No Room For Error: Exception Handling in PL/SQL

Make No Mistakes When it Comes to Error Handling

Steven Feuerstein

Steven.feuerstein@quest.com

QUEST SOFTWARE

Exception Handling in PL/SQL

Execute Application Code

Handle Exceptions

- The PL/SQL language provides a powerful, flexible "event-driven" architecture to handle errors which arise in your programs
  - No matter how an error occurs, it will be trapped by the corresponding handler
- Is this good? Yes and no
  - You have many choices in building exception handlers
  - There is no one right answer for all situations, all applications
  - This usually leads to an inconsistent, incomplete solution

You Need Strategy & Architecture

- To build a robust PL/SQL application, you need to decide on your strategy for exception handling, and then build a code-based architecture for implementing that strategy
- In this e-seminar, we will:
  - Explore the features of PL/SQL error handling to make sure we have common base of knowledge
  - Examine the common problems developers encounter with exception handling
  - Construct a prototype for an infrastructure component that enforces a standard, best practice-based approach to trapping, handling and reporting errors

The PLVexc package (part of the Knowledge Xpert for PL/SQL) is a more complete implementation.

Flow of Exception Handling

```
PROCEDURE financial_review
IS
BEGIN
   calc_profits (1996);

   calc_expenses (1996);

   DECLARE
      v_str VARCHAR2(1);
   BEGIN
      v_str := 'abc';
   END;
EXCEPTION
   WHEN OTHERS
   THEN
      log_error;
END;
```

```
PROCEDURE calc_profits
IS BEGIN
   numeric_var := 'ABC';

EXCEPTION
   WHEN VALUE_ERROR THEN
      log_error;
      RAISE;
END;
```

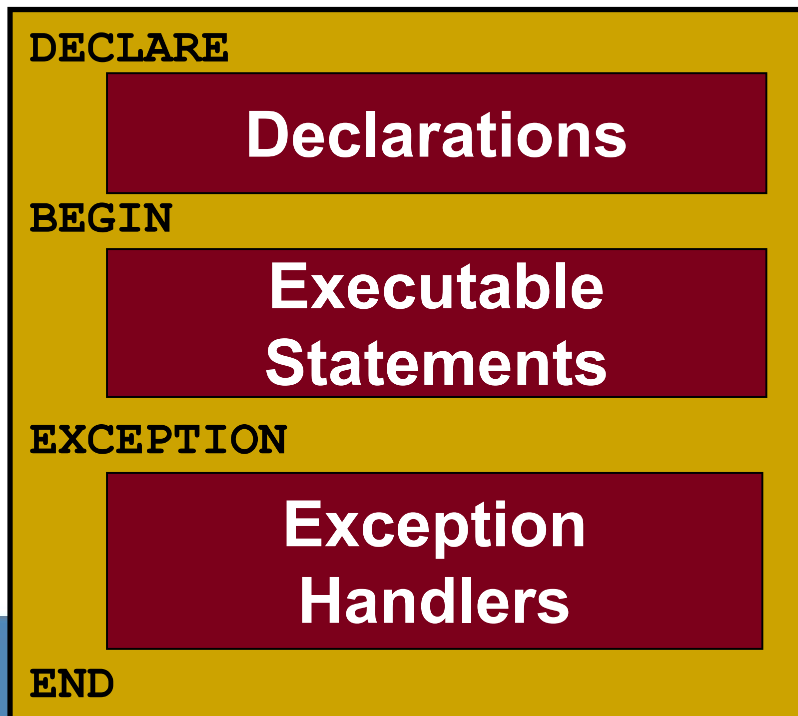
```
PROCEDURE calc_expenses
IS BEGIN
   ...
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      NULL;
END;
```

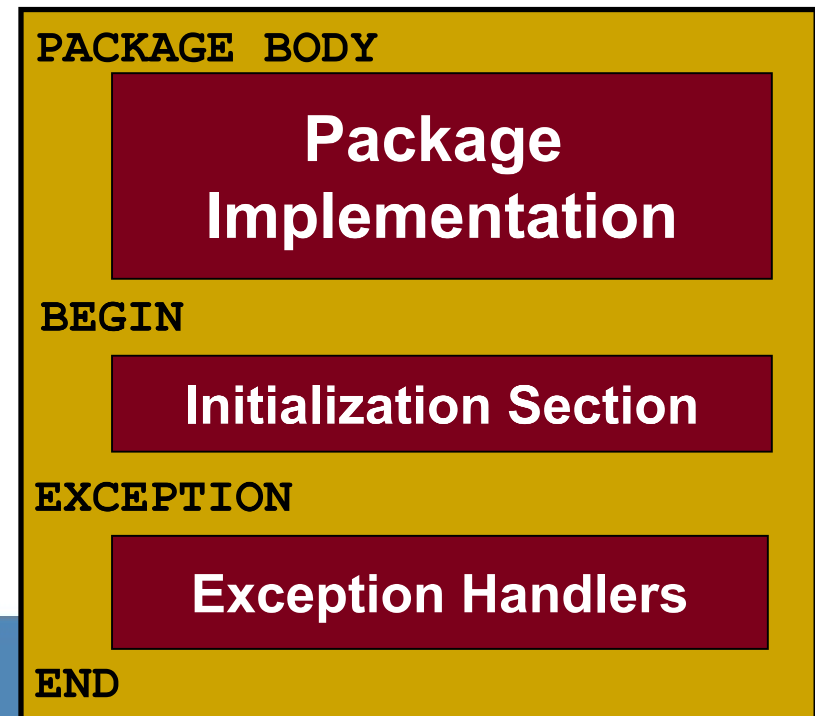
Scope and Propagation Reminders

- You can never go home
  - Once an exception is raised in a block, that block's executable section closes. But you get to decide what constitutes a block
- Once an exception is handled, there is no longer an exception (unless another exception is raised)
  - The next line in the enclosing block (or the first statement following the return point) will then execute
- If an exception propagates out of the outermost block, then that exception goes unhandled
  - In most environments, the host application stops
  - In SQL*Plus and most other PL/SQL environments, an automatic ROLLBACK occurs

What the Exception Section Covers

- The exception section only handles exceptions raised in the executable section of a block
  - For a package, this means that the exception section only handles errors raised in the initialization section





Continuing Past an Exception

- Emulate such behavior by enclosing code within its own block

All or Nothing

```
PROCEDURE cleanup_details (id_in IN NUMBER) IS
BEGIN
   DELETE FROM details1 WHERE pky = id_in;
   DELETE FROM details2 WHERE pky = id_in;
END;
```

The "I Don't Care" Exception Handler

```
PROCEDURE cleanup_details (id_in IN NUMBER) IS
BEGIN
   BEGIN
      DELETE FROM details1 WHERE pky = id_in;
   EXCEPTION WHEN OTHERS THEN NULL;
   END;
   BEGIN
      DELETE FROM details2 WHERE pky = id_in;
   EXCEPTION WHEN OTHERS THEN NULL;
   END;
END;
```

Exceptions and DML

- DML statements are not rolled back by an exception unless it goes unhandled
  - This gives you more control over your transaction, but it also can lead to complications. What if you are logging errors to a database table? That log is then a part of your transaction
  - You may generally want to avoid "unqualified" ROLLBACKs and instead always specify a savepoint

Or use autonomous transactions in Oracle8i!

```
WHEN NO_DATA_FOUND THEN
  ROLLBACK TO last_log_entry;
  INSERT INTO log VALUES (...);
  SAVEPOINT last_log_entry;
END;
```

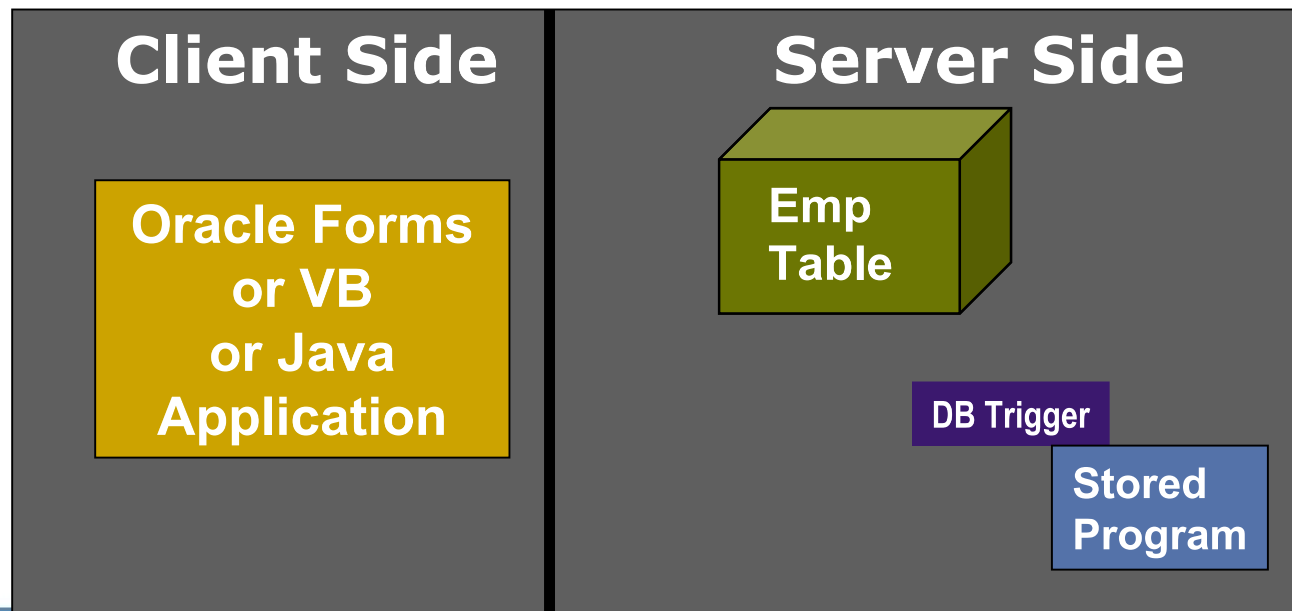
Logging with Auton Transactions

- With Oracle8i, you can now define a PL/SQL block to execute as an "autonomous transaction"
  - Any changes made within that block can be saved or reversed without affecting the outer or main transaction

```
CREATE OR REPLACE PACKAGE BODY log IS
   PROCEDURE putline (
      code_in IN INTEGER, text_in IN VARCHAR2)
   IS
      PRAGMA AUTONOMOUS_TRANSACTION;
   BEGIN
     INSERT INTO logtab
       VALUES (code_in, text_in,
          SYSDATE, USER, SYSDATE, USER);
      COMMIT;
   EXCEPTION WHEN OTHERS THEN ROLLBACK;
   END;
END;
```

Application-Specific Exceptions

- Raising and handling an exception specific to the application requires special treatment
  - This is particularly true in a client-server environment with Oracle Developer



Communicating an Application Error

- Use the RAISE_APPLICATION_ERROR built-in procedure to communicate an error number and message across the client-server divide
  - Oracle sets aside the error codes between -20000 and -20999 for your application to use. RAISE_APPLICATION_ERROR can only be used those error numbers

```
RAISE_APPLICATION_ERROR
   (num binary_integer,
    msg varchar2,
    keeperrorstack boolean default FALSE);
```

The following code from a database triggers shows a typical usage of RAISE_APPLICATION_ERROR

```
IF :NEW.birthdate > ADD_MONTHS (SYSDATE, -1 * 18 * 12)
THEN
   RAISE_APPLICATION_ERROR
      (-20347, 'Employee must be 18.');
END IF;
```

Poll

- Which of the following applies to you:
  - A. We never use the -20NNN error numbers.
  - B. We use them, but have no controls in place over which numbers are used, and how.
  - C. We use the -20NNN error numbers and have set up a utility or process to manage selection of and use of these numbers.

Handling App. Specific Exceptions

Handle in OTHERS with check against SQLCODE...

```
BEGIN
   INSERT INTO emp (empno, deptno, birthdate)
      VALUES (100, 200, SYSDATE);
EXCEPTION
   WHEN OTHERS THEN
      IF SQLCODE = -20347 ...
END;
```

Server-side Database

Or handle with named exception, declared on client side ...

```
DECLARE
   emp_too_young EXCEPTION;
   PRAGMA EXCEPTION_INIT (emp_too_young, -20347);
BEGIN
   INSERT INTO emp (empno, deptno, birthdate)
      VALUES (100, 200, SYSDATE);
EXCEPTION
   WHEN emp_too_young THEN ...
END;
```

Server-side Database

The Ideal, But Unavailable Solution

Declare the exception in one place (server) and reference it (the error number or name) throughout your application.

```
CREATE OR REPLACE PACKAGE emp_rules IS
   emp_too_young EXCEPTION;
END;
```

Server side pkg defines exception.

Database trigger raises exception.

```
IF birthdate > ADD_MONTHS (SYSDATE, -216) THEN
   RAISE emp_rules.emp_too_young;
END IF;
```

```
BEGIN
   INSERT INTO emp VALUES (100, 200, SYSDATE);
EXCEPTION
   WHEN emp_rules.emp_too_young THEN ...
END;
```

Client side block handles exception.

But this won't work with Oracle Developer! If it's got a dot and is defined on the server, it can only be a function or procedure, not an exception or constant or variable...

Desperately Seeking Clarity

- Hopefully everyone now feels more confident in their understanding of how exception handling in PL/SQL works
- Let's move on to an examination of the challenges you face as you build an application and seek to build into it consistent error handling
- After that, we take a look at how you might build a generic, reusable infrastructure component to handle the complexities of exception handling

All-Too-Common Handler Code

```
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      v_msg := 'No company for id ' || TO_CHAR (v_id);
      v_err := SQLCODE;
      v_prog := 'fixdebt';
      INSERT INTO errlog VALUES
         (v_err, v_msg, v_prog, SYSDATE, USER);

   WHEN OTHERS THEN
      v_err := SQLCODE;
      v_msg := SQLERRM;
      v_prog := 'fixdebt';
      INSERT INTO errlog VALUES
         (v_err, v_msg, v_prog, SYSDATE, USER);
      RAISE;
```

- If every developer writes exception handler code on their own, you end up with an unmanageable situation
  - Different logging mechanisms, no standards for error message text, inconsistent handling of the same errors, etc.

Some Dos and Don'ts

- Make decisions about exception handling *before* starting your application development. Here are my recommendations:

DISCOURAGE individual developer usage of RAISE_APPLICATION_ERROR, PRAGMA EXCEPTION_INIT, explicit (hard-coded) -20,NNN error numbers, hard-coded error messages, exposed exception handling logic.

ENCOURAGE use of standardized components, including programs to raise application-specific exception, handle (log, re-raise, etc.) errors, and rely on pre-defined error numbers and messages.

Checking Standards Compliance

- Whenever possible, try to move beyond document-based standards
  - Instead, build code to both help people deploy standards and create tools to help verify that they have complied with standards

```
CREATE OR REPLACE PROCEDURE progwith (str IN VARCHAR2)
IS
   CURSOR objwith_cur (str IN VARCHAR2)
   IS
      SELECT DISTINCT name
        FROM USER_SOURCE
       WHERE UPPER (text) LIKE '%' || UPPER (str) || '%';
BEGIN
   FOR prog_rec IN objwith_cur (str)
   LOOP
      DBMS_OUTPUT.PUT_LINE (prog_rec.name);
   END LOOP;
END;
```

Pre-Defined -20,NNN Errors

1. Assign Error Number
2. Declare Named Exception
3. Associate Number w/Name

```
PACKAGE errnums
IS
   en_general_error CONSTANT NUMBER := -20000;
   exc_general_error EXCEPTION;
   PRAGMA EXCEPTION_INIT
      (exc_general_error, -20000);

   en_must_be_18 CONSTANT NUMBER := -20001;
   exc_must_be_18 EXCEPTION;
   PRAGMA EXCEPTION_INIT
      (exc_must_be_18, -20001);

   en_sal_too_low CONSTANT NUMBER := -20002;
   exc_sal_too_low EXCEPTION;
   PRAGMA EXCEPTION_INIT
      (exc_sal_too_low , -20002);

   max_error_used CONSTANT NUMBER := -20002;

END errnums;
```

But don't write this code manually - generate it instead! Visit www.quest-pipelines.com and the PL/SQL Pipeline Archives. Search for msginfo.pkg.

Reusable Exception Handler Package

```
PACKAGE errpkg
IS
   PROCEDURE raise (err_in IN INTEGER);

   PROCEDURE recNstop (err_in IN INTEGER := SQLCODE,
      msg_in IN VARCHAR2 := NULL);

   PROCEDURE recNgo (err_in IN INTEGER := SQLCODE,
      msg_in IN VARCHAR2 := NULL);

   FUNCTION errtext (err_in IN INTEGER := SQLCODE)
     RETURN VARCHAR2;

END errpkg;
```

Generic Raise

Record and Stop

Record and Continue

Message Text Consolidator

Implementing a Generic RAISE

- Hides as much as possible the decision of whether to do a normal
- RAISE or call RAISE_APPLICATION_ERROR
  - Also forces developers to rely on predefined message text

```
PROCEDURE raise (err_in IN INTEGER) IS
BEGIN
   IF err_in BETWEEN -20999 AND -20000
   THEN
      RAISE_APPLICATION_ERROR (err_in, errtext (err_in));
   ELSIF err_in IN (100, -1403)
   THEN
      RAISE NO_DATA_FOUND;
   ELSE
      EXECUTE IMMEDIATE
         'DECLARE myexc EXCEPTION; ' ||
         '   PRAGMA EXCEPTION_INIT (myexc, ' ||
               TO_CHAR (err_in) || ');' ||
         'BEGIN  RAISE myexc; END;');
   END IF;
END;
```

Enforce use of standard message

Re-raise *almost* any exception using Dynamic PL/SQL!

No More Hard Coding of Error Info

- With the generic raise procedure and the pre-defined error numbers, you can write high-level, readable, maintainable code
  - The individual developers make fewer decisions, write less code, and rely on pre-built standard elements
- Let's revisit that trigger logic using the infrastructure elements...

```
PROCEDURE validate_emp (birthdate_in IN DATE) IS
BEGIN
   IF ADD_MONTHS (SYSDATE, 18 * 12 * -1) < birthdate_in
   THEN
      errpkg.raise (errnums.en_must_be_18);
   END IF;
END;
```

No more hard-coded strings or numbers.

Deploying Standard Handlers

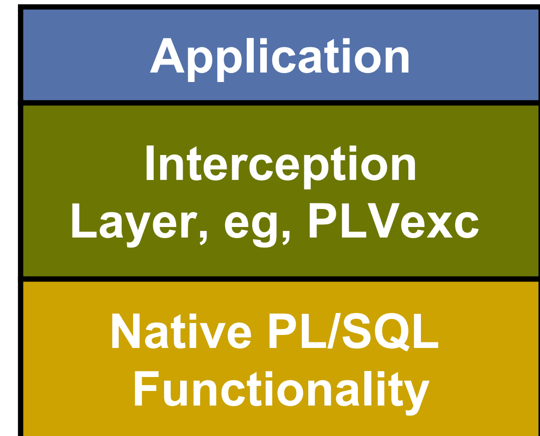
- The rule: developers should *only* call a pre-defined handler inside an exception section
  - Make it impossible for developers to *not* build in a consistent, high-quality way
  - They don't have to make decisions about the form of the log and how the process should be stopped

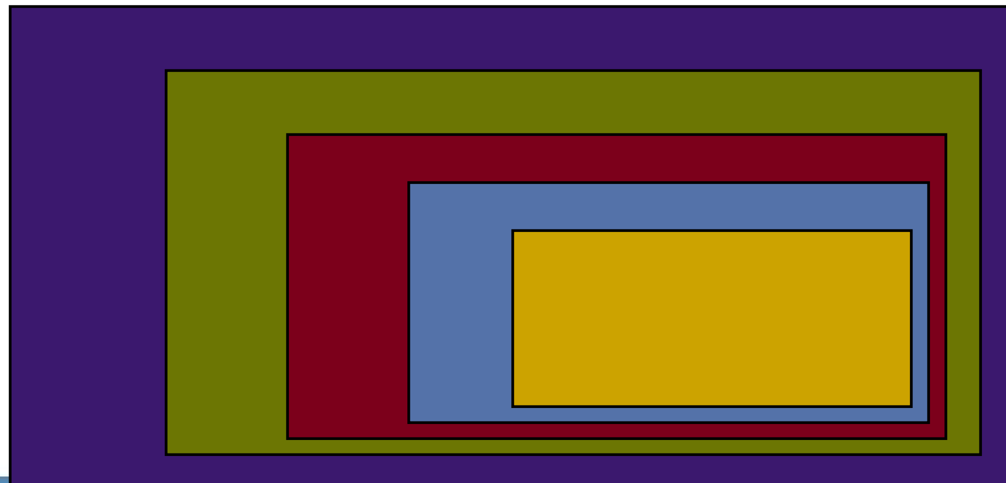
```
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN
      errpkg.recNgo (
         SQLCODE,
         ' No company for id ' || TO_CHAR (v_id));

   WHEN OTHERS
   THEN
      errpkg.recNstop;
END;
```

Now the developer simply *describes* the desired action.

Added Value of a Handler Package

- Once all developers are using the handler package, you can add value in a number of directions:
  - Store templates and perform runtime substitutions
  - Offer the ability to "bail out" of a program





Make No Mistakes...With an Exception Handling Architecture and Strategy

- Make sure you understand how it all works
  - Exception handling is tricky stuff
- Set standards before you start coding
  - It's not the kind of thing you can easily add in later
- Use standard infrastructure components
  - Everyone and all programs need to handle errors the same way

QUESTIONS & ANSWERS