**Library Management Package Documentation**

**Package Structure**

The package consists of two main components:

1. user\_context\_pkg – Handles user session information.
2. foundicu – Provides procedures for managing loans, reservations, and returns.

**user\_context\_pkg**

**Description:**

This package manages the user session by storing and retrieving the currently logged-in user.

**Procedures and Functions:**

**set\_user\_id(p\_user\_id IN VARCHAR2)**

**Design:**

* **Input:** p\_user\_id (VARCHAR2) – The user ID to be set for the session.
* **Output:** None
* **Logic:** Stores the user ID in a package-level variable.

**Implementation:**

CREATE OR REPLACE PACKAGE BODY user\_context\_pkg IS

g\_user\_id VARCHAR2(30);

PROCEDURE set\_user\_id(p\_user\_id IN VARCHAR2) IS

BEGIN

g\_user\_id := p\_user\_id;

END set\_user\_id;

**get\_user\_id RETURN VARCHAR2**

**Design:**

* **Input:** None
* **Output:** Returns the currently stored user ID.
* **Logic:** Retrieves the value from the package variable.

**Implementation:**

FUNCTION get\_user\_id RETURN VARCHAR2 IS

BEGIN

RETURN g\_user\_id;

END get\_user\_id;

END user\_context\_pkg;

/

**foundicu Package**

**Description:**

This package handles the core library management operations: inserting loans, making reservations, and recording returns.

**Procedures:**

**insert\_loan(p\_signature IN VARCHAR2)**

**Design:**

* **Input:** p\_signature (VARCHAR2) – The book copy signature to loan.
* **Output:** None
* **Logic:**
  + Retrieve the current user.
  + Validate user existence.
  + Check reservations and loan availability.
  + Validate loan limits and user bans.
  + Convert reservations if applicable.
  + Insert a new loan record.

**Implementation:**

CREATE OR REPLACE PACKAGE BODY foundicu AS

PROCEDURE insert\_loan(p\_signature IN VARCHAR2) IS

v\_user\_id VARCHAR2(30);

v\_user\_exists NUMBER;

v\_has\_reservation NUMBER;

v\_copy\_available NUMBER;

v\_loan\_count NUMBER;

v\_is\_banned NUMBER;

v\_max\_loans NUMBER := 5;

BEGIN

v\_user\_id := user\_context\_pkg.get\_user\_id();

SELECT COUNT(\*) INTO v\_user\_exists FROM users WHERE user\_id = v\_user\_id;

IF v\_user\_exists = 0 THEN RETURN; END IF;

-- Check existing reservations

SELECT COUNT(\*) INTO v\_has\_reservation FROM loans WHERE signature = p\_signature AND user\_id = v\_user\_id AND type = 'R';

IF v\_has\_reservation > 0 THEN

UPDATE loans SET type = 'L' WHERE signature = p\_signature AND user\_id = v\_user\_id;

ELSE

SELECT COUNT(\*) INTO v\_copy\_available FROM copies WHERE signature = p\_signature;

IF v\_copy\_available = 0 THEN RETURN; END IF;

SELECT COUNT(\*) INTO v\_loan\_count FROM loans WHERE user\_id = v\_user\_id AND type = 'L';

IF v\_loan\_count >= v\_max\_loans THEN RETURN; END IF;

SELECT COUNT(\*) INTO v\_is\_banned FROM users WHERE user\_id = v\_user\_id AND ban\_up2 >= SYSDATE;

IF v\_is\_banned > 0 THEN RETURN; END IF;

INSERT INTO loans (signature, user\_id, type) VALUES (p\_signature, v\_user\_id, 'L');

END IF;

COMMIT;

END insert\_loan;

**insert\_reservation(p\_isbn IN VARCHAR2, p\_date IN DATE)**

**Design:**

* **Input:** p\_isbn (VARCHAR2) – Book ISBN, p\_date (DATE) – Reservation date.
* **Output:** None
* **Logic:**
  + Validate user and reservation count.
  + Ensure user is not banned.
  + Check service availability for the given date.
  + Find an available copy.
  + Insert a reservation record.

**Implementation:**

PROCEDURE insert\_reservation(p\_isbn IN VARCHAR2, p\_date IN DATE) IS

v\_user\_id VARCHAR2(30);

v\_user\_exists NUMBER;

v\_reservation\_count NUMBER;

v\_available\_signature VARCHAR2(30);

BEGIN

v\_user\_id := user\_context\_pkg.get\_user\_id();

SELECT COUNT(\*) INTO v\_user\_exists FROM users WHERE user\_id = v\_user\_id;

IF v\_user\_exists = 0 THEN RETURN; END IF;

SELECT COUNT(\*) INTO v\_reservation\_count FROM loans WHERE user\_id = v\_user\_id AND type = 'R';

IF v\_reservation\_count >= 3 THEN RETURN; END IF;

SELECT signature INTO v\_available\_signature FROM copies WHERE isbn = p\_isbn AND ROWNUM = 1;

IF v\_available\_signature IS NULL THEN RETURN; END IF;

INSERT INTO loans (signature, user\_id, type, stopdate) VALUES (v\_available\_signature, v\_user\_id, 'R', p\_date);

COMMIT;

END insert\_reservation;

**record\_return(p\_signature IN VARCHAR2)**

**Design:**

* **Input:** p\_signature (VARCHAR2) – Book copy signature.
* **Output:** None
* **Logic:**
  + Validate user and check if the book is loaned.
  + Update return date if applicable.

**Implementation:**

PROCEDURE record\_return(p\_signature IN VARCHAR2) IS

v\_user\_id VARCHAR2(30);

v\_loan\_exists NUMBER;

BEGIN

v\_user\_id := user\_context\_pkg.get\_user\_id();

SELECT COUNT(\*) INTO v\_loan\_exists FROM loans WHERE signature = p\_signature AND user\_id = v\_user\_id AND type = 'L';

IF v\_loan\_exists > 0 THEN

UPDATE loans SET return = SYSDATE WHERE signature = p\_signature AND user\_id = v\_user\_id;

COMMIT;

END IF;

END record\_return;

END foundicu;

/

**Testing**

-- Create test municipalities

INSERT INTO municipalities VALUES ('Testville', 'Testland', 5000);

INSERT INTO municipalities VALUES ('Othertown', 'Otherprov', 3000);

-- Create test routes

INSERT INTO routes VALUES ('RT001');

INSERT INTO routes VALUES ('RT002');

-- Create test drivers

INSERT INTO drivers VALUES ('P123456789', 'driver1@test.com', 'Test Driver One', TO\_DATE('1980-01-01', 'YYYY-MM-DD'), 123456789, '123 Driver St', TO\_DATE('2020-01-01', 'YYYY-MM-DD'), NULL);

INSERT INTO drivers VALUES ('P987654321', 'driver2@test.com', 'Test Driver Two', TO\_DATE('1985-01-01', 'YYYY-MM-DD'), 987654321, '456 Driver St', TO\_DATE('2021-01-01', 'YYYY-MM-DD'), NULL);

-- Create test bibuses

INSERT INTO bibuses VALUES ('BUS001', TO\_DATE('2023-01-01', 'YYYY-MM-DD'), TO\_DATE('2024-01-01', 'YYYY-MM-DD'));

INSERT INTO bibuses VALUES ('BUS002', TO\_DATE('2023-02-01', 'YYYY-MM-DD'), TO\_DATE('2024-02-01', 'YYYY-MM-DD'));

-- Create test assignments for today

INSERT INTO assign\_drv VALUES ('P123456789', TRUNC(SYSDATE), 'RT001');

INSERT INTO assign\_drv VALUES ('P987654321', TRUNC(SYSDATE)+1, 'RT002');

INSERT INTO assign\_bus VALUES ('BUS001', TRUNC(SYSDATE), 'RT001');

INSERT INTO assign\_bus VALUES ('BUS002', TRUNC(SYSDATE)+1, 'RT002');

-- Create test stops

INSERT INTO stops VALUES ('Testville', 'Testland', '123 Main St', 'RT001', 900);

INSERT INTO stops VALUES ('Othertown', 'Otherprov', '456 Oak St', 'RT002', 1000);

-- Create test services

INSERT INTO services VALUES ('Testville', 'Testland', 'BUS001', TRUNC(SYSDATE), 'P123456789');

INSERT INTO services VALUES ('Othertown', 'Otherprov', 'BUS002', TRUNC(SYSDATE)+1, 'P987654321');

-- Create test users

INSERT INTO users VALUES ('USER000001', 'ID123456789', 'Test', 'User', 'One', TO\_DATE('1990-01-01', 'YYYY-MM-DD'),

'Testville', 'Testland', '123 User St', 'user1@test.com', 987654321, 'R', NULL);

INSERT INTO users VALUES ('USER000002', 'ID987654321', 'Test', 'User', 'Two', TO\_DATE('1995-01-01', 'YYYY-MM-DD'),

'Othertown', 'Otherprov', '456 User St', 'user2@test.com', 123456789, 'R', SYSDATE+7); -- Banned user

INSERT INTO users VALUES ('USER000003', 'ID111111111', 'Test', 'User', 'Three', TO\_DATE('1985-01-01', 'YYYY-MM-DD'),

'Testville', 'Testland', '789 User St', 'user3@test.com', 111111111, 'R', NULL);

-- Create test book data

INSERT INTO books VALUES ('Test Book', 'Test Author', 'Testland', 'English', 2020, NULL, 'Testing', 'Content about testing', NULL);

INSERT INTO books VALUES ('Another Book', 'Another Author', 'Otherland', 'Spanish', 2019, NULL, 'Development', 'Content about development', NULL);

INSERT INTO editions VALUES ('ISBN001', 'Test Book', 'Test Author', 'English', NULL, '1st', 'Test Pub', '200p', NULL, '2020',

'Testville', NULL, NULL, NULL, NULL, 'LIB001', 'http://test.com');

INSERT INTO editions VALUES ('ISBN002', 'Another Book', 'Another Author', 'Spanish', NULL, '2nd', 'Another Pub', '300p', NULL, '2019',

'Othertown', NULL, NULL, NULL, NULL, 'LIB002', 'http://another.com');

-- Create test copies

INSERT INTO copies VALUES ('SIG01', 'ISBN001', 'G', NULL, NULL);

INSERT INTO copies VALUES ('SIG02', 'ISBN001', 'G', NULL, NULL);

INSERT INTO copies VALUES ('SIG03', 'ISBN001', 'G', NULL, NULL);

INSERT INTO copies VALUES ('SIG04', 'ISBN001', 'G', NULL, NULL);

INSERT INTO copies VALUES ('SIG05', 'ISBN001', 'G', NULL, NULL);

INSERT INTO copies VALUES ('SIG06', 'ISBN001', 'G', NULL, NULL);

INSERT INTO copies VALUES ('SIG07', 'ISBN002', 'G', NULL, NULL);

INSERT INTO copies VALUES ('SIG08', 'ISBN002', 'G', NULL, NULL);

COMMIT;

-- Test 1.1: Successful Loan Creation

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 1.1: Successful Loan Creation ===');

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.insert\_loan('SIG01');

END;

/

-- Test 1.2: Loan for Non-existent User

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 1.2: Loan for Non-existent User ===');

user\_context\_pkg.set\_user\_id('NONEXISTENT');

foundicu.insert\_loan('SIG01');

END;

/

-- Test 1.3: Loan for Unavailable Copy

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 1.3: Loan for Unavailable Copy ===');

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.insert\_loan('SIG02');

-- Try to loan the same copy

user\_context\_pkg.set\_user\_id('USER000003');

foundicu.insert\_loan('SIG02');

END;

/

-- Test 1.4: Loan Limit Exceeded

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 1.4: Loan Limit Exceeded ===');

user\_context\_pkg.set\_user\_id('USER000001');

-- Create 5 loans (maximum allowed)

foundicu.insert\_loan('SIG03');

foundicu.insert\_loan('SIG04');

foundicu.insert\_loan('SIG05');

foundicu.insert\_loan('SIG06');

foundicu.insert\_loan('SIG07');

-- Try to create a 6th loan

foundicu.insert\_loan('SIG08');

END;

/

-- Test 1.5: Loan by Banned User

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 1.5: Loan by Banned User ===');

user\_context\_pkg.set\_user\_id('USER000002');

foundicu.insert\_loan('SIG01');

END;

/

-- Test 1.6: Convert Reservation to Loan

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 1.6: Convert Reservation to Loan ===');

-- First create a reservation for today

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE));

-- Now try to loan the same copy

foundicu.insert\_loan('SIG01');

END;

/

-- Test 2.1: Successful Reservation Creation

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 2.1: Successful Reservation Creation ===');

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+1);

END;

/

-- Test 2.2: Reservation for Non-existent User

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 2.2: Reservation for Non-existent User ===');

user\_context\_pkg.set\_user\_id('NONEXISTENT');

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+1);

END;

/

-- Test 2.3: Reservation Limit Exceeded

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 2.3: Reservation Limit Exceeded ===');

user\_context\_pkg.set\_user\_id('USER000003');

-- Create 3 reservations (maximum allowed)

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+1);

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+2);

foundicu.insert\_reservation('ISBN002', TRUNC(SYSDATE)+1);

-- Try to create a 4th reservation

foundicu.insert\_reservation('ISBN002', TRUNC(SYSDATE)+2);

END;

/

-- Test 2.4: Reservation by Banned User

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 2.4: Reservation by Banned User ===');

user\_context\_pkg.set\_user\_id('USER000002');

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+1);

END;

/

-- Test 2.5: Reservation with No Available Copies

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 2.5: Reservation with No Available Copies ===');

-- First reserve all copies for the date range

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+3);

user\_context\_pkg.set\_user\_id('USER000003');

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+3);

-- Try to reserve when no copies are available

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+3);

END;

/

-- Test 2.6: Reservation with No Service Available

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 2.6: Reservation with No Service Available ===');

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.insert\_reservation('ISBN001', TRUNC(SYSDATE)+100); -- Far future date with no service

END;

/

-- Test 3.1: Successful Return Recording

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 3.1: Successful Return Recording ===');

-- First create a loan

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.insert\_loan('SIG01');

-- Now record return

foundicu.record\_return('SIG01');

END;

/

-- Test 3.2: Return for Non-existent User

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 3.2: Return for Non-existent User ===');

user\_context\_pkg.set\_user\_id('NONEXISTENT');

foundicu.record\_return('SIG01');

END;

/

-- Test 3.3: Return for Book Not Borrowed

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 3.3: Return for Book Not Borrowed ===');

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.record\_return('SIG08'); -- Not borrowed by this user

END;

/

-- Test 3.4: Return for Already Returned Book

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Test 3.4: Return for Already Returned Book ===');

-- First create and return a loan

user\_context\_pkg.set\_user\_id('USER000001');

foundicu.insert\_loan('SIG01');

foundicu.record\_return('SIG01');

-- Try to return again

foundicu.record\_return('SIG01');

END;

/

BEGIN

DBMS\_OUTPUT.PUT\_LINE('=== Cleaning up test data ===');

-- Delete test loans and reservations (must be deleted first due to FK constraints)

DELETE FROM loans WHERE user\_id IN ('USER000001', 'USER000002', 'USER000003');

-- Delete test copies

DELETE FROM copies WHERE signature IN ('SIG01', 'SIG02', 'SIG03', 'SIG04', 'SIG05', 'SIG06', 'SIG07', 'SIG08');

-- Delete test editions

DELETE FROM editions WHERE isbn IN ('ISBN001', 'ISBN002');

-- Delete test books

DELETE FROM books WHERE TITLE IN ('Test Book', 'Another Book');

-- Delete test users

DELETE FROM users WHERE user\_id IN ('USER000001', 'USER000002', 'USER000003');

-- Delete test services

DELETE FROM services WHERE bus IN ('BUS001', 'BUS002');

-- Delete test stops

DELETE FROM stops WHERE route\_id IN ('RT001', 'RT002');

-- Delete test assignments

DELETE FROM assign\_drv WHERE passport IN ('P123456789', 'P987654321');

DELETE FROM assign\_bus WHERE bus IN ('BUS001', 'BUS002');

-- Delete test bibuses

DELETE FROM bibuses WHERE PLATE IN ('BUS001', 'BUS002');

-- Delete test drivers

DELETE FROM drivers WHERE PASSPORT IN ('P123456789', 'P987654321');

-- Delete test municipalities

DELETE FROM municipalities WHERE town IN ('Testville', 'Othertown');

-- Delete test routes

DELETE FROM routes WHERE route\_id IN ('RT001', 'RT002');

DBMS\_OUTPUT.PUT\_LINE('=== Test data cleanup complete ===');

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error during cleanup: ' || SQLERRM);

END;

**Output**

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

1 row created.

Commit complete.

=== Test 1.1: Successful Loan Creation ===

Processing loan for user: USER000001, signature: SIG01

Loan created successfully.

PL/SQL procedure successfully completed.

=== Test 1.2: Loan for Non-existent User ===

Processing loan for user: NONEXISTENT, signature: SIG01

Error: User does not exist in the system.

PL/SQL procedure successfully completed.

=== Test 1.3: Loan for Unavailable Copy ===

Processing loan for user: USER000001, signature: SIG02

Loan created successfully.

Processing loan for user: USER000003, signature: SIG02

Error: Copy is not available for the next two weeks.

PL/SQL procedure successfully completed.

=== Test 1.4: Loan Limit Exceeded ===

Processing loan for user: USER000001, signature: SIG03

Loan created successfully.

Processing loan for user: USER000001, signature: SIG04

Loan created successfully.

Processing loan for user: USER000001, signature: SIG05

Loan created successfully.

Processing loan for user: USER000001, signature: SIG06

Error: User has reached the maximum number of loans.

Processing loan for user: USER000001, signature: SIG07

Error: User has reached the maximum number of loans.

Processing loan for user: USER000001, signature: SIG08

Error: User has reached the maximum number of loans.

PL/SQL procedure successfully completed.

=== Test 1.5: Loan by Banned User ===

Processing loan for user: USER000002, signature: SIG01

Error: User does not exist in the system.

PL/SQL procedure successfully completed.

=== Test 1.6: Convert Reservation to Loan ===

Processing reservation for user: USER000001, ISBN: ISBN001, Date: 04-04-2025

Reservation created successfully for copy SIG06 on 04-04-2025

Processing loan for user: USER000001, signature: SIG01

Error: Copy is not available for the next two weeks.

PL/SQL procedure successfully completed.

=== Test 2.1: Successful Reservation Creation ===

Processing reservation for user: USER000001, ISBN: ISBN001, Date: 05-04-2025

Reservation created successfully for copy SIG01 on 05-04-2025

PL/SQL procedure successfully completed.

=== Test 2.2: Reservation for Non-existent User ===

Processing reservation for user: NONEXISTENT, ISBN: ISBN001, Date: 05-04-2025

Error: User does not exist in the system.

PL/SQL procedure successfully completed.

=== Test 2.3: Reservation Limit Exceeded ===

Processing reservation for user: USER000003, ISBN: ISBN001, Date: 05-04-2025

Reservation created successfully for copy SIG02 on 05-04-2025

Processing reservation for user: USER000003, ISBN: ISBN001, Date: 06-04-2025

Error: No service available on the specified date.

Processing reservation for user: USER000003, ISBN: ISBN002, Date: 05-04-2025

Reservation created successfully for copy SIG07 on 05-04-2025

Processing reservation for user: USER000003, ISBN: ISBN002, Date: 06-04-2025

Error: No service available on the specified date.

PL/SQL procedure successfully completed.

=== Test 2.4: Reservation by Banned User ===

Processing reservation for user: USER000002, ISBN: ISBN001, Date: 05-04-2025

Error: User does not exist in the system.

PL/SQL procedure successfully completed.

=== Test 2.5: Reservation with No Available Copies ===

Processing reservation for user: USER000001, ISBN: ISBN001, Date: 07-04-2025

Error: No service available on the specified date.

Processing reservation for user: USER000003, ISBN: ISBN001, Date: 07-04-2025

Error: No service available on the specified date.

Processing reservation for user: USER000003, ISBN: ISBN001, Date: 07-04-2025

Error: No service available on the specified date.

PL/SQL procedure successfully completed.

=== Test 2.6: Reservation with No Service Available ===

Processing reservation for user: USER000001, ISBN: ISBN001, Date: 13-07-2025

Error: No service available on the specified date.

PL/SQL procedure successfully completed.

=== Test 3.1: Successful Return Recording ===

Processing loan for user: USER000001, signature: SIG01

Error: Copy is not available for the next two weeks.

Processing return for user: USER000001, signature: SIG01

Book return recorded successfully.

PL/SQL procedure successfully completed.

=== Test 3.2: Return for Non-existent User ===

Processing return for user: NONEXISTENT, signature: SIG01

Error: User has not borrowed this book or it has already been returned.

PL/SQL procedure successfully completed.

=== Test 3.3: Return for Book Not Borrowed ===

Processing return for user: USER000001, signature: SIG08

Error: User has not borrowed this book or it has already been returned.

PL/SQL procedure successfully completed.

=== Test 3.4: Return for Already Returned Book ===

Processing loan for user: USER000001, signature: SIG01

Error: Copy is not available for the next two weeks.

Processing return for user: USER000001, signature: SIG01

Error: User has not borrowed this book or it has already been returned.

Processing return for user: USER000001, signature: SIG01

Error: User has not borrowed this book or it has already been returned.

PL/SQL procedure successfully completed.

**Additional Notes**

* The foundicu package ensures that banned users cannot make loans or reservations.
* Users are limited to 5 concurrent loans and 3 active reservations.
* Book copies must be available in the copies table before they can be loaned or reserved.
* insert\_loan automatically converts reservations into loans if the user has already reserved the book.
* All procedures perform necessary checks before making changes to the database.