```
--------------------------------------------------------------------------------
----------
------------------------------SPECIFYING THE TIMING OF
TRIGGERS------------------------------
--------------------------------------------------------------------------------
----------
----------------- The create code of the first trigger
create or replace trigger first_trigger
before insert or update on employees_copy
begin
  dbms_output.put_line('An insert or update occurred in employees_copy table!.');
end;
----------------- sql commands to or not to run the trigger
update employees_copy set salary = salary + 100;
delete from employees_copy;
--------------------------------------------------------------------------------
----------
--------------------------------STATEMENT & ROW LEVEL
TRIGGERS--------------------------------
--------------------------------------------------------------------------------
----------
----------------- before statement level trigger example
create or replace trigger before_statement_emp_cpy
before insert or update on employees_copy
begin
  dbms_output.put_line('Before Statement Trigger is Fired!.');
end;
----------------- after statement level trigger example
create or replace trigger after_statement_emp_cpy
after insert or update on employees_copy
begin
  dbms_output.put_line('After Statement Trigger is Fired!.');
end;
----------------- before row level trigger example
create or replace trigger before_row_emp_cpy
before insert or update on employees_copy
for each row
begin
  dbms_output.put_line('Before Row Trigger is Fired!.');
end;
----------------- after row level trigger example
create or replace trigger after_row_emp_cpy
after insert or update on employees_copy
for each row
begin
  dbms_output.put_line('After Row Trigger is Fired!.');
end;
----------------- sql queries used in this lecture
update employees_copy set salary = salary + 100 where employee_id = 100;
update employees_copy set salary = salary + 100 where employee_id = 99;
update employees_copy set salary = salary + 100
where department_id = 30;
--------------------------------------------------------------------------------
----------
-------------------------------:NEW & :OLD QUALIFIERS IN
TRIGGERS----------------------------
--------------------------------------------------------------------------------
----------
create or replace trigger before_row_emp_cpy
```

```
before insert or update or delete on employees_copy
referencing old as O new as N
for each row
begin
  dbms_output.put_line('Before Row Trigger is Fired!.');
  dbms_output.put_line('The Salary of Employee '||:o.employee_id
    ||' -> Before:'|| :o.salary||' After:'||:n.salary);
--------------------------------------------------------------------------------
----------
--------------------------------USING CONDITIONAL PREDICATES
--------------------------------
--------------------------------------------------------------------------------
----------
create or replace trigger before_row_emp_cpy
before insert or update or delete on employees_copy
referencing old as O new as N
for each row
begin
  dbms_output.put_line('Before Row Trigger is Fired!.');
  dbms_output.put_line('The Salary of Employee '||:o.employee_id
    ||' -> Before:'|| :o.salary||' After:'||:n.salary);
  if inserting then
    dbms_output.put_line('An INSERT occurred on employees_copy table');
  elsif deleting then
    dbms_output.put_line('A DELETE occurred on employees_copy table');
  elsif updating ('salary') then
    dbms_output.put_line('A DELETE occurred on the salary column');
  elsif updating then
    dbms_output.put_line('An UPDATE occurred on employees_copy table');
  end if;
end;
--------------------------------------------------------------------------------
----------
------------------------USING RAISE_APPLICATION_ERROR PROCEDURE WITH
TRIGGERS----------------
--------------------------------------------------------------------------------
----------
create or replace trigger before_row_emp_cpy
before insert or update or delete on employees_copy
referencing old as O new as N
for each row
begin
  dbms_output.put_line('Before Row Trigger is Fired!.');
  dbms_output.put_line('The Salary of Employee '||:o.employee_id
    ||' -> Before:'|| :o.salary||' After:'||:n.salary);
  if inserting then
    if :n.hire_date > sysdate then
      raise_application_error(-20000,'You cannot enter a future hire..');
    end if;
  elsif deleting then
    raise_application_error(-20001,'You cannot delete from the employees_copy
table..');
  elsif updating ('salary') then
    if :n.salary > 50000 then
      raise_application_error(-20002,'A salary cannot be higher than 50000..');
    end if;
  elsif updating then
    dbms_output.put_line('An UPDATE occurred on employees_copy table');
  end if;
```

```
end;
--------------------------------------------------------------------------------
----------
-------------------------------USING UPDATE OF EVENT IN
TRIGGERS----------------------------
--------------------------------------------------------------------------------
----------
create or replace trigger prevent_updates_of_constant_columns
before update of hire_date,salary on employees_copy
for each row
begin
  raise_application_error(-20005,'You cannot modify the hire_date and salary
columns');
end;
--------------------------------------------------------------------------------
----------
---------------------------------USING WHEN CLAUSE ON
TRIGGERS-----------------------------
--------------------------------------------------------------------------------
----------
create or replace trigger prevent_high_salary
before insert or update of salary on employees_copy
for each row
when (new.salary > 50000)
begin
  raise_application_error(-20006,'A salary cannot be higher than 50000!.');
end;
--------------------------------------------------------------------------------
----------
-----------------------------------USING INSTEAD OF
TRIGGERS--------------------------------
--------------------------------------------------------------------------------
----------
----------------- creating a complex view -----------------
CREATE OR REPLACE VIEW VW_EMP_DETAILS AS
  SELECT UPPER(DEPARTMENT_NAME) DNAME, MIN(SALARY) MIN_SAL, MAX(SALARY) MAX_SAL
    FROM EMPLOYEES_COPY JOIN DEPARTMENTS_COPY
    USING (DEPARTMENT_ID)
    GROUP BY DEPARTMENT_NAME;
----------------- updating the complex view -----------------
UPDATE VW_EMP_DETAILS SET DNAME = 'EXEC DEPT' WHERE
  UPPER(DNAME) = 'EXECUTIVE';
----------------- Instead of trigger -----------------
CREATE OR REPLACE TRIGGER EMP_DETAILS_VW_DML
  INSTEAD OF INSERT OR UPDATE OR DELETE ON VW_EMP_DETAILS
  FOR EACH ROW
  DECLARE
    V_DEPT_ID PLS_INTEGER;
  BEGIN

  IF INSERTING THEN
    SELECT MAX(DEPARTMENT_ID) + 10 INTO V_DEPT_ID FROM DEPARTMENTS_COPY;
    INSERT INTO DEPARTMENTS_COPY VALUES (V_DEPT_ID, :NEW.DNAME,NULL,NULL);
  ELSIF DELETING THEN
    DELETE FROM DEPARTMENTS_COPY WHERE UPPER(DEPARTMENT_NAME) = UPPER(:OLD.DNAME);
  ELSIF UPDATING('DNAME') THEN
    UPDATE DEPARTMENTS_COPY SET DEPARTMENT_NAME = :NEW.DNAME
      WHERE UPPER(DEPARTMENT_NAME) = UPPER(:OLD.DNAME);
  ELSE
```

```
    RAISE_APPLICATION_ERROR(-20007,'You cannot update any data other than
department name!.');
  END IF;
END;
-------------------------------------------------------------------------------------
----------
----------------------------------CREATING DISABLED
TRIGGERS--------------------------------
-------------------------------------------------------------------------------------
----------
create or replace trigger prevent_high_salary
before insert or update of salary on employees_copy
for each row
disable
when (new.salary > 50000)
begin
  raise_application_error(-20006,'A salary cannot be higher than 50000!.');
end;
-------------------------------------------------------------------------------------
----------
-----------------------------REAL-WORLD EXAMPLES ON DML
TRIGGERS-----------------------------
-------------------------------------------------------------------------------------
----------
create sequence seq_dep_cpy
  start with 280
  increment by 10;
----------------- primary key example
create or replace trigger trg_before_insert_dept_cpy
before insert on departments_copy
for each row
begin
  --select seq_dep_cpy.nextval into :new.department_id from dual;
  :new.department_id := seq_dep_cpy.nextval;
end;
-----------------
insert into departments_copy
        (department_name,manager_id,location_id)
        values
        ('Security',200,1700);
-----------------
desc departments_copy;
----------------- creating the audit log table
create table log_departments_copy
        (log_user varchar2(30), log_date date, dml_type varchar2(10),
         old_department_id number(4), new_department_id number(4),
         old_department_name varchar2(30), new_department_name varchar2(30),
         old_manager_id number(6), new_manager_id number(6),
         old_location_id number(4), new_location_id number(4));
----------------- audit log trigger
create or replace trigger trg_department_copy_log
after insert or update or delete on departments_copy
for each row
declare v_dml_type varchar2(10);
begin
  if inserting then
    v_dml_type := 'INSERT';
  elsif updating then
    v_dml_type := 'UPDATE';
```

```
  elsif deleting then
    v_dml_type := 'DELETE';
  end if;
  insert into log_departments_copy values
    (user, sysdate, v_dml_type,
     :old.department_id, :new.department_id,
     :old.department_name, :new.department_name,
     :old.manager_id, :new.manager_id,
     :old.location_id, :new.location_id);
end;
----------------- other sql codes used in this lecture
insert into departments_copy (department_name, manager_id,location_id)
    values ('Cyber Security', 100, 1700);

select * from LOG_DEPARTMENTS_COPY;
update departments_copy set manager_id = 200 where DEPARTMENT_NAME = 'Cyber
Security';
delete from departments_copy where DEPARTMENT_NAME = 'Cyber Security';
--------------------------------------------------------------------------------
----------
------------------------------------ COMPOUND TRIGGERS
------------------------------------
--------------------------------------------------------------------------------
----------
----------------- The first simple compound trigger
create or replace trigger trg_comp_emps
for insert or update or delete on employees_copy
compound trigger
v_dml_type varchar2(10);
  before statement is
   begin
    if inserting then
      v_dml_type := 'INSERT';
    elsif updating then
      v_dml_type := 'UPDATE';
    elsif deleting then
      v_dml_type := 'DELETE';
    end if;
    dbms_output.put_line('Before statement section is executed with the '||
v_dml_type ||' event!.');
  end before statement;
  before each row is
  t number;
    begin
      dbms_output.put_line('Before row section is executed with the '||v_dml_type
||' event!.');
  end before each row;
  after each row is
    begin
      dbms_output.put_line('After row section is executed with the '||v_dml_type
||' event!.');
  end after each row;
  after statement is
    begin
      dbms_output.put_line('After statement section is executed with the '||
v_dml_type ||' event!.');
  end after statement;
end;
-----------------
```

```
CREATE OR REPLACE TRIGGER TRG_COMP_EMPS
  FOR INSERT OR UPDATE OR DELETE ON EMPLOYEES_COPY
  COMPOUND TRIGGER
    TYPE T_AVG_DEPT_SALARIES IS TABLE OF EMPLOYEES_COPY.SALARY%TYPE INDEX BY
PLS_INTEGER;
    AVG_DEPT_SALARIES T_AVG_DEPT_SALARIES;

  BEFORE STATEMENT IS
    BEGIN
      FOR AVG_SAL IN (SELECT AVG(SALARY) SALARY , NVL(DEPARTMENT_ID,999)
DEPARTMENT_ID
                        FROM EMPLOYEES_COPY GROUP BY DEPARTMENT_ID) LOOP
        AVG_DEPT_SALARIES(AVG_SAL.DEPARTMENT_ID) := AVG_SAL.SALARY;
      END LOOP;
  END BEFORE STATEMENT;

  AFTER EACH ROW IS
    V_INTERVAL NUMBER := 15;
    BEGIN
       IF :NEW.SALARY > AVG_DEPT_SALARIES(:NEW.DEPARTMENT_ID) +
AVG_DEPT_SALARIES(:NEW.DEPARTMENT_ID)*V_INTERVAL/100 THEN
        RAISE_APPLICATION_ERROR(-20005,'A raise cannot be '|| V_INTERVAL|| '
percent higher than
                                   its department''s average!');
       END IF;
  END AFTER EACH ROW;

  AFTER STATEMENT IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE('All the changes are done successfully!');
  END AFTER STATEMENT;

END;
-------------------------------------------------------------------------------
----------
------------------------------------ MUTATING TABLE ERRORS
---------------------------------
-------------------------------------------------------------------------------
----------
----------------- A mutating table error example
create or replace trigger trg_mutating_emps
before insert or update on employees_copy
for each row
    declare
    v_interval number := 15;
    v_avg_salary number;
    begin
    select avg(salary) into v_avg_salary from employees_copy where department_id
= :new.department_id;
      if :new.salary > v_avg_salary*v_interval/100 then
        RAISE_APPLICATION_ERROR(-20005, 'A raise cannot be '|| v_interval|| '
percent higher than its department''s average');
      end if;
end;
----------------- Getting mutating table error within a compound trigger
create or replace trigger trg_comp_emps
for insert or update or delete on employees_copy
compound trigger
  type t_avg_dept_salaries is table of employees_copy.salary%type index by
```

```
pls_integer;
  avg_dept_salaries t_avg_dept_salaries;
  before statement is
   begin
    for avg_sal in (select avg(salary) salary,nvl(department_id,999) department_id
from employees_copy group by department_id) loop
      avg_dept_salaries(avg_sal.department_id) := avg_sal.salary;
    end loop;
  end before statement;

  after each row is
    v_interval number := 15;
    begin
    update employees_copy set commission_pct = commission_pct;
      if :new.salary > avg_dept_salaries(:new.department_id)*v_interval/100 then
        RAISE_APPLICATION_ERROR(-20005, 'A raise cannot be '|| v_interval|| '
percent higher than its department''s average');
      end if;
  end after each row;
  after statement is
    begin
      dbms_output.put_line('All the updates are done successfully!.');
  end after statement;
end;
----------------- An example of getting maximum level of recursive SQL levels
create or replace trigger trg_comp_emps
for insert or update or delete on employees_copy
compound trigger
  type t_avg_dept_salaries is table of employees_copy.salary%type index by
pls_integer;
  avg_dept_salaries t_avg_dept_salaries;
  before statement is
   begin
    update employees_copy set commission_pct = commission_pct where employee_id =
100;
    for avg_sal in (select avg(salary) salary,nvl(department_id,999) department_id
from employees_copy group by department_id) loop
      avg_dept_salaries(avg_sal.department_id) := avg_sal.salary;
    end loop;
  end before statement;

  after each row is
    v_interval number := 15;
    begin
      if :new.salary > avg_dept_salaries(:new.department_id)*v_interval/100 then
        RAISE_APPLICATION_ERROR(-20005, 'A raise cannot be '|| v_interval|| '
percent higher than its department''s average');
      end if;
  end after each row;
  after statement is
    begin
      update employees_copy set commission_pct = commission_pct where employee_id =
100;
      dbms_output.put_line('All the updates are done successfully!.');
  end after statement;
end;
```