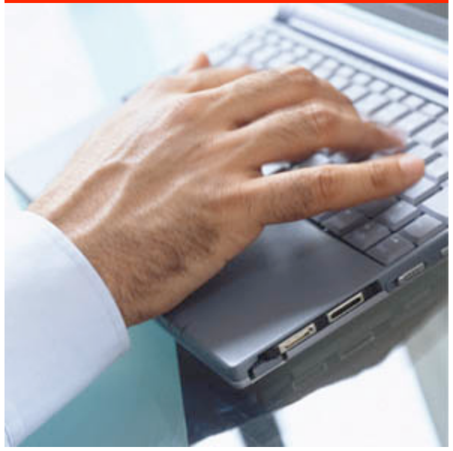
Contract-oriented PL/SQL programming

John Beresniewicz
Technical staff, Server Technologies
Oracle USA

Agenda



- Theory
  - Design by Contract
- Technique
  - Standardized assertions
  - Rigorous precondition checking
  - Function-oriented programming
- Practice
  - Useful contract patterns
  - Example: prime number package

Theory
Design by contract

References

- *Object-oriented Software Construction, 2nd Edition*
  - Bertrand Meyer. Prentice Hall PTR 1997.
- *The Pragmatic Programmer*
  - Andrew Hunt, Dave Thomas. Addison Wesley Longman, Inc 2000.
- *Design by Contract, by Example*
  - Richard Mitchell, Jim McKim. Addison-Wesley 2002.
- *Design Patterns and Contracts*
  - Jean-Marc Jezequel, Michel Train, Christine Mingins. Addison Wesley Longman 2000.

Bertrand Meyer
Object Success

"Design by Contract is a powerful metaphor that... makes it possible to design software systems of much higher reliability than ever before; the key is understanding that reliability problems (more commonly known as bugs) largely occur at module boundaries, and most often result from inconsistencies in both sides' expectations."

Contracts

- Legal contracts govern interactions between parties
- Each party has obligations and expects benefits under the contract
- Each party typically benefits from the other’s obligations, and vice versa

Software contracts

- Software modules have interactions as calling and called program
  - Client-supplier relationship between modules
  - Interaction boundary also known as the API
- We can describe these interactions as conforming to contracts
- Formalizing and enforcing software contracts promotes reliability and maintainability

Contract elements

- PRECONDITIONS
  - What will be true when module is entered?
  - Caller obligation and supplier benefit
- POSTCONDITIONS
  - What will be true when module completes?
  - Supplier obligation and caller benefit
- INVARIANTS
  - Any state that will not be changed as a result of module execution

Contracting benefits

- Preconditions allow suppliers to trust data in
- Postconditions allow callers to trust data out
- Rigid postcondition obligations promote stricter module scoping
  - Better functional decomposition
- Bugs are detected early and isolated easily
- Trusted data + correct algorithms = bug-free code

Software defects (bugs)

- Contract violations are BUGS…always
- Precondition violations are caller bugs
- Postcondition violations are supplier bugs

Bertrand Meyer

Object-oriented Software Construction

"Good contracts are those which exactly specify the rights and obligations of each party...In software design, where correctness and robustness are so important, we need to spell out the terms of the contracts as a prerequisite to enforcing them. Assertions provide the means to state precisely what is expected from and guaranteed to each side in these arrangements."

Assertions: enforcing contracts

- Test a contract condition and complain if not met
- Contract condition is expressed as a BOOLEAN
  - Contract violated when expression is FALSE
- Complaint is to raise a well-known EXCEPTION
- Implement in PL/SQL as a simple procedure

Benefits of assertions

- Help to write correct software
- Documentation aid
- Support for testing, debugging and QA
- Support for software fault tolerance

Source: Object-oriented Software Construction

PL/SQL assert

```
PROCEDURE assert (condition_in IN BOOLEAN)
IS
BEGIN
   if NOT NVL(condition_in,FALSE)
   then
      raise ASSERTFAIL;
   end if;
END assert;
```

- Exits silently or raises ASSERTFAIL
- Null input throws ASSERTFAIL

Asserting contract elements

```
 FUNCTION distance
   (x_in in number, y_in in number) RETURN number
IS
  l_return number := null;

 BEGIN
  assert_pre(x_in not null,'x_in not null');
  assert_pre(y_in not null,'y_in not null');
  assert_pre(y_in > 0,'y_in >0');

  l_return := ABS(x_in - y_in);

  assert_post(l_return >=0,'return>=0');

  RETURN l_return;
END distance;
```

Technique
Minimum-optimum

Technique

- Standardize
  - Standard private local assert procedure in all packages
  - Standard common exception ASSERTFAIL in all packages
- Enforce preconditions aggressively
  - Crash on ASSERTFAIL
- Modularize
  - Each module implements a distinct function
  - Package implements a coherent set of modules

Standard declarations

```
-- standard package public assertion declarations
ASSERTFAIL     EXCEPTION;
PRAGMA EXCEPTION_INIT(ASSERTFAIL, -20999);

 -- standard package private declarations
ASSERTFAIL_C   CONSTANT INTEGER := -20999;
PKGNAME_C      CONSTANT VARCHAR2(20)  := 'Primes'
```

- All packages declare private version of common exception ASSERTFAIL
  - Externally identical but internally each can be distinguished

- Reserve an exception number for global convention

Original standard assert (ca 2001)

```
-- standard local packaged assertion procedure
PROCEDURE assert (bool_IN IN BOOLEAN
                 ,msg_IN   IN VARCHAR2 := null)
IS
  BEGIN
    IF NOT NVL(bool_IN,FALSE) -- assertfail on null
    THEN
      RAISE_APPLICATION_ERROR
        ( ASSERTFAIL_C, 'ASSERTFAIL:'||PKGNAME_C||
           ':'||SUBSTR(NVL(msg_IN,'nomsg'),1,200) );
   END IF;
END assert;
```

- This procedure identical and private to all packages

Proposed asserts (2007)

```
PROCEDURE assert_pre
    (condition_in IN BOOLEAN
    ,progname_in IN progname_t
    ,msg_in      IN varchar2)
IS
BEGIN
  assert(condition_in
        ,progname_in||':PRE:'||msg_in);
END assert_pre;
```

- We similarly define assert_post

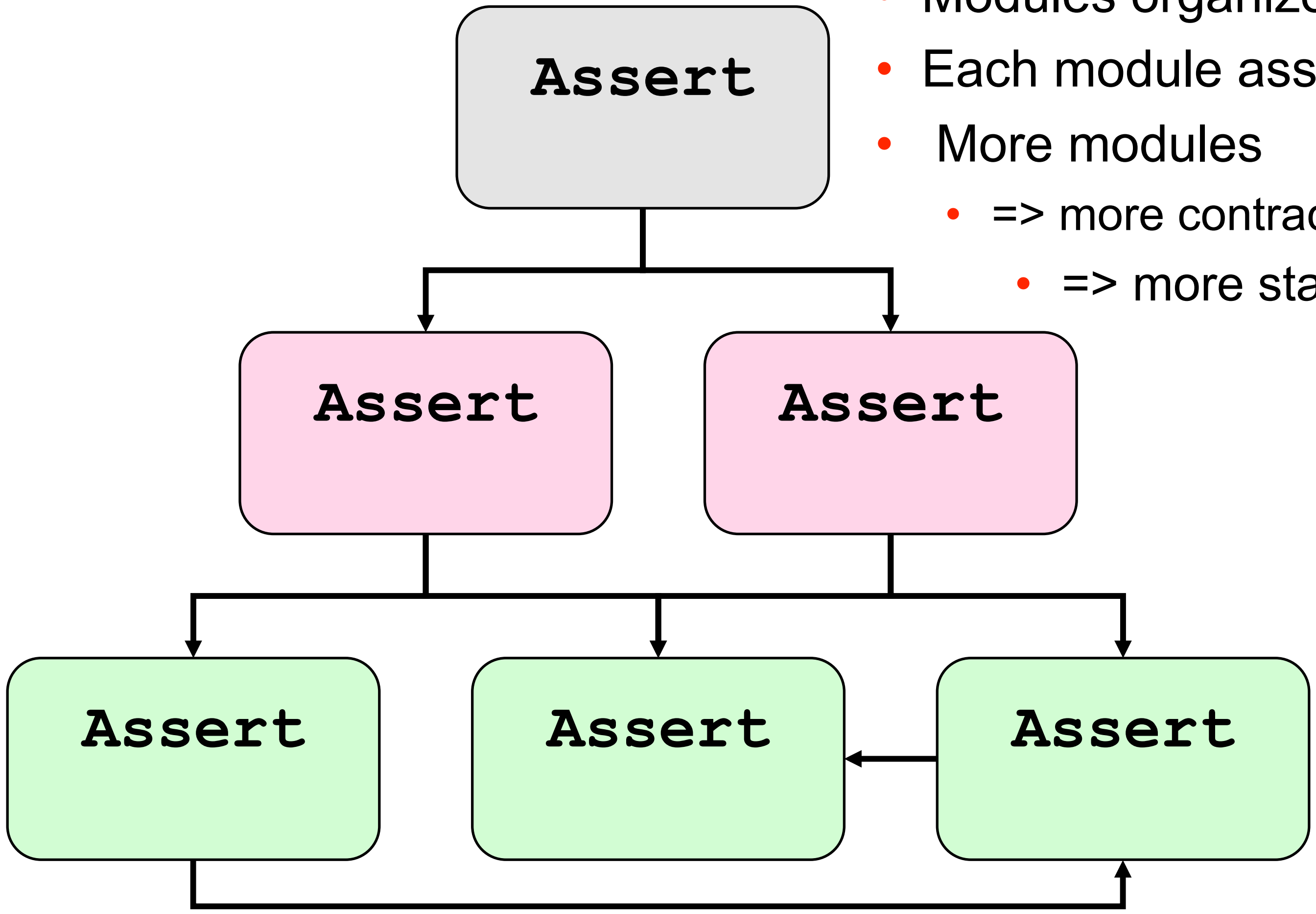
Aggressively assert preconditions

- A contract enforced on one side only is still a contract enforced
  - Think of it as tests built into code and checked at runtime
- Decomposition increases stability through multiplication of contracts
  - Thus an incentive to fine-grain modularization
- Enables trusted software layer to grow up through the hierarchy

Modularize ruthlessly and coherently

- Hierarchical layers of packages and modules
- Top layer: outward facing
    - Accept user input
    - Present high-level client interfaces and errors
- Middle layers: complex application logic
- Bottom layer: trusted base components
    - Ruthless precondition testing

Assertion-hardened code



- Modules organized hierarchically
- Each module asserts preconditions
- More modules
  - => more contracts
    - => more stable code

Crash on ASSERTFAIL

- It is a bug and must become known
- Catch ASSERTFAIL only for good reason:
  - Testing
  - Refactoring
    - Making a contract less demanding
    - Externalizing a local check function
- Dead Programs Tell No Lies
  - Hunt and Thomas, The Pragmatic Programmer

Testing

- Testing takes client (calling program) perspective
  - Whereas coding takes called program perspective
- Regression test postconditions
  - Issue: only externally facing modules can be externally tested
- Unit test contract preconditions against many inputs

```
assert(0=computefunction(0.2,100),’test 1001’);
```

Performance considerations

- Estimates for contracting overhead (from Meyer):
  - 50% for precondition checking
  - 100-200% for postcondition and invariant checking
  - Minimum-optimum principle favors precondition enforcement
- Oracle 10g conditional compilation enables assertions to be compiled into or out of pl/sql programs
- CAR Hoare quote

C.A.R. Hoare

“The current practice of compiling subscript range checks into the machine code while a program is being tested, then suppressing the checks during production runs, is like a sailor who wears his life preserver while training on land but leaves it behind when he sails!”

http://whats.all.this.brouhaha.com/?page_id=479

10g Conditional assertions

```
PROCEDURE trusted (p1_in in number)
IS
BEGIN
  $IF $$ASSERTPRE
  $THEN
        assert(p1_in > 0);
  $END
/* do trusted logic */
END trusted;
```

```
SQL> alter session set
              PLSQL_CCFLAGS='ASSERTPRE:TRUE;
SQL> alter procedure trusted compile;
```

- After recompiling assert will be included
- Recommend restricted rather than global application

BOOLEAN functions in contracts

```
FUNCTION ready_to_process(ID_in) return BOOLEAN;

PROCEDURE process_id(ID_in) IS
BEGIN
  assert(ready_to_process(ID_in)); --pre

  /* do the processing */
END;
```

- This can be very powerful, and potentially very dangerous
- When is it OK to assert functions as preconditions?

Bertrand Meyer
Object-oriented Software Construction

"If you exert the proper care by sticking to functions that are simple and self-evidently correct, the use of function routines in assertions can provide you with a powerful means of abstraction."

Check functions

```
FUNCTION check_all_ok return BOOLEAN is
BEGIN
   assert(condition1,…);
   assert(condition2,…);
   …
   assert(conditionN,…);
   return TRUE;
END check_all_ok;
```

- Compound set of assertions
- Self-evidently correct - TRUE or ASSERTFAIL
- Useful for compressing shared sets of preconditions

```
assert(check_all_ok,module_name||’:check_all_ok’);
```

Practice
Using contracts in programs

Useful contract patterns

- NOT NULL IN
  - Near-universal precondition
- RETURN NOT NULL
  - Very attractive postcondition
- RETURN BOOLEAN NOT NULL
  - Direct function references in conditional code
- RETURN BOOLEAN TRUE or ASSERTFAIL
  - Check functions

PL/QL Best Practices, 2nd Edition

```
FUNCTION excuse_in_use (excuse_in IN excuse_excuse_t)
RETURN BOOLEAN

IS

   c_progname CONSTANT progname_t:='EXCUSE_IN_USE';
  l_return BOOLEAN;

BEGIN

   -- check caller obligations
  assert_pre(condition_in => excuse_in is not null
            ,progname_in  => c_progname
            ,msg_in       => 'excuse_in not null');


   -- compute return value
  l_return := g_excuses_used.EXISTS(excuse_in);


   -- check return obligations
  assert_post(condition_in => l_return is not null
             ,progname_in  => c_progname
             ,msg_in       => 'l_return not null');

   RETURN l_return;
END excuse_in_use;
```

Bugs find you…

```
SQL> l
  1  begin
  2     if excuse_tracker.excuse_in_use('lame excuse')
  3     then
  4        dbms_output.put_line('lame excuse in use');
  5     elsif
  6        excuse_tracker.excuse_in_use(NULL)
  7     then
  8        dbms_output.put_line('NULL is no excuse!');
  9     end if;
 10* end;

SQL> /

begin
*
ERROR at line 1:
ORA-20999: ASSERTFAIL:EXCUSE_TRACKER:EXCUSE_IN_USE:PRE:excuse_in not null
ORA-06512: at "SYS.EXCUSE_TRACKER", line 62
ORA-06512: at "SYS.EXCUSE_TRACKER", line 85
ORA-06512: at "SYS.EXCUSE_TRACKER", line 30
ORA-06512: at line 6
```

Primes package: factoring by division

```
FUNCTION factor(num_IN INTEGER, divisor_IN INTEGER)
RETURN INTEGER

IS

  tmp_power  integer; tmp_num integer;

BEGIN

 assert(num_IN IS NOT NULL,'factor:num_IN not null');
 assert(divisor_IN IS NOT NULL,'factor:divisor_IN not null');
 assert(divisor_IN >1 AND num_IN >=divisor_IN
                                ,'factor:1<divisor<=num_IN');

  tmp_num := num_IN;  tmp_power := 0;  -- (1) initialize

  WHILE MOD(tmp_num,divisor_IN) = 0 -- still divides evenly
 LOOP

     tmp_power := tmp_power + 1;
    tmp_num := tmp_num / divisor_IN;

  END LOOP;

  RETURN tmp_power;

END factor;
```

Concluding remarks

- Design by Contract principles are powerful constructs for producing correct software
- Using standardized package assertions we can rigidly enforce contract preconditions in PL/SQL
- Native support for DbC in PL/SQL highly desirable
  - Standardized ASSERTFAIL exception
  - Precondition, postcondition, and invariant assertion syntax
  - Produce source line and offending condition automatically

10g goodies

- Conditional compilation: PLSQL_CCFLAGS
  - Compile assertions into or out of code
- Inquiry directives
  - $$PLSQL_UNIT
    - Replace constant literal PKGNAME_C
  - $$PLSQL_LINE
    - Identify line of code throwing ASSERTFAIL
- DBMS_PREPROCESSOR
  - Surface the line of code throwing ASSERTFAIL

dbc10g.sql – 10g assert experiments

```
-- new 10g signature
 PROCEDURE assert (line_no_IN    IN PLS_INTEGER
                  ,bool_expr_IN IN BOOLEAN
                  ,msg_IN       IN VARCHAR2 := null);
 -- simple test procedure
 PROCEDURE p1
 is
 begin
 assert($$PLSQL_LINE -- assert comment on first line
       ,25 < to_number('24')     -- assert boolean expr
       ,to_char(SYSDATE,'YYYY:MM:DD:HH24:MI:SS')); --msg
 end p1;
```

10g assert: debug info in ASSERTFAIL

```
BEGIN dbc10g.p1; END;


*
ERROR at line 1:
ORA-20999: ASSERTFAIL:PKGBDY:DBC10G.18:    assert( 18          -- assert
comment on first line
,25 < to_number('24')    -- assert boolean expr
:2008:10:10:22:06:53
ORA-06512: at "SYS.DBC10G", line 75
ORA-06512: at "SYS.DBC10G", line 18
ORA-06512: at line 1
```

ORACLE®