# Tutorial: Stored Procedures

Students at the National University of Ngendipura (NUN) buy books for their studies. They also lend and borrow books to and from other students. Your company, Apasaja Private Limited, is commissioned by NUN Students Association (NUNStA) to implement an online book exchange system that records information about students, books that they own and books that they lend and borrow.

The database records the name, faculty, and department of each student. Each student is identified in the system by her email. The database also records the date at which the student joined the university (year attribute).

The database records the title, authors, publisher, year and edition and the ISBN-10 and ISBN-13 for each book. The International Standard Book Number, ISBN-10 or -13, is an industry standard for the unique identification of books. It is possible that the database records books that are not owned by any students (because the owners of a copy graduated or because the book was advised by a lecturer for a course but not yet purchased by any student.)

The database records the date at which a book copy is borrowed and the date at which it is returned. We refer to this information as a loan record.

For auditing purposes the database records information about the books, the copies and the owners of the copies as long as the owners are students or as there are loan records concerning the copies. For auditing purposes the database records information about graduated students as long as there are loan records concerning books that they owned.

**This tutorial uses the schema and data for the database created in "Tutorial: Creating and Populating Tables" including all the updates done during the tutorial.**

## Questions

*Not all questions will be discussed during tutorial. You are expected to attempt them before coming to the tutorial. You may be randomly called to present your answer during tutorial. You are encouraged to discuss them on Canvas Discussion.*

1. **Stored Functions and Procedures**.
   (a) Write a function `borrow_book_func` that, given the the email of a borrower ( `VARCHAR(256)` ), the ISBN13 of a book ( `CHAR(14)` ), and the borrow date ( `DATE` ), checks whether there is an available copy of the book, and, if that is the case, inserts a new loan record of the copy by the borrower. Return a message indicating success or failure of insertion.

Additionally, execute the following scenario using your function.

> Adeline Wong, with email awong007@msn.com, tries to borrow 3 copies of "Applied Calculus" by Deborah Hughes-Hallett, et al. with ISBN13 value of 978-0470170526.

**Comments:**

One possible function is the following.

**Code: Function**

```
CREATE OR REPLACE FUNCTION borrow_book_func (
  borrower_email VARCHAR (256), isbn13 CHAR (14), borrow_date DATE
) RETURNS TEXT AS $$
DECLARE
  available_copy RECORD ;
BEGIN
  -- Check for a copy of the book that is not currently borrowed
  --   (i.e., no active loan)
  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 -- No available copy found, return a message
  THEN
    RETURN 'No available copies of the book with ISBN13 : ' || isbn13;
  ELSE -- An available copy found
    -- Insert a new record into the loan table to record the borrowing
    INSERT INTO loan (borrower, owner, book, copy, borrowed)
    VALUES (borrower_email, available_copy.owner,
      available_copy.book, available_copy.copy, borrow_date);
    -- Return a success message
    RETURN 'Book with ISBN13 : ' || isbn13 ||
      ' has been successfully borrowed by ' || borrower_email;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

**Code: Invocation**

```
SELECT borrow_book_func ('awong007@msn.com', '978-0470170526',
  CURRENT_DATE);
SELECT borrow_book_func ('awong007@msn.com', '978-0470170526',
  CURRENT_DATE);
SELECT borrow_book_func ('awong007@msn.com', '978-0470170526',
  CURRENT_DATE);
```

(b) Write a procedure `borrow_book_proc` that, given the the email of a borrower ( `VARCHAR(256)` ), the ISBN13 of a book ( `CHAR(14)` ), and the borrow date ( `DATE` ), checks whether there is an available copy of the book, and, if that is the case, inserts a new loan record of the copy by the borrower. Raise a notice [3] indicating success or failure of insertion. Additionally, execute the following scenario using your procedure.

> Adeline Wong, with email `awong007@msn.com`, tries to borrow 4 copies of `"Calculus: Single Variable"` by Deborah Hughes-Hallett, et al. with ISBN13 value of `978-0470089156`.

**Comments:**

One possible procedure is the following.

**Code: Procedure**

```
CREATE OR REPLACE PROCEDURE borrow_book_proc (
  borrower_email VARCHAR (256), isbn13 CHAR (14), borrow_date DATE
) AS $$
DECLARE
  available_copy RECORD;
BEGIN
  -- Check for a copy of the book that is not currently borrowed
  --   (i.e., no active loan)
  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 -- No available copy found, raise notice
  THEN
    RAISE NOTICE 'No available copies of the book with ISBN13 : %',
      isbn13;
    RETURN;
  ELSE -- An available copy found
    -- Insert a new record into the loan table to record the borrowing
    INSERT INTO loan (borrower, owner, book, copy, borrowed)
    VALUES (borrower_email, available_copy.owner,
      available_copy.book, available_copy.copy, borrow_date);
    -- Raise a success message
    RAISE NOTICE 'Book with ISBN13 : % has been successfully
      borrowed by %', isbn13, borrower_email;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

**Note:** In cases of failure, you can also `RAISE EXCEPTION` to stop any further execution. You may then choose to trap [4] the exception (equivalent to a `catch` clause) and recover from it; otherwise, the exception will end the current transaction, and the database is rolled back to the state before the transaction begins.

**Code: Invocation**

```
CALL borrow_book_proc ('awong007@msn.com', '978-0470089156',
  CURRENT_DATE);
CALL borrow_book_proc ('awong007@msn.com', '978-0470089156',
  CURRENT_DATE);
CALL borrow_book_proc ('awong007@msn.com', '978-0470089156',
  CURRENT_DATE);
CALL borrow_book_proc ('awong007@msn.com', '978-0470089156',
  CURRENT_DATE);
```

2. **Cursor**.

   (a) Write a function `borrow_diff` that takes in no parameter and does the following steps. First, sort the students in terms of the number of books borrowed in descending order. If there are multiple students with the same number of books borrowed, they can appear in any order. Then, find the difference in the number of books borrowed between a student $S$ and the previous student in the sorted order. For the first student, let the difference be `NULL`.

   The output should be a table with the following columns.

| email | diff |
|---|---|

   `email` is the student email in the given sorted order. `diff` is the difference amount as mentioned above.

**Comments:**

One possible function is the following.

**Code: Cursor**

```
CREATE OR REPLACE FUNCTION borrow_diff ()
RETURNS TABLE (email VARCHAR (256), diff BIGINT) AS $$
DECLARE
  curs CURSOR FOR (
    SELECT s.email AS email, COUNT(l.book) AS count
    FROM student s LEFT JOIN loan l
      ON s.email = l.borrower
    GROUP BY s.email
    ORDER BY count DESC
  );
  prev BIGINT;
  rec RECORD;
BEGIN
  prev := -1;
  OPEN curs;                          -- (1) Open cursor

  LOOP
    FETCH NEXT FROM curs INTO rec;    -- (2) Fetch from cursor
    EXIT WHEN NOT FOUND;              -- (3a) Not found: EXIT

    email := rec.email;               -- (3b) Found: COMPUTE
    IF prev = -1
    THEN
      diff := NULL;
      prev := rec.count;
      RETURN NEXT;
    ELSE
      diff := prev - rec.count;
      prev := rec.count;
      RETURN NEXT;
    END IF;
  END LOOP;

  CLOSE curs;                         -- (4) Close cursor
  RETURN;
END;
$$ LANGUAGE plpgsql;
```

The basic idea is to record the number of books borrowed by the previous row in the variable `prev`. We then use this value to compute the difference for the current row. Do not forget to update the value of `prev`.

**Note:** We use `BIGINT` for `diff` and `prev` because they come from `COUNT()` which returns a `BIGINT`. We also use `FETCH NEXT` although `FETCH` is sufficient (as `NEXT` is the default fetch direction) to make the code more readable.

We can easily obtain alternative solutions by using a simpler SQL query for the cursor, at the cost of more complex loop logic. In general, this will be the trade-off when writing stored functions: how comfortable we are in writing SQL queries, versus in writing imperative programs.

**Comments:**

Stored functions/procedures follow a different paradigm from SQL queries. In SQL queries, we describe **what** is the data we want to have without specifying how to compute it. On the other hand, in stored functions, we describe **how** to compute the data.

The language `plpgsql` has all the necessary constructs to compute any possible computation that can be done with other languages. In particular, we can do selection (e.g., `IF` - `THEN` - `ELSE` - `END IF`) and we can do repetition (e.g., `LOOP` - `END LOOP`).

However, a typical repetition has the structure of `LOOP` - `END LOOP` with the use of explicit `EXIT WHEN NOT FOUND` in the middle. This is closer to the following in a C-like language.

```
while(true) { if (not found) { break; }}
```

`plpgsql` is close to the BASIC language family. A common feature is the use of `END` to close a block as opposed to the use of braces (i.e., `{ ... }`) in C-like languages or indentation in Python.

## References


[1] S. Bressan and B. Catania. *Introduction to Database Systems*. McGraw-Hill Education, 2006. ISBN: 9780071246507.

[2] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. 2nd ed. Prentice Hall Press, 2008. ISBN: 9780131873254.

[3] *PostgreSQL Docs: Errors and Messages*. `https://www.postgresql.org/docs/current/plpgsql-errors-and-messages.html`. [Online; last accessed 2025].

[4] *PostgreSQL Docs: Trapping Errors*. `https://www.postgresql.org/docs/current/plpgsql-control-structures.html#PLPGSQL-ERROR-TRAPPING`. [Online; last accessed 2025].

[5] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. 2nd. USA: McGraw-Hill, Inc., 2000. ISBN: 0072440422.

[6] *W3schools Online Web Tutorials*. `https://www.w3schools.com/`. [Online; last accessed 2025].