Item: 7 (Ref:1z0-147e.5.4.6)

Examine this package specification and body:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which statement about the CURRENT_AVG_COST_PER_TICKET variable is true?

- ○ It can only be referenced from outside the package.
- ○ It can be referenced from outside and within the package.
- ○ It can be referenced by all constructs within the package.
- ○ It must be moved to the specification to compile successfully.

Answer:
It can be referenced by all constructs within the package.

Explanation:

Variables declared in the package body declaration can be referenced from all constructs within the package. These variables are considered private and cannot be referenced from outside the package.

Objective:
Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 14 (Ref:1z0-147e.5.4.4)

While creating a package, you placed the function name in the specification and the body. Which type of construct have you created?

- ○ public
- ○ illegal
- ○ private
- ○ one-time only

Answer:
public

Explanation:
All constructs must be declared in the package body. Constructs that are also declared in a package specification are public and can be referenced from inside or outside of the package.

Objective:
Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 15 (Ref:1z0-147e.5.4.5)

Examine this package specification and body:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which statement about the FIND_SEATS_SOLD procedure is true?

- ○ It can be referenced within a SQL statement.
- ○ It can only be referenced from within the package.
- ○ It can be referenced from within and outside of the package.
- ○ It cannot also be specified in the specification because it is specified in the body.

Answer:
It can be referenced from within and outside of the package.

Explanation:
Procedures declared in a package specification are public and can be referenced from inside or outside of the package.

Objective:

Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 16 (Ref:1z0-147e.5.4.8)

Examine this package specification:

```
CREATE OR REPLACE PACKAGE theater_package
IS
PROCEDURE find_cpt
(v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER);
PROCEDURE update_theater (v_name IN VARCHAR2);
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
PROCEDURE add_theater;
END theater_package;
```

Which statement about the procedures in this specification is true?

- ○ They are public procedures.
- ○ They are private procedures.
- ○ Each procedure is missing code and, therefore, is illegal.
- ○ Each procedure contains an argument list and, therefore, is illegal.

Answer:
They are public procedures.

Explanation:
Constructs listed in a package specification are considered public and can be invoked from outside the package.

Objective:
Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 20 (Ref:1z0-147e.5.3.2)

Examine this package specification and body:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which will successfully invoke the FIND_SEATS_SOLD procedure within SQL*Plus?

- ○ EXECUTE find_seats_sold (500,11);
- ○ theater_pck.find_seats_sold (500,11);
- ○ EXECUTE theater_pck.find_seats_sold (500,11);
- ○ SELECT find_seats_sold(movie_id, theater_id)
  FROM gross_receipt;

Answer:
EXECUTE theater_pck.find_seats_sold (500,11);

Explanation:

Procedures within a package must be invoked using the package name as a prefix.

The EXECUTE command is used in SQL*Plus to invoke PL/SQL constructs.

Procedures, stand-alone or part of a package, cannot be invoked from a SQL statement.

Objective:
Create Packages

Sub-Objective:
Create packages: Create related variables, cursors, constants, exceptions, procedures, and functions

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 29 (Ref:1z0-147e.5.4.7)

Examine this package specification and body:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
v_total_budget NUMBER;
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which statement about the V_TOTAL_BUDGET variable is true?

- ◯ It must also be declared in the body.
- ◯ It can only be referenced from inside the package.
- ◯ It can only be referenced from outside the package.
- ◯ It can be referenced from inside and outside the package.

Answer:
It can be referenced from inside and outside the package.

Explanation:
Variables declared in the specification are public and can be referenced from inside and outside of the package.

Objective:
Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 58 (Ref:1z0-147e.5.5.3)

Examine this package body:

```
CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

BEGIN
current_avg_cost_per_ticket := 8.50;
END theater_pck;
```

Which statement about the value of CURRENT_AVG_COST_PER_TICKET is true?

- It is 0 until explicitly referenced.
- It is assigned 8.50 each time the package is referenced.
- It is assigned 8.50 only when it is explicitly referenced.
- It is assigned 8.50 when the package is first invoked within a session.

Answer:
It is assigned 8.50 when the package is first invoked within a session.

Explanation:
The body of a package has a header, declaration, and executable section.

The executable section of a package body is specified using the BEGIN keyword after all subprograms are declared and before the END package_name. Code that is written in this section is one-time-only code; it executes the first time the package is referenced within a session. It will not execute this code again unless the user changes sessions or the package is recompiled.

The CURRENT_AVG_COST_PER_TICKET is assigned 8.50 the first time the packaged is referenced and retains this value for the session.

Objective:
Create Packages

Sub-Objective:
Invoke a package construct

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 62 (Ref:1z0-147e.5.5.4)

Examine this function:

```
CREATE OR REPLACE FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END;
```

This function is owned by the account, PROD. The user, JSMITH, must execute this function. Which GRANT command(s) should be issued?

- GRANT EXECUTE ON get_budget TO jsmith;
- GRANT EXECUTE, SELECT ON studio TO jsmith;
- GRANT EXECUTE, SELECT ON get_budget TO jsmith;
- GRANT SELECT ON studio TO jsmith;
  GRANT EXECUTE ON get_budget TO jsmith;

Answer:
GRANT EXECUTE ON get_budget TO jsmith;

Explanation:
Unless you are the owner of a PL/SQL construct, you must be granted the EXECUTE privilege to run it.

The executor only requires the EXECUTE privilege and does not require the privileges on the objects referenced within the construct unless the AUTHID CURRENT_USER clause in the header to require privileges of the executor on the referenced objects. If this clause had been added to the construct, JSMITH would also need the SELECT privilege on the STUDIO table.

Objective:
Create Packages

Sub-Objective:
Invoke a package construct

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 67 (Ref:1z0-147e.5.7.2)

Evaluate this statement:

DROP PACKAGE dept_pack;

Which statement is true?

- ○ This statement removes only the package body.
- ○ The statement removes only the package specification.
- ○ The statement removes the package specification and the package body.
- ○ This statement contains a syntax error.

Answer:
The statement removes the package specification and the package body.

Explanation:
The DROP PACKAGE package_name; statement removes the package specification and the body. To remove only the package body, issue the DROP PACKAGE BODY package_name; statement. The DROP PACKAGE BODY statement retains the package specification.

Objective:
Create Packages

Sub-Objective:
Drop Packages

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 78 (Ref:1z0-147e.5.4.9)

Examine this package specification and body:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which statement about the GET_BUDGET function is true?

- ◯ It can be referenced within a SQL statement.
- ◯ It can be referenced from outside of the package.
- ◯ It can only be referenced from within the package.
- ◯ It must also be specified in the specification because it is specified in the body.

Answer:
It can only be referenced from within the package.

Explanation:
Private constructs are declared in the package body only. These constructs can only be invoked from a construct within the same package. They cannot be invoked from outside the package.

Objective:

Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 92 (Ref:1z0-147e.5.5.2)

Examine this function:

```
CREATE OR REPLACE FUNCTION set_budget
(v_studio_id IN NUMBER, v_new_budget IN NUMBER)
RETURN number
IS
BEGIN
UPDATE studio
SET yearly_budget = v_new_budget
WHERE id = v_studio_id;
COMMIT;
RETURN SQL%ROWCOUNT;
END;
```

While executing this in SQL*Plus, you want to see the value of SQL%ROWCOUNT displayed on the screen. Which line of code will accomplish this?

- ○ OUTPUT.PUT_LINE(TO_CHAR(SQL%ROWCOUNT));
- ○ DBMS_DEBUG.PUT_LINE(TO_CHAR(SQL%ROWCOUNT));
- ○ DBMS_OUTPUT.DISPLAY(TO_CHAR(SQL%ROWCOUNT));
- ○ DBMS_OUTPUT.PUT_LINE(TO_CHAR(SQL%ROWCOUNT));

Answer:
DBMS_OUTPUT.PUT_LINE(TO_CHAR(SQL%ROWCOUNT));

Explanation:
DBMS_OUTPUT is an Oracle supplied package that allows you to display messages during a SQL*Plus session. PUT_LINE is a procedure within this package that places a line of text into a buffer and then displays the contents of the buffer to the screen.

To view the results of the DBMS_OUTPUT package in SQL*Plus, you must first issue the SET SERVEROUTPUT ON command.

Objective:
Create Packages

Sub-Objective:
Invoke a package construct

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 94 (Ref:1z0-147e.5.5.1)

Which command must you issue in SQL*Plus to display the result of the DBMS_OUTPUT package?

- ○ SET ECHO ON
- ○ SET OUTPUT ON
- ○ SET FEEDBACK ON
- ○ SET SERVEROUTPUT ON

Answer:
SET SERVEROUTPUT ON

Explanation:
DBMS_OUTPUT is an Oracle supplied package that allows you to display messages during a SQL*Plus session. PUT_LINE is a procedure within this package that places a line of text into a buffer and then displays the contents of the buffer to the screen.

You must issue the command, SET SERVEROUTPUT ON, for this package to work in SQL*Plus.

Objective:
Create Packages

Sub-Objective:
Invoke a package construct

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 95 (Ref:1z0-147e.5.8.1)

Which statement about packages is true?

- ○ A package can be called.
- ○ A package can be nested.
- ○ A package can be parameterized.
- ○ Package contents can be shared by multiple applications.

Answer:
Package contents can be shared by multiple applications.

Explanation:
Unlike a procedure or function, the package itself cannot be called, parameterized, or nested. However, once compiled, the contents of a package can be shared by multiple applications.

Objective:
Create Packages

Sub-Objective:
Identify benefits of Packages

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 112 (Ref:1z0-147e.5.8.2)

Which statement about packages is true?

- ○ Package constructs stored in the database are public.
- ○ Package constructs defined within the package specification are private.
- ○ Private and public package constructs are both declared and defined within the package body.
- ○ A private package construct can only be referenced only by other constructs within the same package.

Answer:
A private package construct can only be referenced only by other constructs within the same package.

Explanation:

A package consists of two parts, the package specification and the package body. A package construct can be either public or private. Public package constructs are declared in the package specification and defined in the package body. Private package constructs are defined only within the package body. Public constructs can be referenced from any Oracle server environment, but private constructs can only be referenced by other constructs that are in the same package.

Objective:

Create Packages

Sub-Objective:

Identify benefits of Packages

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 129 (Ref:1z0-147e.5.3.1)

You decide to use packages to logically group related programming constructs. Which two types of constructs can be grouped within a package? (Choose two.)

- ☐ view
- ☐ cursor
- ☐ variable
- ☐ database trigger
- ☐ application trigger

Answer:
cursor
variable

Explanation:
Database and application triggers are constructs that cannot be part of a package.

Though a package may contain a construct that references a view, the view itself, cannot be part of a package.

Objective:
Create Packages

Sub-Objective:
Create packages: Create related variables, cursors, constants, exceptions, procedures, and functions

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 133 (Ref:1z0-147e.5.3.3)

Examine this package specification and body:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which code will successfully assign a value to the CURRENT_AVG_COST_PER_TICKET variable within SQL*Plus?

-
  ```
  BEGIN

  current_avg_cost_per_ticket := 6.75;

  END;
  ```

-
  ```
  BEGIN

  theater_pck.current_avg_cost_per_ticket := 6.75;

  END;
  ```

-
  ```
  BEGIN
  SELECT AVG(cost_per_ticket)
  INTO current_avg_cost_per_ticket
  FROM gross_receipt;
  END;
  ```

○ This variable is private to the package and cannot be directly assigned a value within SQL*Plus.

Answer:
This variable is private to the package and cannot be directly assigned a value within SQL*Plus.

Explanation:
Constructs declared in the package body only are considered to be private and cannot be invoked or referenced from outside the package.

Objective:
Create Packages

Sub-Objective:
Create packages: Create related variables, cursors, constants, exceptions, procedures, and functions

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 141 (Ref:1z0-147e.5.4.1)

Examine this package specification and body:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which code will successfully invoke the GET_BUDGET function within SQL*Plus?

-
  ```
  VARIABLE g_budget_amount NUMBER
  EXECUTE :g_budget_amount := get_budget(11);
  ```

-
  ```
  VARIABLE g_budget_amount NUMBER
  EXECUTE theater_pck.get_budget(g_budget_amount, 11);
  ```

-
  ```
  VARIABLE g_budget_amount NUMBER
  EXECUTE :g_budget_amount := theater_pck.get_budget(11);
  ```
- This function cannot be referenced from outside the package.

Answer:
This function cannot be referenced from outside the package.

Explanation:

Constructs declared in the package body only are considered to be private and cannot be invoked from outside the package.

Objective:
Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 161 (Ref:1z0-147e.5.2.1)

Examine this code:

```
CREATE OR REPLACE PACKAGE prod_pack
IS
g_tax_rate NUMBER := .08;
END prod_pack;
```

Which statement about this code is true?

- ○ This package specification can exist without a body.
- ○ This package body can exist without a specification.
- ○ This package body cannot exist without a specification.
- ○ This package specification cannot exist without a body.

Answer:
This package specification can exist without a body.

Explanation:
A package specification is created using the CREATE OR REPLACE PACKAGE statement. Package specifications without a body are used to create public variables and do not contain subprograms.

Objective:
Create Packages

Sub-Objective:
Identify a package specification and body

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 162 (Ref:1z0-147e.5.2.3)

Examine this procedure:

```
PROCEDURE find_cpt
(v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER)
IS
BEGIN
IF v_cost_per_ticket > 8.50 THEN
SELECT cost_per_ticket
INTO v_cost_per_ticket
FROM gross_receipt
WHERE movie_id = v_movie_id;
END IF;
END;
```

You decide to create this procedure within the THEATER_PCK package. It will be accessible outside of the package. What will you add to the package specification?

-
  ```
  PROCEDURE find_cpt
  (v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER);
  ```

-
  ```
  PUBLIC PROCEDURE find_cpt
  (v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER);
  ```

-
  ```
  PROCEDURE find_cpt
  (v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER)
  IS
  BEGIN
  IF v_cost_per_ticket > 8.50 THEN
  SELECT cost_per_ticket
  INTO v_cost_per_ticket
  FROM gross_receipt
  WHERE movie_id = v_movie_id;
  END IF;
  END;
  ```

-
  ```
  PUBLIC PROCEDURE find_cpt
  (v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER)
  IS
  BEGIN
  IF v_cost_per_ticket > 8.50 THEN
  SELECT cost_per_ticket
  INTO v_cost_per_ticket
  FROM gross_receipt
  WHERE movie_id = v_movie_id;
  END IF;
  END;
  ```

Answer:
PROCEDURE find_cpt (v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER);

Explanation:

To make a procedure public within a package, declare the procedure header in the specification. The header includes the procedure name and arguments:

```
PROCEDURE find_cpt
(v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER);
```

Objective:

Create Packages

Sub-Objective:
Identify a package specification and body

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 166 (Ref:1z0-147e.5.4.2)

Examine this code:

```
BEGIN
theater_pck.v_total_seats_sold_overall := theater_pck.get_total_for_year;
END;
```

For this code to be successful, what must be true?

- ○ Only the GET_TOTAL_FOR_YEAR variable must exist in the specification of the THEATER_PCK package.
- ○ Only the V_TOTAL_SEATS_SOLD_OVERALL variable must exist in the specification of the THEATER_PCK package.
- ○ Both the V_TOTAL_SEATS_SOLD_OVERALL variable and the GET_TOTAL_FOR_YEAR function must exist only in the body of the THEATER_PCK package.
- ○ Both the V_TOTAL_SEATS_SOLD_OVERALL variable and the GET_TOTAL_FOR_YEAR function must exist in the specification of the THEATER_PCK package.

Answer:

Both the V_TOTAL_SEATS_SOLD_OVERALL variable and the GET_TOTAL_FOR_YEAR function must exist in the specification of the THEATER_PCK package.

Explanation:
Only constructs declared in the package specification are public and can be referenced from outside the package by prefixing them with the package name.

Objective:
Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Item: 167 (Ref:1z0-147e.5.4.3)

Examine this package specification and body:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which type of variable is CURRENT_AVG_COST_PER_TICKET?

- public
- private
- invalid
- constant

Answer:
private

Explanation:

Variables declared in the package body declaration can be referenced from all constructs within the package. These variables are considered private and cannot be referenced from outside the package.

Objective:

Create Packages

Sub-Objective:
Designate a package construct as either public or private

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Packages

Creating Database Triggers

Item: 11 (Ref:1z0-147e.9.2.2)

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER prevent_gross_modification
{additional trigger code}
BEGIN
IF TO_CHAR(sysdate,'DY') = 'MON' THEN
RAISE_APPLICATION_ERROR(-20000, 'Gross receipts cannot be entered on Monday');
END IF;
END;
```

This trigger must fire before each DELETE of the GROSS_RECEIPT table. It should fire only once for the entire DELETE statement. Which additional trigger code must you add?

- ○ AFTER DELETE ON gross_receipt
- ○ BEFORE (gross_receipt) DELETE
- ○ BEFORE DELETE ON gross_receipt
- ○ FOR EACH ROW DELETED FROM gross_receipt

Answer:
BEFORE DELETE ON gross_receipt

Explanation:

Trigger timing is specified using the BEFORE or AFTER keywords. If the trigger body should execute before the event, use BEFORE. If the trigger body should execute after the event, use AFTER.

Trigger events include INSERT, UPDATE, and DELETE. If the event is an UPDATE of the triggering table, use UPDATE ON <table name>. If the event is an update of a particular column or columns, use UPDATE OF <column name, column name> ON <table_name>.

For example:
AFTER UPDATE ON emp
AFTER UPDATE OF sal ON emp

Objective:

Creating Database Triggers

Sub-Objective:

List how triggers are used

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 13 (Ref:1z0-147e.9.1.3)

Procedures and functions are explicitly executed. This is different from a database trigger. When is a database trigger executed?

- ○ when the transaction is committed
- ○ during a data manipulation statement
- ○ when it is explicitly invoked by another construct
- ○ when an Oracle supplied package references the trigger

Answer:
during a data manipulation statement

Explanation:
A database trigger is implicitly invoked due to an event that is specified upon the trigger's creation. The events include Data Manipulation Language (DML) and Data Definition Language (DDL) commands.

Objective:
Creating Database Triggers

Sub-Objective:
Describe the different types of triggers

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 23 (Ref:1z0-147e.9.4.1)

Examine this trigger:

```
CREATE OR REPLACE TRIGGER audit_gross_receipt
AFTER DELETE OR UPDATE OF seats_sold, cost_per_ticket ON gross_receipt
BEGIN
...
END;
```

How many times will the trigger body execute upon invocation?

- ○ once
- ○ twice
- ○ once for each row deleted or updated
- ○ once for each row deleted or seats_sold_cost_per_ticket updated

Answer:
once

Explanation:
A trigger without the FOR EACH ROW clause is a statement trigger; it executes the trigger body once for the entire event.

A row trigger fires for each row affected by the event.

An event is a Data Manipulation Language (DML) or a Data Definition Language (DDL) command event.

Objective:
Creating Database Triggers

Sub-Objective:
Create a DML trigger

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 36 (Ref:1z0-147e.9.2.4)

When a database trigger routine does not have to take place before the triggering event, which timing should you assign to the trigger?

- ○ ON
- ○ OFF
- ○ AFTER
- ○ BEFORE

Answer:
AFTER

Explanation:
Trigger timing is specified using the BEFORE or AFTER keywords. If the trigger body should execute before the event, use the BEFORE keyword. If the trigger body should execute after the event, use the AFTER keyword.

Objective:
Creating Database Triggers

Sub-Objective:
List how triggers are used

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 47 (Ref:1z0-147e.9.2.5)

Modifications to the THEATER table are not allowed during the last week in December. When creating a database trigger to enforce this rule, which timing will you use to be most efficient?

- ○ AFTER
- ○ WEEKLY
- ○ BEFORE
- ○ MONTHLY

Answer:
BEFORE

Explanation:
Trigger timing is specified using BEFORE or AFTER. If the trigger body should execute before the event, use BEFORE. If the trigger body should execute after the event, use AFTER.

In this case, it would be much more efficient to create a BEFORE trigger, stopping the event before it begins. If you create an AFTER trigger, the routine will be performed and then rolled back.

Objective:
Creating Database Triggers

Sub-Objective:
List how triggers are used

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 52 (Ref:1z0-147e.9.2.6)

```
CREATE OR REPLACE TRIGGER update_studio
BEFORE UPDATE OF yearly_budget ON STUDIO
FOR EACH ROW
BEGIN
...
END;
```

Which event will invoke this trigger?

- ○ STUDIO table update
- ○ YEARLY_BUDGET column update
- ○ STUDIO table insert, update, or delete
- ○ any column update other than YEARLY_BUDGET

Answer:
YEARLY_BUDGET column update

Explanation:

"BEFORE UPDATE OF yearly_budget ON STUDIO" specifies that the trigger should only be invoked if the YEARLY_BUDGET column of the STUDIO table is updated.

If "BEFORE UPDATE ON studio" is specified, the trigger will execute when any column of the STUDIO table is updated.

Objective:

Creating Database Triggers

Sub-Objective:

List how triggers are used

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 55 (Ref:1z0-147e.9.4.2)

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER audit_gross_modification
AFTER INSERT OR DELETE ON gross_receipt
BEGIN
INSERT INTO audit_gross
VALUES (USER, SYSDATE);
END;
```

To test this trigger, you delete 30 rows from the GROSS_RECEIPT table. How many rows are inserted into the AUDIT_GROSS table due to this event?

- ○ 1
- ○ 30
- ○ 31
- ○ none

Answer:
1

Explanation:
The trigger is statement level; it fires only once for the entire triggering event. Therefore, even though 30 rows were deleted from the GROSS_RECEIPT table, only one was inserted into the AUDIT_GROSS table.

Objective:
Creating Database Triggers

Sub-Objective:
Create a DML trigger

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 60 (Ref:1z0-147e.9.8.2)

The auditing utility in Oracle records the type of data manipulation operation and not the actual changed values. To enhance auditing by capturing the new and old values, you create which type of trigger?

- ○ row only
- ○ statement only
- ○ either statement or row
- ○ neither statement nor row

Answer:
row only

Explanation:
Only row level database triggers can capture the old and new values. Using the qualifiers, :OLD and :NEW, within a trigger body, allows you to capture the value before the execution of the trigger body and the value after the execution of the trigger body.

Objective:
Creating Database Triggers

Sub-Objective:
Create a row level trigger

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 68 (Ref:1z0-147e.9.11.1)

Which trigger timing can be used when creating a trigger on a view which does not allow DML?

- ○ AFTER only
- ○ BEFORE only
- ○ INSTEAD OF only
- ○ BEFORE and AFTER only
- ○ BEFORE, AFTER, and INSTEAD OF

Answer:
INSTEAD OF only

Explanation:
INSTEAD OF triggers provide a transparent way of modifying views that are not inherently modifiable directly through DML statements.

BEFORE and AFTER triggers execute before and after triggering DML events. If a view does not allow DML, these trigger types will not fire. An INSTEAD OF trigger must be used.

Objective:
Creating Database Triggers

Sub-Objective:
Create an INSTEAD of trigger

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 69 (Ref:1z0-147e.9.10.1)

For which trigger timing can you reference the NEW and OLD qualifiers?

- ○ row only
- ○ statement only
- ○ statement and row
- ○ Oracle Forms trigger

Answer:
row only

Explanation:
The OLD and NEW qualifiers allow you to reference the value of a column before the trigger event (OLD) and the value after the trigger event (NEW).

Only row level triggers can reference OLD and NEW qualifiers.

Objective:
Creating Database Triggers

Sub-Objective:
Use the OLD and NEW qualifiers in a database trigger

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 74 (Ref:1z0-147e.9.8.1)

Examine this trigger:

```
CREATE OR REPLACE TRIGGER budget_trig
AFTER INSERT ON studio
FOR EACH ROW
DECLARE
v_sum NUMBER;
BEGIN
SELECT sum(yearly_budget)
INTO v_sum
FROM studio;

UPDATE parent_company
SET overall_budget = v_sum;
END;
```

You insert a row into the STUDIO table and receive this message:

```
ORA-04091: table SCOTT.STUDIO is mutating, trigger/function may not see it
```

How do you correct this error?

- ○ Change the timing to BEFORE.
- ○ Remove the AFTER INSERT ON clause.
- ○ Remove the SUM function; it is not allowed in database triggers.
- ○ Convert it to a statement level database trigger by removing FOR EACH ROW.

Answer:
Convert it to a statement level database trigger by removing FOR EACH ROW.

Explanation:
If a row level database trigger attempts to read the same table that the triggering event is on, the table is considered to be mutating. It cannot get a read-consistent query during the middle of the triggering event.

Statement level database triggers do not have this restriction because they execute either right before the event or right after the event.

In this case, the database trigger is a row level trigger. It executes when an INSERT occurs on the STUDIO table. During the event, it is trying to read the STUDIO table. This creates the mutation problem and the trigger fails, causing the INSERT statement to fail.

Objective:
Creating Database Triggers

Sub-Objective:
Create a row level trigger

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 91 (Ref:1z0-147e.9.6.1)

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER update_show_gross
{additional trigger information}
BEGIN
{additional code}
END;
```

This trigger should execute for each row when the SEATS_SOLD or COST_PER_TICKET columns are updated and when a row is inserted into the GROSS_RECEIPT table. Which trigger information must you add?

- ○ BEFORE (gross_receipt, seats_sold, cost_per_ticket) INSERT, UPDATE
- ○ BEFORE INSERT OR UPDATE OF seats_sold, cost_per_ticket ON gross_receipt
- ○ BEFORE INSERT OR UPDATE(seats_sold, cost_per_ticket) ON gross_receipt
  FOR EVERY ROW
- ○ BEFORE INSERT OR UPDATE OF seats_sold, cost_per_ticket ON gross_receipt
  FOR EACH ROW

Answer:
BEFORE INSERT OR UPDATE OF seats_sold, cost_per_ticket ON gross_receiptFOR EACH ROW

Explanation:

Trigger timing is specified using BEFORE or AFTER. If the trigger body should execute before the event, use BEFORE. If the trigger body should execute after the event, use AFTER.

Trigger events include INSERT, UPDATE, and DELETE. If the event is an UPDATE of the triggering table, use UPDATE ON <table name>. If the event is an update of a particular column or columns, use UPDATE OF <column name, column name> ON <table_name>.

For example:
AFTER UPDATE ON EMP
AFTER UPDATE OF SAL ON EMP

Objective:

Creating Database Triggers

Sub-Objective:

Describe the trigger firing sequence options

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 97 (Ref:1z0-147e.9.14.2)

Each month a SQL*Loader application is executed to insert approximately 1 million rows into the GROSS_RECEIPT table. This table has three database triggers that execute for each row inserted. Which command can you issue immediately before the SQL*Loader operation to improve performance?

- ○ DISABLE TRIGGERS ON gross_receipt;
- ○ DISABLE ALL TRIGGERS ON gross_receipt;
- ○ ALTER TRIGGER ALL DISABLE ON gross_receipt;
- ○ ALTER TABLE gross_receipt DISABLE ALL TRIGGERS;

Answer:
ALTER TABLE gross_receipt DISABLE ALL TRIGGERS;

Explanation:

There are two ways to prevent a database trigger from executing. It can be disabled or dropped. Dropping a database trigger, as with any object, is a permanent action. It is removed from the data dictionary and cannot be rolled back. Disabling a database trigger turns off execution. It still exists in the data dictionary but it will not execute until it is enabled.

To prevent a particular trigger from executing temporarily, you should disable it using the ALTER TRIGGER command. Use the ALTER TABLE command to disable all triggers on a particular table.

Objective:

Creating Database Triggers

Sub-Objective:

Alter a trigger status

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 98 (Ref:1z0-147e.9.14.3)

The CHECK_THEATER trigger of the THEATER table has been disabled. Which command can you issue to enable this trigger?

- ○ ENABLE check_theater;
- ○ ENABLE TRIGGER check_theater;
- ○ ALTER TRIGGER check_theater ENABLE;
- ○ ALTER TABLE check_theater ENABLE check_theater;

Answer:
ALTER TRIGGER check_theater ENABLE;

Explanation:

There are two ways to prevent a database trigger from executing. It can be disabled or dropped. Dropping a database trigger, as with any object, is a permanent action. It is removed from the data dictionary and cannot be rolled back. Disabling a database trigger turns off execution. It still exists in the data dictionary, but it will not execute until it is enabled.

To temporarily prevent a particular trigger from executing, you should disable it using the ALTER TRIGGER command. Use the ALTER TABLE command to disable all triggers on a particular table.

Objective:

Creating Database Triggers

Sub-Objective:

Alter a trigger status

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 99 (Ref:1z0-147e.9.15.1)

The AUDIT_THEATER trigger on the THEATER table is no longer needed and must be removed. Which command will successfully remove this trigger from the database?

- ○ DROP audit_theater;
- ○ DROP TRIGGER audit_theater;
- ○ ALTER TRIGGER audit_theater DROP;
- ○ ALTER TABLE theater DROP TRIGGER audit_theater;

Answer:
DROP TRIGGER audit_theater;

Explanation:

There are two ways to prevent a database trigger from executing. It can be disabled or dropped. Dropping a database trigger, as with any object, is a permanent action. It is removed from the data dictionary with the DROP TRIGGER command. Disabling a database trigger turns off execution. It still exists in the data dictionary, but will not execute until it is enabled.

Objective:

Creating Database Triggers

Sub-Objective:

Remove a trigger

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 100 (Ref:1z0-147e.9.14.1)

Due to a disk failure, the AUDIT_THEATER table is unavailable until further notice. The CHECK_THEATER database trigger references this table when a DML operation is performed on the THEATER table. Which command should you issue to prevent this database trigger from executing until this problem is resolved?

- ○ ALTER TRIGGER check_theater DROP;
- ○ MODIFY TRIGGER check_theater DROP;
- ○ ALTER TRIGGER check_theater DISABLE;
- ○ MODIFY TRIGGER check_theater DISABLE;

Answer:
ALTER TRIGGER check_theater DISABLE;

Explanation:

There are two ways to prevent a database trigger from executing. It can be disabled or dropped. Dropping a database trigger, as with any object, is a permanent action. It is removed from the data dictionary and cannot be rolled back. Disabling a database trigger turns off execution. It still exists in the data dictionary, but it will not execute until it is enabled.

If you only need to prevent a trigger from executing temporarily, you should disable it using the ALTER TRIGGER command.

Objective:

Creating Database Triggers

Sub-Objective:

Alter a trigger status

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 108 (Ref:1z0-147e.9.7.1)

You created a database trigger that will be executed for all data manipulation statements on the THEATER table. Within the code, you will determine which type of manipulation has caused the trigger to execute. Which would you use to test for the type of manipulation being performed?

- ○ DBMS_SQL
- ○ DBMS_TRIGGER
- ○ DBMS_CHECK_MANIPULATION_TYPE
- ○ DELETING, UPDATING, and INSERTING

Answer:
DELETING, UPDATING, and INSERTING

Explanation:
If a trigger is created for multiple events, the body can determine the event that invoked the trigger by referencing trigger predicates (functions).

INSERTING
UPDATING [('column name')]
DELETING

Each returns a BOOLEAN value and therefore, are used frequently in IF statements.

Objective:
Creating Database Triggers

Sub-Objective:
Use conditional predicates in a DML trigger

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 151 (Ref:1z0-147e.9.1.1)

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER prevent_gross_modification
{additional trigger code}
BEGIN
IF TO_CHAR(sysdate,'DY') = 'MON' THEN
RAISE_APPLICATION_ERROR(-20000, 'Gross receipts cannot be entered on Monday');
END IF;
END;
```

This trigger must fire before each DELETE, INSERT, and UPDATE of the GROSS_RECEIPT table. It should fire only once for the entire data manipulation statement. Which additional trigger code must you add?

- ◯ BEFORE (gross_receipt) DELETE, INSERT, UPDATE
- ◯ AFTER DELETE OR INSERT OR UPDATE ON gross_receipt
- ◯ BEFORE DELETE OR INSERT OR UPDATE ON gross_receipt
- ◯ BEFORE DELETE OR INSERT OR UPDATE ON gross_receipt
  FOR EACH ROW

Answer:
BEFORE DELETE OR INSERT OR UPDATE ON gross_receipt

Explanation:
Trigger timing is specified using the BEFORE or AFTER keywords. This indicates when the trigger should fire in relation to the triggering event. The triggering event is a data manipulation command.

Use the FOR EACH ROW clause if you want the trigger to execute for each row affected by the data manipulation command.

In this example, the FOR EACH ROW clause is not used because we want the trigger to only execute once for the entire event.

Objective:
Creating Database Triggers

Sub-Objective:
Describe the different types of triggers

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Item: 163 (Ref:1z0-147e.9.1.2)

You created a database trigger that will be executed for all data manipulation statements on the THEATER table. Within the code, you will determine which type of manipulation has caused the trigger to execute. Which would you use to test for the type of manipulation being performed?

- ○ DBMS_SQL
- ○ DBMS_TRIGGER
- ○ DBMS_CHECK_MANIPULATION_TYPE
- ○ DELETING, UPDATING, and INSERTING

Answer:
DELETING, UPDATING, and INSERTING

Explanation:
If a trigger is created for multiple events, the body can determine the event that invoked the trigger by referencing trigger predicates (functions).

INSERTING
UPDATING [('column name')]
DELETING

Each returns a BOOLEAN value and therefore, are used frequently in IF statements.

Objective:
Creating Database Triggers

Sub-Objective:
Describe the different types of triggers

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Database Triggers

Creating Functions

Item: 6 (Ref:1z0-147e.3.3.4)

Examine this function:

```
CREATE OR REPLACE FUNCTION set_budget
(v_studio_id IN NUMBER, v_new_budget IN NUMBER)
RETURN BOOLEAN
IS
BEGIN
UPDATE studio
SET yearly_budget = v_new_budget
WHERE id = v_studio_id;
IF SQL%FOUND THEN
RETURN TRUE;
ELSE
RETURN FALSE;
END IF;

COMMIT;
END;
```

Which code will successfully invoke this function?

-
  ```
  SELECT id, set_budget(11,500000000)
  FROM studio;
  ```

-
  ```
  DECLARE
  v_updated_flag BOOLEAN;
  BEGIN
  set_budget(11,500000000);
  END;
  ```

-
  ```
  VARIABLE g_updated_flag BOOLEAN
  BEGIN
  g_updated_flag := set_budget(11,500000000);
  END;
  ```

-
  ```
  DECLARE
  v_updated_flag BOOLEAN;
  BEGIN
  v_updated_flag := set_budget(11,500000000);
  END;
  ```

Answer:
DECLARE v_updated_flag BOOLEAN;BEGIN v_updated_flag := set_budget(11,500000000);END;

Explanation:
Functions can be invoked in SQL statements unless they return a non-server data type, such as BOOLEAN.

Functions must be executed as part of an expression in PL/SQL. They cannot be invoked like a procedure.

```
DECLARE
v_updated_flag BOOLEAN;
BEGIN
v_updated_flag := set_budget(11,500000000);
END;
```

Objective:
Creating Functions

Sub-Objective:
List how a function can be invoked

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 24 (Ref:1z0-147e.3.3.2)

Examine this function:

```
CREATE OR REPLACE FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;

RETURN v_yearly_budget;
END;
```

Which set of statements will successfully invoke this function within SQL*Plus?

- ◯ VARIABLE g_yearly_budget NUMBER
  :g_yearly_budget := GET_BUDGET(11);
- ◯ VARIABLE g_yearly_budget NUMBER
  EXECUTE g_yearly_budget := GET_BUDGET(11);
- ◯ VARIABLE g_yearly_budget NUMBER
  EXECUTE :g_yearly_budget := GET_BUDGET(11);
- ◯ VARIABLE :g_yearly_budget NUMBER
  EXECUTE :g_yearly_budget := GET_BUDGET(11);

Answer:
VARIABLE g_yearly_budget NUMBEREXECUTE :g_yearly_budget := GET_BUDGET(11);

Explanation:
Functions must be invoked as part of an expression that utilizes the return value. In SQL*Plus, use the VARIABLE command to create a host variable and use it when invoking the function.

```
VARIABLE g_yearly_budget NUMBER
EXECUTE :g_yearly_budget := get_budget(11);
```

Notice, the G_YEARLY_BUDGET variable is only prefixed with a colon within the EXECUTE statement because this statement is actually an implicit PL/SQL anonymous block. When referencing host variables within a PL/SQL construct, you must use the colon to distinguish them from local PL/SQL variables.

Objective:
Creating Functions

Sub-Objective:
List how a function can be invoked

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 32 (Ref:1z0-147e.3.3.1)

Examine this function:

```
CREATE OR REPLACE FUNCTION get_budget
(v_studio_id IN NUMBER, v_max_budget IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;

IF v_yearly_budget > v_max_budget THEN
RETURN v_max_budget;
ELSE
RETURN v_yearly_budget;
END IF;

END;
```

Which set of statements will successfully invoke this function within SQL*Plus?

- ○ SELECT id, name, get_budget(id,200)
  FROM studio;
- ○ VARIABLE g_yearly_budget NUMBER
  EXECUTE g_yearly_budget := get_budget(11, 4000000000);
- ○ VARIABLE g_yearly_budget NUMBER
  RUN :g_yearly_budget := get_budget(v_studio_id => 11,v_max_budget => 4000000000);
- ○ VARIABLE g_yearly_budget NUMBER
  EXECUTE g_yearly_budget := get_budget(v_max_budget => 4000000000, v_studio_id => 11);

Answer:
SELECT id, name, get_budget(id,200)FROM studio;

Explanation:

Functions can be invoked in SQL statements unless they return a non-server data type, such as BOOLEAN.

Functions can also be executed as part of an expression in PL/SQL. They cannot be invoked like a procedure.

```
DECLARE
v_updated_flag BOOLEAN;
BEGIN
v_updated_flag := set_budget(11,500000000);
END;
```

Functions can be executed in SQL*Plus using the VARIABLE command.

```
VARIABLE g_yearly_budget NUMBER
EXECUTE :g_yearly_budget := GET_BUDGET(11, 4000000000);
```

G_YEARLY_BUDGET is prefixed with a colon only within the EXECUTE statement because this statement is actually an implicit PL/SQL anonymous block. When referencing host variables within a PL/SQL construct, you must use the colon to distinguish them from local PL/SQL variables.

Objective:

Creating Functions

Sub-Objective:
List how a function can be invoked

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 33 (Ref:1z0-147e.3.3.3)

For which purpose are formal parameters used when creating functions?

- ○ restricting pragma references
- ○ passing values to the function
- ○ bypassing directives to the compiler
- ○ prompting the end user for information

Answer:
passing values to the function

Explanation:
Functions, like procedures, use formal parameters to transfer values to and from the calling environment. Unlike procedures, OUT arguments are not typically used with functions. Information is transferred out using the RETURN statement.

Objective:
Creating Functions

Sub-Objective:
List how a function can be invoked

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 40 (Ref:1z0-147e.3.3.5)

A stored function can be invoked in many different ways. Which invocation example is NOT valid?

- ○ executing the stored function within an INSERT statement
- ○ executing the stored function within a server-side function
- ○ executing the stored function within a client-side function
- ○ executing the stored function within a CHECK constraint of a table

Answer:
executing the stored function within a CHECK constraint of a table

---

Explanation:
Functions cannot be invoked within a CHECK constraint, but can be used in SELECT, INSERT, UPDATE, and DELETE statements. They can also be invoked from within other server-side or client-side functions.

Objective:
Creating Functions

Sub-Objective:
List how a function can be invoked

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 57 (Ref:1z0-147e.3.4.1)

How do functions simplify maintainability?

- ○ by limiting changes to logic to one location
- ○ by eliminating the need for an executable section
- ○ by eliminating the need for the storage of object code
- ○ by ignoring side effects when invoked from within a SQL statement

Answer:
by limiting changes to logic to one location

Explanation:
Functions and procedures improve maintainability by storing a block of logic in one place. Therefore, subsequent changes to the logic occur in one place.

Objective:
Creating Functions

Sub-Objective:
List the advantages of user-defined functions in SQL statements

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 81 (Ref:1z0-147e.3.3.6)

A stored function can be invoked in many different ways. Which invocation example is NOT valid?

- ○ executing the stored function within an INSERT statement
- ○ executing the stored function within a server-side function
- ○ executing the stored function within a client-side function
- ○ executing the stored function within the DEFAULT clause of the CREATE TABLE statement

Answer:
executing the stored function within the DEFAULT clause of the CREATE TABLE statement

---

Explanation:
Functions cannot be invoked within the DEFAULT clause of the CREATE TABLE statement, but can be used in SELECT, INSERT, UPDATE, and DELETE statements. They can also be invoked from within other server-side or client-side functions.

Objective:
Creating Functions

Sub-Objective:
List how a function can be invoked

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 93 (Ref:1z0-147e.3.6.2)

Examine this statement:

```
SELECT id, theater_pck.get_budget(id)
FROM studio;
```

What must be true about the GET_BUDGET function for this statement to be successful?

- ○ It must be remote.
- ○ It must not modify the database.
- ○ It must also exist as a stand-alone function.
- ○ It must only exist in the body of THEATER_PCK.

Answer:
It must not modify the database.

Explanation:
Functions, standalone or part of a package, cannot modify the database if they are invoked from a SQL statement.

Prior to Oracle8i, a packaged function required the PRAGMA RESTRICT_REFERENCES directive in the specification to guarantee that the function did not modify the database.

Objective:
Creating Functions

Sub-Objective:
Describe the restrictions on calling functions from SQL statements

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 102 (Ref:1z0-147e.3.2.4)

Examine this function:

```
CREATE OR REPLACE FUNCTION get_budget
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END;
```

What additional code is needed to compile this function successfully?

- ○ Remove the first RETURN statement.
- ○ Change "RETURN number" to "OUTPUT number".
- ○ Add "(v_studio_id IN NUMBER)" right after the IS keyword.
- ○ Add "(v_studio_id IN NUMBER)" right before the RETURN statement of the header.

Answer:
Add "(v_studio_id IN NUMBER)" right before the RETURN statement of the header.

Explanation:
Arguments must be declared in the header section and precede the RETURN statement. The header section is declared after the function name and before the IS keyword.

The function is missing the argument declaration.

```
CREATE OR REPLACE FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
```

Objective:
Creating Functions

Sub-Objective:
Create a function

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 110 (Ref:1z0-147e.3.8.2)

Which code successfully calculates commission returning it to the calling environment?

-
  ```
  CREATE OR REPLACE FUNCTION calc_comm
  (v_emp_id IN NUMBER)
  RETURN number
  IS
  v_total NUMBER;
  BEGIN
  SELECT SUM(ord.total)
  INTO v_total
  FROM ord,customer
  WHERE ord.custid = customer.custid
  AND customer.repid = v_emp_id;
  END;
  ```

-
  ```
  CREATE OR REPLACE FUNCTION calc_comm
  (v_emp_id IN NUMBER)
  IS
  v_total NUMBER;
  BEGIN
  SELECT SUM(ord.total)
  INTO v_total
  FROM ord,customer
  WHERE ord.custid = customer.custid
  AND customer.repid = v_emp_id;
  RETURN (v_total * .20);
  END;
  ```

-
  ```
  CREATE OR REPLACE FUNCTION calc_comm
  (v_emp_id IN NUMBER)
  IS
  RETURN number
  v_total NUMBER;
  BEGIN
  SELECT SUM(ord.total)
  INTO v_total
  FROM ord,customer
  WHERE ord.custid = customer.custid
  AND customer.repid = v_emp_id;
  RETURN (v_total * .20);
  END;
  ```

-
  ```
  CREATE OR REPLACE FUNCTION calc_comm
  (v_emp_id IN NUMBER)
  RETURN number
  IS
  v_total NUMBER;
  BEGIN
  SELECT SUM(ord.total)
  INTO v_total
  FROM ord,customer
  WHERE ord.custid = customer.custid
  AND customer.repid = v_emp_id;
  RETURN (v_total * .20);
  END;
  ```

Answer:

CREATE OR REPLACE FUNCTION calc_comm (v_emp_id IN NUMBER) RETURN numberIS v_total NUMBER;BEGIN SELECT SUM(ord.total) INTO v_total FROM ord,customer WHERE ord.custid = customer.custid AND customer.repid = v_emp_id; RETURN (v_total * .20);END;

Explanation:

Functions must contain a RETURN statement in the header to specify the data type of the value to be returned. Although a function will compile successfully without a RETURN statement in the executable section, it will generate a run-time error if no value is returned to the calling environment when executed.

The header information includes the function name, arguments, and the RETURN statement. It must come before the IS keyword.

Objective:

Creating Functions

Sub-Objective:

Describe the differences between procedures and functions

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 116 (Ref:1z0-147e.3.5.1)

You have just created a PL/SQL user-defined function called CALC_COMM. Which statement will successfully test it?

- EXECUTE calc_comm;
- SELECT *
  FROM calc_comm(emp);
- EXECUTE calc_comm(SELECT total FROM ord);
- SELECT *
  FROM ord
  GROUP BY ordid
  HAVING calc_comm(total) > 5000;

Answer:

SELECT *FROM ordGROUP BY ordidHAVING calc_comm(total) > 5000;

Explanation:
Stored user-defined functions can be invoked as part of a PL/SQL expression or SQL statement. However, they cannot be used in the FROM clause.

You cannot use EXECUTE to invoke a function unless you have a variable to hold the returning value.
For example:

```
SQL> VARIABLE g_return_value NUMBER
SQL> EXECUTE :g_return_value := calc_comm(4000);
```

Objective:
Creating Functions

Sub-Objective:
List where user-defined functions can be called from within an SQL statement

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 118 (Ref:1z0-147e.3.1.1)

Which subprogram type can be invoked from within a SQL statement?

- ○ function
- ○ procedure
- ○ public packaged procedure
- ○ private packaged procedure
- ○ private packaged function

Answer:
function

Explanation:

Only functions can be invoked from within a SQL statement. They can be either stand-alone or packaged as a public function.

Objective:

Creating Functions

Sub-Objective:

Define what a stored function is

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 123 (Ref:1z0-147e.3.2.1)

Procedures and functions can be created and stored in the database or in an Oracle Developer application. How is performance improved when storing procedures and functions in the database?

- ○ Network roundtrips are reduced.
- ○ The object code is created during execution.
- ○ Network traffic is decreased by bundling commands.
- ○ The source code is stored externally, and the object code is stored internally.

Answer:
Network roundtrips are reduced.

Explanation:

The source and object code of database stored procedures and functions are stored like other objects, in physical files assigned to the appropriate tablespace. When executed from a client application, they require only one call. Since the object code is stored on the database side, there is no need to send it across the network.

Executing procedures and functions stored in an Oracle Developer application will process each PL/SQL statement and pass each SQL statement across the network to the database to be processed, dramatically increasing network roundtrips.

Objective:

Creating Functions

Sub-Objective:

Create a function

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 126 (Ref:1z0-147e.3.6.1)

Procedures and functions are very similar. For which reason would you choose a function over a procedure?

- ○ A function allows Oracle group functions.
- ○ A function can be used in a SQL statement.
- ○ A function guarantees read consistency, while a procedure cannot.
- ○ A function can only be executed from a previously created procedure.

Answer:
A function can be used in a SQL statement.

Explanation:
If the stored function returns an Oracle server internal data type and does not modify database tables, it can be invoked from within a SQL statement.

Objective:
Creating Functions

Sub-Objective:
Describe the restrictions on calling functions from SQL statements

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 139 (Ref:1z0-147e.3.8.3)

Which two subprogram headers are correct? (Choose two.)

- ☐ CREATE OR REPLACE PROCEDURE get_sal
  IS
  (v_sal IN number)
- ☐ CREATE OR REPLACE PROCEDURE get_sal
  (v_sal IN number)
  IS
- ☐ CREATE OR REPLACE FUNCTION calc_comm
  RETURN number
  (p_amnt IN number)
- ☐ CREATE OR REPLACE FUNCTION calc_comm
  (p_amnt IN number)
  RETURN number
- ☐ CREATE OR REPLACE FUNCTION calc_comm
  (p_amnt IN number(3,2))
  RETURN number

Answer:
CREATE OR REPLACE PROCEDURE get_sal(v_sal IN number)IS
CREATE OR REPLACE FUNCTION calc_comm(p_amnt IN number)RETURN number

---

Explanation:
When creating functions, the argument list must be defined prior to the RETURN statement.

```
CREATE OR REPLACE FUNCTION calc_comm
(p_amnt IN NUMBER)
RETURN number
...
```

When creating procedures, the argument list must be defined prior to the IS keyword.

```
CREATE OR REPLACE PROCEDURE get_sal
(v_sal IN NUMBER)
IS
...
```

Both procedures and functions define local variables after the IS keyword.

Objective:
Creating Functions

Sub-Objective:
Describe the differences between procedures and functions

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 164 (Ref:1z0-147e.3.7.1)

The GET_BUDGET function is no longer needed and should be removed. Which command will successfully remove this function from the database?

- ○ DROP get_budget;
- ○ REMOVE get_budget;
- ○ DROP FUNCTION get_budget;
- ○ ALTER get_budget DROP FUNCTION;

Answer:
DROP FUNCTION get_budget;

Explanation:
Use the DROP FUNCTION command to drop a function from the database. If successful, both the object code and the source code will be deleted from the data dictionary.

Objective:
Creating Functions

Sub-Objective:
Remove a function

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 165 (Ref:1z0-147e.3.2.3)

Examine this function:

```
CREATE OR REPLACE FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
END;
```

To execute this function successfully, what additional code must be added to the executable section?

- ○ RETURN;
- ○ RETURN get_budget;
- ○ OUT v_yearly_budget;
- ○ RETURN v_yearly_budget;

Answer:
RETURN v_yearly_budget;

Explanation:

Functions must contain two RETURN statements. One RETURN statement must exist in the header section to specify the data type to be returned. Another RETURN statement must exist in the executable section to return the value.

This function is missing the RETURN statement required in the executable section.

RETURN v_yearly_budget;

Objective:

Creating Functions

Sub-Objective:

Create a function

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 168 (Ref:1z0-147e.3.2.2)

When creating a function in SQL*Plus, you receive an error message stating that the function created with compilation errors.

What must you do to see the compilation errors?

- ○ Issue the SHOW ERRORS command.
- ○ Issue the DISPLAY ERRORS command.
- ○ Query the FUNCTION_ERRORS data dictionary view.
- ○ Do nothing. The errors will display after a five-second delay.

Answer:
Issue the SHOW ERRORS command.

Explanation:
Procedure or function compilation errors can be viewed in SQL*Plus by issuing the SHOW ERRORS command or by querying the USER_ERRORS data dictionary view.

Objective:
Creating Functions

Sub-Objective:
Create a function

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 169 (Ref:1z0-147e.3.8.1)

Which two statements are true? (Choose two.)

- ☐ A function must return a value.
- ☐ Functions and procedures must return a value.
- ☐ Functions and procedures must contain IN arguments.
- ☐ A function can be invoked from within a PL/SQL expression.
- ☐ A procedure must be invoked from within a PL/SQL expression.

Answer:
A function must return a value.
A function can be invoked from within a PL/SQL expression.

Explanation:
Only functions must return a value and can be invoked as part of a PL/SQL expression. Functions can also be invoked from a SQL statement if it returns an Oracle server internal data type and does not modify database tables.

Neither functions nor procedures require IN arguments.

Objective:
Creating Functions

Sub-Objective:
Describe the differences between procedures and functions

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Item: 174 (Ref:1z0-147e.3.2.5)

Examine this function:

```
CREATE OR REPLACE FUNCTION set_budget
(v_studio_id IN NUMBER, v_new_budget IN NUMBER)
IS
BEGIN
UPDATE studio
SET yearly_budget = v_new_budget
WHERE id = v_studio_id;

IF SQL%FOUND THEN
RETURN TRUE;
ELSE
RETURN FALSE;
END IF;

COMMIT;
END;
```

Which code must be added to successfully compile this function?

- Add "RETURN;" immediately before the IS keyword.
- Add "RETURN NUMBER" immediately before the IS keyword.
- Add "RETURN BOOLEAN" immediately after the IS keyword.
- Add "RETURN BOOLEAN" immediately before the IS keyword.

Answer:
Add "RETURN BOOLEAN" immediately before the IS keyword.

Explanation:

Functions must contain two RETURN statements. One RETURN statement must exist in the header section to specify the data type to be returned. Another RETURN statement must exist in the executable section to return the value.

This function is missing the RETURN statement in the header section. It must be listed immediately before the IS keyword.

```
CREATE OR REPLACE FUNCTION set_budget
(v_studio_id IN NUMBER, v_new_budget IN NUMBER)
RETURN BOOLEAN
IS
BEGIN
...
END;
```

Objective:
Creating Functions

Sub-Objective:
Create a function

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Functions

Creating Procedures

Item: 5 (Ref:1z0-147e.2.3.10)

Examine this procedure:

```
CREATE PROCEDURE add_theater
IS
BEGIN
INSERT INTO theater
VALUES (35, 'Riverplace Theatre', '1222 River Drive, Austin, Tx.');
END;
```

This procedure already exists in the database. You have made a change to the code and want to recreate the procedure. Which command must you now use to prevent an error?

- ○ REPLACE PROCEDURE
- ○ RECREATE PROCEDURE
- ○ UPDATE PROCEDURE WITH
- ○ CREATE OR REPLACE PROCEDURE

Answer:
CREATE OR REPLACE PROCEDURE

Explanation:
If the procedure already exists in the database, you must use the REPLACE keyword.

A script file containing the procedure code using the CREATE OR REPLACE clause for subsequent code changes should be created.

Objective:
Creating Procedures

Sub-Objective:
Create a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 19 (Ref:1z0-147e.2.3.9)

When creating the ADD_PROD procedure in SQL*Plus, you receive this message:

Warning: Procedure created with compilation errors.

What was saved to the data dictionary?

- ○ source code only
- ○ compilation errors only
- ○ source code and compilation errors
- ○ nothing

Answer:
source code and compilation errors

Explanation:
During creation of a procedure, the source code is stored in the data dictionary and can be accessed by querying the USER_SOURCE or ALL_SOURCE data dictionary views.

Compilation errors are also stored in the data dictionary and can be viewed by typing SHOW ERRORS in SQL*Plus or by querying the USER_ERRORS view.

Objective:
Creating Procedures

Sub-Objective:
Create a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 26 (Ref:1z0-147e.2.3.3)

Evaluate this procedure:

```
CREATE OR REPLACE PROCEDURE remove_department
(v_deptno IN NUMBER(9))
IS
BEGIN
DELETE dept
WHERE deptno = v_deptno;
END;
```

Why does this statement fail when compiled?

- ○ The keyword AS is missing.
- ○ Specifying a precision for a formal parameter is not permitted.
- ○ V_DEPTNO should be declared as IN OUT.
- ○ The DELETE statement has a syntax error.

Answer:
Specifying a precision for a formal parameter is not permitted.

Explanation:

When declaring a formal argument, only the data type should be specified when declaring an argument. The precision for a formal argument is not allowed.

The option stating the argument should be declared as IN OUT is incorrect because the procedure does not pass a value back to the calling environment. IN is the default type of parameter and passes a constant value from the calling environment to the procedure. The option stating the DELETE statement has a syntax error is incorrect because the statement uses the correct DELETE statement syntax. The option stating that the AS keyword is incorrect because is not required.

Objective:

Creating Procedures

Sub-Objective:

Create a procedure

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 38 (Ref:1z0-147e.2.5.2)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE find_seats_sold
(v_movie_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id;
END;
```

The value of V_SEATS_SOLD must be returned to the calling environment. Which change should you make to the code?

- ○ Declare V_SEATS_SOLD as an OUT argument.
- ○ Declare V_SEATS_SOLD as a RETURN argument.
- ○ Add RETURN V_SEATS_SOLD immediately before the IS keyword.
- ○ Add RETURN V_SEATS_SOLD immediately before the END keyword.

Answer:
Declare V_SEATS_SOLD as an OUT argument.

Explanation:

Procedures can return values to the calling environment using OUT arguments. Arguments are declared after the procedure name and before the IS keyword.

The procedure after adding the OUT argument:

```
CREATE OR REPLACE PROCEDURE find_seats_sold
(v_movie_id IN NUMBER, v_seats_sold OUT NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE MOVIE_ID = v_movie_id;
END;
```

Objective:

Creating Procedures

Sub-Objective:

List the types of parameter modes

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 39 (Ref:1z0-147e.2.3.4)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE remove_department
(v_deptno IN NUMBER)
IS
BEGIN
DELETE dept
WHERE deptno = v_deptno;
END;
```

After executing this procedure, you receive this message:

```
ORA-02292: integrity constraint (SCOTT.FK_DEPTNO) violated - child record found
```

What must be added to the procedure to handle this error?

- ○ Create an EXCEPTION section, and add code to handle this PL/SQL predefined exception.
- ○ Add an IF statement immediately after the DELETE statement to check the value of SQL%NOTFOUND.
- ○ Nothing should be added to this procedure. It must only be executed using valid department numbers.
- ○ Declare a new exception and associate it with error code -2292. Create an exception section, and add code to handle this non-predefined exception that you just declared.

Answer:

Declare a new exception and associate it with error code -2292. Create an exception section, and add code to handle this non-predefined exception that you just declared.

Explanation:

To handle this error successfully, an exception must be declared and associated with the error code. This procedure should be recreated as follows:

```
CREATE OR REPLACE PROCEDURE remove_department
(v_deptno IN NUMBER)
IS
fk_recs_exist EXCEPTION;
PRAGMA EXCEPTION_INIT (fk_recs_exist, -2292);
BEGIN
DELETE dept
WHERE deptno = v_deptno;
EXCEPTION
WHEN fk_recs_exist THEN
DBMS_OUTPUT.PUT_LINE('Cannot delete this department');
END;
```

The execution of this code results in the following:

```
SQL> execute remove_department (10)
Cannot delete this department

PL/SQL procedure successfully completed.
```

Objective:

Creating Procedures

Sub-Objective:

Create a procedure

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 42 (Ref:1z0-147e.2.3.1)

When declaring arguments within a procedure, which specification is NOT allowed?

- ○ %TYPE
- ○ data type
- ○ %ROWTYPE
- ○ maximum length

Answer:
maximum length

Explanation:
Only the data type is specified when declaring an argument. Maximum length is not allowed.

You can use %TYPE for scalar arguments and %ROWTYPE for record arguments.

Objective:
Creating Procedures

Sub-Objective:
Create a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 48 (Ref:1z0-147e.2.3.2)

Which statement about declaring parameters is true?

- ○ Only data type is required.
- ○ Data type and maximum length are required.
- ○ Data type and maximum length are not required.
- ○ Only maximum length is required for the VARCHAR2 data type.

Answer:
Only data type is required.

Explanation:
When declaring arguments, you can only specify the data type. Maximum length is not allowed.

Example:

```
CREATE OR REPLACE PROCEDURE remove_department
(v_deptno IN NUMBER)
IS
BEGIN
DELETE dept
WHERE deptno = v_deptno;
END;
```

Objective:
Creating Procedures

Sub-Objective:
Create a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 56 (Ref:1z0-147e.2.8.2)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE find_cpt
(v_movie_id {argument mode} NUMBER, v_cost_per_ticket {argument mode} NUMBER)
IS
BEGIN
IF v_cost_per_ticket > 8.50 THEN
SELECT cost_per_ticket
INTO v_cost_per_ticket
FROM gross_receipt
WHERE movie_id = v_movie_id;
END IF;
END;
```

Which mode should be used for V_COST_PER_TICKET?

- ○ IN
- ○ OUT
- ○ RETURN
- ○ IN OUT

Answer:
IN OUT

Explanation:
This argument must be declared as IN OUT. Only IN OUT arguments can be both read and modified. IN arguments are read only and OUT arguments are write only.

The V_COST_PER_TICKET argument is read using the IF statement and modified using the SELECT INTO statement.

Objective:
Creating Procedures

Sub-Objective:
Create a procedure with parameters

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 61 (Ref:1z0-147e.2.8.3)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE find_cpt
(v_movie_id {argument mode} NUMBER, v_cost_per_ticket {argument mode} NUMBER)
IS
BEGIN
IF v_cost_per_ticket > 8.50 THEN
SELECT cost_per_ticket
INTO v_cost_per_ticket
FROM gross_receipt
WHERE movie_id = v_movie_id;
END IF;
END;
```

Which argument mode should be used for V_MOVIE_ID?

- ○ IN
- ○ OUT
- ○ IN OUT
- ○ IN RETURN

Answer:
IN

Explanation:
The value of V_MOVIE_ID is used in the WHERE clause to determine which row to return. Since it is only being read and not modified, it should be declared as an IN argument.

Objective:
Creating Procedures

Sub-Objective:
Create a procedure with parameters

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 65 (Ref:1z0-147e.2.7.2)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE FIND_ORDERS
(v_total IN sales_order.total%TYPE)
IS
CURSOR c1 IS SELECT order_id
FROM sales_order
WHERE total > v_total;
BEGIN
FOR sales_order_rec in c1 LOOP
--process the row
END LOOP;
END;
```

This procedure returns all orders with a total greater than an amount that is passed in the V_TOTAL parameter. Occasionally, a user might want to process all orders regardless of the total amount. They could do this by passing 0 in the V_TOTAL parameter, however, they would prefer not to pass anything. Which change can you make to the procedure to allow a user to process all orders in the SALES_ORDER table without having to pass a 0 total amount?

- Remove only the V_TOTAL parameter.
- Remove the NVL function from the WHERE clause.
- Use (v_total IN sales_order.total%TYPE => 0) as the parameter definition.
- Use (v_total IN sales_order.total%TYPE DEFAULT 0) as the parameter definition.

Answer:
Use (v_total IN sales_order.total%TYPE DEFAULT 0) as the parameter definition.

Explanation:

Using the DEFAULT keyword when defining parameters enables the parameter to be assigned a value if one is not given during invocation.

Using (v_total IN sales_order.total%TYPE DEFAULT 0) as the parameter definition allows a user to execute the procedure without specifying a total amount.

Example:

```
EXECUTE find_orders;
```

Because a value was not specified during invocation, the DEFAULT of 0 will be assigned to the V_TOTAL parameter. All orders with a total amount greater than 0 will be processed.

Objective:

Creating Procedures

Sub-Objective:

Describe the DEFAULT option for parameters

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 71 (Ref:1z0-147e.2.11.1)

Which statement about error propagation is true?

- ○ An exception can propagate across remote procedure calls.
- ○ The RAISE; statement in an exception handler ends the current exception.
- ○ An exception raised inside an exception handler will not propagate to the enclosing block.
- ○ When an exception is raised in a called procedure, control goes to the exception section of that block.

Answer:
When an exception is raised in a called procedure, control goes to the exception section of that block.

Explanation:
When an exception is raised in a procedure, the control goes to the exception section of that block. If the exception is handled, the block terminates and control is returned to the calling program.

The option stating that an exception can propagate across remote procedure calls is incorrect because an exception cannot propagate across remote procedure calls. The option stating that the RAISE; statement in an exception handler ends the current exception is incorrect because it raises an exception in the in a called procedure and is used in the executable section of a procedure.

Objective:
Creating Procedures

Sub-Objective:
Describe how exceptions are propagated

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 77 (Ref:1z0-147e.2.8.5)

Examine this procedure:

```
CREATE PROCEDURE update_theater
IS
BEGIN
UPDATE theater
SET name = v_name
WHERE id = 34;
END;
```

Because a value for the new theater name must be passed to this procedure upon invocation, you decide to create a parameter called V_NAME to hold the value. To be successful, which additional change must you make to this procedure?

- ○ Add (v_name IN VARCHAR2) immediately after the IS keyword.
- ○ Add v_name VARCHAR2(30); immediately after the IS keyword.
- ○ Add (v_name IN VARCHAR2) immediately before the IS keyword.
- ○ Add (v_name IN VARCHAR2) immediately after the BEGIN keyword.

Answer:
Add (v_name IN VARCHAR2) immediately before the IS keyword.

Explanation:
Passing a value to a procedure during invocation requires an IN argument. Arguments are declared after the procedure name and before the IS keyword.

The procedure after adding the IN argument:

```
CREATE PROCEDURE update_theater
(v_name IN VARCHAR2)
IS
BEGIN
UPDATE theater
SET name = v_name
WHERE id = 34;
END;
```

Objective:
Creating Procedures

Sub-Objective:
Create a procedure with parameters

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 87 (Ref:1z0-147e.2.12.1)

The CALC_COMM procedure is no longer needed and should be removed. Which command will successfully remove this procedure from the database?

- ○ DROP calc_comm;
- ○ REMOVE calc_comm;
- ○ DROP PROCEDURE calc_comm;
- ○ ALTER calc_comm DROP PROCEDURE;

Answer:
DROP PROCEDURE calc_comm;

Explanation:
Use the DROP PROCEDURE command to drop a procedure from the database.

Objective:
Creating Procedures

Sub-Objective:
Remove a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 88 (Ref:1z0-147e.2.12.2)

You have just successfully dropped the CALC_COMM procedure and deleted the script file containing the source code. Which command can you execute to recover this procedure?

- ○ ROLLBACK;
- ○ ROLLBACK TO PROCEDURE calc_comm;
- ○ ALTER PROCEDURE calc_comm COMPILE;
- ○ Only the database administrator can recover this procedure using backups.

Answer:
Only the database administrator can recover this procedure using backups.

Explanation:
The DROP PROCEDURE command is a DDL command and is, therefore, auto-committing. A committed transaction cannot be rolled back. Without a script file containing the source code, only the DBA can recover this procedure.

Objective:
Creating Procedures

Sub-Objective:
Remove a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 90 (Ref:1z0-147e.2.7.3)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE find_cpt
(v_movie_id IN NUMBER, v_cost_per_ticket IN OUT NUMBER DEFAULT 0)
IS
BEGIN
IF v_cost_per_ticket > 8.50 THEN
SELECT cost_per_ticket
INTO v_cost_per_ticket
FROM gross_receipt
WHERE movie_id = v_movie_id;
END IF;
END;
```

Why does this statement fail when executed?

- ○ MOVIE_ID is not declared.
- ○ The precision of V_MOVIE_ID was not specified.
- ○ V_COST_PER_TICKET should be specified as OUT.
- ○ The SELECT INTO statement contains a syntax error.
- ○ The declaration of V_COST_PER_TICKET cannot have a DEFAULT value.

Answer:
The declaration of V_COST_PER_TICKET cannot have a DEFAULT value.

Explanation:

IN parameters can be initialized to a default value allowing the default value to be overwritten or accepted. IN OUT and OUT parameters cannot have a default value declared.

The option stating that the precision of V_MOVIE_ID was not specified is incorrect you cannot specify the precision a parameter. The option stating that V_COST_PER_TICKET should be specified as OUT is incorrect because a value needs to be passed from the calling environment to satisfy the programming logic. The option stating that the SELECT INTO statement contains a syntax error is incorrect because the statement does not contain a syntax error.

Objective:

Creating Procedures

Sub-Objective:

Describe the DEFAULT option for parameters

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 103 (Ref:1z0-147e.2.1.1)

Which three statements about procedures are true? (Choose three.)

- ☐ They are typically written in SQL.
- ☐ They add functionality to SQL DML statements.
- ☐ They promote reusability and maintainability.
- ☐ They perform actions and can accept parameters.
- ☐ They add functionality to SQL SELECT statements.
- ☐ They require at least one executable statement in the procedure body.

Answer:
They promote reusability and maintainability.
They perform actions and can accept parameters.
They require at least one executable statement in the procedure body.

Explanation:

Procedures usually contain code that is executed from more than one application. Storing code in one location makes it ideally suitable for reusability and maintainability.

Procedures cannot be used in SQL statements and do not have to return a value.

Objective:

Creating Procedures

Sub-Objective:

Define what a stored procedure is

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 113 (Ref:1z0-147e.2.8.4)

Which statement about formal parameters is true?

- ○ You can assign a default value to an OUT formal parameter.
- ○ An IN OUT formal parameter does not require a value before returning to the calling environment.
- ○ You cannot assign a value to an IN formal parameter inside the procedure in which it is being used.
- ○ You cannot change the value of an OUT formal parameter inside the procedure in which it is being used.
- ○ The OUT or IN OUT formal parameter defines the value used in the executable section of a PL/SQL block.

Answer:
You cannot assign a value to an IN formal parameter inside the procedure in which it is being used.

Explanation:
Formal arguments allow you to transfer values to and from the calling environment. The three procedural parameter modes are IN, OUT, and IN OUT. IN passes a constant value from the calling environment to the procedure. OUT passes a constant value from the procedure to the calling environment. IN OUT passes a value from the calling environment into the procedure and maybe a different value back to the calling environment using the same parameter. Because IN is the default mode, it is not required to specify the IN parameter mode. IN parameters can be initialized to a default value using the DEFAULT option, but this is not required. IN OUT and OUT parameters cannot have a default value declared.

Objective:
Creating Procedures

Sub-Objective:
Create a procedure with parameters

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 124 (Ref:1z0-147e.2.1.3)

Examine this procedure:

```
CREATE PROCEDURE add_theater
IS
BEGIN
INSERT INTO theater
VALUES (35, 'Riverplace Theatre', '1222 River Drive, Austin, Tx.');
END;
```

Which three statements about this procedure are true? (Choose three.)

- ☐ The ADD_THEATER procedure is written in SQL.
- ☐ The ADD_THEATER procedure can be shared by multiple programs.
- ☐ The ADD_THEATER procedure will be stored in the database as a schema object.
- ☐ The ADD_THEATER procedure will execute with the privileges of its owner, by default.
- ☐ The ADD_THEATER procedure has three parts: the specification, the body, and the exception handler.

Answer:
The ADD_THEATER procedure can be shared by multiple programs.
The ADD_THEATER procedure will be stored in the database as a schema object.
The ADD_THEATER procedure will execute with the privileges of its owner, by default.

Explanation:
Procedures execute with the privileges of its owner, by default. This allows indirect access to database objects and allows data security. Users only need to be granted the privilege to execute the procedure. Procedures are stored in the database as a schema object. Because procedures are stored in the database in one location, they can be shared by multiple programs.

The option stating that the ADD_THEATER procedure has three parts is incorrect because this procedure does not have an exception handler. The option stating that the procedure is written in SQL is incorrect because the procedure is written in PL/SQL.

Objective:
Creating Procedures

Sub-Objective:
Define what a stored procedure is

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 130 (Ref:1z0-147e.2.3.8)

Examine this procedure:

```
1. CREATE OR REPLACE PROCEDURE remove_department
2. (v_deptno IN NUMBER)
3. IS
4. BEGIN
5. DELETE dept
6. WHERE deptno = v_deptno;
7. EXCEPTION
8. WHEN DEPT_DELETE_EXCEPTION
9. THEN DBMS_OUTPUT.PUT_LINE ('Cannot delete this department.');
10. END;
```

Which line of this procedure creates an error when compiled?

- ○ 1
- ○ 2
- ○ 5
- ○ 6
- ○ 8

Answer:
8

Explanation:
Line 8 will be flagged by the flagged by the parser because the DEPT_DELETE_EXCEPTION is not defined.

Objective:
Creating Procedures

Sub-Objective:
Create a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 134 (Ref:1z0-147e.2.2.3)

The MODIFY_PAYROLL procedure contains many SQL statements and will be executed from multiple client applications. Where should this procedure be stored?

- ○ server only
- ○ system global area
- ○ client applications only
- ○ server and client applications

Answer:
server only

Explanation:
A procedure that contains multiple SQL statements should be stored on the server to dramatically reduce the amount of network traffic when executed from a client machine.

If the procedure is stored in an Oracle Developer application, each SQL statement must be sent separately to the server to be processed. If the procedure is stored on the server, the application simply executes it with one call.

Objective:
Creating Procedures

Sub-Objective:
List the development steps for creating a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 142 (Ref:1z0-147e.2.2.2)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE update_employee
(v_emp_id IN NUMBER)
IS
v_comm NUMBER;

PROCEDURE calc_comm
IS
v_total NUMBER;
BEGIN
SELECT SUM(ord.total)
INTO v_total
FROM ord,customer
WHERE ord.custid = customer.custid
AND customer.repid = v_emp_id;
v_comm := v_total * .20;
END calc_comm;

v_percentage NUMBER;
BEGIN
SELECT percentage
INTO v_percentage
FROM daily_figures
WHERE TRUNC(figure_date) = TRUNC(SYSDATE);

IF v_percentage > 33 THEN
calc_comm;
END IF;

END;
```

Why does this code cause an error when compiled?

- ○ The SUBPROGRAM keyword does not exist.
- ○ CALC_COMM must be invoked using the EXECUTE command.
- ○ CALC_COMM must be declared after all local variable declarations.
- ○ CALC_COMM must be declared before all local variable declarations.

Answer:
CALC_COMM must be declared after all local variable declarations.

Explanation:

Subprograms must be declared after all local variables.

```
CREATE OR REPLACE PROCEDURE update_employee
(v_emp_id IN NUMBER)
IS
v_comm NUMBER;
v_percentage NUMBER;

PROCEDURE calc_comm
IS
v_total NUMBER;
BEGIN
SELECT SUM(ord.total)
```

```
INTO v_total
FROM ord,customer
WHERE ord.custid = customer.custid
AND customer.repid = v_emp_id;
v_comm := v_total * .20;
END calc_comm;

BEGIN
SELECT percentage
INTO v_percentage
FROM daily_figures
WHERE TRUNC(figure_date) = TRUNC(SYSDATE);

IF v_percentage > 33 THEN
calc_comm;
END IF;

END;
```

Objective:
Creating Procedures

Sub-Objective:
List the development steps for creating a procedure

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 148 (Ref:1z0-147e.2.7.1)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END;
```

Which command will successfully invoke this procedure in SQL*Plus?

- ○ RUN find_seats_sold (v_theater_id => 500);
- ○ EXECUTE find_seats_sold (v_movie_id => 34);
- ○ EXECUTE find_seats_sold (v_theater_id => 500);
- ○ RUN find_seats_sold (v_movie_id => DEFAULT, v_theater_id => 500);

Answer:
EXECUTE find_seats_sold (v_theater_id => 500);

Explanation:

Because the V_MOVIE_ID argument is declared with a DEFAULT clause, it does not require a value when invoking this procedure. Because the V_THEATER_ID argument was not declared with a DEFAULT clause, it requires a value at invocation.

Argument values may be specified using the positional or the named method. However, the named method must be used when not all arguments are specified or when arguments are specified in a different order than the declaration. The named method requires the use of the "=>" operator.

Objective:

Creating Procedures

Sub-Objective:

Describe the DEFAULT option for parameters

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 150 (Ref:1z0-147e.2.3.7)

Which statement about declaring arguments for procedures is true?

- ○ Precision must be specified for VARCHAR2 arguments.
- ○ Each declared argument must have a mode specified.
- ○ An IN argument passes a value from a procedure to the calling environment.
- ○ Formal arguments allow you to transfer values to and from the calling environment.

Answer:
Formal arguments allow you to transfer values to and from the calling environment.

Explanation:

Formal arguments allow you to transfer values to and from the calling environment. The three procedural parameter modes are IN, OUT, and IN OUT. IN passes a constant value from the calling environment to the procedure. OUT passes a constant value from the procedure to the calling environment. IN OUT passes a value from the calling environment into the procedure and maybe a different value back to the calling environment using the same parameter. Because IN is the default mode, it is not required to specify the IN parameter mode.

The option stating that precision must be specified for VARCHAR2 arguments is incorrect. Specifying the precision for a formal argument is not allowed.

Objective:

Creating Procedures

Sub-Objective:

Create a procedure

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 155 (Ref:1z0-147e.2.2.1)

What should be placed after the IS keyword when creating a procedure?

- ○ parameters
- ○ local variables only
- ○ global variables only
- ○ global and local variables

Answer:
local variables only

Explanation:

Local variables must be placed after the IS keyword.

Example:

```
CREATE OR REPLACE PROCEDURE check_sal
IS
v_sal emp.sal%type;
BEGIN
SELECT sal
INTO v_sal
FROM emp
WHERE ename = 'SMITH';
END;
```

The variable V_SAL is declared and can be referenced in the BEGIN section. The DECLARE keyword (found in anonymous blocks) must not be used.

Objective:

Creating Procedures

Sub-Objective:

List the development steps for creating a procedure

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 159 (Ref:1z0-147e.2.1.2)

You execute this code:

```
CREATE OR REPLACE PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
BEGIN
null;
END;
```

Which statement is true?

- ○ The statement compiles, and the procedure is created.
- ○ The statement complies with errors because there is no exception handling.
- ○ The statement complies with errors because the body does not contain a SQL statement.
- ○ The statement complies, and the procedure is stored as a data dictionary object in the database.

Answer:
The statement compiles, and the procedure is created.

Explanation:
This code will compile, and a procedure will be created.

The option stating that the statement complies with errors because there is no exception handling is incorrect because an exception handling section is optional. The declaration and executable sections are required. The option stating that the statement complies, and the procedure is stored as a data dictionary object in the database is incorrect because stored procedures are stored as schema objects. The option stating that the statement complies with errors because the body does not contain a SQL statement is incorrect because SQL statements are not required in a stored procedure. The executable section can contain SQL statements, but are not required.

Objective:
Creating Procedures

Sub-Objective:
Define what a stored procedure is

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 173 (Ref:1z0-147e.2.7.4)

Which statement about the use of the DEFAULT clause in the declaration of a formal parameter is true?

- ○ IN parameters must be initialized with a default value.
- ○ OUT parameters must be initialized with a default value.
- ○ IN parameters cannot be initialized with a default value.
- ○ IN OUT parameters cannot be initialized with a default value.

Answer:
IN OUT parameters cannot be initialized with a default value.

Explanation:
IN parameters can be initialized to a default value using the DEFAULT option, but this is not required. IN OUT and OUT parameters cannot have a default value declared.

Objective:
Creating Procedures

Sub-Objective:
Describe the DEFAULT option for parameters

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Managing Dependencies

Item: 1 (Ref:1z0-147e.11.9.1)

When a local procedure that references a remote procedure is compiled, what is recorded in the p-code or object code?

- ○ both timestamps of the local and remote procedures
- ○ the timestamp of the local procedure compilation only
- ○ the timestamp of the remote procedure compilation only
- ○ no timestamps, unless a reference is made to PRAGMA RESTRICT_REFERENCES

Answer:
both timestamps of the local and remote procedures

Explanation:

When compiling a local subprogram that invokes a remote subprogram, the compilation timestamp of the remote subprogram is stored in the object code of the local subprogram. When executing the local subprogram, the compilation timestamp of the remote subprogram recorded in its object code will be compared to the actual timestamp. If the actual timestamp of the remote subprogram is later than what is recorded in the local subprogram's object code, an error will result. This will mark the local subprogram invalid. Executing it a second time will result in a successful recompilation because the new timestamp of the remote subprogram will be recorded in the object code of the local subprogram.

You should always manually recompile the local subprogram after the remote subprogram has recompiled.

Objective:

Managing Dependencies

Sub-Objective:

Describe when a remote dependency is unsuccessfully recompiled

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 8 (Ref:1z0-147e.11.1.4)

When a change is made to a referenced database object, what can be said about existing dependencies?

- ○ Only direct dependent objects are affected.
- ○ Only indirect dependent objects are affected.
- ○ Only those dependencies that directly reference a database table are affected.
- ○ All direct and indirect dependent objects are affected.

Answer:
All direct and indirect dependent objects are affected.

Explanation:
A modification to a referenced database object will result in marking all dependent objects invalid. If the object is a PL/SQL construct, it will be recompile automatically upon the next execution.

Objective:
Managing Dependencies

Sub-Objective:
Track procedural dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 9 (Ref:1z0-147e.11.1.1)

Which two statements about dependent objects are true? (Choose two.)

- ☐ A procedure that updates a table is a dependent object.
- ☐ A procedure that calls another procedure is a dependent object.
- ☐ A table that is modified by a trigger is a dependent object.
- ☐ A function that is called by another function is a dependent object.
- ☐ A view that is queried by a procedure is a dependent object.

Answer:
A procedure that updates a table is a dependent object.
A procedure that calls another procedure is a dependent object.

Explanation:

Objects that reference other objects as part of their definition are dependent objects. The object being referenced by the dependent object is referred to as a referenced object. The option stating that a procedure that updates a table is a dependent object is true because the procedure references the table; therefore, it is a dependent object. The option stating that a procedure that calls another procedure is a dependent object is true because the procedure references another procedure; therefore, it is a dependent object.

The remaining options are incorrect because of all the objects are referenced objects, not dependent objects. The option stating that a table that is modified by a trigger is a dependent object is incorrect because the trigger is the dependent object. The option stating that a function that is called by another function is a dependent object is incorrect because the calling function is the dependent object. The option stating that a view that is queried by a procedure is a dependent object is incorrect because the procedure is the dependent object.

Objective:

Managing Dependencies

Sub-Objective:

Track procedural dependencies

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 30 (Ref:1z0-147e.11.1.10)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE calculate_budget
IS
v_budget studio.yearly_budget%TYPE;
BEGIN
v_budget := get_budget@proddb(11);
IF v_budget < 30000000 THEN
set_budget@proddb (11, 300000000);
END IF;
END;
```

The dependency mode is set to TIMESTAMP.

The local procedure, CALCULATE_BUDGET, was compiled yesterday at 8:00 a.m., after compiling the GET_BUDGET remote function. The GET_BUDGET remote function was recompiled at 4:00 p.m. today. What can be said about the subsequent executions of CALCULATE_BUDGET?

- ○ All future execution attempts will result in a runtime error.
- ○ The first execution attempt will result in a recompilation and, if successful, will re-execute.
- ○ Unless the formal arguments of GET_BUDGET have changed, CALCULATE_BUDGET will execute without recompilation.
- ○ The first execution attempt will result in a runtime error. The second execution attempt will result in a recompilation and, if successful, will re-execute.

Answer:

The first execution attempt will result in a runtime error. The second execution attempt will result in a recompilation and, if successful, will re-execute.

Explanation:

When compiling a local construct (CALCULATE_BUDGET) that invokes a remote construct (GET_BUDGET), the compilation timestamp of the remote construct (8:00 a.m.) is stored in the object code of the local construct. When executing the local construct, the compilation timestamp of the remote procedure recorded in its object code will be compared to the actual compilation timestamp. If the actual timestamp of the remote construct (4:00 p.m. after recompilation) is later than what is recorded in the local construct's object code, an error will result. This will mark the local construct invalid. Executing it a second time will result in a successful recompilation because the new timestamp of the remote construct will be recorded in the object code of the local construct.

Objective:

Managing Dependencies

Sub-Objective:

Track procedural dependencies

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 41 (Ref:1z0-147e.11.3.1)

How does a local dependent object differ from a remote dependent object?

- ○ Local dependent objects are on the same node in the same database.
- ○ Remote dependent objects are on the same node in the same database.
- ○ Local dependent objects are on different nodes in the same database.
- ○ Remote dependent objects are on the same node in different databases.

Answer:
Local dependent objects are on the same node in the same database.

Explanation:
Objects that reference other objects as part of their definition are dependent objects. The object being referenced by the dependent object is referred to as a referenced object. A dependent object may either be local or remote. A local dependency is on the same node in the same database. A remote dependency occurs when the objects are on separate nodes.

Objective:
Managing Dependencies

Sub-Objective:
Define local dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 43 (Ref:1z0-147e.11.1.6)

Examine this package:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
v_total_budget NUMBER;
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

You are considering adding a call to the stand-alone function, SET_BUDGET. If you make this change, what should you be concerned with if the code of SET_BUDGET is modified in the future?

- ○ The entire package body will be invalidated and will recompile upon the next execution.
- ○ The entire package body will be invalidated and must be manually compiled before the next execution.
- ○ The entire package specification will be invalidated and will recompile upon the next execution.
- ○ The entire package specification will be invalidated and must be manually compiled before the next execution.

Answer:
The entire package body will be invalidated and will recompile upon the next execution.

Explanation:

Changing a stand-alone construct that is invoked by a package will mark the body of the package invalid. It will recompile upon the next invocation if not done manually.

Objective:
Managing Dependencies

Sub-Objective:
Track procedural dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 44 (Ref:1z0-147e.11.1.9)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE calculate_budget
IS
v_budget studio.yearly_budget%TYPE;
BEGIN
v_budget := get_budget (11);
IF v_budget < 30000000 THEN
set_budget (11, 300000000);
END IF;
END;
```

If the GET_BUDGET function is currently marked as invalid, when will it be recompiled implicitly?

- ○ never, but may be manually recompiled
- ○ the next time the SET_BUDGET procedure is analyzed
- ○ the second time the GET_BUDGET function is executed
- ○ the next time the CALCULATE_BUDGET procedure is executed

Answer:
the next time the CALCULATE_BUDGET procedure is executed

Explanation:
The status of a construct is checked upon invocation. If it is invalid, the construct will be recompiled. If the compilation is successful, it will then execute.

Objective:
Managing Dependencies

Sub-Objective:
Track procedural dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 45 (Ref:1z0-147e.11.1.3)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE calculate_budget
IS
v_budget studio.yearly_budget%TYPE;
v_studio_rec studio%ROWTYPE;
BEGIN
SELECT *
INTO v_studio_rec
FROM studio
WHERE NAME = 'Vince Productions';

v_budget := get_budget (v_studio_rec.id);
IF v_budget < 30000000 THEN
set_budget (11, 300000000);
END IF;
END;
```

Which effect could you expect if a new column were added to the STUDIO table?

- ○ Since the code references %ROWTYPE, no affect will be noticed.
- ○ The CALCULATE_BUDGET procedure would be marked valid and would automatically compile successfully upon the next execution.
- ○ The CALCULATE_BUDGET procedure would be marked invalid and would automatically compile successfully upon the next execution.
- ○ The CALCULATE_BUDGET procedure would be marked invalid and would automatically compile unsuccessfully upon the next execution.

Answer:

The CALCULATE_BUDGET procedure would be marked invalid and would automatically compile successfully upon the next execution.

Explanation:
This procedure is returning a row from the STUDIO table into a record defined using %ROWTYPE. This declares the record with fields that match each column of the STUDIO table. If a column is added to the STUDIO table, CALCULATE_BUDGET will be marked invalid. Upon the next execution of CALCULATE_BUDGET, it will recompile and declare the record based on the structure of the STUDIO table at that moment.

Objective:
Managing Dependencies

Sub-Objective:
Track procedural dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 49 (Ref:1z0-147e.11.1.2)

Examine this function:

```
CREATE OR REPLACE FUNCTION set_budget
(v_studio_id IN NUMBER, v_new_budget IN NUMBER)
RETURN number
IS
BEGIN
UPDATE studio
SET yearly_budget = v_new_budget
WHERE id = v_studio_id;
COMMIT;
RETURN SQL%ROWCOUNT;
END;
```

This function is executed from within a procedure called, CALCULATE_BUDGET. When executing CALCULATE_BUDGET, which relationships will be identified?

- ○ a direct dependency between SET_BUDGET and STUDIO only
- ○ a direct dependency between SET_BUDGET and CALCULATE_BUDGET only
- ○ a direct dependency between SET_BUDGET, CALCULATE_BUDGET, and STUDIO
- ○ a direct dependency between SET_BUDGET and CALCULATE_BUDGET, a direct dependency between SET_BUDGET and STUDIO, and an indirect dependency between CALCULATE_BUDGET and STUDIO

Answer:
a direct dependency between SET_BUDGET and CALCULATE_BUDGET, a direct dependency between SET_BUDGET and STUDIO, and an indirect dependency between CALCULATE_BUDGET and STUDIO

Explanation:
The function, SET_BUDGET, queries the STUDIO table (a direct dependency). CALCULATE_BUDGET executes SET_BUDGET (a direct dependency). Therefore, an indirect dependency exists between CALCULATE_BUDGET and STUDIO.

Objective:
Managing Dependencies

Sub-Objective:
Track procedural dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 50 (Ref:1z0-147e.11.1.8)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE calculate_budget
IS
BEGIN
v_budget := get_budget (11);
IF v_budget < 30000000 THEN
set_budget(11, 300000000);
END IF;
END;
```

A table that SET_BUDGET references has been accidentally dropped. Until this problem is resolved, what is the status of these procedures?

- ○ Only SET_BUDGET is marked invalid.
- ○ Only GET_BUDGET and SET_BUDGET are marked invalid.
- ○ Only CALCULATE_BUDGET and SET_BUDGET are marked invalid.
- ○ All three are marked invalid.

Answer:
Only CALCULATE_BUDGET and SET_BUDGET are marked invalid.

Explanation:
SET_BUDGET accesses the STUDIO table, which is invoked from the CALCULATE_BUDGET procedure. Therefore, an indirect relationship exists between STUDIO and CALCULATE_BUDGET. When a dependent object is changed, all objects involved in a direct or indirect relationship are marked invalid. In this case, both SET_BUDGET and CALCULATE_BUDGET are marked invalid.

Objective:
Managing Dependencies

Sub-Objective:
Track procedural dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 64 (Ref:1z0-147e.11.11.1)

Examine this code:

```
CREATE OR REPLACE PROCEDURE find_seats_sold
(v_movie_id IN NUMBER)
IS
v_seats_sold NUMBER(3);
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id;
END;
```

You are concerned with future changes to the database, such as modifying the maximum length of the SEATS_SOLD column of the GROSS_RECEIPT table. Which change should you make to guarantee successful recompilation of this procedure in the event of such a change?

- ○ Define V_SEATS_SOLD as GROSS_RECEIPT.SEATS_SOLD%TYPE.
- ○ Define V_SEATS_SOLD using the PRAGMA EXCEPTION_INIT compiler directive.
- ○ Change V_SEATS_SOLD to be a formal argument rather than a local variable.
- ○ Define V_SEATS_SOLD using the PRAGMA RESTRICT_REFERENCES compiler directive.

Answer:
Define V_SEATS_SOLD as GROSS_RECEIPT.SEATS_SOLD%TYPE.

Explanation:

%TYPE is used to declare a scalar variable that matches the data type and maximum length of the specified column.

```
v_seats_sold gross_receipt.seats_sold%TYPE;
```

If the SEATS_SOLD column is subsequently altered in the database, the constructs containing references to the column are marked invalid. The constructs must be recompiled before the next execution. The compilation will reflect the new alterations of the columns.

If the SEATS_SOLD column is defined as:

```
v_seats_sold number(4);
```

and the size of the SEATS_SOLD column of the GROSS_RECEIPT table is increased, all constructs containing this declaration might fail upon the next execution.

Objective:
Managing Dependencies

Sub-Objective:
List how to minimize dependency failures

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 72 (Ref:1z0-147e.11.11.3)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE calculate_budget
IS
v_budget studio.yearly_budget%TYPE;
BEGIN
v_budget := get_budget (11);
IF v_budget < 30000000 THEN
set_budget (11, 300000000);
END IF;
END;
```

For which reason would the recompilation of CALCULATE_BUDGET be unsuccessful?

- ○ a change to the code of SET_BUDGET
- ○ a change to the code of GET_BUDGET
- ○ a change to the arguments of SET_BUDGET or GET_BUDGET
- ○ a change to the maximum length of the YEARLY_SALARY column of the STUDIO table

Answer:
a change to the arguments of SET_BUDGET or GET_BUDGET

Explanation:
If the formal arguments of a construct change, all constructs that invoke that construct must be modified to reflect the change. If not, recompilation of these constructs will fail.

Objective:
Managing Dependencies

Sub-Objective:
List how to minimize dependency failures

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 83 (Ref:1z0-147e.11.4.1)

Which data dictionary table can you query to determine all stand-alone procedures that reference the THEATER_PCK package?

- ○ USER_OBJECTS
- ○ USER_DEPTREE
- ○ USER_PACKAGES
- ○ USER_DEPENDENCIES

Answer:
USER_DEPENDENCIES

Explanation:
USER_DEPENDENCIES is a data dictionary table that can be queried to view all dependencies between all objects within a user schema.

Objective:
Managing Dependencies

Sub-Objective:
View dependency information in the data dictionary views

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 84 (Ref:1z0-147e.11.5.3)

Which three tables or views could you use to help track dependencies? (Choose three.)

- ☐ DEPTREE
- ☐ IDEPTREE
- ☐ DEPENDENCIES
- ☐ USER_OBJECTS
- ☐ USER_DEPENDENCIES

Answer:
DEPTREE
IDEPTREE
USER_DEPENDENCIES

Explanation:

You can query USER_DEPENDENCIES to display all the direct dependencies on a schema object, not just the ones you created. The ALL_DEPENDENCIES and DBA_DEPENDENCIES views have an OWNER column to reference the owner of the object. To display direct and indirect and dependencies, run the utldtree.sql script and execute the DEPTREE_FILL procedure. The syntax for the DEPTREE_FILL procedure is:

```
EXECUTE deptree_fill ('object_type', 'object_owner', 'object_name')
```

After executing the DEPTREE_FILL procedure, you can query the DEPTREE and IDEPTREE views to display the dependencies.

Objective:

Managing Dependencies

Sub-Objective:

Describe how the UTLDTREE script is used

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 89 (Ref:1z0-147e.11.5.2)

Which statement will create the dependency structure of the DEPARTMENT table in the PROD schema?

- ◌ EXECUTE deptree_fill ('PROD.DEPARTMENT')
- ◌ EXECUTE deptree_fill ('PROD', 'DEPARTMENT')
- ◌ EXECUTE deptree_fill ('TABLE', 'PROD.DEPARTMENT')
- ◌ EXECUTE deptree_fill ('TABLE', 'PROD', 'DEPARTMENT')

Answer:
EXECUTE deptree_fill (&apos;TABLE&apos;, &apos;PROD&apos;, &apos;DEPARTMENT&apos;)

Explanation:

To display indirect dependencies, run the utldtree.sql script and execute the DEPTREE_FILL procedure. The syntax for the DEPTREE_FILL procedure is:

EXECUTE deptree_fill ('object_type', 'object_owner', 'object_name')

After executing the DEPTREE_FILL procedure, you can query the DEPTREE and IDEPTREE views to display the dependencies.

Objective:

Managing Dependencies

Sub-Objective:

Describe how the UTLDTREE script is used

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 105 (Ref:1z0-147e.11.7.1)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE calculate_budget
IS
v_budget studio.yearly_budget%TYPE;
BEGIN
v_budget := get_budget@proddb(11);
IF v_budget < 30000000 THEN
set_budget (11, 300000000);
END IF;
END;
```

You are about to add an argument to the local procedure, CALCULATE_BUDGET. What effect will this have on the remote procedure, GET_BUDGET?

- ○ It will be marked as invalid
- ○ It will be recompiled at the next execution.
- ○ It will be recompiled at the second execution.
- ○ There is no effect.

Answer:
There is no effect.

Explanation:
Since GET_BUDGET does not invoke CALCULATE_BUDGET, changing the argument list of CALCULATE_BUDGET has no effect on GET_BUDGET.

Objective:
Managing Dependencies

Sub-Objective:
Describe a remote dependency

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 107 (Ref:1z0-147e.11.4.2)

Which data dictionary view can you query to examine all the dependencies between the objects that you own?

- ○ USER_OBJECTS
- ○ USER_RELATIONS
- ○ USER_DEPENDENCIES
- ○ USER_RELATIONSHIPS

Answer:
USER_DEPENDENCIES

Explanation:
The USER_DEPENDENCIES view contains the dependencies between the objects you own. For example, you could view all the procedures, functions, and packages that reference the EMP table.

Objective:
Managing Dependencies

Sub-Objective:
View dependency information in the data dictionary views

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 111 (Ref:1z0-147e.11.4.3)

Evaluate this SELECT statement:

```
SELECT *
FROM user_dependencies
WHERE referenced_name = 'EMPLOYEE';
```

The EMPLOYEE table is in your schema. Which result will this statement accomplish?

- ○ displays all the direct dependencies on the EMPLOYEE table
- ○ displays only the direct dependencies you created on the EMPLOYEE table
- ○ displays the name of the indirect dependencies on the EMPLOYEE table
- ○ displays the OWNER column to indicate the schema of any referenced objects

Answer:
displays all the direct dependencies on the EMPLOYEE table

Explanation:
You can query USER_DEPENDENCIES to display all the direct dependencies on a schema object, not just the ones you created. The ALL_DEPENDENCIES and DBA_DEPENDENCIES views have an OWNER column to reference the owner of the object. To display indirect dependencies, run the utldtree.sql script and execute the DEPTREE_FILL procedure.

Objective:
Managing Dependencies

Sub-Objective:
View dependency information in the data dictionary views

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 114 (Ref:1z0-147e.11.6.1)

Which two views, when created, are queried to display indirect dependencies? (Choose two.)

- ☐ DEPTREE
- ☐ IDEPTREE
- ☐ USER_IDTREE
- ☐ USER_OBJECTS
- ☐ USER_DEPENDENCIES

Answer:
DEPTREE
IDEPTREE

Explanation:
The UTLDTREE.SQL script file creates additional views and a procedure to provide a much easier alternative method of viewing indirect dependencies.

After executing this script file, you can invoke the DEPTREE_FILL procedure and view the results by querying the DEPTREE or IDEPTREE views.

Objective:
Managing Dependencies

Sub-Objective:
Describe how the IDEPTREE and DEPTREE procedures are used

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 127 (Ref:1z0-147e.11.2.1)

Which type(s) of objects does Oracle keep track of?

- ◯ all object types
- ◯ database triggers only
- ◯ tables, procedures, and functions only
- ◯ packages, procedures, and functions only

Answer:
all object types

Explanation:

Dependencies exist between objects that reference other objects. This information is stored in the data dictionary and can be viewed by the USER_DEPENDENCIES view.

Executing the UTLDTREE.SQL script file creates additional views and a procedure to provide a much easier alternative method of viewing indirect and direct dependencies.

Objective:

Managing Dependencies

Sub-Objective:

Describe dependent objects and referenced objects

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 128 (Ref:1z0-147e.11.2.2)

Which statement about a dependent procedure is true?

- ○ A dependent procedure only directly references a view, sequence, procedure, or packaged procedure or function.
- ○ A dependent procedure only indirectly references a view, sequence, procedure, or packaged procedure or function.
- ○ A dependent procedure directly or indirectly references a view, sequence, procedure, or packaged procedure or function.
- ○ A dependent procedure neither directly nor indirectly references a view, sequence, procedure, or packaged procedure or function.

Answer:

A dependent procedure directly or indirectly references a view, sequence, procedure, or packaged procedure or function.

Explanation:
Objects that reference other objects as part of their definition are dependent objects. The object being referenced by the dependent object is referred to as a referenced object. A procedure or function may either directly or indirectly reference a table, view, sequence, procedure, function, or packaged procedures or functions. A procedure or function indirectly references an object through an intermediate view, procedure, function, or packaged function or procedure.

Objective:
Managing Dependencies

Sub-Objective:
Describe dependent objects and referenced objects

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 149 (Ref:1z0-147e.11.1.11)

Examine this package:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
v_total_budget NUMBER;
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number;
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

The stand-alone procedure, VALIDATE_THEATER_REVENUE, references the GET_BUDGET function of this package. What result would you expect if the package body was changed and recompiled?

- The next execution of VALIDATE_THEATER_REVENUE would fail.
- The status of VALIDATE_THEATER_REVENUE would not be affected.
- The status of VALIDATE_THEATER_REVENUE would be marked invalid.
- The next execution of VALIDATE_THEATER_REVENUE would invoke a recompilation.

Answer:
The status of VALIDATE_THEATER_REVENUE would not be affected.

Explanation:

Modifying only the body of a package does not affect dependent objects. Therefore, VALIDATE_THEATER_REVENUE is not affected.

If the modification included changing the formal arguments, the package specification would be modified as well, resulting in all dependent objects being marked invalid.

Objective:
Managing Dependencies

Sub-Objective:
Track procedural dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 152 (Ref:1z0-147e.11.1.7)

Examine this function:

```
CREATE OR REPLACE FUNCTION set_budget
(v_studio_id IN NUMBER, v_new_budget IN NUMBER)
RETURN number
IS
BEGIN
UPDATE studio
SET yearly_budget = v_new_budget
WHERE id = v_studio_id;
COMMIT;
RETURN SQL%ROWCOUNT;
END;
```

This function is executed from within a procedure called CALCULATE_BUDGET. The database administrator has just informed you that a new column has been added to the STUDIO table. What effect will this have?

- ○ Only SET_BUDGET will be marked invalid.
- ○ Only CALCULATE_BUDGET will be marked invalid.
- ○ SET_BUDGET and CALCULATE_BUDGET will be marked invalid.
- ○ SET_BUDGET, CALCULATE_BUDGET, and STUDIO will be marked invalid.

Answer:
SET_BUDGET and CALCULATE_BUDGET will be marked invalid.

Explanation:
SET_BUDGET accesses the STUDIO table, which is invoked from the CALCULATE_BUDGET procedure. An indirect relationship exists between STUDIO and CALCULATE_BUDGET. When a dependent object is changed, all objects involved in a direct or indirect relationship are marked invalid. In this case, both SET_BUDGET and CALCULATE_BUDGET are marked invalid.

Objective:
Managing Dependencies

Sub-Objective:
Track procedural dependencies

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Item: 154 (Ref:1z0-147e.11.5.1)

To easily determine indirect dependencies between specified objects, additional views can be created. Which script file, provided by Oracle, will create these views when executed?

- ○ deptree.sql
- ○ catalog.sql
- ○ utldtree.sql
- ○ dbms_tree.sql

Answer:
utldtree.sql

Explanation:
The UTLDTREE.SQL script file creates additional views and a procedure to provide a much easier alternative method of viewing indirect dependencies. After executing this script file, you can invoke the DEPTREE_FILL procedure and view the results by querying the DEPTREE or IDEPTREE views.

Objective:
Managing Dependencies

Sub-Objective:
Describe how the UTLDTREE script is used

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Dependencies

Managing Subprograms

Item: 17 (Ref:1z0-147e.4.3.4)

All users in the HR_EMP role have UPDATE privileges on the EMPLOYEE table. You create the UPDATE_EMPLOYEE procedure. HR_EMP users should only be able to update the EMPLOYEE table using this procedure. Which two statements should you execute? (Choose two.)

- ☐ GRANT UPDATE ON employee TO hr_emp;
- ☐ GRANT SELECT ON employee to hr_emp;
- ☐ REVOKE UPDATE ON employee FROM hr_emp;
- ☐ REVOKE UPDATE ON employee FROM public;
- ☐ GRANT EXECUTE ON update_employee TO hr_emp;

Answer:
REVOKE UPDATE ON employee FROM hr_emp;
GRANT EXECUTE ON update_employee TO hr_emp;

Explanation:

Unless you are the owner of the PL/SQL construct, you must be granted the EXECUTE object privilege to run it or have the EXECUTE ANY PROCEDURE system privilege. By default, a PL/SQL procedure executes under the security domain of its owner. This means that a user can invoke the procedure without privileges on the procedures underlying objects. To allow HR_EMP users to execute the procedure, you must issue the GRANT EXECUTE ON update_employee TO hr_emp; statement. To prevent HR_EMP users from updating the EMPLOYEE table unless they are using the UPDATE_EMPLOYEE procedure, you must issue the REVOKE UPDATE ON employee FROM hr_emp;

The remaining options are incorrect.

Objective:

Managing Subprograms

Sub-Objective:

Contrast invokers rights with definers rights

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 22 (Ref:1z0-147e.4.3.3)

The database administrator has informed you that the CREATE PROCEDURE privilege has been granted to your account. Which objects can you now create?

- ○ procedures only
- ○ packages and procedures only
- ○ functions and procedures only
- ○ procedures, functions, and packages

Answer:
procedures, functions, and packages

Explanation:
The CREATE PROCEDURE privilege allows a user to create procedures, functions, and packages. There is not a separate privilege for functions and packages.

Objective:
Managing Subprograms

Sub-Objective:
Contrast invokers rights with definers rights

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 28 (Ref:1z0-147e.4.3.1)

Which statement concerning the use of a procedure is true?

- ○ A user needs only the RESOURCE role to execute a procedure.
- ○ A user must be given privileges to the procedure and all underlying tables.
- ○ A user needs only the privilege to execute the procedure and does not need privileges on the underlying tables.
- ○ If a user has the privilege to query a table and that table is referenced within a procedure, the user can execute that procedure.

Answer:
A user needs only the privilege to execute the procedure and does not need privileges on the underlying tables.

Explanation:
Unless you are the owner of the PL/SQL construct, you must be granted the EXECUTE object privilege to run it or have the EXECUTE ANY PROCEDURE system privilege. By default, a PL/SQL procedure executes under the security domain of its owner. This means that a user can invoke the procedure without privileges on the procedures underlying objects. If you wanted the procedure to execute under the executing owner's privileges and not the owner, you would use the AUTHID CURRENT_USER option.

The options stating that a user needs only the RESOURCE role to execute a procedure and if a user has the privilege to query a table and that table is referenced within a procedure, the user can execute that procedure are incorrect. To execute a procedure, the user must own the procedure, have been grated the EXECUTE object privilege on the procedure, or have been granted the EXECUTE ANY PROCEDURE system privilege.

Objective:
Managing Subprograms

Sub-Objective:
Contrast invokers rights with definers rights

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 51 (Ref:1z0-147e.4.3.2)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE update_theater
(v_name IN VARCHAR2)
IS
BEGIN
DELETE theater
WHERE id = 34;
END update_theater;
```

This procedure is owned by PROD. The user JSMITH must execute this procedure. Which command(s) must PROD execute to grant the necessary privileges to JSMITH?

- ○ GRANT EXECUTE ON update_theater TO jsmith;
- ○ GRANT EXECUTE, DELETE ON theater TO jsmith;
- ○ GRANT EXECUTE, DELETE ON update_theater TO jsmith;
- ○ GRANT DELETE ON theater TO jsmith;
  GRANT EXECUTE ON update_theater TO jsmith;

Answer:
GRANT EXECUTE ON update_theater TO jsmith;

Explanation:

A user must be issued the EXECUTE privilege on the procedure. The user does not require the DELETE privilege on the THEATER table.

If the new AUTHID CURRENT_USER directive is specified, the user would be required to have the DELETE privilege on the THEATER table.

Objective:

Managing Subprograms

Sub-Objective:

Contrast invokers rights with definers rights

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 70 (Ref:1z0-147e.4.4.8)

Which view can you query to determine the validity of a particular procedure?

- ○ USER_STATUS
- ○ USER_SOURCE
- ○ USER_OBJECTS
- ○ USER_PROCEDURES

Answer:
USER_OBJECTS

Explanation:
The USER_OBJECTS view contains the STATUS column that indicates the validity of each construct. If a construct is invalid, it will be automatically recompiled upon the next invocation.

The ALL_OBJECTS view also contains the STATUS column.

Objective:
Managing Subprograms

Sub-Objective:
Identify views in the data dictionary to manage stored objects

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 85 (Ref:1z0-147e.4.4.1)

Which data dictionary table can you query to determine all stand-alone procedures that reference the THEATER_PCK package?

- ○ USER_OBJECTS
- ○ USER_DEPTREE
- ○ USER_PACKAGES
- ○ USER_DEPENDENCIES

Answer:
USER_DEPENDENCIES

Explanation:
USER_DEPENDENCIES is a data dictionary table that can be queried to view all dependencies between all objects within a user schema.

Objective:
Managing Subprograms

Sub-Objective:
Identify views in the data dictionary to manage stored objects

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 104 (Ref:1z0-147e.4.4.4)

You have lost the script file that contains the source code for the THEATER_PCK package. Which command will produce the source code stored in the database?

- ○ SELECT text
  FROM user_source
  WHERE name = 'THEATER_PCK';
- ○ SELECT text
  FROM user_objects
  WHERE name = 'THEATER_PCK';
- ○ SELECT text
  FROM user_packages
  WHERE name = 'THEATER_PCK';
- ○ SELECT text
  FROM user_programs
  WHERE name = 'THEATER_PCK';

Answer:
SELECT textFROM user_sourceWHERE name = &apos;THEATER_PCK&apos;;

Explanation:
The USER_SOURCE view contains a column called TEXT. This column contains the source code of the particular procedure, function, or package.

The source code of database triggers are not included in this view. Instead, query the USER_TRIGGERS view.

Objective:
Managing Subprograms

Sub-Objective:
Identify views in the data dictionary to manage stored objects

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 109 (Ref:1z0-147e.4.4.2)

Which data dictionary view can you query to examine all the dependencies between the objects that you own?

- ○ USER_OBJECTS
- ○ USER_RELATIONS
- ○ USER_DEPENDENCIES
- ○ USER_RELATIONSHIPS

Answer:
USER_DEPENDENCIES

Explanation:
The USER_DEPENDENCIES view contains the dependencies between the objects you own. For example, you could view all the procedures, functions, and packages that reference the EMP table.

Objective:
Managing Subprograms

Sub-Objective:
Identify views in the data dictionary to manage stored objects

References:
1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 132 (Ref:1z0-147e.4.4.6)

Which data dictionary view must you query to determine when a particular procedure or function was created?

- ○ USER_SOURCE
- ○ USER_STATUS
- ○ USER_OBJECTS
- ○ USER_CONSTRUCTS

Answer:
USER_OBJECTS

Explanation:

The USER_OBJECTS view contains a column called CREATED. This column contains the date and time a particular construct was created.

Objective:

Managing Subprograms

Sub-Objective:

Identify views in the data dictionary to manage stored objects

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 136 (Ref:1z0-147e.4.4.3)

Your stored procedure, GET_BUDGET, has a logic problem and must be modified. The script that contains the procedure code has been misplaced. Which data dictionary view can you query to capture the source code for this procedure?

- ○ USER_SOURCE
- ○ USER_OBJECTS
- ○ USER_CONSTRUCTS
- ○ USER_PROCEDURES

Answer:
USER_SOURCE

Explanation:

The USER_SOURCE view contains a column called TEXT. This column contains the source code of the particular procedure, function, or package.

The source code of database triggers is not included in this view. Instead, it can be obtained by querying the USER_TRIGGERS view.

Objective:

Managing Subprograms

Sub-Objective:

Identify views in the data dictionary to manage stored objects

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Item: 156 (Ref:1z0-147e.4.4.5)

Due to a change to a particular table, you are concerned with the number of stored procedures and functions that may have been affected. Which table can you query to check the status of each subprogram and determine which procedures and functions must be recompiled?

- ○ USER_STATUS
- ○ USER_SOURCE
- ○ USER_OBJECTS
- ○ USER_CONSTRUCTS

Answer:
USER_OBJECTS

Explanation:

The USER_OBJECTS table contains a column called STATUS. This column has two possible values: VALID or INVALID.

Invalid constructs will be compiled automatically upon the next execution.

A developer can explicitly compile constructs using the ALTER command.

For example:

```
ALTER PROCEDURE check_sal COMPILE;
```

Objective:

Managing Subprograms

Sub-Objective:

Identify views in the data dictionary to manage stored objects

References:

1. Oracle9i: Develop PL/SQL Program Units - Managing Subprograms

Manipulating Large Objects

Item: 18 (Ref:1z0-147e.8.2.3)

Which Oracle supplied package manages LOBs?

- ○ UTL_LOB
- ○ DBMS_LOB
- ○ UTL_MANAGE_LOB
- ○ DBMS_MANAGE_LOB

Answer:
DBMS_LOB

Explanation:
The DMBS_LOB package contains functions and procedures to access and manipulate internal and external LOBs. This package contains two types of functions and routines, mutators and observers. Mutators modify LOB values and observers read LOB values. The mutators include APPEND, COPY, ERASE, TRIM, WRITE, FILECLOSE, FILECLOSEALL, and FILEOPEN. The observers include COMPARE, FILEGETNAME, INSTR, GETLENGTH, READ, SUBSTR, FILEEXISTS, and FILEISOPEN. The FILECLOSE, FILECLOSEALL, FILEEXISTS, FILEGETNAME, FILEISOPEN, and FILEOPEN routines are specific to BFILEs.

The remaining options are invalid.

Objective:
Manipulating Large Objects

Sub-Objective:
Create and maintain LOB data types

References:
1. Oracle9i: Develop PL/SQL Program Units - Manipulating Large Objects

Item: 34 (Ref:1z0-147e.8.4.1)

Which function or procedure would you use to initialize a BFILE column for inserting data?

- ○ BFILE
- ○ TO_BFILE
- ○ BFILENAME
- ○ DBMS_LOB.READ
- ○ DBMS_LOB.FILEEXISTS

Answer:
BFILENAME

Explanation:

LOB data types store large, unstructured data, including text, graphic images, and video clips. There are two types of LOBs, internal and external. All LOB columns store a locator. If a LOB is internal, the locator points to a LOB segment inside the database. If a LOB is external, the locator points to an external OS location. BFILE represents a binary file stored in the OS outside of the database and is an external LOB. The BFILE stores a file locator to the external file. External LOBs are read-only and cannot participate in transactions. BFILENAME is a built-in function that can be used to initialize a BFILE column to point to an external file. Use the BFILENAME function in an INSERT statement to associate a BFILE column value with an external OS file.

Objective:

Manipulating Large Objects

Sub-Objective:

Identify and manage Bfiles: Create directories, Use Bfiles, Load Bfiles, Use Bfilename Function

References:

1. Oracle9i: Develop PL/SQL Program Units - Manipulating Large Objects

Item: 66 (Ref:1z0-147e.8.5.1)

Which statement would you use to migrate a LONG data type to a LOB data type?

- ○ ALTER TABLE
- ○ ALTER COLUMN
- ○ CREATE TABLE
- ○ ALTER DATABASE
- ○ You cannot migrate a LONG data type column to a LOB data type.

Answer:
ALTER TABLE

Explanation:
You can migrate LONG columns to LOB columns to move existing tables containing LONG columns to use LOBs. To migrate the data from a LONG column to a CLOB or an NCLOB or a LONG RAW to a BLOB, use the ALTER TABLE statement.

Objective:
Manipulating Large Objects

Sub-Objective:
Migrate from LONG to LOB

References:
1. Oracle9i: Develop PL/SQL Program Units - Manipulating Large Objects

Item: 106 (Ref:1z0-147e.8.4.2)

Which procedure would you use to close a BFILE that is being accessed?

- ○ DBMS_LOB.FILECLOSE
- ○ DBMS_LOB.WRITE
- ○ DBMS_LOB.BFILENAME
- ○ DBMS_LOB.READ
- ○ DBMS_LOB.FILEEXISTS

Answer:
DBMS_LOB.FILECLOSE

Explanation:

LOB data types store large, unstructured data, including text, graphic images, and video clips. There are two types of LOBs, internal and external. All LOB columns store a locator. If a LOB is internal, the locator points to a LOB segment inside the database. If a LOB is external, the locator points to an external OS location. BFILE represents a binary file stored in the OS outside of the database and is an external LOB. The BFILE stores a file locator to the external file. External LOBs are read-only and cannot participate in transactions. Use DBMS_LOB.FILECLOSE to close a BFILE that is being accessed.

Objective:

Manipulating Large Objects

Sub-Objective:

Identify and manage Bfiles: Create directories, Use Bfiles, Load Bfiles, Use Bfilename Function

References:

1. Oracle9i: Develop PL/SQL Program Units - Manipulating Large Objects

Item: 119 (Ref:1z0-147e.8.2.2)

Which statement about internal LOBs is true?

- ○ A BFILE is an internal LOB.
- ○ Internal LOBs are stored in the database.
- ○ Internal LOBs can be accessed only in read-only mode from an Oracle server.
- ○ The Oracle9i server performs an implicit conversion between internal and external LOBs.

Answer:
Internal LOBs are stored in the database.

Explanation:

LOB data types store large, unstructured data, including text, graphic images, and video clips. There are two types of LOBs, internal and external. All LOB columns store a locator. If a LOB is internal, the locator points to a LOB segment inside the database. If a LOB is external, the locator points to an external OS location. BFILE represents a binary file stored in the OS outside of the database and is an external LOB. The BFILE stores a file locator to the external file. External LOBs are read-only and cannot participate in transactions. The Oracle9i server performs implicit conversions between CLOB and VARCHAR2 data types. Implicit conversions between other LOBs are not possible.

Objective:

Manipulating Large Objects

Sub-Objective:

Create and maintain LOB data types

References:

1. Oracle9i: Develop PL/SQL Program Units - Manipulating Large Objects

Item: 146 (Ref:1z0-147e.8.5.2)

The resume column is of LONG data type and has a NOT NULL constraint.

Evaluate this statement:

```
ALTER TABLE employee
MODIFY (resume CLOB);
```

Which statement is true?

- ○ The statement will add a new column to an existing table.
- ○ The NOT NULL constraint will not be migrated to the LOB column.
- ○ The statement will convert a LONG data type column to a LOB data type.
- ○ The statement will return a syntax error.

Answer:
The statement will convert a LONG data type column to a LOB data type.

Explanation:

You can migrate LONG columns to LOB columns to move existing tables containing LONG columns to use LOBs. To migrate the data from a LONG column to a CLOB or an NCLOB or a LONG RAW to a BLOB, use the ALTER TABLE statement. The syntax is:

```
ALTER TABLE [schema.] table_name
MODIFY (long_col_name {CLOB | BLOB | NCLOB};
```

In this scenario, the ALTER TABLE statement migrates the RESUME column that is a LONG data type to a CLOB data type.

Objective:

Manipulating Large Objects

Sub-Objective:

Migrate from LONG to LOB

References:

1. Oracle9i: Develop PL/SQL Program Units - Manipulating Large Objects

Item: 147 (Ref:1z0-147e.8.2.1)

Which type of LOB represents a multi-byte character object?

- ○ BLOB
- ○ CLOB
- ○ BFILE
- ○ NCLOB

Answer:
NCLOB

Explanation:
LOB data types store large, unstructured data, including text, graphic images, and video clips. NCLOB represents a multibyte character object. BLOB represents a binary large object. CLOB represents a character large object. BFILE represents a binary file stored in the OS outside of the database. The BFILE stores a file locator to the external file.

Objective:
Manipulating Large Objects

Sub-Objective:
Create and maintain LOB data types

References:
1. Oracle9i: Develop PL/SQL Program Units - Manipulating Large Objects

More Package Concepts

Item: 21 (Ref:1z0-147e.6.5.2)

Examine this package:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
current_avg_cost_per_ticket NUMBER := 0;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number;

END theater_pck;
```

You issue these statements in SQL*Plus:

```
EXECUTE theater_pck.current_avg_cost_per_ticket := 10;
ROLLBACK;
```

What is true about the state of CURRENT_AVG_COST_PER_TICKET?

- ○ It is 10 for all user sessions.
- ○ It is 0 for the next transaction.
- ○ It is 0 for the remainder of the session.
- ○ It is 10 for the remainder of the session.

Answer:
It is 10 for the remainder of the session.

Explanation:

Package variables are those defined in the specification or in the declaration of the body.

The state of a package variable persists throughout the current user session. Therefore, CURRENT_AVG_COST_PER_TICKET remains 10 for the duration of the current session, unless it is changed or the package is invalidated. It is not removed at the end of a transaction.

Objective:

More Package Concepts

Sub-Objective:

Identify persistent states in package variables and cursors

References:

1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 35 (Ref:1z0-147e.6.5.4)

Examine this package:

```
CREATE OR REPLACE PACKAGE prod_pack
IS
CURSOR c1 is
SELECT *
FROM product;

PROCEDURE order_product
(p1 IN NUMBER, p2 IN NUMBER);
END prod_pack;

CREATE OR REPLACE PACKAGE BODY prod_pack
IS

PROCEDURE order_product
(p1 IN NUMBER, p2 IN NUMBER)
IS
CURSOR c2 IS
SELECT *
FROM ord;
BEGIN
OPEN c2;
...
END;
END prod_pack;
```

After being opened, which cursor or cursors persist for the entire user session?

- ○ C1 only
- ○ C2 only
- ○ both C1 and C2
- ○ neither C1 nor C2

Answer:
C1 only

Explanation:

Package variables are those defined in the specification or in the declaration of the body.

The state of a package variable persists throughout the current user session. Therefore, C1 will be opened for the duration of the session unless it is closed before the session ends.

C2 is declared within the ORDER_PRODUCT procedure. The state of C2 is persistent only during the execution of ORDER_PRODUCT.

Objective:

More Package Concepts

Sub-Objective:

Identify persistent states in package variables and cursors

References:

1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 46 (Ref:1z0-147e.6.5.3)

Examine this package:

```
CREATE OR REPLACE PACKAGE prod_pack
IS
CURSOR c1 is
SELECT *
FROM product;

PROCEDURE order_product
(p1 IN NUMBER, p2 IN NUMBER);
END prod_pack;

CREATE OR REPLACE PACKAGE BODY prod_pack
IS
CURSOR c2 IS
SELECT *
FROM ord;

PROCEDURE order_product
(p1 IN NUMBER, p2 IN NUMBER)
IS
BEGIN
OPEN c2;
...
END;
END prod_pack;
```

You execute these commands in SQL*Plus:

```
EXECUTE OPEN prod_pack.c1;
EXECUTE prod_pack.order_product(100,1);
```

Which cursor or cursors will be opened for the duration of the session, unless it is explicitly closed?

- ○ C1 only
- ○ C2 only
- ○ both C1 and C2
- ○ neither C1 nor C2

Answer:
both C1 and C2

Explanation:
Package variables are those defined in the specification or in the declaration of the body.

The state of a package variable persists throughout the current user session. Therefore, C1 and C2 will be open for the duration of the session unless they are closed before the session ends.

C1 is explicitly opened and C2 is opened when the ORDER_PRODUCT procedure is invoked.

Objective:
More Package Concepts

Sub-Objective:
Identify persistent states in package variables and cursors

References:
1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 53 (Ref:1z0-147e.6.7.1)

You have just moved a stand-alone function into a package. What could you add to the package specification to check the purity level of this function?

- ○ PRAGMA PURITY_CHECK
- ○ PRAGMA EXCEPTION_INIT
- ○ PRAGMA FUNCTION_PURITY
- ○ PRAGMA RESTRICT_REFERENCES

Answer:
PRAGMA RESTRICT_REFERENCES

Explanation:

In prior releases of Oracle, packaged functions must include a guarantee in the package specification that the body of the function does not update the database. PRAGMA RESTRICT_REFERENCES is a directive to the compiler indicating that the package body should not compile correctly if it fails one of its references.

Oracle9i can read the body of the packaged function during SQL statement processing and does not require this directive.

Objective:

More Package Concepts

Sub-Objective:

Invoke packaged functions with SQL

References:

1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 59 (Ref:1z0-147e.6.7.3)

Why would you use the PRAGMA RESTRICT_REFERENCES compiler directive within a package specification?

- ○ to allow a function to bypass all package restrictions
- ○ to allow a procedure to bypass all package restrictions
- ○ to specify the purity level of a function to be used in a SQL statement
- ○ to specify the purity level of a procedure to be used in a SQL statement

Answer:
to specify the purity level of a function to be used in a SQL statement

---

Explanation:

Certain restrictions apply when using packaged functions within SQL statements. For example, a function cannot insert, update, or delete rows if it is to be used in a SQL statement.

Purity level is the guarantee that the packaged function does not violate these restrictions. Specifying the PRAGMA RESTRICT_REFERENCES in the package specification guarantees that the function does not violate the required restrictions. If the function does not meet these requirements, the package body compilation will fail.

Objective:

More Package Concepts

Sub-Objective:

Invoke packaged functions with SQL

References:

1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 63 (Ref:1z0-147e.6.7.4)

Examine this package:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number;

END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

Which code could you add to the specification to check for the purity level of the GET_BUDGET function?

- PRAGMA PURITY_LEVEL(get_budget,PURITY(1)));
- PRAGMA RESTRICT_REFERENCES(get_budget,WNDS);
- PRAGMA RESTRICT_REFERENCES(get_budget,NO_UPDATE);
- PRAGMA EXCEPTION_INIT(get_budget,PURITY(NO_UPDATE));

Answer:
PRAGMA RESTRICT_REFERENCES(get_budget,WNDS);

Explanation:

Certain restrictions apply when using packaged functions within SQL statements. For example, a function cannot insert, update, or delete rows if it is to be used in a SQL statement.

Purity level is the guarantee that the packaged function does not fail these restrictions. Therefore, specifying the PRAGMA RESTRICT_REFERENCES in the package specification guarantees that the function does not fail the required restrictions. If the function does not meet these requirements, the package body compilation will fail.

To guarantee that the packaged function does not insert, update, or delete rows, use PRAGMA RESTRICT_REFERENCES (function name, WNDS).

WNDS is an acronym for "Writes No Database State".

Objective:
More Package Concepts

Sub-Objective:
Invoke packaged functions with SQL

References:
1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 73 (Ref:1z0-147e.6.6.2)

When using a PL/SQL stored package, which statement about side effects is true?

- ○ Side effects are changes to database tables or public packaged variables declared in the package specification.
- ○ Side effects do not delay the execution of a query.
- ○ All side effects are allowed when a function is called from a SQL query or DML statement.
- ○ Side effects are the yield order-dependent results of a function called from a SQL statement.

Answer:

Side effects are changes to database tables or public packaged variables declared in the package specification.

Explanation:

A function called from a SQL query or a DML statement must be free of various side effects. Side effects are changes to database tables or public packaged variables declared in the package specification. To ensure that the function is free of side effects, three restrictions apply to stored functions called from SQL statements. A function called from a SQL or DML statement cannot end the current transaction, create or roll back a savepoint, or alter the system or session. A function called from a query statement or from a parallelized DML statement cannot execute a DML statement or modify the database. A function called from a DML statement cannot read or modify the particular table being modified by the DML statement.

Side effects can delay the execution of a query. Side effects can yield order-dependent results, but are not themselves the order-dependent results.

Objective:

More Package Concepts

Sub-Objective:

Identify restrictions on using packaged functions in SQL statements

References:

1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 75 (Ref:1z0-147e.6.6.1)

Which three statements about restrictions on package functions used in SQL? (Choose three.)

- ☐ A function called from a DML statement can read or modify the particular table being modified by the DML statement.
- ☐ A function called from a DML statement cannot read or modify the particular table being modified by the DML statement.
- ☐ A function called from a query statement or from a parallelized DML statement can execute a DML statement or modify the database.
- ☐ A function called from a query statement or from a parallelized DML statement cannot execute a DML statement or modify the database.
- ☐ A function called from a query or DML statement can end the current transaction, create or roll back a savepoint, or alter the system or session.
- ☐ A function called from a query or DML statement cannot end the current transaction, create or roll back a savepoint, or alter the system or session.

Answer:

A function called from a DML statement cannot read or modify the particular table being modified by the DML statement.

A function called from a query statement or from a parallelized DML statement cannot execute a DML statement or modify the database.

A function called from a query or DML statement cannot end the current transaction, create or roll back a savepoint, or alter the system or session.

Explanation:

A function called from a SQL query or a DML statement must be free of various side effects. Side effects are changes to database tables or public packaged variables declared in the package specification. To ensure that the function is free of side effects, three restrictions apply to stored functions called from SQL statements. A function called from a SQL or DML statement cannot end the current transaction, create or roll back a savepoint, or alter the system or session. A function called from a query statement or from a parallelized DML statement cannot execute a DML statement or modify the database. A function called from a DML statement cannot read or modify the particular table being modified by the DML statement.

Objective:

More Package Concepts

Sub-Objective:

Identify restrictions on using packaged functions in SQL statements

References:

1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 101 (Ref:1z0-147e.6.1.1)

Which two statements about package overloading are true? (Choose two.)

- ☐ The subprograms must be local.
- ☐ The subprograms can be local or remote.
- ☐ It allows you to exceed the maximum number of subprograms.
- ☐ Two subprograms with the same name must differ only in return type.
- ☐ Two subprograms with the same name and number of formal parameters must have at least one parameter defined with a different data type.

Answer:

The subprograms must be local.

Two subprograms with the same name and number of formal parameters must have at least one parameter defined with a different data type.

Explanation:

To take advantage of overloading, a package is created with two different subprograms with the same name, but different argument lists. The arguments must differ in number, order, or data type family.

Only local or packaged subprograms can be overloaded.

Objective:

More Package Concepts

Sub-Objective:

Write packages that use the overloading feature

References:

1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 125 (Ref:1z0-147e.6.1.5)

Which types of constructs can be overloaded within a package?

- ○ functions only
- ○ procedures only
- ○ public variables only
- ○ procedures and functions only
- ○ procedures, functions, and public variables

Answer:
procedures and functions only

Explanation:
Specifying two or more constructs with the same name within a package is called overloading. This allows you to execute the same procedure name with different arguments to perform different actions.

Only procedures and functions can be overloaded within a package, variables cannot.

Objective:
More Package Concepts

Sub-Objective:
Write packages that use the overloading feature

References:
1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 138 (Ref:1z0-147e.6.7.2)

Examine this package:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);
FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number;
END theater_pck;

CREATE OR REPLACE PACKAGE BODY theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
v_budget studio.yearly_budget%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END find_seats_sold;

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number
IS
v_yearly_budget NUMBER;
BEGIN
SELECT yearly_budget
INTO v_yearly_budget
FROM studio
WHERE id = v_studio_id;
RETURN v_yearly_budget;
END get_budget;

END theater_pck;
```

The GET_BUDGET function will be executed within a SQL statement. Which compiler directive could you use to check for the purity level of this function during compilation?

- ○ PRAGMA PURITY_LEVEL
- ○ PRAGMA EXCEPTION_INIT
- ○ PRAGMA EXCEPT_REFERENCES
- ○ PRAGMA RESTRICT_REFERENCES

Answer:
PRAGMA RESTRICT_REFERENCES

Explanation:

Certain restrictions apply when using packaged functions within SQL statements. For example, a function cannot INSERT, UPDATE, or DELETE rows if it is to be used in a SQL statement.

Purity level is the guarantee that the packaged function does not violate these restrictions.

Specifying the PRAGMA RESTRICT_REFERENCES in the package specification guarantees that the function does not violate the required restrictions. If the function does not meet these requirements, the package body compilation will fail.

Objective:
More Package Concepts

Sub-Objective:
Invoke packaged functions with SQL

References:
1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 143 (Ref:1z0-147e.6.1.4)

Examine this package specification and execution:

```
CREATE OR REPLACE PACKAGE prod_pack
IS
PROCEDURE order_product
(p1 IN NUMBER, p2 IN CHAR);
PROCEDURE order_product
(p3 IN NUMBER, p4 IN VARCHAR2);
END prod_pack;
```

When executing this packaged procedure, you receive this message:

```
PLS-00307: too many declarations of 'ORDER_PRODUCT' match this call
```

What caused this error?

- ○ Each procedure must have a unique name.
- ○ The names of the arguments must be identical.
- ○ Each argument name must be unique because there are the same number of arguments.
- ○ At least one argument must be in a different data type family because there are the same number of arguments.

Answer:
At least one argument must be in a different data type family because there are the same number of arguments.

Explanation:
Specifying two or more constructs with the same name within a package is called overloading. This allows you to execute the same procedure name with different arguments to perform different actions.

The argument lists must differ in number, order, or data type family to guarantee exclusivity during invocation.

If the argument lists have the same number of arguments, at least one argument position must differ in data type family.

In this case, p2 and p4 are in the same data type family.

Objective:
More Package Concepts

Sub-Objective:
Write packages that use the overloading feature

References:
1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 153 (Ref:1z0-147e.6.5.1)

Examine this package:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
current_avg_cost_per_ticket NUMBER;

PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER);

FUNCTION get_budget
(v_studio_id IN NUMBER)
RETURN number

END theater_pck;
```

You issue this statement in SQL*Plus:

```
EXECUTE theater_pck.current_avg_cost_per_ticket := 10;
```

What is true about the state of CURRENT_AVG_COST_PER_TICKET?

- It is 10 for all user sessions.
- It is 0 due to an illegal reference.
- It is 10 for the duration of the statement.
- It is 10 for the duration of the current session.

Answer:
It is 10 for the duration of the current session.

Explanation:
Package variables are those defined in the specification or in the declaration of the body.

The state of a package variable persists throughout the current user session. Therefore, CURRENT_AVG_COST_PER_TICKET remains 10 for the duration of the current session, unless it is changed or the package is invalidated.

Objective:
More Package Concepts

Sub-Objective:
Identify persistent states in package variables and cursors

References:
1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 157 (Ref:1z0-147e.6.1.2)

Examine this package specification:

```
CREATE OR REPLACE PACKAGE prod_pack
IS
PROCEDURE order_product
(p1 IN NUMBER, p2 IN NUMBER);
PROCEDURE order_product
(p1 IN NUMBER, p2 IN VARCHAR2);
PROCEDURE order_product;
END prod_pack;
```

Which header can be added to this package specification without violating the rules of package overloading?

- ○ PROCEDURE order_product (p1 VARCHAR2);
- ○ PROCEDURE order_product (p3 IN NUMBER, p4 IN NUMBER);
- ○ PROCEDURE order_product (p1 IN NUMBER, p2 IN NUMBER);
- ○ PROCEDURE order_product (p3 IN NUMBER, p4 IN VARCHAR2);

Answer:
PROCEDURE order_product (p1 VARCHAR2);

Explanation:
To take advantage of overloading, a package is created with two different subprograms with the same name, but different argument lists. The arguments must differ in number, order, or data type family.

Subprograms with argument lists that only differ in names are not able to be overloaded, and an error will occur at runtime.

Objective:
More Package Concepts

Sub-Objective:
Write packages that use the overloading feature

References:
1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 170 (Ref:1z0-147e.6.1.3)

Examine this package specification:

```
CREATE OR REPLACE PACKAGE theater_pck
IS
PROCEDURE find_seats_sold
(v_movie_id IN NUMBER, v_theater_id IN NUMBER);
END theater_pck;
```

You want to take advantage of package overloading. Which procedure header fails the rules of overloading?

- ○ PROCEDURE find_seats_sold;
- ○ PROCEDURE find_seats_sold
  (p1 IN NUMBER, p2 IN VARCHAR2);
- ○ PROCEDURE find_seats_sold
  (v_movie_id IN VARCHAR2, v_theater_id IN VARCHAR2);
- ○ PROCEDURE find_seats_sold
  (v_movie_id IN BINARY_INTEGER, v_theater_id IN NUMBER);

Answer:
PROCEDURE find_seats_sold (v_movie_id IN BINARY_INTEGER, v_theater_id IN NUMBER);

Explanation:
To take advantage of overloading, a package is created with two different subprograms with the same name but different argument lists. The arguments must differ in number, order, or data type family.

Subprograms with argument lists that only differ in names are not able to be overloaded and an error will occur at runtime.

BINARY_INTEGER is in the same data type family as NUMBER.

Objective:
More Package Concepts

Sub-Objective:
Write packages that use the overloading feature

References:
1. Oracle9i: Develop PL/SQL Program Units - More Package Concepts

Item: 27 (Ref:1z0-147e.10.6.1)

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER update_studio
BEFORE UPDATE OF yearly_budget ON STUDIO
FOR EACH ROW
DECLARE
v_max_budget NUMBER;
BEGIN
SELECT max(yearly_budget)
INTO v_max_budget
FROM studio;
IF :new.yearly_budget > v_max_budget THEN
:new.yearly_budget := v_max_budget;
END IF;
END;
```

After creating this database trigger successfully, you test it by updating the YEARLY_BUDGET column to a value greater than the maximum value already in the STUDIO table. What result can you expect?

- ○ The STUDIO table is mutating and an error is returned.
- ○ The YEARLY_BUDGET column will be set to the maximum value.
- ○ The STUDIO table is mutating and the trigger execution is ignored.
- ○ Referencing the NEW qualifier in a BEFORE trigger is illegal and an error is returned.

Answer:
The STUDIO table is mutating and an error is returned.

Explanation:
If a row level database trigger attempts to read the same table that the triggering event is on, the table is considered to be mutating. It cannot get a read-consistent query because it is in the middle of the triggering event.

Statement level database triggers execute either right before the event or right after the event so reading the same table as the triggering event table is allowable.

In this case, the database trigger is a row level trigger. It executes when an UPDATE occurs on the YEARLY_BUDGET column of the STUDIO table. During the event, it is trying to read the STUDIO table. This creates the mutation problem and the trigger fails, causing the UPDATE statement to fail.

Objective:
More Trigger Concepts

Sub-Objective:
Describe the cause of a mutating table

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 31 (Ref:1z0-147e.10.3.1)

The code logic of the PREVENT_GROSS_MODIFICATION trigger seems to be incorrect. You have lost the script file that contains the code for this trigger. Which data dictionary view can you query to examine the code for this trigger?

- ○ USER_OBJECTS
- ○ USER_TRIGGERS
- ○ USER_CONSTRUCTS
- ○ USER_SOURCE_CODE

Answer:
USER_TRIGGERS

Explanation:
The source of a database trigger can be found in the TRIGGER_BODY column of the USER_TRIGGERS view.

Objective:
More Trigger Concepts

Sub-Objective:
Create a trigger for a DDL

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 37 (Ref:1z0-147e.10.2.3)

You create two database triggers on two different tables. Both triggers perform the exact same code. What can you do to modularize this functionality?

- ○ Combine the two database triggers into one INSTEAD OF trigger.
- ○ Combine the two database triggers into one statement level trigger.
- ○ Replace the database trigger with a deferred referential constraint.
- ○ Create a procedure with the code, and modify each database trigger to execute the new procedure.

Answer:
Create a procedure with the code, and modify each database trigger to execute the new procedure.

Explanation:
If two different database triggers perform the exact same code, create a procedure with the code and have each database trigger execute the new procedure. Future modifications to the code only occur in one place: the procedure.

Objective:
More Trigger Concepts

Sub-Objective:
Describe events that cause database triggers to fire

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 76 (Ref:1z0-147e.10.6.2)

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER cascade_updates
AFTER UPDATE OF id ON studio
FOR EACH ROW
BEGIN
UPDATE producer
SET studio_id = :new.id
WHERE studio_id = :old.id;
END;
```

The STUDIO_ID column of the PRODUCER table is a foreign key that references the ID column of the STUDIO table. You test it by updating a row in the STUDIO table and changing the ID column. What result can you expect?

- ○ An error will be returned because the STUDIO table is mutating.
- ○ The trigger code will be ignored and, therefore, the PRODUCER table will not be updated.
- ○ The STUDIO_ID column of the PRODUCER table will be updated with the new value of the ID column of the STUDIO table.
- ○ Since the timing is AFTER, this trigger cannot reference the NEW qualifier and, therefore, an error will be returned.

Answer:
An error will be returned because the STUDIO table is mutating.

Explanation:
A parent table is a table that contains a primary key that is referenced by a foreign key of the same table or another table. The table with the foreign key is called the child table.

The only way this trigger will execute successfully is if you drop or disable the foreign key constraint.

Objective:
More Trigger Concepts

Sub-Objective:
Describe the cause of a mutating table

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 79 (Ref:1z0-147e.10.7.3)

Evaluate this code:

```
CREATE OR REPLACE TRIGGER update_studio
BEFORE UPDATE OF yearly_budget ON STUDIO ON THEATER
FOR EACH ROW
BEGIN
...
END;
```

Why does this trigger return an error upon compilation?

- ○ The FOR EACH ROW clause should be FOR EVERY ROW.
- ○ A database trigger cannot be created on more than one table.
- ○ A database trigger cannot be based on an updated column event.
- ○ The name of a database trigger must not use special characters such as '_'.

Answer:
A database trigger cannot be created on more than one table.

Explanation:
"BEFORE UPDATE OF yearly_budget ON STUDIO ON THEATER" is invalid. A database trigger can only be created on one table.

Objective:
More Trigger Concepts

Sub-Objective:
List what triggers can be implemented for

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 80 (Ref:1z0-147e.10.7.4)

The TOTAL_GROSS column of the THEATER table can be derived from the GROSS_RECEIPT and ADVANCED_RECEIPT tables. Which implementation will you use to keep the TOTAL_GROSS column synchronized with changes to the GROSS_RECEIPT and ADVANCED_RECEIPT tables?

- ○ Create one database trigger to update the TOTAL_GROSS whenever the tables are modified.
- ○ Create one package that will execute implicitly to update the TOTAL_GROSS column whenever the tables are modified.
- ○ Create two database triggers, one for each table, to update the TOTAL_GROSS column whenever the tables are modified.
- ○ Add a deferred referential constraint on the TOTAL_GROSS column. This will reference the two tables and will automatically keep the TOTAL_GROSS column synchronized.

Answer:

Create two database triggers, one for each table, to update the TOTAL_GROSS column whenever the tables are modified.

Explanation:
A database trigger can only be created for one table. In this example, a separate database trigger must be created for each table. Then, regardless of which table is modified, the TOTAL_GROSS column of the THEATER table will be recalculated.

Objective:
More Trigger Concepts

Sub-Objective:
List what triggers can be implemented for

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 86 (Ref:1z0-147e.10.9.1)

You own a table named GROSS_RECEIPT that contains sales data. You have disabled certain database triggers on the GROSS_RECEIPT table. Which two data dictionary views can you query to determine the status of all the triggers you have created on this table? (Choose two.)

- ☐ USER_SOURCE
- ☐ USER_STATUS
- ☐ USER_OBJECTS
- ☐ USER_TRIGGERS
- ☐ USER_CONSTRUCTS

Answer:
USER_OBJECTS
USER_TRIGGERS

Explanation:

You can find the status of a trigger that you own by querying the USER_OBJECTS or the USER_TRIGGERS data dictionary view. Both views contain a column called STATUS. In USER_OBJECTS, the possible values for the STATUS column are VALID, INVALID, and N/A. In USER_TRIGGERS, the possible values for the STATUS column are ENABLED or DISABLED.

You could also query the ALL_OBJECTS or the ALL_TRIGGERS data dictionary view to display all triggers that you or other users have created. Or, if you have appropriate privileges, you could query the DBA_OBJECTS or the DBA_TRIGGERS data dictionary view to display all triggers for the GROSS_RECEIPT table.

Objective:

More Trigger Concepts

Sub-Objective:

View trigger information in the data dictionary views

References:

1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 120 (Ref:1z0-147e.10.2.1)

Which event occurs when a database trigger fails?

- ○ The DML event that invoked the database trigger is committed.
- ○ The DML is successful. Each processed row is marked invalid.
- ○ The DML is successful. The modified table is marked invalid.
- ○ The DML event that invoked the database trigger is rolled back.

Answer:
The DML event that invoked the database trigger is rolled back.

Explanation:
If a database trigger fails, the DML operation that invoked the trigger is rolled back.

Objective:
More Trigger Concepts

Sub-Objective:
Describe events that cause database triggers to fire

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 140 (Ref:1z0-147e.10.3.2)

You created the PREVENT_GROSS_MODIFICATION trigger on the GROSS_RECEIPT table. Which command will enable SCOTT the privilege to access this trigger?

- ○ GRANT SELECT, EXECUTE ON gross_receipt TO scott;
- ○ GRANT SELECT ON prevent_gross_modification TO scott;
- ○ GRANT EXECUTE ON prevent_gross_modification TO scott;
- ○ GRANT SELECT, INSERT, UPDATE, DELETE ON gross_receipt TO scott;

Answer:
GRANT SELECT, INSERT, UPDATE, DELETE ON gross_receipt TO scott;

Explanation:

Database triggers are implicitly executed by the trigger event. To execute a trigger, you must perform one of these events. Users require SELECT, INSERT, UPDATE, and DELETE privileges on tables they do not own. Granting these privileges to a user allows the user to implicitly execute the trigger by modifying the table.

Objective:

More Trigger Concepts

Sub-Objective:

Create a trigger for a DDL

References:

1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 145 (Ref:1z0-147e.10.1.1)

The TOTAL_GROSS column of the THEATER table can be derived from the GROSS_RECEIPT table. To keep the value of this column consistent during modifications to the GROSS_RECEIPT table, which type of construct will you create?

- ○ package
- ○ function
- ○ procedure
- ○ database trigger

Answer:
database trigger

Explanation:

A common and beneficial use for database triggers is maintaining derived columns. The TOTAL_GROSS column of the THEATER table is a derived column. It can be calculated with data queried from another table in the database. The problem with derived columns is that if a change is made to the data of the other table, the values of the derived column are not consistent.

The total gross per theater can be calculated by querying the GROSS_RECEIPT table. A database trigger can be created on this table to automatically update the TOTAL_GROSS column of the THEATER table.

Although you could place this logic in a Forms trigger, the derived column will only be updated the Forms application. Database triggers ensure that the code will be executed wherever the event occurs, regardless from which application.

Check constraints cannot contain SELECT statements.

Packaged subprograms must be executed explicitly and are not feasible for this situation.

Objective:

More Trigger Concepts

Sub-Objective:

Define what a database trigger is

References:

1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 158 (Ref:1z0-147e.10.7.2)

Examine these two tables:

THEATER

ID
NAME
ADDRESS
TOTAL_GROSS

GROSS_RECEIPT

MOVIE_ID
THEATER_ID
SHOW_DATE
SEATS_SOLD
COST_PER_TICKET

For every movie shown, the number of seats sold and cost per ticket is multiplied resulting in the gross sales for that showing. The THEATER contains a column called TOTAL_GROSS that must be updated with this new amount. If you want TOTAL_GROSS to be automatically updated regardless of which application updates the GROSS_RECEIPT table, which type of construct will you create?

- ○ forms trigger
- ○ database trigger
- ○ package private procedure
- ○ check constraint on seats_sold and cost_per_ticket

Answer:
database trigger

Explanation:
A common and beneficial use for database triggers is maintaining derived columns. The TOTAL_GROSS column of the THEATER table is a derived column. It can be calculated with data queried from another table in the database. The problem with derived columns is that if a change is made to the data of the other table, the values of the derived column are not consistent.

The total gross per theater can be calculated by querying the GROSS_RECEIPT table. A database trigger can be created on this table to automatically update the TOTAL_GROSS column of the THEATER table.

Although you could place this logic in a Forms trigger, the derived column will only be updated from the Forms application. Database triggers will execute from every application that performs the event that defines the trigger.

Check constraints cannot contain SELECT statements.

Packaged subprograms must be executed explicitly and are not feasible for this situation.

Objective:
More Trigger Concepts

Sub-Objective:
List what triggers can be implemented for

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 160 (Ref:1z0-147e.10.2.2)

Modifications to the THEATER table are not allowed during the last week in December. When creating a database trigger to enforce this rule, which trigger type will you use to be most efficient?

- ○ row
- ○ weekly
- ○ statement
- ○ Oracle Forms

Answer:
statement

Explanation:
Statement level database triggers fire once for the entire triggering event. By default, triggers are statement level. Creating a row level trigger requires the FOR EACH ROW clause.

In this case, it should be a statement level trigger because you are going to stop the entire event if it is the last week in December. A row level trigger would execute for every row processed during the event, checking to see if it is the last week in December.

Objective:
More Trigger Concepts

Sub-Objective:
Describe events that cause database triggers to fire

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Item: 171 (Ref:1z0-147e.10.7.1)

Examine this database trigger:

```
CREATE OR REPLACE TRIGGER cascade_updates
AFTER UPDATE OF id ON studio
FOR EACH ROW
BEGIN
UPDATE producer
SET studio_id = :new.id
WHERE studio_id = :old.id;
END;
```

For this database trigger to execute successfully, which type of constraint must NOT exist?

- ○ a check constraint on the STUDIO_ID column of the PRODUCER table
- ○ a not null constraint on the STUDIO_ID column of the PRODUCER table
- ○ a foreign key constraint on the STUDIO_ID column of the PRODUCER table referencing the ID column of the STUDIO table
- ○ a foreign key constraint on the STUDIO_ID column of the PRODUCER table referencing the ID column of the PRODUCER table

Answer:
a foreign key constraint on the STUDIO_ID column of the PRODUCER table referencing the ID column of the STUDIO table

---

Explanation:
A parent table is a table that contains a primary key that is referenced by a foreign key of the same table or another table. The table with the foreign key is called the child table. The only way this trigger will execute successfully is if you drop or disable the foreign key constraint.

Objective:
More Trigger Concepts

Sub-Objective:
List what triggers can be implemented for

References:
1. Oracle9i: Develop PL/SQL Program Units - More Trigger Concepts

Oracle Supplied Packages
Item: 12 (Ref:1z0-147e.7.4.9)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE manage_temp_tables
IS
BEGIN
DELETE FROM temp_table1;
DROP TABLE temp_table2;
END;
```

Why would you receive an error during compilation?

- The DROP command within a PL/SQL construct must be executed using the DBMS_DROP package.
- Data definition commands within a PL/SQL construct must be executed using the DBMS_SQL package.
- Data manipulation commands within a PL/SQL construct must be executed using the DBMS_SQL package.
- Data manipulation commands within a PL/SQL construct must be executed using the DBMS_DML package.

Answer:
Data definition commands within a PL/SQL construct must be executed using the DBMS_SQL package.

Explanation:
Data Definition Language (DDL) commands cannot be executed from within a PL/SQL construct unless you use the DBMS_SQL package.

Data Manipulation Language (DML) commands can be executed directly without the need of a package.

Objective:
Oracle Supplied Packages

Sub-Objective:
Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:
1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 25 (Ref:1z0-147e.7.4.6)

Which procedure of the DBMS_OUTPUT supplied package allows you to place messages in a buffer to be displayed at a later time?

- ○ PUT
- ○ PLACE_LINE
- ○ OUTPUT_LINE
- ○ CREATE_LINE

Answer:
PUT

Explanation:

DBMS_OUTPUT is an Oracle supplied package that allows you to display messages during a SQL*Plus session. PUT is the procedure within this package that allows you to add text to the buffer only. You can then display the contents of the buffer using the PUT_LINE or NEW_LINE procedure. PUT_LINE is a procedure within this package that places a line of text into a buffer and then displays the contents of the buffer to the screen. NEW_LINE is a procedure within this package that places an end-of-line marker in the output buffer.

To view the results of the DBMS_OUTPUT package in SQL*Plus, you must first issue the SET SERVEROUTPUT ON command.

Objective:

Oracle Supplied Packages

Sub-Objective:

Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:

1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 82 (Ref:1z0-147e.7.4.10)

Which two Oracle supplied packages can you use to perform DDL commands within a PL/SQL program unit? (Choose two.)

- ☐ DBMS_SQL
- ☐ DBMS_JOB
- ☐ DBMS_DDL
- ☐ DBMS_PIPE
- ☐ DBMS_OUTPUT

Answer:
DBMS_SQL
DBMS_DDL

Explanation:
DBMS_SQL is an Oracle supplied package that allows you to perform Data Definition Language commands (DDL) within a PL/SQL construct. It also allows you to create dynamic SQL.

DBMS_DDL is an Oracle supplied package that allows you to perform certain Data Definition Language (DDL) commands within a PL/SQL construct.

Objective:
Oracle Supplied Packages

Sub-Objective:
Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:
1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 115 (Ref:1z0-147e.7.4.1)

Which Oracle supplied package allows you to send a message to another user of the same session?

- ○ DBMS_SQL
- ○ DBMS_JOB
- ○ DBMS_PIPE
- ○ DBMS_MESSAGE

Answer:
DBMS_PIPE

Explanation:

DBMS_PIPE is the Oracle supplied package that allows two or more sessions connected to the same instance to communicate through a pipe.

You use several subprograms within this package to create, send, accept, and read the contents of a pipe.

Objective:

Oracle Supplied Packages

Sub-Objective:

Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:

1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 117 (Ref:1z0-147e.7.3.1)

Which function of the DBMS_SQL package returns the total number of rows affected by the last operation?

- ◯ PARSE
- ◯ EXECUTE
- ◯ FETCH_ROWS
- ◯ OPEN_CURSOR

Answer:
EXECUTE

Explanation:

The EXECUTE function of the DBMS_SQL package executes the SQL statement and returns the number of rows processed.

The PARSE procedure of the DBMS_SQL package immediately parses the SQL statement specified.

The FETCH_ROWS function of the DBMS_SQL package fetches a row or rows from an open cursor.

The OPEN_CURSOR function of the DBMS_SQL package opens a cursor, obtaining a pointer to memory.

Objective:

Oracle Supplied Packages

Sub-Objective:

Use EXECUTION IMMEDIATE

References:

1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 121 (Ref:1z0-147e.7.4.2)

After adding a new column to the EMP table, you must recompile all dependent PL/SQL constructs. Which package/procedure can you invoke?

- ○ DBMS_COMPILE.MODIFIED
- ○ DBMS_SQL.ALTER_COMPILE
- ○ DBMS_RECOMPILE.INVALID
- ○ DBMS_DDL.ALTER_COMPILE

Answer:
DBMS_DDL.ALTER_COMPILE

Explanation:

DBMS_DDL is an Oracle supplied package that allows you to perform Data Definition Language (DDL) commands within a PL/SQL construct.

ALTER_COMPILE is one procedure of this package that allows you to compile PL/SQL constructs.

Objective:

Oracle Supplied Packages

Sub-Objective:

Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:

1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 122 (Ref:1z0-147e.7.4.5)

You are debugging a PL/SQL program unit in SQL*Plus. When using DBMS_OUTPUT.PUT_LINE, you do not see the output on the screen. Which command must you execute to see the results of DBMS_OUTPUT.PUT_LINE?

- SET ECHO ON
- SET VERIFY ON
- SET FEEDBACK ON
- SET SERVEROUTPUT ON

Answer:
SET SERVEROUTPUT ON

Explanation:
DBMS_OUTPUT is an Oracle supplied package that allows you to display messages during a SQL*Plus session. To view the execution results of DBMS_OUTPUT in SQL*Plus, you must first issue the SET SERVEROUTPUT ON command.

Objective:
Oracle Supplied Packages

Sub-Objective:
Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:
1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 131 (Ref:1z0-147e.7.4.4)

Which Oracle supplied package allows you to create automated, unattended scheduling of PL/SQL program units?

- ○ DBMS_JOB
- ○ DBMS_DDL
- ○ DBMS_PIPE
- ○ DBMS_OUTPUT

Answer:
DBMS_JOB

Explanation:
DBMS_JOB is an Oracle supplied package that allows you to create jobs or schedules. These jobs are created by specifying the time to execute, iteration, and the action or actions to perform.

Objective:
Oracle Supplied Packages

Sub-Objective:
Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:
1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 135 (Ref:1z0-147e.7.4.8)

A procedure must insert rows into a table. The name of this table is not known until runtime. Which Oracle supplied package must you use to accomplish this task?

- ○ DBMS_SQL
- ○ DBMS_PIPE
- ○ DBMS_INSERT
- ○ DBMS_DYNAMIC

Answer:
DBMS_SQL

Explanation:
DBMS_SQL is an Oracle supplied package that allows you to perform Data Definition Language commands (DDL) within a PL/SQL construct. It also allows you to create dynamic SQL.

Dynamic SQL is a statement that is not complete in the source code. It is not completed until runtime. For example, the name of a table could be passed through a parameter and used to complete the INSERT statement within this procedure.

Objective:
Oracle Supplied Packages

Sub-Objective:
Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:
1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 137 (Ref:1z0-147e.7.4.3)

The DOWNLOAD_CORE_DATA procedure must be invoked each day at 7:00 am. Which action will you take to perform this task?

- Invoke the procedure manually each day at 7:00 am.
- Create a job to execute this procedure daily using the DBMS_JOB package.
- Create a job to execute this procedure daily using the DBMS_SQL package.
- Specify the PRAGMA RUN_DAILY compiler directive in the procedure's declaration.

Answer:
Create a job to execute this procedure daily using the DBMS_JOB package.

Explanation:
DBMS_JOB is the Oracle supplied package that allows you to execute procedures or schedule tasks to execute intermittently.

You could invoke the procedure manually each day at 7:00 am, but it is not the best answer for this scenario.

Objective:
Oracle Supplied Packages

Sub-Objective:
Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:
1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 144 (Ref:1z0-147e.7.3.2)

When using DBMS_SQL to create dynamic SQL, which processing phase will check the validity of the statement?

- ○ bind
- ○ parse
- ○ fetch
- ○ execute

Answer:
parse

Explanation:

When a SQL statement is processed, it must pass through the parse, bind, execute, and fetch phases.

During the parse phase, the statement is checked for syntax errors and validity. All object references are resolved and the user's privileges to those objects are checked.

Objective:

Oracle Supplied Packages

Sub-Objective:

Use EXECUTION IMMEDIATE

References:

1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Item: 172 (Ref:1z0-147e.7.4.7)

When invoking the Oracle supplied DBMS_SQL package, under which user account are the operations executed?

- ○ SYS only
- ○ SYSTEM only
- ○ current user only
- ○ SYS and current user only

Answer:
current user only

Explanation:
Operations performed with DBMS_SQL are executed under the current user. They are not executed under the SYS or SYSTEM accounts.

Objective:
Oracle Supplied Packages

Sub-Objective:
Describe the use and application of some Oracle server-supplied packages: DBMS_DDL, DBMS_JOB, Submit Jobs, DBMS_OUTPUT, UTL_FILE, UTL_HTTP, and UTL_TCP

References:
1. Oracle9i: Develop PL/SQL Program Units - Oracle Supplied Packages

Overview of PL/SQL Programs

Item: 2 (Ref:1z0-147e.1.4.1)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE find_seats_sold
(v_movie_id IN NUMBER DEFAULT 34, v_theater_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id
AND theater_id = v_theater_id;
END;
```

Which command will successfully invoke this procedure in SQL*Plus?

- ○ EXECUTE find_seats_sold;
- ○ RUN find_seats_sold (v_theater_id => 500, v_movie_id => 34);
- ○ EXECUTE find_seats_sold (v_theater_id => 500, v_movie_id => 34);
- ○ EXECUTE find_seats_sold (v_theater_id := 500, v_movie_id := 34);

Answer:
EXECUTE find_seats_sold (v_theater_id => 500, v_movie_id => 34);

Explanation:
You can specify argument values using the positional or named method. The named method requires the use of the "=>" operator to specify a value for each argument.

The named method:
EXECUTE find_seats_sold (v_theater_id => 500, v_movie_id => 34);

The positional method:
EXECUTE find_seats_sold (500, 34);

The value of 500 is assigned to the first argument listed in the procedure header and 34 is assigned to the second argument.

Objective:
Overview of PL/SQL Programs

Sub-Objective:
Describe how a stored procedure/function is invoked

References:
1. Oracle9i: Develop PL/SQL Program Units - Overview of PL/SQL Subprograms

Item: 3 (Ref:1z0-147e.1.1.1)

Which type of construct should you create to solely perform an action without returning a value?

- ○ view
- ○ function
- ○ procedure
- ○ packaged function

Answer:
procedure

Explanation:
Procedures are usually created to perform an action without returning a value. Procedures can return a value using an OUT argument. Functions must return a value.

Objective:
Overview of PL/SQL Programs

Sub-Objective:
Describe a PL/SQL program construct

References:
1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 4 (Ref:1z0-147e.1.3.1)

For which reason might you create a subprogram within a procedure?

- ○ to bypass a pragma restriction
- ○ to bypass the CURRENT AUTHID restriction
- ○ to allow execution within a SQL statement
- ○ to store a repeating block of code once without creating a separate construct

Answer:
to store a repeating block of code once without creating a separate construct

---

Explanation:

Subprograms allow you to create just one occurrence of a piece of code that must be executed in different locations of a procedure.

Use subprograms when the code is only executed within the procedure. If the code is executed from outside the procedure, then the subprogram should be written as a packaged or stand-alone procedure instead.

Example: (calc_comm is the subprogram)

```
CREATE OR REPLACE PROCEDURE update_employee
(v_emp_id IN NUMBER)
IS
v_comm NUMBER;

PROCEDURE calc_comm
IS
v_total NUMBER;
BEGIN
SELECT SUM(ord.total)
INTO v_total
FROM ord,customer
WHERE ord.custid = customer.custid
AND customer.repid = v_emp_id;
v_comm := v_total * .20;
END calc_comm;

BEGIN
...
calc_comm;
...
calc_comm;
...
calc_comm;
END;
```

Objective:

Overview of PL/SQL Programs

Sub-Objective:

List the benefits of subprograms

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 10 (Ref:1z0-147e.1.2.1)

The UPDATE_EMPLOYEE procedure contains an algorithm that calculates an employee's commission multiple times throughout the program. If a change is made to the algorithm, the change must be made multiple times. How can this procedure be modified to simplify the code and reduce duplicated code?

- ○ Add an algorithm exception handler.
- ○ Create a library containing the algorithm.
- ○ Add a local subprogram containing the algorithm.
- ○ Create multiple anonymous blocks containing the algorithm.

Answer:
Add a local subprogram containing the algorithm.

Explanation:

Subprograms allow you to create just one occurrence of a piece of code that must be executed in different locations of a procedure.

Use local subprograms when the code is only executed within the procedure. If the code will be executed from outside the procedure, then the subprogram should be written as a packaged or stand-alone procedure instead.

Example: (calc_comm is the subprogram)

```
CREATE OR REPLACE PROCEDURE update_employee
(v_emp_id IN NUMBER)
IS
v_comm NUMBER;

PROCEDURE calc_comm
IS
v_total NUMBER;
BEGIN
SELECT SUM(ord.total)
INTO v_total
FROM ord,customer
WHERE ord.custid = customer.custid
AND customer.repid = v_emp_id;
v_comm := v_total * .20;
END calc_comm;

BEGIN
...
calc_comm;
...
calc_comm;
...
calc_comm;
END;
```

Objective:

Overview of PL/SQL Programs

Sub-Objective:

List the components of a PL/SQL block

References:

1. Oracle9i: Develop PL/SQL Program Units - Creating Procedures

Item: 54 (Ref:1z0-147e.1.4.2)

Examine this procedure:

```
CREATE OR REPLACE PROCEDURE find_seats_sold
(v_movie_id IN NUMBER)
IS
v_seats_sold gross_receipt.seats_sold%TYPE;
BEGIN
SELECT seats_sold
INTO v_seats_sold
FROM gross_receipt
WHERE movie_id = v_movie_id;
END;
```

Which command will successfully invoke this procedure in SQL*Plus?

- ○ RUN find_seats_sold(34);
- ○ EXECUTE find_seats_sold;
- ○ EXECUTE find_seats_sold (34);
- ○ find_seats_sold ('Riverplace Theater');

Answer:
EXECUTE find_seats_sold (34);

Explanation:

Executing a procedure in SQL*Plus requires the EXECUTE command. This procedure has one IN argument. Therefore, use this command:

```
EXECUTE find_seats_sold (34);
```

Objective:

Overview of PL/SQL Programs

Sub-Objective:

Describe how a stored procedure/function is invoked

References:

1. Oracle9i: Develop PL/SQL Program Units - Overview of PL/SQL Subprograms

Item: 96 (Ref:1z0-147e.1.4.3)

When invoking a procedure, you can specify the arguments using the positional method by listing the values in the order of the argument list. Which method would you use to list values in an arbitrary order?

- ○ FIFO
- ○ list
- ○ type
- ○ named

Answer:
named

Explanation:

You can specify argument values using the positional or named method. The named method requires the use of the "=>" operator to specify a value for each argument and allows for an arbitrary assignment of values.

The named method:
EXECUTE find_seats_sold (v_theater_id => 500, v_movie_id => 34);

The positional method:
EXECUTE find_seats_sold (500, 34);

The value of 500 is assigned to the first argument listed in the procedure header and 34 is assigned to the second argument.

Objective:

Overview of PL/SQL Programs

Sub-Objective:

Describe how a stored procedure/function is invoked

References:

1. Oracle9i: Develop PL/SQL Program Units - Overview of PL/SQL Subprograms