

# IT5008: Tutorial 6 — Stored Procedures and Triggers

Pratik Karmakar

School of Computing,  
National University of Singapore

AY25/26 S1



# Scenario

Students at the **National University of Ngendipura (NUN)** buy, lend, and borrow books.

NUNStA commissions *Apasaja Private Limited* to implement an online book exchange system that records:

- Students: name, faculty, department, **email**, join year.
- Books: title, authors, publisher, edition, **ISBN10**, **ISBN13**.
- Loans: borrowed date, returned date (NULL if active).

Auditing preserves records of graduated students and copies with loans.

This tutorial uses the schema/data from “Creating and Populating Tables.”

# Questions

## 1 Stored Functions and Procedures

- (a) Implement `borrow_book` (function/procedure) to check availability of a copy, insert a loan, and return/raise a message.
  - Scenario: Adeline Wong (`awong007@msn.com`) tries to borrow 3 copies of “Applied Calculus” (ISBN13=978-0470170526).

## 2 Triggers

- (a) Create a trigger that enforces: a student may have at most 3 active loans (local strategy).
- (b) Create a trigger that ensures globally no student exceeds 3 active loans (global strategy).

# 1(a). Function borrow\_book

```
CREATE OR REPLACE FUNCTION borrow_book (  
    borrower_email VARCHAR(256),  
    isbn13 CHAR(14),  
    borrow_date DATE  
) RETURNS TEXT AS $$  
DECLARE  
    available_copy RECORD;  
BEGIN  
    SELECT * INTO available_copy  
    FROM copy c  
    WHERE c.book = isbn13  
    AND NOT EXISTS (  
        SELECT 1 FROM loan l  
        WHERE l.book=c.book  
        AND l.copy=c.copy  
        AND l.owner=c.owner  
        AND l.returned IS NULL)  
    LIMIT 1;
```

```
IF NOT FOUND THEN  
    RETURN 'No available copies of '||isbn13;  
ELSE  
    INSERT INTO loan (borrower, owner, book,  
        copy, borrowed)  
    VALUES (borrower_email, available_copy.owner,  
        available_copy.book,  
        available_copy.copy, borrow_date);  
    RETURN 'Book '||isbn13||' borrowed by  
        '||borrower_email;  
END IF;  
END;  
$$ LANGUAGE plpgsql;
```

# 1(a). Procedure borrow\_book

```
CREATE OR REPLACE PROCEDURE borrow_book (  
    borrower_email VARCHAR(256),  
    isbn13 CHAR(14),  
    borrow_date DATE  
) AS $$  
DECLARE  
    available_copy RECORD;  
BEGIN  
    SELECT * INTO available_copy  
    FROM copy c  
    WHERE c.book = isbn13  
    AND NOT EXISTS (  
        SELECT 1 FROM loan l  
        WHERE l.book=c.book  
        AND l.copy=c.copy  
        AND l.owner=c.owner  
        AND l.returned ISNULL)  
    LIMIT 1;
```

```
IF NOT FOUND THEN  
    RAISE NOTICE 'No copies of %', isbn13;  
ELSE  
    INSERT INTO loan (borrower, owner, book,  
        copy, borrowed)  
    VALUES (borrower_email, available_copy.owner,  
        available_copy.book,  
        available_copy.copy, borrow_date);  
    RAISE NOTICE 'Book % borrowed by %', isbn13,  
        borrower_email;  
END IF;  
END;  
$$ LANGUAGE plpgsql;
```

## 2(a). Local Loan Limit Trigger

Max 3 active loans per student.

```
1 CREATE OR REPLACE FUNCTION check_local_loan_limit()
2 RETURNS TRIGGER AS $$
3 DECLARE active_loan_count INT;
4 BEGIN
5     SELECT COUNT(*) INTO active_loan_count
6     FROM loan l
7     WHERE l.borrower = NEW.borrower
8         AND l.returned ISNULL;
9
10    IF active_loan_count >= 3 THEN
11        RETURN NULL; -- prevent insert
12    ELSE
13        RETURN NEW;
14    END IF;
15 END;
16 $$ LANGUAGE plpgsql;
17
18 CREATE TRIGGER enforce_local_loan_limit_insert
19 BEFORE INSERT ON loan
20 FOR EACH ROW EXECUTE FUNCTION check_local_loan_limit();
```

## 2(b). Global Loan Limit Trigger

Ensure no student exceeds 3 active loans.

```
1 CREATE OR REPLACE FUNCTION check_global_loan_limit()
2 RETURNS TRIGGER AS $$
3 DECLARE violating_student RECORD;
4 BEGIN
5     SELECT l.borrower INTO violating_student
6     FROM loan l
7     WHERE l.returned ISNULL
8     GROUP BY l.borrower
9     HAVING COUNT(*) > 3;
10
11     IF violating_student IS NOT NULL THEN
12         RAISE EXCEPTION '% exceeds loan limit', violating_student;
13     ELSE
14         RETURN NEW;
15     END IF;
16 END;
17 $$ LANGUAGE plpgsql;
18
19 CREATE TRIGGER enforce_global_loan_limit
20 AFTER INSERT OR UPDATE ON loan
21 FOR EACH ROW EXECUTE FUNCTION check_global_loan_limit();
```

## Guidelines & Remarks

- Functions return a value; procedures raise notices or exceptions.
- Local vs Global strategies: local checks per row; global checks across all rows.
- Use **BEFORE** triggers with `RETURN NULL` to cancel insertions.
- Use **AFTER** triggers with `RAISE EXCEPTION` for global consistency.
- Dropping schema (`DROP SCHEMA ... CASCADE`) ensures removal of triggers/functions.



Questions?

Drop a mail at: [pratik.karmakar@u.nus.edu](mailto:pratik.karmakar@u.nus.edu)