Department of Computer Science and Engineering

Islamic University of Technology (IUT)

A subsidiary organ of OIC

CSE 4508: RDBMS

Name: Anm Zahid Hossain Milkan

Student ID: 200041202
Section: 2
Semester: 5th
Academic Year: 2022-23

Date of Submission: 14/11/23

CREATION OF NECESSARY TABLES:

```
--from previous lab
ALTER TABLE Books
ADD availability NUMBER DEFAULT 0;
CREATE TABLE Borrowed_Books (
    txn_id NUMBER(10) PRIMARY KEY,
    b_id NUMBER(10),
    borrower_name VARCHAR2(100),
    borrowed_date DATE,
    returning_date DATE,
    CONSTRAINT fk_borrowed_books_book
    FOREIGN KEY (b_id)
    REFERENCES Books (book_id)
);

CREATE TABLE book_transaction_log (
    log_id NUMBER PRIMARY KEY,
    txn_type VARCHAR2(10),
    txn_date TIMESTAMP,
    b_id NUMBER,
    borrower_name VARCHAR2(100),
    borrowed_date DATE,
    returning_date DATE
);
```

TASK 1

Insert some books, and borrowed_books data into their respective tables. Create relationships
between the tables, ensuring that foreign keys are properly set to maintain data integrity. 1. Create a trigger ‘update_book_availability’ to automatically update the availability status of a
book in the books table when a book is borrowed or returned. Logic:

If a new record is inserted into borrowed_books, the trigger decreases the availability of the corresponding book by 1. If an existing record is updated, or the returning_date is behind of the
PRESENT_DATE, the trigger increases the availability of the corresponding book by 1.

Tigger Type:
The trigger is an AFTER INSERT OR UPDATE trigger on the borrowed_book table

SQL:

```
ALTER TABLE Books
ADD availability NUMBER DEFAULT 0;
-- Create the Borrowed_Books table
CREATE TABLE Borrowed_Books (
    txn_id NUMBER(10) PRIMARY KEY,
    b_id NUMBER(10),
    borrower_name VARCHAR2(100),
    borrowed_date DATE,
    returning_date DATE,
    CONSTRAINT fk_borrowed_books_book
    FOREIGN KEY (b_id)
    REFERENCES Books (book_id)
```

```
);

CREATE TABLE book_transaction_log (
    log_id NUMBER PRIMARY KEY,
    txn_type VARCHAR2(10),
    txn_date TIMESTAMP,
    b_id NUMBER,
    borrower_name VARCHAR2(100),
    borrowed_date DATE,
    returning_date DATE
);
```

TASK 2:

In the previous scenario, create another trigger that logs changes to the 'borrowed_books'table in a separate 'book_transaction_log' table.
In this connection, create a function called 'log_book_transaction' that encapsulates this trigger to insert records into the 'book_transaction_log' table based on the operations (INSERT or UPDATE) in the borrowed_books table.

SQL:

```
-- Create a sequence for log_id in book_transaction_log
table
CREATE SEQUENCE log_id_sequence
START WITH 1
INCREMENT BY 1;

-- Create the function to log book transactions
CREATE OR REPLACE FUNCTION log_book_transaction(
```

```
    p_txn_type VARCHAR2,
    p_b_id NUMBER,
    p_borrower_name VARCHAR2,
    p_borrowed_date DATE,
    p_returning_date DATE
) RETURN NUMBER
AS
BEGIN
    INSERT INTO book_transaction_log (
        log_id,
        txn_type,
        txn_date,
        b_id,
        borrower_name,
        borrowed_date,
        returning_date
    ) VALUES (
        log_id_sequence.NEXTVAL,
        p_txn_type,
        CURRENT_TIMESTAMP,
        p_b_id,
        p_borrower_name,
        p_borrowed_date,
        p_returning_date
    );

    RETURN log_id_sequence.CURRVAL;
END log_book_transaction;
/
```

For the logging the borrowed books the following trigger is created:

```
-- Create the trigger to call the log_book_transaction
function
CREATE OR REPLACE TRIGGER log_borrowed_books_changes
AFTER INSERT OR UPDATE ON borrowed_books
FOR EACH ROW
DECLARE
    v_txn_type VARCHAR2(10);
    logid NUMBER;
BEGIN
    IF INSERTING THEN
        v_txn_type := 'INSERT';
    ELSIF UPDATING THEN
        v_txn_type := 'UPDATE';
    END IF;

    -- Call the log_book_transaction function
    logid := log_book_transaction(
        v_txn_type,
        :NEW.b_id,
        :NEW.borrower_name,
        :NEW.borrowed_date,
        :NEW.returning_date
    );

END log_borrowed_books_changes;
/
```

TASK 3

Assume that both triggers (in Questions 1 & 2) are set to fire AFTER INSERT OR UPDATE on the borrowed_books table. Specify the ideal firing order for these triggers to ensure proper functionality and data consistency. Provide specific examples of the chosen firing order of the triggers. If the firing order is changed, discuss how it might impact the behavior of the system.

Solution:

Ideal Firing Order Scenario:
1. Initially, the `update_book_availability` trigger should be fired.
2. Subsequently, the `log_borrowed_books_changes` trigger should be executed.

The reasoning behind this sequence is to ensure that the `update_book_availability` trigger updates the book's availability before any logging takes place. The update trigger should precede the logging trigger to guarantee that the log records the most recent and accurate information.

Potential Impact of Changing Firing Order:
If the firing order is altered, there is a risk that the logging trigger might record data before the availability is updated. This would result in the system logging outdated information.

Methods to Address the Issue:
There are two approaches to resolve this potential issue:

1. Combine Triggers:

- Merge the logic of the `update_book_availability` and `log_borrowed_books_changes` triggers into a single trigger.
- Ensure that the logic within the combined trigger follows the desired order of execution.

2. Use Precedence (Not Standard Oracle Command):

- Utilize the "follows" or "precedes" keywords to establish precedence between triggers.
- Note that these keywords might not be universally supported across all Oracle devices and versions.

Example of Precedence:

SQL:

```
ALTER TRIGGER log_borrowed_books_changes
ENABLE
FOLLOWS update_book_availability;
```

By adopting one of these methods, the firing order can be controlled to ensure the correct sequence of trigger execution.