iNeuron

Python

Interview Questions

1. In PL/SQL, how do you add a new table row using DML operations?

New table rows can be added using DML (Data Manipulation Language) methods in PL/SQL. The INSERT statement, which describes the columns and values of a specific row that you want to add to your database, is used to do this.

For instance: (column1, column2,...) INSERT INTO MY TABLE VALUES ('value1', 'value2',...); This will create a new table row in "my_table" containing the appropriate values for each provided column. You can insert several rows simultaneously using stored procedures or dynamic SQL.

2. How can you use Pseudo columns to manipulate data within PL/SQL?
Pseudo columns are virtual columns that present particular values in a SELECT statement but have no actual existence within the underlying table. In PL/SQL, pseudo-columns can manipulate data by providing additional information for operations such as sorting or replacing existing column names when selecting records from tables.

For example, you could use the ROWID pseudo-column to retrieve the unique row identifier of each record without having it explicitly defined in your SELECT

query like so: SELECT ... FROM my_table ORDER BY ROWID;

This would result in all queried rows being sorted according to their row identifiers instead of any other previously specified order criteria.

3. Describe the purpose of raising an exception with a raise statement in PL/SQL?
Raising an exception is a process of indicating the occurrence of errors within PL/SQL programs. It can be done using the raise statement, which specifies particular error codes so that users can quickly recognize and react to any potential issues they may have encountered during execution.

By raising exceptions, developers can handle and control errors more efficiently than before since it helps them identify exactly what might have gone wrong, provide appropriate user feedback, or take corrective action whenever necessary.

Such Raise statements also help maintain data integrity through transactions by allowing you to roll back changes if something unexpected occurs while executing a script.

4. Explain what happens when the commit statement changes are made using DML operations within the context of a sequence of statements implemented using PL/SQL programming language?
The commit statement is used in PL/SQL to save changes made using DML operations such as INSERT, UPDATE, and DELETE. When this commit statement is included at the end of a sequence of statements, it tells the database server that all data manipulation language (DML) commands should be permanently recorded within its tables on disk.

This means that even if an error occurs during execution or the program stops before completion for any other reason, whatever modifications have been successfully committed will remain effective and won't require manual intervention from users afterwords.

The commit statement also serves to release any resources that were previously held by a cursor or other database objects while the sequence of statements was being executed.

5. What is a SQL Cursor, and what purpose does it serve within the context of PL/SQL programming language?
Within the context of PL/SQL programming language, a SQL Cursor is an internal structure used to handle and process multiple rows of data. It acts as a pointer within your program which points at each row returned from queries in sequence so that you can take appropriate action depending on the result set specified by such statement (e.g., updating records).

As a result, developers may use SQL cursors to retrieve data regularly without constantly relying on database server resources, which greatly reduces overhead when working with huge amounts of data.

6. How can you write a single query in PL/SQL to search for duplicate values?
In PL/SQL, you may use the SELECT query with the DISTINCT keyword to extract the distinct results from a specified table or view and do a search for duplicate values.

The single query should look like this: SELECT DISTINCT * FROM <tablename>; This will return all distinct (unique) records from your selected table or view, allowing you to identify any duplicates within your data set quickly.

Example:

```
DECLARE
-- Declare a cursor to store the query result
CURSOR duplicate_cursor IS
SELECT column_name, COUNT(*) AS count
FROM table_name
GROUP BY column_name
HAVING COUNT(*) > 1;

-- Variables to store the query result
column_value table_name.column_name%TYPE;
row_count INTEGER := 0;
BEGIN
-- Open the cursor
OPEN duplicate_cursor;

-- Fetch the cursor records
LOOP
FETCH duplicate_cursor INTO column_value, row_count;
EXIT WHEN duplicate_cursor%NOTFOUND;

-- Print the duplicate values
DBMS_OUTPUT.PUT_LINE('Duplicate value: ' || column_value || ',
Count: ' || row_count);
END LOOP;

-- Close the cursor
CLOSE duplicate_cursor;
END;
/
```

Reference Output:

Duplicate value: PQR, Count: 4

7. What different object types are available when using PL/SQL?
PL/SQL has six different object types that can be used to write efficient code.

These include tables, views, sequences, synonyms, packages, and procedures.

Tables are structures that store data in rows and columns;
Views provide a virtual table where the underlying query is stored as text;
Sequences generate unique values based on certain conditions provided in their definition;
Synonyms act as aliases to other database objects like tables or functions;
Packages allow multiple related procedures and functions to be grouped within an object type; and
Procedures offer a set of SQL statements that, when called, will execute them sequentially.

8. Is it possible to execute an entire block of code with one command in PL/SQL?
Yes, executing an entire block of code with one command is possible using PL/SQL's EXECUTE IMMEDIATE statement syntax. This allows you to group multiple statements into one and execute them

together as a single unit without having to declare each line separately beforehand:

EXECUTE IMMEDIATE 'BEGIN <statement1>; <statement2>; ... END';

Example:

```
BEGIN
EXECUTE IMMEDIATE '
DECLARE
-- Declare variables
first_name VARCHAR2(50);
last_name VARCHAR2(50);
BEGIN
-- Assign values to variables
first_name := ''John'';
last_name := ''Wink'';

-- Display the full name
DBMS_OUTPUT.PUT_LINE(''Full Name: '' || first_name || '' '' ||
last_name);
END;
';
END;
/
```

Output:

Full Name: John Wink

9. How can you use string literals within my queries while programming with PL/SQL?
String literals are used when writing queries in PL/SQL to represent text-based information such as names, titles and descriptions, etc., stored inside database tables or views previously by other applications using character strings instead of numerical integers and floats, etc.

You can write string literals directly into your queries simply by enclosing them within single quote marks ('): SELECT * FROM <tablename> WHERE name = 'John Doe';

10. How can you select a range of values within my PL/SQL query?
PL/SQL allows you to select a range of values within your query using the BETWEEN operator.

For example, SELECT * FROM mytable WHERE somecolumn BETWEEN x AND y;

In the example, it records all from the 'mytable' table where the value in column 'somecolumn' is between two specified values (x and y) are returned as part of the result set.

You can also choose not to specify one end of a range, such as selecting only those records which have values greater than or equal to x with: SELECT * FROM mytable WHERE somecolumn >= x;

Combining operators like =, <>, <=, >= etc., with BETWEEN allows it to create more complex queries for obtaining data based on specific ranges.

Example:

```
DECLARE

start_value INTEGER := 30;
end_value INTEGER := 40;

BEGIN
-- Using BETWEEN operator
SELECT column_name
INTO variable_name
FROM table_name
WHERE column_name BETWEEN start_value AND end_value;

SELECT column_name
INTO variable_name
FROM table_name
WHERE column_name >= start_value AND column_name <=
end_value;
END;
```

11. What are the modes of parameters available in PL/SQL when passing arguments to stored procedures?
When passing arguments to stored procedures in PL/SQL, there are different modes of parameters available which can be used. The four main modes are IN, OUT, INOUT, and DEFAULT, which indicate the type of input accepted for a particular parameter or argument.

The mode 'IN' indicates an argument passed into the procedure from its calling environment;
'OUT' specifies that data will flow back to its calling environment; and

'INOUT' signifies both incoming and outgoing information between a given procedure's environment.
Finally, when no explicit mode is specified, it defaults to 'DEFAULT' mode, where any changes this parameter makes within the execution context remain local.
Modes of parameters available in PL/SQL

12. What is the purpose of single-line comments in PL/SQL?
Single-line comments in PL/SQL are used to add remarks and notes to code. They begin with two hyphens (--) followed by the comment, which can span a single line of text. When compiling your program or script, the compiler ignores anything on the same line after the double hyphens, allowing developers to clarify their work for other readers without changing its function. For example, the following single-line comment in PL/SQL explains what a line of code does without changing how it functions. This code checks if there are any new entries in the database table.

Example:

```
DECLARE
-- Declare variables
first_name VARCHAR2(50);
last_name VARCHAR2(50);
BEGIN
-- Assign values to variables
first_name := 'John'; -- First name
last_name := 'Doe'; -- Last name

-- Display the full name
```

```
DBMS_OUTPUT.PUT_LINE('Full Name: ' || first_name || ' ' ||
last_name);
END;
/
```

13. How can we write a multi-line comment within a PL/SQL program?
Multi-line comments in PL/SQL are used to add remarks and notes over multiple lines of code. They begin with a double hyphen (--) followed by an asterisk (*), then the comment text, and finally, they end with an asterisk (*) followed by two hyphens (--).

Anything between these characters will be ignored when compiling your program or script, allowing developers to clarify their work for other readers without changing its functions.

For example, the following multi-line comment in PL/SQL explains what several lines of code do together: --* This block checks if there is new data added to the database table since the last execution *--

14. What are row-level triggers, and how do they work in PL/SQL programming?
Row-level triggers are a type of trigger in PL/SQL programming that fire for each row affected by the triggering statement. They allow developers to specify actions based on changes made to individual rows, such as inserting data into logging tables or raising an exception if certain conditions are unmet.

When triggered, these stored procedures will modify or query information related to the modified row and make decisions

accordingly before executing any additional statements. Row-level triggers can be beneficial when writing complex business rules as they offer precise control over data processing within a database application.

15. Can PL/SQL commands store or display graphic images?
Yes, PL/SQL commands can store and display graphic images. Graphics files such as JPEGs or PNGs can be stored in the database within BLOB (binary large objects) fields that efficiently store digital properties. Likewise, these binary data types can then be used to output graphical elements from the results of a query via display packages like Oracle's own HTP package.

This package allows developers to store pictures directly into their application databases and then render them on webpages without any extra work by utilizing functions like htp.image or even generating dynamic bar charts with graphs with code similar to htp.p ('');

16. How can network traffic be monitored with the help of PL/SQL commands?
Network traffic can be monitored with the help of PL/SQL commands using database packet sniffing. Packet sniffing is a method of intercepting and analyzing network packets sent over various networks, including local area and wide-area networks.

With this technique, developers can use PL/SQL packages such as DBMS_NETWORK_PACKET to look at all incoming and outgoing network requests from their program or script in order to detect

malicious activity or analyze performance issues like slow response times.

17. What types of records may be manipulated through PL/SQL programming?
PL/SQL programming allows developers to manipulate different types of records. These include database tables, views, sequences, synonyms and functions that can all be manipulated in order to facilitate different operations, such as creating new entries or modifying existing ones.

Additionally, PL/SQL programs support complex transactions like manipulating multiple related components, allowing them to handle heavier workloads with fewer conflicts than single-execution statements alone.

18. What is the use of package body statements while working on a complex database system with PL/SQL code?
The package body statement in PL/SQL is used to develop complex database systems with procedural logic. A package body is a collection of related functions and procedures whose code must be defined for the program to run properly, and it can also serve as a wrapper around several other elements, like any variables and type definitions it needs.

With this form of encapsulation, developers can better organize their programs into logical groupings based on what they do, making them

easier to read while maintaining their performance when executing tasks.

19. What is a cursor variable, and how can it be used in PL/SQL?
A cursor variable is a pointer or reference to the memory address of an Oracle server-side cursor that enables stored procedures and functions in PL/SQL to access data from relational databases.

It can be used to retrieve rows from one or more database tables by using SQL queries within PL/SQL code. The returned result set is assigned into a record structure that remains accessible for the duration of the program execution.

Cursor Variable in PL/SQL

20. How do you check for duplicate records within an outer query in PL/SQL?
In PL/SQL, you can check for duplicate records within an outer query using the SELECT DISTINCT clause. The SELECT DISTINCT clause ensures that only the specified columns' distinct (unique) values are returned in a record set.

When combined with other clauses such as GROUP BY and ORDER BY, it can detect duplicate rows based on specific criteria or group them using various column combinations.

Additionally, aggregate functions such as COUNT, MAX and MIN can be used to calculate statistics on the returned individual records.