**LAB 7**

**Chapter 15**

**Exercise 1**:

Write a script that creates and calls a stored procedure named insert\_glaccount. First, code a statement that creates a procedure that adds a new row to the General\_Ledger\_Accounts table in the AP schema. To do that, this procedure should have two parameters, one for each of the two columns in this table. Then, code a CALL statement that tests this procedure. (Note that this table doesn’t allow duplicate account descriptions.)

USE ap;

DROP PROCEDURE IF EXISTS insert\_glaccount;

DELIMITER //

CREATE PROCEDURE insert\_glaccount

(

account\_number\_param INT,

account\_description\_param VARCHAR(50)

)

BEGIN

INSERT INTO general\_ledger\_accounts

VALUES (account\_number\_param, account\_description\_param);

END//

DELIMITER ;

-- Test fail:

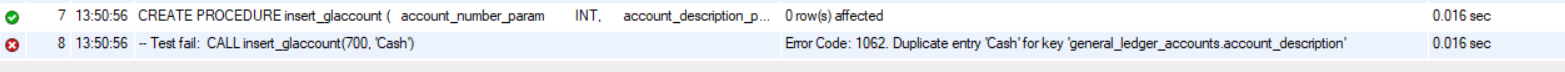
CALL insert\_glaccount(700, 'Cash');

-- Test success:

CALL insert\_glaccount(700, 'Internet Services');

-- Clean up:

DELETE FROM general\_ledger\_accounts WHERE account\_number = 700;

****

**Exercise 2**:

Write a script that creates and calls a stored function named test\_glaccounts\_description. First, create a function that tests whether an account description is already in the General\_Ledger\_Accounts table. To do that, this function should accept one parameter for the account description, and it should return a value of 1 if the account description is in the table or 0 if it isn’t. (Note: If a SELECT statement doesn’t return any data, it raises a NOT FOUND condition that your function can handle.)

USE ap;

DROP FUNCTION IF EXISTS test\_glaccounts\_description;

DELIMITER //

CREATE FUNCTION test\_glaccounts\_description

(

account\_description\_param VARCHAR(50)

)

RETURNS INT

DETERMINISTIC READS SQL DATA

BEGIN

DECLARE account\_description\_var VARCHAR(50);

SELECT account\_description

INTO account\_description\_var

FROM general\_ledger\_accounts

WHERE account\_description = account\_description\_param;

IF account\_description\_var IS NOT NULL THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

END//

DELIMITER ;

-- Test success:

SELECT test\_glaccounts\_description('Book Inventory') AS message;

-- Test fail:

SELECT test\_glaccounts\_description('Fail') AS message;

A screenshot of a computer

Description automatically generated

**Chapter 16**

**Exercise 1**

Create a trigger named invoices\_after\_update. This trigger should insert the old data about the invoice into the Invoices\_Audit table after the row is updated. Then, test this trigger with an appropriate UPDATE statement. If the Invoices\_Audit table doesn’t exist, you can use the code shown in figure 16-3 to create it.

USE ap;

**CREATE TABLE invoices\_audit**

**(**

**vendor\_id INT NOT NULL,**

**invoice\_number VARCHAR(50) NOT NULL,**

**invoice\_total DECIMAL(9,2) NOT NULL,**

**action\_type VARCHAR(50) NOT NULL,**

**action\_date DATETIME NOT NULL**

**)**

DROP TRIGGER IF EXISTS invoices\_after\_update;

DELIMITER //

CREATE TRIGGER invoices\_after\_update

AFTER UPDATE ON invoices

FOR EACH ROW

BEGIN

INSERT INTO invoices\_audit VALUES

(OLD.vendor\_id, OLD.invoice\_number, OLD.invoice\_total,

'UPDATED', NOW());

END//

DELIMITER ;

UPDATE invoices

SET payment\_total = 100

WHERE invoice\_id = 112;

SELECT \* FROM invoices\_audit;

-- clean up

UPDATE invoices

SET payment\_total = 0

WHERE invoice\_id = 112;

-- clean up

-- DELETE FROM invoices\_audit WHERE vendor\_id = 110 LIMIT 100;

