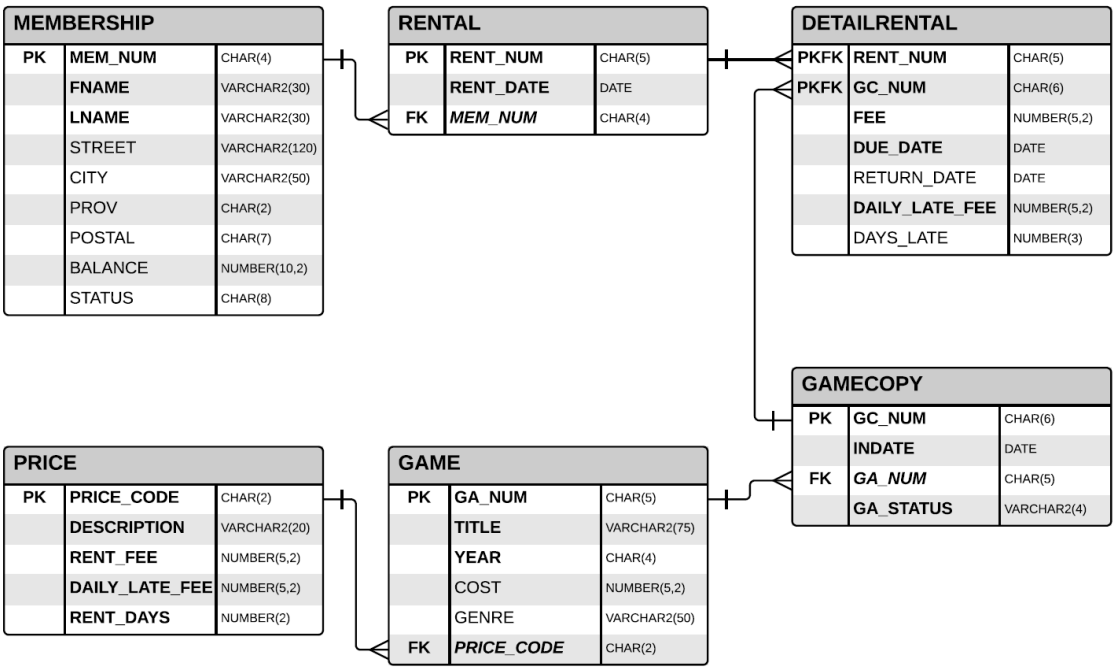
The following problems expand on the GameUR project.



1. Alter the DETAILRENTAL table to include an attribute named DAYS_LATE to store integers up to 4 digits. The attribute should accept null values.

2. Alter the GAMECOPY table to include an attribute named GA_STATUS to store character data up to 4 characters long. The attribute should not accept null values. The attribute should have a constraint to enforce the domain (“IN”, “OUT”, and “LOST”), and have a default value of “IN”.

3. Update the GA_STATUS attribute of the GAMECOPY table using a subquery to set the GA_STATUS to “OUT” for all games that have a null value in the RETURN_DATE attribute of the DETAILRENTAL table.

4. Alter the PRICE table to include an attribute named RENT_DAYS to store integers up to 2 digits. The attribute should not accept null values and should have a default value of 3.

5. Update the PRICE table to place the values shown in the following table in the RENT_DAYS attribute.

PRICE_CODE	RENT_DAYS
1	5
2	3
3	5
4	7

6. Write an SQL statement that will return the game number, title, and number of copies that are inventoried.

```
Game Number TITLE                    Inventory
----------- ---------------------- ----------
1234        BATTLEFIELD 1                   3
1235        MASS Effect Andromeda           2
1236        EverQuest                       4
1237        Fable II                        3
1238        Mario Kart 8 Deluxe             2
1239        Blur                            1
1245        Grand Theft Auto V              2
1246        Halo 4                          2
1247        Wipeout 3                       1

9 rows selected.
```

7. Write an SQL statement that will return the game number, title, and number of copies that are available to be rented.

```
Game Number TITLE                    Inventory
----------- ---------------------- ----------
1234        BATTLEFIELD 1                   2
1235        MASS Effect Andromeda           2
1236        EverQuest                       3
1237        Fable II                        3
1238        Mario Kart 8 Deluxe             2
1239        Blur                            1
1245        Grand Theft Auto V              2
1246        Halo 4                          1
1247        Wipeout 3                       1

9 rows selected.
```

8. Create two sequences named: **SEQ_GAME_COPY** to start with 70000, increment by 1, and not cache any values plus **SEQ_RENT_NUM** to start with 1200, increment by 1, and not cache any values.

PROCEDURES TO HANDLE EACH TABLE ENTRY

(*Raise exceptions but don't handle them yet. Errors are to be thrown back to the main program for error handling.*)

INCLUDE THE ASKED FOR TESTING PROCEDURE CALLS IN YOUR SCRIPT

9. Create a stored procedure named **PRC_ADD_GAMECOPY** to insert new rows into the **GAMECOPY** table.
   a. The game number will be provided as a parameter.
   b. Use a **Count()** function to verify that the game number exists in the **GAME** table. If it does not exist (Count returns a 0), then a message should be displayed stating that the game number does not exist – no data should be written to the database.
   c. If it does exist then insert a new row in the **GAMECOPY** table using the sequence created in *Part 8 (SEQ_GAME_COPY)* to generate the value for **GC_NUM**, the current system date for the value for **IN_DATE**, **GA_NUM** provided as the parameter and set **GA_STATUS** as "IN".
   d. Display the following information to the screen, as verification and what game copy number to apply to the game package.

```
ADDED COPY NUMBER: 70002 for game - Halo 4
```

   e. **Raise** any necessary **Application Errors** – which are to be handled by the main/calling program.

   Execute this procedure, adding a game copy of the game 1246 (Halo 4), to show it is working correctly. Be sure to include the code to execute this procedure in your SQL script.

10. Create a stored procedure named **PRC_MEM_INFO** to display a renter's membership information. The procedure should satisfy the following conditions.
    a. The membership number will be provided as a parameter.
    b. Use a similar method as you used in the procedure from part 9 in **PRC_MEM_INFO** to verify the membership number.
    c. If the membership number does exist then display the last name + first name, street, and postal code.
    d. No data changes should occur with this procedure.
    e. **Raise** any necessary **Application Errors** – which are to be handled by the main/calling program.

Sample Screen: Member Number **111**

```
MANN, STACY
2789 EAST COOK AVENUE    E4C 1C2
```

Sample Screen: Member Number **411**

```
ORA-20011: NO MATCHING MEMBERSHIP_ID FOR 411
```

Execute this procedure, showing the info of member number 111 (Mann, Stacy), to show it is working correctly. Be sure to include the code to execute this procedure in your SQL script.

11. Create a stored procedure named **PRC_NEW_RENTAL** to insert new rows in the **RENTAL** table. The procedure should satisfy the following conditions.
    a. The membership number will be provided as a parameter.
    b. Use the procedure (**PRC_MEM_INFO**) from Part 10 to validate the membership number.
    c. If the membership does exist, then retrieve the membership balance and display a message stating the balance amount as the previous balance. (For example, if the membership has a balance of $5.00, then display "Balance: $5.00".)
    d. Insert a new row in the **RENTAL** table using the sequence created in *Part 8* to generate the value for **RENT_NUM**, the current system date for the value for **RENT_DATE**, and the membership number provided as the value for **MEM_NUM**.
    e. **Raise** any necessary **Application Errors** – which are to be handled by the main/calling program.

Sample Screen: Member Number **111**

```
MANN, STACY
2789 EAST COOK AVENUE    E4C 1C2

Balance:     $8.00
```

```
RENT_ RENT_DATE  MEM_
----- ---------- ----
1001  2017-03-01 103
1002  2016-12-12 105
1003  2016-11-13 102
1004  2017-03-10 110
1005  2017-02-04 111
1006  2017-02-25 107
1007  2016-12-12 104
1008  2017-01-01 105
1009  2016-10-12 111
50075 2017-05-26 111
```

Resultant change in Rental table

Sample Screen: Member Number **411**

```
ERROR at line 1:
ORA-20021: NO MATCHING MEMBERSHIP_ID FOR 411... NO RENTAL INFO STORED
```

Do not test this procedure yet; wait until 12 is completed when you will test it there.

12. Create a stored procedure named **PRC_NEW_DETAIL** to insert new rows in the **DETAILRENTAL** table. The procedure should satisfy the following requirements. Currently, customers can only rent one game at a time.
    a. The game number will be provided as a parameter.
    b. Verify the game number exists in the **GAME** table. If it does not exist, then display a message that the game does not exist, and do not write any data to the database.
    c. If the game number does exist;
        a. Then retrieve a single corresponding **GC_NUM** that has the **GA_STATUS** of "IN"; the indicator that a copy of the game is available for rent.
        b. If no **GC_NUM** can be retrieved, then display a message that no game copies are available for rent at this time – list the expected **DUE_DATE's** of the games out on rental.
    d. If the status is "IN", then retrieve the values of **RENT_FEE**, **DAILY_LATE_FEE**, and **RENT_DAYS** associated with the game from the **PRICE** table.
    e. Calculate the due date for the game rental by adding the number of days found in **RENT_DAYS** above to the current system date - **SYSDATE**.
    f. Insert a new row in the **DETAILRENTAL** table using the previous/current value returned by **SEQ_RENT_NUM** (Part 8) as the **RENT_NUM**, the game number provided in the parameter as the **GC_NUM**, the **RENT_FEE** as the value for **FEE**, the due date calculated above for the **DUE_DATE, DAILY_LATE_FEE** as the value for **DAILY_LATE_FEE**, and **NULL** for the **RETURN_DATE**.
    g. Change the game copy's status to "**OUT**" in the **GAMECOPY** table.
    h. Display the rental details to the screen – example shown below.
    i. **Raise** any necessary **Application Errors** – which are to be handled by the main/calling program.

    *To test this procedure, **PRC_NEW_RENTAL** must be run first.*

    Sample Screen: Game Number **1246**

```
RENTAL NUMBER:  #50076

GAME COPY #61369        TITLE: Halo 4
RENTAL FEE $2.5         LATE FEE $1.5
DUE BACK IN 5 DAYS - 2019-MAY-31
```

    If game 1246 was not available for rent, then you would see something like:

```
NO GAME CURRENTLY AVAILABLE FOR RENT UNTIL: JUN-05-2019     MESSAGE NUMBER -20031
```

    Sample Screen: Game Number **1222**

```
ORA-20032: NO MATCHING GAME ID FOR 1222
```

    Execute this procedure along with PRC_NEW_RENTAL, creating a rental for member number 111 (Mann, Stacy), of the game 1246 (Halo 4). Include this in your SQL script.

13. Call your procedures in an anonymous block called **GAMERENTAL** (file called gamerental.sql). The block should satisfy the following requirements.
    a. Prompts for the two pieces of information: Member number and Game number to be rented. Use DEFINE statements for preset these for testing/grading. See the note below.
    b. The procedures you have created are to be 'called' in the appropriate order using the rental information obtained from the user in part a.
    c. Be sure to verify that changes have been made to the appropriate tables.
    d. Since this is the main/calling program, errors must be trapped, handled, and properly displayed at this level.
    e. Test as shown below.
    f. Be sure to call your anonymous block from your script using the @C:\path\to\file.sql command.

Note: You can set default values for your bind variables when testing using the DEFINE keyword.

Before your anonymous block, use DEFINE statements like the ones below:

```
DEFINE member_num = 111;
DEFINE game_num = 1246;
```

Simply comment out these lines if you want to test with different values.

```
SQL> @gamerental
Enter value for member_num: 111
Enter value for game_num: 1246

        MANN, STACY
        2789 EAST COOK AVENUE   E4C 1C2

        Balance: $8.00


        RENTAL NUMBER: #50098

        GAME COPY #61369       TITLE: Halo 4
        RENTAL FEE $2.5        LATE FEE $1.5
        DUE BACK IN 5 DAYS - 2019-JUN-05

SQL>
SQL> @gamerental
Enter value for member_num: 411
Enter value for game_num: 1246

SOMETHING IS WRONG WITH YOUR REQUEST

PLEASE READ MESSAGE BELOW FOR EXPLANATION.

        NO MATCHING MEMBERSHIP ID FOR 411           MESSAGE NUMBER -20011
SQL> @gamerental
Enter value for member_num: 111
Enter value for game_num: 1222

        MANN, STACY
        2789 EAST COOK AVENUE   E4C 1C2

        Balance: $8.00

SOMETHING IS WRONG WITH YOUR REQUEST

PLEASE READ MESSAGE BELOW FOR EXPLANATION.

         NO MATCHING GAME ID FOR 1222           MESSAGE NUMBER -20032
SQL>
```

14. Rework the stored procedure named **PRC_NEW_DETAIL** (see part 12) into **PRC_NEW_DETAIL_V2** so as to meet new operational requirements set by **GameUR** management - the following are the new requirements.
    a. Management has decided that instead of one game rental per member that members may rent up to 3 games at a time.
    b. Alter your procedure, such that it can accept between 1 and 3 game numbers as parameters.
    c. Basic coding of your procedure remains the same – you just need to code for the using the same code more than once which was the point of part 12. Reuse of existing code should be your focus.
    d. No error should be raised, if at least one of the games supplied to the procedure is available to rent. If a game copy is not available for rental a simple message should appear stating such and when a copy should be available for rental.
    e. Add the feature that the total cost of the rental will be listed on the screen and that amount will be added to the member's balance in the **MEMBERSHIP** table.
    f. Due to constraints you will need to run the procedure **PRC_NEW_RENTAL** before testing this procedure.
    g. Rework part 13 - **GAMERENTAL**, so more than 1 game number is prompted for. Save this as gamerental2.sql.

    Call your **GAMERENTAL2** code block from your script with the values shown below, using DEFINE to preset the values:

```
SQL> @gamerental2
Enter value for member_num: 111
Enter value for game_num: 1246
Enter value for game_num: 1239
Enter value for game_num:

        MANN, STACY
        2789 EAST COOK AVENUE   E4C 1C2

        Balance: $8.00

        RENTAL NUMBER:  #50102

        GAME COPY #61369        TITLE: Halo 4
        RENTAL FEE $2.5         LATE FEE $1.5
        DUE BACK IN 5 DAYS - 2019-JUN-05


        RENTAL NUMBER:  #50102

        GAME COPY #61388        TITLE: Blur
        RENTAL FEE $2.5         LATE FEE $1.5
        DUE BACK IN 5 DAYS - 2019-JUN-05

        TOTAL RENTAL FEE: $5.00
        **************************

        NEW BALANCE: $13.00
SQL>
```

Run your **GAMERENTAL2** code block again from your script with values shown below, using DEFINE statements to set new values, to test a game not being available:

```
SQL> @gamerental2
Enter value for member_num: 111
Enter value for game_num: 1246
Enter value for game_num: 1239
Enter value for game_num:

        MANN, STACY
        2789 EAST COOK AVENUE   E4C 1C2

        Balance: $13.00

        RENTAL NUMBER:  #50114

        GAME COPY #61353        TITLE: Grant Theft Auto V
        RENTAL FEE $2.5         LATE FEE $1.5
        DUE BACK IN 5 DAYS - 2019-JUN-05


        THE GAME Blur IS CURRENTLY NOT AVAILABLE FOR RENT UNTIL: JUN-05-2019

        TOTAL RENTAL FEE: $5.00
        ***************************

        NEW BALANCE: $13.00
SQL>
```

15. Create a stored procedure named **PRC_RETURN_GAME** enter data about the return of games that had been rented. The procedure should satisfy the following requirements.
    a. The game copy number will be provided as a parameter.
    b. Verify the game copy number exists in the **DETAILRENTAL** table. If it does not exist, display a message that the game copy number provided was not found and do not write any data to the database.
    c. If the game copy number does exist, then use a **Count()** function to ensure that the game has only one record in **DETAILRENTAL** for which it does not have a return date. If more than one row in **DETAILRENTAL** indicates that the game is rented but not returned, display an error message that the game has multiple outstanding rentals and do not write any data to the database.
    d. If the game copy has only one outstanding rental, then update the **RETURN_DATE** to the current system date and update the game copy status to "**IN**" for that game copy in the **GAMECOPY** table.
    e. Then display a message stating that the game was successfully returned.
    f. **Raise** any necessary **Application Errors** – which are to be handled by the main/calling program.

    Execute this procedure returning the game copy 61367.

    ```
    GAME SUCCESSFULLY RETURNED
    ```

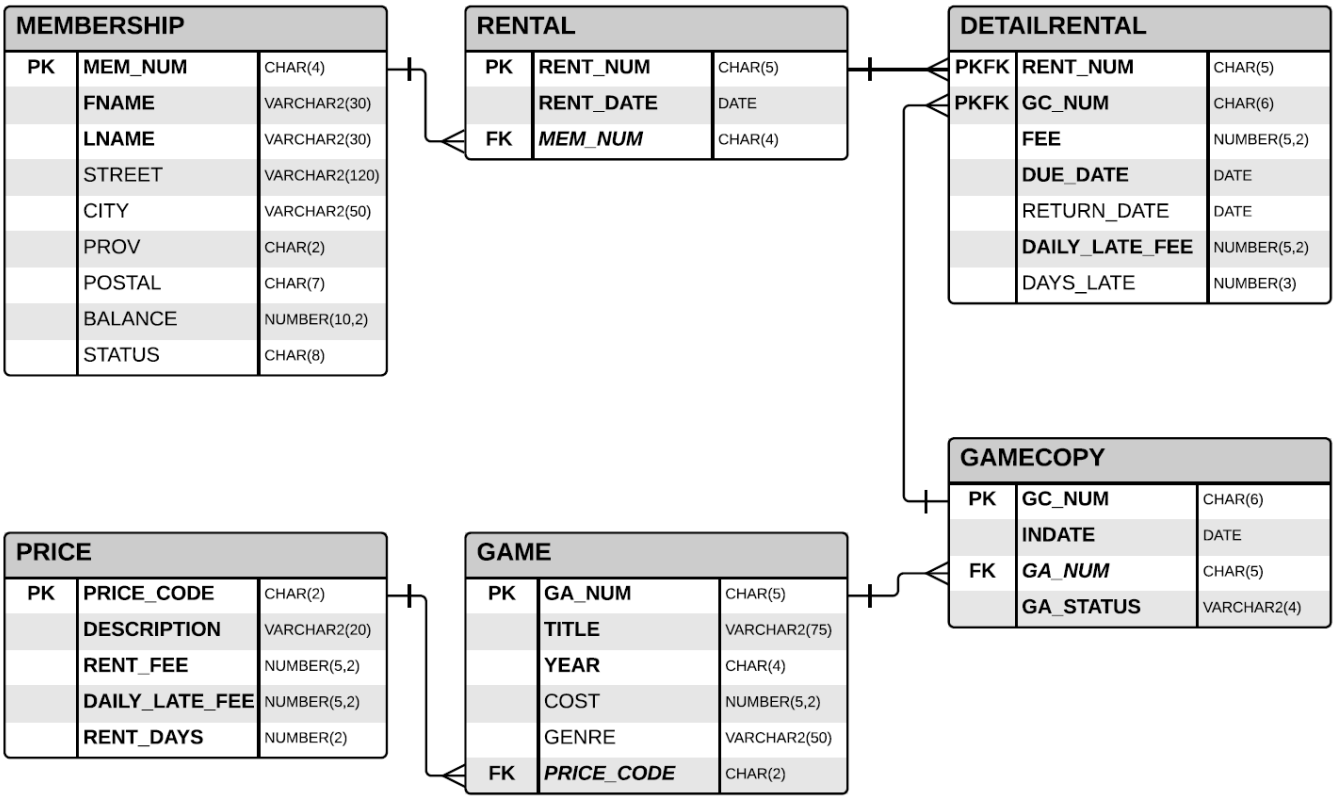
    Execute this procedure returning the game copy 61367.

    ```
    ORA-20044: NO MATCHING GAME COPY FOR 61377 ... NO RENTAL RETURN POSSIBLE
    ```

Create the following *Trigger* to handle events on table entry

16. Create a trigger named **TRG_LATE_RETURN** that will write the correct value to **DAYS_LATE** in the **DETAILRENTAL** table whenever a game is returned. The trigger should execute as a **BEFORE** trigger when the **RETURN_DATE** or **DUE_DATE** attributes are updated. The trigger should satisfy the following conditions.
    a. If the return date is null, then the days late should be null also.
    b. If the return date is the day of the due date or earlier, then the game is not considered late, and the **DAYS_LATE** field should have a value of zero (0).
    c. If the return date is later then the due date then the game is considered late. The number of days late should be calculated and stored in the **DAYS_LATE** field.
    d. Should the **DUE_DATE** be changed (for whatever reason, either prior to or after a return) then the number of days late will need to be recalculated and stored in the **DAYS_LATE** field.

    Create SQL statements to test and validate all these trigger conditions. Include these statements in your SQL script.


MEMBERSHIP
PK MEM_NUM CHAR(4)
FNAME VARCHAR2(30)
LNAME VARCHAR2(30)
STREET VARCHAR2(120)
CITY VARCHAR2(50)
PROV CHAR(2)
POSTAL CHAR(7)
BALANCE NUMBER(10,2)
STATUS CHAR(8)
RENTAL
PK RENT_NUM CHAR(5)
RENT_DATE DATE
FK MEM_NUM CHAR(4)
DETAILRENTAL
PKFK RENT_NUM CHAR(5)
PKFK GC_NUM CHAR(6)
FEE NUMBER(5,2)
DUE_DATE DATE
RETURN_DATE DATE
DAILY_LATE_FEE NUMBER(5,2)
DAYS_LATE NUMBER(3)
GAMECOPY
PK GC_NUM CHAR(6)
INDATE DATE
FK GA_NUM CHAR(5)
GA_STATUS VARCHAR2(4)
PRICE
PK PRICE_CODE CHAR(2)
DESCRIPTION VARCHAR2(20)
RENT_FEE NUMBER(5,2)
DAILY_LATE_FEE NUMBER(5,2)
RENT_DAYS NUMBER(2)
GAME
PK GA_NUM CHAR(5)
TITLE VARCHAR2(75)
YEAR CHAR(4)
COST NUMBER(5,2)
GENRE VARCHAR2(50)
FK PRICE_CODE CHAR(2)