Using a PL/SQL Table API to Implement SmartDB/Thick Database Principles

Peter Koletzke
Technical Director &
Principal Instructor

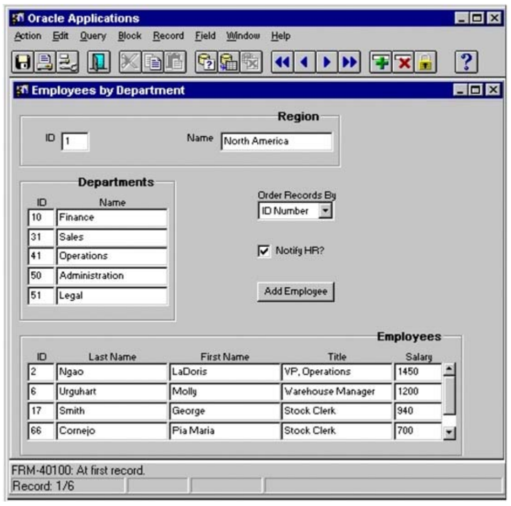
A Questions/Answers document appears at the end of this file

ACE Director

MICHIGAN ORACLE USERS SUMMIT

Scenario 1

- Legacy Oracle Forms application
  - Originally built in SQL*Forms 3.0
    - No PL/SQL in the 6.0 database
  - Migrated to current version
    - PL/SQL remains in the application
    - Little or no stored PL/SQL



Region

ID 1 Name North America

Departments

ID	Name
10	Finance
31	Sales
41	Operations
50	Administration
51	Legal

Order Records By: ID Number

[✓] Notify HR?

Add Employee

Employees

ID	Last Name	First Name	Title	Salary
2	Ngao	LaDoris	VP, Operations	1450
6	Urguhart	Molly	Warehouse Manager	1200
17	Smith	George	Stock Clerk	940
66	Cornejo	Pia Maria	Stock Clerk	700

FRM-40100: At first record.
Record: 1/6

Scenario 2

- Java application developed by Java developers
  - Java developers designed the database objects
  - All database access logic written in Java code
  - Little or no stored PL/SQL

PL

Java

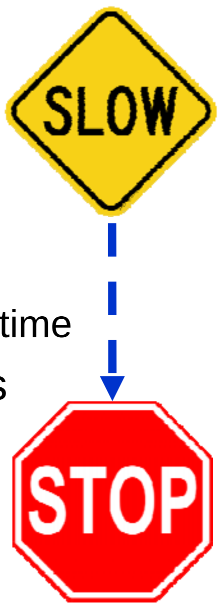
Scenario 3

- JavaScript application
  - Client or server JS accesses database objects
  - Little or no stored PL/SQL

PL

Result

- Works in Test and QA, even in UAT
- Works initially in Production
- Performance lags with increased usage:
  - Logic accessing database requires time
  - Excessive network messages to the database takes time
- **Also**: changing development languages requires a complete rewrite



Solution: Thick Database!

Agenda

- What is Thick Database?
- Implementing the Table API
- Why Thick?
- Variations and Strategies

Slides and Q&A, and recording will be on *Mike Gangler's Musings.*

Code samples at bit.ly/tapi_code

About Thick Database

- A code development strategy
  - Maximize use of database code to simplify the user interface
  - The user's device (client) runs minimal code
- Name plays off the term "thin client"
  - A "Year of the Internet" term
  - Means most processing occurs on a server
  - Thick database means "thin client"
- Dr. Paul Dorsey of Dulcian was an early proponent

A.k.a.
- Thick Database Approach
- Thick Database Paradigm
- Fat Database
- Smart Database
- SmartDB

Oracle prefers

Important!

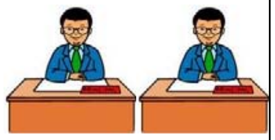
- ***The realization of ThickDB principles is up to the practitioner***
- There is no “One Way”
- What follows is the “My Way”
  - Successfully used for over ten years
  - Implements basic ThickDB/SmartDB strategies
    - Centralize code in the database
    - Minimize code on the client side
    - Protect code from tampering

Companion Strategy

- System for tracking business rules
  - System requirements/specifications written in user-friendly syntax
  - Store definitions in database tables
- With a bit of effort
  - Can link business rules to code implementations and test plans
- With more effort
  - Can generate code from business rule definitions
- With significant effort
  - Application can read business rules dynamically and generate code on the fly

ORACLE APPLICATION EXPRESS

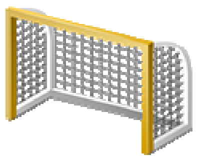
ORACLE BUSINESS RULES



The Goal

- **Maintain data integrity using database objects**
  - Database constraints only go part way
  - Triggers supplement constraints but are still not enough
    - Fall short for cross-row, cross-table validation
- Procedural code fills the gap
- Still missing: CREATE ASSERTION
  - Stores validation as a database object
  - Applications cannot corrupt data integrity

https://community.oracle.com/ideas/13028



Where Do You Put the Code?

- **Goal**: Always validate data using database code
  - Constraints and triggers enforce data integrity
  - Force table access to be through API only
  - Data will be valid regardless of the application
- **But**: simple, non-SQL validations can be ***repeated*** in client or application server code; e.g.:
  - “Number of children is a positive integer.”
  - “Birthdate is on or before today.

Bottom Line

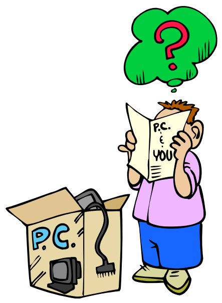
Put your application code in the database!

Agenda

- What is Thick Database?
- Implementing the Table API
- Why Thick?
- Variations and Strategies

Protect Code With Multiple Schemas

- Two
  - Connect schema
  - Table owner schema (owns PL/SQL code and tables)
- OR Three
  - Connect schema
  - PL/SQL schema (bus rules and TAPI code; views) – (AUTHID DEFINER)
  - Table owner schema (owns tables)
- OR Four [(1)]
  - API and bus rules code are separated
- Synonyms - optional
- DDL triggers – to guard against additional grants or disabling triggers or creating synonyms



(1) *Why Use PL/SQL?* By Bryn Llewellyn

Table API

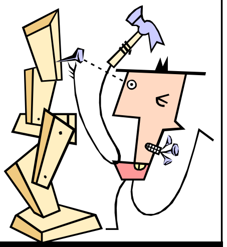
- A PL/SQL package per table
  - All data modification ("DML") is accomplished through procedures
    - Single-row: INS(), UPD(), DEL(), LCK()
    - Multi-row: INS_ALL(), UPD_ALL(), DEL_ALL(), LCK_ALL()
    - Examples here are for single-row scenarios
  - Procedures call business rules validation code
- No grants to table (two-schema strategy)

Demo 1

Code samples at bit.ly/tapi_code

Two Calling Methods

- Both use data validation procedures
- Method 1. Called by the client or UI
  - Table API procedures, PL/SQL not SQL
- Method 2. Called by view trigger
  - UI client issues SQL to the database view
  - Database view trigger calls Table API procedures
    - Variation: add view API procedures that call Table API procedures
- In all cases, use the Table API



Method 1 - Called By Client or UI

- Procedure call instead of SQL for INSERT, UPDATE, DELETE
- Client/UI performs COMMIT
- Some tools are oriented towards SQL calls (for example, ADF)
  - Use Method 2 for these
- Benefit: access to any table and statement
  - Not true for Method 2

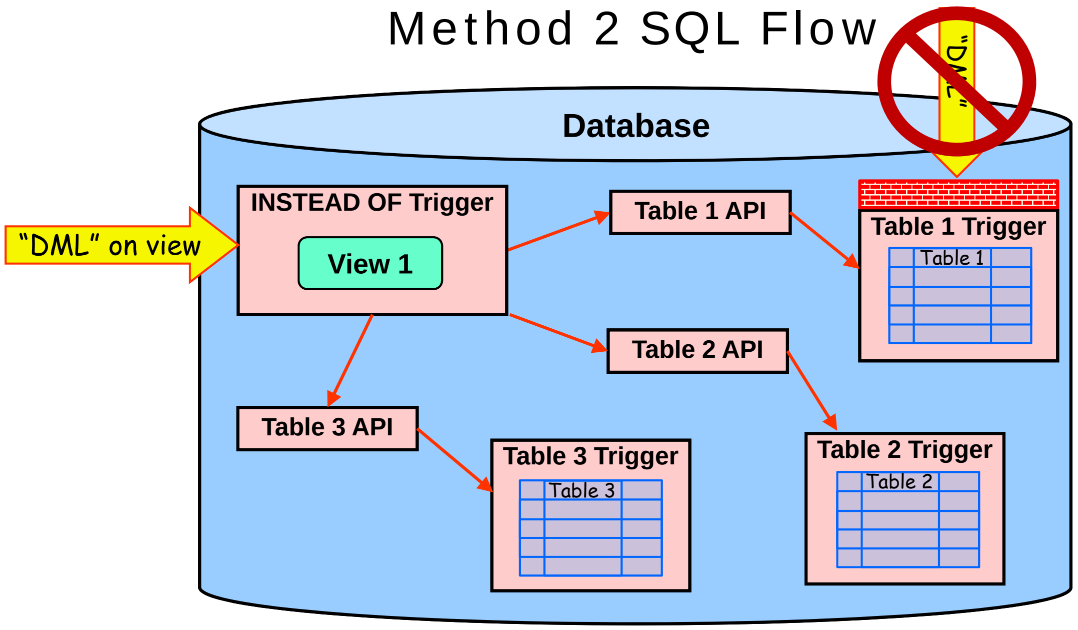
Demo 2

Method 2 - Database Views and Triggers

- Views on tables requiring access
- INSTEAD OF triggers on the views
  - –INSERT, UPDATE, DELETE row-level trigger calls TAPI
  - –Some validation needs special handling
    - Statement-level triggers on tables or non-transactional procedures
    - Cross-row and cross-table validation

Demo 3

Method 2 SQL Flow



Optional Table API Component – Both Methods

- Package enforcement BOOLEAN global variable
  - Trigger evaluates it to prevent “DML” statements outside of the package
    - Applies only to table owner because table has no grants
  - Set to FALSE by default
    - TAPI procedures set it to TRUE before issuing INSERT, UPDATE, DELETE
    - Set to FALSE after
  - Forces access only by Table API

Package Enforcement Global Variable

```
CREATE OR REPLACE TRIGGER employees_trbr
   BEFORE INSERT OR UPDATE OR DELETE
   ON employees
   FOR EACH ROW
BEGIN
   --
   IF NOT employees_pkg.g_allow_dml
   THEN
      RAISE_APPLICATION_ERROR(-20199,
         'You may not issue INSERT, UPDATE, or ' ||
         'DELETE statements to this table.');
   END IF;
   -- other code for validating rules
END employees_trbr;
```

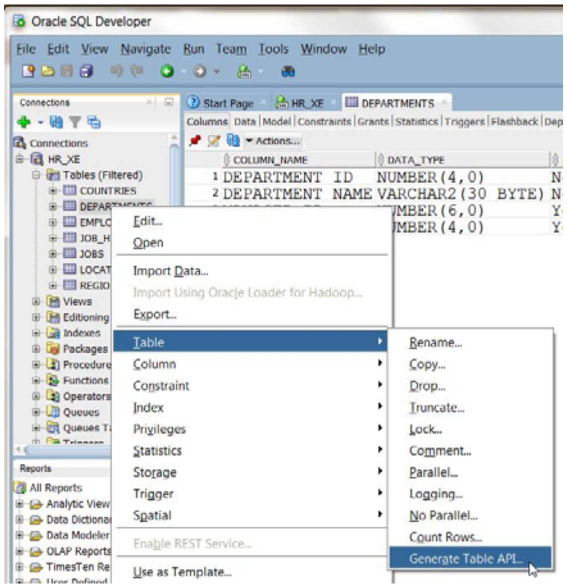
Generate the Stub Code

- It's all cookie cutter stuff at the start
  - Table API – triggers and packages
  - View INSTEAD OF trigger
- Use a prebuilt generator
  - Oracle DD Generator (SQL Dev Extension)
    - https://www.oddgen.org/
  - André Borngräber, Ottmar Gobrecht
    - https://github.com/OraMUC/table-api-generator
- Or roll your own generator

Demo 4

SQL*Developer Solution

- Select table
- From right-click menu:
  - Table → Generate Table API
  - Ctrl-F7 to reformat
- Use as a basis for your own code

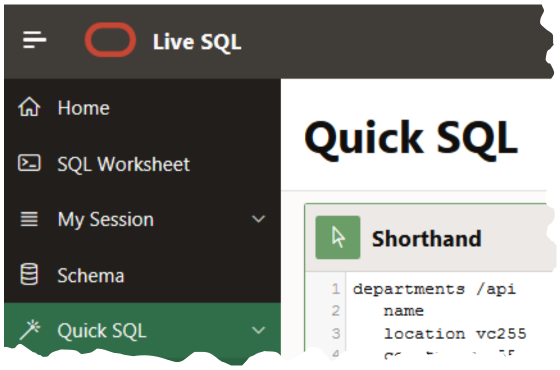


```
  COLUMN_NAME      DATA_TYPE
1 DEPARTMENT_ID    NUMBER(4,0)          N
2 DEPARTMENT_NAME  VARCHAR2(30 BYTE)    N
                   MBER(6,0)            Y
                   MBER(4,0)            Y
```

Demo 5

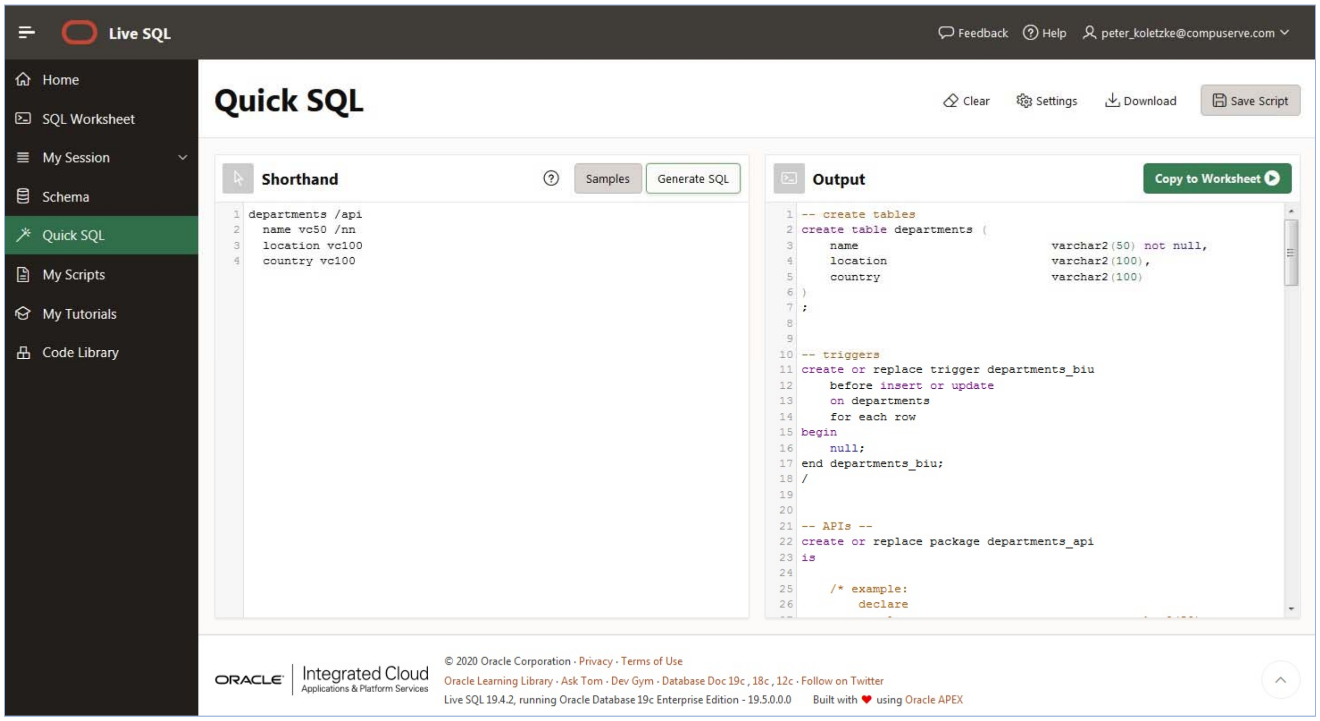
Another Tool: Quick SQL

- quicksql.oracle.com
  - An APEX app
  - Use your Oracle login
- Several youtube videos
- Shorthand syntax generates DDL code (even data!)
- Settings control variations
  - Table API is one of the options ("**`/api`**")



```
departments /api
    name
    location vc255
```

Demo 6



```
departments /api
  name vc50 /nn
  location vc100
  country vc100
```

```
-- create tables
create table departments (
    name                          varchar2(50) not null,
    location                      varchar2(100),
    country                       varchar2(100)
)
;


-- triggers
create or replace trigger departments_biu
    before insert or update
    on departments
    for each row
begin
    null;
end departments_biu;
/


-- APIs --
create or replace package departments_api
is

    /* example:
        declare
```

ORACLE | Integrated Cloud Applications & Platform Services

© 2020 Oracle Corporation · Privacy · Terms of Use
Oracle Learning Library · Ask Tom · Dev Gym · Database Doc 19c , 18c , 12c · Follow on Twitter
Live SQL 19.4.2, running Oracle Database 19c Enterprise Edition - 19.5.0.0.0   Built with ❤ using Oracle APEX

Agenda

- What is Thick Database?
- Implementing the Table API
- Why Thick?
- Variations and Strategies

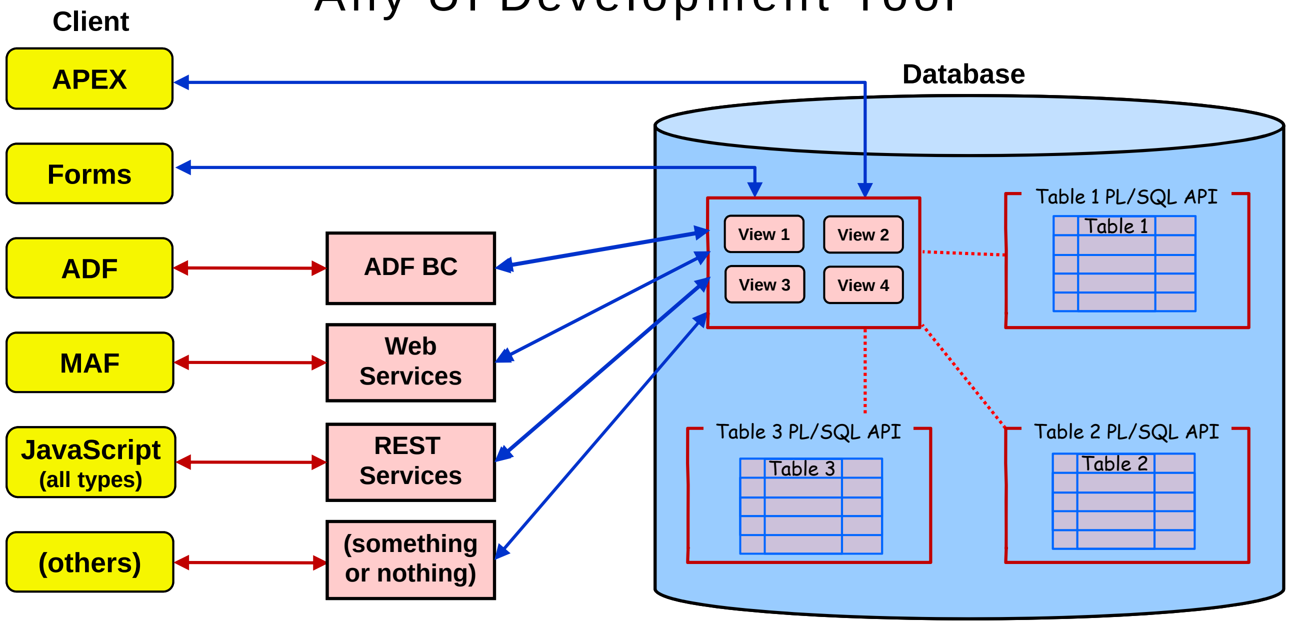
Thick Benefits

- Faster performance
  - Code is close to data storage – fewer messages, easy access
  - Views also reduce the number of round trip messages needed
  - Result: saves cloud costs
- Proper use of staff – UI (client) and DB code separation
  - User interface developers can concentrate on UI code
  - Database code developers can concentrate on database code to support the UI
- Productivity
  - Can greatly simplify user interface code

Some Changes Require Less Rewriting

- UI technology changes
  - If most code is in database, only UI needs rewriting
- Table refactoring
  - For example, if a set of tables used in UI views is normalized into more tables
    - Joins and query of view can be updated
    - UI may not need to change

Any UI Development Tool



Better Performance

- Toon Koppelaars' Research – youtube video
  - "NoPLSql and Thick Database Approaches with Toon Koppelaars"
  - Links later in the presentation
- Now uses the term "SmartDB"
- Tests to compare row-by-row processing for:
  - App in plain Java (no framework)
  - App in PL/SQL (not recommended)
  - App in PL/SQL using set processing (better)

Indicates content used with Toon's permission

Toon's Test Scenario

- Row-by-row processing
- Implement a number of business rules
  - Process 5 million rows
  - Various types of SQL statements
- Capture time elapsed and database CPU use
  - PL/SQL – No app server process
  - NoPlSql (Java) – JVM on database server, no network

Results

- NoPlsql to ThickDB (PL/SQL)
  - – Elapsed drops by 3X → #ThickDB is faster
  - – DB-CPU drops by 2X → #ThickDB is more scalable
    - Fewer app servers needed to scale for more users
- Why?
  - – Java start up and shut down for each statement
  - – Java requires messaging across network
- Comparing NoPlsql to **set-based** PL/SQL
  - – PL/SQL is 62x faster

Gets work done faster while at same time using less CPU

33

Drawbacks of ThickDB

- Time and effort required
  - – Design and set up (there is no One Way)
  - – Documenting standards
  - – Instructing staff
- Requirements on the IT shop side
  - – Architect/database designer
  - – Expert coder
    - Develop generic code “engines” to run and/or generate business rules code
- Need buy-in from management
  - – For all of the above

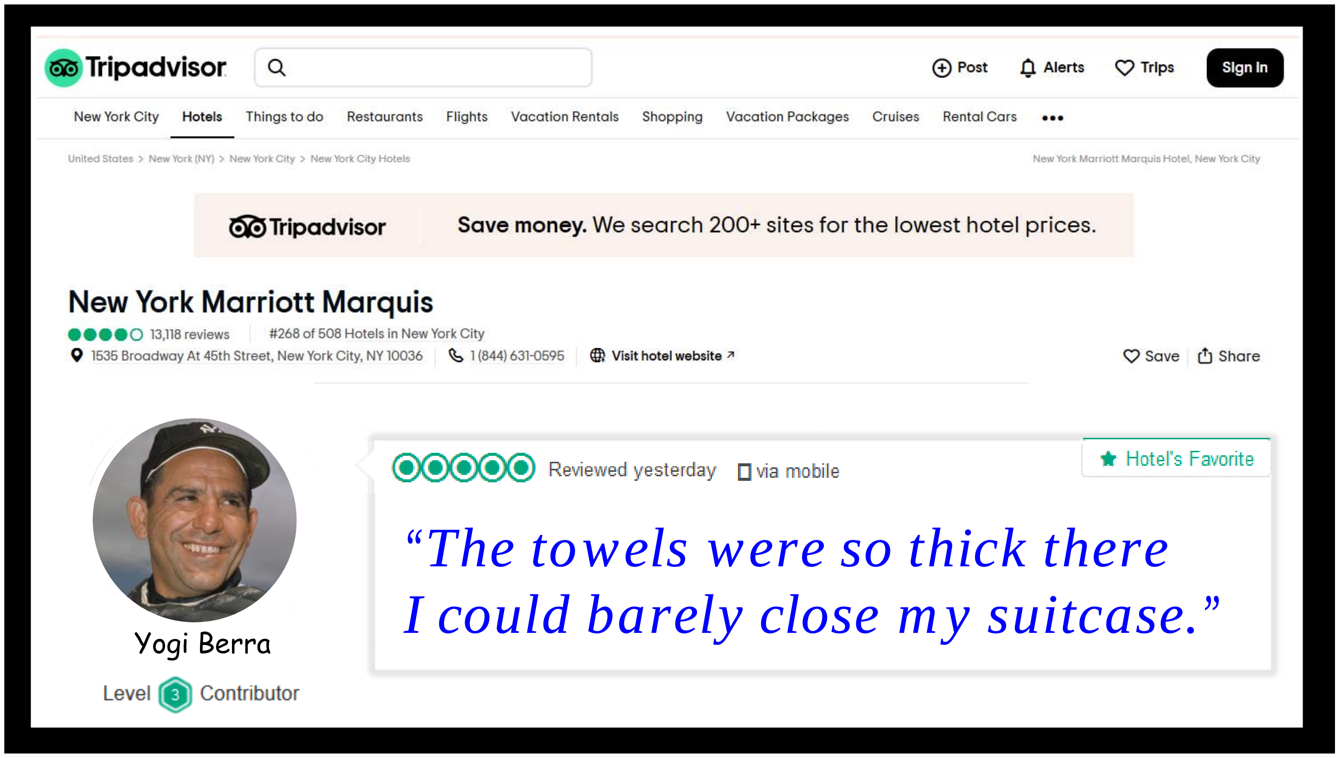
34

When Not to Use Thick Database

- If your organization is dedicated to “database independence”
  - Changing from Oracle to SQL Server, for example
    - This is not a simple endeavor; arguably more expensive
  - Forces applications to use ANSI SQL only
    - Applications become “thicker” than the database
  - Product app companies may need to be DB independent
- If your applications have few or simple business rules
  - Overhead of ThickDB may not be worthwhile

Agenda

- What is Thick Database?
- Implementing the Table API
- Why Thick?
- Variations and Strategies



Tripadvisor

Post Alerts Trips Sign in

New York City · Hotels · Things to do · Restaurants · Flights · Vacation Rentals · Shopping · Vacation Packages · Cruises · Rental Cars

United States > New York (NY) > New York City > New York City Hotels

New York Marriott Marquis Hotel, New York City

Tripadvisor **Save money.** We search 200+ sites for the lowest hotel prices.

New York Marriott Marquis

●●●●○ 13,118 reviews | #268 of 508 Hotels in New York City

1535 Broadway At 45th Street, New York City, NY 10036 | 1 (844) 631-0595 | **Visit hotel website** ↗

Save | Share

Yogi Berra

Level 3 Contributor

●●●●● Reviewed yesterday ▯ via mobile

★ Hotel's Favorite

"The towels were so thick there I could barely close my suitcase."

Variation

- Call a PL/SQL function to retrieve data
  - No database view needed (no SELECT)
  - Virtual Private Database policies can filter data to all SELECT statements instead
  - You are responsible for handling the result set
  - Tools oriented towards SQL need more code for this
  - More flexibility in parameters

Do You Need an Oracle Database?

- No, but…
  - A central location for business rules code is necessary
    - Best in a database
  - Views are needed to hide details of the data storage
    - INSTEAD OF triggers may not be available
    - So application may be responsible for calling the central code
  - Table API concept may be possible
    - DB2 supports PL/SQL
    - Postgres has PL/pgSQL
    - You can always just allow access to views not tables

How to Transition to Thick Database

- Like applying any other standard while “in flight”
- Apply it 100% to new applications
- Apply it to enhancements for existing applications
- Start small
  - Incorporate user interface interaction with database views
- Refine as you go
- “Completely Thick” can be a longer-term goal

Thick Database Team Success

- If your team is expert in a development discipline other than PL/SQL, selling ThickDB is difficult
  - Experts in database design, SQL, and PL/SQL do not have deep UI skills
  - Experts in UI do not have deep DB skills
  - Or, at least, these are very rare
- Success lies in “divide and conquer”
  - Small team of DB developers
  - Any size team of UI developers
  - Exact numbers depend on the workload

Resources

- www.dulcian.com
  - Look in Resources | Conference Presentations…”thick database”
- Steven Feuerstein – Oracle Database PL/SQL and EBR Blog
  - https://blogs.oracle.com/plsql-and-ebr/entry/why_use_pl_sql
  - https://blogs.oracle.com/plsql-and-ebr/noplsql-versus-thickdb
  - http://stevenfeuersteinonplsql.blogspot.com/2018/05/the-smartdb-resource-center.html
- Toon Koppelaars
  - https://www.youtube.com/watch?v=8jiJDflpw4Y
  - http://www.prohuddle.com/webinars/ToonKoppelaars/ThickDB.php
  - https://community.oracle.com/docs/DOC-1018915

More Resources

- More Toon
  - http://thehelsinkideclaration.blogspot.com/2017/06/my-noplsql-versus-smartdb-deep-dive.html
  - #SmartDB Office Hours Archives
    - https://asktom.oracle.com/pls/apex/f?p=100:551:::NO:551:P551_CLASS_ID:3241:
- Mike Smithers' Blog
  - https://mikesmithers.wordpress.com/tag/thick-database-paradigm/#post-6712
- Ask TOM Office Hours – PL/SQL topics
  - https://asktom.oracle.com/pls/apex/f?p=100:500:::NO:RP,500::

Summary

- Thick DB emphasizes database code
- Thick Database can improve UI simplicity, productivity, system performance, application accuracy, security
- There is no "One Way" to implement
- Table API is the core technique
- Call TAPI from view triggers or the UI/client
- Incorporating ThickDB requires some ramp-up time: use a phased approach

Designer Handbook
Developer Advanced Forms & Reports
JDeveloper 3 Handbook
ORACLE9i JDeveloper Handbook
ORACLE JDeveloper 10g Handbook
ORACLE JDEVELOPER for Forms & PL/SQL Developers: A Guide to J2EE Development
Oracle JDeveloper 11g Handbook
A Guide to Fusion Web Development

Thank you for attending this MOUS Monday

- Code samples: bit.ly/tapi_code
- Slides and a Q&A document here:
  - *Mike Gangler's Musings* (https://mjgangler.wordpress.com/)

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: 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: 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&A For NYOUG/Viscosity Presentation
“Using a PL/SQL Table API to Implement SmartDB/Thick Database Principles”

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.

###