**Assignment 3a CSC 401**

**Name: Varunan Asokan**

**Date:**

1. Create two stored PL/SQL procedures that process movie rental and movie return, based on the movie id and member id. More specifically:

* Procedure MOVIE\_RENT\_SP takes in three parameters: the movie id and the member id, and a payment method. It adds a new record to the MM\_RENTAL table. It should also update the movie inventory, which is the MOVIE\_QTY column of the MM\_MOVIE table.

SQL> create sequence rental\_id\_seq start with 13;

Sequence created.

**Note: This sequence will enable the auto-increment of rental\_id column. Since the value of last record for the rental\_id value is 12, so the sequence starts with 13 and will be incremented by 1 for each subsequent record that is inserted.**

Create or replace procedure MOVIE\_RENT\_SP(movieid number, memberid number, paymentmethod number) as

 FOREIGN\_KEY\_VIOLATION EXCEPTION;

 PRAGMA EXCEPTION\_INIT(FOREIGN\_KEY\_VIOLATION, -2291);

begin

 savepoint start\_transaction;

 insert into MM\_RENTAL(rental\_id,member\_id,movie\_id,checkout\_date,payment\_methods\_id) values(rental\_id\_seq.nextval,memberid,movieid,sysdate,paymentmethod);

 update MM\_MOVIE set movie\_qty=movie\_qty-1 where movie\_id=movieid;

 commit;

Exception

When FOREIGN\_KEY\_VIOLATION then

 dbms\_output.put\_line('Foreign Key Violation. Value not found in Parent table');

 rollback to start\_transaction;

When INVALID\_NUMBER Then

 dbms\_output.put\_line('Please enter only numbers');

 rollback to start\_transaction;

When ROWTYPE\_MISMATCH then

 dbms\_output.put\_line(' Invalid data type or precision');

 rollback to start\_transaction;

When VALUE\_ERROR then

 dbms\_output.put\_line('Error with data type');

When OTHERS Then

 dbms\_output.put\_line('An error has occurred due to invalid data');

 rollback to start\_transaction;

end;

**Records in the MM\_RENTAL table BEFORE the stored procedure is executed**

SQL> select \* from mm\_rental;

RENTAL\_ID MEMBER\_ID MOVIE\_ID CHECKOUT\_D CHECKIN\_D PAYMENT\_METHODS\_ID

 ---------------------------------------------------------------------------------------------------------------------------

 1 10 11 04-JUN-03 2

 2 10 8 04-JUN-03 2

 3 12 6 04-JUN-03 2

 4 13 3 04-JUN-03 5

 5 13 5 04-JUN-03 5

 6 13 11 04-JUN-03 5

 7 14 10 04-JUN-03 2

 8 14 7 04-JUN-03 2

 9 12 4 04-JUN-03 4

 10 12 12 04-JUN-03 4

 11 12 3 04-JUN-03 4

 12 13 4 04-JUN-03 5

12 rows selected.

**Records in the MM\_MOVIE table BEFORE the stored procedure is executed**

SQL> select \* from mm\_movie;

 MOVIE\_ID MOVIE\_TITLE MOVIE\_CAT\_ID MOVIE\_VALUE MOVIE\_QTY S

---------------------------------------------------------------------------------------------------------------------

 1 Alien 1 10 5

 2 Bladerunner 1 8 3

 3 Star Wars 1 15 11

 4 Texas Chainsaw Masacre 2 7 2

 5 Jaws 2 7 1

 6 The good, the bad and the ugly 3 7 2

 7 Silverado 3 7 1

 8 Duck Soup 4 5 1

 9 Planes, trains and automobiles 4 5 3

 10 Waking Ned Devine 4 12 4

 11 Deep Blue Sea 5 14 3

 12 The Fifth Element 5 15 5

12 rows selected.

**Executing the stored Procedure**

SQL> exec MOVIE\_RENT\_SP(10,12,4);

PL/SQL procedure successfully completed.

**Records in the MM\_RENTAL table AFTER the stored procedure is executed**

SQL> select \* from mm\_rental;

RENTAL\_ID MEMBER\_ID MOVIE\_ID CHECKOUT\_D CHECKIN\_D PAYMENT\_METHODS\_ID

 ---------------------------------------------------------------------------------------------------------------------------

 1 10 11 04-JUN-03 2

 2 10 8 04-JUN-03 2

 3 12 6 04-JUN-03 2

 4 13 3 04-JUN-03 5

 5 13 5 04-JUN-03 5

 6 13 11 04-JUN-03 5

 7 14 10 04-JUN-03 2

 8 14 7 04-JUN-03 2

 9 12 4 04-JUN-03 4

 10 12 12 04-JUN-03 4

 11 12 3 04-JUN-03 4

 12 13 4 04-JUN-03 5

 **13 12 10 05-JUN-09 4**

13 rows selected.

**Records in the MM\_MOVIE table AFTER the stored procedure is executed**

SQL> select \* from mm\_movie;

 MOVIE\_ID MOVIE\_TITLE MOVIE\_CAT\_ID MOVIE\_VALUE MOVIE\_QTY S

---------------------------------------------------------------------------------------------------------------------

 1 Alien 1 10 5

 2 Bladerunner 1 8 3

 3 Star Wars 1 15 11

 4 Texas Chainsaw Masacre 2 7 2

 5 Jaws 2 7 1

 6 The good, the bad and the ugly 3 7 2

 7 Silverado 3 7 1

 8 Duck Soup 4 5 1

 9 Planes, trains and automobiles 4 5 3

 **10 Waking Ned Devine 4 12 3**

 11 Deep Blue Sea 5 14 3

 12 The Fifth Element 5 15 5

12 rows selected.

**Note: The records marked in bold indicate that the stored procedure has worked properly. While in the mm\_rental table, a new record is inserted. In case of mm\_movie table the table is updated and the available movie quantity is decremented by 1.**

* Procedure MOVIE\_RETURN\_SP takes in two parameters: the movie id and the member id. Based on these two values, it identifies the rental record in the MM\_RENTAL table and records the current date in the CHECKIN\_DATE column. It also needs to update the movie inventory in the MM\_MOVIE table.

Create or replace procedure MOVIE\_RETURN\_SP(movieid number, memberid number) as

movid mm\_rental.movie\_id%type;

memid mm\_rental.member\_id%type;

begin

 savepoint start\_transaction;

 select movie\_id,member\_id into movid, memid from mm\_rental where movie\_id=movieid;

 update MM\_RENTAL set checkin\_date=sysdate where member\_id=memberid and movie\_id=movieid;

 update MM\_MOVIE set movie\_qty=movie\_qty+1 where movie\_id=movieid;

commit;

Exception

When NO\_DATA\_FOUND then

 dbms\_output.put\_line('Memberid or movie id not found');

 rollback to start\_transaction;

When INVALID\_NUMBER Then

 dbms\_output.put\_line('Please enter only numbers');

 rollback to start\_transaction;

When ROWTYPE\_MISMATCH then

 dbms\_output.put\_line(' Invalid data type or precision');

 rollback to start\_transaction;

When VALUE\_ERROR then

 dbms\_output.put\_line('Error with data type');

 rollback to start\_transaction;

When OTHERS Then

 dbms\_output.put\_line('An error has occurred due to invalid data');

 rollback to start\_transaction;

end;

**Executing Stored Procedure**

SQL> exec movie\_return\_sp(10,12);

PL/SQL procedure successfully completed.

**Records in the MM\_RENTAL table AFTER the stored procedure is executed**

SQL> select \* from mm\_rental;

RENTAL\_ID MEMBER\_ID MOVIE\_ID CHECKOUT\_D CHECKIN\_D PAYMENT\_METHODS\_ID

 ---------------------------------------------------------------------------------------------------------------------------

 1 10 11 04-JUN-03 2

 2 10 8 04-JUN-03 2

 3 12 6 04-JUN-03 2

 4 13 3 04-JUN-03 5

 5 13 5 04-JUN-03 5

 6 13 11 04-JUN-03 5

 7 14 10 04-JUN-03 2

 8 14 7 04-JUN-03 2

 9 12 4 04-JUN-03 4

 10 12 12 04-JUN-03 4

 11 12 3 04-JUN-03 4

 12 13 4 04-JUN-03 5

 **13 12 10 05-JUN-09 05-JUN-09 4**

13 rows selected.

**Records in the MM\_MOVIE table AFTER the stored procedure is executed**

SQL> select \* from mm\_movie;

 MOVIE\_ID MOVIE\_TITLE MOVIE\_CAT\_ID MOVIE\_VALUE MOVIE\_QTY S

---------------------------------------------------------------------------------------------------------------------

 1 Alien 1 10 5

 2 Bladerunner 1 8 3

 3 Star Wars 1 15 11

 4 Texas Chainsaw Masacre 2 7 2

 5 Jaws 2 7 1

 6 The good, the bad and the ugly 3 7 2

 7 Silverado 3 7 1

 8 Duck Soup 4 5 1

 9 Planes, trains and automobiles 4 5 3

 **10 Waking Ned Devine 4 12 4**

 11 Deep Blue Sea 5 14 3

 12 The Fifth Element 5 15 5

12 rows selected.

Note: Both tables have been updated. The check-in date has been inserted in the mm\_rental table while the movie quantity has been updated in the mm\_movie table.

* Make sure you validate the parameters: The movie id and member id must indicate an existing movie and member, respectively, and the payment method must be one of the valid payment method codes.

**Run tests that include various cases of calling these procedures with valid and invalid parameter values to prove that your code identifies the case and reacts properly.**

**Test 1: Checking for foreign key violations**

SQL> exec movie\_rent\_sp(13,12,2);

Foreign Key Violation. Value not found in Parent table

PL/SQL procedure successfully completed.

Note: In this case 13 is an invalid movie id as there are only movies in the table from 1 to 12. This exception is raised due to the following two lines in the procedure code:

FOREIGN\_KEY\_VIOLATION EXCEPTION;

 PRAGMA EXCEPTION\_INIT(FOREIGN\_KEY\_VIOLATION, -2291);

The error number 2291 is an Oracle system defined error number indicating foreign key violation. This is mapped to an user-defined exception namely FOREIGN\_KEY\_VIOLATION using the Exception\_Init() directive.

**Test 2: Invalid precision data.**

SQL> exec movie\_rent\_sp(12,12,123);

An error has occurred due to invalid data

PL/SQL procedure successfully completed.

In this case they Payment method value given in the procedure is a 3-digit number, while the payment\_method\_type field in the table is of 2-digit precision.

**Test 3: Checking for data not found (For the second stored procedure)**

SQL> exec movie\_return\_sp(123,123);

Memberid or movie id not found

PL/SQL procedure successfully completed.

This is for the second stored procedure. If user enters a movie id or member id that is not found then this error message is displayed.2. Write a function that retrieves the movie stock information and formats it in a friendly message to display for user requests. The display should resemble the following: "Star Wars is available: 11 on the shelf".

More specifically, create a function named MOVIE\_STOCK\_SF that takes in a movie id as a parameter. It should retrieve from the MM\_MOVIE table the movie title and quantity information, build the output string and return it.

Again, make sure that your function behaves properly when given a non-existing movie id. Also, make sure that if a movie exists, but there are no copies available, you display a message like: "Star Wars is currently not available" rather than "Star Wars is available: 0 on the shelf".

Create or replace function MOVIE\_STOCK\_SF(movieid number) return varchar is

begin

declare

details\_string varchar2(50);

title mm\_movie.movie\_title%type;

quantity mm\_movie.movie\_qty%type;

begin

select movie\_title,movie\_qty into title, quantity from mm\_movie where movie\_id=movieid;

if (quantity>0) then

 details\_string:=title || ' is available : ' ||quantity || ' on the shelf';

else

 details\_string:=title || ' is currently not available';

end if;

return details\_string;

Exception

When NO\_DATA\_FOUND then

 details\_string:='Movie id not found';

 return details\_string;

When INVALID\_NUMBER Then

 details\_string:='Please enter only numbers';

 return details\_string;

When ROWTYPE\_MISMATCH then

 details\_string:=' Invalid data type or precision';

 return details\_string;

When VALUE\_ERROR then

 details\_string:='Error with data type';

 return details\_string;

When OTHERS Then

 details\_string:='An error has occurred due to invalid data';

 return details\_string;

end;

end;

**Run tests with function calls that cover all possible scenarios including existing and non-existing movie ids, and available and unavailable movies. You probably will need to change some data in your tables to create a particular scenario.**

**Executing the Function:**

**Scenario 1:**

In order to execute the function, I have created a small test block a named Test.sql from where the procedure is called:

declare

begin

dbms\_output.put\_line(MOVIE\_STOCK\_SF(121));

end;

Output:

SQL> @Test

 5 /

Movie id not found

**Scenario 2:**

declare

begin

dbms\_output.put\_line(MOVIE\_STOCK\_SF(12));

end;

SQL> @Test

 5 /

The Fifth Element is available : 5 on the shelf

PL/SQL procedure successfully completed.

**Scenario 3:**

declare

begin

dbms\_output.put\_line(MOVIE\_STOCK\_SF(5));

end;

However for this scenario to work, I have updated the table so that record with movie\_id as 5 has quantity as 0.

SQL> update mm\_movie set movie\_qty=0 where movie\_id=5;

1 row updated.

SQL> @Test

 5 /

Jaws is currently not available