Chapter 8 Problems, Part 3

31. Create a trigger named **trg\_line\_total** to write the LINE\_TOTAL value in the LINE table every time you add a new LINE row. (The LINE\_TOTAL value is the product of the LINE\_UNITS and LINE\_PRICE values.)

DELIMITER //

CREATE OR REPLACE TRIGGER trg\_line\_total

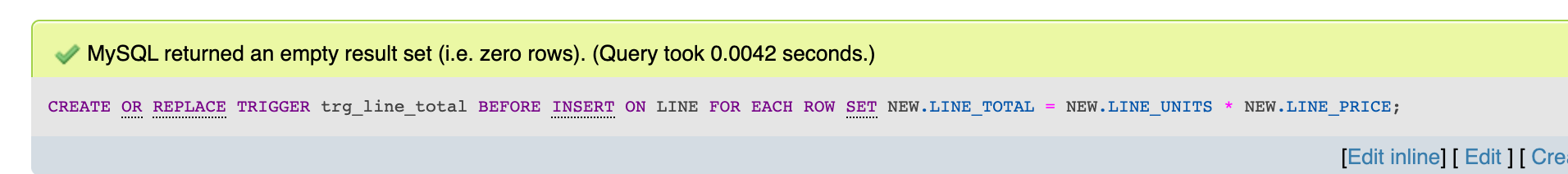
BEFORE INSERT ON LINE

FOR EACH ROW

SET NEW.LINE\_TOTAL = NEW.LINE\_UNITS \* NEW.LINE\_PRICE;

//

DELIMITER ;



1. Create a trigger named **trg\_line\_prod** that automatically updates the quantity on hand for each product sold after a new LINE row is added.

DELIMITER $$

CREATE OR REPLACE TRIGGER trg\_line\_prod

AFTER INSERT ON LINE

FOR EACH ROW

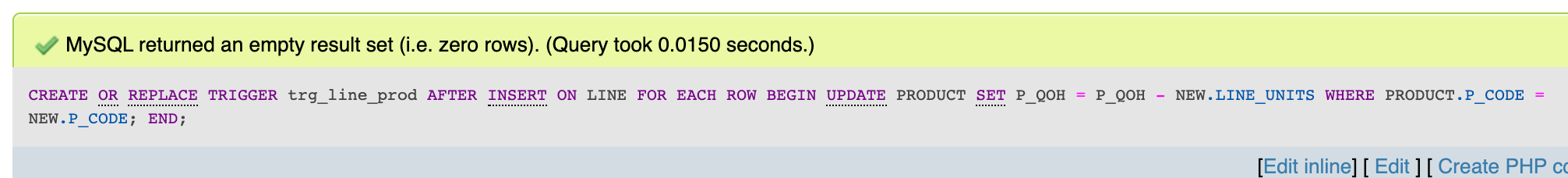
BEGIN

UPDATE PRODUCT

SET P\_QOH = P\_QOH - NEW.LINE\_UNITS

WHERE PRODUCT.P\_CODE = NEW.P\_CODE;

END;

$$

1. Create a stored procedure named **prc\_inv\_amounts** to update the INV\_SUB- TOTAL, INV\_TAX, and INV\_TOTAL. The procedure takes the invoice number as a parameter. The INV\_SUBTOTAL is the sum of the LINE\_TOTAL amounts for the invoice, the INV\_TAX is the product of the INV\_SUBTOTAL and the tax rate (8 percent), and the INV\_TOTAL is the sum of the INV\_SUBTOTAL and the INV\_TAX.

Best guess:

DELIMITER $$

CREATE OR REPLACE PROCEDURE PRC\_INV\_AMOUNTS (IN INV\_IN INT) AS

INV\_SUBT INT := 0;

INV\_INTAX INT := 0;

BEGIN

SELECT SUM(LINE\_TOTAL) INTO INV\_SUBT FROM LINE

WHERE LINE.INV\_NUMBER = INV\_IN;

INV\_INTAX := INV\_SUBT \* .08;

UPDATE INVOICE

SET INV\_SUBTOTAL = INV\_SUBT,

INV\_TAX = INV\_INTAX,

INV\_TOTAL = INV\_SUBT + INV\_INTAX

WHERE INV\_NUMBER = INV\_IN;

END;

$$

1. Create a procedure named **prc\_cus\_balance\_update** that will take the invoice number as a parameter and update the customer balance. (*Hint*: You can use the DECLARE section to define a TOTINV numeric variable that holds the computed invoice total.)

Best guess:

DELIMITER $$

CREATE OR REPLACE PROCEDURE PRC\_CUS\_BAL\_UPDATE (IN INV\_IN INT) AS

CUS\_NUM INT := 0;

BEGIN

SELECT CUS\_CODE INTO CUS\_NUM

FROM INVOICE

WHERE INVOICE.INV\_NUM = INV\_IN;

UPDATE CUSTOMER

SET CUS\_BALANCE = CUS\_BALANCE + (SELECT INV\_TOTAL FROM INVOICE WHERE INV\_NUMBER = INV\_IN)

WHERE CUS\_CODE = CUS\_NUM;

END;

$$

**CHAPTER 10 PROBLEMS**

1. Suppose that you are a manufacturer of product ABC, which is composed of parts A, B, and C. Each time a new product ABC is created, it must be added to the product inventory, using the PROD\_QOH in a table named PRODUCT. Also, each time the product is created, the parts inventory, using PART\_QOH in a table named PART, must be reduced by one each of parts A, B, and C. The sample database contents are shown in Table P10.1.

Given the preceding information, answer Questions a through e.

* 1. How many database requests can you identify for an inventory update for both

PRODUCT and PART?

For the PRODUCT table, you could do an update request to update the inventory of ABC. You could also complete a request simply to view the inventory number.

For the PART table, you could set a trigger so that every time an ABC item is purchased the PROD\_QOH values for parts A, B, and C are updated accordingly.

* 1. Using SQL, write each database request you identified in Step a.

UPDATE PRODUCT

SET PROD\_QOH = inventory number

WHERE PROD\_CODE = ‘ABC’;

SELECT \* FROM PRODUCT

DELIMITER $$

CREATE OR REPLACE TRIGGER trg\_part\_qoh

AFTER UPDATE ON PRODUCT

FOR EACH ROW

BEGIN

UPDATE PART

SET PART\_QOH = PART\_QOH - 1

WHERE PRODUCT.P\_CODE = ‘ABC’;

END $$

DELIMITER;

* 1. Write the complete transaction(s).

UPDATE PRODUCT

SET PROD\_QOH = PROD\_QOH – 1

WHERE PROD\_CODE = ‘ABC’;

UPDATE PART

SET PART\_QOH = PART\_QOH – 1

WHERE PART\_CODE IN (‘A’, ‘B’, ‘C’)

* 1. Write the transaction log, using Table 10.1 as your template.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRL\_ID | TRX\_NUM | PREV\_PTR | NEXT\_PTR | OP | TABLE | ROW\_ID | ATT | BEFORE | AFTER |
| 100 | 200 | NULL | 101 | UPDATE | START |  |  |  |  |
| 101 | 200 | 100 | 102 | UPDATE | PRODUCT | ABC | PROD\_QOH | 1205 | 1206 |
| 102 | 200 | 101 | 103 | UPDATE | PART | A | PART\_QOH | 567 | 566 |
| 103 | 200 | 102 | 104 | UPDATE | PART | B | PART\_QOH | 98 | 97 |
| 104 | 200 | 103 | 105 | UPDATE | PART | C | PART\_QOH | 549 | 548 |
| 105 | 200 | 104 | NULL | UPDATE | END |  |  |  |  |

* 1. Using the transaction log you created in Step d, trace its use in database recovery.

If the database lost power/connection during any of the PART\_QOH update commands, the transaction would have to be rollbacked in order to ensure the PART\_QOH values are consistent with the PROD\_QOH values.

1. Describe the three most common problems with concurrent transaction execution. Explain how concurrency control can be used to avoid those problems.

The three most common problems are lost updates, uncommitted data, and inconsistent retrievals. In order to avoid these problems, DBMS’s will typically implement a scheduler. The scheduler interleaves database operations to ensure serializability and isolation of transactions. In doing this, the scheduler will use either locking or time stamping methods.

1. What DBMS component is responsible for concurrency control? How is this feature used to resolve conflicts?

The above answer also answers this question. It should also be mentioned that the scheduler will implement a serializable schedule in order to yield the same result as a serial set of operations.