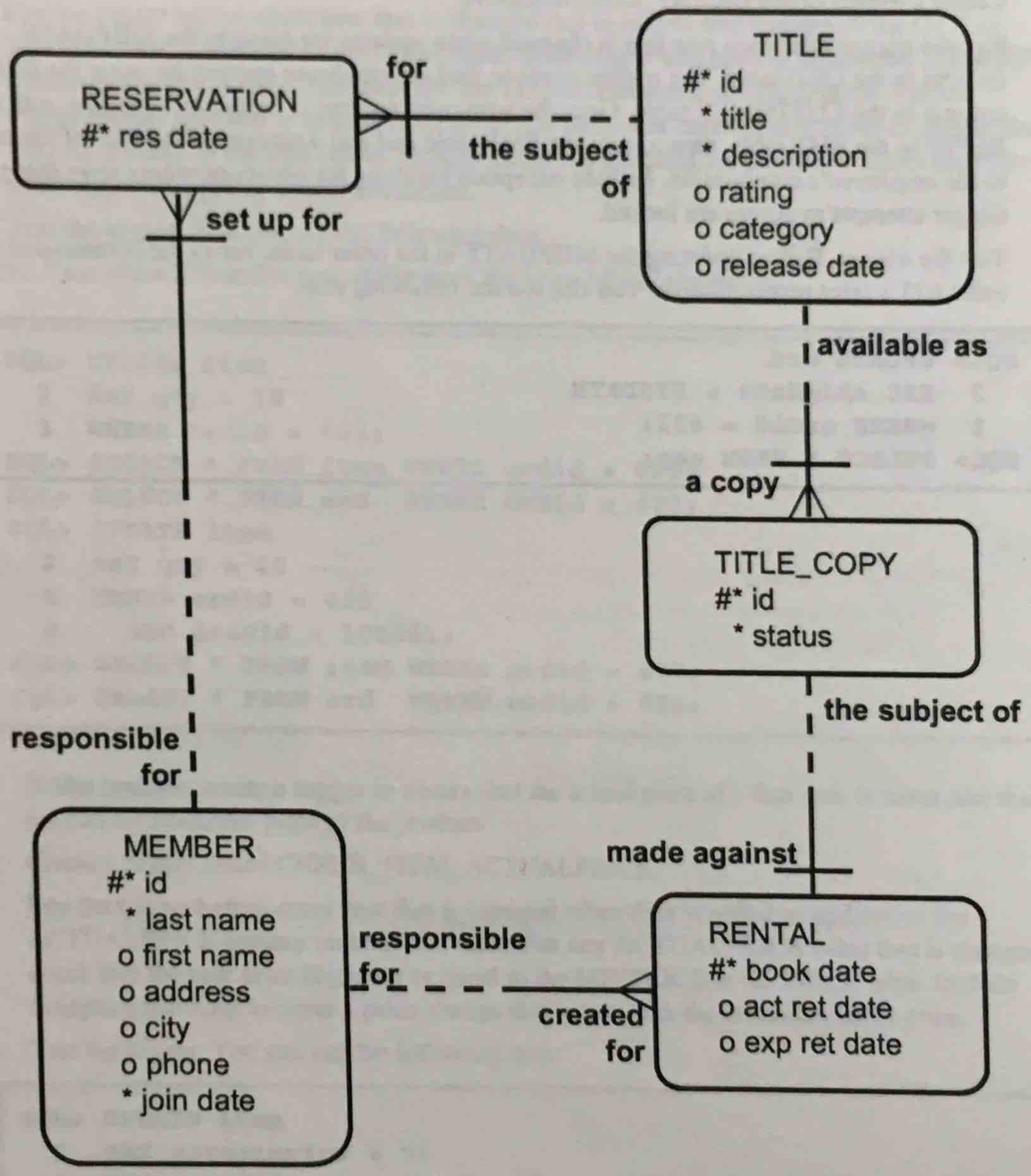
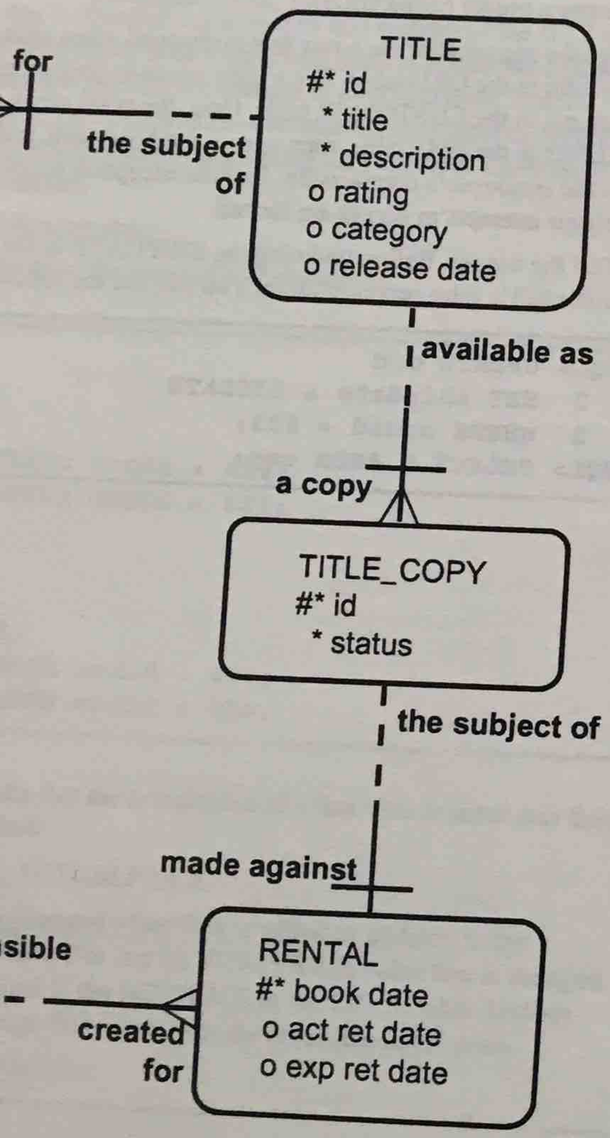
Part B: ENTITY RELATIONSHIP DIAGRAM



DIAGRAM



Part B

In this case study exercise, create a package named VIDEO which contains procedures and functions for a Video Store application. This application allows customers to become a member of the video store. Any members can rent movies, return rented movies and reserve movies. Additionally, you'll create a trigger to ensure that any data in the video tables is modified only during business hours.

Create the package using SQL*Plus and use the 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 diagram is shown on the facing page.

Part B

1. Run the script BUILDVID.SQL to create all of the required tables and sequences needed for this exercise.

2. Create a package named VIDEO with the following procedures and functions:
   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.
   b. 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.
   c. 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 number, the copy id number 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.
   d. RESERVE_MOVIE: a private procedure that executes only if all of 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 out a message indicating that a movie is reserved and its expected date of return.
   e. EXCEPTION_HANDLER: a private procedure that is called from the exception handler of the public programs. Pass to this procedure the SQLCODE number, 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.

Part B

```
SQL> EXEC video.return_movie(98, 1, 'AVAILABLE')
Put this movie on hold -- reserved by member #107

PL/SQL procedure successfully completed.

SQL> EXEC video.return_movie(95, 3, 'AVAILABLE')

PL/SQL procedure successfully completed.

SQL> exec video.return_movie(111, 1, 'RENTED')
begin video.return_movie(111, 1, 'RENTED') end;

*
ERROR at line 1:
ORA-20999: Unhandled error in RETURN_MOVIE.
Please contact your application
administrator with the following information:
ORA-01403: no data found
ORA-06512: at "SCOTT.VIDEO", line 14
ORA-06512: at "SCOTT.VIDEO", line 76
ORA-06512: at line 1
```

Part B

3. The business hours for the video store are 8:00 a.m. to 10:00 p.m. Sunday through Friday, and 8:00 a.m. to 12:00 midnight on Saturday. To ensure that the tables can only be modified during these hours, create a stored procedure that is called by triggers on the tables.

   a. Create a stored procedure called TIME_CHECK which checks the current time against the business hour times. If the current time is not within the business hours, use the RAISE_APPLICATION_ERROR procedure to give an appropriate message.

   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.

   c. Test your trigger.

Note: In order for your trigger to fail, you need to change the time to be outside the range of your current time in class. For example, while testing, you may want valid video hours in your trigger to be from 6:00 p.m. to 8:00 a.m.