Congratulations! You passed!

TO PASS 80% or higher

Keep Learning

GRADE
100%

Graded Quiz: Test your understanding of Advanced RDB & SQL

LATEST SUBMISSION GRADE

100%

1. When will this trigger get executed? **1 / 1 point**

```
DELIMITER $$

CREATE TRIGGER log_name_change_tgr

  AFTER UPDATE ON customer
    FOR EACH ROW

    BEGIN

    INSERT INTO name_change_log
        SET old_name = OLD.customer_name,
            new_name = NEW.customer_name;

  END $$

DELIMITER ;
```

- (●) After any row in the table **customer** is updated
- ( ) After a new row is inserted into the table **customer**.
- ( ) Before any row in the table **customer** is updated
- ( ) After an existing row is deleted from the table **customer**.

Correct
Correct. The event is "AFTER UPDATE".

2. Which of these SQL server functions is an **aggregate** function? **1 / 1 point**

- ( ) CONCAT( )
- (●) SUM( )
- ( ) FORMAT( )
- ( ) CHAR_LENGTH ( )

Correct
Correct!

3. Why will this common table expression generate an **error** when we try to execute it? **1 / 1 point**

```
WITH customer_cte (customer_name) AS
(
SELECT
    custome_id,
    customer_name
FROM customer
ORDER BY
    customer_name DESC
)

SELECT * from customer_cte;
```

- [x] The number of columns to be generated is one (customer_name) but two columns are selected in the SQL statement. They must agree in numbers.
- [ ] You cannot sort results in descending order when using a common table expression.
- [ ] A **WHERE** clause is missing
- [ ] The keyword **DETERMINISTIC** is missing.

✓ **Correct**
Correct!

4. Which of the followings are not stored entities in the database and must be embedded in other entities or saved as a stand-alone text file? **1 / 1 point**

- [ ] Triggers
- [ ] Stored procedures
- [ ] Server functions
- [x] Common table expressions

✓ **Correct**
Correct!

5. Why would this trigger definition cause an **error** when it is executed? **1 / 1 point**

```
DELIMITER $$

CREATE TRIGGER decrease_inventory_tgr

  DURING UPDATE ON customer_order_line_item

    FOR EACH ROW

    BEGIN

    UPDATE merchandise_item
        SET qoh = qoh - NEW.quantity
        WHERE merchandise_item_id = NEW.merchandise_item_id;

  END $$

DELIMITER ;
```

- [ ] **NEW** values are not available for **UPDATE** triggers.
- [ ] **$$** is not a valid delimiter for SQL statements.
- [x] **DURING UPDATE** is not a valid trigger event.
- [ ] The body of the trigger needs a stop condition.

✓ **Correct**
Correct!

6. What would be the result of executing this SQL statement on this table **student**? **1 / 1 point**

```
SELECT AVG(age) FROM student;
```

Table **student**:

surname	given_name	age
Goodgrade	Peter	5

Excel	Jennifer	4
Starr	Rincon	9

- (●)

AVG(age)
6

- ( )

AVG(age)
4

- ( )

AVG(age)
18

- ( )

AVG(age)
9

✓ **Correct**

Correct!

7. What does this stored function do? **1 / 1 point**

```
DELIMITER $$

CREATE FUNCTION get_final_mark (
  request_student_id CHAR(10)
    )
RETURNS INT

DETERMINISTIC

BEGIN

  RETURN
    (
      SELECT final_mark
      FROM student_mark
      WHERE student_id = request_student_id
    );

END $$

DELIMITER ;
```

- ( ) It will cause an infinite loop and time out.
- (●) It returns the column **final_mark** from the table **student_mark** for the row that has a **student_id** matching the **requested_student_id** that is passed to the function.
- ( ) It returns the column **final_mark** of **ALL** the rows from the table **student_mark.**
- ( ) It will generate an error because a stored function cannot return values to the caller.

✓ **Correct**

Correct!

8. What is the **stop condition** for the following recursive common table expression? **1 / 1 point**

8. What is the **stop condition** for the following recursive common table expression?

1 / 1 point

```
WITH RECURSIVE factorial_cte(n, factorial_of_n) AS
(
SELECT 0, 1

UNION ALL

SELECT n+1, factorial_of_n * (n+1)
FROM factorial_cte
WHERE n < 15
)

select * FROM factorial;
```

- ○ when n = 1
- ◉ when n is >= 15
- ○ when n = 0
- ○ when n >= 1307674368000 (the factorial of 15)

✓ **Correct**

Correct!

9. This stored procedure adds up all the scores of an athlete. How does that score get returned to the caller?

1 / 1 point

```
DELIMITER $$

CREATE PROCEDURE tabulate_scores_stp(
    IN target_athlete_id CHAR(10),
    OUT total_score INT)

BEGIN
    SELECT SUM(score)
    FROM athlete_score
    WHERE athlete_id = target_athlete_id;
END$$

DELIMITER ;
```

- ○ When the caller assigns the result of the procedure call to a user variable like this:

```
SET @total_score = CALL tabulate_score("ATHLETE001");
```

- ○ This stored procedure does not communicate the result with the caller.
- ◉ Through the output parameter **total_score**.
- ○ It inserts a new row in **athlete score** with the total score.

✓ **Correct**

Correct!

10. This stored procedure is supposed to return the total of an order to the caller. The result should have returned 19000 but it is 0. **No error** messages were generated. Why is it **not** working as we intended?

1 / 1 point

```
DELIMITER $$

CREATE PROCEDURE get_order_total_stp(
    IN request_order_id CHAR(10),
    IN total_to_return INT)

BEGIN
    SELECT SUM(subtotal) INTO total_to_return
    FROM customer_order
    WHERE customer_order_id = request_order_id;
END$$

DELIMITER ;

SET @total = 0;
CALL get_order_total_stp("INV1234567", @total);
SELECT @total;
```

Table **customer_order:**

customer_order_id	line_item_number	subtotal
INV1234567	1	3000
INV1234567	2	4000
INV1234567	3	12000

Result:

@total
0

- **The code**

```
1  CALL get_order_total_stp("INV1234567", @total);
```

should have been:

```
1  SET @total = CALL get_order_total_stp("INV1234567", @total);
```

- (●) The parameter **total_to_return** is specified as **IN**, an input parameter, and any changes to its value inside of the body of the stored procedure is not reflected outside of the body.
- ( ) The parameter **total_to_return** should have been declared as a decimal.
- ( ) The equal sign **=** should have been double equal sign **==**

✓ **Correct**
Well done!