**ISS4014 – Database Systems and Web Integration**

**Chapter 08 – Team Case**

Read the introduction of the case on EliteVideo in chapter 8, starting on page 433. Read the instructions here first and then see the textbook for additional information. Some instructions have been modified to account for the use of MySQL DBMS. Rather than copy SQL statements and images of them running below each problem step, compile the solution to each step into a single SQL file and label each solution with an SQL comment line. See below for an example.

/\* EliteVideo Solution SQL statements.

Team Members: (list team members here)

Team Report: (list each team member, what they did, and how much time each team member contributed to the solutions)

\*/

-- problem 47

CREATE TABLE PRICE (

PRICE\_CODE INTEGER PRIMARY KEY,

PRICE\_DESCRIPTION VARCHAR(20) NOT NULL ,

PRICE\_RENTFEE DECIMAL(5,2),

PRICE\_DAILYLATEFEE DECIMAL(5,2),

);

CREATE TABLE MOVIE (

MOVIE\_NUM INTEGER PRIMARY KEY,

MOVIE\_TITLE VARCHAR(75) NOT NULL,

…

-- problem 48

-- PRICE

INSERT INTO PRICE VALUES (1, 'Standard', 2, 1);

INSERT INTO PRICE VALUES (2, 'New Release', 3.5, 3);

INSERT INTO PRICE VALUES (3, 'Discount', 1.5, 1);

INSERT INTO PRICE VALUES (4, 'Weekly Special', 1, .5);

-- MOVIE

INSERT INTO MOVIE VALUES (1234, 'The Cesar Family Christmas', 2016, 39.95, 'FAMILY', 2);

INSERT INTO MOVIE VALUES (1235, 'Smokey Mountain Wildlife', 2013, 59.95, 'ACTION', 1);

…

1. Write the SQL code to create the table structures for the entities shown in Figure P8.47. The structures should contain the attributes specified in the ERD. Use data types that are appropriate for the data that will need to be stored in each attribute. Enforce primary key and foreign key constraints as indicated by the ERD. **The solution should include six create table statements.**
2. The following tables provide a very small portion of the data that will be kept in the database. The data needs to be inserted into the database for testing purposes. Write the INSERT commands necessary to place the following data in the tables created in Problem 47. (If required by your DBMS, be certain to save the rows permanently.). **Option: Create a single valid insert statement for each of the six tables using the first line from each table and email it to the instructor. The instructor will then email you back a complete list of all insert statements.**
3. Write the SQL command to change the movie year for movie number 1245 to 2014.
4. Write the SQL command to change the price code for all action movies to price code 3.
5. Write a single SQL command to increase all price rental fee values in the PRICE table by $0.50.
6. Alter the DETAILRENTAL table to include a derived attribute named DETAIL\_DAYSLATE to store integers of up to three digits. The attribute should accept null values.
7. Update the DETAILRENTAL table to set the values in DETAIL\_RETURNDATE to include a time component. Make each entry match the values shown in the following table. **The solution will include an ALTER statement and several UPDATE statements.**
8. Alter the VIDEO table to include an attribute named VID\_STATUS to store character data up to four characters long. The attribute should have a constraint to enforce the domain (“IN,” “OUT,” and “LOST”) and have a default value of “IN.”
9. Update the VID\_STATUS attribute of the VIDEO table using a subquery to set the VID\_STATUS to “OUT” for all videos that have a null value in the DETAIL\_RETURNDATE attribute of the DETAILRENTAL table.
10. Alter the PRICE table to include an attribute named PRICE\_RENTDAYS to store integers of up to two digits. The attribute should not accept null values, and it should have a default value of 3.
11. Update the PRICE table to place the values shown in the following table in the PRICE\_RENTDAYS attribute. **The solution will include multiple UPDATE statements.**
12. Create a trigger named trg\_late\_return that will write the correct value to DETAIL\_DAYSLATE in the DETAILRENTAL table whenever a video is returned. The trigger should execute as a BEFORE trigger when the DETAIL\_RETURNDATE or DETAIL\_DUEDATE attributes are updated. The trigger should satisfy the following conditions:

* If the return date is null, then the days late should also be null.
* If the return date is not null, then the days late should determine if the video is returned late.
* If the return date is noon of the day after the due date or earlier, then the video is not considered late, and the days late should have a value of zero (0).
* If the return date is past noon of the day after the due date, then the video is considered late, so the number of days late must be calculated and stored.

**Rerun the UPDATE statements from problem 53 with a SELECT \* FROM DETAILRENTAL statement before and after the UPDATEs to ensure the trigger is functioning correctly. You should see the days late column updated with the correct values.**

1. Create a trigger named trg\_mem\_balance that will maintain the correct value in the membership balance in the MEMBERSHIP table when videos are returned late. The trigger should execute as an AFTER trigger when the due date or return date attributes are updated in the DETAILRENTAL table. The trigger should satisfy the following conditions:

* Calculate the value of the late fee before the update that triggered this execution of the trigger. The value of the late fee is the days late multiplied by the daily late fee. If the previous value of the late fee was null, then treat it as zero (0).
* Calculate the value of the late fee after the update that triggered this execution of the trigger. If the value of the late fee is now null, then treat it as zero (0).
* Subtract the prior value of the late fee from the current value of the late fee to determine the change in late fee for this video rental.
* If the amount calculated in Part c is not zero (0), then update the membership balance by the amount calculated for the membership associated with this rental.

1. **Alter the RENTAL table so the rental number primary key is set to auto increment and the current increment value is set to 1100.**
2. Create a stored procedure named prc\_new\_rental to insert new rows in the RENTAL table. The procedure should satisfy the following conditions:

* The membership number will be provided as a parameter.
* Use a Count() function to verify that the membership number exists in the MEMBERSHIP table. If it does not exist, then a message should be displayed that the membership does not exist and no data should be written to the database.
* If the membership does exist, then retrieve the membership balance and display a message that the balance amount is the previous balance. (E.g., if the membership has a balance of $5.00, then display “Previous balance: $5.00”.)
* Insert a new row in the rental table using the **auto\_increment** created above to generate the value for RENT\_NUM, the current system date for the RENT\_DATE value, and the membership number provided as the value for MEM\_NUM.

1. Create a stored procedure named prc\_new\_detail to insert new rows in the DETAILRENTAL table. The procedure should satisfy the following requirements:

* The video number will be provided as a parameter.
* Verify that the video number exists in the VIDEO table. If it does not exist, then display a message that the video does not exist, and do not write any data to the database.
* If the video number does exist, then verify that the VID\_STATUS for the video is “IN.” If the status is not “IN,” then display a message that the video’s return must be entered before it can be rented again, and do not write any data to the database.
* If the status is “IN”, then retrieve the values of the video’s PRICE\_RENTFEE, PRICE\_DAILYLATEFEE, and PRICE\_RENTDAYS from the PRICE table.
* Calculate the due date for the video rental by adding the number of days in PRICE\_RENTDAYS to 11:59:59PM (hours:minutes:seconds) in the current system date.
* Insert a new row in the DETAILRENTAL table using the previous value returned by RENT\_NUM\_SEQ as the RENT\_NUM, the video number provided in the parameter as the VID\_NUM, the PRICE\_RENTFEE as the value for DETAIL\_FEE, the due date calculated above for the DETAIL\_DUEDATE, PRICE\_DAILYLATEFEE as the value for DETAIL\_DAILYLATEFEE, and null for the DETAIL\_RETURNDATE.

1. Create a stored procedure named prc\_return\_video to enter data about the return of videos that have been rented. The procedure should satisfy the following requirements.

* The video number will be provided as a parameter.
* Verify that the video number exists in the VIDEO table. If it does not exist, display a message that the video number provided was not found and do not write any data to the database.
* If the video number does exist, then use a Count() function to ensure that the video 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 video is rented but not returned, display an error message that the video has multiple outstanding rentals and do not write any data to the database.
* If the video does not have any outstanding rentals, then update the video status to “IN” for the video in the VIDEO table, and display a message that the video had no outstanding rentals but is now available for rental. If the video has only one outstanding rental, then update the return date to the current system date, and update the video status to “IN” for that video in the VIDEO table. Then display a message that the video was successfully returned.