SQLite Writeup

## Preface

In summary, this document is the result of our findings involving the resume-state task assigned to us for sprint 3 of this class. To summarize, we found that the libraries available to us for this task were limited in functionality, somewhat forcing us to roll our own functionality or drop the task altogether. Ultimately, our decision as a group was to drop the state-resume task, below we will describe the reasoning behind our actions and go into detail about the limitations of the previously mentioned libraries.

## Limitations of the library

We chose to use “SQLite-net extensions” as our library for SQLite. This is a fork of the “SQLite-net” library; “SQLite-net extensions” adds (basic) relationship support. This library has a few limitations with it. Some of these limitations also extend to other SQLite libraries we looked at.

- No support for composite primary keys. Only one column is allowed to be in the primary key
- Classes that are involved in relationships *have* to have an auto generated numeric primary key

## The roadblock

We could have played along with the limitations of the library and made everything have a throw away primary key that is not serialized into JSON. In doing this we hit the big roadblock, which we believe comes from the stiffness of SQL itself.

We have a type called *NameTypeObject* that is just that- a class that has two string fields- one for a name and one for a type. This is used to model both fields in a class and parameters in a method. This creates complications for modeling class fields and method parameters in SQL. Technically, there are two one to many relationships here: Class and NameTypeObject for fields, Method and NameTypeObject for parameters. For modeling a one-to-many relationship, “SQLite-net extensions” requires having a collection of the type of the “many” side (in this case NameTypeObject) stored in the “one” side (the Class or Method), and then the PK of the “one” side stored in the type of the “many” side and referenced through an

FK. Doing this would not work. A NameTypeObject could not be stored without a Class existing and a method existing for the foreign keys to work.

## Conclusions

In conclusion, our findings resulted in numerous issues that would have prevented much success if bound to working with a third party library. The example we gave of one such library, “SQLite-net extensions”, lacked certain features that definitely would have made our lives easier. These included the idea of composite primary keys, to allow for multiple columns to make up a primary key within the SQL database. The other feature, which would have caused issues down the line, was the requirement of auto generated numeric primary keys for classes involved in relationships. There would be potential solutions to these issues, but at the end of the day it would have increased the complexity of this task significantly.

For example:

- Investigating another more flexible library. We did not find too many that we believed to be simple to learn and integrate. One other alternative was ADO.NET, but we did not look much into it as others (people on the internet) said it was complex.
- Making it work with “SQLite-Net extensions”. We could have created an explicit Parameter and Field type. Did not like this idea because the two types would only differ in the foreign key.
- Writing our own solution/ SQL queries to bridge the gaps