



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

200
1822 - 2022

Advanced MySQL Concepts

*Man Geni Addysg Uwch yng Nghymru
Where Higher Education Began in Wales*

www.uwtsd.ac.uk

Learning Objectives

- Recap: Transactions
- Understand what stored procedures are
- Learn syntax and basic creation.
- Apply them in practical examples.



Quick Recap: Transactions

- **ACID properties:** Atomicity, Consistency, Isolation, Durability
START TRANSACTION, COMMIT, ROLLBACK
- **Scenario:** Transfer money between accounts – all-or-nothing operations.

```
BEGIN;
```

```
UPDATE bank_accounts SET balance = balance - 200  
WHERE account_holder = 'Alice';
```

```
UPDATE bank_accounts SET balance = balance + 200  
WHERE account_holder = 'Bob';
```

```
COMMIT;
```

```
-- Rollback;
```

Stored Procedure

- Predefined SQL code blocks executed repeatedly.
- can perform operations like inserting, updating, and deleting data.

Stored Procedure- General Syntax

```
CREATE PROCEDURE procedure_name ( [parameter1  
datatype, parameter2 datatype, ...] )  
BEGIN  
    -- SQL statements  
END
```

Updating copies in book table using stored Procedures

Stored Procedure

```
CREATE TABLE Books (  
    BookID SERIAL PRIMARY KEY,  
    Title VARCHAR(100),  
    CopiesAvailable INT  
);
```

Stored Procedure

```
INSERT INTO Books (Title,CopiesAvailable)
VALUES
('Web Development',5),
('Machine Learning',8),
('Database Systems',3),
('Artificial Intelligence', 7);
```

Updating copies in book table using stored Procedures

```
CREATE OR REPLACE PROCEDURE
UpdateCopies(book_id INT, new_copies INT)
LANGUAGE plpgsql
AS $$
BEGIN
UPDATE Books
SET CopiesAvailable = new_copies
WHERE BookID = book_id;
END;
$$;
```

Stored Procedure

```
CALL UpdateCopies (1, 10) ;
```

-- This updates the book with BookID 1 to have 10 copies.

Stored procedures = reusable transactional logic.

Login to Moodle and complete the tutorial for Stored procedure under tutorial section

Triggers

Lesson Learning Outcomes

- Understand what triggers are and why they are used.
- Identify BEFORE and AFTER triggers.
- Know the meaning of OLD and NEW keywords.
- Apply triggers in a real scenario.

A trigger → *SQL code that runs automatically when data changes.*

It reacts to:

- INSERT
- UPDATE
- DELETE

When Do Triggers Run?

Triggers fire on:

- **INSERT**
- **UPDATE**
- **DELETE**

Each one can be:

- **BEFORE**
- **AFTER**

Total = 6 types

Example: BEFORE INSERT, AFTER UPDATE, etc

Syntax of a Trigger

```
CREATE TRIGGER trigger_name  
BEFORE/AFTER INSERT/UPDATE/DELETE  
ON table_name  
FOR EACH ROW  
BEGIN  
    -- trigger logic  
END;
```

What Triggers Cannot Do

- Cannot take user input
- Cannot be manually executed
- Cannot run SELECT *queries* that return multiple rows

Create a trigger that prevents negative CopiesAvailable in the Books table.

```
CREATE TABLE Books (  
    BookID SERIAL PRIMARY KEY,  
    Title VARCHAR(100),  
    CopiesAvailable INT  
);  
  
INSERT INTO Books (Title, CopiesAvailable) VALUES  
('Database Systems', 5),  
('Python Programming', 3),  
('Web Development', 10);
```

Create the BEFORE UPDATE trigger

This trigger will automatically correct negative values.

Step 1 – Create the trigger function

```
CREATE OR REPLACE FUNCTION
prevent_negative_copies()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    IF NEW.CopiesAvailable < 0 THEN
        NEW.CopiesAvailable := 0;
    END IF;
    RETURN NEW;
END;
$;$;
```

Step 2 – Attach the trigger to the table

```
CREATE TRIGGER  
prevent_negative_copies_trigger  
BEFORE UPDATE ON Books  
FOR EACH ROW  
EXECUTE FUNCTION  
prevent_negative_copies();
```

Test the trigger

Try updating a book with a negative number:

```
UPDATE Books  
SET CopiesAvailable = -5  
WHERE BookID = 1;
```

```
SELECT * FROM Books;
```

BookID	Title	CopiesAvailable
1	Database Systems	5
2	Python Programming	3
3	Web Development	10

Trying to change 5 to -5



BookID	Title	CopiesAvailable
1	Database Systems	0
2	Python Programming	3
3	Web Development	10

What happens with a valid update:

```
UPDATE Books  
SET CopiesAvailable = 2  
WHERE BookID = 1;
```

```
SELECT * FROM Books;
```

Create a trigger that decreases the copies of books when the book is borrowed (a new row is inserted in loan table).

Trigger

```
CREATE TABLE Loans (  
    LoanID INT AUTO_INCREMENT PRIMARY KEY,  
    BookID INT,  
    LoanDate DATE,  
    FOREIGN KEY (BookID) REFERENCES  
Books (BookID)  
);
```

```
CREATE TRIGGER LoanTrigger  
AFTER INSERT ON Loans  
FOR EACH ROW  
EXECUTE FUNCTION decrease_copies();
```

```
-- Trigger after inserting a loan
```

```
CREATE OR REPLACE FUNCTION decrease_copies()  
RETURNS TRIGGER AS $$  
BEGIN  
    UPDATE Books  
    SET CopiesAvailable = CopiesAvailable - 1  
    WHERE BookID = NEW.BookID;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

Trigger

```
INSERT INTO Loans (BookID, MemberID,  
LoanDate) VALUES  
(1, 2, '2025-10-01');
```

Or

```
INSERT INTO Loans (BookID, MemberID,  
LoanDate)  
VALUES (1, 2, CURRENT_DATE);
```

Further Reading :

Chapter 8 Advanced SQL

Database Systems: A Practical Approach to Design, Implementation, and Management Sixth edition by • Thomas Connolly • Carolyn Begg