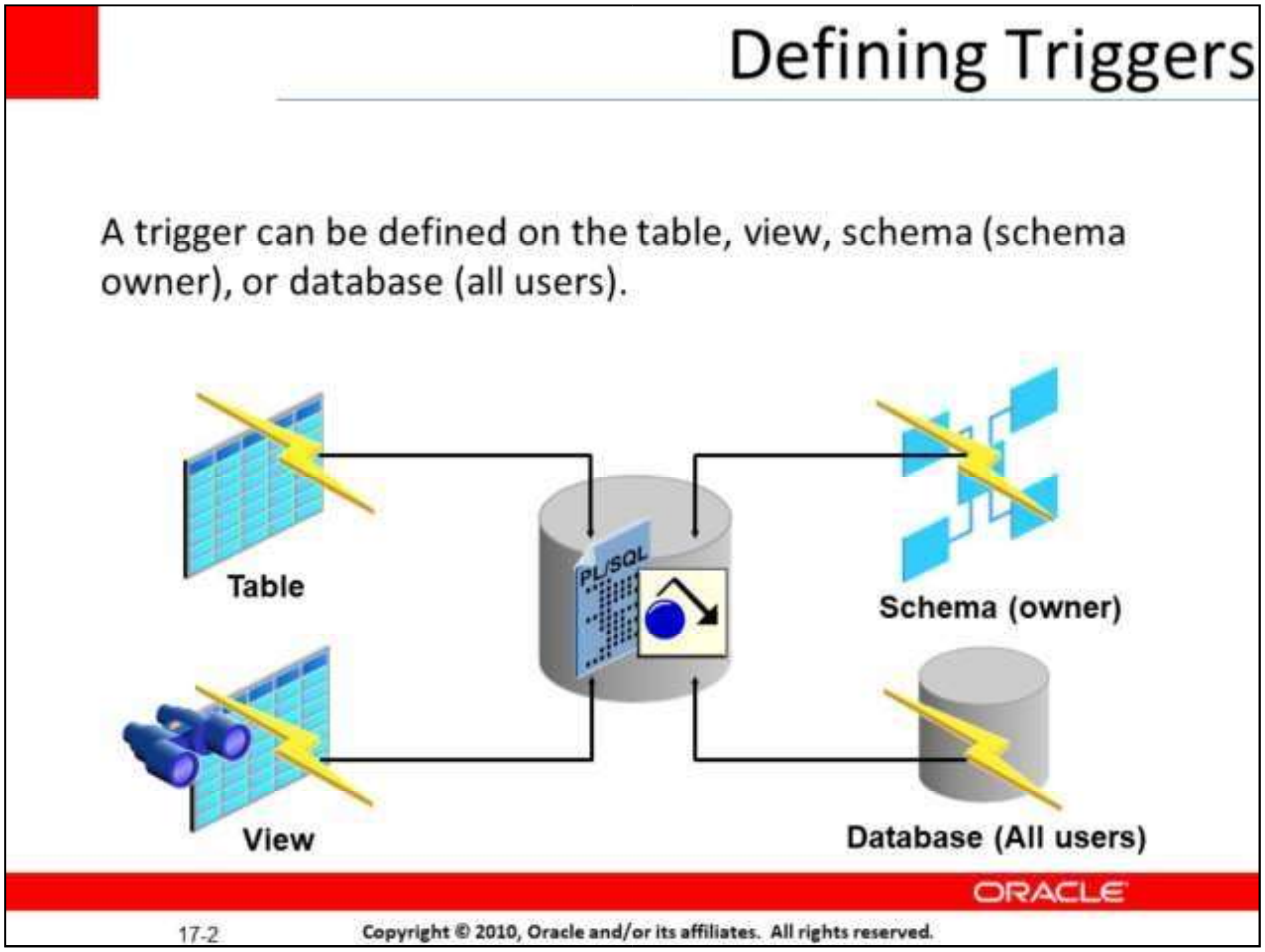
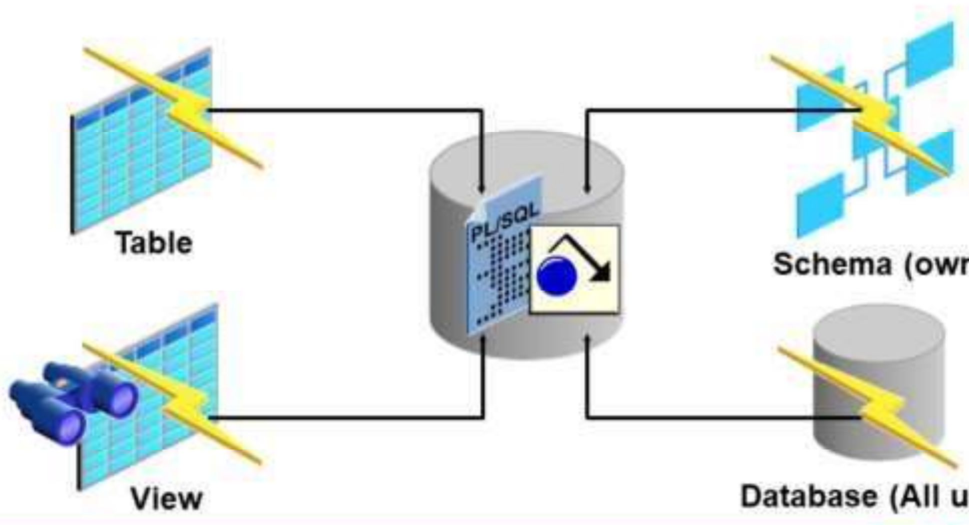


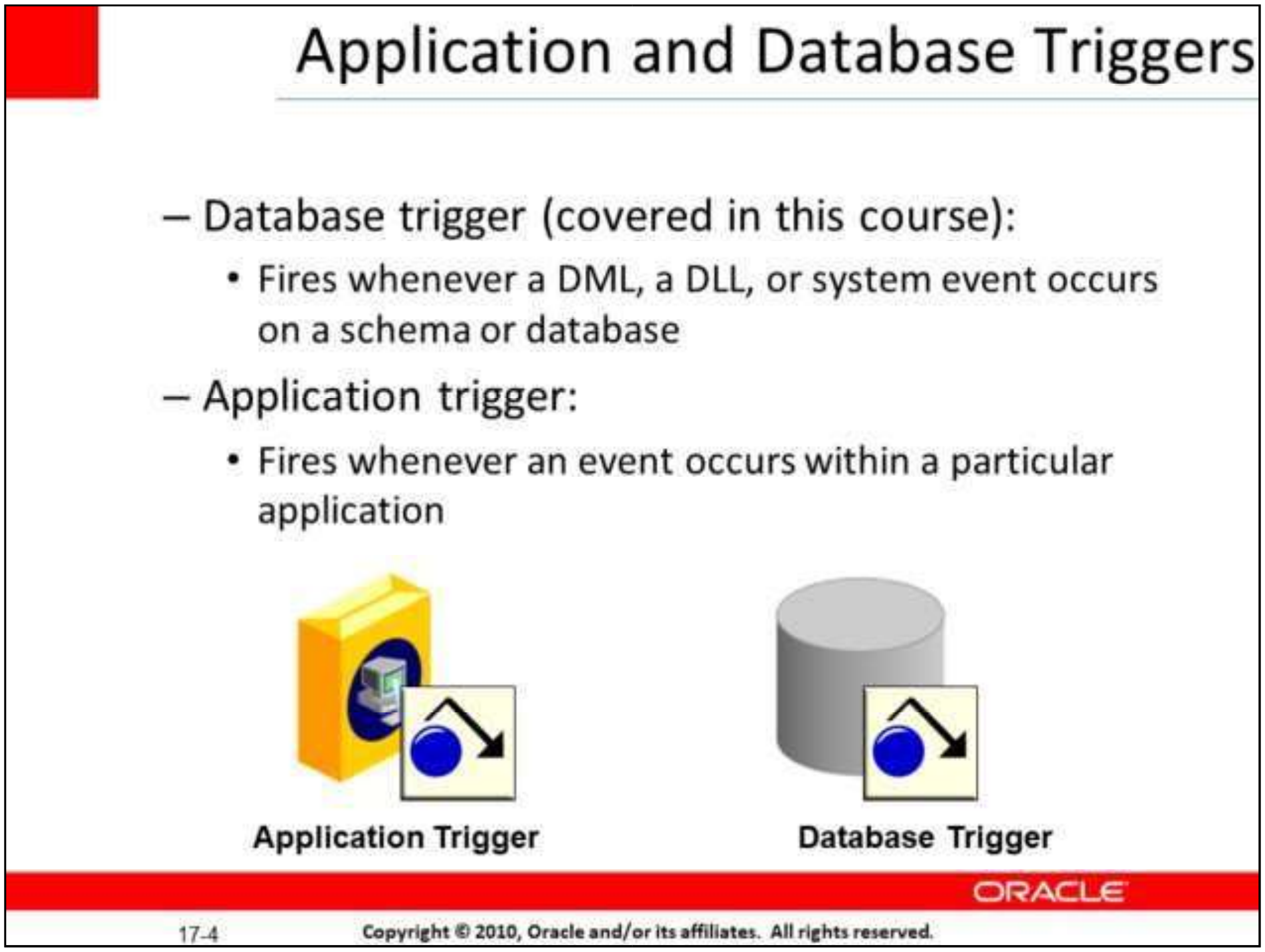
Defining Triggers

A trigger can be defined on the table, view, schema (schema owner), or database (all users).



Working with Triggers: Overview

Triggers are similar to stored procedures. A trigger stored in the database contains PL/SQL in the form of an anonymous block, a call statement, or a compound trigger block. However, procedures and triggers differ in the way that they are invoked. A procedure is explicitly run by a user, application, or trigger. Triggers are implicitly fired by the Oracle database when a triggering event occurs, no matter which user is connected or which application is being used.



Application and Database Triggers

– Database trigger (covered in this course):
  - Fires whenever a DML, a DLL, or system event occurs on a schema or database

– Application trigger:
  - Fires whenever an event occurs within a particular application

Application Trigger
Database Trigger

Triggering Event or Statement

A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire. A triggering event can be one or more of the following:

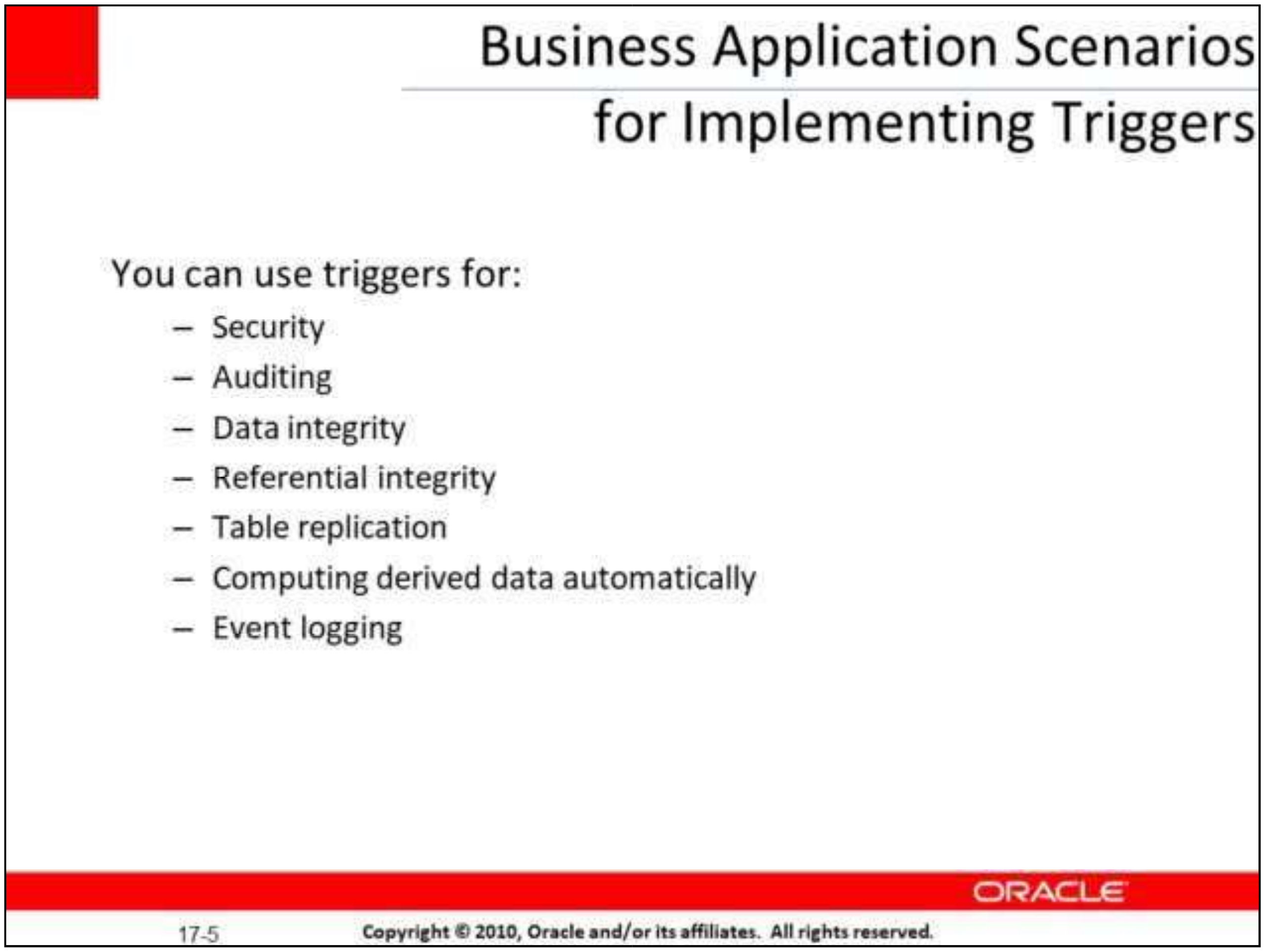
- An INSERT, UPDATE, or DELETE statement on a specific table (or view, in some cases)
- A CREATE, ALTER, or DROP statement on any schema object
- A database startup or instance shutdown
- A specific error message or any error message
- A user logon or logoff

Types of Triggers

Application triggers execute implicitly whenever a particular data manipulation language (DML) event occurs within an application. An example of an application that uses triggers extensively is an application developed with Oracle Forms Developer.

Database triggers execute implicitly when any of the following events occur:

- DML operations on a table
- DML operations on a view, with an INSTEADOF trigger
- DDL statements, such as CREATE and ALTER



Business Application Scenarios for Implementing Triggers

You can use triggers for:

- Security
- Auditing
- Data integrity
- Referential integrity
- Table replication
- Computing derived data automatically
- Event logging

ORACLE

17-5 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

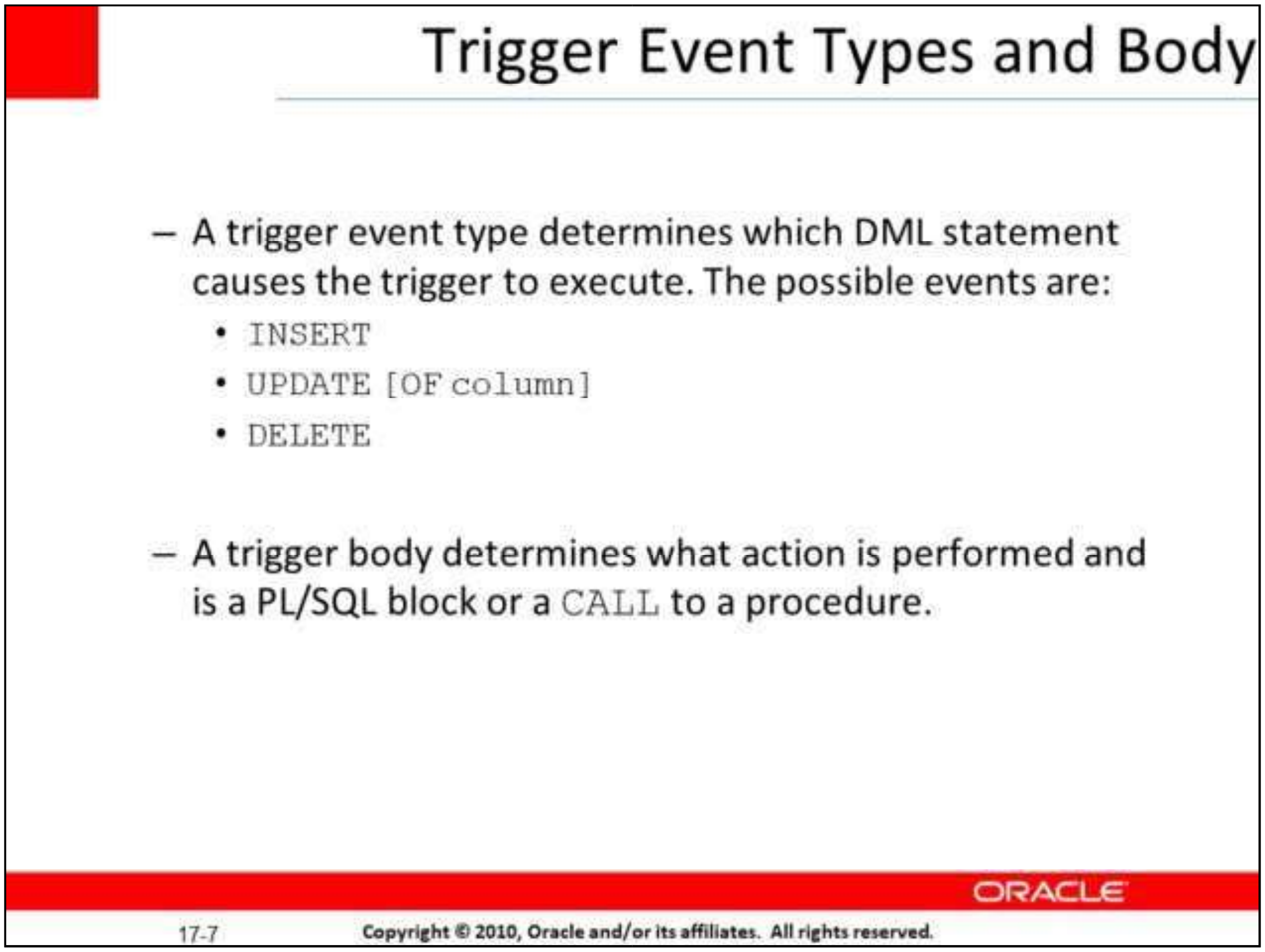
This is the case no matter which user is connected or which application is used. Database triggers also execute implicitly when some user actions or database system actions occur (for example, when a user logs on or the DBA shuts down the database).

Database triggers can be system triggers on a database or a schema (covered in the next lesson). For databases, triggers fire for each event for all users; for a schema, they fire for each event for that specific user. Oracle Forms can define, store, and run triggers of a different sort. However, do not confuse Oracle Forms triggers with the triggers discussed in this lesson.

Business Application Scenarios for Implementing Triggers

Develop database triggers in order to enhance features that cannot otherwise be implemented by the Oracle server or as alternatives to those provided by the Oracle server.

- Security: The Oracle server allows table access to users or roles. Triggers allow table access according to data values.
- Auditing: The Oracle server tracks data operations on tables. Triggers track values for data operations on tables.
- Data integrity: The Oracle server enforces integrity constraints. Triggers implement complex integrity rules.
- Referential integrity: The Oracle server enforces standard referential integrity rules. Triggers implement nonstandard functionality.



Trigger Event Types and Body

- A trigger event type determines which DML statement causes the trigger to execute. The possible events are:
  - `INSERT`
  - `UPDATE [OF column]`
  - `DELETE`
- A trigger body determines what action is performed and is a PL/SQL block or a `CALL` to a procedure.

ORACLE

17-7 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

- Table replication: The Oracle server copies tables asynchronously into snapshots. Triggers copy tables synchronously into replicas.
- Derived data: The Oracle server computes derived data values manually. Triggers compute derived data values automatically.
- Event logging: The Oracle server logs events explicitly. Triggers log events transparently.

Note

In this lesson, we will discuss the BEFORE, AFTER, and INSTEADOF triggers. The other trigger types are discussed in the lesson titled “Creating Compound, DDL, and Event Database Triggers.”

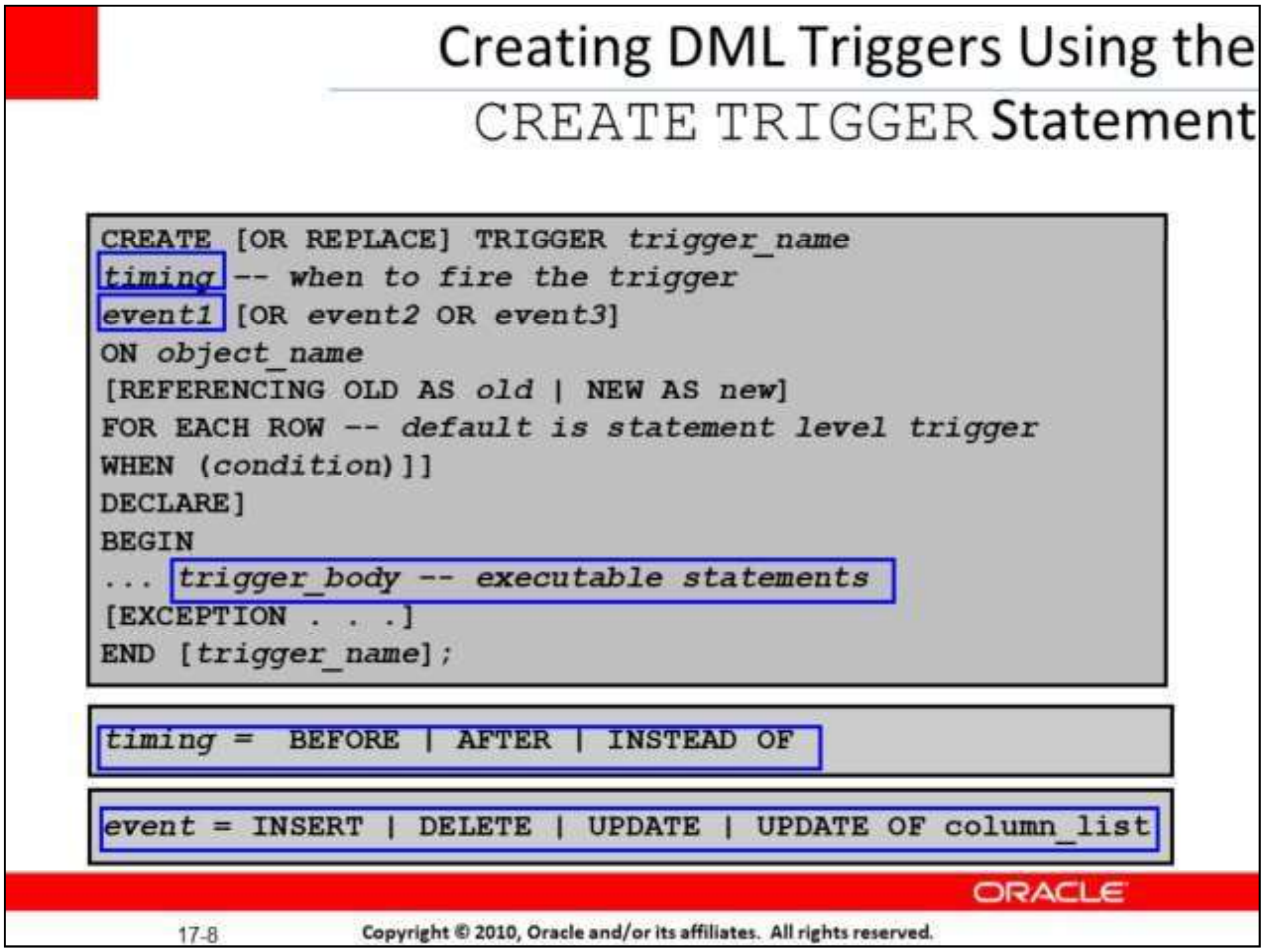
Triggering Event Types

The triggering event or statement can be an INSERT, UPDATE, or DELETE statement on a table.

- When the triggering event is an UPDATE statement, you can include a column list to identify which columns must be changed to fire the trigger. You cannot specify a column list for an INSERT or for a DELETE statement because it always affects entire rows. . . . UPDATE OF salary . . .
- The triggering event can contain one, two, or all three of these DML operations.
  . . . INSERT or UPDATE or DELETE . . . INSERT or
  UPDATE OF job_id . . .

The trigger body defines the action—that is, what needs to be done when the triggering event is issued. The PL/SQL block can contain SQL and PL/SQL statements, and can define PL/SQL constructs such as variables, cursors, exceptions, and so on. You can also call a PL/SQL procedure or a Java procedure.

Note: The size of a trigger cannot be greater than 32 KB.



Creating DML Triggers Using the CREATE TRIGGER Statement

```
CREATE [OR REPLACE] TRIGGER trigger_name
timing -- when to fire the trigger
event1 [OR event2 OR event3]
ON object_name
[REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW -- default is statement level trigger
WHEN (condition)]]
DECLARE]
BEGIN
... trigger_body -- executable statements
[EXCEPTION . . .]
END [trigger_name];
```

```
timing =  BEFORE | AFTER | INSTEAD OF
```

```
event = INSERT | DELETE | UPDATE | UPDATE OF column_list
```

17-8

Creating DML Triggers

The components of the trigger syntax are:

- trigger_name uniquely identifies the trigger.
- timing indicates when the trigger fires in relation to the triggering event. Values are BEFORE, AFTER, and INSTEADOF.
- event identifies the DML operation causing the trigger to fire. Values are INSERT, UPDATE[OFcolumn], and DELETE.
- object_name indicates the table or view associated with the trigger.
- For row triggers, you can specify:
  - A REFERENCING clause to choose correlation names for referencing the old and new values of the current row (default values are OLD and NEW)
  - FOREACHROW to designate that the trigger is a row trigger
  - A WHEN clause to apply a conditional predicate, in parentheses, which is evaluated for each row to determine whether or not to execute the trigger body
- The trigger_body is the action performed by the trigger, implemented as either of the following:
  - An anonymous block with a DECLARE or BEGIN, and an END

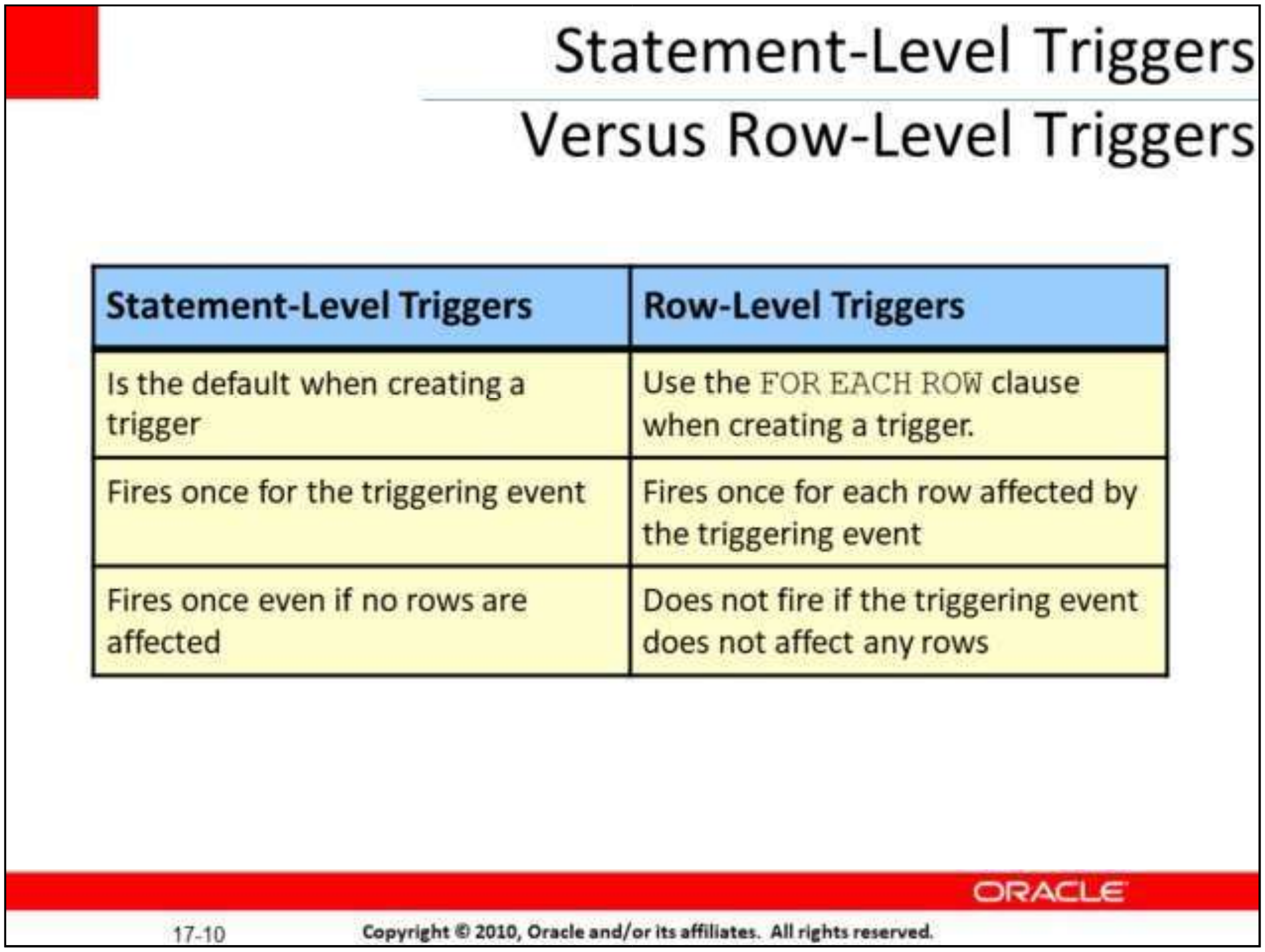
- A CALL clause to invoke a stand-alone or packaged stored procedure, such as:

```
CALL my_procedure;
```

Trigger Timing

The BEFORE trigger timing is frequently used in the following situations:

- To determine whether the triggering statement should be allowed to complete (This eliminates unnecessary processing and enables a rollback in cases where an exception is raised in the triggering action.)
- To derive column values before completing an INSERT or UPDATE statement
- To initialize global variables or flags, and to validate complex business rules The AFTER triggers are frequently used in the following situations:
- To complete the triggering statement before executing the triggering action



Statement-Level Triggers Versus Row-Level Triggers

Statement-Level Triggers	Row-Level Triggers
Is the default when creating a trigger	Use the FOR EACH ROW clause when creating a trigger.
Fires once for the triggering event	Fires once for each row affected by the triggering event
Fires once even if no rows are affected	Does not fire if the triggering event does not affect any rows

17-10 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

- To perform different actions on the same triggering statement if a BEFORE trigger is already present

The INSTEADOF triggers provide a transparent way of modifying views that cannot be modified directly through SQL DML statements because a view is not always modifiable. You can write appropriate DML statements inside the body of an INSTEADOF trigger to perform actions directly on the underlying tables of views.

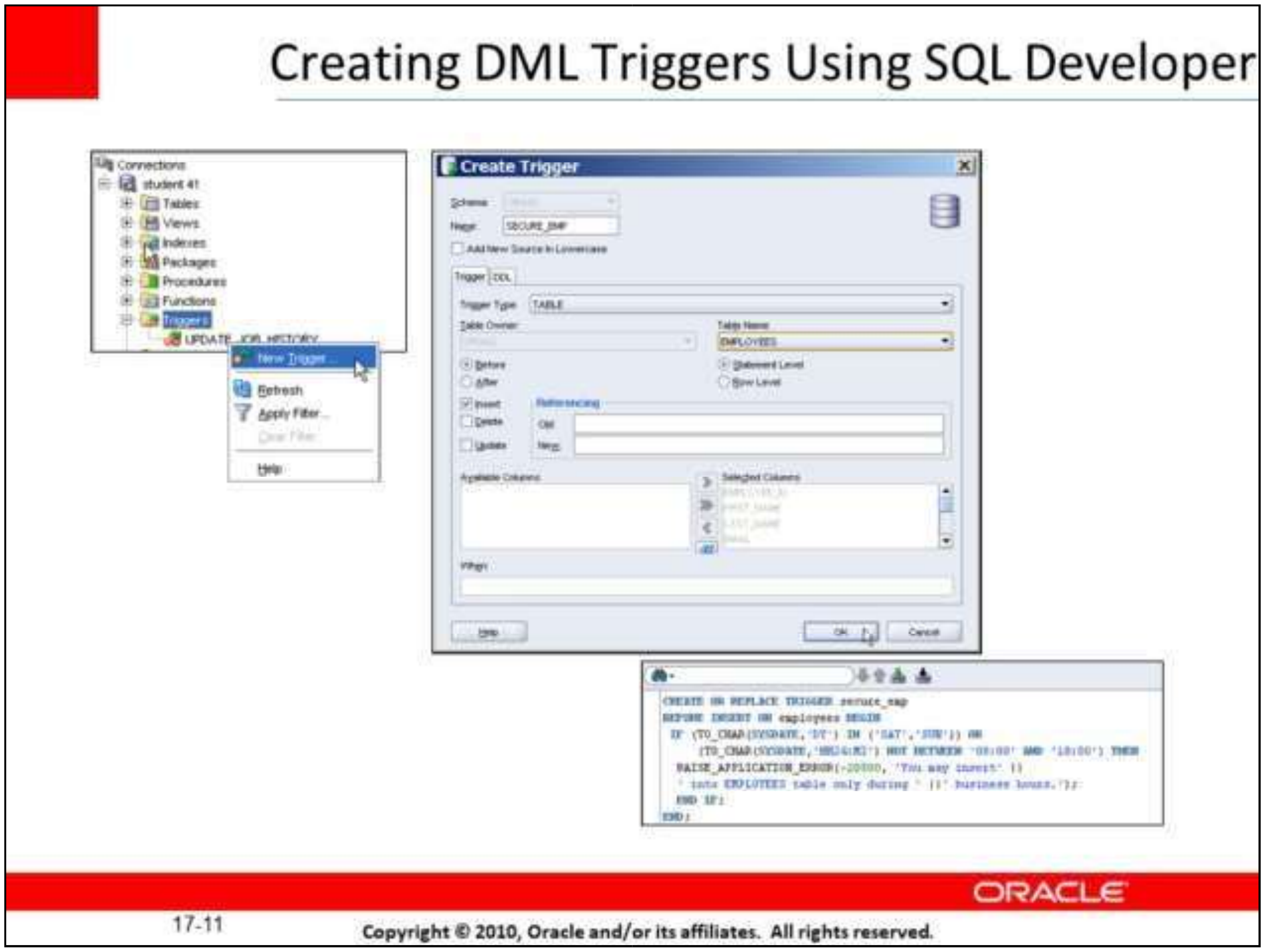
If it is practical, replace the set of individual triggers with different timing points with a single compound trigger that explicitly codes the actions in the order you intend. If two or more triggers are defined with the same timing point, and the order in which they fire is important, then you can control the firing order using the FOLLOWS and PRECEDES clauses.

Types of DML Triggers

You can specify that the trigger will be executed once for every row affected by the triggering statement (such as a multiple row UPDATE) or once for the triggering statement, no matter how many rows it affects.

Statement Trigger

A statement trigger is fired once on behalf of the triggering event, even if no rows are affected at all. Statement triggers are useful if the trigger action does not depend on the data from rows that



Creating DML Triggers Using SQL Developer

Name: SECURE_EMP

Trigger Type: TABLE

Table Name: EMPLOYEES

Before [✓]

Statement Level [✓]

Insert [✓]

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON employees BEGIN
 IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
     (TO_CHAR(SYSDATE,'HH24:MI') NOT BETWEEN '08:00' AND '18:00') THEN
  RAISE_APPLICATION_ERROR(-20500, 'You may insert' ||
  ' into EMPLOYEES table only during ' ||' business hours.');
  END IF;
END;
```

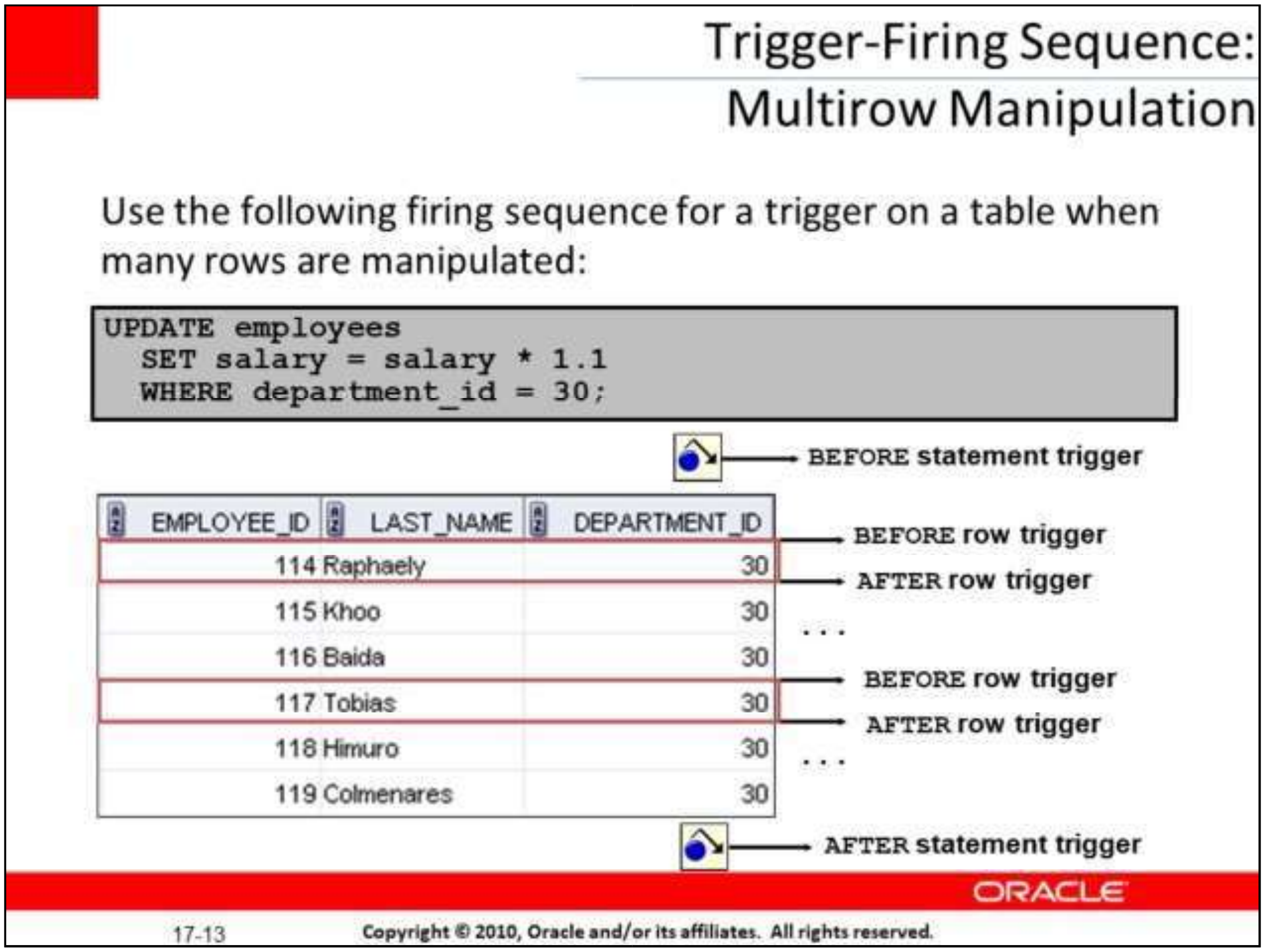
17-11

are affected or on data provided by the triggering event itself (for example, a trigger that performs a complex security check on the current user).

Row Trigger

A row trigger fires each time the table is affected by the triggering event. If the triggering event affects no rows, a row trigger is not executed. Row triggers are useful if the trigger action depends on data of the rows that are affected or on data provided by the triggering event itself.

Note: Row triggers use correlation names to access the old and new column values of the row being processed by the trigger.



Trigger-Firing Sequence: Multirow Manipulation

Use the following firing sequence for a trigger on a table when many rows are manipulated:

```
UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 30;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
114	Raphaely	30
115	Khoo	30
116	Baida	30
117	Tobias	30
118	Himuro	30
119	Colmenares	30

BEFORE statement trigger
BEFORE row trigger
AFTER row trigger
. . .
BEFORE row trigger
AFTER row trigger
. . .
AFTER statement trigger
ORACLE

17-13 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Trigger-Firing Sequence: Single-Row Manipulation

Create a statement trigger or a row trigger based on the requirement that the trigger must fire once for each row affected by the triggering statement, or just once for the triggering statement, regardless of the number of rows affected.

When the triggering DML statement affects a single row, both the statement trigger and the row trigger fire exactly once.

Example

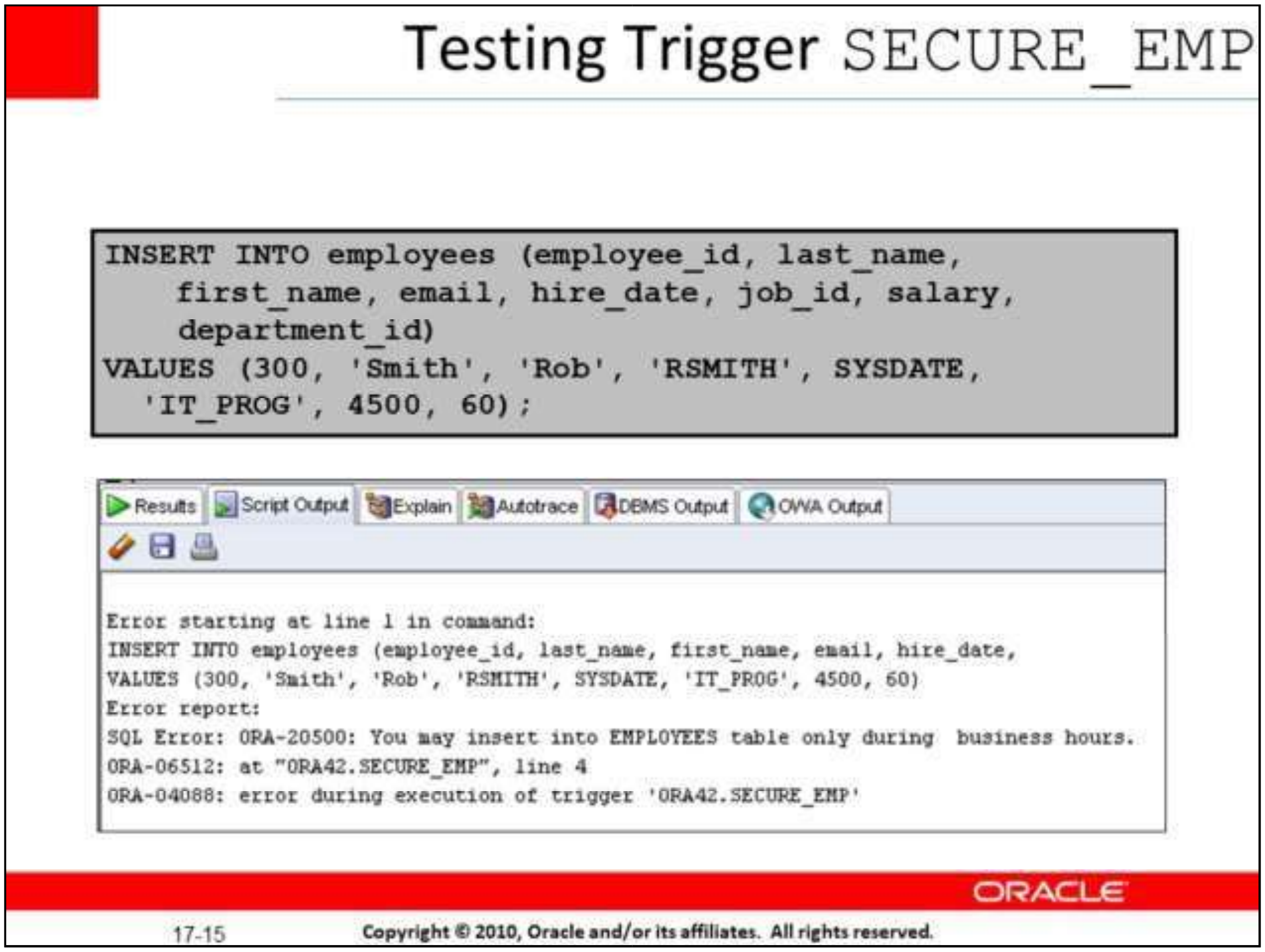
The SQL statement in the slide does not differentiate statement triggers from row triggers because exactly one row is inserted into the table using the syntax for the INSERT statement shown in the slide.

Trigger-Firing Sequence: Multirow Manipulation

When the triggering DML statement affects many rows, the statement trigger fires exactly once, and the row trigger fires once for every row affected by the statement.

Example

The SQL statement in the slide causes a row-level trigger to fire a number of times equal to the number of rows that satisfy the WHERE clause (that is, the number of employees reporting to department 30).



Testing Trigger SECURE_EMP

```
INSERT INTO employees (employee_id, last_name,
    first_name, email, hire_date, job_id, salary,
    department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
   'IT_PROG', 4500, 60);
```

```
Error starting at line 1 in command:
INSERT INTO employees (employee_id, last_name, first_name, email, hire_date,
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE, 'IT_PROG', 4500, 60)
Error report:
SQL Error: ORA-20500: You may insert into EMPLOYEES table only during  business hours.
ORA-06512: at "ORA42.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'ORA42.SECURE_EMP'
```

17-15

Creating a DML Statement Trigger

In the example in the slide, the SECURE_EMP database trigger is a BEFORE statement trigger that prevents the INSERT operation from succeeding if the business condition is violated. In this case, the trigger restricts inserts into the EMPLOYEES table during certain business hours, Monday through Friday.

If a user attempts to insert a row into the EMPLOYEES table on Saturday, then the user sees an error message, the trigger fails, and the triggering statement is rolled back. Remember that the RAISE_APPLICATION_ERROR is a server-side built-in procedure that returns an error to the user and causes the PL/SQL block to fail.

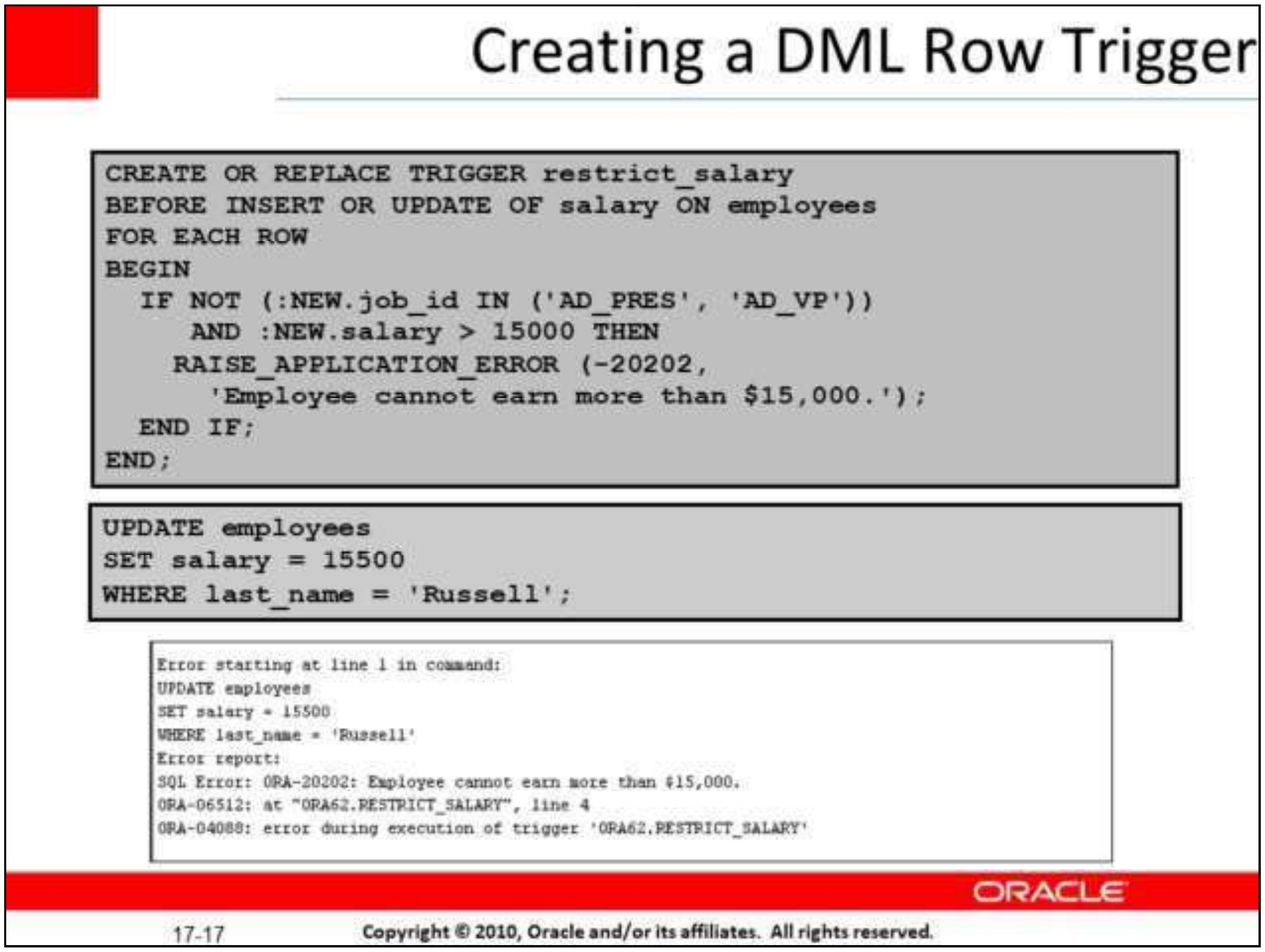
When a database trigger fails, the triggering statement is automatically rolled back by the Oracle server.

Testing SECURE_EMP

To test the trigger, insert a row into the EMPLOYEES table during nonbusiness hours. When the date and time are out of the business hours specified in the trigger, you receive the error message shown in the slide.

Detecting the DML Operation That Fired a Trigger

If more than one type of DML operation can fire a trigger (for example, ON INSERT OR



Creating a DML Row Trigger

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
  IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
     AND :NEW.salary > 15000 THEN
    RAISE_APPLICATION_ERROR (-20202,
      'Employee cannot earn more than $15,000.');
  END IF;
END;
```

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

```
Error starting at line 1 in command:
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell'
Error report:
SQL Error: ORA-20202: Employee cannot earn more than $15,000.
ORA-06512: at "ORA62.RESTRICT_SALARY", line 4
ORA-04088: error during execution of trigger 'ORA62.RESTRICT_SALARY'
```

17-17

DELETE OR UPDATE OF Emp_tab), the trigger body can use the conditional predicates INSERTING, DELETING, and UPDATING to check which type of statement fired the trigger.

You can combine several triggering events into one by taking advantage of the special conditional predicates INSERTING, UPDATING, and DELETING within the trigger body.

Example

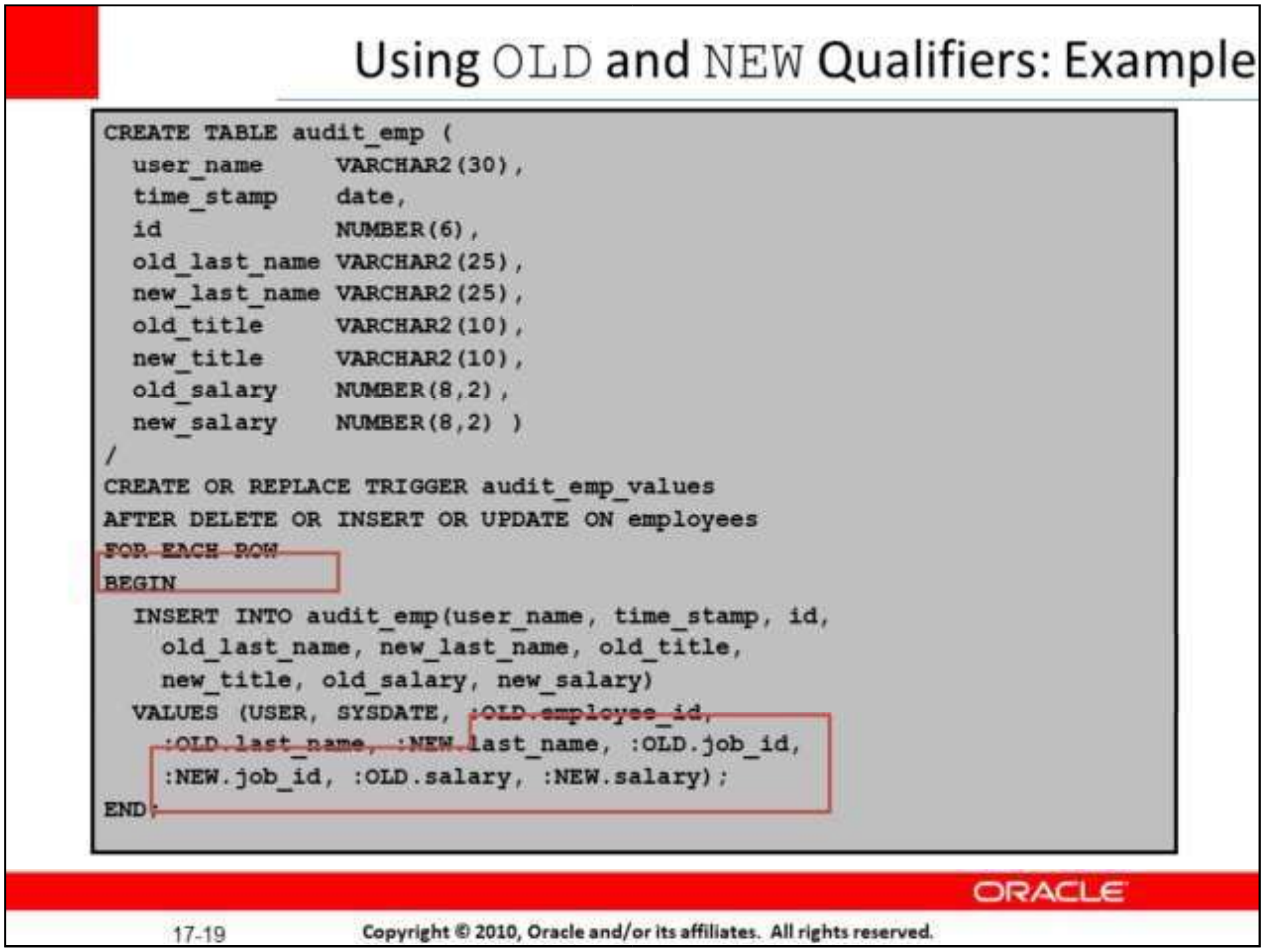
Create one trigger to restrict all data manipulation events on the EMPLOYEES table to certain business hours, 8 AM to 6 PM, Monday through Friday.

Creating a DML Row Trigger

You can create a BEFORE row trigger in order to prevent the triggering operation from succeeding if a certain condition is violated.

In the first example in the slide, a trigger is created to allow only employees whose job IDs are either AD_PRES or AD_VP to earn a salary of more than 15,000. If you try to update the salary of employee Russell whose employee ID is SA_MAN, the trigger raises the exception displayed in the slide.

Note: Before executing the first code example in the slide, make sure you disable the secure_emp and secure_employees triggers.



Using OLD and NEW Qualifiers: Example

```
CREATE TABLE audit_emp (
  user_name     VARCHAR2(30),
  time_stamp    date,
  id            NUMBER(6),
  old_last_name VARCHAR2(25),
  new_last_name VARCHAR2(25),
  old_title     VARCHAR2(10),
  new_title     VARCHAR2(10),
  old_salary    NUMBER(8,2),
  new_salary    NUMBER(8,2) )
/
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO audit_emp(user_name, time_stamp, id,
    old_last_name, new_last_name, old_title,
    new_title, old_salary, new_salary)
  VALUES (USER, SYSDATE, :OLD.employee_id,
    :OLD.last_name, :NEW.last_name, :OLD.job_id,
    :NEW.job_id, :OLD.salary, :NEW.salary);
END;
```

ORACLE

17-19

Using OLD and NEW Qualifiers

Within a ROW trigger, you can reference the value of a column before and after the data change by prefixing it with the OLD and NEW qualifiers.

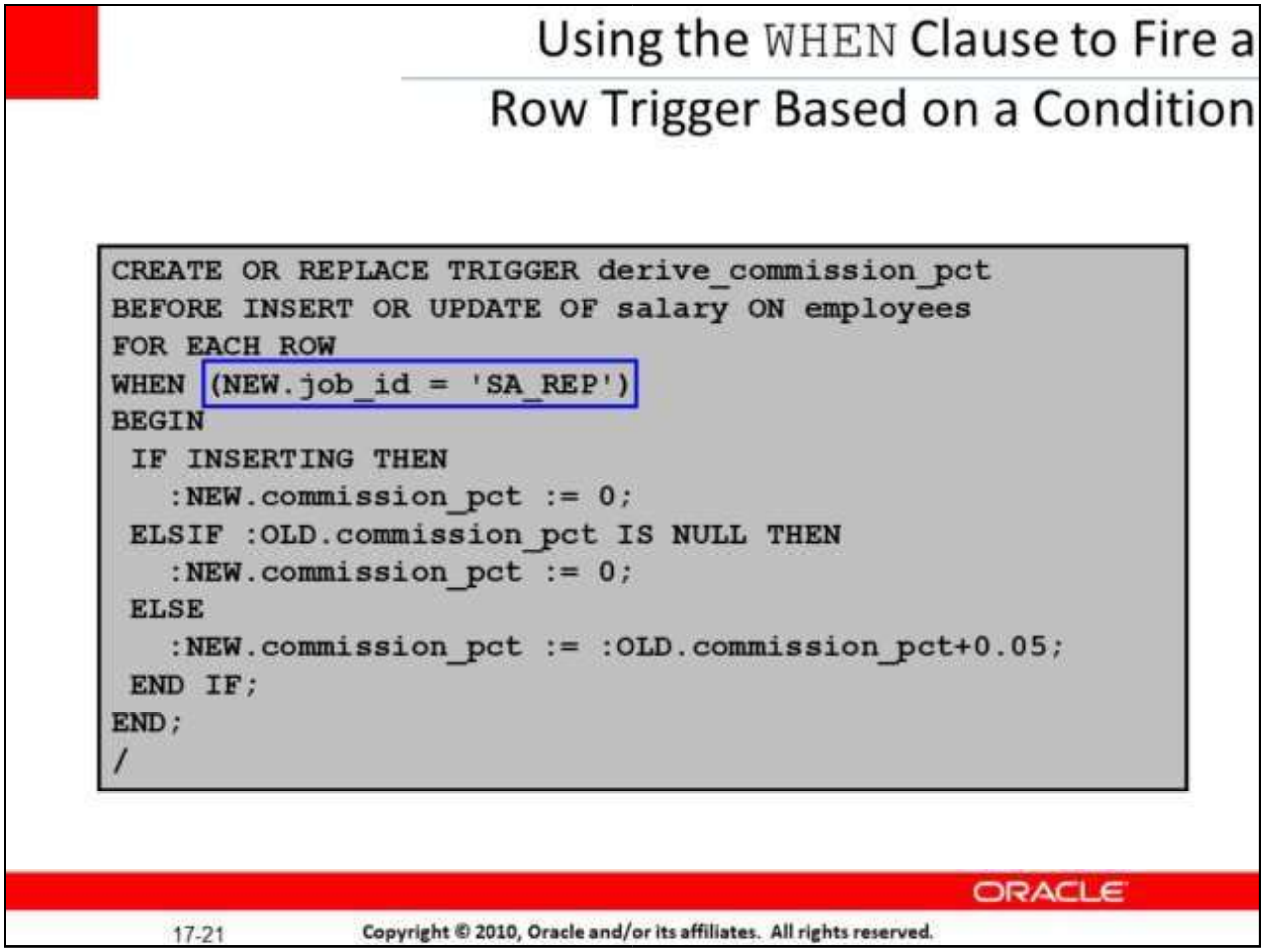
Note

- The OLD and NEW qualifiers are available only in ROW triggers.
- Prefix these qualifiers with a colon (:) in every SQL and PL/SQL statement.
- There is no colon (:) prefix if the qualifiers are referenced in the WHEN restricting condition.
- Row triggers can decrease the performance if you perform many updates on larger tables.

Using OLD and NEW Qualifiers: Example

In the example in the slide, the AUDIT_EMP_VALUES trigger is created on the EMPLOYEES table. The trigger adds rows to a user table, AUDIT_EMP, logging a user's activity against the EMPLOYEES table. The trigger records the values of several columns both before and after the data changes by using the OLD and NEW qualifiers with the respective column name. Using OLD and NEW Qualifiers: Example the Using AUDIT_EMP Table

Create a trigger on the EMPLOYEES table to add rows to a user table, AUDIT_EMP, logging a user's activity against the EMPLOYEES table. The trigger records the values of several columns



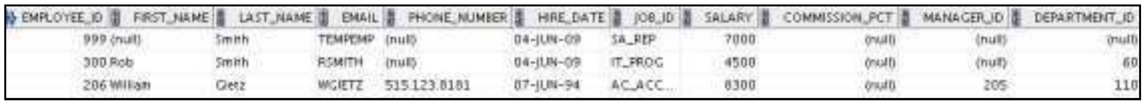
Using the WHEN Clause to Fire a Row Trigger Based on a Condition

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
 IF INSERTING THEN
   :NEW.commission_pct := 0;
 ELSIF :OLD.commission_pct IS NULL THEN
   :NEW.commission_pct := 0;
 ELSE
   :NEW.commission_pct := :OLD.commission_pct+0.05;
 END IF;
END;
/
```

17-21

both before and after the data changes by using the OLD and NEW qualifiers with the respective column name.

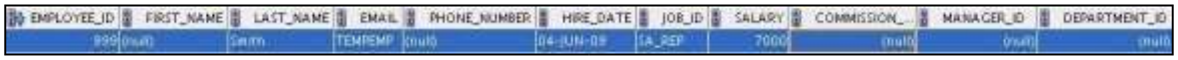
The following is the result of inserting the employee record into the EMPLOYEES table:



EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
999	(null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)
300	Rob	Smith	RSMITH	(null)	04-JUN-09	IT_PROG	4500	(null)	(null)	60
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACC...	8300	(null)	205	110

. . .

The following is the result of updating the salary for employee "Smith":



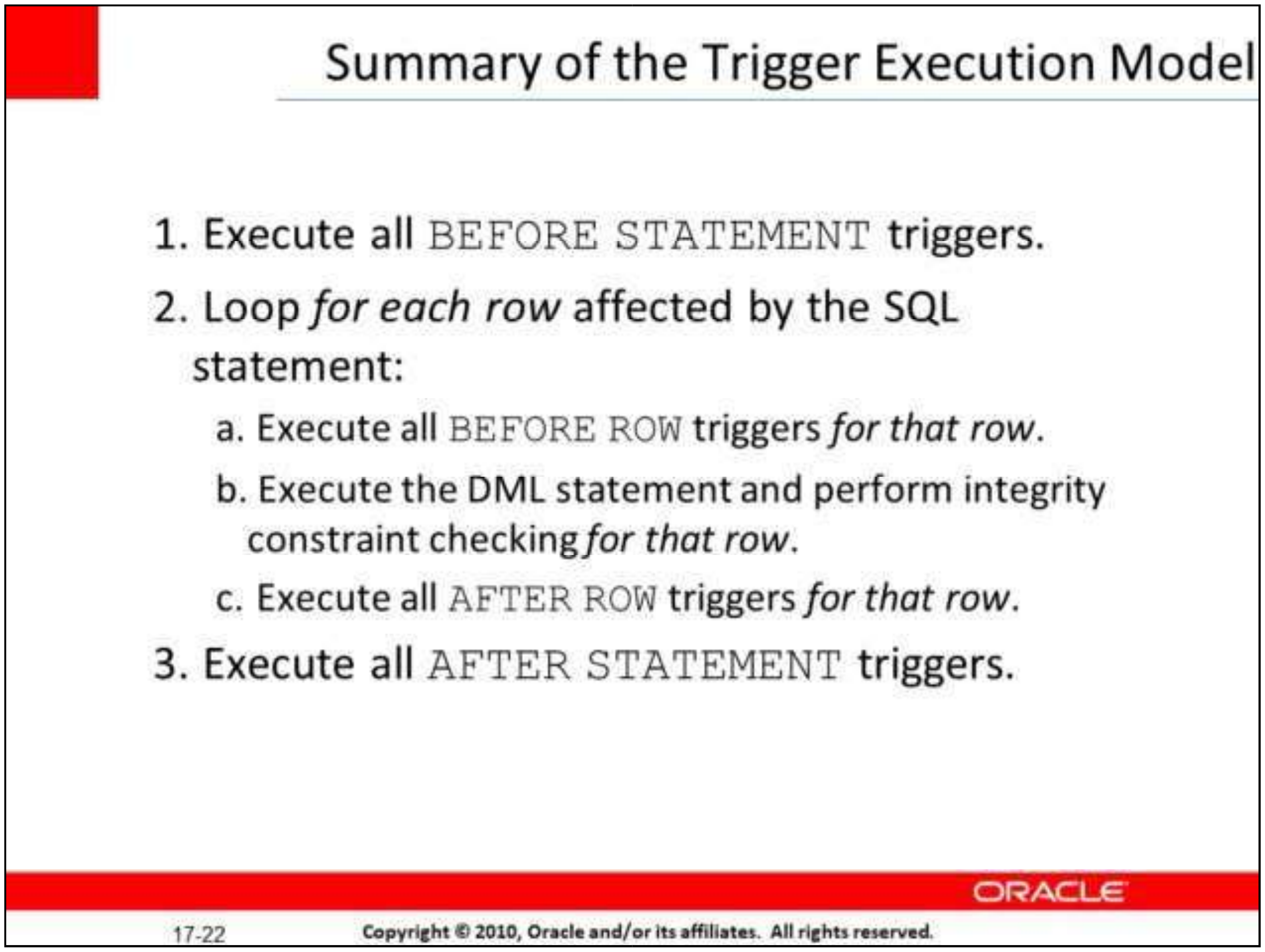
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_...	MANAGER_ID	DEPARTMENT_ID
999	(null)	Smith	TEMPEMP	(null)	04-JUN-09	SA_REP	7000	(null)	(null)	(null)

. . .

Restricting a Row Trigger: Example

Optionally, you can include a trigger restriction in the definition of a row trigger by specifying a Boolean SQL expression in a WHEN clause. If you include a WHEN clause in the trigger, then the expression in the WHEN clause is evaluated for each row that the trigger affects.

If the expression evaluates to TRUE for a row, then the trigger body executes on behalf of that row. However, if the expression evaluates to FALSE or NOTTRUE for a row (unknown, as with nulls), then the trigger body does not execute for that row. The evaluation of the WHEN clause



Summary of the Trigger Execution Model

1. Execute all BEFORE STATEMENT triggers.
2. Loop *for each row* affected by the SQL statement:
   a. Execute all BEFORE ROW triggers *for that row*.
   b. Execute the DML statement and perform integrity constraint checking *for that row*.
   c. Execute all AFTER ROW triggers *for that row*.
3. Execute all AFTER STATEMENT triggers.

17-22 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

does not have an effect on the execution of the triggering SQL statement (in other words, the triggering statement is not rolled back if the expression in a WHEN clause evaluates to FALSE).

Note: A WHEN clause cannot be included in the definition of a statement trigger.

In the example in the slide, a trigger is created on the EMPLOYEES table to calculate an employee's commission when a row is added to the EMPLOYEES table, or when an employee's salary is modified.

The NEW qualifier cannot be prefixed with a colon in the WHEN clause because the WHEN clause is outside the PL/SQL blocks.

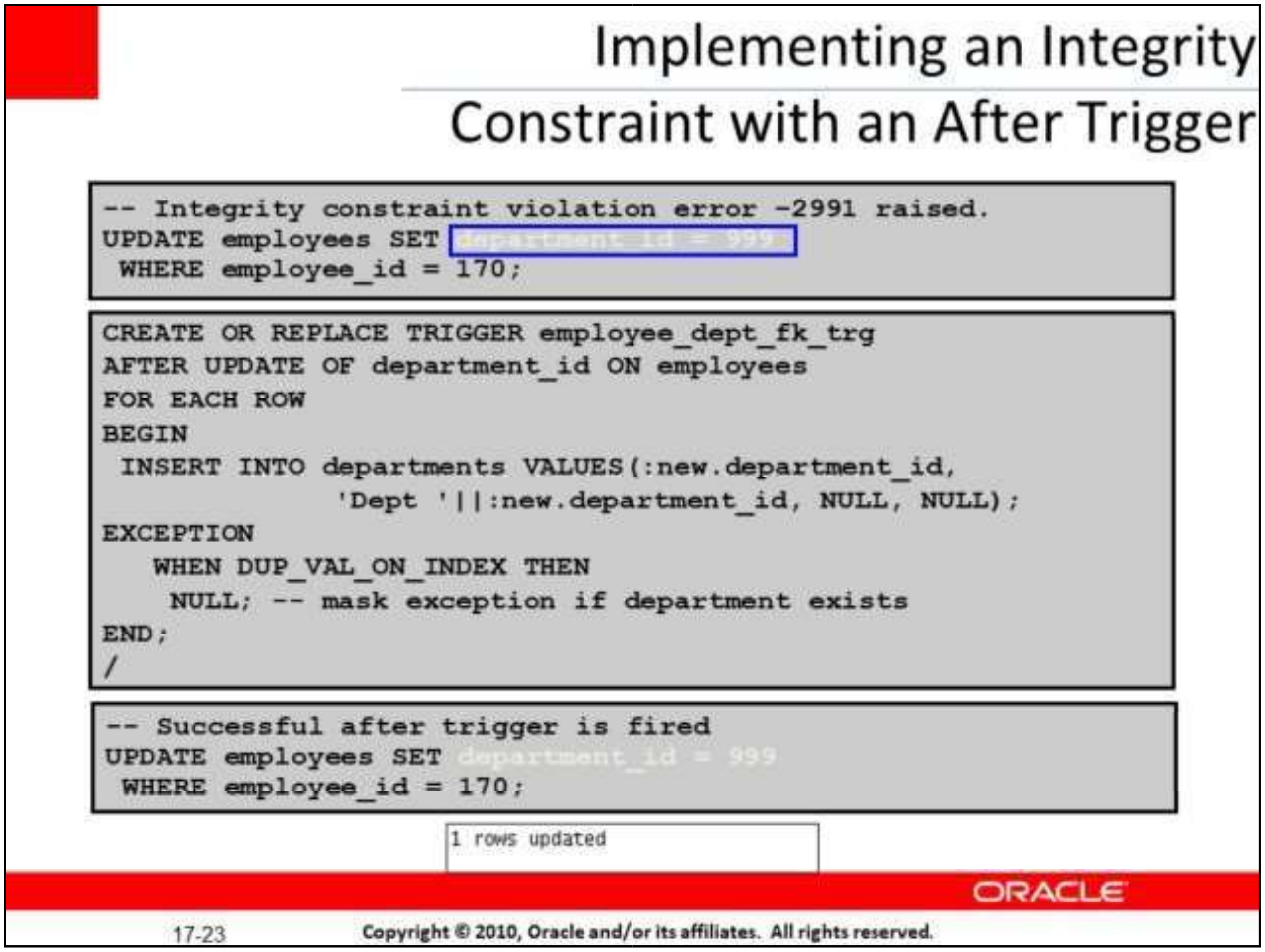
Trigger Execution Model

A single DML statement can potentially fire up to four types of triggers:

- BEFORE and AFTER statement triggers
- BEFORE and AFTER row triggers

A triggering event or a statement within the trigger can cause one or more integrity constraints to be checked. However, you can defer constraint checking until a COMMIT operation is performed.

Triggers can also cause other triggers—known as cascading triggers—to fire.



Implementing an Integrity Constraint with an After Trigger

```
-- Integrity constraint violation error -2991 raised.
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
```

```
CREATE OR REPLACE TRIGGER employee_dept_fk_trg
AFTER UPDATE OF department_id ON employees
FOR EACH ROW
BEGIN
 INSERT INTO departments VALUES(:new.department_id,
             'Dept '||:new.department_id, NULL, NULL);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
    NULL; -- mask exception if department exists
END;
/
```

```
-- Successful after trigger is fired
UPDATE employees SET department_id = 999
 WHERE employee_id = 170;
```

```
1 rows updated
```

ORACLE

17-23 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

All actions and checks performed as a result of a SQL statement must succeed. If an exception is raised within a trigger and the exception is not explicitly handled, then all actions performed because of the original SQL statement are rolled back (including actions performed by firing triggers). This guarantees that integrity constraints can never be compromised by triggers. When a trigger fires, the tables referenced in the trigger action may undergo changes by other users' transactions. In all cases, a read-consistent image is guaranteed for the modified values that the trigger needs to read (query) or write (update).

Note: Integrity checking can be deferred until the COMMIT operation is performed.

Implementing an Integrity Constraint with an After Trigger

The example in the slide explains a situation in which the integrity constraint can be taken care of by using an AFTER trigger. The EMPLOYEES table has a foreign key constraint on the DEPARTMENT_ID column of the DEPARTMENTS table.

In the first SQL statement, the DEPARTMENT_ID of the employee 170 is modified to 999. Because department 999 does not exist in the DEPARTMENTS table, the statement raises exception –2291 for the integrity constraint violation.

The EMPLOYEE_DEPT_FK_TRG trigger is created and it inserts a new row into the DEPARTMENTS table by using :NEW.DEPARTMENT_ID for the value of the new department's
DEPARTMENT_ID. The trigger fires when the UPDATE statement modifies the DEPARTMENT_ID of employee 170 to 999. When the foreign key constraint is checked, it is successful because the trigger inserted the department 999 into the DEPARTMENTS table. Therefore, no exception occurs unless the department already exists when the trigger attempts to

insert the new row. However, the EXCEPTION handler traps and masks the exception allowing the operation to succeed.

Note: Although the example shown in the slide is somewhat contrived due to the limited data in the HR schema, the point is that if you defer the constraint check until the commit, you then have the ability to engineer a trigger to detect that constraint failure and repair it prior to the commit acion.