents could become big (be aware of 10MB limit per doc).
  + Costly Add/Update operations because the large documents. (More than the SQL Server).

Should not be a big problem because most of the operations a for reading the tree not updating it.

# Project architecture proposal:

## Logic and Code strategy

Before analyzing the architecture let’s set some points:

* There are 4 responsibilities for the project:
  + **Web stuff**: Web Api, Controllers, data serialization to JSON, etc.
  + **Business logic**: like checking if the person has a partner when adding a child.
  + **Data access repository**: to access the data here due to the topics discussed above, we could use a strategy like pattern so we can access the data on SQL Server and MongoDb and we could compare which option is better.
  + **Model**: simple C# POCOs

## Diagram



## For the SQL project:

We use Dapper instead of SQL Server,

It’s a lightweight ORM and has better performance than EF.