PassQueen
a.k.a dumpsqueen.com

Oracle Database 19c: Program with PL/SQL

Oracle 1z0-149

Version Demo

Total Demo Questions: 10

Total Premium Questions: 107

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, PL/SQL Program Constructs	40
Topic 2, Managing PL/SQL Subprograms	18
Topic 3, Managing PL/SQL Packages	12
Topic 4, Managing Dependencies	4
Topic 5, Managing Errors	16
Topic 6, Using Dynamic SQL	3
Topic 7, Using Advanced Interface Methods	2
Topic 8, Working with Collections	6
Topic 9, Working with Cursors	6
Total	107

QUESTION NO: 1

Examine this table in the SH schema:

```
DESC products

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 PDT_ID                                    NOT NULL NUMBER
 PDT_NAME                                           VARCHAR2(10)
 PRICE                                              NUMBER
```

User SH executes this code:

```
DECLARE
  v_price NUMBER := 1000;
  v_pdt_name VARCHAR2(15);
BEGIN
   SELECT pdt_name INTO v_pdt_name
     FROM products
     WHERE price = v_price;

  ---placeholder

END;
/
```

The program must terminate with a user-defined message and no rows displayed if more than one product's price is 1000.

With which option must “---placeholder” be replaced?

A.

B.

C.

D.

E.

ANSWER: A

Explanation:

The correct replacement is the code shown in the first image option because it ensures that the single-row query detects multiple matching products before the program can display a product row. In PL/SQL, a `SELECT ... INTO` statement is intended to return exactly one row. If the predicate matching a price of 1000 finds more than one row, Oracle raises the predefined `TOO_MANY_ROWS` exception (associated with ORA-01422). The exception handler can then invoke `RAISE_APPLICATION_ERROR` to terminate execution with an application-specific message and error number in the permitted application range. Because normal processing does not proceed past the failing query, the output statement that would display product data is never reached; therefore, no product rows are displayed. This approach uses Oracle's built-in cardinality enforcement for single-row queries and provides a controlled, meaningful application error rather than allowing an unhandled Oracle error to be returned. Oracle documents the behavior of `SELECT INTO`, including the exception raised when more than one row is returned, in its PL/SQL static SQL documentation. The permitted use and behavior of `RAISE_APPLICATION_ERROR` are documented in the PL/SQL exception-handling documentation.

QUESTION NO: 2

Which code will successfully create a BODILESS PACKAGE to standardize CONSTANTS and EXCEPTIONS declarations?

A)

```
CREATE OR REPLACE PACKAGE std_const_err_pkg IS
vtax CONSTANT NUMBER(3):=3;
e_seq EXCEPTION;
PRAGMA EXCEPTION_INIT( -2277,e_seq);
e_fk EXCEPTION;
PRAGMA EXCEPTION_INIT( -2292,e_fk);
End;
```

B)

```
CREATE OR REPLACE PACKAGE std_const_err_pkg IS
vtax CONSTANT NUMBER(3);
e_seq Exception;
PRAGMA EXCEPTION_INIT(e_seq,-2277);
e_fk EXCEPTION;
PRAGMA EXCEPTION_INIT(e_fk, -2292);
End;
```

C)

```
CREATE OR REPLACE PACKAGE std_const_err_pkg IS
vtax CONSTANT NUMBER(3):=3;
e_seq Exception;
PRAGMA EXCEPTION_INIT(e_seq,-2277);
e_fk EXCEPTION;
PRAGMA EXCEPTION_INIT(e_fk, -2292);
End;
```

D)

```
CREATE OR REPLACE PACKAGE std_const_err_pkg IS
vtax NUMBER(3) CONSTANT:=3;
e_seq Exception;
PRAGMA EXCEPTION_INIT(e_seq,-2277);
e_fk EXCEPTION;
PRAGMA EXCEPTION_INIT(e_fk, -2292);
End;
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: C

Explanation:

A package can be created without a package body when its specification contains only declarations that do not require executable implementations. This is appropriate for a shared standards package whose purpose is to expose named constants and user-defined exceptions for use by other PL/SQL program units. The correct package specification uses `CREATE PACKAGE`, declares each constant with its datatype and required initial value, declares exceptions in the package specification, and closes the declaration with `END package_name;`. Once compiled, the package members are referenced by qualified names such as `package_name.constant_name` or `package_name.exception_name`. Because constants and exception names are declarations rather than subprogram implementations, no `CREATE PACKAGE BODY` statement is needed. Oracle documents that a package body is optional when the package specification contains no subprograms or cursors requiring a body. This pattern centralizes application-wide values and exception definitions while making their names available to any PL/SQL unit granted access to the package. See Oracle's PL/SQL Packages documentation and the package specification reference.

QUESTION NO: 3

Which three are true about PL/SQL subprograms? (Choose three.)

A. Results of a subprogram can be cached in the SGA such that sessions connected to the same instance can reuse these results when available.

B. Users granted execute privilege on a procedure compiled with definer's rights require grants to access objects belonging to the definer that are referenced in the procedure.

C. Subprograms are cached by default and shared among users, thereby reducing memory requirements.

D. Reuse of parsed PL/SQL code from the shared SQL area reduces parsing overhead.

E. A subprogram's session state is retained even if any of the session's instantiated subprograms are invalidated and revalidated.

F. Host variables can be referenced inside any PL/SQL subprogram.

G. A PL/SQL procedure can invoke an external code block written in a different programming language.

ANSWER: C D G

Explanation:

Stored PL/SQL subprograms are cached by default in the shared pool and can be shared by users of the same database instance. This shared representation avoids maintaining separate compiled copies for each user and therefore helps reduce memory consumption. Oracle describes stored program units as being stored in the database and managed through the shared pool and library cache.

Reuse of parsed PL/SQL code from the shared SQL area reduces parsing overhead because Oracle can use an existing, valid cached representation rather than repeatedly parsing and compiling the same program unit. This library-cache behavior is an important performance benefit of stored PL/SQL, particularly for frequently executed procedures, functions, packages, and triggers.

A PL/SQL procedure can invoke an external code block written in another programming language through Oracle external subprogram support. An external procedure or function can be implemented in a language such as C and declared for invocation from PL/SQL; Oracle then uses the appropriate external-call mechanism to execute that implementation. See Oracle's PL/SQL Language Reference overview and its documentation on external subprograms.

QUESTION NO: 4

Which two are true about using the ACCESSIBLE BY clause? (Choose two.)

A. It can be used in the declaration of object types.

B. It must be specified in the heading of a package specification.

C. The check is enforced by this clause for direct access and access through dynamic SQL.

D. It can be used for individual procedures and functions declared in a package specification.

E. It must be specified in the heading of a package body.

ANSWER: A B

Explanation:

`ACCESSIBLE BY` is a compile-time access-control feature for PL/SQL program units. It can be attached to an object-type declaration, so that the type's use is limited to the named accessor units. This supports encapsulation by allowing an application to expose a type only to trusted packages, procedures, functions, or other permitted units. Oracle documents object types among the schema objects for which this clause can be specified.

When access is being restricted for a package, the clause belongs in the package declaration header in the package specification. The package specification is the public declaration of the package and is where Oracle records the list of units authorized to access that package. A package body implements the package but is not the location for declaring this package-level access restriction. The restriction is validated by Oracle during PL/SQL compilation for statically resolvable references, helping establish intended internal APIs and prevent unauthorized direct dependencies between application program units.

See Oracle's ACCESSIBLE BY clause reference and the PL/SQL packages documentation.

QUESTION NO: 5

Which three are true about functions and procedures? (Choose three.)

A. The ACCESSIBLE BY clause can be used only for procedures.

B. In a function, every execution path must lead to a RETURN statement.

C. Both can have only constants as actual parameters for IN mode parameters.

D. Both can be invoked from within SQL statements.

E. In a procedure the RETURN statement cannot specify an expression.

F. In a function every RETURN statement must specify an expression.

ANSWER: B E F

Explanation:

In a function, every execution path must lead to a RETURN statement because a PL/SQL function is designed to return a value to its caller. The function body must contain a RETURN statement that supplies a value of the declared return datatype, and execution must not be allowed to complete without returning that value. If control reaches the end of a function without executing a RETURN statement, PL/SQL raises the runtime error `ORA-06503: PL/SQL: Function returned without value`. This makes ensuring a value-returning path essential when writing conditional logic in a function.

In a procedure the RETURN statement cannot specify an expression. A procedure can use `RETURN;` to immediately stop execution and return control to its caller, but procedures do not return a value through a RETURN statement. Values produced by a procedure are instead commonly passed back through `OUT` or `IN OUT` parameters.

In a function every RETURN statement must specify an expression. That expression provides the function result and must be compatible with the datatype declared in the function header's `RETURN` clause. Oracle documents the distinct RETURN-

statement rules for functions and procedures in its PL/SQL language reference: RETURN Statement and PL/SQL Subprograms.

QUESTION NO: 6

Examine these statements:

```
Drop procedure calling_proc;

CREATE OR REPLACE PROCEDURE protected_proc
  ACCESSIBLE BY (calling_proc)
AS
BEGIN
  DBMS_OUTPUT.put_line('TEST1 : protected_proc');
END;
```

Which is true?

A. It will result in a compilation error for protected_proc because calling_proc does not exist.

B. It will result in a compilation error for protected_proc because calling_proc must be prefixed with the schema name.

C. It will result in a successful compilation because objects referenced in an ACCESSIBLE BY clause are not checked at compile time.

D. With adequate privileges, PROTECTED_PROC procedure can be called by other programs apart from CALLING_PROC.

ANSWER: C

Explanation:

It will result in a successful compilation because objects referenced in an ACCESSIBLE BY clause are not checked at compile time. The `ACCESSIBLE BY` clause protects a stored PL/SQL unit by restricting which named PL/SQL units can invoke it. However, Oracle does not validate the existence of the units named as accessors when it compiles the protected procedure. Therefore, `protected_proc` can be compiled even if `calling_proc` has not yet been created at that point in the script. This allows dependent program units to be created in a convenient deployment order, including creating the protected unit before the unit that will later call it.

Once `calling_proc` exists and invokes `protected_proc`, Oracle applies the accessibility restriction based on the stored unit identity and permits the call when it matches the declaration in `ACCESSIBLE BY`. This feature is intended to enforce encapsulation among PL/SQL program units; it is not an existence check performed while compiling the protected unit. Oracle documents that accessors named in an `ACCESSIBLE BY` clause are not checked for existence at compilation time. See the Oracle Database 19c ACCESSIBLE BY clause documentation and the PL/SQL subprograms documentation.

QUESTION NO: 7

Examine these statements and output:

```
CONNECT ora1/ora1@pdb1

CREATE OR REPLACE PROCEDURE proc1( v1 OUT NUMBER) is
BEGIN
DBMS_OUTPUT.PUT_LINE(v1*10);
END;
/
```

The procedure is created successfully.

```
GRANT EXECUTE ON proc1 TO ora2;
 grant succeeded.
```

User ora2 has password ora2 in pdb1.

Which script will execute successfully?

A)

```
CONNECT ora2/ora2@pdb1
SET SERVEROUTPUT ON
/
BEGIN
  EXEC ora1.proc1;
END;
```

B)

```
CONNECT ora2/ora2@pdb1
SET SERVEROUTPUT ON
DECLARE
x NUMBER:=5;
BEGIN
  ora1.proc1(x);
END;
```

C)

```
CONNECT ora2/ora2@pdb1
SET SERVEROUTPUT ON
DECLARE
x NUMBER:=5;
BEGIN
  x:=proc1;
DBMS_OUTPUT.PUT_LINE(x)
END;
```

D)

```
CONNECT ora2/ora2@pdb1
SET SERVEROUTPUT ON
/
BEGIN
  EXEC proc1;
END;
```

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: B

Explanation:

The script represented by `Option B` is the successful one. The relevant principle is that a stored PL/SQL procedure executes in the database and container context established by its definition and by the session that invokes it. In a multitenant database, a local user such as `ora2` exists only within its specified PDB, so the connection and execution steps must target `pdb1` and use privileges applicable in that PDB. The successful script establishes the required PDB context and invokes the already compiled procedure using the appropriate user/session context. Oracle validates the procedure when it is compiled, but privileges and name resolution relevant to execution are applied according to whether the unit is definer-rights or invoker-rights and according to the current database container. This is why correct connection targeting and execution context are essential when a local PDB user is involved. Oracle documents PL/SQL execution-rights behavior, including invoker-rights units, in the PL/SQL Language Reference. The scope and behavior of local users in a PDB are described in the Oracle Multitenant Administration Guide.

QUESTION NO: 8

Which two are true about exception handling? (Choose two.)

A. Internally defined exceptions can be handled only by the OTHERS exception handler.

B. All declared exceptions are raised implicitly by the runtime system.

C. User-defined exceptions can be defined in the declarative part of any PL/SQL anonymous block, subprogram, or package.

D. Only predefined exceptions and user-defined exceptions can have a user-declared name associated with them.

E. Predefined exceptions are globally declared in the standard package.

ANSWER: C E

Explanation:

User-defined exceptions can be declared in the declarative section of an anonymous PL/SQL block, a subprogram, or a package. Declaring an exception creates a named exception that application code can raise explicitly with the `RAISE` statement and handle in an exception section. A package declaration can also expose such an exception to other program

units, while an exception declared in a local block or subprogram has the scope of that program unit. This provides a clear way to model and handle application-specific error conditions.

Predefined exceptions are also correctly described as globally declared in the `STANDARD` package. PL/SQL makes these names available automatically, so code can reference and handle common Oracle errors, such as `NO_DATA_FOUND`, `TOO_MANY_ROWS`, and `ZERO_DIVIDE`, without first declaring them. Oracle documents that predefined exceptions have predefined names and that their declarations are supplied through the globally available `STANDARD` package. This built-in availability lets exception handlers use the meaningful predefined names directly wherever PL/SQL exception handling is supported. See Oracle's PL/SQL Error Handling documentation and its Predefined Exceptions reference.

QUESTION NO: 9

Which three are true about anonymous blocks and subprograms? (Choose three.)

A. Named subprograms cannot be called from other packages.

B. PROCEDURE subprograms can accept parameters.

C. A FUNCTION subprogram must return a value.

D. Anonymous blocks cannot use packaged variables.

E. Named subprograms are stored in the database server.

F. Anonymous blocks must always start with the Declare keyword.

G. FUNCTION subprograms must be called and passed through one or more parameters.

ANSWER: B C E

Explanation:

PROCEDURE subprograms can accept parameters because a procedure declaration can define formal parameters in its parameter list. These parameters provide the interface through which a caller supplies input, receives output, or both, using the supported parameter modes. A procedure performs an action but does not return a value as its invocation result.

A FUNCTION subprogram must return a value. Its declaration specifies a return datatype, and its executable section must contain a `RETURN` statement that returns a value compatible with that datatype. This return value lets a function be used in expressions and assignments, subject to the applicable PL/SQL and SQL rules.

Named subprograms are stored in the database server when created as stored procedures or functions, either as standalone schema objects or as package components. Oracle stores their compiled form and metadata in the database, allowing appropriately privileged users and applications to invoke them later. This persistence distinguishes named stored subprograms from anonymous blocks, which are submitted for execution without being stored as reusable program units. Oracle documentation describes procedure parameters, function return values, and stored PL/SQL units in the PL/SQL Language Reference: Subprograms and PL/SQL Language Fundamentals.

QUESTION NO: 10

Examine the structure of the ora1.depts table:

```
Column Name      Null  Type
-----------      ----  ----
DEPARTMENT_ID    NOT NULL NUMBER(4)
DEPARTMENT_NAME  NOT NULL VARCHAR2(30)
MANAGER_ID               NUMBER(6)
LOCATION_ID              NUMBER(4)
```

Now, examine these statements issued by user ora1 which execute successfully:

Create or replace view dep_vu as select * from depts;

Alter table depts add dep_email varchar2(20);

Finally, examine this block of code executed by user ora1:

```
set serveroutput on

declare
x number;
begin
SELECT count(*)
into x colCount
FROM all_tab_columns
WHERE table_name = 'DEP_VU' and
      owner='ORA1';
dbms_output.put_line(x);
end;
/
```

Which is true?

A. DEP_VU must be manually recompiled to successfully run this code.

B. It will run successfully producing a result of 4.

C. It will result in an error because table depts has been altered.

D. It will run successfully producing a result of 5.

ANSWER: B

Explanation:

It will run successfully producing a result of 4. When Oracle creates a view, an asterisk in the defining query is expanded using the columns that exist in the base table at that time. Therefore, `DEP_VU` is created with the four columns originally present in `DEPTS`; the asterisk is not retained as a dynamic instruction to include future columns. Adding `DEP_EMAIL` to `DEPTS` changes the base table, but it does not add that new column to the existing view definition. Oracle records the dependency and may mark the view invalid following the DDL change. On its next use, Oracle automatically attempts to recompile the dependent view. Because its original four-column projection remains valid, that recompilation succeeds transparently and the PL/SQL block can execute. Consequently, the view metadata examined by the block still represents four columns, which produces the value 4. To expose the new base-table column through the view, the view must be explicitly recreated or replaced with a definition that includes it. Oracle documents view creation and the way a view's query defines its columns in the CREATE VIEW reference; dependency invalidation and automatic recompilation are described in the Oracle Database Administrator's Guide.