Currying Connections

These are some of the techniques used to connect our various databases from work. This approach is taken because there was a notorious connection leak issue that brought down our 5-college database servers during registration when there is heavy traffic on the portal. Additionally, it simplifies and eliminates thousands of lines of code in our code base.

Why Curry?

Currying allows us to combine together expressions and execute them at one time.

In our code base, there appears to be 4 common patterns when doing database-related operations. **Scalar**, which executes a command and returns the first column of the first row of the query; if here are no rows, a null is expected. **Non-query**, which returns the rows affected, typically, this is executed for updates or deletes. **Rows/Row**, here we expect rows to be returned, the former approach was to use the ADO API to fill the data table and acquire the data through the data from it.

Here is a typical flow of how a database connection runs.

1. **Scalar**: Connect to DB -> Execute SQL Cmd -> Supply Needed Parameters -> Execute Scalar -> Acquire First Column of First Row or Null
2. **Non-Query**: Connect to DB -> Execute SQL Cmd -> Supply Needed Parameters -> Execute Non-query (updates, deletes, ect) -> Rows Affected
3. **Rows**: Connect to DB -> Execute SQL Cmd -> Supply Needed Parameters -> Use ADO API's to fill data rows -> Extract the data
4. **Row**: Same as *Rows*, but we only use one.

With currying, there is a simple way to handle these operations and guarantee no connection leaks.

For instance, compare the curried version . . .

```
Db
  .Jics
  .Proc("spGetFwkUser")
  .Param("@uid", userid)
  .Row();
```

to the non-curried version . . .

```
using (var conn = new SqlConnection(<... conn str ...>))
{
    using(var proc = conn.CreateCommand())
    {

      proc.CommandType = CommandType.StoredProcedure;
```

```
        proc.CommandText = "sp";
        proc.Parameters.Add("@uid", "spGetFwkUser");

        conn.Open();

        var rdr = proc.ExecuteReader();

        if (!rdr.HasRows)
            return null;
        else
        {
            var users = new List<FwkUsers>();
            while (rdr.Read())
            {
                // parse
                // and
                // extract
                // data
                // . . .
            }
            return users;
        }

      }
  }
```

Please keep in mind that in the curried version, we use .Rows() which returns a Dictionary<string, object> that represents a row (where the key is the column's name--always of type string), the value of that column and is whatever object it is). In many cases, this allows us to forward the data model straight to the front-end as json.