NOvA Databases - Lessons Learned

(and opinions formed)

Jonathan Paley

DUNE Database Workshop

December 3, 2019

My Role

- As an early-career postdoc (IOW, very naive), I was asked in 2006 if I'd be interested in taking on responsibilities for NOvA databases. I knew nothing about databases, but had long been curious about them, so I agreed.
- In this role, I was responsible for developing a system to provide QA/QC/Installation tracking, including schema development (mostly for the NOvA Project). We called this our "hardware database".
- This role eventually evolved to also include:
  - creation and maintenance of DAQ and DCS databases
  - schema development for DAQ and DCS systems
  - schema development for offline calibrations
  - development of generic C++ API database interface
- As we went from "Project" to "Operating Experiment", a lot of my effort went into replicating the various databases and making the data available to the experiment (note, "replication" may sound trivial, but we did not have the same tools back then!)
- As a result, I learned a great deal about databases (although I still feel like a novice), and perhaps more importantly, I necessarily learned a great deal about nearly every aspect of the experiment.

This Talk

- Here I present some thoughts and opinions that I have formed over the years in my role as the NOvA Database Coordinator.
- I am sure that I have forgotten to include some “lessons learned” in this talk, what I mention here is more or less off the top of my scattered brain.
- If nothing else, here are my key take-away messages:
  - we need both the DUNE/LBNF Project and DUNE Experiment to work together and understand the needs of each other, and to communicate those needs to the Database Teamwhat each needs more people involved
  - we need very few people to work on the database implementation
  - we need a lot of people to work on “interfaces” (see next slide)
  - we need flexible database interfaces that focus on real use cases
- I don’t have many slides here, but hopefully there will be lots of discussion.

Lesson Learned: What “Databases” Really Means in Experiments

- IMHO, databases are really all about **interfaces.** Eg:
  - Project ↔ Experiment
  - Online ↔ Offline
  - Engineers | Technicians | Computing Pros ↔ Physicists
  - Calibration experts ↔ Analyzers
- When someone says “we need a database to…”, what they most likely mean is “we need to create an interface for a user to determine ‘Y’ based on information from a database table that contains ‘X’”.
- creating databases and tables is the “easy” part**
  - designing the tables is slightly trickier**
  - understanding both sides of the interfaces is critical and non-trivial. Requires buy-in from both sides.
- Example: Project needs to track purchase, delivery, test-stand results and installation of component X in detector. 5 years after the Project has ended, experiment needs to extract test-stand results for X installed in location A of the detector.

**** Labs and universities have experts who can either do the work, or provide excellent guidance.**

Lesson Learned: Get Buy-In From Everyone

- Buy-in means:
  - agreement to store all relevant data in a database in a timely fashion
  - commitment to providing documentation with details of the meaning of, and relationships between, data columns
  - person-power to help with interfaces, both to push to and pull from the database. In many cases, several different interfaces may be necessary.
- On NOvA, the Project was successful in getting "factory" managers to push relevant information to databases.
  - A hardware database and general interface to push data to it were set up by Fermilab.
  - Nevertheless, several factories went their own way and stored data in a local database because it was "quick and easy" (meaning, they didn't take the time to document their tables).
- There was very little (~no) thought put into how to make this information easily accessible to the collaboration.
- As a result, many hundreds of hours have been spent by collaborators to figure out how to extract relevant information needed for calibrations and improved simulations. In many cases, we have had to give up on getting "as-built" information.

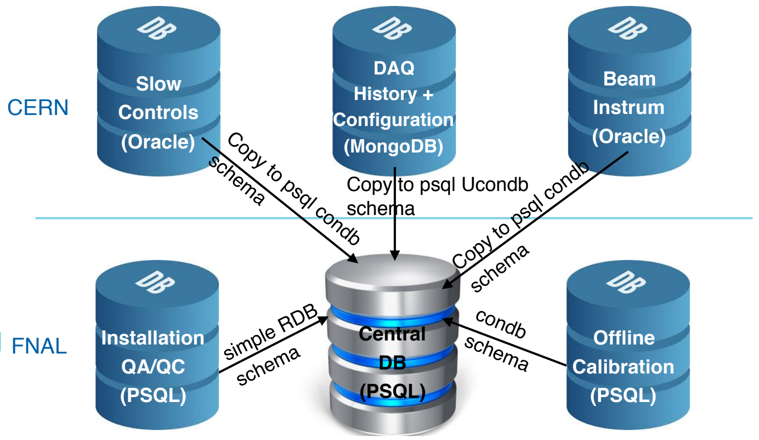
Lesson Learned: Build a Team from the Beginning

- I was the only person tasked with the role described on the previous slide.
- The “Database Czar” should not be an early career physicist. It can get political.
- However, there are real benefits to being involved, so early career physicists should be encouraged to work on some aspects.
- That said, a great deal of institutional knowledge has been lost on NOvA as either early career or temporary Project workers moved on to greener pastures without properly documenting the meaning of the content of their tables. So one of the core responsibilities of team members is **documentation**.
- Make use of database expertise whenever/wherever available. Eg, I was eventually able to pass the responsibility for maintaining and replicating all NOvA database servers on to the database admins in the Fermilab CD.
  - They have real, relevant training, and do a way better job at this than I do.
  - It freed me up to focus more on other things, both in NOvA operations and data analysis.

Lesson Learned: Accept that you will be dealing with multiple databases

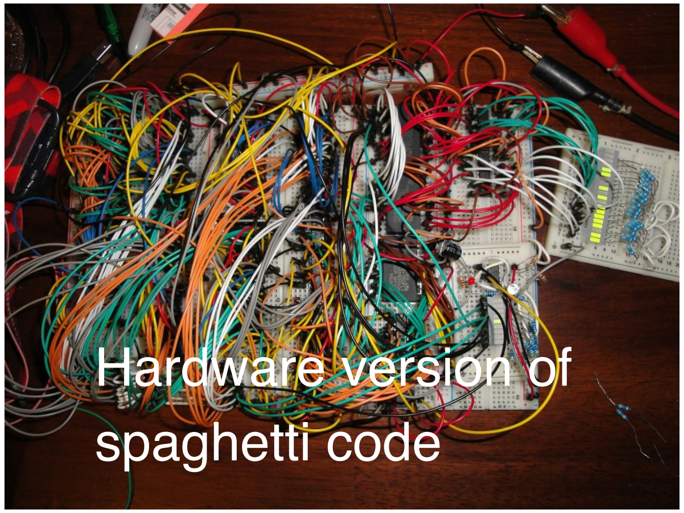
- Fact: Databases are an afterthought and not integrated into the design of the Projects/Experiments.
- Fact: Our experiments are complicated, and many systems already have databases set up (eg, beam, slow controls, DAQ, etc.).
- Fact: Cost and schedule pressures lead many to the path of least resistance, which is to implement and populate a local database.
- On NOvA, I tried early on to have everyone store their information in a single database. This attempt failed, but I later arrived at the conclusion that this does not need to be a requirement, and in fact would not solve the underlying problem of lack of proper interfaces.
- In many cases, "local" databases can be further distilled, the data reformatted and copied to a central server for convenient access to collaborators. This was done on both NOvA and ProtoDUNE.
- It is critical that our offline database interface be designed with the reality that data will likely live in multiple locations and/or have different formats.

HERDING CATS:
"A futile attempt to control that which is inherently uncontrollable."


CERN
DB
Slow Controls (Oracle)
DB
DAQ History + Configuration (MongoDB)
DB
Beam Instrum (Oracle)
Copy to psql condb schema
Copy to psql Ucondb schema
Copy to psql condb schema
FNAL
DB
Installation QA/QC (PSQL)
simple RDB schema
Central DB (PSQL)
condb schema
DB
Offline Calibration (PSQL)

Lesson Learned: Analyzers Don't Want to Need to Know How to Query a Database

- Analyzers have better things to do than to learn how to query and write to a database table.
- Although there are C and Python APIs for databases, they generally require the user to execute SQL statements.
- On NOvA we have used a DBI that generates and executes the necessary queries, and unpacks the data under the hood.
  - This has worked well for most tables, eg, DAQ, DCS and Calibration.
  - Strongly suggest we use something like this in DUNE. (note, we use a de-NOvAfied version of this in ProtoDUNE).
- I feel this approach avoided the development of spaghetti code throughout the NOvA code base (although one could argue that the DBI is a bit of spaghetti code itself, but at least it's centralized…)
- Although the number of people who will actually develop code to extract data from databases is very small, it's worthwhile to have a simple DBI.



Lesson Learned: Never Connect Directly to a DB

- Every experiment I have worked on that made use of modern grid computing has crashed "the database" by having too many simultaneous jobs try to read tables.
- The open source databases I have experience with can typically handle 100-200 simultaneous connections.
- Web services typically handle "traffic" much better.
- Fermilab developed several new database web services that also cache query results so that repeated queries don't keep going back to the database. This was a game changer for us.
- Three types of web services:
  - **Query Engine (QE)**: web front end to simple database queries of a single table.
  - **Conditions Database (condb)**: optimized for structured data that are a function of a "validity time" (timestamp+detector+dataType+…)
  - **Unstructured Conditions Database (ucondb)**: similar to condb in concept, but unstructured data
- Note, we did have to add more web servers specifically for NOvA to handle the large load.
- Additional optimizations of how we actually record and query in our own offline code also significantly reduced the load on the web service and sped up user jobs.

Lesson Learned: Ability to Version and Patch Tables Should Be a Requirement

- The ability to “version” a table means that one can take a snapshot of the state of the database at any given time, and refer to that state and any time later.
  - Ensures reproducibility.
  - Requiring groups to version a table prior to large production campaigns ensures someone is paying attention to the state of the table.
- The ability to “patch” a table means that one can make small changes to individual entries in the table.
  - A version cannot be directly patched. A patch applied to a version would create a new “state”, likely requiring a new version.
  - Patches are important for tables with very large numbers of entries, especially conditions tables that span a large range of validity times.
- The Fermilab condb implemented versioning (via tags) from the very beginning. I believe patching is a work in progress.
- Versioning has played a critical role in NOvA offline production campaign, and has ensured reproducibility.