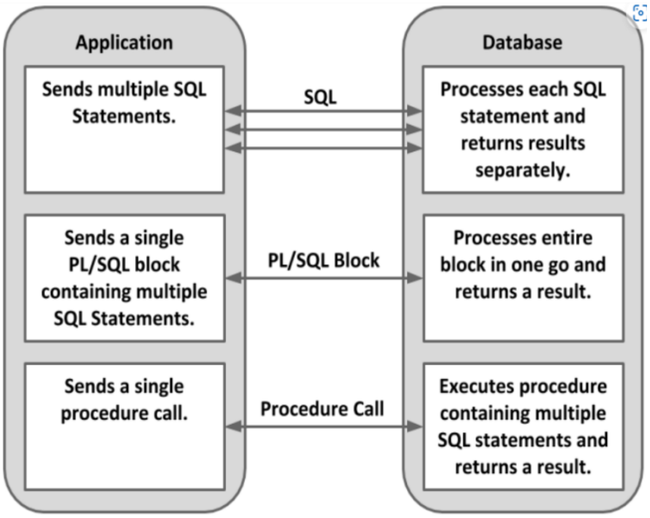
RDBMS-01-Introduction to PLSQL

What is PLSQL

- PL/SQL is a procedural extension of SQL, making it extremely simple to write procedural code that includes SQL as if it were a single language.
- The data types in PL/SQL are a super-set of those in the database, so you rarely need to perform data type conversions when using PL/SQL.



Application
Database
Sends multiple SQL Statements.
SQL
Processes each SQL statement and returns results separately.
Sends a single PL/SQL block containing multiple SQL Statements.
PL/SQL Block
Processes entire block in one go and returns a result.
Sends a single procedure call.
Procedure Call
Executes procedure containing multiple SQL statements and returns a result.

Features of PLSQL

- PL/SQL is basically a procedural language, which provides the functionality of decision making, iteration and many more features of procedural programming languages.
- PL/SQL can execute a number of queries in one block using single command.
- One can create a PL/SQL unit such as procedures, functions, packages, triggers, and types, which are stored in the database for reuse by applications.
- PL/SQL provides a feature to handle the exception which occurs in PL/SQL block known as exception handling block.
- Applications written in PL/SQL are portable to computer hardware or operating system where Oracle is operational.
- PL/SQL Offers extensive error checking

Difference Between SQL and PLSQL

SQL	PLSQL
SQL is a single query that is used to perform DML and DDL operations.	PL/SQL is a block of codes that used to write the entire program blocks/ procedure/ function, etc
It defines what needs to be done, rather than how things need to be done	PL/SQL is procedural that defines how the things needs to be done
Executes as a single statement.	Executes as a whole block.
Mainly used to manipulate data.	Mainly used to create an application
Cannot contain PL/SQL code in it.	It is an extension of SQL, so it can contain SQL inside it.

Dr. Codd's Rules

RULE-00: The Foundation Rule:

- The database must be in relational form.
- So that the system can handle the database through its relational capabilities.

RULE-01: Information Rule:

- A database contains various information, and this information must be stored in each cell of a table in the form of rows and columns.

RULE-02: Guaranteed Access Rule:

- Every single or precise data (atomic value) may be accessed logically from a relational database using the combination of primary key value, table name, and column name.

RULE-03: Systematic Treatment of Null Values:

- This rule defines the systematic treatment of Null values in database records.
- The null value has various meanings in the database, like missing the data, no value in a cell, inappropriate information, unknown data and the primary key should not be null

RULE-04: Active/Dynamic Online Catalogue based on the relational model:

- It represents the entire logical structure of the descriptive database that must be stored online and is known as a database dictionary.
- It authorizes users to access the database and implement a similar query language to access the database

RULE-05: Comprehensive Data Sub-Language Rule:

- The relational database supports various languages, and if we want to access the database, the language must be the explicit, linear or well-defined syntax, character strings and supports the comprehensive: data definition, view definition, data manipulation, integrity constraints, and limit transaction management operations.
- If the database allows access to the data without any language, it is considered a violation of the database

RULE-06: View Updating Rule:

- All views table can be theoretically updated and must be practically updated by the database systems.

RULE-07: Relational Level Operation (High-Level Insert, Update and delete) Rule:

- A database system should follow high-level relational operations such as insert, update, and delete in each level or a single row.
- It also supports union, intersection and minus operation in the database system.

RULE-08: Physical Data Independence Rule:

- All stored data in a database or an application must be physically independent to access the database.
- Each data should not depend on other data or an application.
- If data is updated or the physical structure of the database is changed, it will not show any effect on external applications that are accessing the data from the database.

RULE-09: Logical Data Independence Rule:

- It is similar to physical data independence. It means, if any changes occurred to the logical level (table structures), it should not affect the user's view (application).
- For example, suppose a table either split into two tables, or two table joins to create a single table, these changes should not be impacted on the user view application.

RULE-10: Integrity Independence Rule:

- A database must maintain integrity independence when inserting data into table's cells using the SQL query language.
- All entered values should not be changed or rely on any external factor or application to maintain integrity.
- It is also helpful in making the database-independent for each front-end application.

RULE-11: Distribution Independence Rule:

- The distribution independence rule represents a database that must work properly, even if it is stored in different locations and used by different end-users.
- Suppose a user accesses the database through an application; in that case, they should not be aware that another user uses particular data, and the data they always get is only located on one site.
- The end users can access the database, and these access data should be independent for every user to perform the SQL queries.

RULE-12: Non Subversion Rule:

- The non-submersion rule defines RDBMS as a SQL language to store and manipulate the data in the database.
- If a system has a low-level or separate language other than SQL to access the database system, it should not subvert or bypass integrity to transform data.

Difference: DBMS and RDBMS

DBMS	RDBMS
DBMS applications store data as file	RDBMS applications store data in a tabular form
In DBMS, data is generally stored in either a hierarchical form or a navigational form	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables
Normalization is not present in DBMS	Normalization is present in RDBMS.
DBMS does not apply any security with regards to data manipulation.	RDBMS defines the integrity constraint for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property

DBMS	RDBMS
DBMS uses file system to store data, so there will be no relation between the tables.	In RDBMS, data values are stored in the form of tables, so a relationship between these data values will be stored in the form of a table as well.
DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
DBMS does not support distributed database.	RDBMS supports distributed database
DBMS is meant to be for small organization and deal with small data. it supports single user.	RDBMS is designed to handle large amount of data. it supports multiple users
Examples of DBMS are file systems, xml etc.	Example of RDBMS are mysql, postgre, sql server, oracle etc

Structure of PLSQL Block

- The basic unit in PL/SQL is a block.
- All PL/SQL programs are made up of blocks, which can be nested within each other
- Typically, each block performs a logical action in the program.

```
DECLARE
-- Declaration Statements (optional) --
BEGIN
-- Executable Statements(mandatory) --
EXCEPTION
-- Exception Handling Statements(optional) --
END;
```

- In `DECLARE` section variables, constants, records as cursors can be declared which stores data temporarily. It basically consists definition of PL/SQL identifiers.
- `BEGIN` is a mandatory section and here the program logic is written to perform any task like loops and conditional statements. It supports all DML commands, DDL commands and SQL built-in functions as well
- `EXCEPTION` section is optional which contains statements that are executed when a run-time error occurs.

```
DECLARE
    my_variable VARCHAR2(50);
BEGIN
    my_variable := 'Hello, world!';
    DBMS_OUTPUT.put_line(my_variable);
END;
```

PLSQL Literals

- Literals are the explicit numeric, character, string or Boolean values which are not represented by an identifier

- Literals are case-sensitive.

Literals	Example
Numeric	75125, 3568, 33.3333333
Character	’A’ ’%’ ’9’ ’ ’ ’z’ ’(’
String	Hello !
Boolean	TRUE, FALSE, NULL
Date and Time	’26-11-2002’ , ’2012-10-29 12:01:01’

Conditional Statements

if Statement

- It is used when you want to execute statements only when condition is TRUE

```
IF condition THEN
    -- statements to be executed if the condition is true --
END IF;
```

```
-- To find wether the number is Even --

DECLARE
    num NUMBER;
BEGIN
    num :=: num;
    IF MOD(num, 2) = 0 THEN
        dbms_output.put_line(num || ' is Even');
    END IF;
END;
```

if-else Statement

- It is used when you want to execute one set of statements when condition is TRUE or a different set of statements when condition is FALSE.

```
IF condition THEN
    -- statements to be executed if the condition is true --
ELSE
    -- statements to be executed if the condition is false --
END IF;
```

```
-- To find wether the number is Odd or Even --

DECLARE
    num NUMBER;
BEGIN
    num :=: num;
    IF MOD(num, 2) = 0 THEN
        dbms_output.put_line(num || ' is Even');
    ELSE
        dbms_output.put_line(num || ' is Odd');
    END IF;
END;
```

if-elseif Statement

- It is used when you want to execute one set of statements when condition1 is TRUE or a different set of statements when condition2 is TRUE.

```
IF condition THEN
    -- statements to be executed if the condition is true --
ELSEIF condition THEN
    -- statements to be executed if the condition is true --
ELSE
    -- statements to be executed if the condition is false --
END IF;
```

```
-- To find out entered number is positive, negative or zero

DECLARE
    num NUMBER(10);
BEGIN
    num:=:num;
    IF(num>0)THEN
        dbms_output.put_line('Number is Positive')
    ELSEIF(num<0)THEN
        dbms_output.put_line('Number is Negative')
    ELSE
        dbms_output.put_line('Number is Zero')
    END IF;
END;
```

Case Statements

- Case statement facilitates you to execute a sequence of statements based on a selector.
- A selector can be anything such as variable, function or an expression that the Case statement checks to a Boolean value.

```
CASE expression
    WHEN condition1 THEN result1;
    WHEN condition2 THEN result2;
    WHEN condition3 THEN result3;
    ...
    WHEN condition-n THEN result-n;
    ELSE result
END CASE;
```

```
-- To determine grade based on score entered by user --
```

```
DECLARE
    score number;
    grade varchar;
BEGIN
    score:=:score;
    CASE
        WHEN score BETWEEN 90 AND 100 THEN grade:='Excellent!';
        WHEN score BETWEEN 80 AND 89 THEN grade:='A';
        WHEN score BETWEEN 70 AND 79 THEN grade:='B';
        WHEN score BETWEEN 60 AND 69 THEN grade:='C';
        WHEN score BETWEEN 50 AND 59 THEN grade:='D';
        WHEN score BETWEEN 40 AND 49 THEN grade:='FAIL';
        ELSE grade:='Invalid Score';
    END CASE;
    dbms_output.put_line('The Grade for the score '||score||' is '||grade);
END;
```

Looping Statement

- A loop statement allows us to execute a statement or group of statements multiple times
- An EXIT statement or an EXIT WHEN statement is required to break the loop.

```
LOOP
    Sequence_of_statements;
END LOOP;
```

```
DECLARE
    x number := 10;
BEGIN
    LOOP
        dbms_output.put_line(x);
        x := x + 10;
        IF x > 50 THEN
            EXIT;
        END IF;
    END LOOP;
    -- after exit, control resumes here --
    dbms_output.put_line('After Exit x is: ' || x);
END;
```

```
DECLARE
    x number := 10;
BEGIN
    LOOP
        dbms_output.put_line(x);
        x := x + 10;
        EXIT WHEN x>50;
    END LOOP;
    -- after exit, control resumes here --
    dbms_output.put_line('After Exit x is: ' || x);
END;
```

While Loop

- It is used to repeatedly executes a target statement as long as a given condition is true.

```
WHILE condition LOOP
    sequence_of_statements
END LOOP;
```

```
DECLARE
    a number(2) := 10;
BEGIN
    WHILE a < 20 LOOP
        dbms_output.put_line('value of a: ' || a);
        a := a + 1;
    END LOOP;
END;
```

For Loop

- A For loop is a repetition control structure that allows to efficiently write a loop that needs to execute a specific number of times.

```
FOR counter IN initial_value .. final_value LOOP
    sequence_of_statements;
END LOOP;
```

```
DECLARE
    a number(2);
BEGIN
    FOR a in 10 .. 20 LOOP
        dbms_output.put_line('value of a: ' || a);
    END LOOP;
END;
```

Reverse For Loop

- In reverse for loop after each iteration, the loop counter is decremented.

```
DECLARE
    a number(2) ;
BEGIN
    FOR a IN REVERSE 10 .. 20 LOOP
        dbms_output.put_line('value of a: ' || a);
    END LOOP;
END;
```

Questions

1. What is PLSQL?
2. Explain Features of PLSQL (MID-04M)
3. What is PLSQL block? Explain with example. (MID-04M)
4. State the difference between SQL and PLSQL. (MID-05M)
5. Explain the difference between DBMS and RDBMS (MID-05M)
6. Describe Codd's Rule in RDBMS (MID-05M)
7. Explain PLSQL Conditional statements with suitable example (MID-05M)
8. Explain syntax for the CASE statement (MID-05M)
9. Explain basic LOOP statement in PLSQL (MID-05M)
10. Write a program to print numbers from 01 to 20 using for loop (MID-06M)
11. Write down PLSQL program to find out maximum of two values entered by user (MID-05M)

@SOUBCA