5. CREATION AND USAGE OF STORED PROCEDURES AND TRIGGERS

Goal: learn how to use and apply the program objects of a database – stored procedures and triggers, using the MySQL database.

5.1. Create and use stored procedures

Create stored procedures by using the CREATE PROCEDURE operator. Therefore, you can create a stored procedure that implements a selection of data from the contract, supplier_org, and supplier_person tables using the following statement (Figure 5.1).

```
DELIMITER //
CREATE PROCEDURE sp_contract()
BEGIN
   SELECT *
   FROM (contract LEFT JOIN supplier_org ON
      contract.supplier_id = supplier_org.supplier_id)
      LEFT JOIN supplier_person ON
      contract.supplier_id = supplier_person.supplier_id;
END //
```

Use the CALL operator to execute a certain procedure.

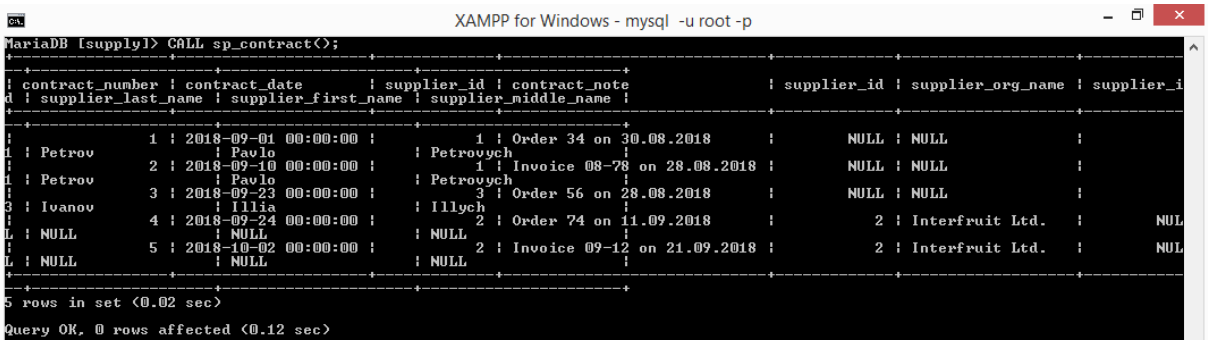

Figure 5.1

To learn about the peculiarities of creating and using procedures with parameters, it is required to create a stored procedure that generates aggregate supply data for a specified interval of dates (Figure 5.2).

```
DELIMITER //
CREATE PROCEDURE sp_contract_total(IN date_from timestamp,
                                   IN date_to timestamp)
BEGIN
   SELECT contract.contract_number, contract.contract_date,
      SUM(supplied.supplied_amount), SUM(supplied.supplied_amount * supplied.supplied_cost)
   FROM contract LEFT JOIN supplied ON contract.contract_number = supplied.contract_number
   WHERE contract.contract_date BETWEEN date_from AND date_to
   GROUP BY contract.contract_number, contract.contract_date;
END //
```

You can call the created procedure using the following statement.

```
CALL sp_contract_total('2018-09-01', '2018-10-31');
```

```
XAMPP for Windows - mysql  -u root -p
MariaDB [supply]> CALL sp_contract_total('2018-09-01', '2018-10-31');
+-----------------+---------------------+-----------------------------+-----------------------------------------------------+
| contract_number | contract_date       | SUM(supplied.supplied_amount) | SUM(supplied.supplied_amount * supplied.supplied_cost) |
+-----------------+---------------------+-----------------------------+-----------------------------------------------------+
|               1 | 2018-09-01 00:00:00 |                          47 |                                            39500.00 |
|               2 | 2018-09-10 00:00:00 |                          24 |                                            11350.00 |
|               3 | 2018-09-23 00:00:00 |                         148 |                                            99600.00 |
|               4 | 2018-09-24 00:00:00 |                         119 |                                            76112.50 |
|               5 | 2018-10-02 00:00:00 |                          64 |                                            45630.00 |
+-----------------+---------------------+-----------------------------+-----------------------------------------------------+
5 rows in set (0.01 sec)
Query OK, 0 rows affected (0.06 sec)
```

Figure 5.2

The next stored procedure is intended to perform various data modification operations for the contract table. This procedure uses the IF operator to control the data flow.

```
DELIMITER //
CREATE PROCEDURE sp_contract_ops(IN op CHAR(1), IN c_num INT, IN c_date TIMESTAMP,
                                 IN s_id INT, IN c_note VARCHAR(100))
BEGIN
   IF op = 'i' THEN
      INSERT INTO contract(contract_date, supplier_id, contract_note)
         VALUES(CURRENT_TIMESTAMP(), s_id, c_note);
   ELSEIF op = 'u' THEN
      UPDATE contract SET contract_date = c_date,
                          supplier_id = s_id,
                          contract_note = c_note
      WHERE contract_number = c_num;
   ELSE
      DELETE FROM contract WHERE contract_number = c_num;
   END IF;
END //
```

The following query allows to create a contract (Figure 5.3).

```
CALL sp_contract_ops('i', 0, '2018-12-16', 2, 'contract inserted');
```

```
XAMPP for Windows - mysql  -u root -p
MariaDB [supply]> CALL sp_contract_ops('i', 0, '2018-12-16', 2, 'contract inserted');
Query OK, 1 row affected (0.01 sec)

MariaDB [supply]> select * from contract;
+-----------------+---------------------+-------------+-----------------------------+
| contract_number | contract_date       | supplier_id | contract_note               |
+-----------------+---------------------+-------------+-----------------------------+
|               1 | 2018-09-01 00:00:00 |           1 | Order 34 on 30.08.2018      |
|               2 | 2018-09-10 00:00:00 |           1 | Invoice 08-78 on 28.08.2018 |
|               3 | 2018-09-23 00:00:00 |           3 | Order 56 on 28.08.2018      |
|               4 | 2018-09-24 00:00:00 |           2 | Order 74 on 11.09.2018      |
|               5 | 2018-10-02 00:00:00 |           2 | Invoice 09-12 on 21.09.2018 |
|               6 | 2018-12-27 13:10:43 |           2 | contract inserted           |
+-----------------+---------------------+-------------+-----------------------------+
6 rows in set (0.00 sec)
```

Figure 5.3

The following query allows to modify the contract (Figure 5.4).

```
CALL sp_contract_ops('u', 6, '2018-12-31', 2, 'contract updated');
```

```
XAMPP for Windows - mysql  -u root -p
MariaDB [supply]> CALL sp_contract_ops('u', 6, '2018-12-31', 2, 'contract updated');
Query OK, 1 row affected (0.01 sec)

MariaDB [supply]> select * from contract;
+-----------------+---------------------+-------------+-----------------------------+
| contract_number | contract_date       | supplier_id | contract_note               |
+-----------------+---------------------+-------------+-----------------------------+
|               1 | 2018-09-01 00:00:00 |           1 | Order 34 on 30.08.2018      |
|               2 | 2018-09-10 00:00:00 |           1 | Invoice 08-78 on 28.08.2018 |
|               3 | 2018-09-23 00:00:00 |           3 | Order 56 on 28.08.2018      |
|               4 | 2018-09-24 00:00:00 |           2 | Order 74 on 11.09.2018      |
|               5 | 2018-10-02 00:00:00 |           2 | Invoice 09-12 on 21.09.2018 |
|               6 | 2018-12-31 00:00:00 |           2 | contract updated            |
+-----------------+---------------------+-------------+-----------------------------+
6 rows in set (0.00 sec)
```

Figure 5.4

The following query allows to delete the contract (Figure 5.5).

```
CALL sp_contract_ops('d', 6, '2018-12-31', 0, '');
```

```
XAMPP for Windows - mysql  -u root -p
MariaDB [supply]> CALL sp_contract_ops('d', 6, '2018-12-31', 0, '');
Query OK, 1 row affected (0.01 sec)

MariaDB [supply]> select * from contract;
+-----------------+---------------------+-------------+-----------------------------+
| contract_number | contract_date       | supplier_id | contract_note               |
+-----------------+---------------------+-------------+-----------------------------+
|               1 | 2018-09-01 00:00:00 |           1 | Order 34 on 30.08.2018      |
|               2 | 2018-09-10 00:00:00 |           1 | Invoice 08-78 on 28.08.2018 |
|               3 | 2018-09-23 00:00:00 |           3 | Order 56 on 28.08.2018      |
|               4 | 2018-09-24 00:00:00 |           2 | Order 74 on 11.09.2018      |
|               5 | 2018-10-02 00:00:00 |           2 | Invoice 09-12 on 21.09.2018 |
+-----------------+---------------------+-------------+-----------------------------+
5 rows in set (0.00 sec)
```

Figure 5.5

5.2. Create and use triggers

Assume that when entering data into the contract table, which stores information on supply contracts, the field contract_date, in which the date of the contract is kept, must be completed. Moreover, if this field is left blank when entering a new contract, the current date must be automatically recorded. This task can be solved by creating a specific trigger using the appropriate command CREATE TRIGGER (Figure 5.6).

```
DELIMITER //
CREATE TRIGGER not_null_date BEFORE INSERT ON contract
FOR EACH ROW
BEGIN
   IF NEW.contract_date IS NULL THEN
      SET NEW.contract_date = CURRENT_TIMESTAMP();
   END IF;
END //
```

To check the trigger, it is required to add a new contract with the next statement.

```
INSERT INTO contract (supplier_id, contract_note) VALUES (1, '');
```

```
XAMPP for Windows - mysql  -u root -p
MariaDB [supply]> INSERT INTO contract (supplier_id, contract_note) VALUES (1, '');
Query OK, 1 row affected (0.01 sec)

MariaDB [supply]> select * from contract;
+-----------------+---------------------+-------------+-----------------------------+
| contract_number | contract_date       | supplier_id | contract_note               |
+-----------------+---------------------+-------------+-----------------------------+
|               1 | 2018-09-01 00:00:00 |           1 | Order 34 on 30.08.2018      |
|               2 | 2018-09-10 00:00:00 |           1 | Invoice 08-78 on 28.08.2018 |
|               3 | 2018-09-23 00:00:00 |           3 | Order 56 on 28.08.2018      |
|               4 | 2018-09-24 00:00:00 |           2 | Order 74 on 11.09.2018      |
|               5 | 2018-10-02 00:00:00 |           2 | Invoice 09-12 on 21.09.2018 |
|               7 | 2018-12-27 13:30:04 |           1 |                             |
+-----------------+---------------------+-------------+-----------------------------+
6 rows in set (0.00 sec)
```

Figure 5.6

The database stores both general supplier information and information that only applies to individuals or legal entities. The simultaneous availability of supplier data in the supplier_org and supplier_person tables is not allowed in terms of business logic. Thus, there is a need for complex control of the relations of referential integrity. To solve this problem we will create a trigger which, when entering the information in the supplier_person table, will control the availability of the code of the respective supplier in the supplier_org table and block the input of the supplier's data as an individual in case if there is already available data on the given supplier as a legal entity (Figure 5.7).

```
DELIMITER //
CREATE TRIGGER check_supplier_org BEFORE INSERT ON supplier_person
FOR EACH ROW
BEGIN
   IF NEW.supplier_id IN (SELECT supplier_id FROM supplier_org) THEN
      SET @message = CONCAT('The person with id ', NEW.supplier_id,
         ' is already stored as the organization!');
      SIGNAL SQLSTATE '45001'
      SET MESSAGE_TEXT = @message;
   END IF;
END //
```

To check the trigger, you must try to add data about supplier 2 (which is already stored in the database as a legal entity) as an individual.

```
INSERT INTO supplier_person VALUES (2, 'Makarov', 'Oleg', 'Petrovych');
```

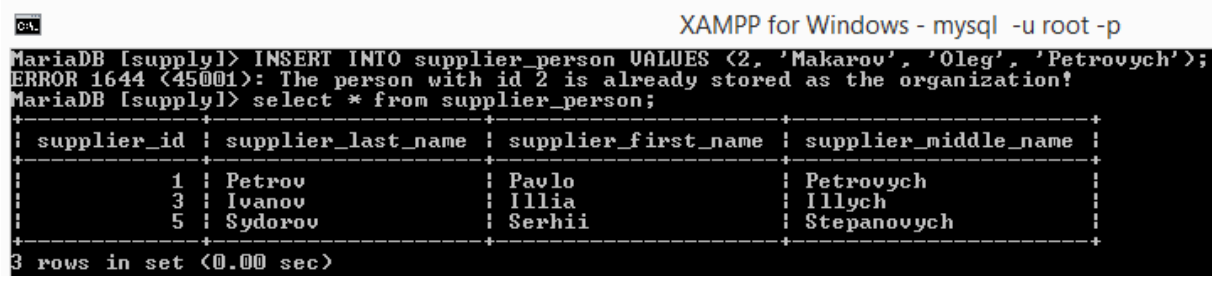

Figure 5.7

To delete stored procedures and triggers, it is required to use the DROP PROCEDURE and DROP TRIGGER operators respectively.

5.3. Questions

1. What is a stored procedure?
2. Name the advantages of stored procedures.
3. What operator is used to create a stored procedure?
4. How to define input or output parameters of a stored procedure?
5. What is the purpose of the IF operator?

6. What is the purpose of BEGIN and END operators?
7. What is a trigger?
8. Name the advantages of triggers.
9. Which operator is used to bind a trigger to a table?
10. Which events related to the table modification operations might be processed with triggers?
11. How to define before or after the table modification operation a trigger should be executed?
12. What are the prefixes NEW and OLD used for?
13. What is the operator SET used for?
14. Which operators are used to remove stored procedures and triggers?