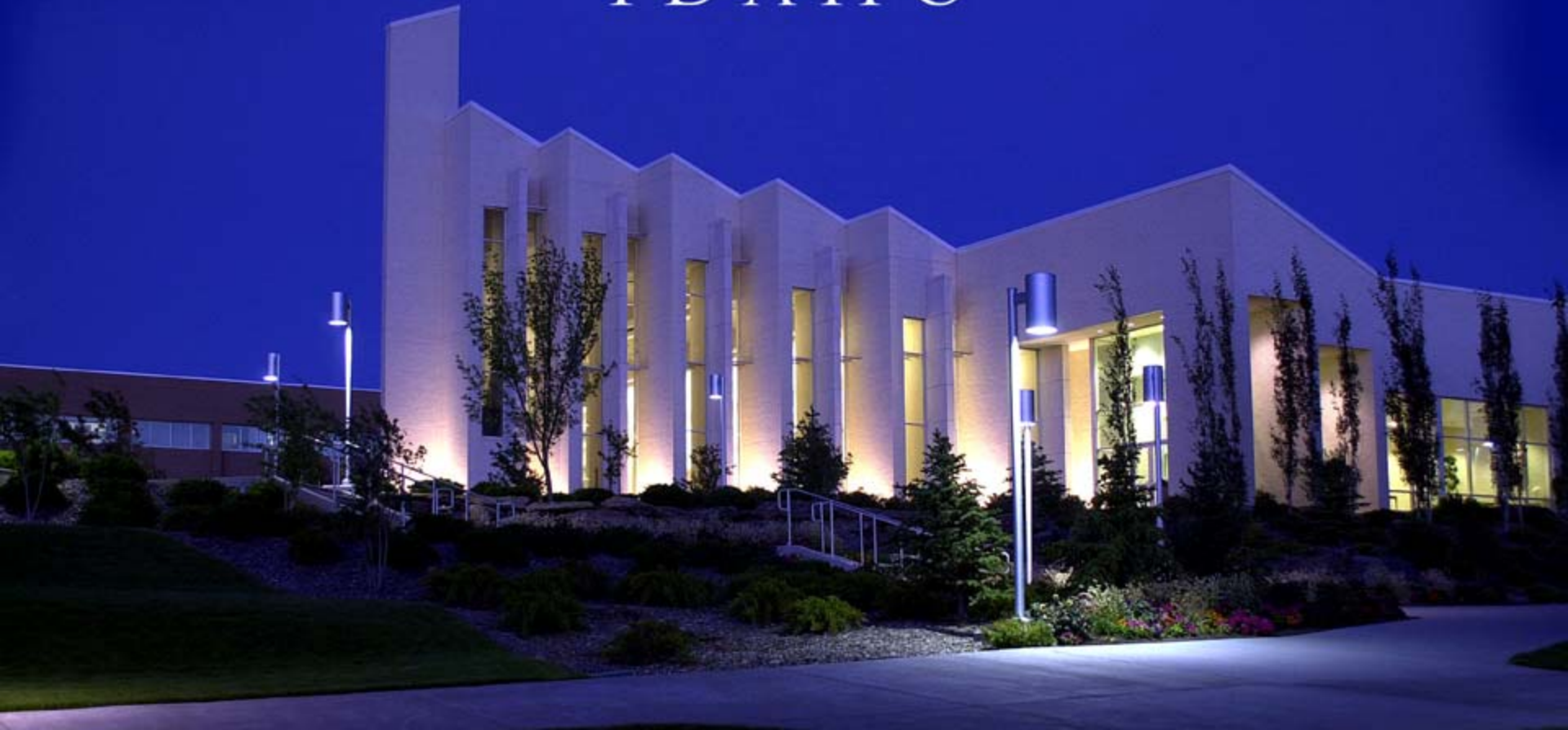


BRIGHAM YOUNG
UNIVERSITY

IDAHO



Oracle Database 11g PL/SQL New Features

Michael McLaughlin
Professor, BYU - Idaho



Oracle Database 11g PL/SQL New Features

- Automatic subprogram inlining
- A `CONTINUE` statement
- A cross-session result cache
- Mixed, named, and positional notation
- Sequences as expressions
- Native code generation
- Dynamic SQL enhancements
- Regular expression enhancements
- A `SIMPLE_INTEGER` datatype
- Compound Database Triggers
- A PL/SQL Hierarchical Profiler
- PL/Scope Compiler Tool
- Generalized Object Invocation

Automatic Subprogram Inlining

Oracle Database 11g

Automatic Subprogram Inlining

- **Subprogram Inlining:**
 - Replaces a call to an external subprogram with a copy of the subprogram.
- **Enable it with `PLSQL_OPTIMIZE_LEVEL`:**
 - Inlining disabled when parameter is set to 1.
 - Uses the `PRAGMA INLINE` when set to 2 (default).
 - Automatic inlining choices are made when set to 3.

Oracle Database 11g

Automatic Subprogram Inlining

- Checking parameter value:

```
SQL> SELECT name, value
  2   FROM   v$parameter
  3   WHERE  REGEXP_LIKE(name, 'PLSQL_OPT*');
```

- Setting parameter to *automatic* inlining:

```
SQL> ALTER SESSION
  2   SET plsql_optimize_level = 3;
```

Oracle Database 11g

Automatic Subprogram Inlining

```
CREATE OR REPLACE inline_demo
( a NUMBER, b NUMBER ) IS
    PRAGMA INLINE(add_integers, 'YES');
BEGIN
    FOR i IN 1..10000 LOOP
        dbms_output.put_line(add_integers(a,b));
    END LOOP;
END;
/
```

- Works with the following statements:
 - Assignments.
 - Calls.
 - Conditional if-then, if-then-else, et cetera.
 - The `CASE`, `CONTINUE WHEN`, `EXECUTE IMMEDIATE`, `EXIT WHEN`, `LOOP`, and `RETURN` statements.

The New CONTINUE Statement

Oracle Database 11g

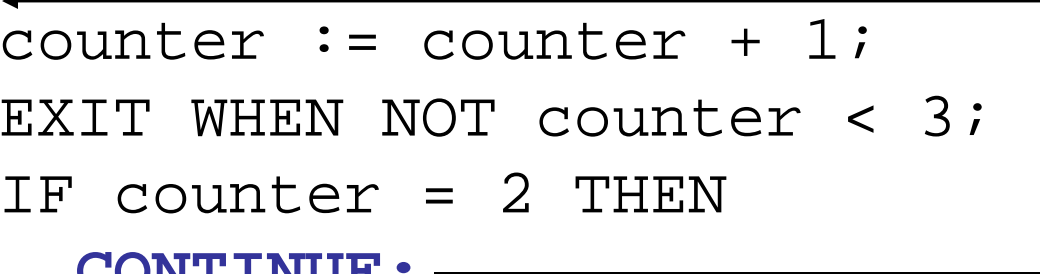
The CONTINUE Statement

- **The CONTINUE Statement**
 - Lets you immediately exit a loop iteration.
 - Risk of an infinite loop in a guard on exit loop.
- **The CONTINUE WHEN Statement**
 - Lets you conditionally exit a loop iteration.
 - Risk of an infinite loop in a guard on exit loop.

Oracle Database 11g

The CONTINUE Statement

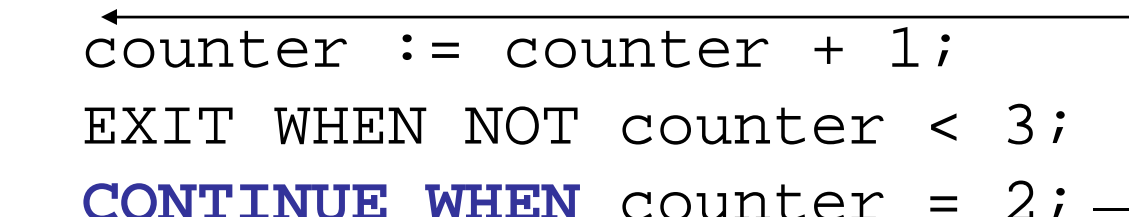
```
DECLARE
  counter NUMBER := 1;
BEGIN
  LOOP
    counter := counter + 1;
    EXIT WHEN NOT counter < 3;
    IF counter = 2 THEN
      CONTINUE;
    END IF;
  END LOOP;
END;
/
```

A diagram consisting of a rectangular box with a black border. Inside the box, the text 'counter := counter + 1;', 'EXIT WHEN NOT counter < 3;', and 'IF counter = 2 THEN' is written. Below this text is the word 'CONTINUE;' in bold blue font. A horizontal line extends from the end of 'CONTINUE;' to the right edge of the box. From the left edge of the box, a horizontal line extends to the left, ending in an arrowhead that points to the 'LOOP' keyword of the code block above.

Oracle Database 11g

The CONTINUE WHEN Statement

```
DECLARE
  counter NUMBER := 1;
BEGIN
  LOOP
    counter := counter + 1;
    EXIT WHEN NOT counter < 3;
    CONTINUE WHEN counter = 2;
  END LOOP;
END;
/
```



Cross Session Result Cache

Oracle Database 11g

Cross Session Function Result Cache

- **Cross-session function result cache:**
 - Lets you share frequently accessed functions in the SGA between sessions.
 - Requires formal parameters are passed by value as `IN` mode only arguments.
 - A `RELIES_ON` clause lets you flush results when the data changes in related tables.
 - A `RELIES_ON` clause takes a single object (table or view) name or a list of object names.

Oracle Database 11g

Cross Session Function Result Cache

```
CREATE OR REPLACE FUNCTION get_title
(partial_title VARCHAR2) RETURN STRINGS
RESULT_CACHE RELIES_ON(item) IS
  counter NUMBER := 1;
  return_value STRINGS := strings();
  CURSOR get_title (partial_title VARCHAR2) IS
    SELECT item_title FROM item
    WHERE
      REGEXP_LIKE(item_title, '*' || partial_title || '*', 'i');
BEGIN
  FOR i IN get_title(partial_title) LOOP
    return_value.EXTEND;
    return_value(counter) := i.item_title;
    counter := counter + 1;
  END LOOP;
  RETURN return_value;
END get_title;
/
```

- Call the result cache into the SGA:

```
SELECT column_value  
FROM TABLE(get_title('Potter'));
```

- The first call with the actual parameter places the result set in the SGA.
- The subsequent calls recall the result set from the SGA unless it ages out.

Oracle Database 11g

Cross Session Function Result Cache

```
SQL> EXEC dbms_result_cache.memory_report
```

```
Result Cache Memory Report
```

```
[Parameters]
```

```
Block Size = 1K bytes
```

```
Maximum Cache Size = 1600K bytes (1600 blocks)
```

```
Maximum Result Size = 80K bytes (80 blocks)
```

```
[Memory]
```

```
Total Memory = 103536 bytes [0.056% of the Shared Pool]
```

```
... Fixed Memory = 5140 bytes [0.003% of the Shared Pool]
```

```
... Dynamic Memory = 98396 bytes [0.053% of the Shared Pool]
```

```
..... Overhead = 65628 bytes
```

```
..... Cache Memory = 32K bytes (32 blocks)
```

```
..... Unused Memory = 26 blocks
```

```
..... Used Memory = 6 blocks
```

```
..... Dependencies = 4 blocks (4 count)
```

```
..... Results = 2 blocks
```

```
..... PLSQL = 1 blocks (1 count)
```

```
..... Invalid = 1 blocks (1 count)
```

Mixed, Named, and Position Notation

Oracle Database 11g

Mixed, Named, and Positional Notation

- **Positional Notation:**
 - Must supply items in positional order, and calls must match datatype for mandatory formal parameters.
- **Named Notation:**
 - Must supply all names and datatype matching values for all mandatory formal parameters.
- **Mixed Notation:**
 - May submit some items in positional notation while submitting others in named notation for all mandatory formal parameters.
 - There is a restriction. All positional notation calls must precede any named calls.

- **Sample function for calls:**

```
CREATE OR REPLACE FUNCTION three
( a NUMBER := 0
, b NUMBER := 0
, c NUMBER := 0 )
RETURN NUMBER IS
BEGIN
    RETURN a + b + c;
END;
/
```

- Positional Notation Call:

```
BEGIN
    dbms_output.put_line(three(3,4,5));
END;
/
```

- **Named Notation Call:**

```
BEGIN
```

```
    dbms_output.put_line(three(a => 3,b => 4,c => 5));
```

```
END;
```

```
/
```

- **Mixed Notation Call:**

```
BEGIN
```

```
    dbms_output.put_line(three(3,b => 4,c => 5));
```

```
END;
```

```
/
```

- **Exclusionary Positional Notation:**
 - Positional notations limits you to excluding the trailing or last parameters when they are optional.
 - If an optional parameter is in the middle of a set of mandatory parameters you must provide the value or a null.

- **Exclusionary Named Notation:**
 - Named notation lets you exclude optional parameters no matter where they occur in the formal parameter lists.

- **Exclusionary Mixed Notation:**
 - Mixed notation lets you exclude optional parameters provided you don't leave a positional gap in the front.
 - If a list of actual parameters begins with a positional call, the variable is assigned to the first positional parameter.

- **Exclusionary Mixed Notation:**

```
BEGIN
```

```
    dbms_output.put_line(three(3,c => 5));
```

```
END;
```

```
/
```

Sequences as Expressions

- Sequences as expressions:

```
DECLARE
  a NUMBER;
BEGIN
  a := a_sequence.nextval; -- As a right operand.
  dbms_output.put_line(a);
END;
/
```

Oracle Database 11g

Sequences as Expressions

- Sequences in embedded SQL statements:

```
BEGIN
  INSERT INTO sample
  VALUES ( a_sequence.nextval );
  FOR i IN (SELECT sample_id FROM sample)
  LOOP
    dbms_output.put_line(i.sample_id);
  END LOOP;
END;
/
```

Native Code Generation

- **Compilation Choices:**
 - Oracle Database 11g supports **Native** and **Interpreted** code compilation.
 - Interpreted compilation is the *default, which appears as a legacy* of prior releases.
 - **Native compilation** is creates an entry in the data dictionary and a shared library file stored in the `SYSTEM` tablespace.

- **Native Code Best Practices:**
 - Use native types, like `SIMPLE_INTEGER`
 - Use to perform demanding math operations.
 - Don't use when the code performs SQL statements because performance gains aren't dramatic.

- **Enabling Native Compilation:**

```
SQL> ALTER SESSION  
2 SET PLSQL_CODE_TYPE=NATIVE;
```

- **Deprecated Parameter:**

```
PLSQL_NATIVE_LIBRARY_DIR
```

Dynamic SQL Enhancements

- **Enhancements:**
 - Dynamic SQL statements can now be larger than 32 K in size.
 - The following statements support CLOB datatypes:
 - EXECUTE IMMEDIATE
 - OPEN FOR
 - DBMS_SQL.PARSE

- **Enhancements:**
 - The `DBMS_SQL` can convert from a `DBMS_SQL` cursor to a system reference cursor:
 - The `DBMS_SQL` can convert from a system reference cursor to a `DBMS_SQL` cursor.

- **Conversion Process:**
 - You use the `TO_REFCURSOR` function from the `DBMS_SQL` package to convert a `DBMS_SQL` cursor to a system reference cursor.
 - You use the `TO_CURSOR_NUMBER` function from the `DBMS_SQL` package to convert a system reference cursor to a `DBMS_SQL` cursor.

Oracle Database 11g

Dynamic SQL Enhancements

```
DECLARE
  -- System reference cursor variables.
  TYPE title_record IS RECORD
  ( item_title      VARCHAR2(60)
  , item_subtitle  VARCHAR2(60));

  TYPE title_table IS TABLE OF title_record;

  title_cursor  SYS_REFCURSOR;
  title_rows    TITLE_TABLE;

  -- DBMS_SQL variables.
  c              INTEGER := dbms_sql.open_cursor;
  fdbk           INTEGER;
  stmt           VARCHAR2(2000);
BEGIN
  ... next slide ...
END;
/
```

Oracle Database 11g

Dynamic SQL Enhancements

```
DECLARE
```

```
    ... see prior slide ...
```

```
BEGIN
```

```
    stmt := 'SELECT item_title, item_subtitle FROM item '  
           || 'WHERE REGEXP_LIKE(item_title, '^Harry Potter')';
```

```
    dbms_sql.parse(c, stmt, dbms_sql.native);
```

```
    fdbk := dbms_sql.execute(c);
```

```
    -- Convert from DBMS_SQL to system reference cursor.
```

```
    title_cursor := dbms_sql.to_refcursor(c);
```

```
    -- Open and read dynamic cursor, then close it.
```

```
    FETCH title_cursor BULK COLLECT INTO title_rows;
```

```
    FOR i IN 1..title_rows.COUNT LOOP
```

```
        dbms_output.put_line(  
            '[' || title_rows(i).item_title || ']'
```

```
            || '[' || title_rows(i).item_subtitle || ']);
```

```
    END LOOP;
```

```
    CLOSE title_cursor;
```

```
END;
```

```
/
```

Regular Expression Enhancements

- **Enhancements:**
 - Added the `REGEXP_COUNT` function.
 - Enable searching substrings in:
 - `REGEXP_SUBSTR`
 - `REGEXP_INSTR`

- **The REGEXP_COUNT Function:**
 - The function lets you count the number of times a specific pattern is found in a string.

- **Title case counting:**

```
SELECT REGEXP_COUNT(story_thread, 'The ' )  
FROM   sample_regexp  
WHERE  sample_regexp = 1;
```

- **Lower case counting:**

```
SELECT REGEXP_COUNT(story_thread, ' the ', 1, 'c')  
FROM   sample_regexp  
WHERE  sample_regexp = 1;
```

- **Case insensitive counting:**

```
SELECT REGEXP_COUNT(  
    story_thread  
    , '((^| +)|(["' ]))(T|t)he(([-:,\.\.;])|( +|$))')  
FROM    sample_regexp  
WHERE   sample_regexp = 1;
```

- **The REGEXP_SUBSTR Function:**

```
SELECT
  REGEXP_SUBSTR(story_thread, '(T|t)(o|he)*', 1, 2, 'c') AS a
, REGEXP_SUBSTR(story_thread, '(T|t)(o|he)*', 1, 2, 'c', 1) AS b
, REGEXP_SUBSTR(story_thread, '(T|t)(o|he)*', 1, 3, 'c', 2) AS c
FROM sample_regexp;
```

- **The output values:**

A	B	C
-----	-----	-----
the	t	he

- The REGEXP_INSTR Function:

```
SELECT
```

```
  REGEXP_INSTR(story_thread, '(T|t)(o|he)*', 1, 2, 0, 'c') AS a  
, REGEXP_INSTR(story_thread, '(T|t)(o|he)*', 1, 2, 0, 'c', 1) AS b  
, REGEXP_INSTR(story_thread, '(T|t)(o|he)*', 1, 3, 0, 'c', 2) AS c  
FROM sample_regexp;
```

- The output values:

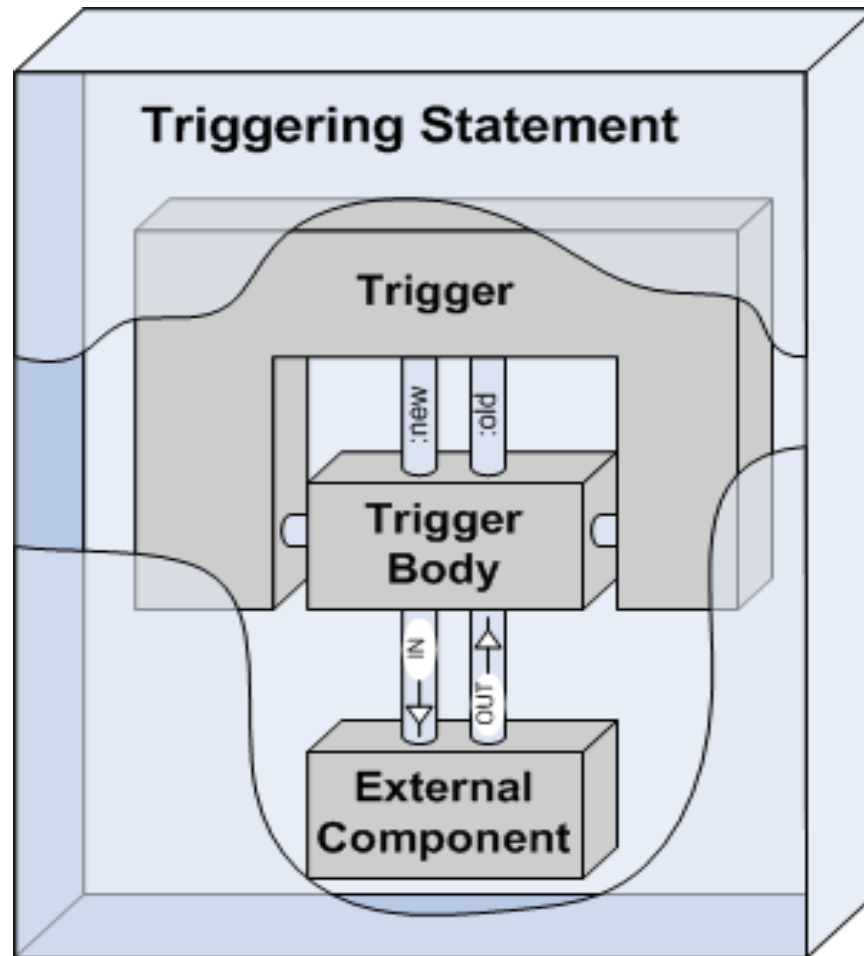
A	B	C
43	43	44

SIMPLE_INTEGER

- **Datatype features:**
 - It truncates overflow in interpreted mode.
 - It suppresses raising an overflow error in interpreted mode.
 - Casting from `PLS_INTEGER` or `BINARY_INTEGER` to a `SIMPLE_INTEGER` makes no conversion
 - Attempts to cast a null to a `SIMPLE_INTEGER` raises a runtime error.
 - It runs faster in native mode because:
 - Overload suppression is turned off.
 - Null value checking is turned off.

Compound Trigger

Oracle Database 11g Compound Trigger



Oracle Database 11g

Compound Trigger & Follows

- **Description:**
 - Compound triggers act as both statement- and row-level triggers when you *insert, update, or delete* data from a table.
 - They let you capture information at four timing points:
 - Before the firing statement
 - Before each row change from the firing statement
 - After each row change from the firing statement
 - After the firing statement
 - You can use compound triggers when you need to take action at both the statement and row events.

Oracle Database 11g Compound Trigger & Follows

- **Sample compound trigger shell:**

```
CREATE OR REPLACE TRIGGER some_trigger
FOR UPDATE ON some_table COMPOUND TRIGGER
    ... global_trigger_types_and_variables ...
BEFORE EACH ROW IS
    ... local_timing_point_types_and_variables ...
BEGIN
    ... row-level statement handling ...
END BEFORE EACH ROW;
AFTER STATEMENT IS
    ... local_timing_point_types_and_variables ...
BEGIN
    ... statement-level statement handling ...
END AFTER STATEMENT;
END;
/
```

- Sample *sequenced* trigger shell:

```
CREATE OR REPLACE TRIGGER some_trigger
```

```
FOR UPDATE ON some_table
```

```
FOR EACH ROW
```

```
FOLLOWS some_other_trigger
```

```
BEGIN
```

```
    ... row-level statement handling ...
```

```
END;
```

```
/
```

Database Resident Connection Pooling

Oracle Database 11g

Database Resource Connection Pooling

- Database Resource Connection Pooling (DRCP) lets you share connections among external programs.
- The `DBMS_CONNECTION_POOL` package lets you start, stop and manage the DRCP process.
- Start the DRCP by calling the following as the `SYS` user:

```
SQL> DBMS_CONNECTION_POOL.START_POOL();
```

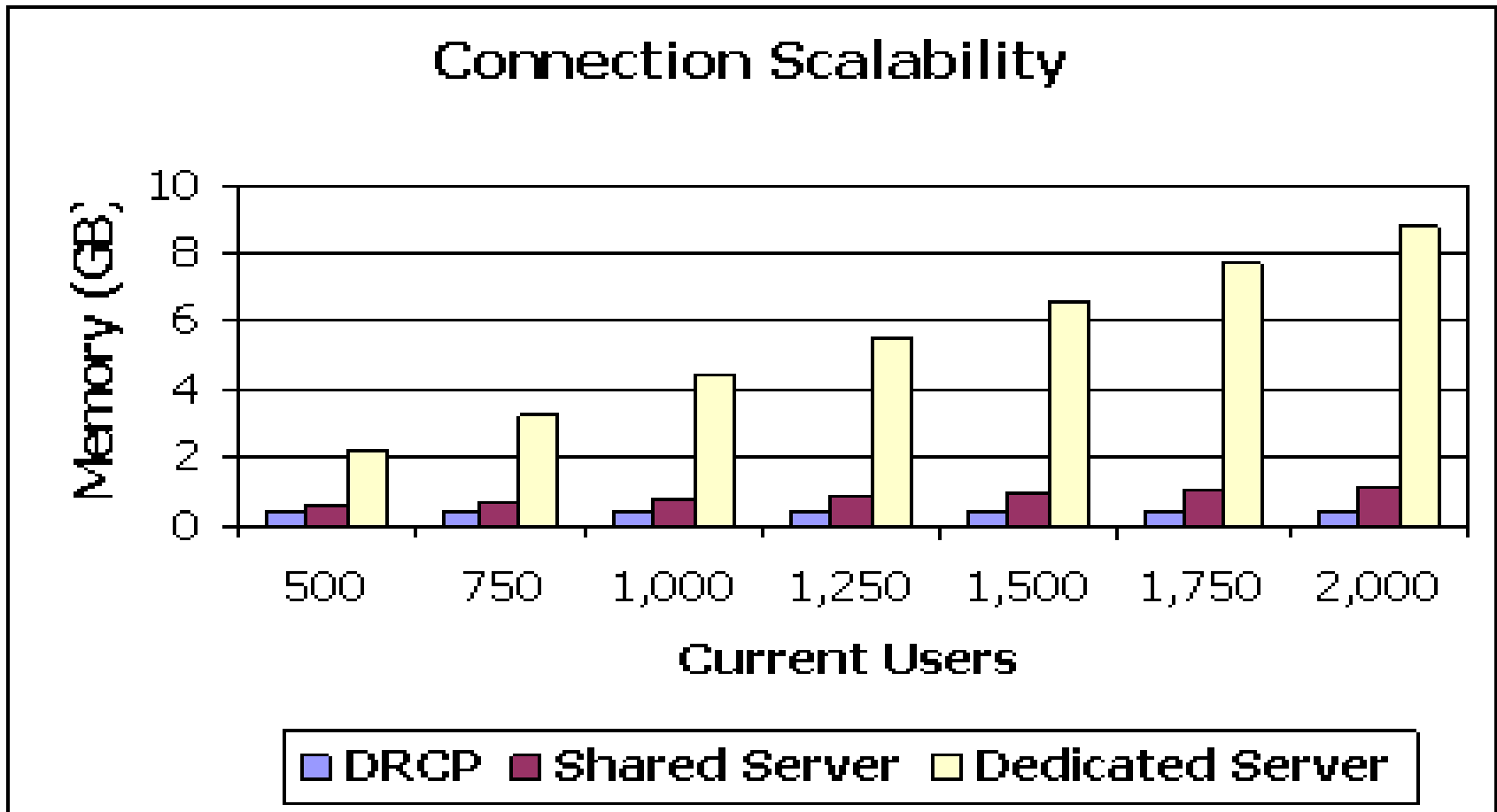
- Stop the DRCP by calling the following as the `SYS` user:

```
SQL> DBMS_CONNECTION_POOL.STOP_POOL();
```

Oracle Database 11g Database Resource Connection Pooling

```
ORCLCP =  
  (DESCRIPTION =  
    (ADDRESS = (PROTOCOL = TCP)  
              (HOST = hostname.domain)  
              (PORT = port_number)  
    )  
    (CONNECT_DATA = (SERVER = POOLED)  
                  (SERVICE_NAME = orcl)  
    )  
  )
```

Oracle Database 11g Database Resource Connection Pooling



A PL/SQL Hierarchical Profiler

- **Description:**
 - A hierarchical profiler tells you which program called what subroutine, and how many times the subroutine was called.
 - The PL/SQL hierarchical profiler stores results in a set of hierarchical profiler tables.
 - It divides the data by subprogram units, including the relationship between calling and called subroutines.
 - It further subdivides execution time by the SQL statement versus PL/SQL execution segments.

Oracle Database 11g

A PL/SQL Hierarchical Profiler

- Configuring the schema
- Collecting profile data
- Understanding raw profiler data
- Analyzing the raw data
- Understanding profiler data
- Using `p1shprof` command-line utility.

Oracle Database 11g

A PL/SQL Hierarchical Profiler

- Configure database:
 - You must first build the utility components in the SYS schema, by running the following:

```
SQL> @?/rdbms/admin/dbmshptab.sql
```

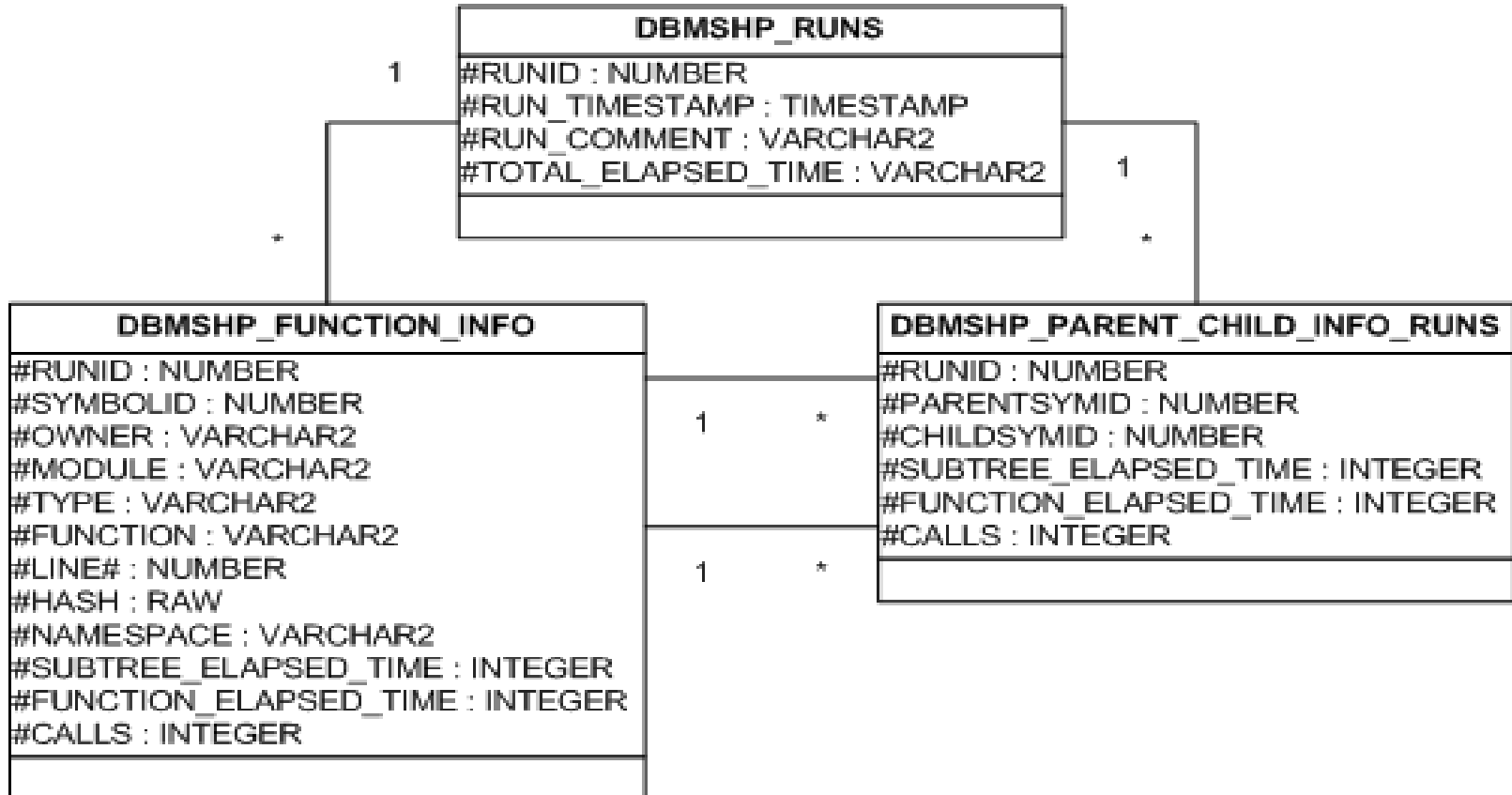
- You then must rerun the configuration in any schema where you want to use the PL/SQL Hierarchical Profiler.
- Grant privileges to the target schema:

```
GRANT EXECUTE ON dbms_hprof TO plsql;
```

```
CREATE OR REPLACE DIRECTORY profiler_dir AS '/tmp/';
```

```
GRANT READ, WRITE ON DIRECTORY profiler_dir TO plsql;
```

A PL/SQL Hierarchical Profiler



Oracle Database 11g

A PL/SQL Hierarchical Profiler

- **Collecting data:**

- Create a test application.

- Create a test program.

- You start collecting data by:

```
dbms_hprof.start_profiling('vir_dir','file_name');
```

- You end collecting data by:

```
dbms_hprof.stop_profiling;
```

Oracle Database 11g

A PL/SQL Hierarchical Profiler

- Analyzing raw data:

- Create a test application.

- Create a test program.

- You start collecting data by:

```
dbms_hprof.start_profiling('vir_dir','file_name');
```

- You end collecting data by:

```
dbms_hprof.stop_profiling;
```

Oracle Database 11g

A PL/SQL Hierarchical Profiler

- Analyzing the raw output file:

```
VARIABLE output NUMBER
```

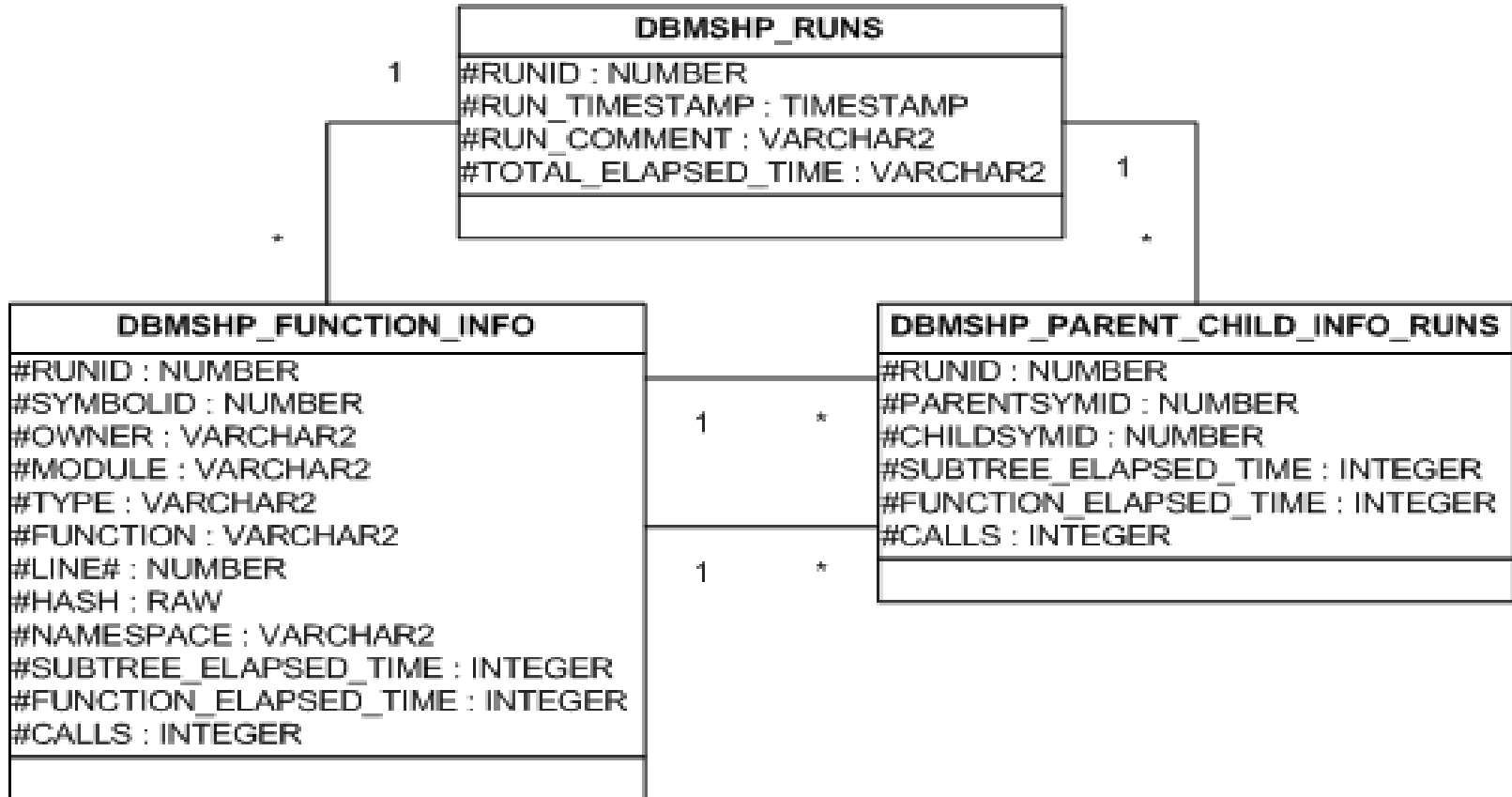
```
SQL> CALL
```

```
2 dbms_hprof.analyze('PROFILER_DIR','Harry.txt')
```

```
3 INTO :output;
```

Oracle Database 11g

A PL/SQL Hierarchical Profiler



Oracle Database 11g

A PL/SQL Hierarchical Profiler

- Reading the raw data:

SP#X 1

P#R

P#C SQL." " ." " ."__sql_fetch_line17" #17

P#X 27

P#R

P#C PLSQL."SYS"."DBMS_OUTPUT": :11."PUT"#5892e4d73b579470 #77

P#X 1

P#R

P#C PLSQL."SYS"."DBMS_OUTPUT": :11."PUT"#5892e4d73b579470 #77

P#X 1

P#R

P#C SQL." " ." " ."__sql_fetch_line17" #17

P#X 23

P#R

Oracle Database 11g

A PL/SQL Hierarchical Profiler

- Querying the analyzed data:

```
SELECT RPAD(' ',level*2,' ')||dfi.owner||'.'||dfi.module
,      dfi.function
,      (dpci.subtree_elapsed_time/1000)
,      (dpci.function_elapsed_time/1000)
,      dpci.calls
FROM    dbmshp_parent_child_info dpci, dbmshp_function_info dfi
WHERE   dpci.runid = dfi.runid
AND     dpci.parentsymid = dfi.symbolid
AND     dpci.runid = 1
CONNECT BY PRIOR dpci.childsymid = dpci.parentsymid
START WITH          dpci.parentsymid = 1;
```

Oracle Database 11g

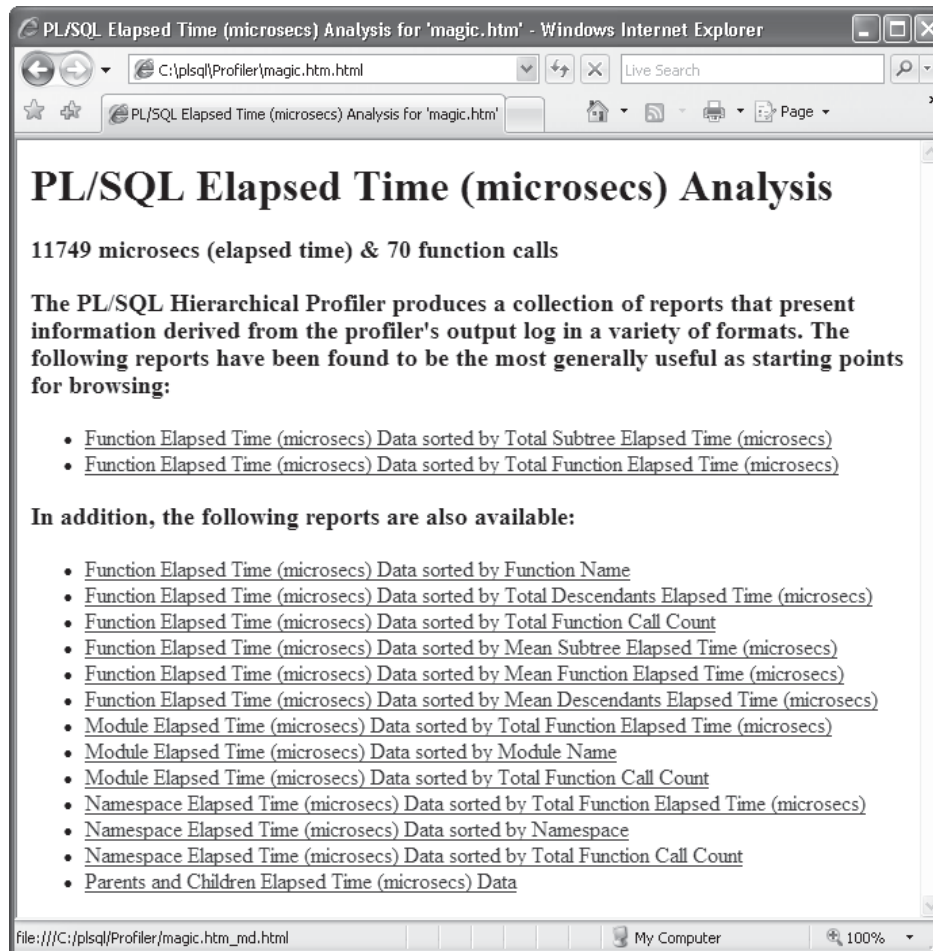
A PL/SQL Hierarchical Profiler

METHOD_NAME	FUNCTION_NAME	Subtree Elapsed Time	Function Elapsed Time	Calls
.	__plsql_vm	.04	.04	11
PLSQL.GLUE_STRINGS	GLUE_STRINGS	.00	.00	0
PLSQL.QUANTITY_ONHAND	QUANTITY_ONHAND	.29	.05	1
PLSQL.QUANTITY_ONHAND	QUANTITY_ONHAND.C	.24	.24	1
PLSQL.QUANTITY_ONHAND	QUANTITY_ONHAND	.12	.03	11
SYS.DBMS_OUTPUT	PUT_LINE	.02	.02	11
SYS.DBMS_OUTPUT	PUT_LINE	.06	.05	11
SYS.DBMS_OUTPUT	PUT	.02	.02	1
PLSQL.QUANTITY_ONHAND	QUANTITY_ONHAND	3.27	3.19	1

9 rows selected.

Oracle Database 11g

A PL/SQL Hierarchical Profiler



The screenshot shows a Windows Internet Explorer browser window displaying a report titled "PL/SQL Elapsed Time (microsecs) Analysis for 'magic.htm'". The browser's address bar shows the file path "C:\plsqli\Profiler\magic.htm.html". The report content includes a sub-header "PL/SQL Elapsed Time (microsecs) Analysis", a summary "11749 microsecs (elapsed time) & 70 function calls", and an introductory paragraph. It lists several report options, with two highlighted in the original image: "Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs)" and "Function Elapsed Time (microsecs) Data sorted by Total Function Elapsed Time (microsecs)". Other options include sorting by Function Name, Total Descendants Elapsed Time, Total Function Call Count, Mean Subtree Elapsed Time, Mean Function Elapsed Time, Mean Descendants Elapsed Time, Module Elapsed Time (sorted by Total Function Elapsed Time and Module Name), Namespace Elapsed Time (sorted by Total Function Elapsed Time and Namespace), and Parents and Children Elapsed Time. The browser's status bar at the bottom shows the file path "file:///C:/plsqli/Profiler/magic.htm_md.html", the system tray with "My Computer", and a zoom level of "100%".

PL/SQL Elapsed Time (microsecs) Analysis for 'magic.htm' - Windows Internet Explorer

C:\plsqli\Profiler\magic.htm.html

PL/SQL Elapsed Time (microsecs) Analysis for 'magic.htm'

PL/SQL Elapsed Time (microsecs) Analysis

11749 microsecs (elapsed time) & 70 function calls

The PL/SQL Hierarchical Profiler produces a collection of reports that present information derived from the profiler's output log in a variety of formats. The following reports have been found to be the most generally useful as starting points for browsing:

- [Function Elapsed Time \(microsecs\) Data sorted by Total Subtree Elapsed Time \(microsecs\)](#)
- [Function Elapsed Time \(microsecs\) Data sorted by Total Function Elapsed Time \(microsecs\)](#)

In addition, the following reports are also available:

- [Function Elapsed Time \(microsecs\) Data sorted by Function Name](#)
- [Function Elapsed Time \(microsecs\) Data sorted by Total Descendants Elapsed Time \(microsecs\)](#)
- [Function Elapsed Time \(microsecs\) Data sorted by Total Function Call Count](#)
- [Function Elapsed Time \(microsecs\) Data sorted by Mean Subtree Elapsed Time \(microsecs\)](#)
- [Function Elapsed Time \(microsecs\) Data sorted by Mean Function Elapsed Time \(microsecs\)](#)
- [Function Elapsed Time \(microsecs\) Data sorted by Mean Descendants Elapsed Time \(microsecs\)](#)
- [Module Elapsed Time \(microsecs\) Data sorted by Total Function Elapsed Time \(microsecs\)](#)
- [Module Elapsed Time \(microsecs\) Data sorted by Module Name](#)
- [Module Elapsed Time \(microsecs\) Data sorted by Total Function Call Count](#)
- [Namespace Elapsed Time \(microsecs\) Data sorted by Total Function Elapsed Time \(microsecs\)](#)
- [Namespace Elapsed Time \(microsecs\) Data sorted by Namespace](#)
- [Namespace Elapsed Time \(microsecs\) Data sorted by Total Function Call Count](#)
- [Parents and Children Elapsed Time \(microsecs\) Data](#)

file:///C:/plsqli/Profiler/magic.htm_md.html My Computer 100%

A PL/Scope

- **Basics:**

- Lets you qualify identifiers in programs.
- It is intended for development databases *only*.
- Is disabled by default.
- You enable by:

```
ALTER SESSION SET PLSCOPE_SETTINGS = 'IDENTIFIERS:ALL';
```

- It remains enabled through the session.
- You can disable it at any time.

Oracle Database 11g PL/Scope

- Auditing impact on database:
 - It logs information to the SYSAUX tablespace:
 - You monitor space impact by:

```
SELECT space_usage_kbytes  
FROM v$sysaux_occupants  
WHERE occupant_name = 'PL/SCOPE';
```

- You can query the identifiers from the:
 - ALL_IDENTIFIERS
 - DBA_IDENTIFIERS
 - USER_IDENTIFIERS
- SQL Developer also leverages the information.

Oracle Database 11g PL/Scope

- Identifiers collected by PL/Scope:

ASSOCIATIVE_ARRAY	FORMAL OUT	RECORD
BFILE	FUNCTION	REFCURSOR
BLOB	INTERVAL	SUBTYPE
BOOLEAN	ITERATOR	SYNONYM
CHARACTER	LABEL	TIME
CLOB	LIBRARY	TIMESTAMP
CONSTANT	NESTED TABLE	TRIGGER
CURSOR	NUMBER	UROWID
DATE	OBJECT	VARRAY
EXCEPTION	OPAQUE	VARIABLE
FORMAL IN	PACKAGE	
FORMAL IN OUT	PROCEDURE	

Generalized Object Invocation

Oracle Database 11g

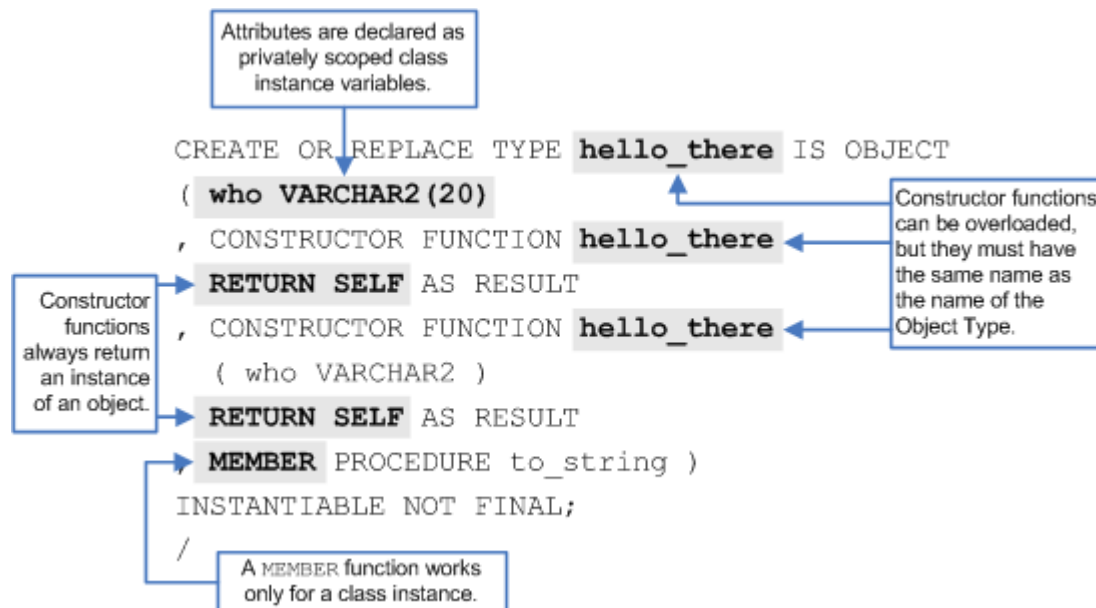
Generalized Object Invocation

- **Object Types:**
 - Have an object type and body
 - Persistent object types:
 - Standalone object types: Are assigned as a column datatype and hold a unique object identifier.
 - Embedded object types: Are declared in another stored program unit, like function, procedure, package, or object type.
 - Transient object types: Are schema-level SQL datatypes that contain constructors and methods.

Oracle Database 11g

Generalized Object Invocation

- Declaring an object type:



Oracle Database 11g

Generalized Object Invocation

- Declaring an object type body:

```

CREATE OR REPLACE TYPE BODY hello_there IS
  CONSTRUCTOR FUNCTION hello_there -- Default constructor.
  RETURN SELF AS RESULT IS
    hello HELLO_THERE := hello_there('Generic Object');
  BEGIN
    self := hello;
    RETURN;
  END hello_there;
  CONSTRUCTOR FUNCTION hello_there -- Overriding constructor.
  (who VARCHAR2) RETURN SELF AS RESULT IS
  BEGIN
    self.who := who;
    RETURN;
  END hello_there;
  MEMBER PROCEDURE to_string IS
  BEGIN
    dbms_output.put_line('Hello ' || self.who || '.');
  END to_string;
END hello_there;
/

```

Oracle Database 11g

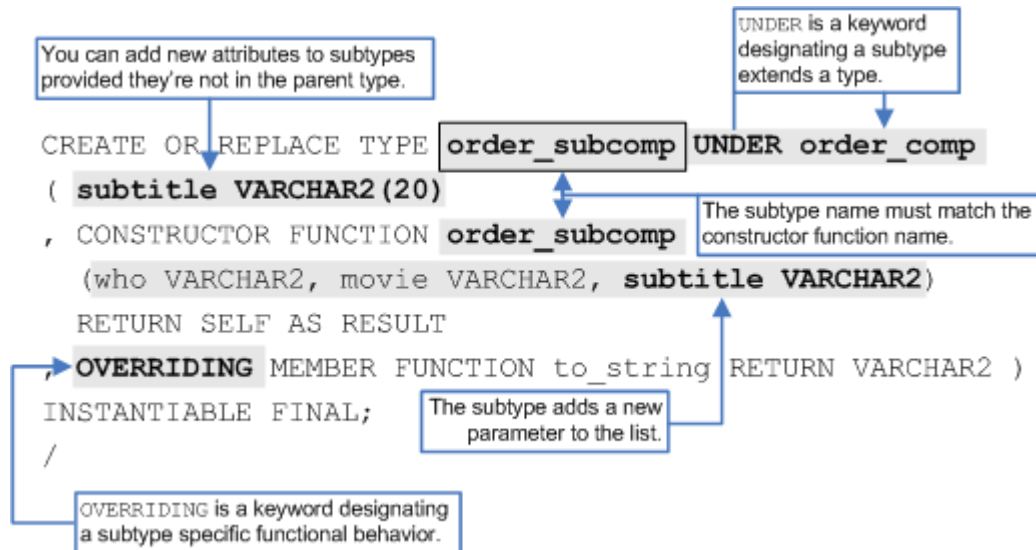
Generalized Object Invocation

- **Object subtypes:**
 - Inherit attributes from parent object types:
 - Subtypes can access parent attributes.
 - Subtypes can assign values to parent attributes.
 - Subtypes can't be overridden.
 - Inherit constructors from parent object types:
 - Subtypes have access to parent constructors.
 - Subtypes can't override parent constructors.
 - Subtypes with constructors that mirror signatures of the parent are ignored but don't raise a compilation error.

Oracle Database 11g

Generalized Object Invocation

- Declaring a subclass of an object type:



Oracle Database 11g

Generalized Object Invocation

- Declaring a subclass object type body:

```

CREATE OR REPLACE TYPE BODY order_subcomp IS
  CONSTRUCTOR FUNCTION order_subcomp
    (who VARCHAR2, movie VARCHAR2, subtitle VARCHAR2)
  RETURN SELF AS RESULT IS
  BEGIN
    self.who := who;
    self.movie := movie;
    self.subtitle := subtitle;
    RETURN;
  END order_subcomp;
OVERRIDING MEMBER FUNCTION to_string RETURN VARCHAR2 IS
  BEGIN
    RETURN (self as order_comp).to_string || '[' || self.subtitle || ']';
  END to_string;
END;
/

```

Only the overriding constructor is implemented in a subtype because they have native access to their parent class constructors.

The attributes are declared in the parent or supertype (also superclass).

The parenthetical call is a generalized invocation because it calls to a parent object type. It places its data inside a copy of the parent class, and runs methods as if it is the parent class.

The `to_string()` method call acts on the superclass method that the subclass overrides.

OVERRIDING is a key word designating the subtype runs this copy of a function that must also exist in the parent.

Oracle Database 11g Questions

