

Creating Database-Event Triggers

- Triggering user event:
 - CREATE, ALTER, or DROP
 - Logging on or off
- Triggering database or system event:
 - Shutting down or starting up the database
 - A specific error (or any error) being raised

ORACLE

18-2

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Triggers on DDL Statements

You can specify one or more types of DDL statements that can cause the trigger to fire. You can create triggers for these events on DATABASE or SCHEMA unless otherwise noted. You can also specify BEFORE and AFTER for the timing of the trigger. The Oracle database fires the trigger in the existing user transaction.

You cannot specify as a triggering event any DDL operation performed through a PL/SQL procedure.

The trigger body in the syntax in the slide represents a complete PL/SQL block. DDL triggers fire only if the object being created is a cluster, function, index, package, procedure, role, sequence, synonym, table, tablespace, trigger, type, view, or user.

Creating Database Triggers

Before coding the trigger body, decide on the components of the trigger.

Triggers on system events can be defined at the database or schema level. For example, a database shutdown trigger is defined at the database level. Triggers on data definition language (DDL) statements, or a user logging on or off, can also be defined at either the database level or schema level. Triggers on data

LOGON and LOGOFF Triggers: Example

```
-- Create the log_trig_table shown in the notes page
-- first

CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON ON SCHEMA
BEGIN
    INSERT INTO log_trig_table(user_id,log_date,action)
    VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF ON SCHEMA
BEGIN
    INSERT INTO log_trig_table(user_id,log_date,action)
    VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

ORACLE

10-4

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

manipulation language (DML) statements are defined on a specific table or a view.

A trigger defined at the database level fires for all users whereas a trigger defined at the schema or table level fires only when the triggering event involves that schema or table.

Triggering events that can cause a trigger to fire:

- A data definition statement on an object in the database or schema
- A specific user (or any user) logging on or off
- A database shutdown or startup
- Any error that occurs

Creating Triggers on System Events

You can create triggers for the events listed in the table in the slide on DATABASE or SCHEMA, except SHUTDOWN and STARTUP, which apply only to DATABASE.

LOGON and LOGOFF Triggers: Example

You can create these triggers to monitor how often you log on and off, or you may want to write a report that monitors the length of time for which you are logged on. When you specify ONSCHEMA, the trigger fires for the specific

CALL Statements in Triggers

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing
event1 [OR event2 OR event3]
ON table_name
[REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
[WHEN condition]
CALL log_execution;
```

```
CREATE OR REPLACE PROCEDURE log_execution IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('log_execution: Employee Inserted');
END;
/
CREATE OR REPLACE TRIGGER log_employee
BEFORE INSERT ON EMPLOYEES
CALL log_execution -- no action is needed
/
```

ORACLE

18-5

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

user. If you specify ONDATABASE, the trigger fires for all users. The definition of the log_trig_table used in the slide examples is as follows:

```
CREATE TABLE
log_trig_table( user_id
VARCHAR2(30), log_date
DATE, action
VARCHAR2(40))
/
```

CALL Statements in Triggers

A CALL statement enables you to call a stored procedure, rather than code the PL/SQL body in the trigger itself. The procedure can be implemented in PL/SQL, C, or Java.

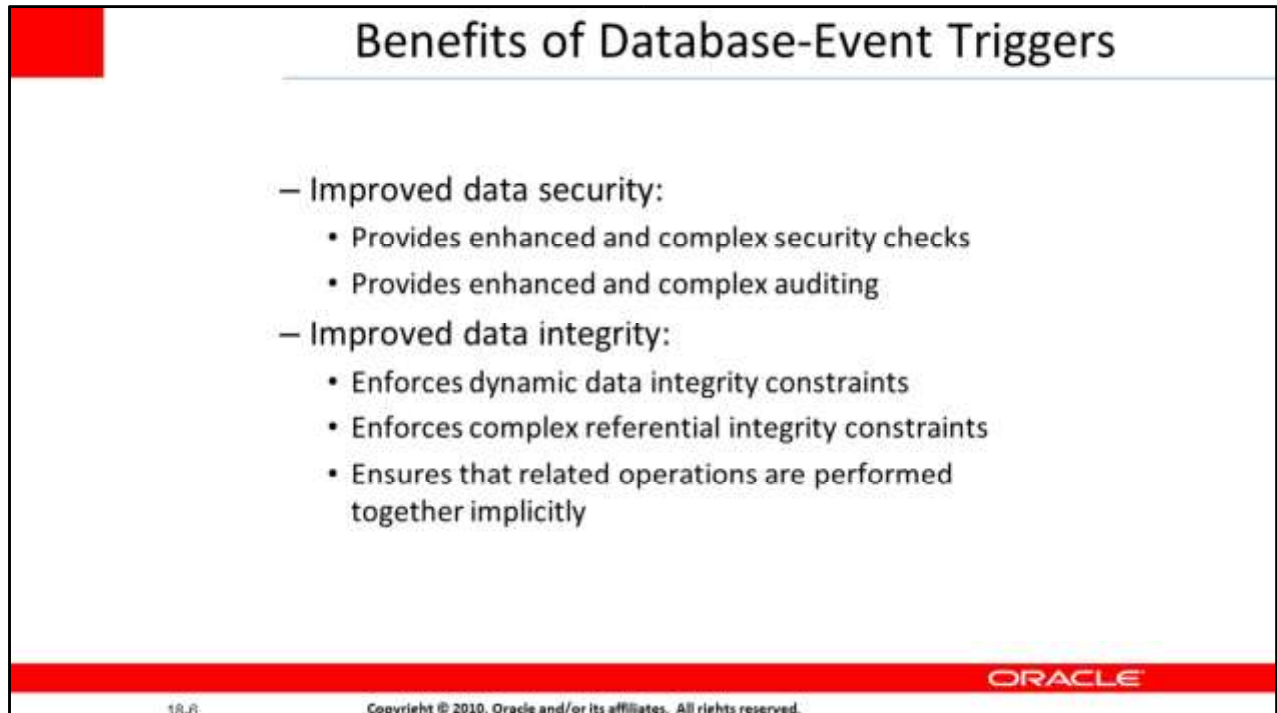
The call can reference the trigger attributes :NEW and :OLD as parameters, as in the following example:

```
CREATE OR REPLACE TRIGGER salary_check
BEFORE UPDATE OF salary, job_id ON
employees
FOR EACH ROW
WHEN (NEW.job_id <> 'AD_PRES')
```

```
CALL check_salary(:NEW.job_id,  
:NEW.salary)
```

Note: There is no semicolon at the end of the CALL statement.

In the preceding example, the trigger calls a check_salary procedure. The procedure compares the new salary with the salary range for the new job ID from the JOBS table.



The slide features a red header bar with the title 'Benefits of Database-Event Triggers' in white. The main content area is white with a list of benefits. At the bottom, there is a red footer bar with the Oracle logo and copyright information.

Benefits of Database-Event Triggers

- Improved data security:
 - Provides enhanced and complex security checks
 - Provides enhanced and complex auditing
- Improved data integrity:
 - Enforces dynamic data integrity constraints
 - Enforces complex referential integrity constraints
 - Ensures that related operations are performed together implicitly

ORACLE

10-g Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Benefits of Database-Event Triggers

You can use database triggers:

As alternatives to features provided by the Oracle server If your requirements are more complex or more simple than those provided by the Oracle server

If your requirements are not provided by the Oracle server at all

System Privileges Required to Manage Triggers

To create a trigger in your schema, you need the CREATETRIGGER system privilege, and you must own the table specified in the triggering statement, have the ALTER privilege for the table in the triggering statement, or have the

System Privileges Required to Manage Triggers

The following system privileges are required to manage triggers:

–The privileges that enable you to create, alter, and drop triggers in any schema:

- GRANT CREATE TRIGGER TO ora61
- GRANT ALTER ANY TRIGGER TO ora61
- GRANT DROP ANY TRIGGER TO ora61

–The privilege that enables you to create a trigger on the database:

- GRANT ADMINISTER DATABASE TRIGGER TO ora61

–The EXECUTE privilege (if your trigger refers to any objects that are not in your schema)

ORACLE

18-7

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ALTERANYTABLE system privilege. You can alter or drop your triggers without any further privileges being required.

If the ANY keyword is used, you can create, alter, or drop your own triggers and those in another schema and can be associated with any user's table. You do not need any privileges to invoke a trigger in your schema. A trigger is invoked by DML statements that you issue. But if your trigger refers to any objects that are not in your schema, the user creating the trigger must have the EXECUTE privilege on the referenced procedures, functions, or packages, and not through roles.

To create a trigger on DATABASE, you must have the ADMINISTER DATABASE TRIGGER privilege. If this privilege is later revoked, you can drop the trigger but you cannot alter it.

Note: Similar to stored procedures, statements in the trigger body use the privileges of the trigger owner, not the privileges of the user executing the operation that fires the trigger.

Guidelines for Designing Triggers

Use triggers to guarantee that related actions are performed for a specific operation and for centralized, global operations that should be fired for

Guidelines for Designing Triggers

- You can design triggers to:
 - Perform related actions
 - Centralize global operations
- You must not design triggers:
 - Where functionality is already built into the Oracle server
 - That duplicate other triggers
- You can create stored procedures and invoke them in a trigger, if the PL/SQL code is very lengthy.
- Excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications.

ORACLE

18-8

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

the triggering statement, independent of the user or application issuing the statement.

Do not define triggers to duplicate or replace the functionality already built into the Oracle database. For example, implement integrity rules using declarative constraints instead of triggers. To remember the design order for a business rule:

Use built-in constraints in the Oracle server, such as primary key, and so on.

Develop a database trigger or an application, such as a servlet or Enterprise JavaBeans (EJB) on your middle tier.

Use a presentation interface, such as Oracle Forms, HTML, JavaServer Pages (JSP) and so on, for data presentation rules.

Excessive use of triggers can result in complex interdependencies, which may be difficult to maintain. Use triggers when necessary, and be aware of recursive and cascading effects.

Avoid lengthy trigger logic by creating stored procedures or packaged procedures that are invoked in the trigger body.

Database triggers fire for every user each time the event occurs on the trigger that is created.

