

ISYS224: DATABASE SYSTEMS (2018)

Department of Computing

Macquarie University

Submitted by: Chun Hei Matthew Lee

Student ID: 45282188

Date Submitted :

Declaration:

In submitting this report, I ,Chun Hei Matthew Lee, declare that this is my own work.

Task 1 CREATING AND POPULATING DATABASE TABLES

no errors

Task 2 PROCEDURE FOR LOAN REPAYMENT

(a) Procedure Implementation

```
delimiter //
drop procedure if exists Repay_Loan //
create procedure Repay_Loan(in from_BSB varchar(10), in from_account varchar(10), in to_loan varchar(10), in amount double)
begin
    Declare no_row_found int default 0; #error handling variable for no row found

    Declare saving_amount double; #variable to store how much you have in your from_account

    #find the corresponding loanBSB and accountNo attached by the loanID
    Declare loan_BSB varchar(10);
    Declare loan_Acc varchar(10);
    Declare loan_left double;

    Declare commonOwner int default 0; #store the number of common owner of both accounts

    declare continue handler for not found set no_row_found = 1;
    | select AccountBal into saving_amount #select the balance from the saving account
    from T_Account
    where BSB=from_BSB and AccountNo = from_account;

    #select the corresponding loanAcct for loanID
    select BSB, AccountNo, AccountBal into loan_BSB, loan_Acc, loan_left
    from T_Loan, T_Account
    where (Loan_AccountBSB=BSB and Loan_AcctNo=AccountNo) and
    to_loan = LoanID;

    if(no_row_found) then
        signal sqlstate '45000' set message_text='accounts not found';
    end if;

    #this query finds the number of CustomerID which is the owner of both accounts-loan account and from_acc
    select count(Customer_ID) into commonOwner
    from T_Own
    where Account_BSB=from_BSB and
    Account_No=from_account and
    (Customer_ID in (select Customer_ID
    from T_Own
    where Account_BSB=loan_BSB
    and Account_No = loan_Acc));

    if(commonOwner=0) then #raise an error if no common Owners found
        signal sqlstate '45000' set message_text = 'the owner of account and owner of loan account are not matched';
    elseif (amount > saving_amount) then
        signal sqlstate '45000' set message_text = 'not enough balance to transfer from account';
    else

        update T_Account
        set AccountBal = AccountBal+amount
        where BSB=loan_BSB and AccountNo=loan_Acc;

        update T_Account
        set AccountBal = saving_amount-amount
        where BSB=from_BSB and AccountNo=from_account;

        INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
        VALUES (to_loan, amount, NOW());

    end if;
end //
delimiter ;
```

(b) Procedure Testing

```
67 ● call Repay_Loan('BSB1','Acct1','L3','5.00'); #customerID are different
68 ● call Repay_Loan('BSB1','Acct1','L2','4.00'); #successful transaction
69 ● call Repay_Loan('BSB1','Acct1','L2','2000.00'); #amount is more than savingBalance
```

Result output

Action Output				
	Time	Action	Response	Duration / Fetch Time
✖ 1	08:46:37	call Repay_Loan('BSB1','Acct1','L3','5.00')	Error Code: 1644, the owner of account and owner of...	0.055 sec
✔ 2	08:46:40	call Repay_Loan('BSB1','Acct1','L2','4.00')	1 row(s) affected	0.191 sec
✖ 3	08:46:47	call Repay_Loan('BSB1','Acct1','L2','2000.00')	Error Code: 1644, not enough balance to transfer fro...	0.064 sec

Result Grid				
BSB	AccountNo	AccountBal	AccountType	
BSB1	Acct1	10	AT1	
BSB1	Acct5	10	AT1	
BSB1	Acct6	10	AT1	
BSB2	Acct2	11	AT3	
BSB3	Acct3	-5000	AT3	
BSB3	Acct4	-7000	AT3	
NULL	NULL	NULL	NULL	

T_Account table output before Successful transaction

Result Grid				
RepaymentNo	Repayment_LoanID	RepaymentAmount	RepaymentDate	
1	L1	1	2017-10-10 00:00:00	
2	L2	2	2018-02-11 00:00:00	
3	L3	2	2018-02-11 00:00:00	
NULL	NULL	NULL	NULL	

T_repayment table output before successful transaction

Result Grid				
RepaymentNo	Repayment_LoanID	RepaymentAmount	RepaymentDate	
1	L1	1	2017-10-10 00:00:00	
2	L2	2	2018-02-11 00:00:00	
3	L3	2	2018-02-11 00:00:00	
4	L2	4	2018-10-30 08:47:13	
NULL	NULL	NULL	NULL	

T_Repayment table out after successful transaction

Result Grid				
BSB	AccountNo	AccountBal	AccountType	
BSB1	Acct1	6	AT1	
BSB1	Acct5	10	AT1	
BSB1	Acct6	10	AT1	
BSB2	Acct2	15	AT3	
BSB3	Acct3	-5000	AT3	
BSB3	Acct4	-7000	AT3	
NULL	NULL	NULL	NULL	

T_Account table output after successful transaction

Assumption for task 2

1. Test code has not been ran in the sql script, so you have to check please run the code
2. no need to apply an error handler for multiple row retrieval using select...into as the BSB and accountNo as composite are always unique in the table as assumption.

Task 3 DATABASE TRIGGER

- ** Rationale behind the type of trigger used **** I would choose before insert trigger for this purpose because it is more easier to validate criteria of all 5 business rules for each new insertion before inserting, and it is more logical to validate data before insert to maintain integrity of the database and consistency of the business rules. I choose to put the trigger in T_Loan because customer is allowed to have many accounts but only one loan account can be attached by maximum one loan. Having a trigger on T_Loan allows me to validate before inserting record based on the 5 business rules.
- **How the trigger is tested **** I will test this trigger based on 5 business rule, for each rule I would try to provide a successful case and a fail case. ****Home Loan**** I use C1 as my tested target, and he has no homeLoans attached at the beginning. So I insert three home loan records to show it is okay to insert home loan, and when I insert the fourth home loan, I should get a message as C1 has 3 home loans already and fourth one is not allowed. ****Individual Loan**** is my second check. Since **previous three individual home Loans are inserted**, I can now can insert the fifth individual loan as C1 was attached to an individual at the start. The sixth one would be failed as shown below (end of this task) because maximum number of individual account is 5. ****Personal loan**** C1 was attached with a personal loan at the start, the second case I insert would be failed(evidence shown end of this task). ****TotalLoanAmount**** I insert a loan with big loan amount exceeding the maximum of

\$10000000, which is shown as fail in the diagram at the end of this task. ****Maximum loan account**** up until this point, in the database, there is 6 loan accounts for C1 assuming the previous successful insertion are ran. So inserting 7th and 8th loan account{ not violating other business rules} should be fine and the 9th insertion would be shown as fail in the diagram at the end of this task.

iii. ****Code for implementing the trigger****

```

1 delimiter //
2 drop trigger if exists task3Trigger //
3 create trigger task3Trigger //
4 before insert on T_Loan//validate after insertion
5 for each row
6 begin
7     declare v_finished boolean default 0;#to track end of cursor
8     declare raiseError int default 0;
9
10    declare temp_CID varchar(10); #used to indicate current customerID in the cursor
11
12    declare temp_homeLoan int;#used to indicate number of home loans possessed by a customerID
13    declare personalLoan_count int;#used to indicate number of home loans possessed by a customerID
14    declare temp_totalLoanAmt decimal(11,2);#used to indicate total aamount of all loans possessed by a customerID
15    declare temp_IndividualLoan int; #used to indicate number of individual loans possessed by a customerID
16    declare temp_totalLoan int;#used to indicate number of total loans possessed by a customerID
17
18    declare msg varchar(100); #set error message
19
20    declare owner_rec cursor for#save all customerID linked to the Account just inserted to the T_Loan
21    select customer_ID
22    from T_Account, T_Own
23    where (BSB=Account_BSB and
24    AccountNo = Account_No) and
25    (BSB=new.Loan_AccountBSB and
26    AccountNo= new.Loan_AcctNo);
27    declare continue handler for not found set v_finished=1;
28
29    open owner_rec;
30    repeat
31    fetch owner_rec into temp_CID;
32
33    if(not v_finished) then
34        set temp_homeLoan =0;
35        set personalLoan_count=0;
36        set temp_totalLoanAmt = 0;
37        set temp_IndividualLoan = 0;
38        set temp_totalLoan = 0;
39
40        set temp_totalLoan = totalLoan(temp_CID)+1; #function used to calculate the total number of loans to a customerID
41        set temp_homeLoan= numberOfHomeLoan(temp_CID);
42        set personalLoan_count= numberOfPersonalLoan(temp_CID);
43        set temp_totalLoanAmt = totalLoanAmount(temp_CID)+new.LoanAmount;# function used to calculate the total Amount of all loans for a customerID
44        #function used to calculate the number of individual loans to a customerID
45        set temp_IndividualLoan = CountOfIndividualAccount(temp_CID)+isIndividualAccount(new.Loan_AccountBSB, new.Loan_AcctNo) ;
46
47        if(new.Loan_Type='LT1') then
48            set temp_homeLoan= temp_homeLoan +1;
49        elseif (new.Loan_Type='LT3') then
50            set personalLoan_count= personalLoan_count+1;
51        end if;
52
53        if (temp_totalLoan >8) then
54            set raiseError =1;
55            set msg = 'maximum number of loans is 8 ';
56            set v_finished=1;
57        elseif(temp_homeLoan>3) then
58            set raiseError =1;
59            set msg = 'maximum(3) of home loans is reached !';
60            set v_finished=1;
61        elseif (personalLoan_count>1) then
62            set raiseError =1;
63            set msg = 'maximum(1) of personal loans is reached !';
64            set v_finished=1;
65        elseif (temp_totalLoanAmt>10000000.00) then
66            set raiseError =1;
67            set msg = 'adding new loan exceeds maximum loan amount $10000000.00 ';
68            set v_finished=1;
69        elseif (temp_IndividualLoan >5) then
70            set raiseError =1;
71            set msg = 'maximum number of individual loans is 5.';
72            set v_finished=1;
73        end if ;
74
75        set temp_CID="";
76    end if ;
77    until v_finished
78    end repeat;
79    close owner_rec;
80
81    if (raiseError=1) then
82        signal sqlstate '45000' set message_text =msg;
83    end if ;
84 end
85 //
86 delimiter ;

```

Function 1-TotalLoan

```
88 delimiter //
89 #function helps to the total number of loans for a customerID which tells you how many loans belong to that customerID
90 drop function if exists totalLoan //
91 create function totalLoan(CID varchar(10))
92     returns int
93     deterministic
94     begin
95
96         declare result int default 0;
97
98         select count(*) into result
99         from T_Account, T_Own, T_Loan
100         where (BSB = Account_BSB and
101             AccountNo=Account_No and
102             BSB = Loan_AccountBSB and
103             AccountNo = Loan_AcctNo) and
104             Customer_ID=CID;
105         return result;
106     end
107 //
108 delimiter ;
109
```

Function 2-numberOfHomeLoan

```
109
110 delimiter //
111 ##function help to find the number of homeLoans for CID
112 drop function if exists numberOfHomeLoan //
113 create function numberOfHomeLoan(CID varchar(11))
114     returns int(11)
115     deterministic
116     begin
117         declare result int default 0;
118
119         select count(*) into result
120         from T_Account, T_Own, T_Loan
121         where (BSB = Account_BSB and
122             AccountNo=Account_No and
123             BSB = Loan_AccountBSB and
124             AccountNo = Loan_AcctNo) and
125             (Loan_Type ='LT1' and Customer_ID= CID);
126
127         return result;
128     end
129 //
130 delimiter ;
131
```

Function 3-numberOfPersonalLoan

```
132 delimiter //
133 #find out the number of personal loan
134 drop function if exists numberOfPersonalLoan //
135 create function numberOfPersonalLoan(CID varchar(11))
136     returns int(11)
137     deterministic
138     begin
139         declare result int default 0;
140         select count(*) into result
141         from T_Account, T_Own, T_Loan
142         where (BSB = Account_BSB and
143             AccountNo=Account_No and
144             BSB = Loan_AccountBSB and
145             AccountNo = Loan_AcctNo) and
146             (Loan_Type ='LT3' and Customer_ID= CID);
147
148         return result;
149     end
150 //
151 delimiter ;
152
```

Function 4 -totalLoanAmount

```
153 delimiter //
154 #check the totalLoanAmount
155 drop function if exists totalLoanAmount //
156 create function totalLoanAmount(CID varchar(11))
157     returns decimal(11,2)
158     deterministic
159 begin
160     declare result int default 0;
161     select sum(LoanAmount) into result
162     from T_Account, T_Own, T_Loan
163     where BSB = Account_BSB and
164           AccountNo=Account_No and
165           BSB = Loan_AccountBSB and
166           AccountNo = Loan_AcctNo and
167           Customer_ID=CID;
168     return result;
169 end
170 //
171 delimiter ;
172
173
```

Function 5-isIndividualAccount

```
174 delimiter //
175 #check if the account is an individualAccount or not
176 # yes return 1, no return 0
177 drop function if exists isIndividualAccount //
178 create function isIndividualAccount(loanBSB varchar(10), loanAcctNo varchar(10))
179     returns int
180     deterministic
181 begin
182     declare result int default 0;
183     select count(Customer_ID) into result
184     from T_Account, T_Own, T_Loan
185     where BSB = Account_BSB and
186           AccountNo=Account_No and
187           BSB = Loan_AccountBSB and
188           AccountNo = Loan_AcctNo and
189           BSB=loanBSB and AccountNo = loanAcctNo;
190
191     if(result=1) then
192         return 1;
193     end if ;
194     return 0;
195 end
196 //
197 delimiter ;
198
```

Function 6-CountOfIndividualAccount

```
89 delimiter //
90 #this function returns the number of individual account for a customerID
91 drop function if exists CountOfIndividualAccount //
92 CREATE FUNCTION CountOfIndividualAccount(CID varchar(11))
93 RETURNS int(11)
94 DETERMINISTIC
95 begin
96     declare v_finished int default 0;
97     declare tempCount int;#keep tracking of number of customerID(s) attached for an account
98
99     declare result int default 0;#store number of individual accounts for a customerID
100
101     declare temp_BSB varchar(10);#store BSB temporarily
102     declare temp_acc varchar(10);#store account number temporarily
103
104     declare loan_AcctRec cursor for
105     select BSB, AccountNo
106     from T_Account, T_Own, T_Loan
107     where (BSB = Account_BSB and
108           AccountNo=Account_No and
109           BSB=Loan_AccountBSB and
110           AccountNo = Loan_AcctNo) and
111           Customer_ID=CID;
112     declare continue handler for not found set v_finished=1;
113     open loan_AcctRec;
114     myLoop : loop
115         set tempCount =0;
116
117         fetch loan_AcctRec into temp_BSB, temp_acc;
118
119         if (v_finished) then
120             leave myLoop;
121         end if;
122         # find out the number of customerID attached to that account related to CID
123         select count(Customer_ID) into tempCount
124         from T_Own
125         where Account_BSB=temp_BSB and
126               Account_No = temp_acc;
127
128         #tempCount =1 means only one customerID attached
129         if(tempCount = 1) then
130             set result = result+1;
131         end if ;
132     end loop;
133     close loan_AcctRec;
134     return result;
135 end
136 //
137 delimiter ;
138
```


iv. **Script for testing the trigger (and test results)**

1	# test cases for homeLoan insertion				
2	●	INSERT INTO T_Account VALUES ('BSB10','Acct1','10.00','AT3');	# test case for first homeloan for C1		
3	●	INSERT INTO T_Own VALUES ('C1','BSB10','Acct1');	# test case for first homeloan for C1		
4	●	INSERT INTO T_Loan VALUES ('L10','0.05','5000.00','LT1','BSB10','Acct1');	#first homeloan for C1 [success]		
5					
6					
7	●	INSERT INTO T_Account VALUES ('BSB11','Acct1','10.00','AT3');	# test case for second homeloan for C1		
8	●	INSERT INTO T_Own VALUES ('C1','BSB11','Acct1');			
9	●	INSERT INTO T_Loan VALUES ('L11','0.05','5000.00','LT1','BSB11','Acct1');	#second homeloan for C1 [success]		
10					
11	●	INSERT INTO T_Account VALUES ('BSB12','Acct1','10.00','AT3');	#test case for third homeloan for C1		
12	●	INSERT INTO T_Own VALUES ('C1','BSB12','Acct1');			
13	●	INSERT INTO T_Loan VALUES ('L12','0.05','5000.00','LT1','BSB12','Acct1');	#third homeloan for C1 [success]		
14					
15					
16	●	INSERT INTO T_Account VALUES ('BSB13','Acct1','10.00','AT3');	#test case for fourth homeloan for C1		
17	●	INSERT INTO T_Own VALUES ('C1','BSB13','Acct1');			
18	●	INSERT INTO T_Loan VALUES ('L13','0.05','5000.00','LT1','BSB13','Acct1');	# fourth homeloan for C1 [fail]		
19					
20	#test cases for individual loan				
21	●	INSERT INTO T_Account VALUES ('BSB20','Acct1','10.00','AT3');	#test case for fifth individual loan for C1		
22	●	INSERT INTO T_Own VALUES ('C1','BSB20','Acct1');			
23	●	INSERT INTO T_Loan VALUES ('L20','0.05','5000.00','LT2','BSB20','Acct1');	# fifth individual loan for C1 [success]		
24					
25	●	INSERT INTO T_Account VALUES ('BSB21','Acct1','10.00','AT3');	#test case for sixth individual loan for C1		
26	●	INSERT INTO T_Own VALUES ('C1','BSB21','Acct1');			
27	●	INSERT INTO T_Loan VALUES ('L21','0.05','5000.00','LT2','BSB20','Acct1');	# sixth individual loan for C1 [fail]		
28					
29					
30	#test cases for personal loan insertion				
31	●	INSERT INTO T_Account VALUES ('BSB14','Acct1','10.00','AT3');	#test case for second personal loan for C1		
32	●	INSERT INTO T_Own VALUES ('C1','BSB14','Acct1');			
33	●	INSERT INTO T_Loan VALUES ('L14','0.05','5000.00','LT3','BSB14','Acct1');	# second personal loan for C1 [fail]		
34					
35	#test cases for totalAmount				
36	●	INSERT INTO T_Account VALUES ('BSB41','Acct1','10.00','AT3');	# test case for maximum loan amount for C2		
37	●	INSERT INTO T_Own VALUES ('C2','BSB41','Acct1');			
38	●	INSERT INTO T_Loan VALUES ('L41','0.05','10000000.00','LT1','BSB41','Acct1');	#second hmaximum loan amount for C2 [fail]		
39					
40	#test cases for maximum loan				
41	●	INSERT INTO T_Account VALUES ('BSB31','Acct1','10.00','AT3');	#test case for seventh loan for C1		
42	●	INSERT INTO T_Own VALUES ('C1','BSB31','Acct1');			
43	●	INSERT INTO T_Own VALUES ('C2','BSB31','Acct1');			
44	●	INSERT INTO T_Loan VALUES ('L31','0.05','5000.00','LT2','BSB31','Acct1');	# seventh loan for C1 [success]		
45					
46	●	INSERT INTO T_Account VALUES ('BSB32','Acct1','10.00','AT3');	#test case for eighth loan for C1		
47	●	INSERT INTO T_Own VALUES ('C1','BSB32','Acct1');			
48	●	INSERT INTO T_Own VALUES ('C2','BSB32','Acct1');			
49	●	INSERT INTO T_Loan VALUES ('L32','0.05','5000.00','LT2','BSB32','Acct1');	# eighth loan for C1 [success]		
50					
51	●	INSERT INTO T_Account VALUES ('BSB33','Acct1','10.00','AT3');	#test case for ninth loan for C1		
52	●	INSERT INTO T_Own VALUES ('C1','BSB33','Acct1');			
53	●	INSERT INTO T_Own VALUES ('C2','BSB33','Acct1');			
54	●	INSERT INTO T_Loan VALUES ('L33','0.05','5000.00','LT2','BSB33','Acct1');	# ninth loan for C1 [fail]		
55					
56					
57					





59	17:55:40	INSERT INTO T_Account VALUES ('BSB10','Acct1','10.00','AT3')	1 row(s) affected	0.0085 sec
60	17:55:40	INSERT INTO T_Own VALUES ('C1','BSB10','Acct1')	1 row(s) affected	0.0070 sec
61	17:55:40	INSERT INTO T_Loan VALUES ('L10','0.05','5000.00','LT1','BSB10','Acct1')	1 row(s) affected	0.019 sec
Open a script file in this editor		INSERT INTO T_Account VALUES ('BSB11','Acct1','10.00','AT3')	1 row(s) affected	0.0054 sec
63	17:55:40	INSERT INTO T_Own VALUES ('C1','BSB11','Acct1')	1 row(s) affected	0.012 sec
64	17:55:40	INSERT INTO T_Loan VALUES ('L11','0.05','5000.00','LT1','BSB11','Acct1')	1 row(s) affected	0.0072 sec
65	17:55:40	INSERT INTO T_Account VALUES ('BSB12','Acct1','10.00','AT3')	1 row(s) affected	0.0082 sec
66	17:55:40	INSERT INTO T_Own VALUES ('C1','BSB12','Acct1')	1 row(s) affected	0.0072 sec
67	17:55:40	INSERT INTO T_Loan VALUES ('L12','0.05','5000.00','LT1','BSB12','Acct1')	1 row(s) affected	0.0078 sec
68	17:55:43	INSERT INTO T_Account VALUES ('BSB13','Acct1','10.00','AT3')	1 row(s) affected	0.0086 sec
69	17:55:43	INSERT INTO T_Own VALUES ('C1','BSB13','Acct1')	1 row(s) affected	0.0071 sec
70	17:55:43	INSERT INTO T_Loan VALUES ('L13','0.05','5000.00','LT1','BSB13','Acct1')	Error Code: 1644, maximum(3) of home loans is reached !	0.0062 sec
71	17:55:48	INSERT INTO T_Account VALUES ('BSB20','Acct1','10.00','AT3')	1 row(s) affected	0.024 sec
72	17:55:48	INSERT INTO T_Own VALUES ('C1','BSB20','Acct1')	1 row(s) affected	0.014 sec
73	17:55:48	INSERT INTO T_Loan VALUES ('L20','0.05','5000.00','LT2','BSB20','Acct1')	1 row(s) affected	0.026 sec
74	17:55:48	INSERT INTO T_Account VALUES ('BSB21','Acct1','10.00','AT3')	1 row(s) affected	0.028 sec
75	17:55:48	INSERT INTO T_Own VALUES ('C1','BSB21','Acct1')	1 row(s) affected	0.023 sec
76	17:55:48	INSERT INTO T_Loan VALUES ('L21','0.05','5000.00','LT2','BSB20','Acct1')	Error Code: 1644, maximum number of individual loans is 5.	0.014 sec
77	17:55:52	INSERT INTO T_Account VALUES ('BSB14','Acct1','10.00','AT3')	1 row(s) affected	0.013 sec
78	17:55:53	INSERT INTO T_Own VALUES ('C1','BSB14','Acct1')	1 row(s) affected	0.0053 sec
79	17:55:53	INSERT INTO T_Loan VALUES ('L14','0.05','5000.00','LT3','BSB14','Acct1')	Error Code: 1644, maximum(1) of personal loans is reached !	0.0052 sec
80	17:55:57	INSERT INTO T_Account VALUES ('BSB41','Acct1','10.00','AT3')	1 row(s) affected	0.0058 sec
81	17:55:57	INSERT INTO T_Own VALUES ('C2','BSB41','Acct1')	1 row(s) affected	0.0052 sec
82	17:55:57	INSERT INTO T_Loan VALUES ('L41','0.05','10000000.00','LT1','BSB41','Acct1')	Error Code: 1644, adding new loan exceeds maximum loan amoun...	0.0046 sec
83	17:56:05	INSERT INTO T_Account VALUES ('BSB31','Acct1','10.00','AT3')	1 row(s) affected	0.0058 sec
84	17:56:05	INSERT INTO T_Own VALUES ('C1','BSB31','Acct1')	1 row(s) affected	0.0054 sec
85	17:56:05	INSERT INTO T_Own VALUES ('C2','BSB31','Acct1')	1 row(s) affected	0.0062 sec
86	17:56:05	INSERT INTO T_Loan VALUES ('L31','0.05','5000.00','LT2','BSB31','Acct1')	1 row(s) affected	0.022 sec
87	17:56:09	INSERT INTO T_Account VALUES ('BSB32','Acct1','10.00','AT3')	1 row(s) affected	0.0076 sec
88	17:56:09	INSERT INTO T_Own VALUES ('C1','BSB32','Acct1')	1 row(s) affected	0.0094 sec
89	17:56:09	INSERT INTO T_Own VALUES ('C2','BSB32','Acct1')	1 row(s) affected	0.0052 sec
90	17:56:09	INSERT INTO T_Loan VALUES ('L32','0.05','5000.00','LT2','BSB32','Acct1')	1 row(s) affected	0.012 sec
91	17:56:12	INSERT INTO T_Account VALUES ('BSB33','Acct1','10.00','AT3')	1 row(s) affected	0.0061 sec
92	17:56:12	INSERT INTO T_Own VALUES ('C1','BSB33','Acct1')	1 row(s) affected	0.0069 sec
93	17:56:12	INSERT INTO T_Own VALUES ('C2','BSB33','Acct1')	1 row(s) affected	0.0058 sec
94	17:56:12	INSERT INTO T_Loan VALUES ('L33','0.05','5000.00','LT2','BSB33','Acct1')	Error Code: 1644, maximum number of loans is 8	0.0065 sec

Assumption for task 3

1. I created loan accounts to test if the trigger works when insert new loan in T_Loan.
2. New insertion is counted when determining if the insertion breaks 5 business rules as if the customer already has 1 personal loan and if the loan type of new insertion for that customer is personal loan, the trigger should raise a error signal to reject the insertion.
3. The way I validate the trigger is using relevant test cases, successful insertion in for verifying previous business rule is needed for testing next business rule.
4. Assuming customers can create multiple accounts (including loan accounts) but, each account can only have maximum one loan.

Task 4 Interest Calculation

1. ****Issues with the relational schema**** no , because the offset table should be in a separate table and is linked to the T_Loan. The offset is based on the individual loan account then there should be a relationship between offset table and T_account table to validate the constraint of the account detail as the account detail would be used to link both table together for foreign key integrity, because I assume the offset table is based on each individual account, so I need to modify the current schema for adding a table and link the offset table to T_account to constraint the integrity of account detail(BSB, accountNo).
2. ****Suggested changes to the relational schema**** As I mentioned in question 1, creating a offset table and linking to the T_account table is necessary in this situation as I want to ensure the data insertion is relevant to the loan account, and the foreign constraint allows me to validate this. There I would create an offset table using BSB
3. ****Implementation of the suggested changes****

Result Grid		  Filter Rows:	 Search	
	offsetDate	offsetBSB	offsetAcctNo	offsetAmount
	NULL	NULL	NULL	NULL

	Time	Action	Response	Duration / Fet
✓ 1	14:11:21	create table offsetTable (offsetDate date not null, offsetBSB varchar(10) not null,...	0 row(s) affected	0.031 sec

```
#this table is created for task 4
create table offsetTable (
  offsetDate date not null,
  offsetBSB varchar(10) not null,
  offsetAcctNo varchar(10) not null,
  offsetAmount decimal(11,2) not null,
  primary key (offsetBSB, offsetAcctNo, offsetDate),
  FOREIGN KEY ( offsetBSB, offsetAcctNo)
  REFERENCES T_Loan ( Loan_AccountBSB, Loan_AcctNo));
```


4. **Interest Calculation procedure**

(i) SQL Code

```
1 delimiter //
2 drop procedure if exists interestCalculator //
3 create procedure interestCalculator (in startingDate date, in endDate date ,in loanID varchar(11))
4 begin
5     declare not_found_row int default 0;
6     declare temp_offsetAmount decimal default 0;
7     declare temp_repaymentAmount decimal default 0 ;
8     declare interest decimal(11,2);
9     declare tempBSB varchar(10);
10    declare tempAcctNo varchar(10);
11    declare interestRate decimal(10,8);
12    declare loanBalance decimal(11,2);
13    declare continue handler for not found set not_found_row =1;
14
15    select BSB, AccountNo, LoanRate,AccountBal into tempBSB, tempAcctNo, interestRate, LoanBalance
16    from T_Account, T_Loan
17    where BSB=Loan_AccountBSB and
18    AccountNo = Loan_AcctNo and
19    T_Loan.LoanID=loanID;
20
21    set interest =0;
22    if(not_found_row) then
23        signal sqlstate '45000' set message_text = 'no account found for that loanID';
24    end if;
25
26    select subDate(endDate, interval 1 day) into endDate;
27
28    set temp_offsetAmount =getOffset(endDate,tempBSB,tempAcctNo);
29
30    while (datediff(endDate, startingDate)>0) do
31        set temp_offsetAmount =getOffset(endDate,tempBSB,tempAcctNo);
32        set interest = interest+ ((loanBalance+temp_offsetAmount)*interestRate/365);
33        set loanBalance=loanBalance-getRepayment(endDate,loanID);
34        select subDate(endDate, interval 1 day) into endDate;
35    end while;
36
37    select concat('Interest for ', loanID, ' is ',interest ) as 'result';
38 end
39 //
40 delimiter ;
41
42 delimiter //
43 drop function if exists getOffset //
44 #get offset amount for a account and bsb and a date
45 create function getOffset(d date, bsb varchar(10), acct varchar(10))
46 returns decimal(11,2)
47 deterministic
48 begin
49     declare row_not_found int default 0;
50     declare result decimal(11,2);
51     declare continue handler for not found set row_not_found =1;
52
53     select offsetAmount into result
54     from offsetTable where
55     offsetBSB=bsb and
56     offsetAcctNo = acct and
57     datediff(d,offsetDate)>=0
58     order by offsetDate desc
59     limit 1;
60
61     if (row_not_found) then
62         return 0;
63     else
64         return result;
65     end if;
66 end
67 //
68 delimiter ;
69
70 delimiter // #getRepayment for a data and loanID
71 drop function if exists getRepayment //
72 create function getRepayment(d date, loan varchar(10))
73 returns decimal(11,2)
74 deterministic
75 begin
76     declare row_not_found int default 0;
77     declare result decimal(11,2);
78     declare continue handler for not found set row_not_found =1;
79
80     select RepaymentAmount into result
81     from T_Repayment where
82     RepaymentDate=d and
83     Repayment_LoanID = loan;
84
85     if (row_not_found) then
86         return 0;
87     else
88         return result;
89     end if;
90 end
91 //
92 delimiter ;
93
```

(ii) Errors/Warnings

errors would be shown if there is no Account found for the loanID

Task 4

Assumption

1. I assume the days in a year is 365 as there is no written requirement for us to calculate the days of a year.
2. The parameters starting date and end date makes my procedure more flexible, as there is no requirements for not adding another parameters.

5. **Testing the procedure for monthly interest calculation**

```
2  /*
3  For task 4
4  */
5
6  #Test case of simple calculator example for task 4
7  INSERT INTO T_Customer VALUES ('C4','matthew','lee','1234567','dMail','2015-10-12');
8  INSERT INTO T_Account VALUES ('BSB50','Acct1','-60000.00','AT3');
9  INSERT INTO T_Own VALUES ('C4','BSB50','Acct1');
10 INSERT INTO T_Loan VALUES ('L50','0.05','670500.00','LT1','BSB50','Acct1');
11 INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
12 VALUES ('L50','5000','2018-10-20');
13 INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
14 VALUES ('L50','3000','2018-10-18');
15
16 INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
17 VALUES ('L50','5000','2018-09-26');
18
19
20 INSERT INTO offsetTable values ('2018-10-21','BSB50','Acct1',7000);
21 INSERT INTO offsetTable values ('2018-10-14','BSB50','Acct1',5000);
22 INSERT INTO offsetTable values ('2018-10-07','BSB50','Acct1',8000);
23 INSERT INTO offsetTable values ('2018-09-30','BSB50','Acct1',6000);
24 INSERT INTO offsetTable values ('2018-09-23','BSB50','Acct1',12000);
25
26
27 #March test case for task 4
28 INSERT INTO T_Customer VALUES ('C5','Yvonne','Lam','12344','dMail','2015-10-12');
29 INSERT INTO T_Account VALUES ('BSB51','Acct1','-60000.00','AT3');
30 INSERT INTO T_Own VALUES ('C5','BSB51','Acct1');
31 INSERT INTO T_Loan VALUES ('L51','0.05','60000.00','LT1','BSB51','Acct1');
32 INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
33 VALUES ('L51','6000','2018-03-20');
34 INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
35 VALUES ('L51','7000','2018-03-18');
36
37 INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
38 VALUES ('L51','5000','2018-02-26');
39
40 INSERT INTO offsetTable values ('2018-03-21','BSB50','Acct1',6000);
41 INSERT INTO offsetTable values ('2018-03-14','BSB50','Acct1',5000);
42 INSERT INTO offsetTable values ('2018-03-07','BSB50','Acct1',8000);
43 INSERT INTO offsetTable values ('2018-02-28','BSB50','Acct1',6000);
44
```

```

44
45 #May test case for task 4
46 • INSERT INTO T_Customer VALUES ('C6','Vicky','Su','123444','dMail','2015-10-12');
47 • INSERT INTO T_Account VALUES ('BSB52','Acct1','-60000.00','AT3');
48 • INSERT INTO T_Own VALUES ('C6','BSB52','Acct1');
49 • INSERT INTO T_Loan VALUES ('L52','0.05','60000.00','LT1','BSB52','Acct1');
50 • INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
51   VALUES ('L52','6000','2018-05-20');
52 • INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
53   VALUES ('L52','7000','2018-05-18');
54
55 • INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
56   VALUES ('L52','5000','2018-04-26');
57
58 • INSERT INTO offsetTable values ('2018-05-21','BSB52','Acct1',7000);
59 • INSERT INTO offsetTable values ('2018-05-14','BSB52','Acct1',5000);
60 • INSERT INTO offsetTable values ('2018-04-26','BSB50','Acct1',12000);
61
734 #August test case for task 4
735 • INSERT INTO T_Customer VALUES ('C7','ricky','Tam','123444','dMail','2015-10-12');
736 • INSERT INTO T_Account VALUES ('BSB53','Acct1','-60000.00','AT3');
737 • INSERT INTO T_Own VALUES ('C7','BSB53','Acct1');
738 • INSERT INTO T_Loan VALUES ('L53','0.05','60000.00','LT1','BSB53','Acct1');
739 • INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
740   VALUES ('L53','6000','2018-08-20');
741 • INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
742   VALUES ('L53','7000','2018-08-18');
743 • INSERT INTO T_Repayment (Repayment_LoanID, RepaymentAmount, RepaymentDate)
744   VALUES ('L53','5000','2018-07-26');
745
746 • INSERT INTO offsetTable values ('2018-08-21','BSB52','Acct1',6000);
747 • INSERT INTO offsetTable values ('2018-08-14','BSB52','Acct1',5000);
748 • INSERT INTO offsetTable values ('2018-07-26','BSB52','Acct1',6000);
749
80
81 • call InterestCalculator('2018-09-24','2018-10-25','L50');
82 • call InterestCalculator('2018-02-24','2018-03-25','L51');
83 • call InterestCalculator('2018-04-24','2018-05-25','L52');
84 • call InterestCalculator('2018-07-24','2018-08-25','L53');
85
86
87

```

Result

(i) On October 25, 2018

Result Grid	Filter RO
result	
▶ Interest for L50 is -243.54	

(ii) On March 25, 2018

Result Grid	Filter RO
result	
▶ Interest for L51 is -269.86	

(iii) On May 25, 2018

Result Grid		Filter Row:
	result	
▶	Interest for L52 is -281.25	

(iv) On August 25, 2018

Result Grid		Filter Row:
	result	
▶	Interest for L53 is -299.86	

Task 5 Transaction Management (Theory)

a. A transaction is a logical unit of work on database, and it is an action or series of actions, carried out by application program or single user, that updates or reads the contents of the database. **Example** $r_1(T_Own)$ to read the all accounts relating to 'C1' in table T_Own is an action carried out by single user.

b. Schedule is a sequence of operations carried out by multiple concurrent transactions, that preserves order of the operations in each individual transaction. This is different with transaction, as schedule can comprise of multiple transactions, each can have multiple actions where as transaction is a logical unit of work. **Example:** $S_1:(r_1(T_Own);r_2(T_Account);)$ involve two transactions-a transaction for reading T_Own and a transaction for reading $T_Account$.

c. Conflict serialisable is that a schedule which:

1. produces the same result as some serial execution
2. Orders conflicting operations
3. There are not conflicting operations in the schedule
4. Can be transformed into a serial schedule by swapping non-conflicting instructions

Conflicting operations are :

- two operations are from different transactions
- operations are on same data item
- at least one of the operations is a write operation

Conflicting operations might lead to lost update, dirty read, and unrepeated read

A conflict serialisable schedule guarantee database consistency because:

If there are no conflicting operations in the schedule, we can freely change the order of the non-conflicting operations, we then can produce the equivalent serial schedule, which guarantee database consistency, if there are no conflicting operations. **Example** $S_1 : r_1(T_Own); r_1(T_Account); w_1(T_Repayment); r_2(T_Loan); r_3(T_Own);$ has no conflicting operations, so it is a conflicting serialised schedule.

d.

Two schedules are regarded as conflicting equivalent if one schedule can be shown to be equivalent to the other by swapping non-conflicting operations, both schedules have the same set of transactions and the orders of conflicting actions are the same, then they leave the database in same state. Since the question says different conflicting serialisable schedules are made up of the same set of transactions, but the question doesn't say the orders of conflicting actions are the same, therefore, assuming the same set of transactions also includes same order of conflicting actions and same ordering of action within each transaction, then the two schedules are serialisable equivalent and they leave the database in the same state, as these two schedules can be proven to be conflict equivalent.

e.

View serialisability is that a schedule is view serialisable if it is view equivalent to a serial schedule. View serialisability allows blind write, therefore, it can protect the confidentiality of data, write data without exposing underlying structure where as blind write is not allowable in conflict serialisable schedule. Blind write can also make the schedules are robust as reading before writing is not needed, which makes the transaction faster.

Example-view serialisable but not conflicting schedule

T1	T2	T3
Read (T_Own)		
	Write(T_Own)	
Write(T_Own)		Write(T_Own)