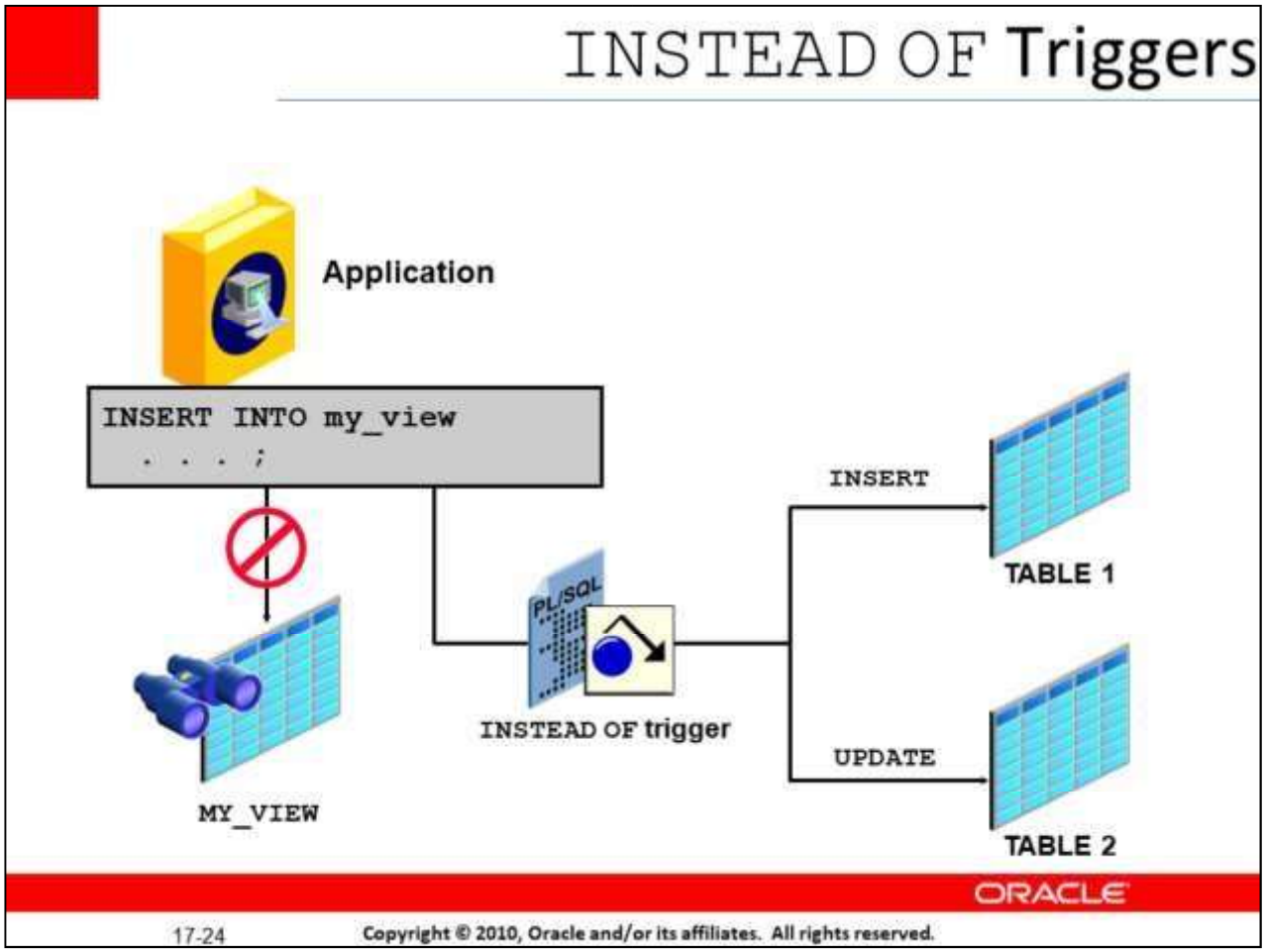
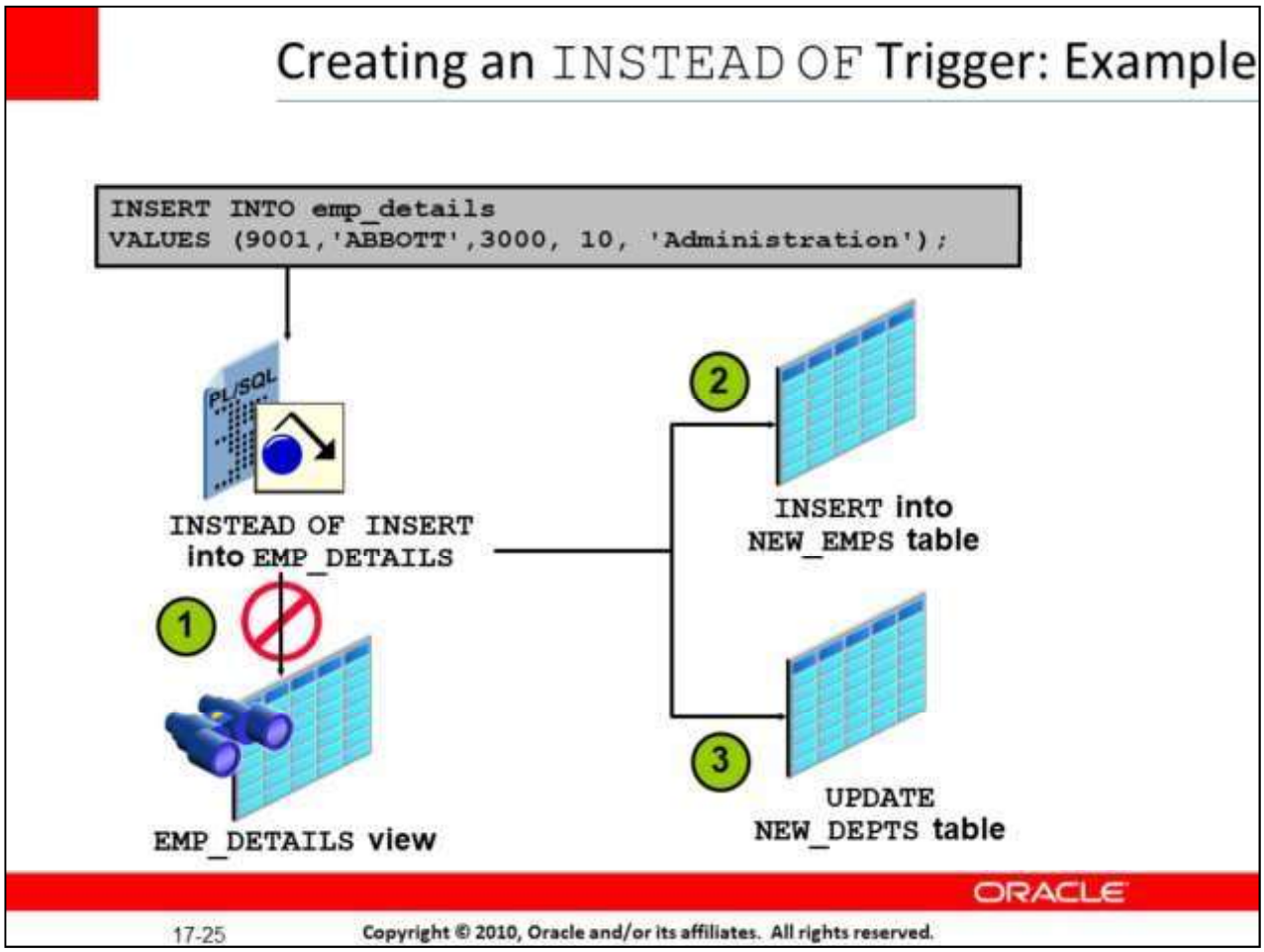


INSTEADOF Triggers

Use INSTEADOF triggers to modify data in which the DML statement has been issued against an inherently un-updatable view. These triggers are called INSTEADOF triggers because, unlike other triggers, the Oracle server fires the trigger instead of executing the triggering statement. These triggers are used to perform INSERT, UPDATE, and DELETE operations directly on the underlying tables. You can write INSERT, UPDATE, and DELETE statements against a view, and the INSTEADOF trigger works invisibly in the background to make the right actions take place. A view cannot be modified by normal DML statements if the view query contains set operators, group functions, clauses such as GROUPBY, CONNECTBY, START, the DISTINCT operator, or joins. For example, if a view consists of more than one table, an insert to the view may entail an insertion into one table and an update to another. So you write an INSTEADOF trigger that fires when you write an insert against the view. Instead of the original insertion, the trigger body executes, which results in an insertion of data into one table and an update to another table.

Note: If a view is inherently updatable and has INSTEADOF triggers, then the triggers take precedence. INSTEADOF triggers are row triggers. The CHECK option for views is not enforced when insertions or updates to the view are performed by using INSTEADOF triggers.

The INSTEADOF trigger body must enforce the check.



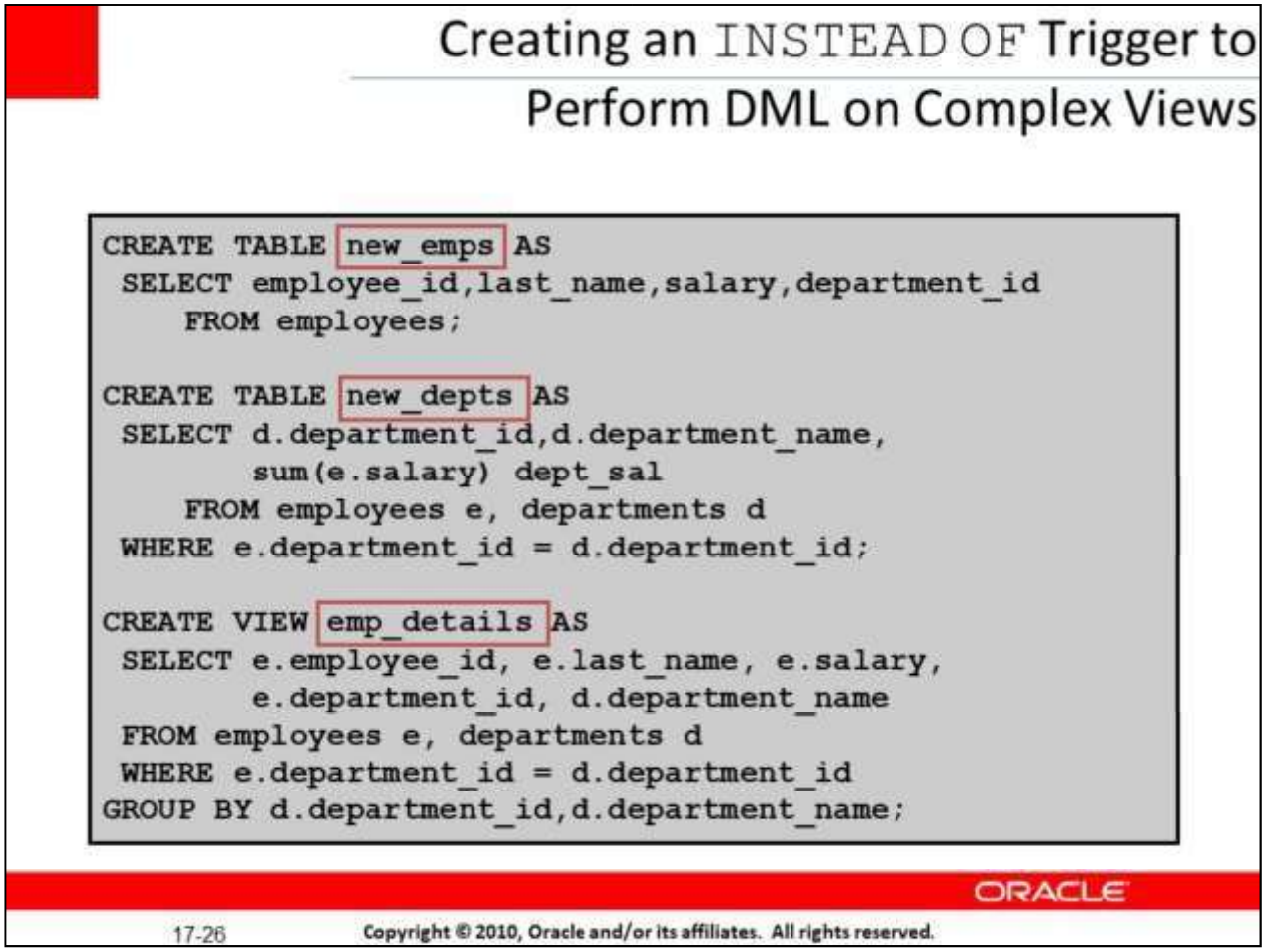
Creating an INSTEADOF Trigger

You can create an INSTEADOF trigger in order to maintain the base tables on which a view is based.

The example in the slide illustrates an employee being inserted into the view EMP_DETAILS, whose query is based on the EMPLOYEES and DEPARTMENTS tables. The NEW_EMP_DEPT (INSTEAD OF) trigger executes in place of the INSERT operation that causes the trigger to fire. The INSTEADOF trigger then issues the appropriate INSERT and UPDATE to the base tables used by the EMP_DETAILS view. Therefore, instead of inserting the new employee record into the EMPLOYEES table, the following actions take place:

1. The NEW_EMP_DEPTINSTEAD OF trigger fires.
2. A row is inserted into the NEW_EMPS table.
3. The DEPT_SAL column of the NEW_DEPTS table is updated. The salary value supplied for the new employee is added to the existing total salary of the department to which the new employee has been assigned.

Note: Before you run the example in the slide, you must create the required structures shown on the next two pages.



Creating an INSTEADOF Trigger (continued)

The example in the slide creates two new tables, NEW_EMPS and NEW_DEPTS, that are based on the EMPLOYEES and DEPARTMENTS tables, respectively. It also creates an EMP_DETAILS view from the EMPLOYEES and DEPARTMENTS tables.

If a view has a complex query structure, then it is not always possible to perform DML directly on the view to affect the underlying tables. The example requires creation of an INSTEADOF trigger, called NEW_EMP_DEPT, shown on the next page. The NEW_DEPT_EMP trigger handles DML in the following way:

- When a row is inserted into the EMP_DETAILS view, instead of inserting the row directly into the view, rows are added into the NEW_EMPS and NEW_DEPTS tables, using the data values supplied with the INSERT statement.
- When a row is modified or deleted through the EMP_DETAILS view, corresponding rows in the NEW_EMPS and NEW_DEPTS tables are affected.

Note: INSTEADOF triggers can be written only for views, and the BEFORE and AFTER timing options are not valid.

Creating an INSTEADOF Trigger (continued)

```
CREATE OR REPLACE TRIGGER new_emp_dept
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_details
FOR EACH ROW
BEGIN
   IF INSERTING THEN INSERT
      INTO new_emps
      VALUES (:NEW.employee_id, :NEW.last_name,
                  :NEW.salary, :NEW.department_id); UPDATE
      new_depts
         SET dept_sal = dept_sal + :NEW.salary
         WHERE department_id = :NEW.department_id;
   ELSIF DELETING THEN
      DELETE FROM new_emps
          WHERE employee_id = :OLD.employee_id; UPDATE new_depts
         SET dept_sal = dept_sal - :OLD.salary
         WHERE department_id = :OLD.department_id;
   ELSIF UPDATING ('salary') THEN
      UPDATE new_emps
         SET salary = :NEW.salary
         WHERE employee_id = :OLD.employee_id;
      UPDATE new_depts
               SET dept_sal = dept_sal +
                                 (:NEW.salary - :OLD.salary) WHERE
         department_id = :OLD.department_id;
   ELSIF UPDATING ('department_id') THEN
      UPDATE new_emps
         SET department_id = :NEW.department_id
          WHERE employee_id = :OLD.employee_id; UPDATE new_depts
         SET dept_sal = dept_sal - :OLD.salary
         WHERE department_id = :OLD.department_id;
      UPDATE new_depts
         SET dept_sal = dept_sal + :NEW.salary
          WHERE department_id = :NEW.department_id; END
   IF;
END;
/
```

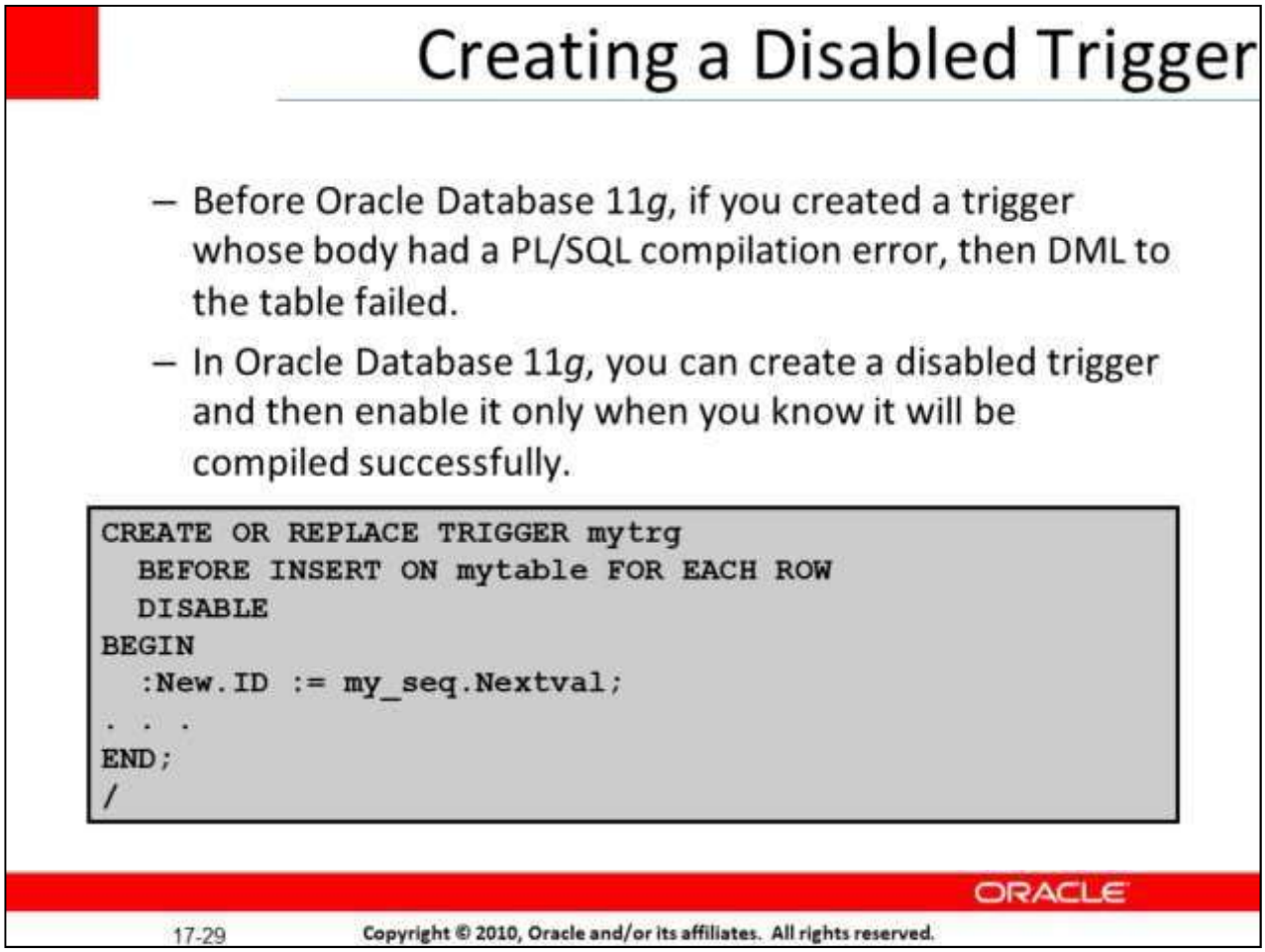
```
DEPARTMENT_ID        DEPARTMENT_NAME                DEPT_SAL
-------------------- ------------------------------ --------------------
10                   Administration                 7400

1 rows selected

EMPLOYEE_ID          LAST_NAME                SALARY               DEPARTMENT_ID
-------------------- ------------------------ -------------------- --------------------
200                  Whalen                   4400                 10
9001                 ABBOTT                   3000                 10

2 rows selected
```

The Status of a Trigger

A trigger is in either of two distinct modes:

- Enabled: The trigger runs its trigger action if a triggering statement is issued and the trigger restriction (if any) evaluates to true (default).
- Disabled: The trigger does not run its trigger action, even if a triggering statement is issued and the trigger restriction (if any) would evaluate to true.

ORACLE



Creating a Disabled Trigger

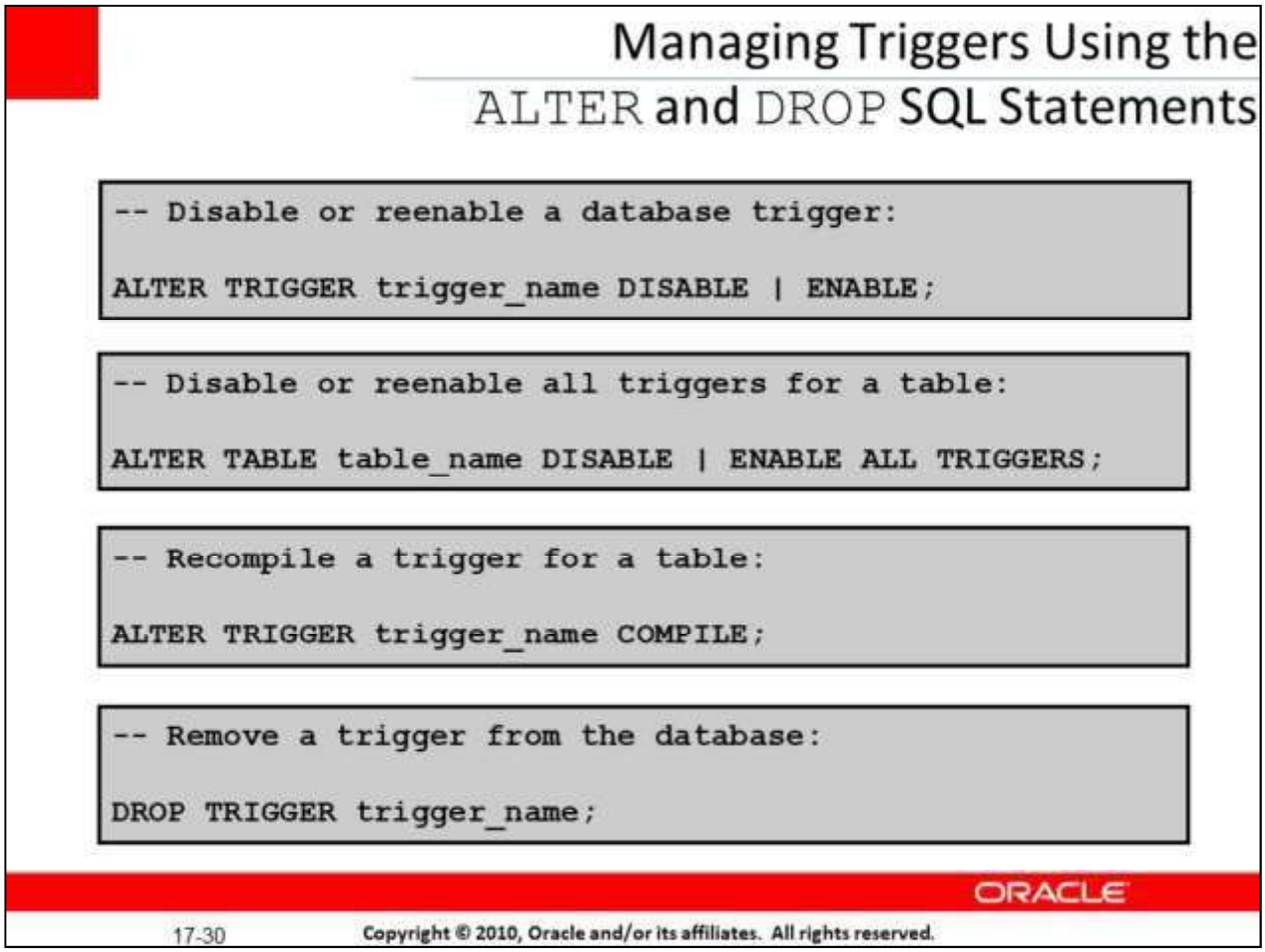
Before Oracle Database 11g, if you created a trigger whose body had a PL/SQL compilation error, then DML to the table failed. The following error message was displayed:
ORA-04098: trigger 'TRG' is invalid and failed re-validation

In Oracle Database 11g, you can create a disabled trigger, and then enable it only when you know it will be compiled successfully.

You can also temporarily disable a trigger in the following situations:

- An object it references is not available.
- You need to perform a large data load, and you want it to proceed quickly without firing triggers.
- You are reloading data.

Note: The code example in the slide assumes that you have an existing sequence named my_seq.



Managing Triggers

A trigger has two modes or states: ENABLED and DISABLED. When a trigger is first created, it is enabled by default. The Oracle server checks integrity constraints for enabled triggers and guarantees that triggers cannot compromise them. In addition, the Oracle server provides readconsistent views for queries and constraints, manages the dependencies, and provides a twophase commit process if a trigger updates remote tables in a distributed database.

Disabling a Trigger

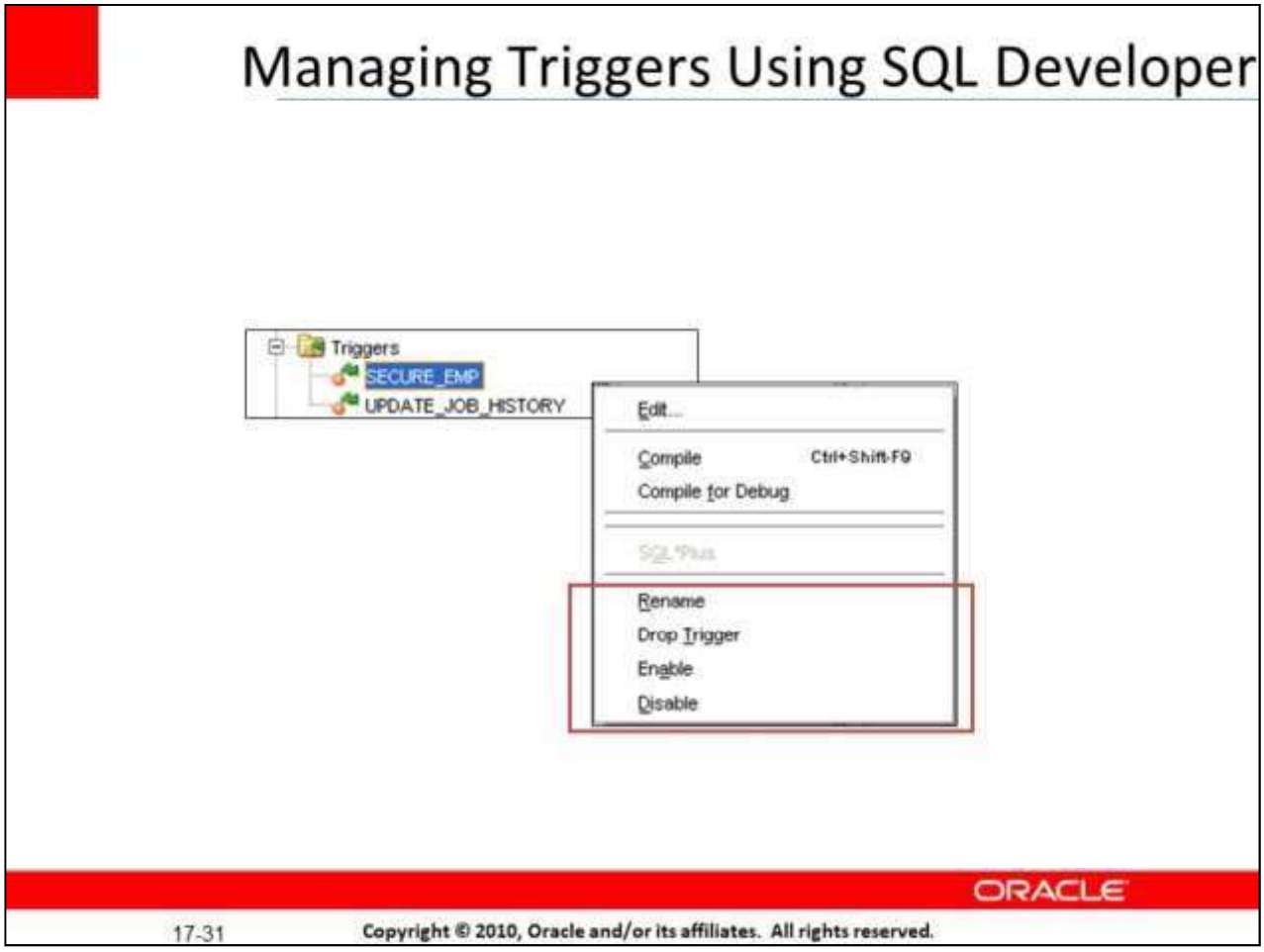
Use the ALTERTRIGGER command to disable a trigger. You can also disable all triggers on a table by using the ALTERTABLE command. You can disable triggers to improve performance or to avoid data integrity checks when loading massive amounts of data with utilities such as SQL*Loader. You might also disable a trigger when it references a database object that is currently unavailable, due to a failed network connection, disk crash, offline data file, or offline tablespace.

Recompiling a Trigger

Use the ALTERTRIGGER command to explicitly recompile a trigger that is invalid.

Removing Triggers

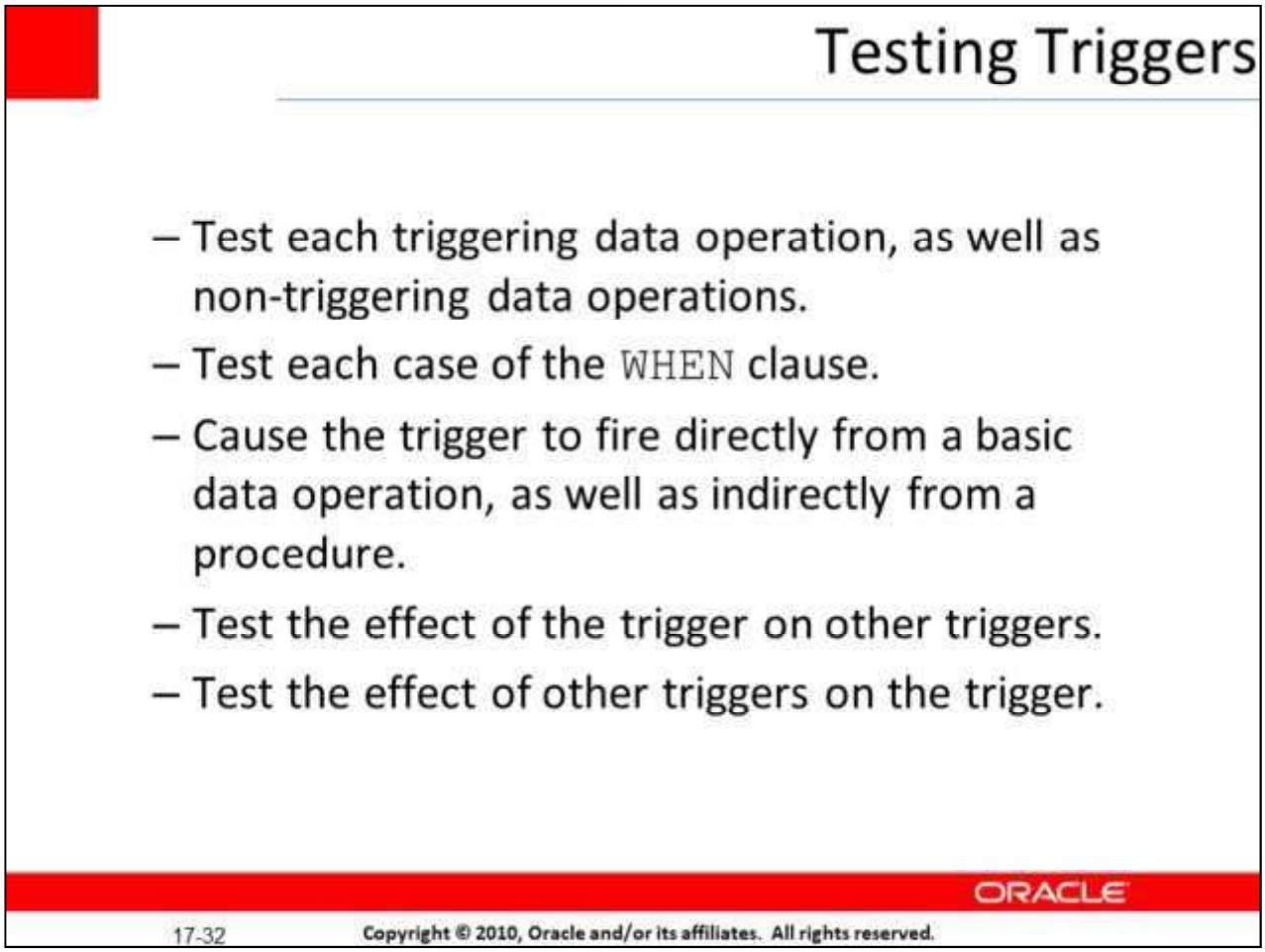
When a trigger is no longer required, use a SQL statement in SQL Developer or SQL*Plus to remove it. When you remove a table, all triggers on that table are also removed.



Managing Triggers Using SQL Developer

You can use the Triggers node in the Connections navigation tree to manage triggers. Right-click a trigger name, and then select one of the following options:

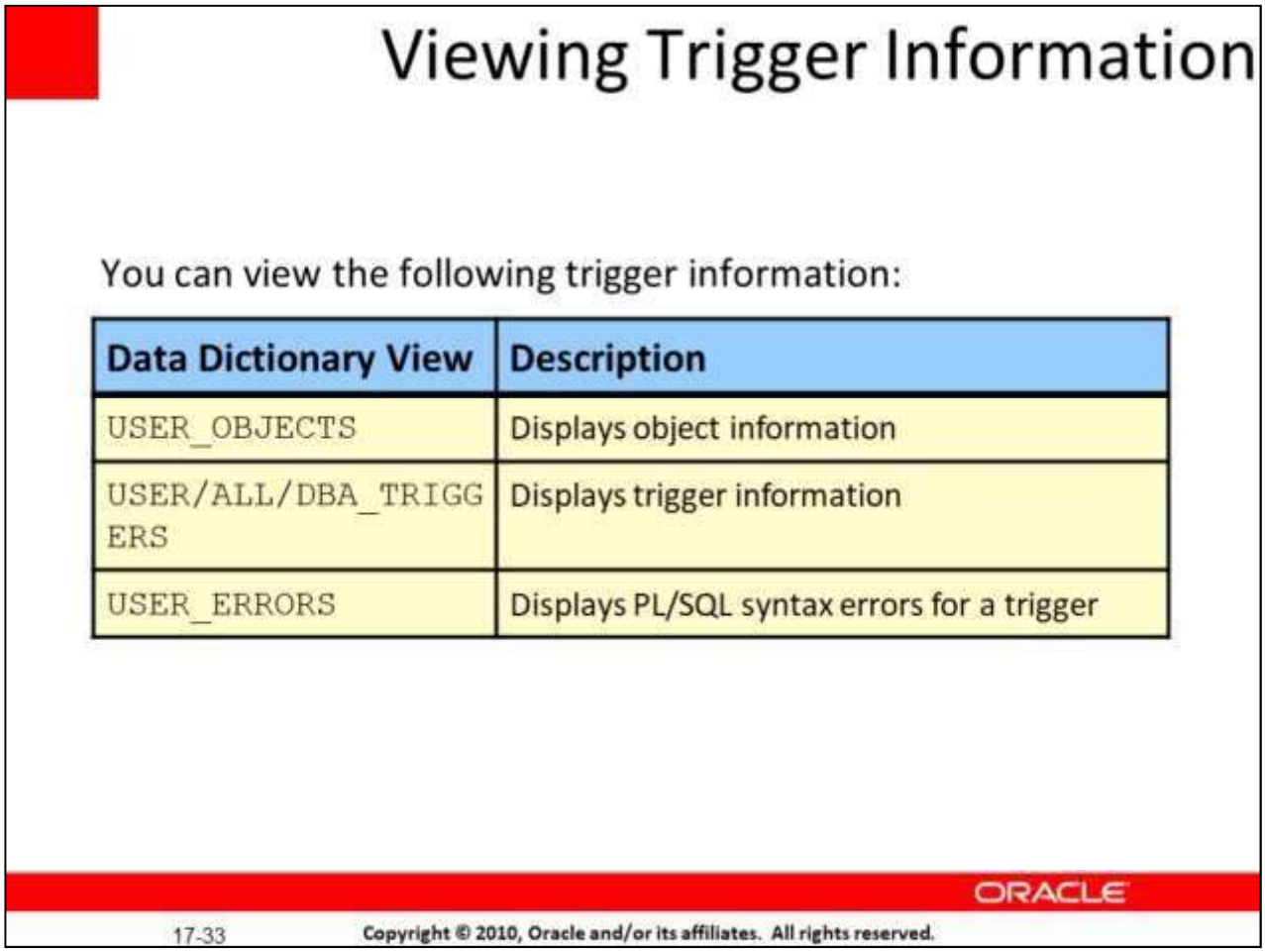
- Edit
- Compile
- Compile for Debug
- Rename
- Drop Trigger
- Enable
- Disable



Testing Triggers

Testing code can be a time-consuming process. Do the following when testing triggers:

- Ensure that the trigger works properly by testing a number of cases separately: - Test the most common success scenarios first.
  - Test the most common failure conditions to see that they are properly managed.
- The more complex the trigger, the more detailed your testing is likely to be. For example, if you have a row trigger with a WHEN clause specified, then you should ensure that the trigger fires when the conditions are satisfied. Or, if you have cascading triggers, you need to test the effect of one trigger on the other and ensure that you end up with the desired results.
- Use the DBMS_OUTPUT package to debug triggers.



Viewing Trigger Information

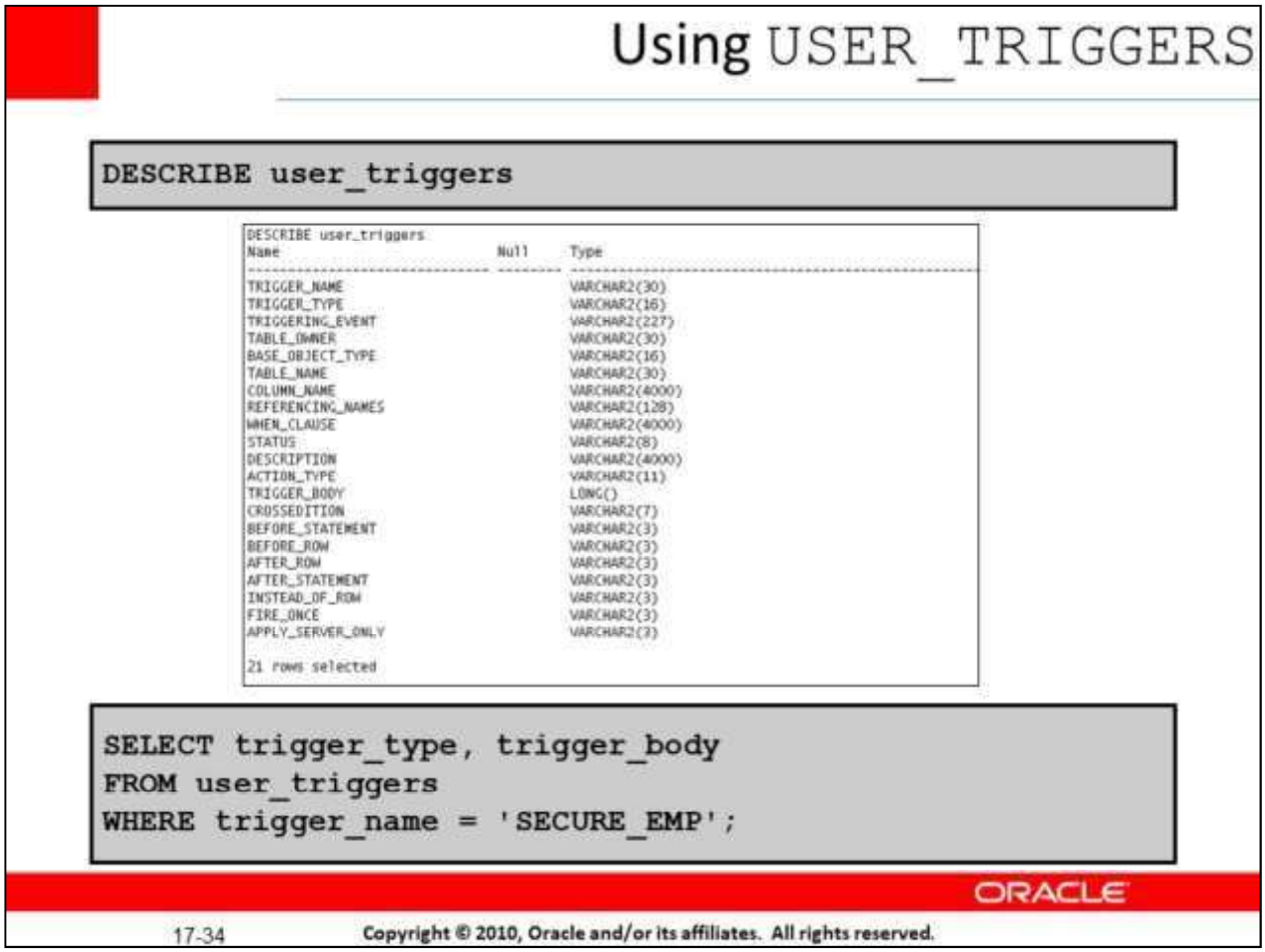
The slide shows the data dictionary views that you can access to get information regarding the triggers.

The USER_OBJECTS view contains the name and status of the trigger and the date and time when the trigger was created.

The USER_ERRORS view contains the details about the compilation errors that occurred while a trigger was compiling. The contents of these views are similar to those for subprograms.

The USER_TRIGGERS view contains details such as name, type, triggering event, the table on which the trigger is created, and the body of the trigger.

The SELECTUsernameFROM USER_USERS; statement gives the name of the owner of the trigger, not the name of the user who is updating the table.



Using USER_TRIGGERS

If the source file is unavailable, then you can use the SQL Worksheet in SQL Developer or SQL*Plus to regenerate it from USER_TRIGGERS. You can also examine the ALL_TRIGGERS and DBA_TRIGGERS views, each of which contains the additional column OWNER, for the owner of the object. The result for the second example in the slide is as follows:

```
TRIGGER_TYPE     TRIGGER_BODY
---------------- --------------------------------------------------------------------------------
BEFORE STATEMENT BEGIN
                  IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
                     (TO_CHAR(SYSDATE,'HH24') NOT BETWEEN '08' AND '18') THEN
                    IF DELETING THEN
                      RAISE_APPLICATION_ERROR(-20502,
                        'You may delete from EMPLOYEES table only during normal business hours.');
                    ELSIF INSERTING THEN
                      RAISE_APPLICATION_ERROR(-20500,
                        'You may insert into EMPLOYEES table only during normal business hours.');
                    ELSIF UPDATING('SALARY') THEN
                      RAISE_APPLICATION_ERROR(-20503,
                        'You may update SALARY only during normal business hours.');
                    ELSE
                      RAISE_APPLICATION_ERROR(-20504,
                        'You may update EMPLOYEES table only during normal business hours.');
                    END IF;
                  END IF;
                 END;
```

Answer: 1, 2, 3, 4, 5