DATABASE ADMINISTRATION

Lab 11 – PL / SQL packages

229840 – Wiktor Bechciński
229850 – Kamil Budzyn

Run the buildvid1.sql script that contains table and sequence definitions.
Run the buildvid2.sql script that contains the data.

```
SQL> @/home/oracle/buildvid1.sql

Table dropped.


Table dropped.


Table dropped.


Table dropped.


Table dropped.

Please wait while tables are created....

Table created.


Table created.


Table created.


Table created.


Table created.

Tables created.

Sequence dropped.


Sequence dropped.

Creating Sequences...

Sequence created.


Sequence created.

Sequences created.
Run buildvid2.sql now to populate the above tables.
SQL> @/home/oracle/buildvid2.sql

1 row created.


1 row created.


1 row created.


1 row created.


1 row created.


1 row created.
```

```
1 row created.

1 row created.

1 row created.

Commit complete.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

Commit complete.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

Commit complete.
```

```
1 row created.


1 row created.


Commit complete.


1 row created.


1 row created.


1 row created.


1 row created.


1 row created.


Commit complete.

** Tables built and data loaded **
SQL>
```

Define a VIDEO_PKG package with the following procedures and features:

NEW_MEMBER :A public procedure that allows you to add a new member to the MEMBER table. To insert the ID member number, use the MEMBER_ID_SEQ sequence. The join date is SYSDATE. The remaining values of the new row are passed as parameters.

NEW_RENTAL: overloaded public function for registering new rentals. You must provide the title number (ID) of the videocassette that the customer wants to rent and either the customer's name or number. The function should return a suggested due date. The return date is 3 days from the date of rental. If the status of the movie that the customer wants to rent is set to AVAILABLE in the table TITLE_COPY for one copy of the film, set it to RENTED. If there is no copy available, you want the function to return NULL. The procedure inserts a new record into the RENTAL table with today's date, as the date of booking, the copy ID number, the member ID number, the title ID number, and the suggested return date. Consider the possibility of clients with the same last name (the function can return NULL in this case and display last names, first names, and corresponding identifiers).

RETURN_MOVIE:A public procedure that updates the status of a videotape (available, rented, or damaged) and sets a return date. The procedure must be followed by a title ID, a copy ID, and a status. Check if there are reservations for this title and display a message if there are any. Update the RENTAL table and set the return date to today. Update the status in the TITLE_COPY table

RESERVE_MOVIE :a private procedure that is performed only if all copies of the film requested in the NEW_RENTAL procedure are RENTED. The member ID number and the title ID number must be submitted to the procedure. Insert the new record into the RESERVATION table. Write a message informing you that you have booked and the expected date of return of the cassette.

3.EXCEPTION_HANDLER: A private procedure that is called from the exceptions section of public entities. Pass the SQLCODE number and subroutine name (as a string) to the procedure. Use the syntax RAISE_APPLICATION_ERROR cause a known exception (violation of primary key (-1), foreign key (-2292)). For other errors, you can leave the default messages.

Validate your definitions with the following scripts:

• Add two members using the sol_apb_04_b_new_rentals.sqlscript.
• Return the video using the sol_apb_04_c_return_movie.sql script.

The opening hours of the rental company are 8:00 a.m. to 10:00 p.m., from Sunday to Friday, on Saturdays 8:00 a.m. to 12:00 a.m.

5. Define a TIME_CHECK procedure that checks the current time for working hours. If these are not working hours, use the RAISE_APPLICATION_ERROR generating the appropriate message.

6. Define a trigger on each of the five tables to run before inserting, updating, or deleting data. From the triggers, invoke the TIME_CHECK procedure.

```
SQL> @/home/oracle/lab11.sql

Package created.


Package body created.


Trigger created.


Trigger created.


Trigger created.


Trigger created.


Trigger created.

SQL>
```

```
SQL> @/home/oracle/sol_apb_04_a.sql
Added new member

PL/SQL procedure successfully completed.

Added new member

PL/SQL procedure successfully completed.

SQL> @/home/oracle/sol_apb_04_b.sql
The movie has been rented successfully.
19-JAN-22

PL/SQL procedure successfully completed.

The movie has been rented successfully.
19-JAN-22

PL/SQL procedure successfully completed.

This movie is now reserved, predictable free term: 15-JAN-22
All copies of this movie are already rented. Added a reservation.

PL/SQL procedure successfully completed.

There are many people with the given name:
Last name: Biri First name: Ben Member ID: 108
Last name: Biri First name: Allan Member ID: 111

PL/SQL procedure successfully completed.

BEGIN DBMS_OUTPUT.PUT_LINE(video_pkg.new_rental(97, 97)); END;

*
ERROR at line 1:
ORA-20001: The specified client or movie does not exist in the database. Error
in: NEW_RENTAL
ORA-06512: at "HR.VIDEO_PKG", line 211
ORA-06512: at "HR.VIDEO_PKG", line 50
ORA-06512: at line 1
```

```
SQL> @/home/oracle/sol_apb_04_c.sql
Returning movie completed.

PL/SQL procedure successfully completed.

Returning movie completed.

PL/SQL procedure successfully completed.

BEGIN video_pkg.return_movie(111, 1, 'RENTED'); END;

*
ERROR at line 1:
ORA-20001: The specified client or movie does not exist in the database. Error
in: NEW_RENTAL
ORA-06512: at "HR.VIDEO_PKG", line 211
ORA-06512: at "HR.VIDEO_PKG", line 176
ORA-06512: at line 1


SQL>
```

lab11.sql file:

```
SET SERVEROUTPUT ON

CREATE OR REPLACE PACKAGE VIDEO_PKG AS
   PROCEDURE NEW_MEMBER (
      p_last_name MEMBER.LAST_NAME%TYPE,
      p_first_name MEMBER.FIRST_NAME%TYPE,
      p_address MEMBER.ADDRESS%TYPE,
      p_city MEMBER.CITY%TYPE,
      p_phone MEMBER.PHONE%TYPE
   );

   function NEW_RENTAL (
   f_member_id MEMBER.MEMBER_ID%TYPE,
   f_title_id TITLE.TITLE_ID%TYPE
   ) RETURN RENTAL.EXP_RET_DATE%TYPE;

   function NEW_RENTAL (
   f_last_name in MEMBER.LAST_NAME%TYPE,
   f_title_id TITLE.TITLE_ID%TYPE
   ) RETURN RENTAL.EXP_RET_DATE%TYPE;

   PROCEDURE RETURN_MOVIE (
      p_title_id TITLE.TITLE_ID%TYPE,
      p_copy_id TITLE_COPY.COPY_ID%TYPE,
      p_copy_status TITLE_COPY.STATUS%TYPE
   );

   PROCEDURE TIME_CHECK;

END VIDEO_PKG;
/
CREATE or REPLACE PACKAGE BODY VIDEO_PKG
as

   PROCEDURE RESERVE_MOVIE (
   p_member_id MEMBER.MEMBER_ID%TYPE,
   p_title_id TITLE.TITLE_ID%TYPE
   );

   PROCEDURE EXCEPTION_HANDLER (
   p_number NUMBER,
   p_message VARCHAR2
   );

   PROCEDURE NEW_MEMBER (
      p_last_name MEMBER.LAST_NAME%TYPE,
      p_first_name MEMBER.FIRST_NAME%TYPE,
      p_address MEMBER.ADDRESS%TYPE,
      p_city MEMBER.CITY%TYPE,
      p_phone MEMBER.PHONE%TYPE
   )IS
   BEGIN
      INSERT INTO MEMBER VALUES(MEMBER_ID_SEQ.nextval, p_last_name, p_first_name, p_address, p_city, p_phone, SYSDATE);
      DBMS_OUTPUT.PUT_LINE('Added new member');
   END NEW_MEMBER;

   function NEW_RENTAL (
   f_member_id MEMBER.MEMBER_ID%TYPE,
   f_title_id TITLE.TITLE_ID%TYPE
   ) RETURN RENTAL.EXP_RET_DATE%TYPE

   IS v_copy_id TITLE_COPY.COPY_ID%TYPE;
   v_exp_ret_date RENTAL.EXP_RET_DATE%TYPE;
   va_copies_count NUMBER := 0;
   vd_copies_count NUMBER := 0;
   copies_count NUMBER := 0;
   v_min_member_id MEMBER.MEMBER_ID%TYPE;
   v_max_member_id MEMBER.MEMBER_ID%TYPE;
   v_min_title_id TITLE.TITLE_ID%TYPE;
   v_max_title_id TITLE.TITLE_ID%TYPE;

   BEGIN
      SELECT min(MEMBER_ID) INTO v_min_member_id FROM MEMBER;
      SELECT max(MEMBER_ID) INTO v_max_member_id FROM MEMBER;
      SELECT min(TITLE_ID) INTO v_min_title_id FROM TITLE;
      SELECT max(TITLE_ID) INTO v_max_title_id FROM TITLE;
      UPDATE MEMBER SET MEMBER_ID = v_min_member_id WHERE MEMBER_ID = v_min_member_id;
      IF f_member_id < v_min_member_id or f_member_id > v_max_member_id or f_title_id > v_max_title_id or f_title_id < v_min_title_id
THEN
         EXCEPTION_HANDLER(-20001, 'NEW_RENTAL');
      END IF;
      SELECT count(COPY_ID) INTO va_copies_count FROM TITLE_COPY WHERE TITLE_ID = f_title_id AND STATUS = 'AVAILABLE';
      SELECT count(COPY_ID) INTO vd_copies_count FROM TITLE_COPY WHERE TITLE_ID = f_title_id AND STATUS = 'DAMAGED';
      SELECT count(COPY_ID) INTO copies_count FROM TITLE_COPY WHERE TITLE_ID = f_title_id;
      IF va_copies_count = 0 THEN
         BEGIN
            IF vd_copies_count = copies_count THEN
               EXCEPTION_HANDLER(-20002, 'NEW_RENTAL');
            ELSE
               BEGIN
                  RESERVE_MOVIE(f_member_id, f_title_id);
                  DBMS_OUTPUT.PUT_LINE('All copies of this movie are already rented. Added a reservation.');
                  RETURN NULL;
               END;
            END IF;
         END;
      ELSE
         BEGIN
            SELECT COPY_ID INTO v_copy_id FROM TITLE_COPY WHERE STATUS = 'AVAILABLE' AND TITLE_ID = f_title_id AND rownum = 1;
            UPDATE TITLE_COPY SET STATUS = 'RENTED' WHERE STATUS = 'AVAILABLE' AND TITLE_ID = f_title_id AND v_copy_id = COPY_ID;
            INSERT INTO RENTAL VALUES (SYSDATE, v_copy_id, f_member_id, f_title_id, NULL, SYSDATE+3);
            SELECT EXP_RET_DATE INTO v_exp_ret_date FROM RENTAL WHERE COPY_ID = v_copy_id AND MEMBER_ID = f_member_id AND
TITLE_ID = f_title_id;
            DBMS_OUTPUT.PUT_LINE('The movie has been rented successfully.');
            RETURN v_exp_ret_date;
         END;
      END IF;
   END NEW_RENTAL;

   function NEW_RENTAL (
   f_last_name in MEMBER.LAST_NAME%TYPE,
   f_title_id TITLE.TITLE_ID%TYPE
   ) RETURN RENTAL.EXP_RET_DATE%TYPE

   IS v_copy_id TITLE_COPY.COPY_ID%TYPE;
   v_exp_ret_date RENTAL.EXP_RET_DATE%TYPE;
   va_copies_count NUMBER := 0;
   vd_copies_count NUMBER := 0;
   copies_count NUMBER := 0;
```

```
  v_last_name_count NUMBER;
  v_member_id MEMBER.MEMBER_ID%TYPE;
  c_last_name MEMBER.LAST_NAME%TYPE;
  c_first_name MEMBER.FIRST_NAME%TYPE;
  c_member_id MEMBER.MEMBER_ID%TYPE;
  v_min_title_id TITLE.TITLE_ID%TYPE;
  v_max_title_id TITLE.TITLE_ID%TYPE;
  CURSOR CUR_NR IS SELECT LAST_NAME, FIRST_NAME, MEMBER_ID FROM MEMBER;

  BEGIN
    SELECT MIN(TITLE_ID) INTO v_min_title_id FROM TITLE;
    SELECT MAX(TITLE_ID) INTO v_max_title_id FROM TITLE;
    UPDATE TITLE SET TITLE_ID = v_min_title_id WHERE TITLE_ID = v_min_title_id;
    SELECT COUNT(MEMBER_ID) INTO v_last_name_count FROM MEMBER WHERE f_last_name = LAST_NAME;
    IF (v_last_name_count > 1) THEN
      BEGIN
      DBMS_OUTPUT.PUT_LINE('There are many people with the given name: ');
      OPEN CUR_NR;
        LOOP
          FETCH CUR_NR INTO c_last_name, c_first_name, c_member_id;
          EXIT WHEN CUR_NR%NOTFOUND;
          IF (c_last_name = f_last_name) THEN
            DBMS_OUTPUT.PUT_LINE('Last name: ' || c_last_name || ' First name: ' || c_first_name || ' Member ID: ' || c_member_id);
          END IF;
        END loop;
      close CUR_NR;
      RETURN NULL;
      END;
    elsIF (v_last_name_count = 1) THEN
      BEGIN
        IF  f_title_id > v_max_title_id OR f_title_id < v_min_title_id THEN
          EXCEPTION_HANDLER(-20004, 'NEW_RENTAL');
        END IF;
        SELECT count(COPY_ID) INTO va_copies_count FROM TITLE_COPY WHERE TITLE_ID = f_title_id AND STATUS = 'AVAILABLE';
        SELECT count(COPY_ID) INTO vd_copies_count FROM TITLE_COPY WHERE TITLE_ID = f_title_id AND STATUS = 'DAMAGED';
        SELECT count(COPY_ID) INTO copies_count FROM TITLE_COPY WHERE TITLE_ID = f_title_id;
        IF va_copies_count = 0 THEN
          BEGIN
            IF vd_copies_count = copies_count THEN
              EXCEPTION_HANDLER(-20002, 'NEW_RENTAL');
            ELSE
              BEGIN
                SELECT MEMBER_ID INTO v_member_id FROM MEMBER WHERE f_last_name = LAST_NAME;
                RESERVE_MOVIE(v_member_id, f_title_id);
                DBMS_OUTPUT.PUT_LINE('All copies of this movie are already rented. Added a reservation.');
                RETURN NULL;
              END;
            END IF;
          END;
        ELSE
          BEGIN
            SELECT COPY_ID INTO v_copy_id FROM TITLE_COPY WHERE STATUS = 'AVAILABLE' AND TITLE_ID = f_title_id AND rownum =
1;
            UPDATE TITLE_COPY SET STATUS = 'RENTED' WHERE STATUS = 'AVAILABLE' AND TITLE_ID = f_title_id AND v_copy_id =
COPY_ID;
            SELECT MEMBER_ID INTO v_member_id FROM MEMBER WHERE f_last_name = LAST_NAME;
            INSERT INTO RENTAL VALUES (SYSDATE, v_copy_id, v_member_id, f_title_id, NULL, SYSDATE+3);
            SELECT EXP_RET_DATE INTO v_exp_ret_date FROM RENTAL WHERE COPY_ID = v_copy_id AND MEMBER_ID = v_member_id
AND TITLE_ID = f_title_id;
            DBMS_OUTPUT.PUT_LINE('The movie has been rented successfully.');
            RETURN v_exp_ret_date;
          END;
        END IF;
      END;
    ELSE
      EXCEPTION_HANDLER(-20003, 'NEW_RENTAL');
    END IF;
  END NEW_RENTAL;

  PROCEDURE RETURN_MOVIE (
    p_title_id TITLE.TITLE_ID%TYPE,
      p_copy_id TITLE_COPY.COPY_ID%TYPE,
      p_copy_status TITLE_COPY.STATUS%TYPE
   )is
   v_min_copy_id TITLE_COPY.COPY_ID%TYPE;
   v_max_copy_id TITLE_COPY.COPY_ID%TYPE;
   v_min_title_id TITLE.TITLE_ID%TYPE;
   v_max_title_id TITLE.TITLE_ID%TYPE;
   rent_count NUMBER := 0;
   BEGIN
      SELECT min(TITLE_ID) INTO v_min_title_id FROM TITLE;
      SELECT max(TITLE_ID) INTO v_max_title_id FROM TITLE;
      UPDATE TITLE SET TITLE_ID = v_min_title_id WHERE TITLE_ID = v_min_title_id;
      IF p_copy_status != 'RENTED' AND p_copy_status != 'AVAILABLE' AND p_copy_status != 'DAMAGED' THEN
         EXCEPTION_HANDLER(-20005, 'NEW_RENTAL');
      END IF;
      SELECT min(COPY_ID) INTO v_min_copy_id FROM TITLE_COPY;
      SELECT max(COPY_ID) INTO v_max_copy_id FROM TITLE_COPY;
      IF p_copy_id < v_min_copy_id OR p_copy_id > v_max_copy_id OR p_title_id > v_max_title_id OR p_title_id < v_min_title_id THEN
        EXCEPTION_HANDLER(-20001, 'NEW_RENTAL');
      END IF;
      SELECT count(COPY_ID) INTO rent_count FROM TITLE_COPY WHERE STATUS = 'RENTED' AND TITLE_ID = p_title_id;
      IF rent_count = 0 THEN
         EXCEPTION_HANDLER(-20006, 'NEW_RENTAL');
      ELSE
         BEGIN
            UPDATE TITLE_COPY SET STATUS = p_copy_status WHERE p_title_id = TITLE_ID AND COPY_ID = p_copy_id;
            UPDATE RENTAL SET act_ret_date = current_date WHERE p_title_id = TITLE_ID AND COPY_ID = p_copy_id;
            DBMS_OUTPUT.PUT_LINE('Returning movie completed.');
         END;
      END IF;
   END RETURN_MOVIE;

   PROCEDURE RESERVE_MOVIE (
   p_member_id MEMBER.MEMBER_ID%TYPE,
   p_title_id TITLE.TITLE_ID%TYPE
   )
   is
   v_free_term RESERVATION.RES_DATE%TYPE;
   BEGIN
      SELECT MIN(EXP_RET_DATE) INTO v_free_term FROM RENTAL WHERE p_title_id = TITLE_ID;
      INSERT INTO RESERVATION VALUES (current_date, p_member_id, p_title_id);
      DBMS_OUTPUT.PUT_LINE('This movie is now reserved, predictable free term: ' || v_free_term);
   END RESERVE_MOVIE;

   PROCEDURE EXCEPTION_HANDLER (
   p_number NUMBER,
   p_message VARCHAR2
   )
   is
   BEGIN
      IF p_number = -20001 THEN
         RAISE_APPLICATION_ERROR(p_number, 'The specified client or movie does not exist in the database. Error in: ' || p_message);
      elsIF p_number = -20002 THEN
         RAISE_APPLICATION_ERROR(p_number, 'Copies are damaged. Error in: ' || p_message);
      elsIF p_number = -20003 THEN
         RAISE_APPLICATION_ERROR(p_number, 'Customer with that last name does not exist. Error in: ' || p_message);
      elsIF p_number = -20004 THEN
         RAISE_APPLICATION_ERROR(p_number, 'Movie does not exist. Error in: ' || p_message);
      elsIF p_number = -20005 THEN
         RAISE_APPLICATION_ERROR(p_number, 'Incorrectly specified status. Error in: ' || p_number);
      elsIF p_number = -20006 THEN
         RAISE_APPLICATION_ERROR(p_number, 'The movie with the given number is not rented. Error in: ' || p_message);
      END IF;
   END EXCEPTION_HANDLER;

PROCEDURE TIME_CHECK IS
   p_day VARCHAR2(20);
   p_hour VARCHAR2(20);
   BEGIN
      SELECT TO_CHAR(sysdate, 'DAY') INTO p_day FROM DUAL;
      SELECT TO_CHAR(sysdate, 'HH24') INTO p_hour FROM DUAL;
```

```
    IF p_day = 'SATURDAY' THEN
      BEGIN
        IF p_hour < 8 OR p_hour > 12 THEN
          RAISE_APPLICATION_ERROR(-20200, 'Rental company is closed');
        END IF;
      END;
    ELSE
      BEGIN
        IF p_hour < 8 OR p_hour > 22 THEN
          RAISE_APPLICATION_ERROR(-20200, 'Rental company is closed');
        END IF;
      END;
    END IF;
  END TIME_CHECK;

END VIDEO_PKG;
/

CREATE OR REPLACE TRIGGER working_hours_RESERVATION
  BEFORE
  DELETE OR INSERT OR UPDATE
  ON RESERVATION FOR EACH ROW
  BEGIN
    VIDEO_PKG.TIME_CHECK;
  END working_hours_RESERVATION;
/
  CREATE OR REPLACE TRIGGER working_hours_title
  BEFORE
  DELETE OR INSERT OR UPDATE
  ON TITLE FOR EACH ROW
  BEGIN
    VIDEO_PKG.TIME_CHECK;
  END working_hours_title;
/
  CREATE OR REPLACE TRIGGER working_hours_copy
  BEFORE
  DELETE OR INSERT OR UPDATE
  ON TITLE_COPY FOR EACH ROW
  BEGIN
    VIDEO_PKG.TIME_CHECK;
  END working_hours_copy;
/
  CREATE OR REPLACE TRIGGER working_hours_rental
  BEFORE
  DELETE OR INSERT OR UPDATE
  ON RENTAL FOR EACH ROW
  BEGIN
    VIDEO_PKG.TIME_CHECK;
  END working_hours_rental;
/
  CREATE OR REPLACE TRIGGER working_hours_member
  BEFORE
  DELETE OR INSERT OR UPDATE
  ON MEMBER FOR EACH ROW
  BEGIN
    VIDEO_PKG.TIME_CHECK;
  END working_hours_member;
/
```