Leverage Oracle SQL and PL/SQL to Simplify Development of Any User Interface Application Using Thick Database Techniques

Peter Koletzke
Technical Director & Principal Instructor

Agenda

• What is Thick Database?
• Thick Database techniques
• Level of Thick

Appendix A - More About Business Rules
Appendix B - Code Samples

Provenance

• Topic is rarely seen now, but not new
  – Using database features to enforce data integrity defined by business rules is obvious
• Started trending many years ago
    • Organized by Dr. Paul Dorsey of Dulcian, Inc. – 2001-2004
  – Thoughts evolved into Thick Database
    • Sessions starting around 2006
**Topic is Still Active**

- Dulcian.com
  - Look in Resources | Conference Presentations
  - Thick Database
- Mike Smithers’ Blog
  - https://mikesmithers.wordpress.com/tag/thick-database-paradigm/
- “Fill the Glass Episode 3 with Cary Millsap & Toon Koppelaars”
  - https://vimeo.com/128613885
- Bryn Llewellyn, Oracle Product Manager

**Guiding Principle**

- Database code that implements business rules
- Database views to represent complex business objects
  - Each view has an accompanying application programming interface (API)
    - Written in PL/SQL
  - Interaction is with view and API

---

**What is a Business Rule?**

- A statement of a behavior, definition, or constraint that allows an organization to achieve its goals.
- Systems analysis is all about determining business rules
  - Often business requirements are equated with business rules
- Used to communicate business with business users
- A full definition of business rules can identify all aspects of an application
  - Possible exceptions: technical details like development software, server specifics

**Sample Business Rules**

- An employee is active in one and only one department at a time.
- The employee’s job end date must be on or after the job start date.
- The value of the “State” portion of an address in the United States must be from the list of US states (including the District of Columbia).
- Saving a change to an employee record archives the old version of the record in a history table.
- Only managers can view salaries other than their own for staff in their department.
**Thick Benefits**

- **Application accuracy**
  - Business rules match application code
  - Test plans can be generated from business rules
- **Productivity**
  - Can greatly simplify user interface code
- **Code reusability**
  - Ease of application maintenance
- **Better performance**
  - Code is close to data storage – fewer messages, easy access
  - Views also reduce the number of round trips needed
- **Proper use of staff**
  - User interface developers can concentrate on UI code
  - Database code developers can concentrate on database code to support the UI

**Simplifies User Interface Work**

- Database views can represent multiple tables
  - Arbitrarily complex logic
  - Aggregate functions: MAX(), COUNT() 
  - Set operators: UNION, MINUS 
  - Calculation functions: first_salary() 
  - Even: a PL/SQL function cast as a table
- One view per application UI page
  - The page submit commits the entire page
  - Reminds one of mainframe “block submit”
  - Back end code deals the data into the proper tables

**Some Changes Require Less Rewriting**

- **UI technology changes**
  - If code is in database, only UI needs rewriting
  - Application logic in database can carry forward
- **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
Front-end Tool “Agnostic”

- Application Express (APEX)
- Application Development Framework (ADF)
- Mobile Application Framework (MAF)
- Forms
- JavaScript
  - JavaScript Extension Toolkit (JET)
  - Mobile Application Accelerator (MAX)
  - Application Builder Cloud Service (ABCS)
- PL/SQL Toolkit
- PHP: Hypertext Processor (PHP)
- Rails
- ColdFusion
- (whatever)

Tools’ Use of Thick Database

- APEX
- Forms
- ADF
- MAF
- JavaScript (all types)
- REST Services
- (something or nothing)

Drawbacks

- Time and effort required
  - Design and set up
  - 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

When Not to Use Thick Database

- If your organization is dedicated to “database independence”
  - Changing from Oracle to SQL Server, for example
  - This is a BIG DEAL
  - Forces applications to use ANSI SQL only
    - Applications are “thicker” than the database
- If your applications have few or simple business rules
  - Overhead of Thick Database may not be worthwhile
Agenda

• What is Thick Database?
  • Thick Database techniques
• Level of Thick

TripAdvisor Review:
Waldorf Astoria Hotel

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

—Yogi Berra

Database Components

• Tables – the usual
  – No grants or synonyms to other schemas
• Table API packages
  – INSERT, UPDATE, DELETE, (SELECT) procedures
  – Call business rules validation code
• Views on the tables
  – Queries can be arbitrarily complex
• INSTEAD OF triggers on the views
  – Call the table API procedures

Table API

• A PL/SQL package per table
  – All data modification (“DML”) is accomplished through procedures
    • INS()
    • UPD()
    • DEL()
    • LCK()
• Procedures are called only from INSTEAD OF view triggers
• No grants to table at all
Optional Table API Components

- A function can act as SELECT
  - A bit trickier and not always necessary
  - Virtual Private Database policies can filter data to all SELECT statements instead
- Package enforcement global variable
  - Trigger uses it to prevent “DML” statements outside of the package
  - Applies only to table owner because table has no grants
  - Access only by Table API

Code samples are available on the Quovera website.

Package Enforcement Trigger

```sql
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;
```

Database Views and Triggers

- Views on tables requiring access
- INSTEAD OF triggers on the views
  - INSERT, UPDATE, DELETE row-level trigger
  - Call Table API procedures
  - Except: cross-row validation requires statement-level triggers

SQL Flow

```
INSTEAD OF Trigger
View 1
Table 1 API
Table 1
Table 1 Trigger
Table 1
Table 2 API
Table 2
Table 2 Trigger
Table 2
Table 3 API
Table 3
Table 3 Trigger
Table 3
```

"DML"
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
  - Steven Feuerstein, for example
    - QXNO productized into QCGU (Quest CodeGen Utility)
      - Includes GUI interface
    - [https://community.oracle.com/community/database/developer-tools/oddgen](https://community.oracle.com/community/database/developer-tools/oddgen)
- Or roll your own generator

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
    - Allow access to views not tables

Agenda

- What is Thick Database?
- Thick Database techniques
  - Level of Thick

How Thick Do You Go?

1. Application code only
   - Conservative Thick Database Approach
2. Business rules repository for documentation
   - Modified Thick Database Approach
3. Code generation from the repository
   - Extreme Thick Database Approach
4. Applying business rules at runtime from repository definitions only
   - Ultra-extreme Thick Database Approach

All use views, table APIs, and INSTEAD OF triggers
Some Decision Points

- Conservative
  - Small shop, few applications
- Modified
  - Medium-sized shop, few “architects” (generic code authors)
- Extreme
  - Upfront time for investing in setup is available
- Ultra-extreme
  - Super-talented generic code authors
  - Team that can dedicate to this approach – analysts

How to Transition to Thick Database

- Like applying any other standard while “in flight”
- Apply it 100% to new applications
- Can apply it to existing application enhancements
- Can start small
  - Incorporate user interface interaction with database views

Oracle Does Not Support Plowing

Too thick to drink, too thin to plow.

—Common saying about Nebraska’s Platte River

Summary

- The Thick Database approach can improve productivity, system performance, application accuracy, UI simplicity, security
- Thick Database is driven by business rules
- Components:
  - Table triggers, minimal grants to tables
  - Database views with INSTEAD OF triggers, table APIs
- Different levels of business rules support: conservative, modified, extreme
- Incorporating it requires some ramp-up time
  - Use a phased approach
What is a Business Rule?

• A statement of a behavior, definition, or constraint that allows an organization to achieve its goals.
• Systems analysis is all about determining business rules
  – Often business requirements are equated with business rules
• Used to communicate business with business users
• A full definition of business rules can identify all aspects of an application
  – Possible exceptions: technical details like development software, server specifics

Business Rules Categories

• Business definition
  – A statement that explains a fact relevant to the business, for example:
    • An employee is active in one and only one department at a time.
• Data validation
  – A statement that describes how data is verified, for example:
    • The employee’s job start date must be or after the job start date.

More Business Rule Categories

• Allowed values
  – Related to data validation
  – Defines field values from a fixed list (hard coded or in a table) or range, for example,
    • The value of the “State” portion of an address in the United States must be from the list of US states (including the District of Columbia).
• System Behavior
  – A statement that guides the internal actions in the system, for example:
    • Saving a change to an employee record archives the old version of the record in a history table.
Another Business Rule Category

• Data privileges
  – Selective to users or (better) roles
  – Defines access to view or modify certain data, for example,
    • Only directors can update salaries for staff in their division.
    • Only managers can view salaries other than their own for staff in their department.
    • Personal data for clients, such as credit card numbers and Medicaid IDs, are only visible to staff who have been cleared to view it.
    • Staff may only view profile information for clients in the department’s territory.

Sample Business Rules

• An employee is active in one and only one department at a time.
• The employee’s job end date must be on or after the job start date.
• The value of the “State” portion of an address in the United States must be from the list of US states (including the District of Columbia).
• Saving a change to an employee record archives the old version of the record in a history table.
• Only managers can view salaries other than their own for staff in their department.

Test Plans and Business Rules

• Business rules statements can be used as or linked with test plans
• Generation of test plans is then just a report
• Each business requirement will be properly tested

Business Requirement

Business Rule
Test Plan
Req #

Where to Place the Business Rules Code?

• Environments these days are multi-tier
  – Database tier
  – Middle tier
    • Application server/web server (SOA, web services, ESB, etc.)
  – Client tier
    • Web browser
    • Mobile device
• Code can be located on one or more tiers
**Primary Assumption**

- Standard relational database constraints are ALWAYS used to protect data integrity
  - Primary key
  - Foreign key
  - Unique key
  - Check constraints
    - NOT NULL
    - Value- or function-base (optional)
- This is true regardless of the database vendor

**Code on the Client Tier**

- Web application consideration:
  - Since HTML is *not* a programming language, you need JavaScript for this
- Benefits
  - Fast feedback to user: very friendly
  - No processing at all on database or middle tiers
- Drawbacks
  - Difficult to maintain business rules documentation
  - Some browsers handle JavaScript differently
  - Possible need to repeat code for each app
    - Potential for omission in a single app
    - Not centralized

**Code on the Middle Tier**

- Business rules code is in the middle tier
  - That is, if there is a middle tier
    - For example, APEX has none
  - ADF
    - Java and XML files for the application
    - Declarative validation rules, EO, VO, App module code
- Benefits
  - Saves database server CPU time
  - Returns messages to user faster and friendlier
- Drawbacks
  - Each app needs to repeat the code for a particular table
  - Requires database roundtrip messages
  - Documenting or checking business rules requires visiting many files unless you use a Rules Engine or other repository

**Code on the Database Tier**

- Thick Database approach
  - Views
  - Table API code
    - Triggers and procedures (and policies) that enforce rules
- Benefits
  - Data integrity is enforced for all applications
  - Business rules code can be generated from metadata or, at least, documented from one source
  - Maintenance requires only database changes
    - Application modification may not be needed
  - Primary language is PL/SQL
- Drawbacks
  - Handling return messages from the database in a friendly way is not a default
  - Places complete burden of validation of data on the database server – possibly more CPU time taken
So, Which is Best?

- Depends on the application
- Database tier (Thick Database) ensures data integrity
  - Any application
- Middle tier saves database round trips
  - If processing only on middle tier
- Client tier provides best interactivity
  - Immediate feedback to user
  - Also saves database round trips

Feature Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Location of Business Rules Code</th>
<th>Client Computer</th>
<th>Application Server</th>
<th>Database Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>User interactivity</td>
<td></td>
<td>10</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Saves client computer resource usage **</td>
<td></td>
<td>2</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Saves roundtrip message to client computer</td>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Saves application server resource usage **</td>
<td></td>
<td>10</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Saves roundtrip message to application server</td>
<td></td>
<td>10</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Saves database server resource usage **</td>
<td></td>
<td>10</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>Saves roundtrip message to database server</td>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ease of maintenance (dependency analysis, adding, updating, reporting) *</td>
<td></td>
<td>2</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Reuse of code</td>
<td></td>
<td>2</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Assurance that business rules are applied to all applications</td>
<td></td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>61</td>
<td>64</td>
<td>77</td>
</tr>
</tbody>
</table>

* Assumes that the business rules repository is not used at runtime or to generate code.
** This feature reflects use of that tier for business rules purposes.

Suggestion

- Modified Database-centric Approach
  - Always code rules in the database
  - Selectively duplicate business rules in the middle tier and client tier
    - Carefully consider each rule
    - Know and document that you are duplicating rules
    - Can even turn off database rule for a transaction if it has been run on the client side
  - Consider using a BR repository tool
    - Home grown or Oracle Business Rules
- Guiding principles
  - Use database code when possible
    - It is the closest to the data == most efficient
  - Save database round trips when possible
    - Client side can check data type, for example

Some Challenges

- Identifying business rules
- Stating them accurately
- Representing the business rules in system programmatic code
- Defining and maintaining business rules statements
- Communicating rules to users
- Synchronizing programmatic code and the business rules repository
EMPDETAILS_VW View

CREATE OR REPLACE FORCE VIEW emp_details_vw
AS
SELECT emp.employee_id,
    emp.job_id,
    emp.manager_id,
    emp.department_id,
    dept.location_id,
    loc.country_id,
    emp.first_name,
    emp.last_name,
    emp.salary,
    emp.commission_pct,
    emp.email,
    emp.phone_number,
    emp.hire_date,
    emp.created_date,
    emp.created_by,
    emp.modified_date,
    emp.modified_by,
    dept.department_name,
    jb.job_title,
    loc.city,
    loc.state_province,
    cntry.country_name,
    reg.region_name
FROM   employees emp, departments dept,
    jobs jb, locations loc,
    countries cntry, regions reg
WHERE  emp.department_id = dept.department_id
AND    dept.location_id = loc.location_id
AND    loc.country_id = cntry.country_id
AND    cntry.region_id = reg.region_id
AND    jb.job_id = emp.job_id;

COMMENT ON TABLE EMPDETAILS_VW IS 'An all-inclusive view of an employee including all organization levels and current job description.';

EMPDETAILS_VW_TRBR Trigger

CREATE OR REPLACE TRIGGER emp_details_vw_trbr
INSTEAD OF DELETE OR INSERT OR UPDATE
ON emp_details_vw
FOR EACH ROW
DECLARE
BEGIN
    IF INSERTING
    THEN
        employees_pkg.ins(
            :NEW.employee_id,
EMPLOYEES_PKG Package

CREATE OR REPLACE PACKAGE employees_pkg IS

  --
  g_allow_dml  BOOLEAN DEFAULT FALSE;

  PROCEDURE ins ( 
    p_employee_id  employees.employee_id%TYPE, 
    p_first_name  employees.first_name%TYPE, 
    p_last_name  employees.last_name%TYPE, 
    p_email employees.email%TYPE, 
    p_phone_number employees.phone_number%TYPE, 
    p_department_id employees.department_id%TYPE, 
    p_job_id employees.job_id%TYPE, 
    p_salary employees.salary%TYPE, 
    p_commission_pct employees.commission_pct%TYPE, 
    p_manager_id employees.manager_id%TYPE, 
    p_created_by employees.created_by%TYPE, 
    p_created_date employees.created_date%TYPE, 
    p_modified_by employees.modified_by%TYPE, 
    p_modified_date employees.modified_date%TYPE);
Appendix B: Code Samples

```sql
p_hire_date employees.hire_date%TYPE,
p_job_id employees.job_id%TYPE,
p_salary employees.salary%TYPE,
p_commission_pct employees.commission_pct%TYPE,
p_manager_id employees.manager_id%TYPE,
p_department_id employees.department_id%TYPE,
p_created_by employees.created_by%TYPE,
p_created_date employees.created_date%TYPE,
p_modified_by employees.modified_by%TYPE,
p_modified_date employees.modified_date%TYPE);
--
PROCEDURE upd(
  p_employee_id employees.employee_id%TYPE,
p_first_name employees.first_name%TYPE,
p_last_name employees.last_name%TYPE,
p_email employees.email%TYPE,
p_phone_number employees.phone_number%TYPE,
p_hire_date employees.hire_date%TYPE,
p_job_id employees.job_id%TYPE,
p_salary employees.salary%TYPE,
p_commission_pct employees.commission_pct%TYPE,
p_manager_id employees.manager_id%TYPE,
p_department_id employees.department_id%TYPE,
p_created_by employees.created_by%TYPE,
p_created_date employees.created_date%TYPE,
p_modified_by employees.modified_by%TYPE,
p_modified_date employees.modified_date%TYPE);
--
PROCEDURE del (  
  p_employee_id employees.employee_id%TYPE);
--
PROCEDURE lck (  
  p_employee_id employees.employee_id%TYPE);
--
END employees_pkg;
CREATE OR REPLACE PACKAGE BODY employees_pkg  
IS  
--
FUNCTION check_insert_rules(  
  p_employee_id employees.employee_id%TYPE,
p_first_name employees.first_name%TYPE,
p_last_name employees.last_name%TYPE,
p_email employees.email%TYPE,
p_phone_number employees.phone_number%TYPE,
p_hire_date employees.hire_date%TYPE,
p_job_id employees.job_id%TYPE,
p_salary employees.salary%TYPE,
p_commission_pct employees.commission_pct%TYPE,
p_manager_id employees.manager_id%TYPE,
p_department_id employees.department_id%TYPE,
p_created_by employees.created_by%TYPE,
```
p_created_date employees.created_date%TYPE,
p_modified_by employees.modified_by%TYPE,
p_modified_date employees.modified_date%TYPE)
RETURN VARCHAR2
IS
  v_error_message VARCHAR2(10000);
BEGIN
  IF p_hire_date < jobs_pkg.job_start_date(p_department_id)
  THEN
    -- "Employee Hire Date must be on or after the job start date."
    v_error_message := message_pkg.message_text(500);
  END IF;
  --
  IF NOT util_pkg.check_list_value(
    'US_STATE', departments_pkg.address_state(p_department_id))
  THEN
    v_error_message := v_error_message || ' ' || message_pkg.message_text(501);
  END IF;
  --
  RETURN v_error_message;
END check_insert_rules;
--
--
PROCEDURE ins (p_employee_id employees.employee_id%TYPE,
p_first_name employees.first_name%TYPE,
p_last_name employees.last_name%TYPE,
p_email employees.email%TYPE,
p_phone_number employees.phone_number%TYPE,
p_hire_date employees.hire_date%TYPE,
p_job_id employees.job_id%TYPE,
p_salary employees.salary%TYPE,
p_commission_pct employees.commission_pct%TYPE,
p_manager_id employees.manager_id%TYPE,
p_department_id employees.department_id%TYPE,
p_created_by employees.created_by%TYPE,
p_created_date employees.created_date%TYPE,
p_modified_by employees.modified_by%TYPE,
p_modified_date employees.modified_date%TYPE)
IS
  v_error_message VARCHAR2(10000);
BEGIN
  g_allow_dml := TRUE;
  -- Or put this call in the table trigger
  v_error_message := check_insert_rules(
    p_employee_id,
    p_first_name,
    p_last_name,
    p_email,
    p_phone_number,
    p_hire_date,
p_job_id,
p_salary,
p_commission_pct,
p_manager_id,
p_department_id,
p_created_by,
p_created_date,
p_modified_by,
p_modified_date);

--
IF v_error_message IS NULL
THEN
    INSERT INTO employees(
        employee_id,
        first_name,
        last_name,
        email,
        phone_number,
        hire_date,
        job_id,
        salary,
        commission_pct,
        manager_id,
        department_id,
        created_by,
        created_date,
        modified_by,
        modified_date)
VALUES (
    p_employee_id,
    p_first_name,
    p_last_name,
    p_email,
    p_phone_number,
    p_hire_date,
    p_job_id,
    p_salary,
    p_commission_pct,
    p_manager_id,
    p_department_id,
    p_created_by,
    p_created_date,
    p_modified_by,
    p_modified_date);

ELSE
    RAISE_APPLICATION_ERROR(-20298, v_error_message);
END IF;

--
q_allow_dml := FALSE;
EXCEPTION
WHEN OTHERS
THEN
Appendix B: Code Samples

```sql
    g_allow_dml := FALSE;
    --
    RAISE_APPLICATION_ERROR(-20299, 'Error inserting: '||SQLERRM);
END ins;
--
--
PROCEDURE upd(
    p_employee_id  employees.employee_id%TYPE,
    p_first_name  employees.first_name%TYPE,
    p_last_name  employees.last_name%TYPE,
    p_email  employees.email%TYPE,
    p_phone_number  employees.phone_number%TYPE,
    p_hire_date  employees.hire_date%TYPE,
    p_job_id  employees.job_id%TYPE,
    p_salary  employees.salary%TYPE,
    p_commission_pct  employees.commission_pct%TYPE,
    p_manager_id  employees.manager_id%TYPE,
    p_department_id  employees.department_id%TYPE,
    p_created_by  employees.created_by%TYPE,
    p_created_date  employees.created_date%TYPE,
    p_modified_by  employees.modified_by%TYPE,
    p_modified_date  employees.modified_date%TYPE)
IS
BEGIN
    g_allow_dml := TRUE;
    --
    -- TODO: Add call to check_update_rules() when it is created. See ins().
    --
    UPDATE employees
    SET
    first_name = p_first_name,
    last_name = p_last_name,
    email = p_email,
    phone_number = p_phone_number,
    hire_date = p_hire_date,
    job_id = p_job_id,
    salary = p_salary,
    commission_pct = p_commission_pct,
    manager_id = p_manager_id,
    department_id = p_department_id,
    created_by = p_created_by,
    created_date = p_created_date,
    modified_by = p_modified_by,
    modified_date = p_modified_date
    WHERE employee_id = p_employee_id;
    --
    g_allow_dml := FALSE;
EXCEPTION
    WHEN OTHERS
    THEN
        g_allow_dml := FALSE;
    --
```
RAISE_APPLICATION_ERROR(-20299, 'Error updating: '||SQLERRM);
END upd;
--
--
PROCEDURE del (p_employee_id employees.employee_id%TYPE)
IS
BEGIN
  g_allow_dml := TRUE;
  --
  --
  -- TODO: Add call to check_delete_rules() when it is created. See ins().
  --
  DELETE FROM employees
  WHERE employee_id = p_employee_id;
  --
  g_allow_dml := FALSE;
EXCEPTION
  WHEN OTHERS
  THEN
    g_allow_dml := FALSE;
    --
    RAISE_APPLICATION_ERROR(-20299, 'Error deleting: '||SQLERRM);
END del;
--
--
PROCEDURE lck (p_employee_id employees.employee_id%TYPE)
IS
  v_dummy PLS_INTEGER;
BEGIN
  g_allow_dml := TRUE;
  --
  SELECT 1
  INTO   v_dummy
  FROM   employees
  WHERE  employee_id = p_employee_id
  FOR UPDATE;
  --
  g_allow_dml := FALSE;
EXCEPTION
  WHEN OTHERS
  THEN
    g_allow_dml := FALSE;
    --
    RAISE_APPLICATION_ERROR(-20299, 'Error locking: '||SQLERRM);
END lck;
--
--
END employees_pkg;
EMPLOYEES_TRBR Trigger

CREATE OR REPLACE TRIGGER employees_trbr
BEFORE INSERT OR UPDATE OR DELETE
ON employees
FOR EACH ROW
DECLARE
    v_error    VARCHAR2(2000);
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;
    --
    -- Note: The following is an alternative to calling the
    --       business rules checks from the table API
    IF INSERTING
    THEN
        v_error := check_insert_rules(
            :NEW.employee_id,
            -- other column values
        ELSIF UPDATING
    THEN
        v_error := check_update_rules(
            :NEW.employee_id,
            -- other column values
        ELSE    -- DELETING
            v_error := check_delete_rules(
                :NEW.employee_id,
                -- other column values
        END IF;
    --
    IF v_error IS NOT NULL
    THEN
        -- fails the trigger and the statement
        RAISE_APPLICATION_ERROR(-20199, v_error);
    END IF;
    --
END employees_trbr;

Table API Code Generation Snippets

-- Column list
SELECT LOWER(column_name)||',' col
FROM   user_tab_columns
WHERE  table_name = 'EMPLOYEES'
ORDER BY column_id;

-- VALUES list
SELECT 'p_'||LOWER(column_name)||',' col
FROM user_tab_columns
WHERE table_name = 'EMPLOYEES'
ORDER BY column_id;

-- Parameter list
SELECT 'p_'||LOWER(column_name)||'  employees.'||LOWER(column_name)||'%TYPE,' col
FROM user_tab_columns
WHERE table_name = 'EMPLOYEES'
ORDER BY column_id;

-- Update columns
SELECT LOWER(column_name)||' = '||
    'p_'||LOWER(column_name)||',' col
FROM user_tab_columns
WHERE table_name = 'EMPLOYEES'
ORDER BY column_id;

-- INSTEAD OF trigger parameters
SELECT ':NEW.'||LOWER(column_name)||',' col
FROM user_tab_columns
WHERE table_name = 'EMPLOYEES'
ORDER BY column_id;