**Using Dynamic SQL**

## What Is Dynamic SQL?

**Dynamic SQL enables you to write programs those reference SQL statements whose full text is not known until runtime.** Before discussing dynamic SQL in detail, a clear definition of static SQL may provide a good starting point for understanding dynamic SQL. Static SQL statements do not change from execution to execution. The full text of static SQL statements are known at compilation, which provides the following benefits:

* Successful compilation verifies that the SQL statements reference valid database objects.
* Successful compilation verifies that the necessary privileges are in place to access the database objects.
* Performance of static SQL is generally better than dynamic SQL.

Because of these advantages, you should use dynamic SQL only if you cannot use static SQL to accomplish your goals, or if using static SQL is cumbersome compared to dynamic SQL. However, static SQL has limitations that can be overcome with dynamic SQL. You may not always know the full text of the SQL statements that must be executed in a PL/SQL procedure. Your program may accept user input that defines the SQL statements to execute, or your program may need to complete some processing work to determine the correct course of action. In such cases, you should use dynamic SQL.

For example, a reporting application in a data warehouse environment might not know the exact table name until runtime. These tables might be named according to the starting month and year of the quarter, for example INV\_01\_1997, INV\_04\_1997, INV\_07\_1997, INV\_10\_1997, INV\_01\_1998, and so on. **You can use dynamic SQL in your reporting application to specify the table name at runtime.**

You might also want to run a complex query with a user-selectable sort order. Instead of coding the query twice, with different ORDER BY clauses, you can construct the query dynamically to include a specified ORDER BY clause.

Dynamic SQL programs can handle changes in data definitions, without the need to recompile. This makes dynamic SQL much more flexible than static SQL.

Dynamic SQL lets you write reusable code because the SQL can be easily adapted for different environments.

**Dynamic SQL also lets you execute data definition language (DDL), Data Control Statements (DCL) statements and other SQL statements that are not supported in directly PLSQL programs.**

## Why Use Dynamic SQL?

You should use dynamic SQL in cases where static SQL does not support the operation you want to perform, or in cases where you do not know the exact SQL statements that must be executed by a PL/SQL procedure. These SQL statements may depend on user input, or they may depend on processing work done by the program. The following sections describe typical situations where you should use dynamic SQL and typical problems that can be solved by using dynamic SQL

### Executing DDL and SCL Statements in PL/SQL

In PL/SQL, you can only execute the following types of statements using dynamic SQL, rather than static SQL:

* Data definition language (DDL) statements, such as CREATE, DROP, GRANT, and REVOKE
* Session control language (SCL) statements, such as ALTER SESSION and SET ROLE

### Executing Dynamic Queries

You can use dynamic SQL to create applications that execute dynamic queries, whose full text is not known until runtime. Many types of applications need to use dynamic queries, including:

* Applications that allow users to input or choose query search or sorting criteria at runtime
* Applications that allow users to input or choose optimizer hints at run time
* Applications that query a database where the data definitions of tables are constantly changing
* Applications that query a database where new tables are created often

Execute immediate:

Execute Immediate is an built in procedure which takes the DDL / DCL / DML statement as a string parameter.

It compiles and it immediately executes it.

The Parsing and the execution is done immediately one after the other.

2 ways are there:

1) Give the DDL / DCL / DML statement directly as a string to Execute Immediate.

For e.g.

begin

Execute Immediate 'Drop table emp101';

end;

2) First assign that DDL / DCL / DML statement to a local varchar variable. And then mention that variable name after Execute Immediate.

For e.g.

Declare

v\_sql varchar(1000);

Begin

v\_sql := 'Drop table Emp102';

Execute Immediate v\_sql;

End;

Using Execute Immediate inside a Procedure:

Create table emp\_trial

As

Select \* from emp;

CREATE OR REPLACE PROCEDURE drop\_table(p\_tablename Varchar)

IS

BEGIN

If Upper(p\_tablename) Not In ('EMP', 'DEPT') Then **-- To prevent SQL Injection**

**EXECUTE IMMEDIATE 'Drop Table ' || p\_tablename;**

End If;

END;

EXECUTE drop\_table ('Emp\_Trial');

When there is a single quote witin single quote then use the escape character.

begin

Execute Immediate 'Insert into emp103(empno,ename,sal) Values(1,''John'',3000)';

end;

/

Note:

Data Value

Escape Character