I saw Mr. Poe’s blog entry and thought I like to solve the projects issues of

XSS injection attacks

Unicode

SQL injection attacks

I don’t have the sample input files to play with but I can put data into a ‘properly normalize schema’

using some employee data.

---

**Curtis Poe’s Linkedin blog 7-mar-2018**

**https://www.linkedin.com/pulse/building-software-consulting-firm-curtis-poe/**

**Building a Software Consulting Firm**

We create software tests which are small applications related to our problem domain, take a few hours to write (at most), and maybe a few more to polish. They can easily be done in under a day. We give the developer an entire week to finish it, starting when they tell us to send them the test.

…

So how do you grade the test? The rule is simple: would you be willing to use the candidate's code for a client? … a Extract, Transform, Load system … . The candidate needs to pay careful attention to data quality. So we can create a simple test where the candidate needs to build a simple web application to upload a CSV or Excel file and store the results in a properly designed SQLite database. The application must then show a page where the customer can select an upload for viewing. We skip authentication, authorization, or any of a number of things we'd like to include because giving the candidate too much in one test isn't fair.

We tell the candidate that the data will be used for "unspecified" reporting. This forces them to properly normalize the database rather than make assumptions about reporting requirements. We also make it clear that the specs are intentionally vague and the candidate must use their judgement (and document it) to guess what to do when there is lack of clarity. ...

At minimum, the candidate must return:

* A README file explaining how to install and run the software
* A NOTES file documenting the assumptions they've made
* The working software
* All materials you have sent them (otherwise, if you update the material after you've sent the test, it may be unclear which version they have received)

So how do you judge if the code is production ready? That will depend on your company. We have no "litmus test" which guarantees a candidate will fail. Instead, we score on various criteria we need (architecture, testing, documentation, and so on) and successful candidates move on to the next round, the structured interview.

…

Side note: At this point, some of you may be thinking that the above test is too simple. In fact, we find that the vast majority of candidates badly fail tests like what we've described. We went two years before we had a single candidate return a properly normalized database. Our data file listed both "Christina Rossetti" and "Christina Rosetti", but with the same customer ID. Few candidates noticed that. Slightly more than half the candidates had XSS injection attacks, many failed our Unicode tests, and a few had SQL injection attacks. Very few wrote tests, documentation was almost non-existent, and many developers sent us single file monstrosities (and one developer saved the file unchanged in a single field claiming that the end user can download the CSV and import it into Excel and use pivot tables for reporting. Effectively, they wrote a CSV to CSV converter.)

**Earlier in the blog entry Mr. Poe describes an ETL project his company did**

If you're not familiar with ETL, it's "Extract, Transform, Load." With a generic interface you can extract from multiple data sources easily (there's major hand-waving in that, of course). With a properly designed database, loading the data is not terribly difficult. It's the "transform" step which does much of the heavy lifting. Is "Dr. Bill Smith at Hospital A" the same as "Dr. William Smith at Hospital A"? Or maybe name of the hospital is misspelled. Or the postcode is wrong. Or the phone number is missing. Or it's formatted strangely. With multiple companies sending us data files via SFTP, web uploads, or email, and sometimes rearranging the columns in the spreadsheets, or renaming the columns, or splitting them or combining them, it would take two days of expert developer work to import a single spreadsheet and after enough growth, this was grinding development to a halt.

Fairly quickly we created a rules-based importer, fully tested and documented, that reduced the time of ingesting a single document from two days to two hours and usually developer help was no longer required. We also had extensive error checking and warnings to make it trivial to find the inevitable issues with source data when they arose.