IS 276 Final Exam – DUE 12/12
Name __**Praminda Imaduwa-Gamage**_____

In this assignment, you create a package named VIDEO_PKG that contains procedures and functions for a video store application. This application enables customers to become a member of the video store. Any member can rent movies, return rented movies, and reserve movies. Additionally, you create a trigger to ensure that any data in the video tables is modified only during business hours.

Create the package by using SQL DEVELOPER and use DBMS_OUTPUT Oracle-supplied package to display messages.

The video store database contains the following tables: TITLE, TITLE_COPY, RENTAL, RESERVATION, and MEMBER. The entity relationship is attached to this assignment.

TASKS:

1. Load and execute the attached script `buildvid1.sql` script to create all the required tables and sequences that are needed for this exercise.
2. Load and execute the attached script `buildvid2.sql` script to populate all the tables created through the `buildvid1.sql script.`

3. Create a package named VIDEO_PKG with the following procedures and functions:

```
create or replace PACKAGE video_pkg
IS
PROCEDURE new_member(
p_lname member.last_name%type,
p_fname member.first_name%type,
p_addr member.address%type,
p_city member.city%type,
p_phone member.phone%type
);

FUNCTION new_rental(
p_mem_id rental.member_id%type,
p_title_id rental.title_id%type
)
return DATE;

FUNCTION new_rental(
p_lname MEMBER.LAST_NAME%type,
p_title_id rental.title_id%type
)
return DATE;

PROCEDURE return_movie(
p_title_id title_copy.title_id%type,
p_copy_id TITLE_COPY.COPY_ID%type,
p_status TITLE_COPY.STATUS%type
);
end video_pkg;
```

```
CREATE OR REPLACE PACKAGE BODY VIDEO_PKG
     IS
     ---SUPPORTING PRIVATE PROCEDURES AND FUNCTIONS
     ---private
       PROCEDURE exception_handler(
         err_code number,
         err_location varchar)
         IS
           BEGIN
            CASE err_code
               WHEN -1 THEN RAISE_APPLICATION_ERROR(-20111,
               'Error: Duplicate Member_ID attempted with '||
               err_location|| ' procedure.',FALSE);
               WHEN -2292 THEN RAISE_APPLICATION_ERROR(-20112,
               'Error: Editing Member_ID, Copy_ID, or
               Title_ID with foreign values with '|| err_location|| '
               procedure.',FALSE);
               WHEN +100 THEN RAISE_APPLICATION_ERROR(-20113,
               'Error: Data not found with '||
               err_location|| ' procedure.',FALSE);
            END CASE;

           EXCEPTION
             WHEN CASE_NOT_FOUND THEN
               RAISE_APPLICATION_ERROR(-20116,'Error: error occured
               with '||err_location||
               ' procedure.',FALSE);
       END exception_handler;

       --private
       PROCEDURE isMember(
       p_member_id MEMBER.MEMBER_ID%TYPE,
       err_code OUT number,
       p_isMember OUT Boolean
       )
       IS
       v_mem_id member.member_id%TYPE;
       BEGIN
         SELECT member_id INTO v_mem_id FROM MEMBER
         WHERE member_id=p_member_id;
         p_isMember:=TRUE;
       EXCEPTION
         WHEN NO_DATA_FOUND THEN
          err_code:=SQLCODE;
          p_isMember:=FALSE;
         WHEN OTHERS THEN
          err_code:=SQLCODE;
          p_isMember:=FALSE;
       END isMember;

       --private
```

```
PROCEDURE isValidTitle(
p_title_id TITLE.TITLE_ID%TYPE,
err_code OUT number,
p_isValidTitle OUT Boolean
)
IS
v_title_id member.member_id%TYPE;
BEGIN
  SELECT TITLE_ID INTO v_title_id FROM TITLE
  WHERE TITLE_ID=p_title_id;
  p_isValidTitle:=TRUE;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   err_code:=SQLCODE;
   p_isValidTitle:= FALSE;
  WHEN OTHERS THEN
   err_code:=SQLCODE;
   p_isValidTitle:=FALSE;
END isValidTitle;

--private
PROCEDURE isValidCopy(
p_copy_id TITLE_COPY.COPY_ID%TYPE,
p_title_id TITLE.TITLE_ID%TYPE,
err_code OUT number,
p_isValidCopy OUT Boolean
)
IS
v_copy_id TITLE_COPY.COPY_id%TYPE;
BEGIN
  SELECT COPY_ID INTO v_copy_id FROM TITLE_COPY
  WHERE COPY_ID=p_copy_id AND TITLE_ID=p_title_id;
  p_isValidCopy:=TRUE;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   err_code:=SQLCODE;
   p_isValidCopy:= FALSE;
  --WHEN OTHERS THEN
  -- err_code:=SQLCODE;
   --p_isValidCopy:= FALSE;
END isValidCopy;

--private
FUNCTION isValidStatus (
p_status_type TITLE_COPY.STATUS%TYPE
)
RETURN BOOLEAN IS
v_status Boolean := FALSE;
BEGIN
  IF (p_status_type = 'AVAILABLE' OR
     p_status_type = 'RENTED' OR
     p_status_type = 'DAMAGED')
```

```
  THEN
     v_status := TRUE;
  END IF;
RETURN v_status;

END isValidStatus;
-----private
FUNCTION isMember (
p_lname MEMBER.LAST_NAME%TYPE
)
RETURN BOOLEAN IS
v_isMember BOOLEAN := FALSE;
v_lname MEMBER.LAST_NAME%TYPE;
BEGIN
  SELECT LAST_NAME INTO v_lname FROM MEMBER
  WHERE LAST_NAME=p_lname;
  v_isMember := TRUE;
RETURN v_isMember;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN v_isMember;
  WHEN TOO_MANY_ROWS THEN
    v_isMember := TRUE;
    RETURN v_isMember;

END isMember;
--private
PROCEDURE get_copy_status(
p_title_id IN title.title_id%TYPE,
p_check_status OUT BOOLEAN
)
IS
v_status VARCHAR(25):='AVAILABLE';
availabe_copy number := 0;
err_code number;
foriegn_key_viloation EXCEPTION;
PRAGMA EXCEPTION_INIT (foriegn_key_viloation,-2292);
BEGIN
  --find number of "AVAILABLE' copies
 SELECT COUNT(*) INTO availabe_copy FROM TITLE_COPY
 WHERE TITLE_COPY.TITLE_ID=p_title_id AND
                        TITLE_COPY.STATUS=v_status;
    IF availabe_copy = 0 THEN
      p_check_status:= FALSE;
    ELSE
      p_check_status:= TRUE;
    END IF;

EXCEPTION
  WHEN NO_DATA_FOUND THEN
   raise_application_error(-20114,'Error: Title_id'||' '||
```

```
        p_title_id||' '||'not found',False);
      WHEN foriegn_key_viloation THEN
       raise_application_error(-20114,'Error: Title_id'||' '||
       p_title_id||' '||'not found',False);
      WHEN others then
        raise_application_error(-20115,'Error occured
        with checking status of copies',False);

   END get_copy_status;
```

a. NEW_MEMBER: a public procedure that adds a new member to the MEMBER table. For the member ID number, use the sequence MEMBER_ID_SEQ; for the join date, use SYSDATE. Pass all other values to be inserted into a new row as parameters.

```
PROCEDURE new_member(
  p_lname member.last_name%type,
  p_fname member.first_name%type,
  p_addr member.address%type,
  p_city member.city%type,
  p_phone member.phone%type
  )
  IS
    err_code number;
 BEGIN
      INSERT INTO MEMBER(member_id,last_name,
      first_name,address,city,phone,join_date)
      VALUES
      (MEMBER_ID_SEQ.NEXTVAL,p_lname,p_fname,p_addr,p_city,
       p_phone,TRUNC(SYSDATE));
 EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    exception_handler(SQLCODE,'new_member');
  --WHEN OTHERS THEN
     --exception_handler(SQLCODE,'new_member');

END new_member;
```

NEW_RENTAL: an overloaded public function to record a new rental. Pass the title ID number for the video that a customer wants to rent, and either the customer's last name or his member ID number into the function.

The function should return the due date for the video. Due dates are three days from the date the video is rented. If the status for a movie requested is listed as AVAILABLE in the TITLE_COPY table for one copy of this title, then update this TITLE_COPY table and set the status to RENTED. If there is no copy available, the function must return NULL.
Then, insert a new record into the RENTAL table identifying the booked date as today's date, the copy ID number, the member ID number, the title ID number, and the expected return date.
Be aware of multiple customers with the same last name. in this case, have the function return NULL, and display a list of the customers' names that match and their ID numbers.

```
FUNCTION new_rental(
```

```
p_mem_id rental.member_id%type,
p_title_id rental.title_id%type
)
RETURN DATE IS
  due_date DATE := null;
  v_copy_id rental.copy_id%type;
  v_copy_status boolean;
  v_status title_copy.status%type:='AVAILABLE';
  err_code number;
  foriegn_key_viloation EXCEPTION;
  member_not_found EXCEPTION;
  title_not_found EXCEPTION;
  PRAGMA EXCEPTION_INIT (foriegn_key_viloation,-2292);
  PRAGMA EXCEPTION_INIT (member_not_found,-20117);
  PRAGMA EXCEPTION_INIT (title_not_found,-20118);
  v_isMember boolean;
  v_isValidTitle boolean;
BEGIN
  isMember(p_mem_id,err_code,v_isMember);
  isValidTitle(p_title_id,err_code,v_isValidTitle);
  IF v_isMember THEN
    IF v_isValidTitle THEN
      get_copy_status(p_title_id,v_copy_status);
      IF v_copy_status = FALSE THEN
      DBMS_OUTPUT.PUT_LINE('Copies not available');
      due_date := null;
      reserve_movie(p_mem_id,p_title_id);
      ELSE
        SELECT COPY_ID INTO v_copy_id FROM TITLE_COPY
        WHERE STATUS = v_status
        AND TITLE_ID = p_title_id
        AND COPY_ID =
        (SELECT MIN(COPY_ID) FROM TITLE_COPY
        WHERE STATUS = v_status
        AND TITLE_ID = p_title_id);

        UPDATE TITLE_COPY tc SET STATUS='RENTED'
        WHERE STATUS = v_status
        AND TITLE_ID = p_title_id
        AND COPY_ID = v_copy_id;

        due_date := SYSDATE + 3;

        INSERT INTO
   RENTAL(book_date,copy_id,member_id,title_id,act_ret_date,
        exp_ret_date)

   VALUES(SYSDATE,v_copy_id,p_mem_id,p_title_id,NULL,due_date);
      END IF;
    ELSE
      raise title_not_found;
    END IF;
```

```
   ELSE
     raise member_not_found;
  END IF;
RETURN due_date;

EXCEPTION
  WHEN member_not_found THEN
    raise_application_error(-20117,'Error: Member_ID '||p_mem_id||'
not found');
  WHEN title_not_found THEN
    raise_application_error(-20118,'Error: Title_ID
'||p_title_id||' not found');
  WHEN no_data_found THEN
    exception_handler(SQLCODE,'new_rental');
  WHEN foriegn_key_viloation THEN
    exception_handler(SQLCODE,'new_rental');
  --WHEN OTHERS THEN
   --exception_handler(SQLCODE,'new_rental');

END new_rental;


FUNCTION new_rental(
p_lname MEMBER.LAST_NAME%type,
p_title_id rental.title_id%type
)
RETURN DATE IS
  due_date DATE:=NULL;
  v_memid member.member_id%type;
  v_copy_id rental.copy_id%type;
  v_copy_status boolean;
  v_isValidTitle boolean;
  v_status title_copy.status%type:= 'AVAILABLE';
  foriegn_key_viloation EXCEPTION;
  title_not_found EXCEPTION;
  last_name_not_found EXCEPTION;
  PRAGMA EXCEPTION_INIT (foriegn_key_viloation,-2292);
  PRAGMA EXCEPTION_INIT (title_not_found,-20118);
  PRAGMA EXCEPTION_INIT (last_name_not_found,-20121);
  err_code number;
  people_same_lname  number:=0;
  CURSOR c_member IS
    SELECT member_id,last_name,first_name FROM member
    WHERE member.last_name=p_lname;
  v_member c_member%ROWTYPE;
BEGIN

  IF isMember(p_lname) THEN
  SELECT COUNT(*) INTO people_same_lname FROM MEMBER
  WHERE LAST_NAME = p_lname;
  ELSE
    raise last_name_not_found;
```

```
  END IF;

  OPEN c_member;
    Loop
    FETCH c_member INTO v_member;
    EXIT WHEN c_member%NOTFOUND;
    IF people_same_lname > 1 THEN
      DBMS_OUTPUT.PUT_LINE(v_member.member_id||' '||
                           v_member.Last_NAME||', '||
                           v_member.first_name);
    ELSE
     v_memid:=v_member.member_id;
    END IF;
  END LOOP;
  CLOSE c_member;

isValidTitle(p_title_id,err_code,v_isValidTitle);

IF v_isValidTitle THEN

IF people_same_lname = 1 THEN
  get_copy_status(p_title_id,v_copy_status);
  IF v_copy_status=FALSE THEN
  due_date := null;
  ELSE
    SELECT COPY_ID INTO v_copy_id FROM TITLE_COPY
    WHERE STATUS= v_status
    AND TITLE_ID = p_title_id
    AND COPY_ID =
    (SELECT MIN(COPY_ID) FROM TITLE_COPY
    WHERE STATUS= v_status
    AND TITLE_ID = p_title_id);

    UPDATE TITLE_COPY SET STATUS='RENTED'
    WHERE STATUS= v_status
    AND TITLE_ID = p_title_id
    AND COPY_ID= v_copy_id;

   due_date := SYSDATE + 3;
   INSERT INTO
RENTAL(book_date,copy_id,member_id,title_id,act_ret_date,
   exp_ret_date)
   VALUES(SYSDATE,v_copy_id,v_memid,p_title_id,NULL,due_date);

  END IF;
END IF;
ELSE
  raise title_not_found;
END IF;

RETURN due_date;
```

```
EXCEPTION
  WHEN last_name_not_found THEN
    raise_application_error(-20121,'Error: Last_Name '||p_lname||'
not found');
  WHEN title_not_found THEN
    raise_application_error(-20118,'Error: Title_ID
'||p_title_id||' not found');
  WHEN no_data_found THEN
    exception_handler(SQLCODE,'new_rental');
  WHEN foriegn_key_viloation THEN
    exception_handler(SQLCODE,'new_rental');
 -- WHEN others then
    --exception_handler(SQLCODE,'new_rental');

END new_rental;
```

b. RETURN_MOVIE: a public procedure that updates the status of a video (available, rented, or damaged) and sets the return date.
Pass the title ID, the copy ID, and the status to this procedure.
Check whether there are reservations for that title and display a message if it is reserved.
Update the RENTAL table and set the actual return date to today's date.
Update the status in the TITLE_COPY table based on the status parameter passed into the procedure.

```
PROCEDURE return_movie(
p_title_id title_copy.title_id%type,
p_copy_id TITLE_COPY.COPY_ID%type,
p_status TITLE_COPY.STATUS%type
)
IS
v_title_id title_copy.title_id%type;
v_reservation number:=0;
title_not_found EXCEPTION;
copy_not_found EXCEPTION;
foriegn_key_viloation EXCEPTION;
status_invalid EXCEPTION;
PRAGMA EXCEPTION_INIT (status_invalid,-20120);
PRAGMA EXCEPTION_INIT (copy_not_found,-20119);
PRAGMA EXCEPTION_INIT (title_not_found,-20118);
PRAGMA EXCEPTION_INIT(foriegn_key_viloation,-2292);
err_code number;
v_isValidCopy boolean;
v_isValidTitle boolean;
BEGIN
isValidTitle(p_title_id,err_code,v_isValidTitle);

IF v_isValidTitle THEN
isValidCopy(p_copy_id,p_title_id,err_code,v_isValidCopy);
SELECT COUNT(*) INTO v_reservation
FROM RESERVATION
WHERE title_id=p_title_id;
```

```
    IF v_reservation > 0 THEN
      DBMS_OUTPUT.PUT_LINE('Title_ID'||' '||p_title_id||' '
      ||'reserved by'||' '||v_reservation||' '||'member/s');
    END IF;

    IF v_isValidCopy THEN
      IF isValidStatus(p_status) THEN
        UPDATE RENTAL SET ACT_RET_DATE= SYSDATE
        WHERE TITLE_ID=p_title_id
        AND COPY_ID=p_copy_id;

        UPDATE TITLE_COPY SET STATUS=p_status
        WHERE TITLE_ID=p_title_id
        AND COPY_ID=p_copy_id;
      ELSE
        raise status_invalid;
      END IF;
    ELSE
      raise copy_not_found;
    END IF;
  ELSE
      raise title_not_found;
  END IF;


  EXCEPTION
    WHEN title_not_found THEN
      raise_application_error(-20118,'Error: Title_ID
'||p_title_id||' not found');
    WHEN copy_not_found THEN
      raise_application_error(-20119,'Error: Copy_ID
'||p_copy_id||' not found');
    WHEN status_invalid THEN
     raise_application_error(-20120,'Error: STATUS '||p_status||'
is invalid');
    WHEN no_data_found THEN
      exception_handler(SQLCODE,'return_movie');
    WHEN foriegn_key_viloation THEN
      exception_handler(SQLCODE,'return_movie');
   -- WHEN others then
      --exception_handler(SQLCODE,'return_movie');
  END return_movie;
```

c. RESERVE_MOVIE: a private procedure that executes only if all the video copies requested in the NEW_RENTAL procedure have a status of RENTED.
Pass the member ID number and the title ID number to this procedure.
Insert a new record into the RESERVATION table and record the reservation date, member ID number, and title ID number.
Print a message indicating that a movie is reserved and its expected date of return.

```
--private
```

```
PROCEDURE reserve_movie(
  p_mem_id member.member_id%type,
  p_title_id title.title_id%type)
  IS
    v_copy_status boolean;
    v_copy_id number;
    v_reserve_times number;
    BEGIN
    get_copy_status(p_title_id,v_copy_status);

    IF v_copy_status=FALSE THEN
      SELECT COUNT(*) INTO v_reserve_times FROM RESERVATION
      WHERE TITLE_ID = p_title_id AND MEMBER_ID = p_mem_id;


      IF v_reserve_times >=1 THEN
         DBMS_OUTPUT.PUT_LINE('Requested item'||' '||
         p_title_id||' '||'reserved already');
      ELSE
        INSERT INTO RESERVATION(RES_DATE, MEMBER_ID,TITLE_ID)
        VALUES(SYSDATE,p_mem_id,p_title_id);
        DBMS_OUTPUT.PUT_LINE('Title_ID'||' '||
        p_title_id||' '||'reserved');
      END IF;
    ELSE
      DBMS_OUTPUT.PUT_LINE('Title_ID'||' '||
      p_title_id||' '||v_copy_id||' '||'available');
    END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
     raise_application_error(-20114,'Error: Title_id'||'
    '||p_title_id||' '||
       'or Member_id'||' '||p_mem_id||' '||'not found in
    Reservations');
  WHEN OTHERS THEN
    raise_application_error(-20115,'Error occured with reserving
    movies');

END reserve_movie;
```

d. EXCEPTION_HANDLER: a private procedure that is called from the exception handler of the public programs.
Pass the SQLCODE number to this procedure, and the name of the program as a text string where the error occurred.
Use RAISE_APPLICATION_ERROR to raise a customized error. Start with a unique key violation (-1) and foreign key violation (-2292). Allow the exception handler to raise a generic error for any other errors.

```
  PROCEDURE exception_handler(
    err_code number,
```

```
    err_location varchar)
    IS
      BEGIN
       CASE err_code
          WHEN -1 THEN RAISE_APPLICATION_ERROR(-20111,
          'Error: Duplicate Member_ID attempted with '||
          err_location|| ' procedure.',FALSE);
          WHEN -2292 THEN RAISE_APPLICATION_ERROR(-20112,
          'Error: Editing Member_ID, Copy_ID, or
          Title_ID with foreign values with '|| err_location|| '
          procedure.',FALSE);
          WHEN +100 THEN RAISE_APPLICATION_ERROR(-20113,
          'Error: Data not found with '||
          err_location|| ' procedure.',FALSE);
       END CASE;

      EXCEPTION
        WHEN CASE_NOT_FOUND THEN
          RAISE_APPLICATION_ERROR(-20116,'Error: error occured
          with '||err_location||
          ' procedure.',FALSE);
  END exception_handler;

END video_pkg;
```

4. Use the following scripts to test your routines:

a. **Add two members: `a_new_members.sql`**

```
EXECUTE video_pkg.new_member('Haas', 'James', 'Chestnut Street',
'Boston', '617-123-4567');
SELECT Member_ID, Last_Name, First_Name, Phone FROM Member where
last_name='Haas';
```

MEMBER_ID	LAST_NAME	FIRST_NAME	PHONE
110	Haas	James	617-123-4567

```
EXECUTE  video_pkg.new_member('Biri', 'Allan',  'Hiawatha
Drive', 'New York', '516-123-4567');
SELECT Member_ID, Last_Name, First_Name, Phone FROM Member where
last_name='Biri';
```

MEMBER_ID	LAST_NAME	FIRST_NAME	PHONE
108	Biri	Ben	614-455-9863
111	Biri	Allan	516-123-4567

b. **Add new video rentals: `b_new_rentals.sql`**

Transaction-1 and Transaction-2

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(110, 98));
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(109, 93));
```

RESULTS:

TITILE_COPY table

COPY_ID	TITLE_ID	STATUS
1	92	AVAILABLE
1	93	AVAILABLE
2	93	RENTED
1	94	AVAILABLE
1	95	AVAILABLE
2	95	AVAILABLE
3	95	RENTED
1	96	AVAILABLE
1	97	AVAILABLE
1	98	RENTED
2	98	AVAILABLE

Before Rental by 109 and 110 Members.

COPY_ID	TITLE_ID	STATUS
1	92	AVAILABLE
1	93	RENTED
2	93	RENTED
1	94	AVAILABLE
1	95	AVAILABLE
2	95	AVAILABLE
3	95	RENTED
1	96	AVAILABLE
1	97	AVAILABLE
1	98	RENTED
2	98	RENTED

After Rental

Copy_ID 2 of the Title 98 was rented by 110. Copy_id 1 of the Title_id 93 was rented by 109.

Transaction-3

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(107, 98));
```

RESERVATION TABLE

RES_DATE	MEMBER_ID	TITLE_ID
09-DEC-16	101	93
08-DEC-16	106	102
10-DEC-16	107	98

.Since there are no more copies of 98 (both copy_id 1 and 2) was rented in transactions 1 and 2 previously.

```
PL/SQL procedure successfully completed.

Copies not available
Title_ID 98 reserved
```

Transaction-4

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental('Biri', 97));
```

```
PL/SQL procedure successfully completed.

108 Biri, Ben
111 Biri, Allan
```

Since there are two members with same last names, the program print their information without a transaction.

Transaction-5

```
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97));
```

```
Error starting at line : 3 in command -
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97))
Error report -
ORA-20117: Error: Member_ID 97 not found
ORA-06512: at "C##P05.VIDEO_PKG", line 292
ORA-06512: at line 1
```

Member_ID 97 does NOT in the database. Hence, this transaction was denied with an error message.

RENT table after transaction-1, 2, 3, 4 and 5. Note that only two transactions led to update tables in above rental processes. The expiration dates for Member_id 109 and 110 were set to 3 days from book dates.

RENT TABLE

BOOK_DATE	COPY_ID	MEMBER_ID	TITLE_ID	ACT_RET_DATE	EXP_RET_DATE
09-DEC-16	2	101	93	(null)	11-DEC-16
08-DEC-16	3	102	95	(null)	10-DEC-16
07-DEC-16	1	101	98	(null)	09-DEC-16
06-DEC-16	1	106	97	08-DEC-16	08-DEC-16
07-DEC-16	1	101	92	08-DEC-16	09-DEC-16
10-DEC-16	2	110	98	(null)	13-DEC-16
10-DEC-16	1	109	93	(null)	13-DEC-16

b. Return movies: `c_return_movie.sql`

Transaction-6

```
EXECUTE video_pkg.return_movie(98, 1, 'AVAILABLE');
SELECT * FROM RENTAL WHERE COPY_ID=1 AND TITLE_ID=98;
SELECT * FROM TITLE_COPY WHERE COPY_ID=1 AND TITLE_ID=98;
```

```
PL/SQL procedure successfully completed.

Title_ID 98 reserved by 1 member/s
```

Since Member_ID 107 was on the RESERVATION list for Title_ID 98 (Transaction-4 in question 4b), on returning Title_id 98, the message "Title_ID 98 reserved by 1 member/s" is printed. The Actual_return_date in the RENTAL table is set to the today's date (the day of the item returned). Updated status to "AVAILABLE" in TITLE_COPY table.

RENTAL AND TITLE_COPY TABLES

```
BOOK_DATE    COPY_ID  MEMBER_ID   TITLE_ID ACT_RET_DATE EXP_RET_DATE
--------- ---------- ---------- ---------- ------------ ------------
07-DEC-16          1        101         98 10-DEC-16    09-DEC-16

   COPY_ID   TITLE_ID STATUS
---------- ---------- ---------------
         1         98 AVAILABLE
```

Transaction-7

```
EXECUTE video_pkg.return_movie(95, 3, 'AVAILABLE');
SELECT * FROM RENTAL WHERE COPY_ID=3 AND TITLE_ID=95;
SELECT * FROM TITLE_COPY WHERE COPY_ID=3 AND TITLE_ID=95;
```

```
PL/SQL procedure successfully completed.

BOOK_DATE    COPY_ID  MEMBER_ID   TITLE_ID ACT_RET_DATE EXP_RET_DATE
--------- ---------- ---------- ---------- ------------ ------------
08-DEC-16          3        102         95 10-DEC-16    10-DEC-16

   COPY_ID   TITLE_ID STATUS
---------- ---------- ---------------
         3         95 AVAILABLE
```

Transaction-8

```
EXECUTE video_pkg.return_movie(111, 1, 'RENTED');
```

```
Error starting at line : 5 in command -
EXECUTE video_pkg.return_movie(111, 1, 'RENTED')
Error report -
ORA-20118: Error: Title_ID 111 not found
ORA-06512: at "C##P05.VIDEO_PKG", line 460
ORA-06512: at line 1
```

There is NO item with TITLE_ID 111 in the database (in TITLE table); hence, an error is prompted.

5. The business hours for the video store are 8:00 AM through 10:00 PM, Sunday through Friday, and 8:00 AM through 12:00 PM on Saturday. To ensure that the tables can be modified only during these hours, create a stored procedure that is called by triggers on the tables
   a. Create a stored procedure called TIME_CHECK that checks the current time against business hours. If the current time is not within business hours, use the RAISE_APPLICATION_ERROR procedure to give an appropriate message.

```
create or replace procedure time_check as
     b_hour_major boolean;
     b_hour_minor boolean;
BEGIN
  b_hour_major:=
     (to_char(SYSDATE,'d') BETWEEN '1' AND '6' AND
          to_char(SYSDATE,'HH24:MI') BETWEEN '08:00' AND
     '22:00');

  b_hour_minor:=(to_char(SYSDATE,'d')='7' AND
           to_char(SYSDATE,'HH24:MI') BETWEEN '08:00' AND
     '12:00');

 IF NOT (b_hour_major OR b_hour_minor) THEN
          RAISE_APPLICATION_ERROR(-20205, 'Modifying records not
          permitted in this hours',FALSE);
          END IF;
end time_check;
```

   b. Create a trigger on each of the five tables. Fire the trigger before data is inserted, updated, and deleted from the tables. Call your TIME_CHECK procedure from each of these triggers.

```
create or replace trigger secure_dml_trg_member
before delete or insert or update on member
begin
TIME_CHECK;
end;

create or replace trigger secure_dml_trg_rent
before delete or insert or update on rental
begin
TIME_CHECK;
end;

create or replace trigger secure_dml_trg_reservation
before delete or insert or update on reservation
begin
TIME_CHECK;
end;

create or replace trigger secure_dml_trg_title
before delete or insert or update on title
begin
TIME_CHECK;
```

```
End;

create or replace trigger secure_dml_trg_title_copy
before delete or insert or update on title_copy
begin
 TIME_CHECK;
end;
```

EXECUTE video_pkg.new_member('Haas', 'James', 'Chestnut Street', 'Boston', '617-123-4567')

```
Error starting at line : 3 in command -
EXECUTE video_pkg.new_member('Haas', 'James', 'Chestnut Street', 'Boston', '617-123-4567')
Error report -
ORA-20205: Modifying records not permitted in this hours
ORA-06512: at "C##P05.TIME_CHECK", line 13
ORA-06512: at "C##P05.SECURE_DML_TRG_MEMBER", line 2
ORA-04088: error during execution of trigger 'C##P05.SECURE_DML_TRG_MEMBER'
ORA-06512: at "C##P05.VIDEO_PKG", line 220
ORA-06512: at line 1
```

EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(109, 93));

```
Error starting at line : 3 in command -
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(109, 93))
Error report -
ORA-20205: Modifying records not permitted in this hours
ORA-06512: at "C##P05.TIME_CHECK", line 13
ORA-06512: at "C##P05.SECURE_DML_TRG_TITLE_COPY", line 2
ORA-04088: error during execution of trigger 'C##P05.SECURE_DML_TRG_TITLE_COPY'
ORA-06512: at "C##P05.VIDEO_PKG", line 271
ORA-06512: at line 1
```

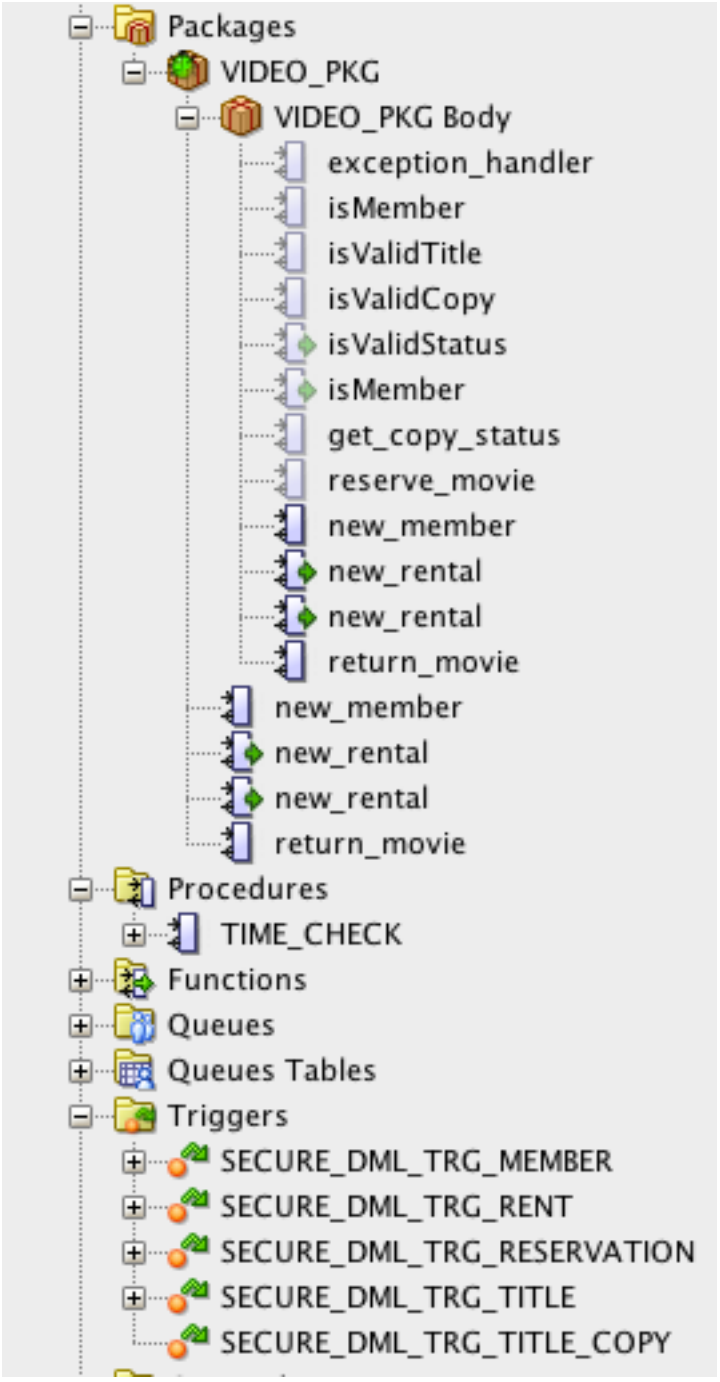
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental('Biri', 97));

```
Error starting at line : 3 in command -
EXEC DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental('Biri', 97))
Error report -
ORA-20205: Modifying records not permitted in this hours
ORA-06512: at "C##P05.TIME_CHECK", line 13
ORA-06512: at "C##P05.SECURE_DML_TRG_TITLE_COPY", line 2
ORA-04088: error during execution of trigger 'C##P05.SECURE_DML_TRG_TITLE_COPY'
ORA-06512: at "C##P05.VIDEO_PKG", line 368
ORA-06512: at line 1
```

EXECUTE video_pkg.return_movie(98, 1, 'AVAILABLE');

```
Error starting at line : 3 in command -
EXECUTE video_pkg.return_movie(98, 1, 'AVAILABLE')
Error report -
ORA-20205: Modifying records not permitted in this hours
ORA-06512: at "C##P05.TIME_CHECK", line 13
ORA-06512: at "C##P05.SECURE_DML_TRG_RENT", line 2
ORA-04088: error during execution of trigger 'C##P05.SECURE_DML_TRG_RENT'
ORA-06512: at "C##P05.VIDEO_PKG", line 433
ORA-06512: at line 1
```

SCREEN SHOT SHOWING VIDEO_PKG PACKAGE , TIME-CHECK STORED PROCEDURE, AND TRIGGERS FOR FIVE TABLES.



-- END --

Praminda Imaduwa-Gamage
12/10/2016