Oracle Triggers

Introduction to Triggers

A trigger is a stored PL/SQL block that is automatically executed (fired) when a specified event occurs in the database. Triggers help enforce business rules, maintain data integrity, and automate system actions.

Key Characteristics of Triggers:

Automatically executed when a specified event occurs.

Associated with a table, view, or schema.

Cannot accept parameters.

Used for enforcing security, auditing, and automatic calculations.

Types of Triggers

Triggers can be classified based on various criteria:

Based on Timing of Execution

BEFORE Trigger - Executes before the triggering event occurs.

AFTER Trigger - Executes after the triggering event occurs.

INSTEAD OF Trigger - Used for views; executes instead of the DML operation.

COMPOUND Trigger - A single trigger with multiple timing points (BEFORE, AFTER, etc.).

Based on the Event Type

DML Triggers - Fires on INSERT, UPDATE, DELETE operations.

DDL Triggers - Fires on CREATE, ALTER, DROP operations.

System/Event Triggers - Fires on database/system events like LOGON, LOGOFF, SHUTDOWN, STARTUP.

Schema-Level Triggers - Fires when any DDL operation is performed in a schema.

Based on Scope of Execution

Row-Level Trigger - Executes once per affected row.

Statement-Level Trigger - Executes once per triggering statement.

Advanced Trigger Concepts

A. Trigger Firing Sequence

When multiple triggers exist on the same table:

- BEFORE triggers fire first.
- Then the DML operation occurs.
- AFTER triggers fire next.
- Statement-level triggers execute before and after row-level triggers.

B. Mutating Table Errors

A mutating table error occurs when a row-level trigger tries to modify the table that fired the trigger. This happens because the table is in an intermediate state during execution.

Ways to Avoid Mutating Table Errors:

- Use compound triggers.
- Use **AUTONOMOUS_TRANSACTION** pragma.
- Use temporary tables to store intermediate results.

C. Compound Triggers

A compound trigger combines multiple triggering events (BEFORE, AFTER) within a single block. It helps in reducing mutating table errors and improving performance.

Advantages:

- Reduces context switching between SQL and PL/SQL.
- Avoids mutating table issues.
- Improves performance by consolidating logic.

D. Trigger vs Constraints

Feature	**Constraints**	**Triggers**
Enforcement	Enforced at a column/table level	Fires on specified conditions
Performance	Faster as executed by optimizer	Slower due to procedural execution
Complexity	Simple rules	Can contain complex logic
Execution Timing	Enforced automatically	Fires based on DML/DDL events

4. Best Practices for Triggers

Keep trigger logic simple and optimized.

Avoid using triggers for enforcing constraints (use CHECK/FOREIGN KEY instead).

Minimize the use of AFTER triggers for better performance.

Use compound triggers when dealing with multiple operations on the same table.

Always document trigger functionality for maintainability.

Avoid committing transactions inside triggers as it may cause transaction inconsistencies.

5. System and Event-Based Triggers

A. System-Level Triggers

System triggers execute based on database events such as login/logout, shutdown, or startup.

Event	**Description**
AFTER LOGON	Executes after a user logs in
BEFORE LOGOFF	Executes before a user logs out
AFTER STARTUP	Executes after the database starts
BEFORE SHUTDOWN	Executes before the database shuts down

B. Auditing with Triggers

Triggers can be used for auditing purposes, such as tracking changes in tables.

Store old and new values in an audit table.

Maintain logs for insert/update/delete operations.

Debugging Tips

Use **DBMS_OUTPUT.PUT_LINE** to print debugging messages.

Check **USER_TRIGGERS** view to list all triggers in a schema.

Use **SHOW ERRORS** to display trigger compilation errors.