

RESPONSE TO CRITIQUE OF PROPOSED NOVA DBI REQUIREMENTS

X

Contents

Overview	2
Issues	2
Requirement Section	2
Design Philosophy	3
Other Concerns or Topics	4
Real World Example	4
Table Structure and Aggregation	4
Results	5
Followup	6
Bibliography	7

Overview

In the companion document (NOvA-DocDB-7705) I laid out some proposed requirements for the NOvA offline database interface and tries to justify them based on the reasoning behind the design of the MINOS database interface (which I believe is also adopted by Daya Bay and T2K with minor customizations) which fulfills those requirements. I also argue that this design is well suited to use by NOvA with only minor modifications; the implementation can start with the existing code base or from a clean slate only taking the concepts.

Issues

Requirement Section

1. *The document is titled "Database Interface Requirements", yet discusses the distribution system for synchronizing databases at remote sites. IMO, synchronization is not part of the DBI (although it is a critical issue that needs to be resolved), so that should be left out.*

Fair enough, though I would contend that the MINOS need for synchronization impacted the DBI/table design. The two-table structure with the common SEQNO grouping was the granularity for checksums. With this, the system could identify missing rows and those with values that differed from the master DB. The DBI was also used as part of the synchronization process and the DBI's ability for reflection (determining the table's structure) was important.

2. *The document does not specify things like "acceptable query times". One thing we need to know is how long users should reasonably have to wait for a table "load" should take. NOvA has (or will have) a lot more channels/rows per run/subrun than MINOS. Is 5 seconds acceptable? Is a minute? The DB Apps group needs this information as they develop their algorithms to balance data compression vs. query time.*

This kind of requirement can only be answered in a context of "reasonable". For a job that is going to run hours, say for reconstruction, then a significant fraction of a minute would be acceptable; for interactive work such a time would no doubt be annoying. Generally, most large tables are for data that gets applied early in the processing (such as calibration) chain in a reconstruction job and then never needs to be accessed again for that file, so long times aren't much of an impact. One can only really time such a query empirically in the most realistic real-world setup one can construct. Such a setup should cover the full detector in a database populated with data much like one would expect near the maturity of the experiment. I've attempted to do such a test, which I've described below.

3. *Requirement #4: I need clarification on this, eg, is testing in the development dB not enough?*

Generally, no. Validating the data to be entered into the database often requires that detector data be processed with only the one table's data modified. Unless there is some provision for segregating the development DB for multiple users that approach isn't usually viable due to the potential for a clash.

Design Philosophy

4. *Currently we are treating the `_offline_` [calibration/alignment] tables as being based on run/subrun. This is because the "merging" scheme we are trying to put together to compress the data is much easier to do when looking at discrete numbers (eg, subrun) than when considering time. Besides, the rows in these tables can only have validities corresponding to runs and subruns (and detector and software release and...).*

I think I laid out in the original document why a time based system is superior. It is more natural for users. It is more flexible: one can map a subrun to a time range, but can not use a subrun base key for something that occurs in the middle of the subrun. A time ordered table structure is natural for DCS data (or data derived from such sources) because the validity of such data is uncorrelated with run/subrun boundaries.

There seems to be some confusion about how ranges were set in MINOS. While the data used to create the values might have come from t_1 to t_2 , the data is often declared valid for the time t_1 to $t_2+\epsilon$, where ϵ is chosen on a table-by-table basis depending on the table author's estimation of how long the constants will continue to remain valid. For some kinds of data that ϵ might be zero, but for most it is not. New entries will then obscure part of the older entry hiding the ϵ part. This allows there to be no gaps. The MINOS system does assume that files will be processed in order if the creation date is not tied to the data and to keep the creation times unique if one does and the data is reprocessed. I believe the T2K schema introduces another layer that can be used to force later passes to take priority.

5. *Now obviously "stuff happens" between and during a run, eg, channels going dead for certain periods of time. For these types of tables, we will have a timestamp-based type of table, but here we don't have the kind of data compression issues we have with the offline tables, so dealing with these is pretty straightforward.*

Why have multiple table designs when one will do?

I think part of the problem of need for "compression" is that processing is being done in a manner that drives this, but is not a good match to the needs of the constants. In MINOS data is reconstructed using existing values of the database ("keep-up" processing) and multiple files from that pass are *then* processed collectively in order to actually calculate the calibration constants. Very few tables are updated on an individual file level. For instance, MINOS strip-to-strip calibrations were updated weekly, drift calibration daily, mip values 20 times in entire lifetime of the experiment, (pulsar) gain (?need to ask experts about frequency).

This avoids any need to do merging (which itself invalidates the "never modify constants" restriction for repeatability) as well as being more naturally tuned to the time constants of the calibrations. Another driver in using multiple files is that often there aren't sufficient statistics (muon tracks through any given cell, etc) in order to update the constants unless one used a period longer than a subrun.

6. *Regarding the Text-based override, in the examples you gave, I am unclear why the change must happen within the database. I would think that one would extract the original data from the dB, and then manipulate it through the Table class. Why does one have to write anything to the database here?*

Text based overrides don't *have* to happen within the database; this wasn't in the requirements section (Requirement 4 is technology agnostic). The description in the discussion section of how MINOS did it was meant to be informative of an approach that worked, not necessarily the only possible means of accomplishing the requirement.

I'm confused about the second half of this question. One has to be able to override what is found in the database with an alternative set of constants. What I don't see how the user has any hooks available for manipulating the data via the Table class when the DBI call is embedded in some regular code such as the Calibrator without modifying the Calibrator code. I'm not seeing what the proposal is that allows users to test new constants.

Other Concerns or Topics

7. The "compression" scheme that I saw discussed can be similarly implemented in the MINOS DBI framework by simply extending the validity time range for the first entry (i.e. choose an ϵ that covers the time one desires), calculating the new values and committing to the DB an aggregate entry if-and-only-if the contents significantly differ from the already existing entry. This approach involves no modification of the database entries and allows full reproducibility and no additional external DB maintenance or compression procedures. The only missing element is a criteria for comparing two data sets and determining if they are close enough (which would need to be developed in any case for "compression").
8. I can image one non-standard table format that might be desirable, or at least code that can interact with a standard table in a non-standard way. Normally the DBI maps a timestamp to some data. The one case is that one might want the standard table structure have validity ranges that represent the start and end of a subrun, and where the main data table is the tuple (*detector, run, subrun, filename, other subrun summary info...*) and one would like to access the information using `detector + run + subrun` as the key.
9. Another issue that was touched upon in our a verbal discussion was the desire to tie data to "releases" (software or production). This could be handled in this framework with the addition of one more non-standard table. This would need to map (`tablename, release`) to `task number` and then use the existing task number capabilities. This would allow multiple "releases" map to the same task.

Real World Example

For this worst-case test I posited a situation where there was two floating point values for each channel, all of which were updated every hour. This is definitely not the norm for data in the database. So far the one case I've seen discussed for NOvA is occupancy numbers done on a subrun basis.

Table Structure and Aggregation

For this trial I've designed a table that holds two numbers per channel. Each table row holds the values for a module of 32 channels (64 floats) + plane + module numbers (2 integers). The data is aggregated at the plane level (i.e. one VLD entry per plane and each far detector SEQNO has 12 entries in the main data table). This aggregation scheme works for all three detectors. (albeit with fewer data entries per VLD for the smaller detectors). One could reduce the fraction consumed by of VLD entries by aggregating by at a yet coarser level (groups of planes) and this could be done differently for the different detectors; it is determined solely by convention of the

table filler. Larger aggregations means more data must be repeated if any value is changed (here it a whole plane's data even if only one channel).

The far detector there are assumed to be 30 blocks of 32 planes per block for a total of 960 planes; for comparison the NDOS detector has roughly 25 time fewer channels. Entries have validity ranges that are 1.5 hours long, so each overlaps part of the previous (as this could force some extra work to be done on the client side). The data values themselves were arbitrary random numbers pulled on the fly during filling; as full floating point values this maximized the amount of text that must be formatted and exchanged between server and client (no artificial squashing of values).

For this test I used the MINOS `offline_dev` database (table collection). The `offline_dev` is handled by the same MySQL server as the production tables. There shouldn't be anything fundamental in this test that depends on MySQL vs PostgreSQL; the constraint was that the version of ROOT available with the MINOS code didn't have the PostgreSQL interface built so that it was accessible via the generic TSQL classes. This MySQL server has recently migrated to a new(er) machine; the machine is a year old but the recent migration means that I lack access to directly look at the file sizes and there was no ganglia monitoring so I can't see the servers load or network traffic.

In a couple of hours it was possible to (1) design the table layout and aggregation, (2) write the class object including its ability to write data to the database as well as receive it, (3) write scripts to create the tables and fill them with artificial data, (4) write a script to retrieve the data.

A full set of entries (960 planes of 12 modules, or 11520 data rows) were made for 1630 validity periods; this would be the equivalent of 68 days for 1 hour subruns. If instead this table were updated on a daily basis this would represent roughly 4 years of accumulation. The table filling was limited by the failure of my home router's connection to the lab (running interactively on `minos54` to fill the DB) which terminated the procedure.

I calculate what did go in as approximately 7Gb on disk based on numbers retrieved using `df` and the SQL query: `show table status like "%NOVA%".` The files aren't visible to anyone but the `mysql` account on the server node.

Results

A typical (fresh) query that required going to the database rather than simply using its cache had statistic of the sort:

```
DbiTimer:NOVAOCCDBI: 11520rows, 3.22560Mb Cpu 15.49000 , elapse 18.27551
```

This time includes querying the DB, creating the row objects, retrieving the TSQL rows, and filling the row objects. This time was dominated by the query itself. Repeated queries, for event times that stepped by 1 second intervals, showed the expected behaviour of not accessing the DB server until the cache's validity range was exceeded. What was unexpected was the DB behaviour of alternating with times of the sort:

```
DbiTimer:NOVAOCCDBI: 11520rows, 3.22560Mb Cpu 0.31000 , elapse 3.38966
```

I haven't had sufficient time to investigate the cause of this behaviour; it appears to always be the first, third, fifth, etc access to the DB that take the long time. This could be a result of the way the internal caching is done. For the purposes of discussion we can simply assume the worst case time. I would have to resume filling the database with additional data to verify that the access time doesn't significantly degrade as additional entries are made. I would not expect this to be the case as the key elements are DB indices and the time to access those should be better than linear (b-tree should be logarithmic).

Followup

I see that the results reported results from Jon are for the NDOS, so I should clear the table and repopulate it with a dataset comparable to that (fewer channels, planes, 2031 subruns). I can also look into installing a version of PostgreSQL on the MINOS machine and building ROOT against it. Then if someone could point me to the right PostgreSQL server I can safely scribble on, I could try that.

Bibliography

Author Last Name, First Name. "Book Title or Reference Title." City: Publisher, Date.

MINOS DBI Package Overview

http://www-numi.fnal.gov/offline_software/srt_public_context/WebDocs/Package_Overviews/DatabaseInterface.html

MINOS DBI Package Rationale

http://www-numi.fnal.gov/offline_software/srt_public_context/DatabaseInterface/doc/dbi_PR.html

MINOS DBI User Manual

http://www-numi.fnal.gov/offline_software/srt_public_context/doc/UserManual/node9.html

Representation of times and dates

http://en.wikipedia.org/wiki/ISO_8601