Q&A For NYOUG/Viscosity Presentation
"Using a PL/SQL Table API to Implement SmartDB/Thick Database Principles"

Q: Is there a limit to the number of error messages?

A: This question relates to the jobs_tapi.check_jobs_insert() sample procedure where error messages are concatenated. The message limit would be the size of the variable that is storing the message (in the sample, 4000 characters). The UI language may also have a limit for the buffer into which the message is returned. To avoid variable oveflow errors like ORA-01044, production code for error_pkg.concat_msg() would include a length check before concatenating to ensure that the message variable size is not exceeded. If you expect error messages to possibly exceed your database version's VARCHAR2 limit (up to 32767 for 12c+) you can use alternatives like the CLOB datatype or store the message in real or PL/SQL table rows and then query that table to return all relevant rows to the UI.

Q: What's the advantage to having Client/UI perform COMMIT instead of the procedure?

A: Ok, if you are always using Method 1–calling the ins() procedure directly from the UI–the location of the COMMIT is a toss-up, but you would also need to handle ROLLBACK in the case of a "DML" failure.

If you are using Method 2– "DML" to the view, with an INSTEAD OF trigger–adding COMMIT to a trigger (or procedure called from a trigger) is messy (https://asktom.oracle.com/pls/apex/asktom.search?tag=commit-in-a-trigger). Also, if you need cross-row or cross-table validation, you may get a mutating table error if you try it from within the trigger (or procedure called by the trigger). In those cases, for Method 2 after the "DML" to the view, you will need a separate procedure call to run those types of validations. The COMMIT and ROLLBACK could be included as part of the code for that procedure, but since you need the UI to COMMIT or ROLLBACK for non-cross-row/cross-table situations, you would need to remember to code those transaction statements sometimes but not other times. That seems messy so I'd just say that the UI should ALWAYS COMMIT/ROLLBACK for Method 2.

Q&A For NYOUG/Viscosity Presentation
"Using a PL/SQL Table API to Implement SmartDB/Thick Database Principles"

Q: If i am not wrong, Oracle EBS applications are written using all the rules you are mentioning here including Table APi. What about Cloud ERP development?

A: I'm not too in-tune with Oracle Apps, but my pretty-sure guess would be that if you are not wrong about EBS using a TAPI, Cloud ERP would also do that. Oracle has been into this technique for a long time. For example, back in the days when Oracle CASE (later "Designer") was in use maybe 25 years ago(?), one key feature was that it would generate TAPIs – in fact, that's where I found the best patterns for my own TAPI code.

Q: Don't you think that there are too many calls from application server to db and degrade the application performance when all the validations are written in PL/SQL.

A: If you're talking about network messages, which is a significant performance factor, there are none because the PL/SQL code is stored in the database not the application server. So calls from the database to the database use no network resources. You may be talking about simple validations, some of which were in my examples, like max salary >= min salary. That one, I'm actually in favor of repeating in the UI (not the app server because of the messages required) because it is very unlikely to change. But notice "repeating" – it should also be coded in a database procedure in case a UI developer misses coding it in the UI.

Other simple validations that are coded as database constraints like formats for dates and numbers and required values should also be in the UI. Same for simple validations that are unlikely to change like that an age is positive and number of children is an integer, but in those examples, a constraint may not be available for the database side so PL/SQL is necessary. It is important that any validation that requires interaction with the database – to query data for example – needs to be in stored PL/SQL code not in UI or app server code. There may be those who think that app server logic can offload a lot of work from the database side, but if the app server is doing queries, it is generating network messages to the database anyway.

As for piling up lots and lots of PL/SQL calls within a single procedure, PL/SQL is a lightning fast language at this point and slowdowns will only be due to inefficient queries, which is a developer's problem to solve not a problem of where to put the code.

Q&A For NYOUG/Viscosity Presentation
"Using a PL/SQL Table API to Implement SmartDB/Thick Database Principles"

Q: is postgres the same as this thick databse?

A: Postgres is a database that includes a procedural language, PL/pgSQL, that looks very much like PL/SQL and is also stored in the database. So you could code PL/pgSQL program units within your Postgres database.

Q: How do you sell this idea to Java developers?

A: This is related to the one question I did answer during the session. If you have database developers in your group, you can sell this idea to Java developers with the major reason that it removes a lot of logic from the Java developers' plates, which means they can concentrate on what they know and probably therefore like to work with best. If you don't have database developers in your group, it's harder to sell but the main selling point, which is not nearly as compelling, is that it is more efficient. (Watching Toon's video and showing his research results can provide the proof.) Most coders would like their code to be efficient but in the interests of project schedule pushes, sometimes getting it done unfortunately takes precedence. Also, using Thick Database strategies may be a culture change which is difficult without management support. Maybe your management would be more receptive to Toon's proofs than your Java developers? The savings in cloud costs from much lower compute times could be a motivation.

Q: Why do you use „:NEW.ID" in the Delete Trigger and not „:OLD.ID"?

A: Excellent catch! (emp_details_vw_trbr.trg) Thank you! Well, clearly I didn't test that part of the code; it should be :OLD.employee_id. I've made that change in the sample code available at https://bit.ly/tapi_code. Thanks again.

Q: What about mass DML operations?

A: The examples were all based on a UI with a single-row. For array processing on a multi-row UI, you would create separate procedures (Method 1) that take array (table or collection) parameters and do FORALL "DML" instead of single-row "DML." The trick is to get the UI to collect the rows into a collection, but some UI tools have objects available for that already. Method 2 (view with INSTEAD OF trigger) does not work as well because

there are no statement-level view triggers. I wonder if you could represent a collection like a VARRAY as a column in a trigger, and in that way pass multiple rows to a view INSTEAD OF trigger? Probably better to go with Method 1 instead.

Q: Please provide example(s) of how trigger constraints fall short in cross-row cross-table constraints.

A: Answered a bit above but say you want to update an employee salary and make sure it is not greater than the salary of the president, you would be querying the same table you are updating. This will cause a mutating table error. You can circumvent that effect in some cases by use of autonomous transaction procedures, but the data you are querying may not be current enough to ensure data integrity. Cross-table queries are also susceptible to mutating tables if there is a relationship between tables, and, for example, the table is defined with CASCADE DELETE.
https://asktom.oracle.com/pls/apex/f?p=100:11:0%3A%3A%3A%3AP11_QUESTION_ID:9579487119866

Q: Performance issues when doing DML using views?

A: You may be thinking that running a query on a view requires an extra step for the database to look up the view text in the data dictionary tables. That step is not required when querying tables, of course. The time required for the database to look up view text is insignificant and would not affect performance. Also it would be surprising to me if "DML" statements issued to views would even need to look up view text because no query is being run. The logic followed by the database would be to ignore the view text and just look for and run the INSTEAD OF trigger. If your question is about DML (including SELECT) not "DML" (excluding SELECT) and you are wondering about the performance of queries to a view, performance depends on how well you've written the SELECT statement. Please reply if I'm missing some point but I've never run into any performance issues when doing "DML" to views.