PLSQL

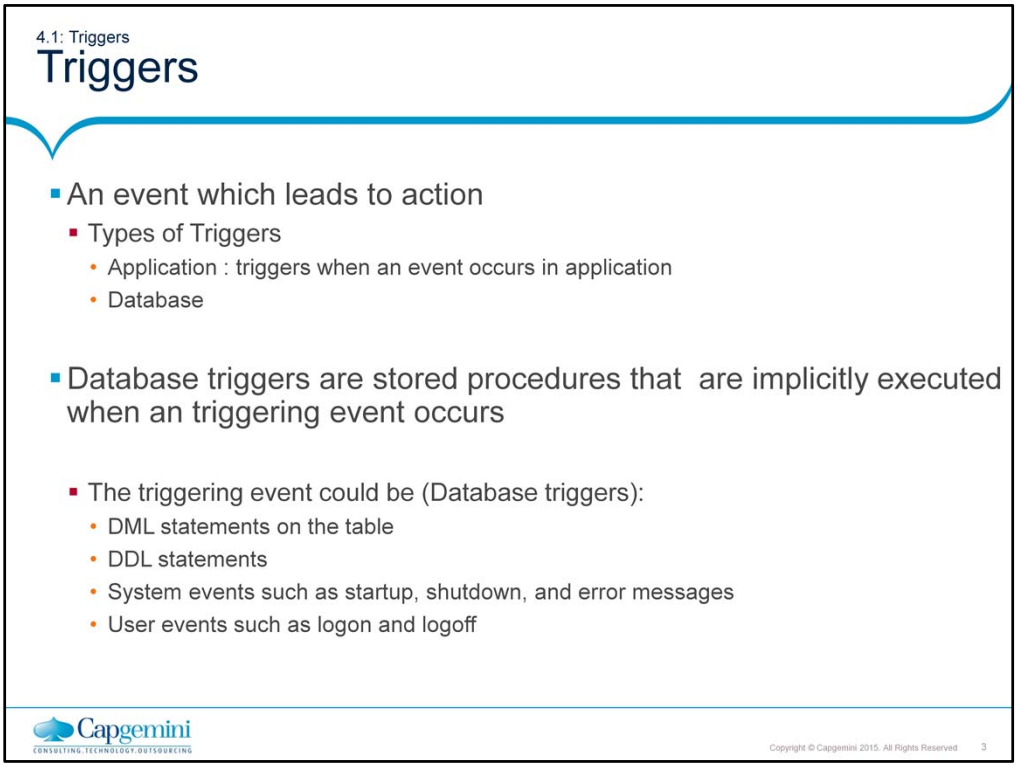
Lesson 04 :Triggers and its types

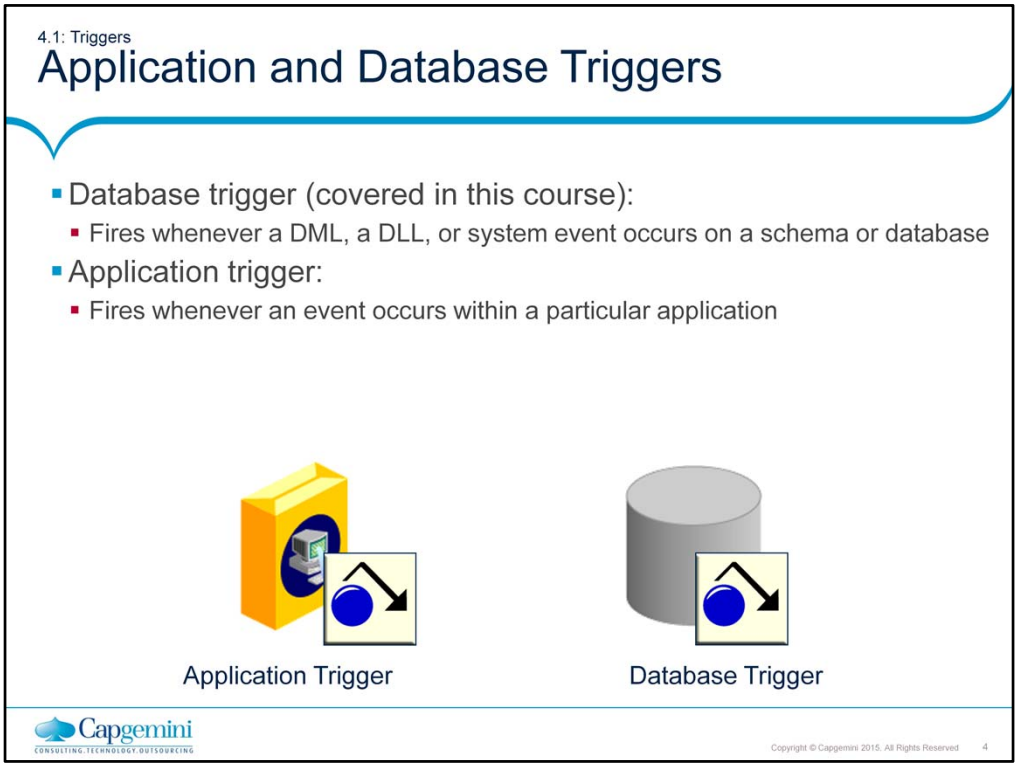
Lesson Objectives

- To understand the following topics:
  - Describe database triggers and their uses
  - Describe the different types of triggers
  - Create database triggers
  - Describe database trigger-firing rules
  - Remove database triggers
  - Display trigger information

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2


4.1: Triggers
Triggers
An event which leads to action
Types of Triggers
Application : triggers when an event occurs in application
Database
Database triggers are stored procedures that are implicitly executed when an triggering event occurs
The triggering event could be (Database triggers):
DML statements on the table
DDL statements
System events such as startup, shutdown, and error messages
User events such as logon and logoff
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
3



Types of Triggers

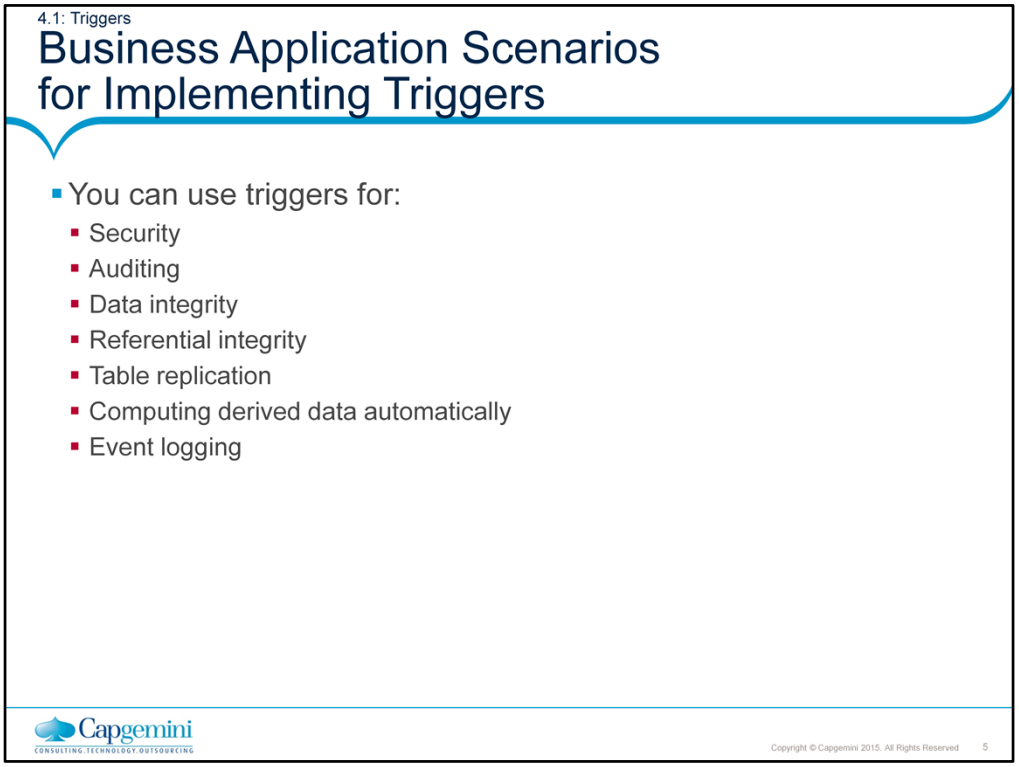
Application triggers execute implicitly whenever a particular data manipulation language (DML) event occurs within an application. An example of an application that uses triggers extensively is an application developed with Oracle Forms Developer.

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

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

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).

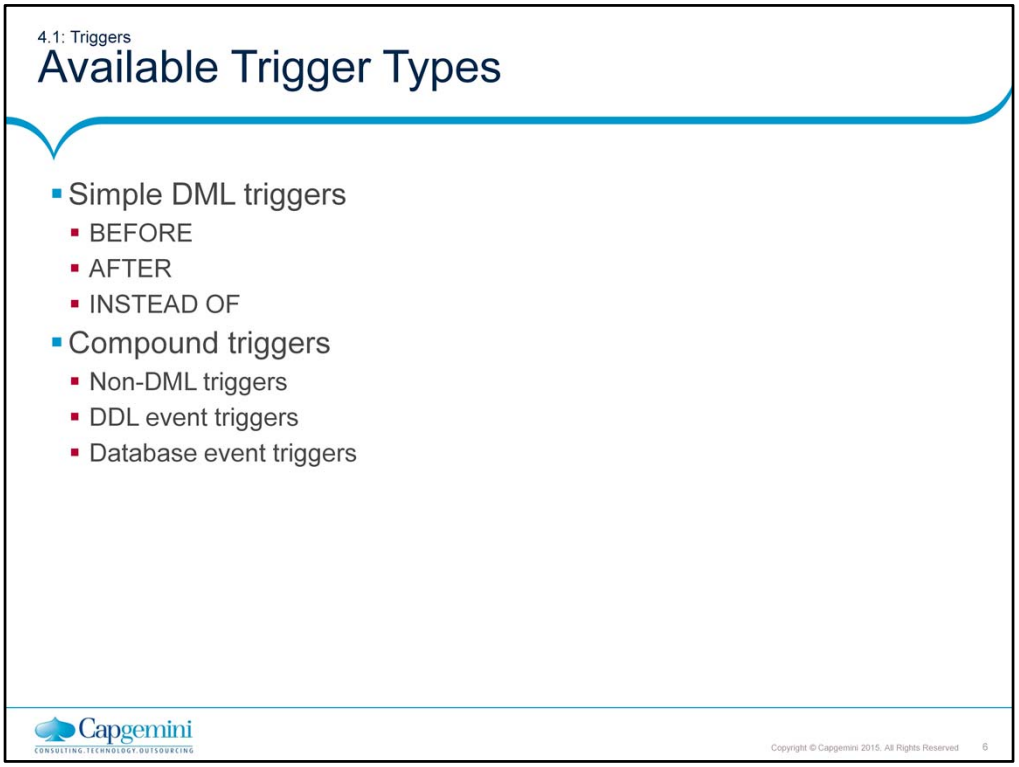
Database triggers can be system triggers on a database or a schema (covered in the next lesson). For databases, triggers fire for each event for all users; for a schema, they fire for each event for that specific user. Oracle Forms can define, store, and run triggers of a different sort. However, do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



Business Application Scenarios for Implementing Triggers

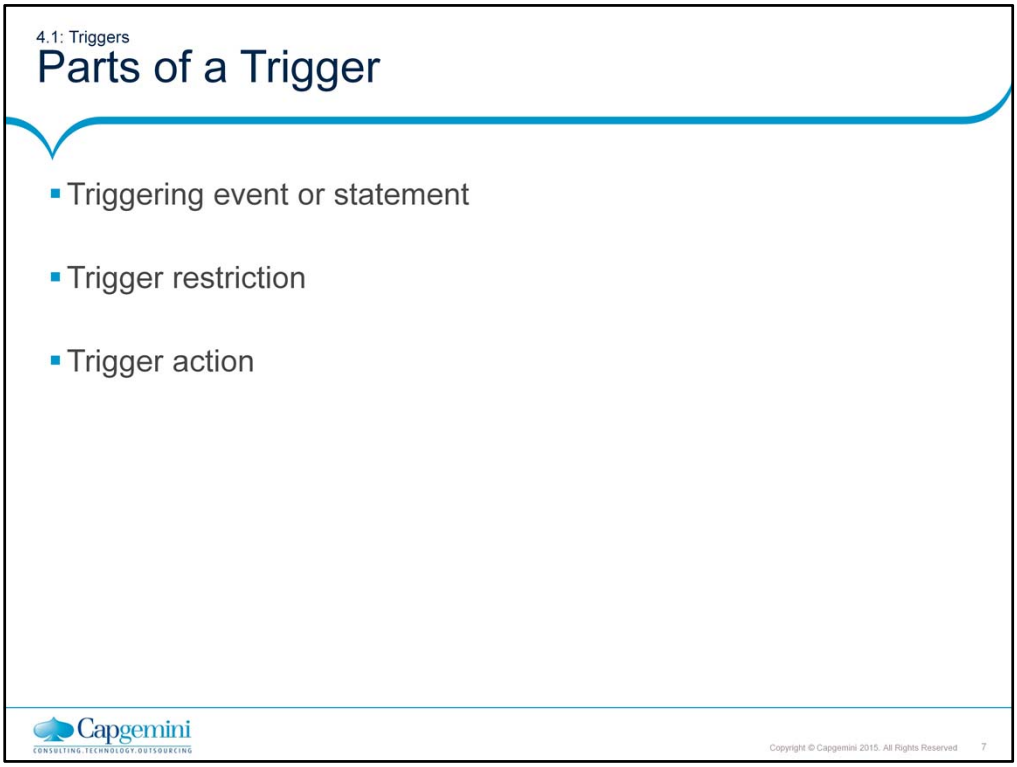
Develop database triggers in order to enhance features that cannot otherwise be implemented by the Oracle server or as alternatives to those provided by the Oracle server.

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.
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 `INSTEAD OF` triggers. The other trigger types are discussed in the lesson titled “Creating Compound, DDL, and Event Database Triggers.”


4.1: Triggers
Parts of a Trigger
Triggering event or statement
Trigger restriction
Trigger action
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
7

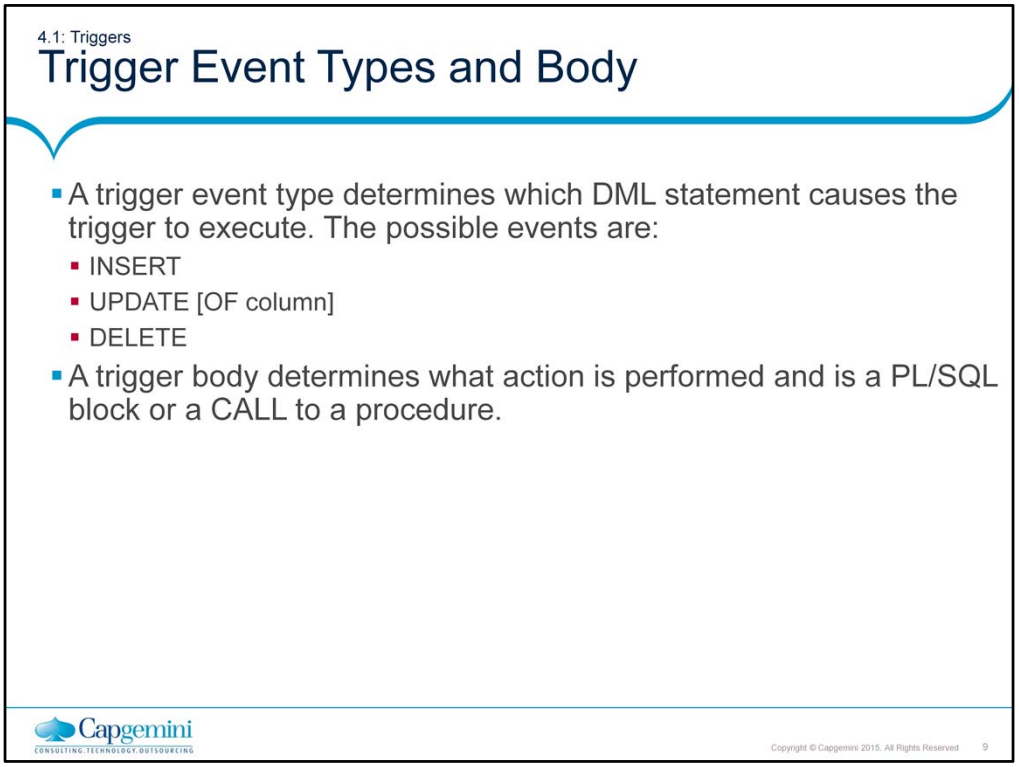
4.1: Triggers

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/view
- 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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8



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.

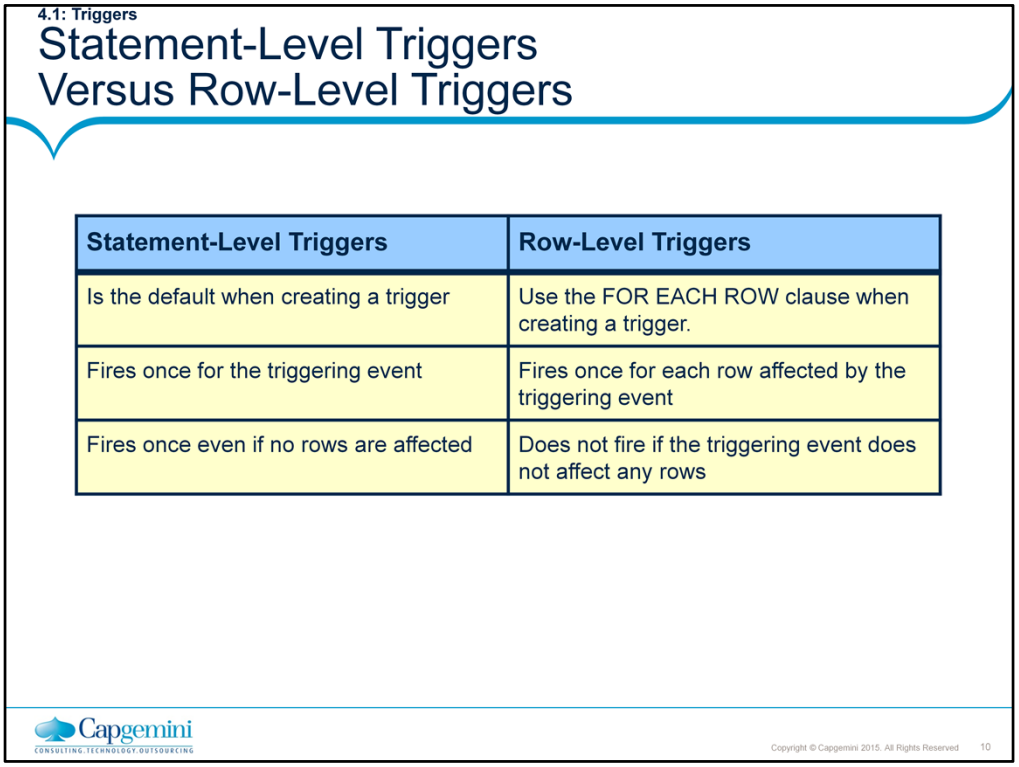
. . . UPDATE OF salary . . .

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.



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 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.

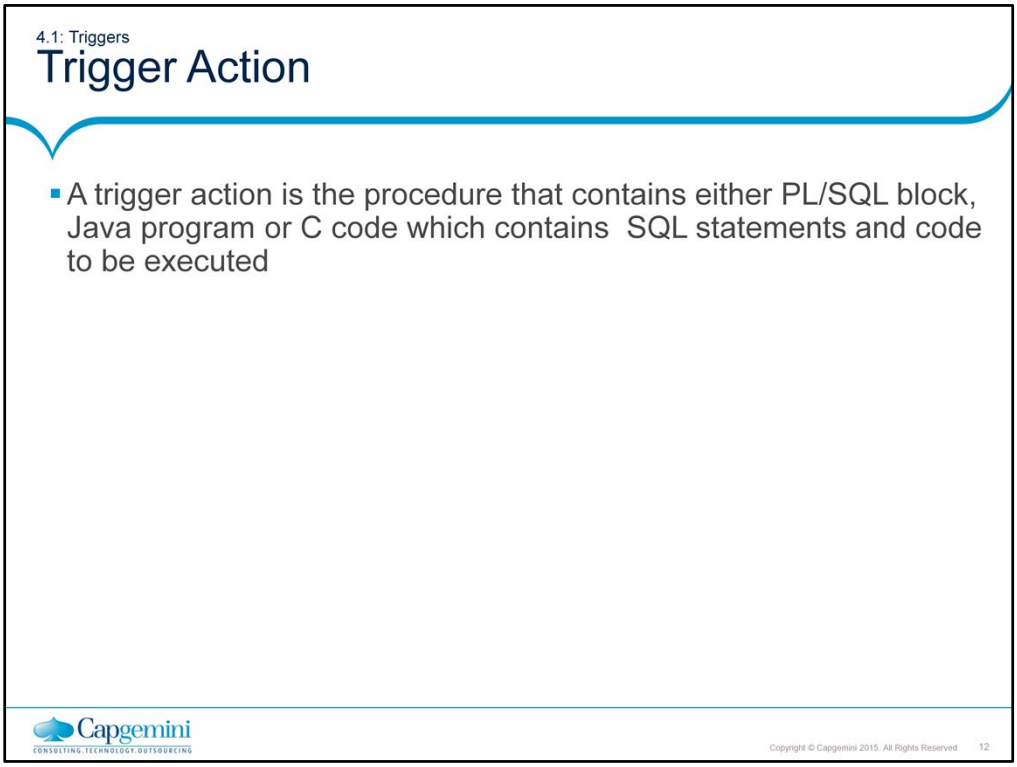
4.1: Triggers

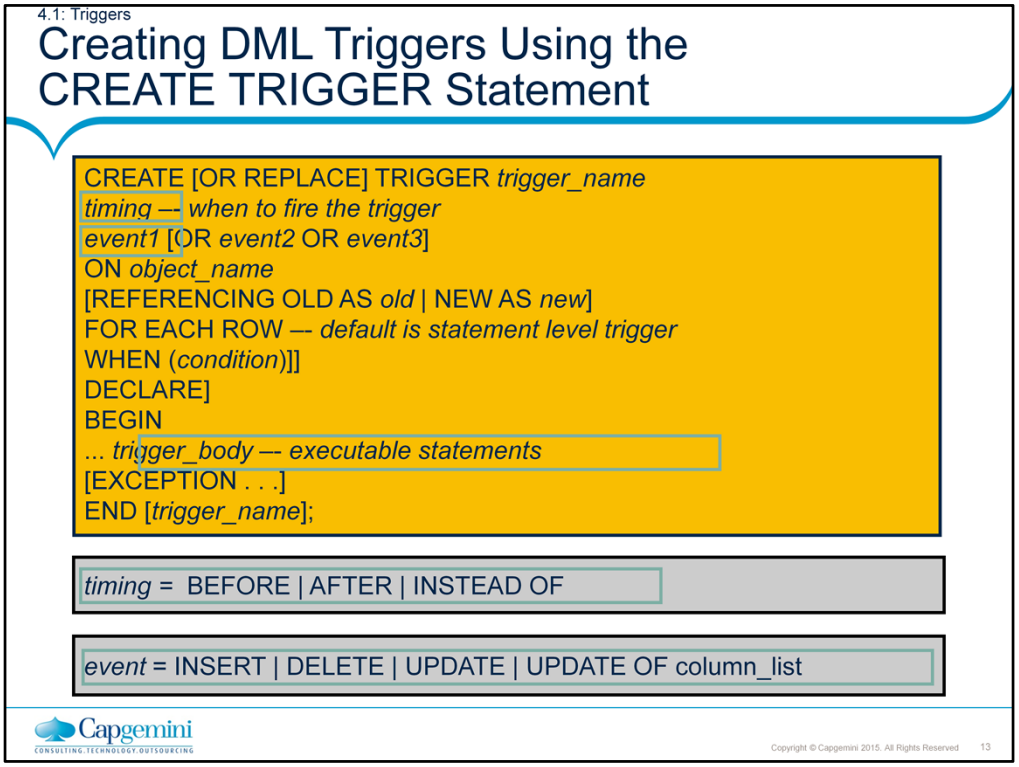
Trigger Restriction

- Specifies a Boolean expression that must evaluate to TRUE for the trigger to fire

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11


4.1: Triggers
Trigger Action
A trigger action is the procedure that contains either PL/SQL block, Java program or C code which contains SQL statements and code to be executed
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
12



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 INSTEAD OF.
- event identifies the DML operation causing the trigger to fire. Values are INSERT, UPDATE [OF column], 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)
- FOR EACH ROW 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

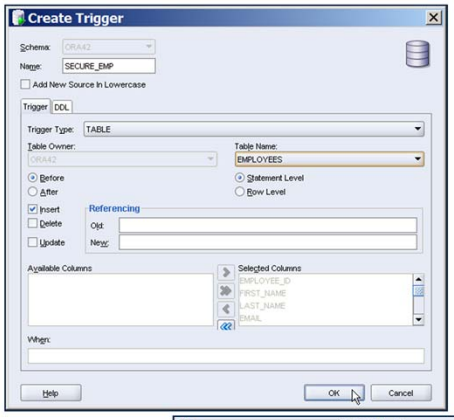
The *trigger_body* is the action performed by the trigger, implemented as either of the following:

- 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;
```

4.1: Triggers

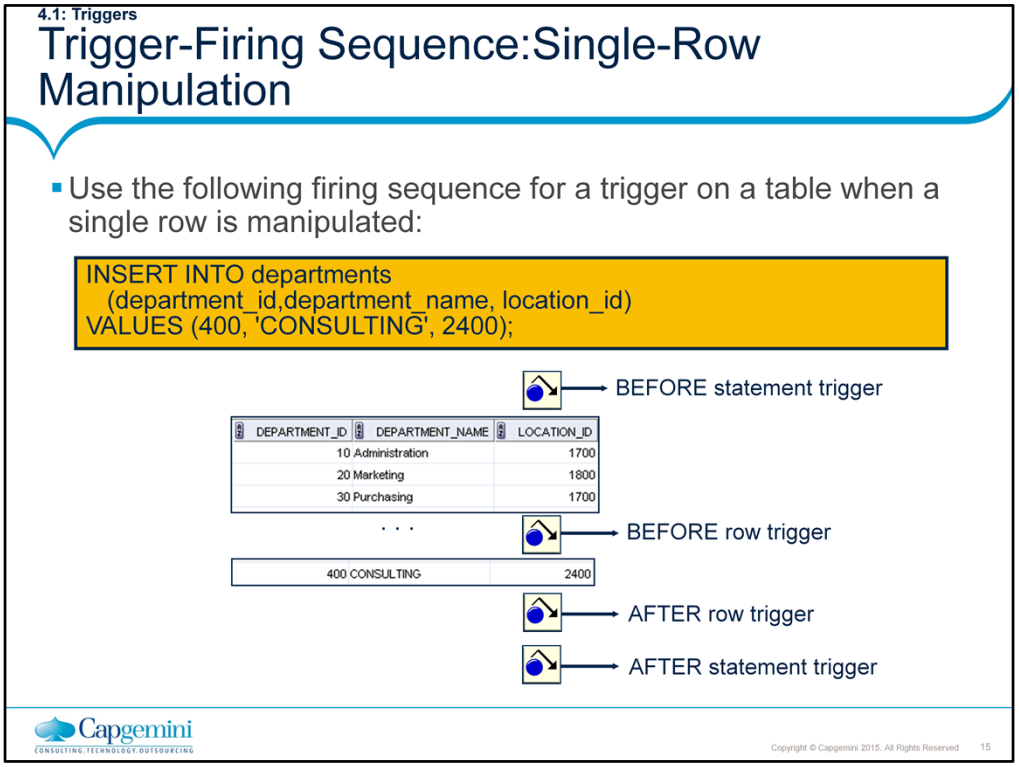
Creating DML Triggers Using SQL Developer




```
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;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14



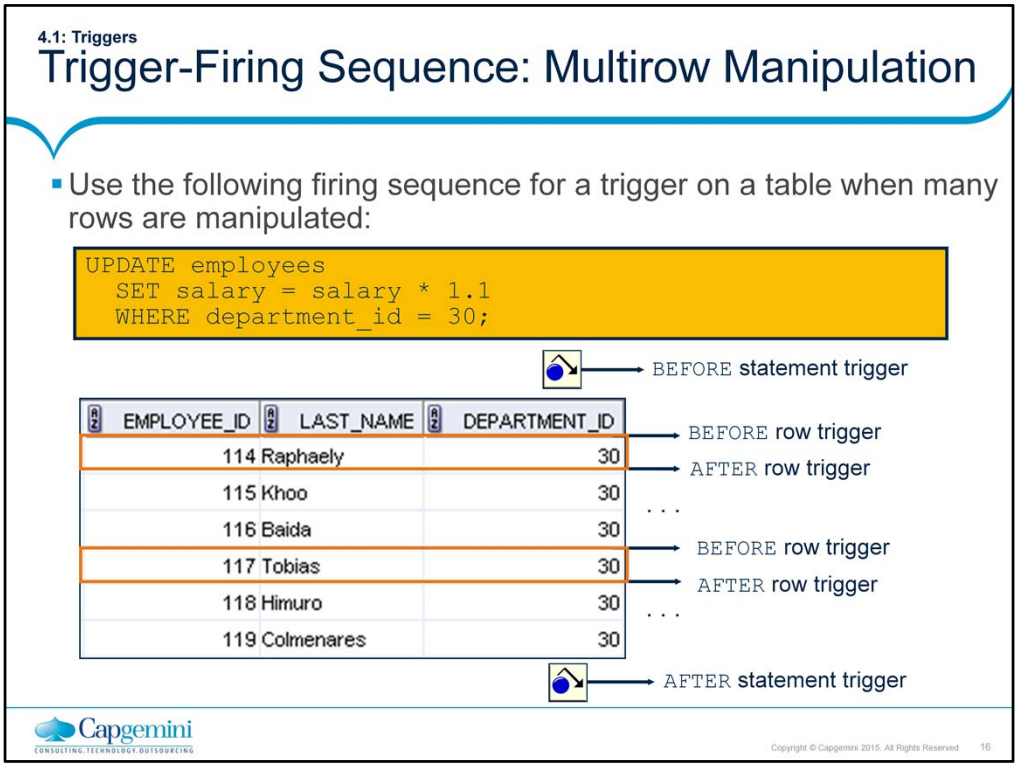
Trigger-Firing Sequence: Single-Row Manipulation

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.

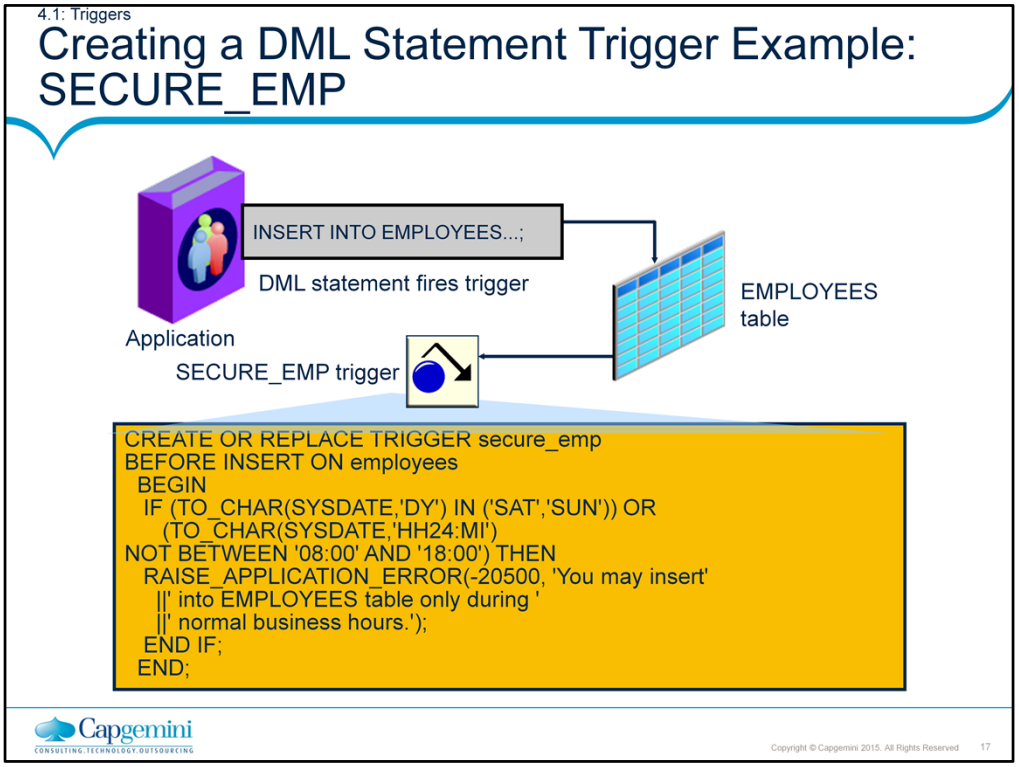


Trigger-Firing Sequence: Multirow Manipulation

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).

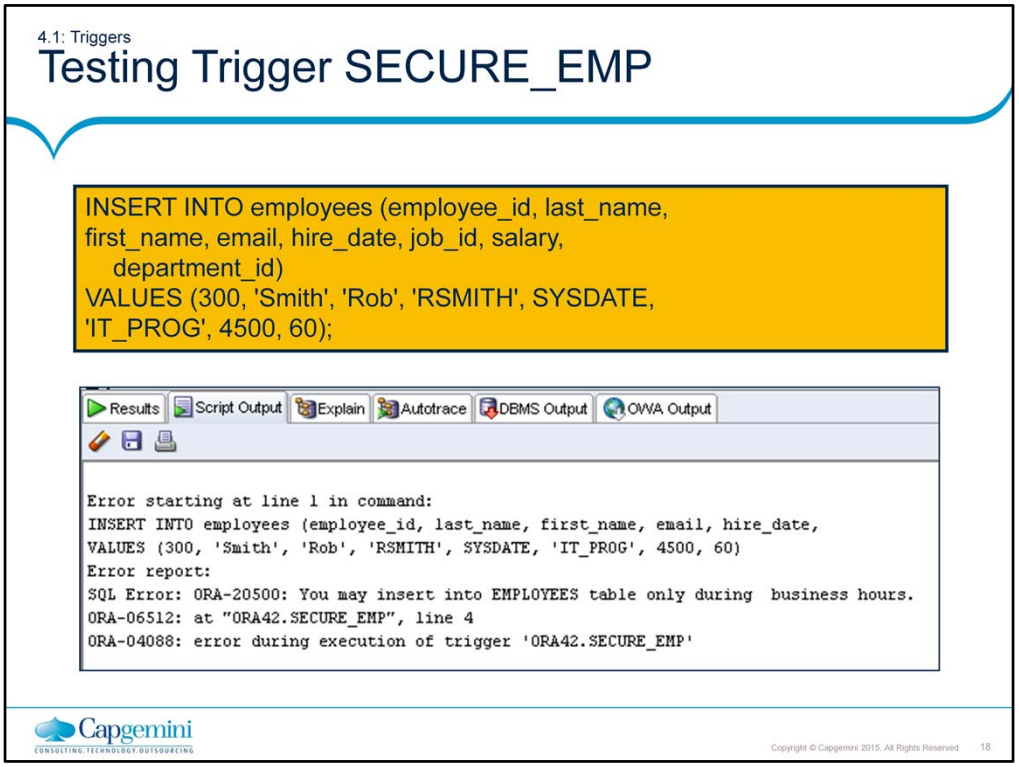


Creating a DML Statement Trigger

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.

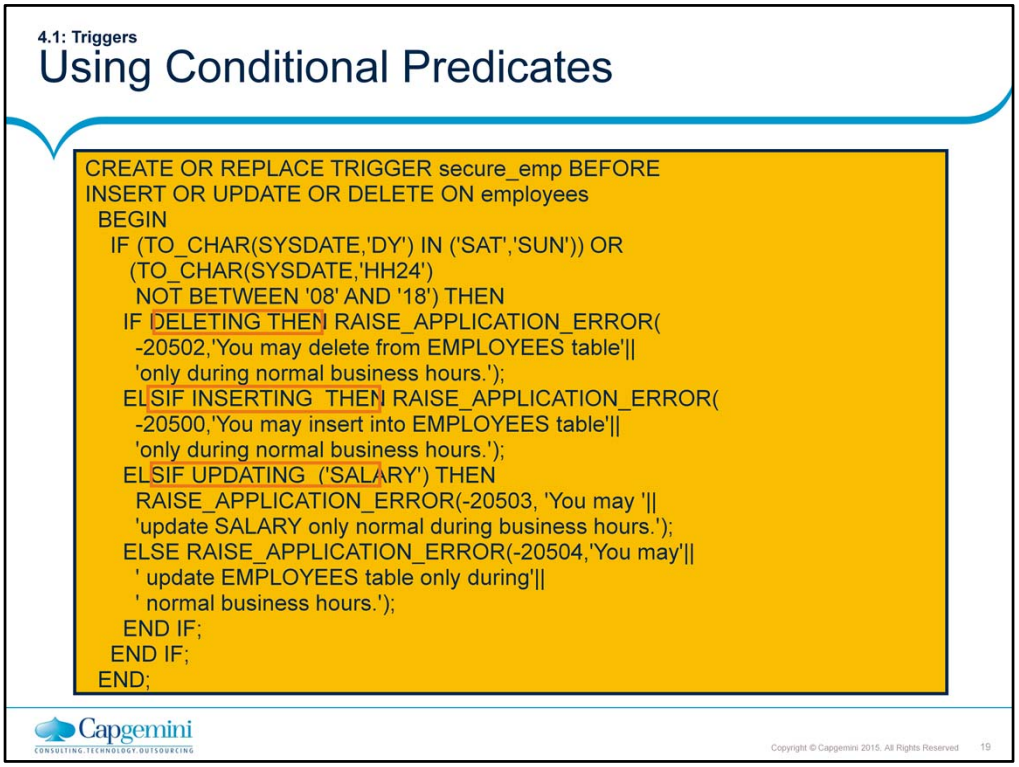
If a user attempts to insert a row into the EMPLOYEES table on Saturday, then the user sees an error message, the trigger fails, and the triggering statement is rolled back. Remember that the RAISE_APPLICATION_ERROR is a server-side built-in procedure that returns an error to the user and causes the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



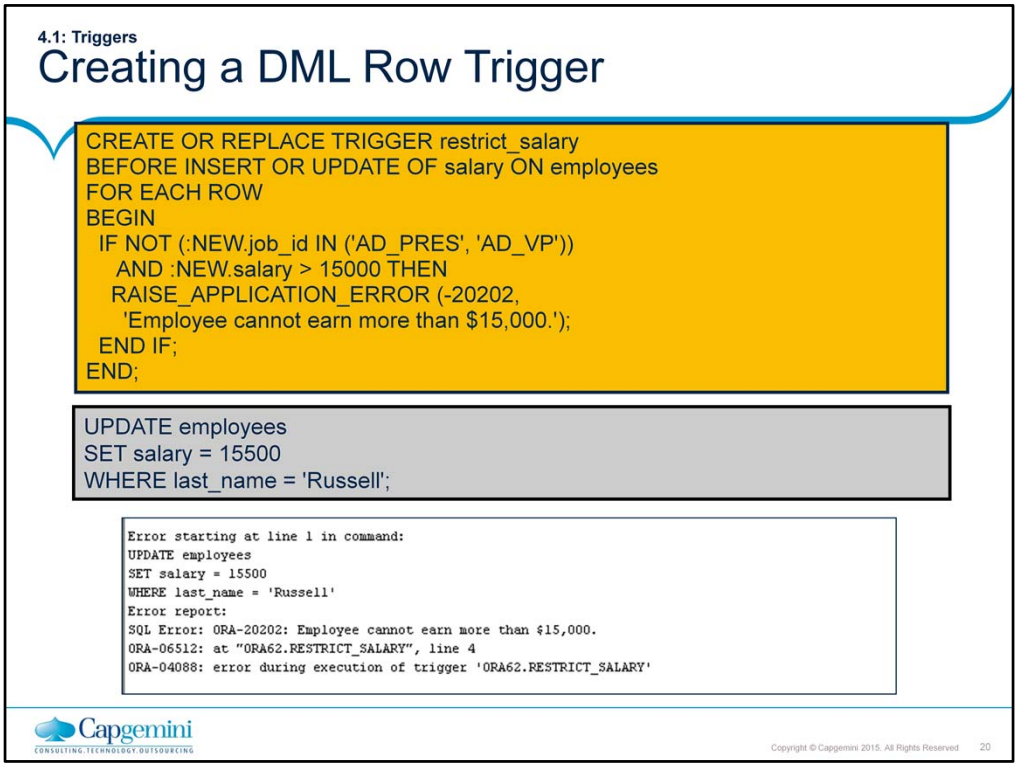
Detecting the DML Operation That Fired a Trigger

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

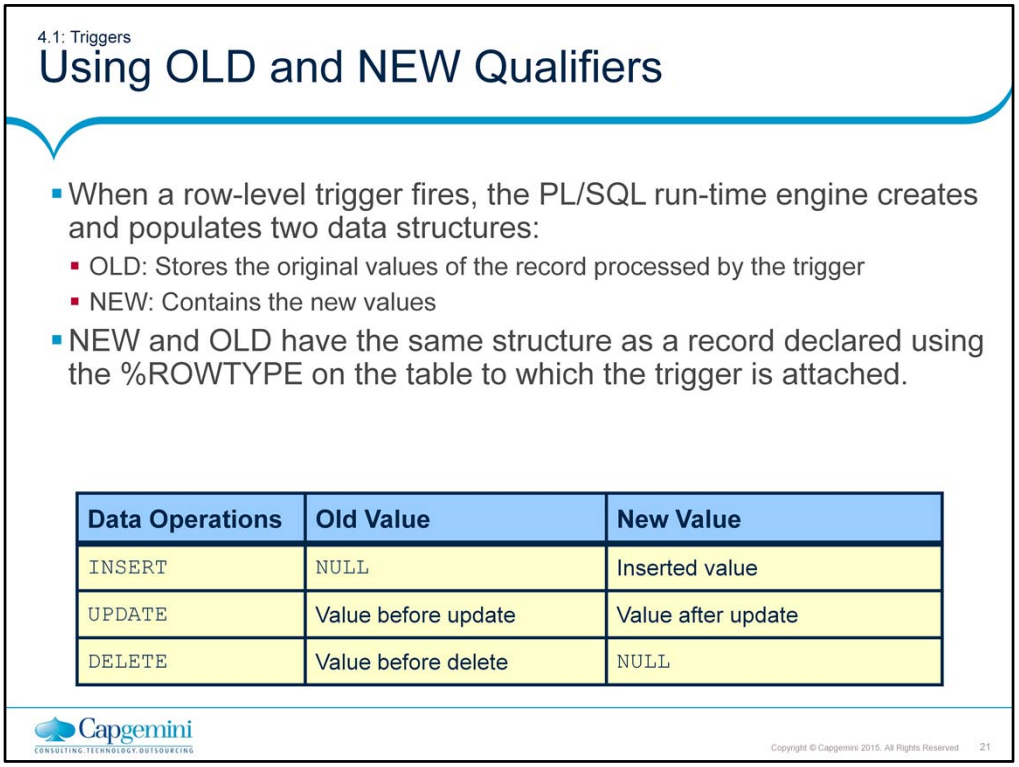


Creating a DML Row Trigger

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

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.

4.1: Triggers

New and Old Values

- New and old values of the DML statements can be processed with :NEW.column_name and :OLD.column_name in the trigger restriction and trigger action .

- Insert will have values in New variable
- Update will have values in New and Old variables
- Delete will have values in Old variable

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

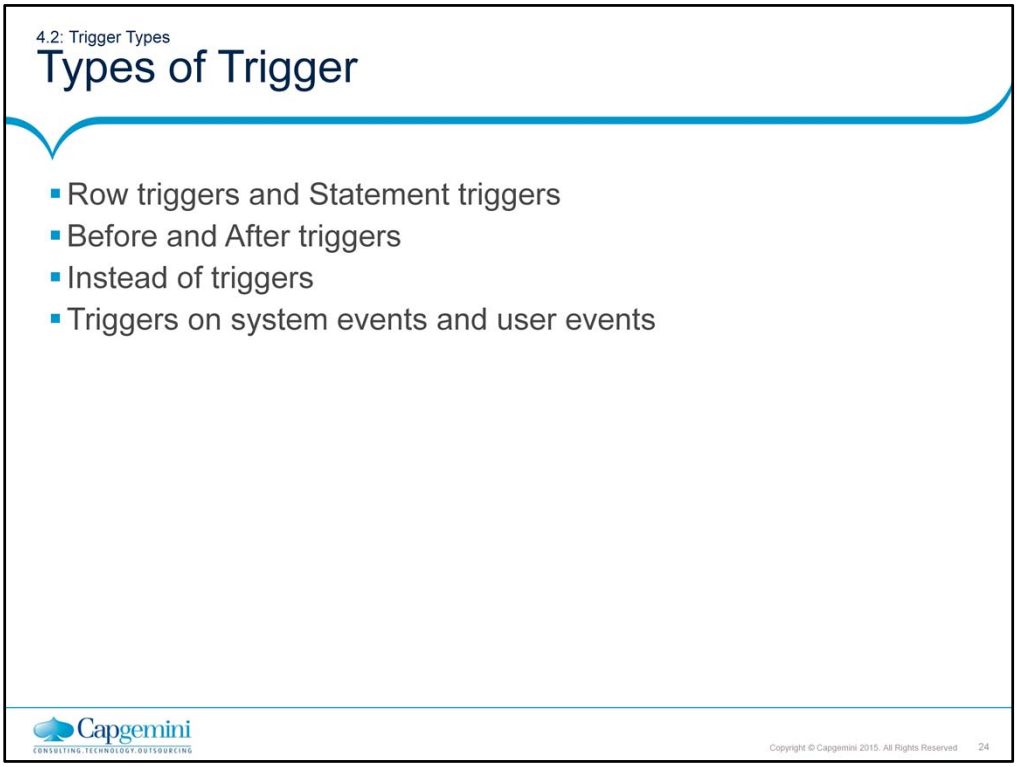
4.1: Triggers

Database Triggers - DML

- Create or replace trigger <trigger_name>
- after/before
- insert/update of <column_list>/delete on <table_name/view_name>
- for each row
- When (<condition>)
- <pl_sql >

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23


4.2: Trigger Types
Types of Trigger
Row triggers and Statement triggers
Before and After triggers
Instead of triggers
Triggers on system events and user events
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
24

4.2: Trigger Types

Row and Statement Triggers

- Row triggers fire once for every row affected by the triggering statement
- Statement triggers fire once on behalf of the triggering event

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 25

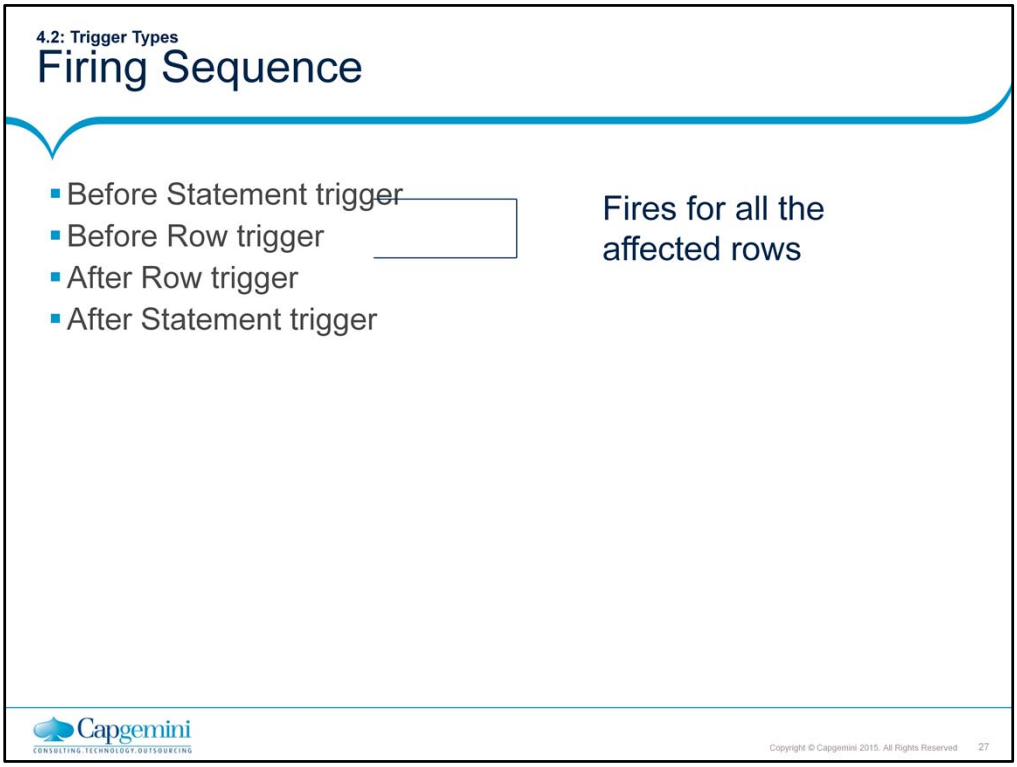
4.2: Trigger Types

Before and After triggers

- Specify the trigger timing
- Are fired by DML statements either before or after the execution of the DML statements
- Apply to row and statement triggers
- Cannot be specified on views
- Trigger type combinations :
- Before statement trigger
- Before row trigger
- After row trigger
- After statement trigger
- You can have multiple triggers of the same type for the same statement for any given table
-

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 26


4.2: Trigger Types
Firing Sequence
Before Statement trigger
Before Row trigger
After Row trigger
After Statement trigger
Fires for all the affected rows
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
27

4.2: Trigger Types

Trigger Firing Order

- Starting from 11g Oracle allows you to specify trigger firing order if more than one trigger is created .
- It is done using FOLLOWS keyword followed by trigger name after which current trigger is to be invoked.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 28

Contd.,

- create or replace trigger test_trg2 before insert on test
for each row
follows test_trg1
begin
insert into testlog values (‘From test_trig2′);
end;

4.2: Trigger Types

Instead of triggers

- Complex views which cannot be modified by DML statements can be modified by using Instead of trigger
- It provides a transparent way of modifying the base tables through the views
- Trigger is fired instead of executing the triggering statement

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 30

4.2: Trigger Types

Triggers on System Events and User Events

- Certain system events like database startup and shutdown and server error messages can be traced through triggers
- User events like user logon and logoff, DDL and DML can also be traced through triggers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31

4.2: Trigger Types

Example of DML trigger

```
create or replace trigger emp_del
before delete on emp
for each row
begin
        insert into emp_history values
        (:old.empno,:old.ename,:old.job,:old.mgr,:old.hiredate,
        :old.sal,:old.comm,:old.deptno);

end ;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 32

4.2: Trigger Types

Example

```
create or replace trigger dept_tot_emp
after insert on emp
for each row
begin

          update dept set tot_emp = tot_emp + 1
         where deptno = :new.deptno;
end;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33

4.2: Trigger Types

Example of DML trigger

```
create or replace trigger  dept_tot_emp
after insert or delete or  update of deptno  on emp
for each row
begin

if inserting or updating then
        update dept set tot_emp = tot_emp + 1
        where deptno = :new.deptno;

end if;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 34

4.2: Trigger Types

Example of DML trigger

```
if updating  or deleting then

        update dept set tot_emp = tot_emp - 1
        where deptno = :old.deptno;
end if;

end;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 35

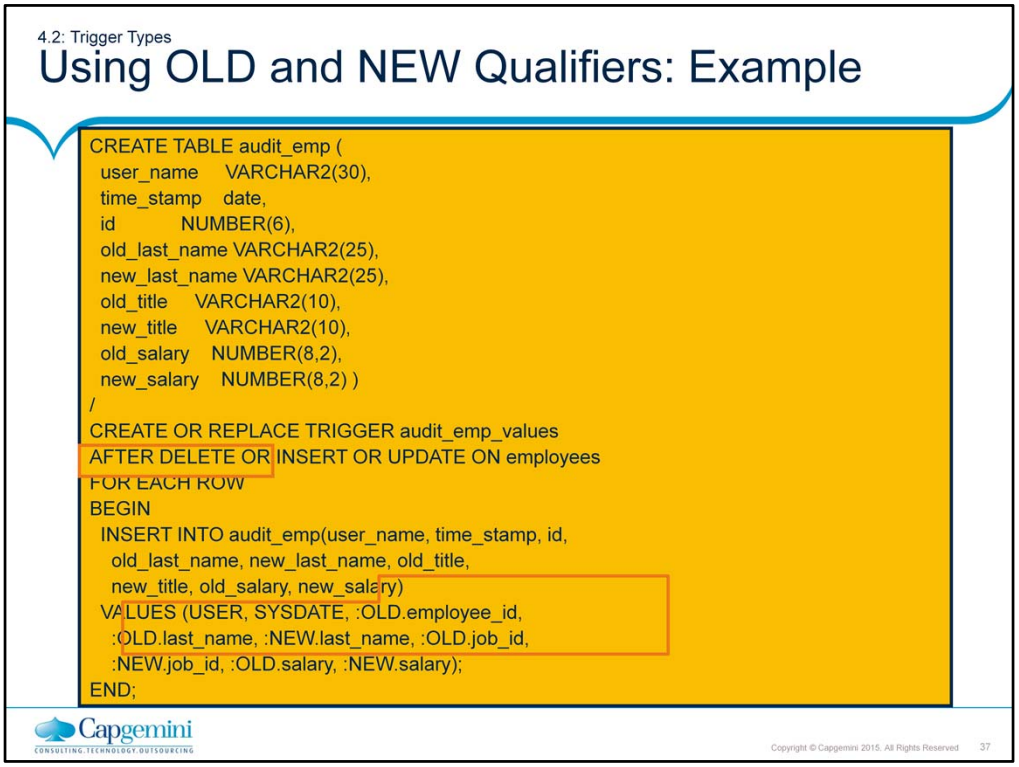
4.2: Trigger Types

Example of DML trigger

```
create or replace trigger  dept_tot_emp
after insert or delete or  update of deptno  on emp
for each row
when(old.deptno <> new.deptno)
begin
if inserting or updating then
              update dept set tot_emp = tot_emp + 1
             where deptno = :new.deptno;
end if;
if updating  or deleting then
           update dept set tot_emp = tot_emp - 1
           where deptno = :old.deptno;
end if;
end;
```

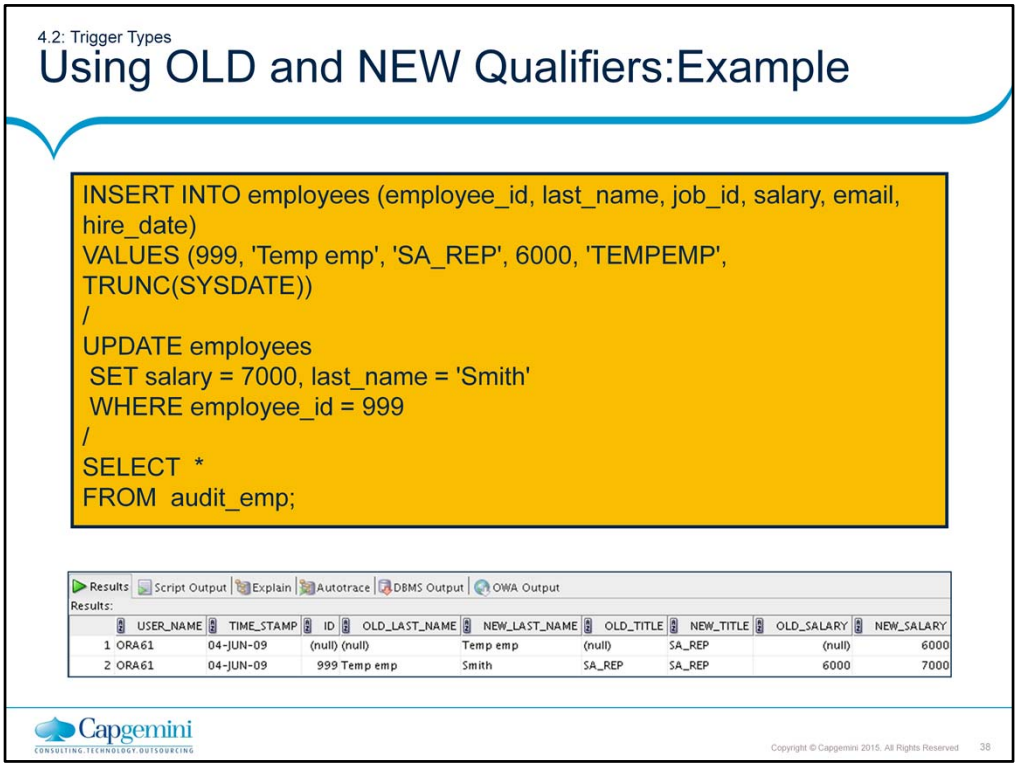
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 36



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 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:

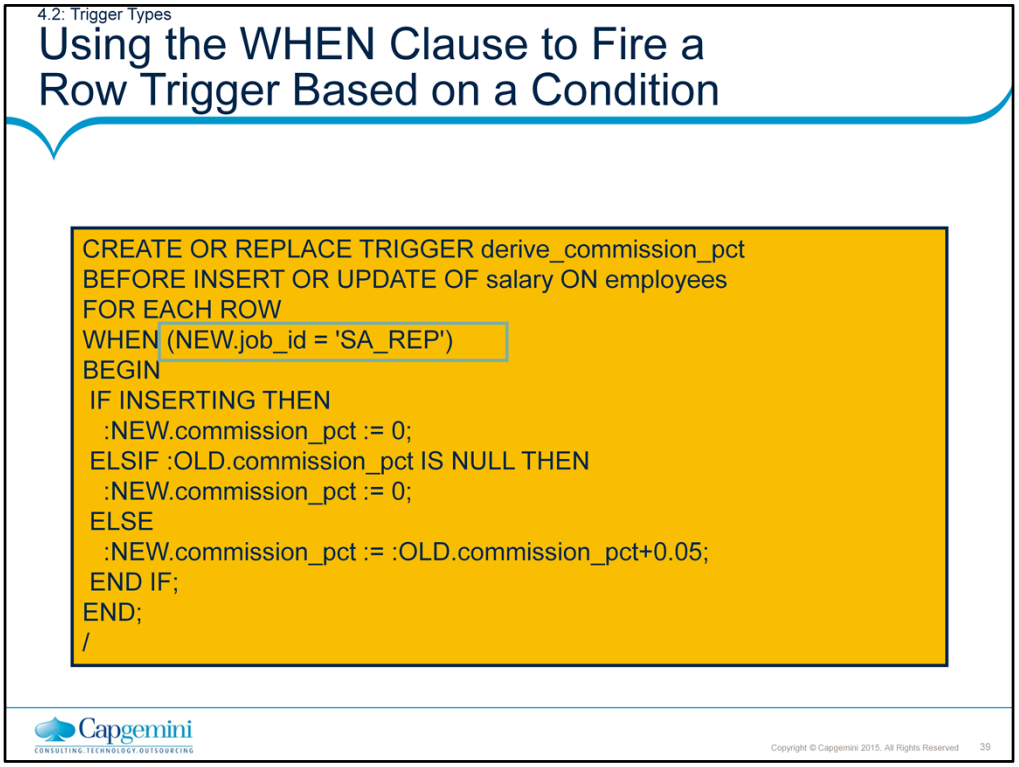
. . .

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_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

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

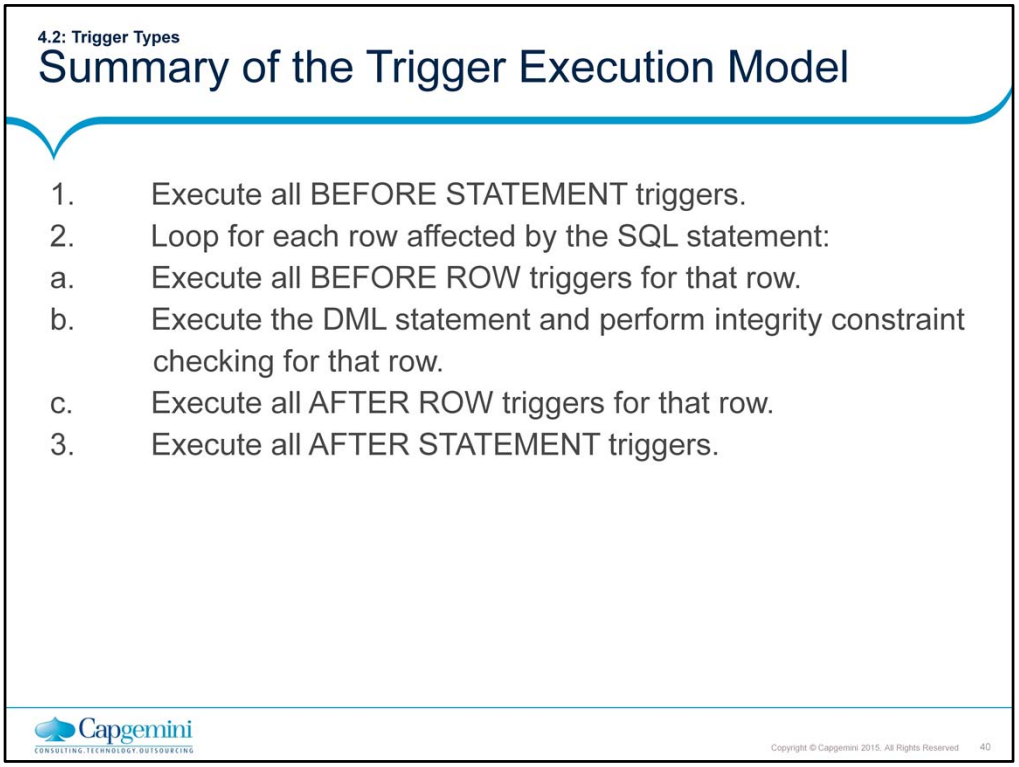
Optionally, you can include a trigger restriction in the definition of a row trigger by specifying a Boolean SQL expression in a `WHEN` clause. If you include a `WHEN` clause in the trigger, then the expression in the `WHEN` clause is evaluated for each row that the trigger affects.

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 `NOT TRUE` for a row (unknown, as with nulls), then the trigger body does not execute for that row. The evaluation of the `WHEN` clause 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

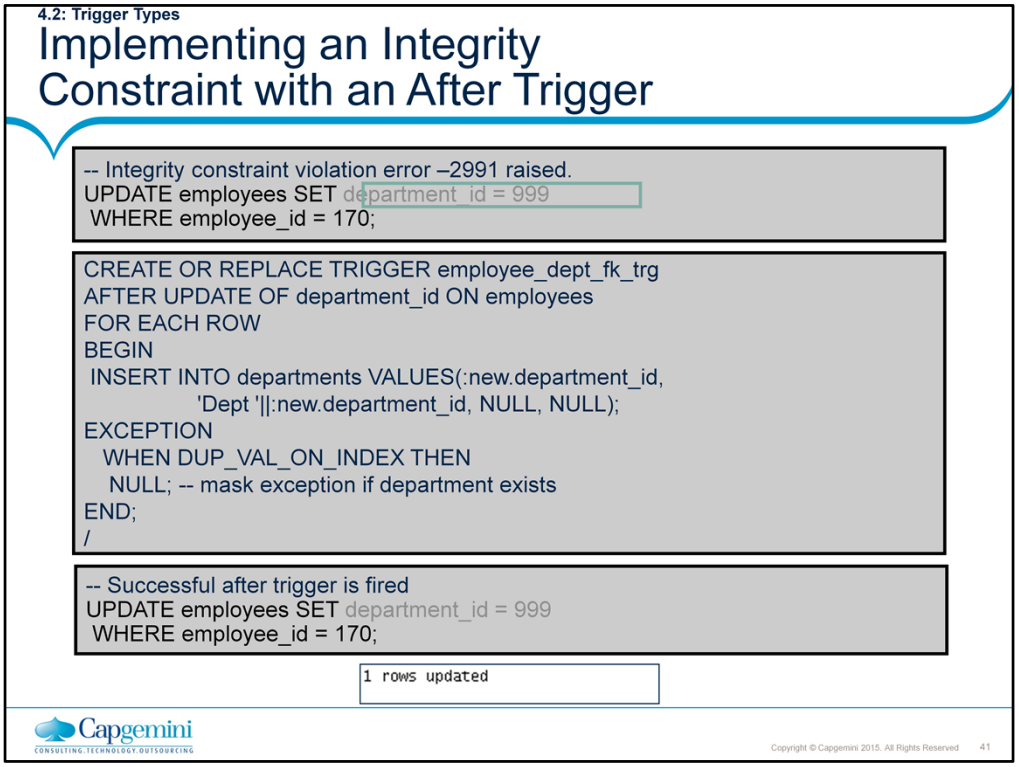
A triggering event or a statement within the trigger can cause one or more integrity constraints to be checked. However, you can defer constraint checking until a COMMIT operation is performed.

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

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 action.

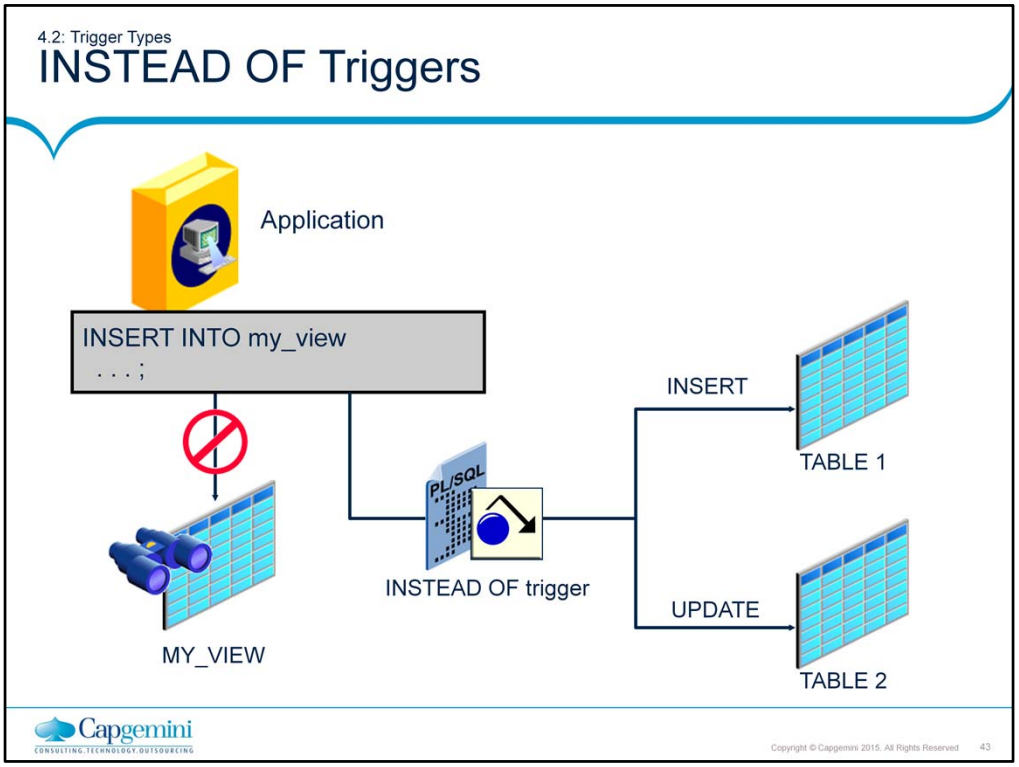
4.2: Trigger Types

Database trigger – Instead off

- Create or replace trigger <trigger_name>
- Instead of insert on<table_name/view_name>
- for each row
- <pl_sql

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 42



INSTEAD OF Triggers

Use INSTEAD OF triggers to modify data in which the DML statement has been issued against an inherently un-updatable view. These triggers are called INSTEAD OF 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 INSTEAD OF 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 GROUP BY, CONNECT BY, 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 INSTEAD OF 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 INSTEAD OF triggers, then the triggers take precedence. INSTEAD OF triggers are row triggers. The CHECK option for views is not enforced when insertions or updates to the view are performed by using INSTEAD OF triggers. The INSTEAD OF trigger body must enforce the check.

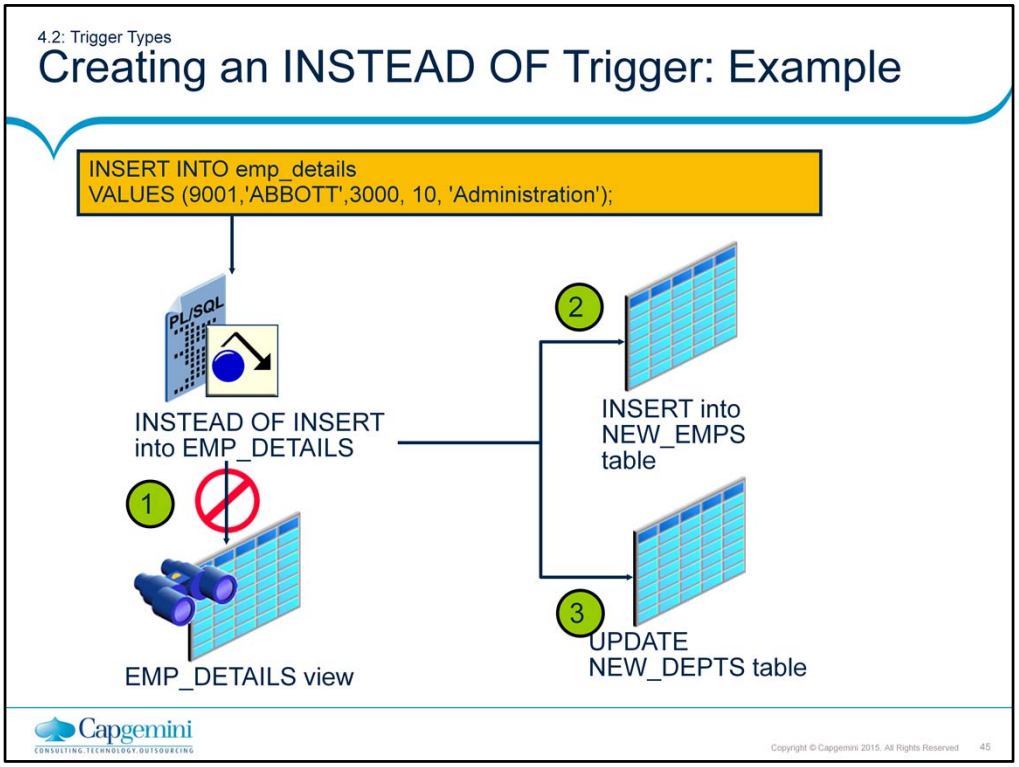
4.2: Trigger Types

Instead of Trigger

```
create or replace trigger emp_details_insert
Instead of insert on emp_details
for each row
begin

insert into emp(EMPNO,ENAME,JOB,MGR,HIREDATE,DEPTNO)
values(:new.empno,:new.ename,:new.job,:new.mgr,:new.hiredate,:new.deptn
o);

insert into emp_addr(EMPNO,ADDRESS,CONTACT)
values (:new.empno,:new.address,:new.contact);

end;
/
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 44



Creating an INSTEAD OF Trigger

You can create an INSTEAD OF 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 INSTEAD OF 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_DEPT INSTEAD 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.

4.2: Trigger Types

Example of Instead of Trigger

- EMP table

- EMPNO
- ENAME
- JOB
- MGR
- HIREDATE
- SAL
- COMM
- DEPTNO

EMP_ADDR table

EMPNO
ADDRESS
CONTACT

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

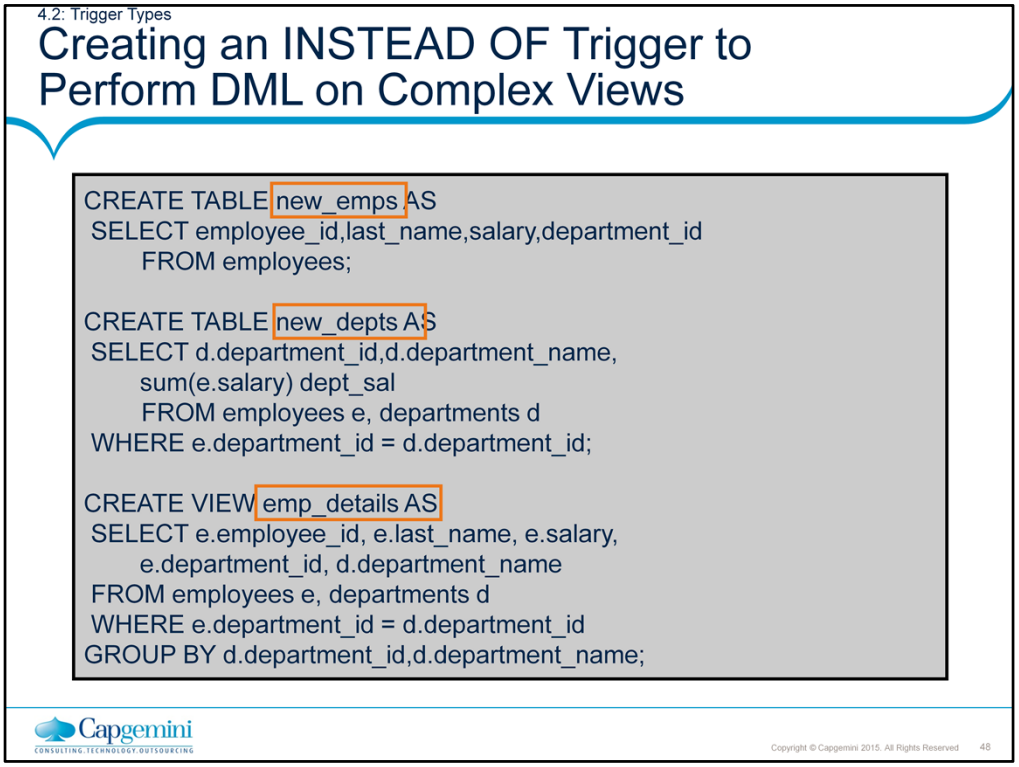
 46

4.2: Trigger Types

Defining Complex View

- create view emp_details
- as
- select e.empno,ename,address,contact,job,mgr,hiredate,deptno from emp e,emp_addr a
- where e.empno = a.empno

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Creating an INSTEAD OF 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 INSTEAD OF 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: INSTEAD OF triggers can be written only for views, and the BEFORE and AFTER timing options are not valid.a

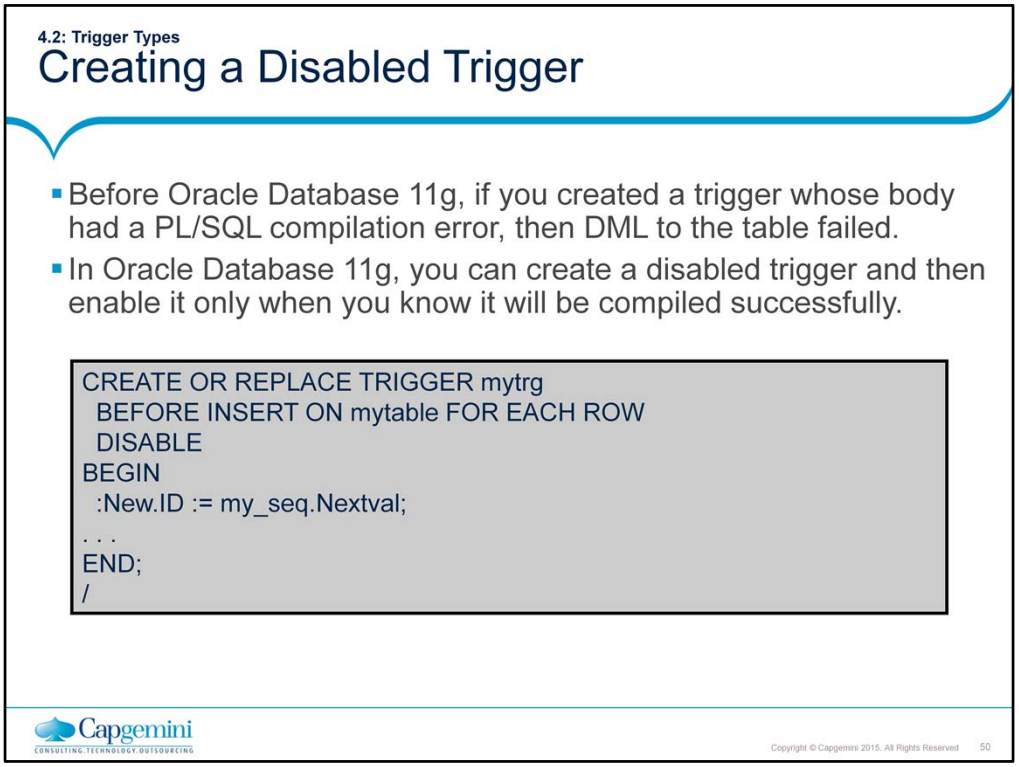
4.2: Trigger Types

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.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 49



Creating a Disabled Trigger

Before Oracle Database 11*g*, 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:

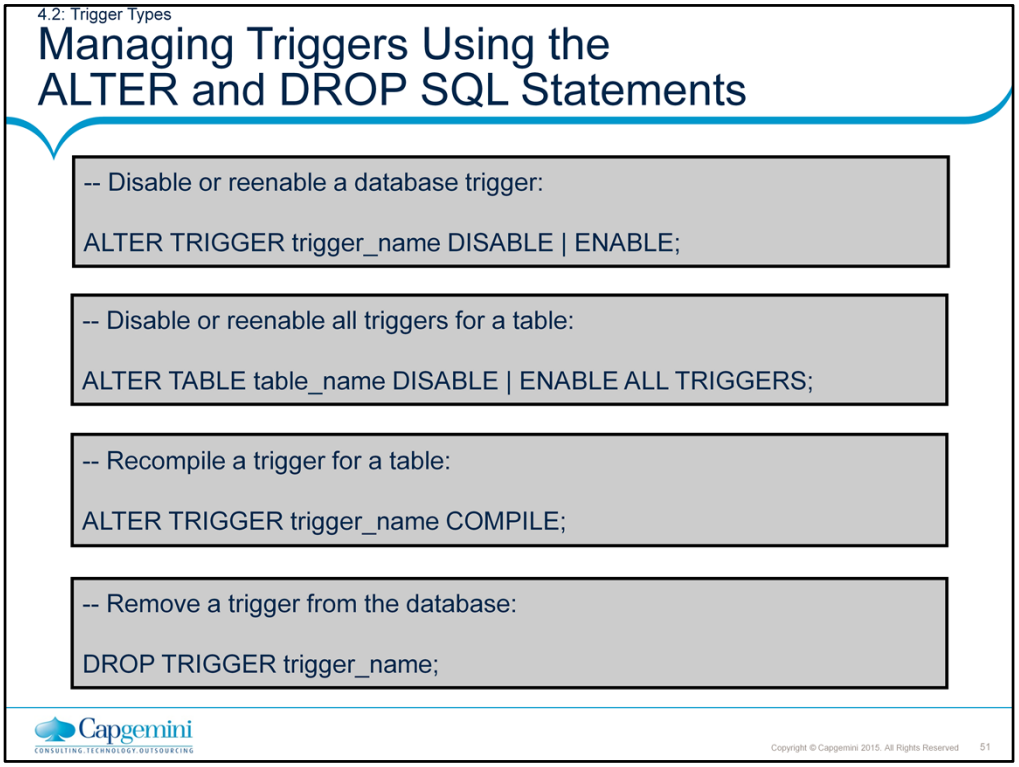
```
ORA-04098: trigger 'TRG' is invalid and failed re-validation
```

In Oracle Database 11*g*, 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 read-consistent views for queries and constraints, manages the dependencies, and provides a two-phase commit process if a trigger updates remote tables in a distributed database.

Disabling a Trigger

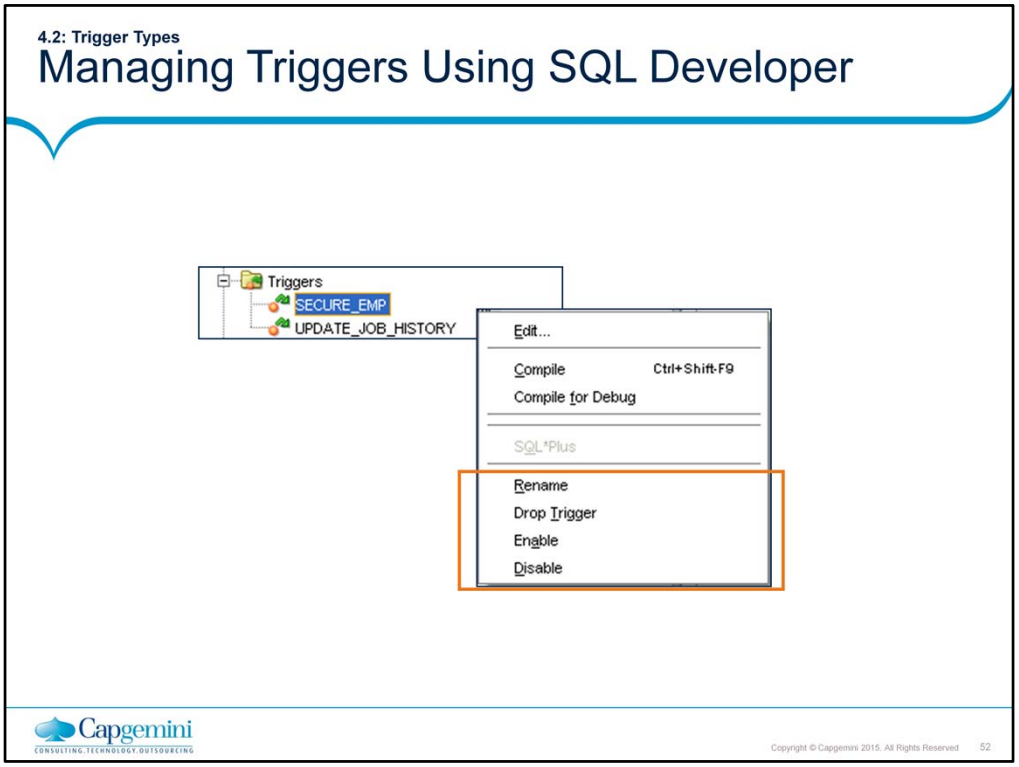
Use the `ALTER TRIGGER` command to disable a trigger. You can also disable all triggers on a table by using the `ALTER TABLE` command. You can disable triggers to improve performance or to avoid data integrity checks when loading massive amounts of data with utilities such as SQL*Loader. You might also 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 `ALTER TRIGGER` command to explicitly recompile a trigger that is invalid.

Removing Triggers

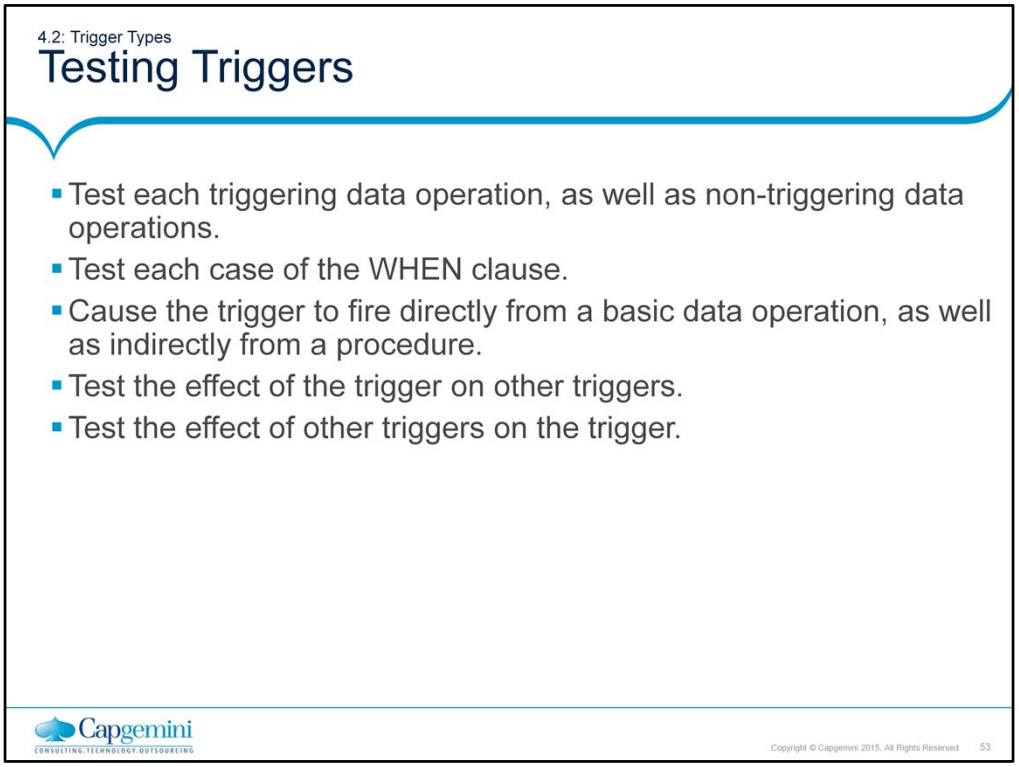
When a trigger is no longer required, use a SQL statement in SQL Developer or SQL*Plus to remove it. When you remove a table, all triggers on that table are also removed.



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:

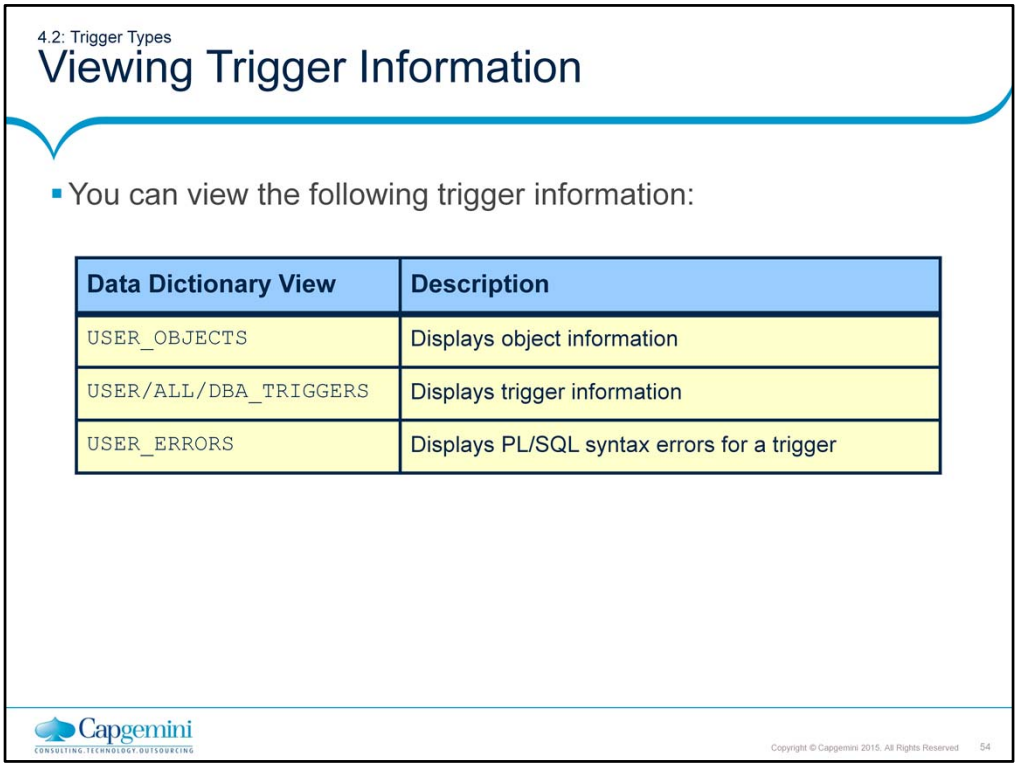
Ensure that the trigger works properly by testing a number of cases separately:

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

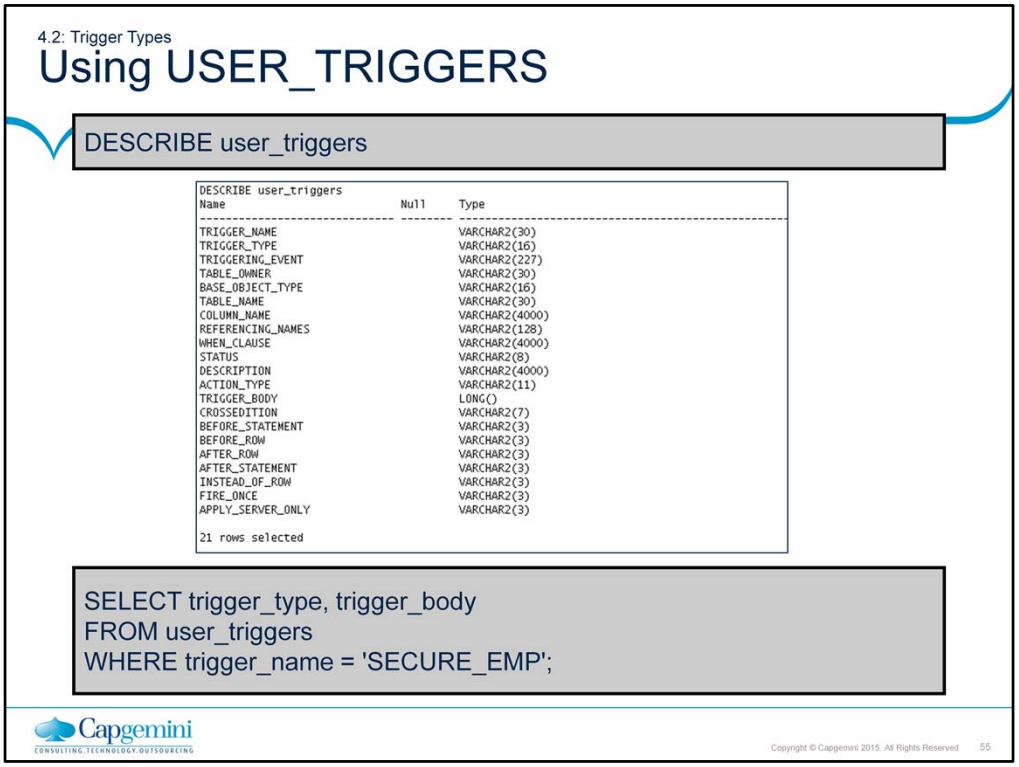
The slide shows the data dictionary views that you can access to get information regarding the triggers.

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 `SELECT Username FROM 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;
```

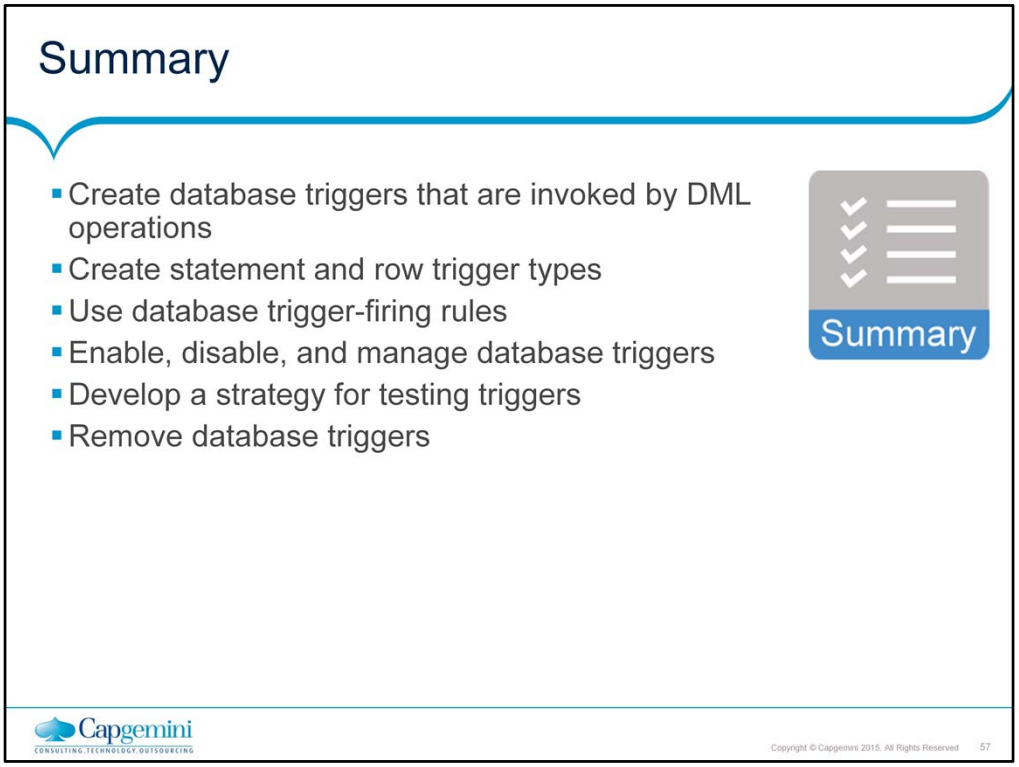
4.2: Trigger Types

Managing Triggers

- Disable or re-enable a database trigger Alter trigger trigger_name disable | enable
- Disable or re-enable all triggers for a table Alter table table name disable | enable all triggers
- Recompile a trigger Alter trigger name compile

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 56



Add the notes here.

Review Question

- Question 1: Triggers should not issue Transaction Control Statements (TCL)
  - True / False
- Question 2: BEFORE DROP and AFTER DROP triggers are fired when a schema object is dropped
  - True / False
- Question 3: The :new and :old records must be used in WHEN clause with a colon
  - True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 58

Add the notes here.

Review Question

- Question 4: A ___ is a table that is currently being modified by a DML statement
- Question 5: A ___ is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 59

Add the notes here.