МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

METHODICAL GUIDELINES
for laboratory works

«Learning the basics of working with DBMS MySQL: The main tools for implementing and supporting the business logic of SQL language»

for students of specialties
121 «Software engineering»,
122 «Computer science»,
126 «Information systems and technologies»

Затверджено
редакційно-видавничою
радою університету,
протокол № 3 від 26.10.2022 р.

Харків
НТУ «ХПІ»
2022

Методичні вказівки до виконання лабораторних робіт за темою «Вивчення основ роботи з СУБД MySQL: Основні засоби реалізації та підтримки бізнес-логіки мови SQL» для студентів спеціальностей 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки» та 126 «Інформаційні системи та технології» / уклад. Д.Л. Орловський, А.М. Копп. – Харків : НТУ «ХПІ», 2021. – 25 с. – Англ. мовою.

Укладачі: Д.Л. Орловський

А.М. Копп

Рецензент: В.В. Москаленко

Кафедра програмної інженерії та інтелектуальних технологій управління

CONTENTS

INTRODUCTION

MySQL Database Management System (DBMS) is a freely distributed relational database developed and maintained by Oracle. From the very beginning, MySQL was developed by the Swedish company MySQL AB, which was later acquired by Sun Microsystems, which, in turn, was later acquired by Oracle.

MySQL is distributed under both the GNU General Public License (GPL) and its commercial license. Under the terms of the GPL, software that uses MySQL libraries must also be distributed under the GPL license. For cases where developers do not want to open the source code of their software, a commercial license is provided. The advantage of a commercial license is quality service support. Contrary to Oracle's MySQL licensing policy and to ensure free DBMS status, a fork of MySQL was created and called MariaDB. This database supports high compatibility with MySQL, ensuring the exact correspondence of the programming interface, the so-called API (Application Programming Interface), and MySQL commands.

MySQL is a great solution for small, medium, and sometimes even large software systems. MySQL is also part of the WAMP (Windows, Apache, MySQL, PHP/Perl/Python) and LAMP (Linux, Apache, MySQL, PHP/Perl/Python) web application development stacks. This database is included in many ready-made assemblies of servers designed for web applications, such as XAMPP (which is proposed for use in this laboratory workshop), OpenServer, Denwer, and more. Recently, however, it is because of openness support, server builders and hosting providers are increasingly incorporating MariaDB into WAMP and LAMP stacks.

Typically, MySQL is used as a server accessed by local or remote clients. However, the distribution also contains a library that provides the deployment of an internal server for standalone applications. In this laboratory workshop, we get acquainted with the implementation and maintenance of business logic in the database.

LABORATORY WORK 1. CREATION AND USING 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.

Progress

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 1.1).

```
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.

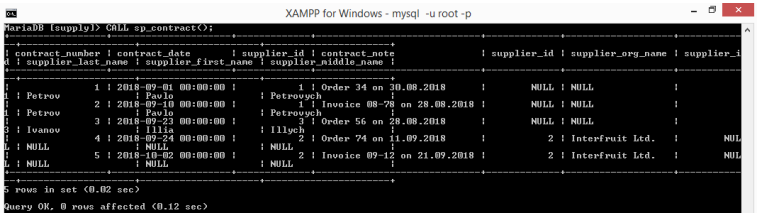

Figure 1.1

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 1.2).

```
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');
```

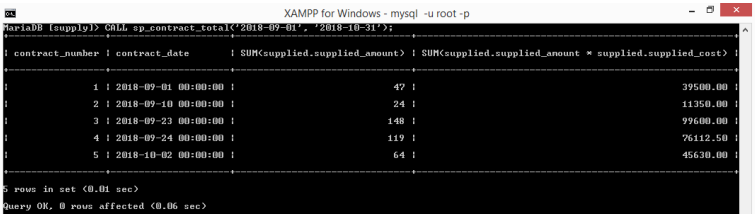

Figure 1.2

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 1.3).

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

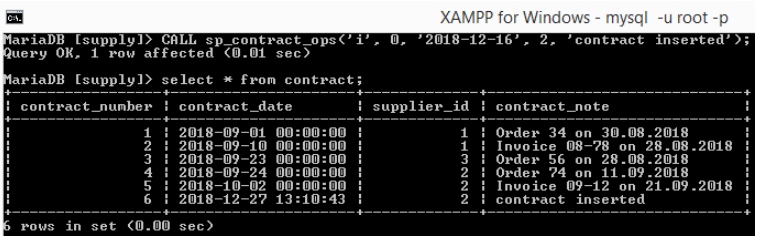

Figure 1.3

The following query allows to modify the contract (figure 1.4).

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

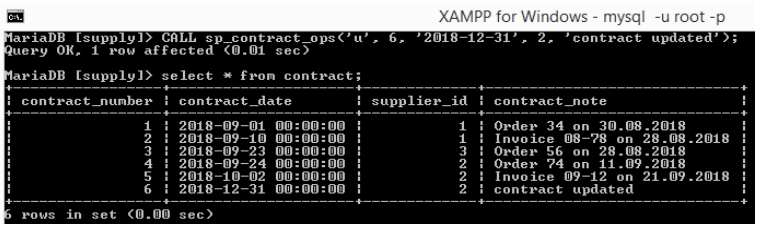

Figure 1.4

The following query allows to delete the contract (figure 1.5).

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

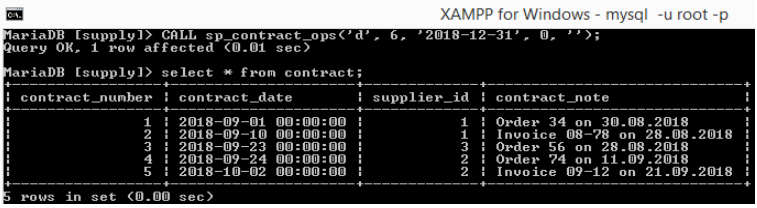

Figure 1.5

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 1.6).

```
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, '');
```

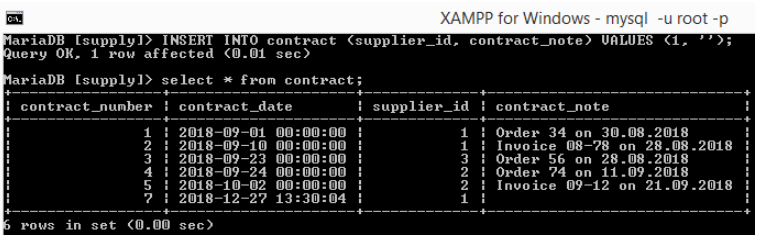

Figure 1.6

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 1.7).

```
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 person.

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

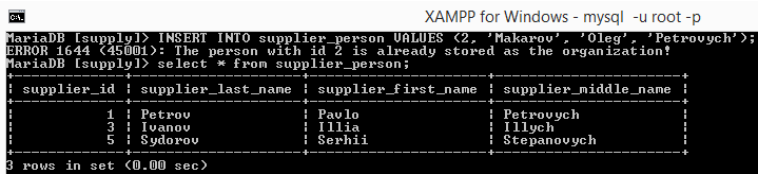

Figure 1.7

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

3. Make a report for the laboratory work

The report should include the main stages of laboratory work and screenshots that demonstrate them.

4. 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?

LABORATORY WORK 2. BASICS OF DATA INTEGRITY CONTROL MECHANISMS

Goal: learn how to use the referential integrity control mechanisms using the MySQL database.

Progress

Warning! It is recommended to create the temporary database using the queries shown in the laboratory work 2. Use this temporary database in this laboratory work.

1. Learn the features of the referential integrity control mechanism NO ACTION

Let's consider the features of the referential integrity mechanism NO ACTION on the example of the relationship between "supplier" and "contract" tables, "supplier" and "supplier_person", "supplier" and "supplier_org". These tables are linked by the supplier_id field. In this regard, the "supplier" table is parent, and the tables "contract", "supplier_org", and "supplier_person" are child tables. In order to learn the features of the mechanism of referential integrity, the following sequence of statements must be executed.

Set the ON DELETE and ON UPDATE parameters that determine the behavior when deleting and updating entries from the parent table.

```
ALTER TABLE contract
DROP FOREIGN KEY contract_ibfk_1;

ALTER TABLE contract
ADD CONSTRAINT contract_ibfk_1 FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE supplier_org
DROP FOREIGN KEY supplier_org_ibfk_1;

ALTER TABLE supplier_org
ADD CONSTRAINT supplier_org_ibfk_1 FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE supplier_person
DROP FOREIGN KEY supplier_person_ibfk_1;

ALTER TABLE supplier_person
ADD CONSTRAINT supplier_person_ibfk_1 FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Assume that due to certain reasons, it is required to remove the supplier with code 4 (figure 2.1).

```
DELETE FROM supplier WHERE supplier_id = 4;
```

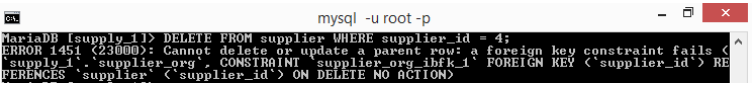

Figure 2.1

Therefore, in order to remove this supplier, you must first delete all the data associated with it. To do this, delete the corresponding entry from the "supplier_org" table and check the availability of contracts with this supplier in the contract table. If there are such contracts, they should also be deleted (it should be kept in mind that there may be a need to remove and a content of these contracts). After that, you need to try to remove the suppliers with code 4 again. If there is no data associated with it, the supplier will be deleted.

Suppose that for some reason there was a need for a supplier with code 5 to change the code to 7 (figure 2.2).

```
UPDATE supplier SET supplier_id = 7 WHERE supplier_id = 5;
```

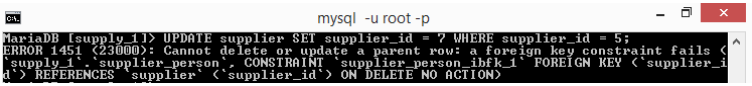

Figure 6.2

Since the contracts with this supplier are not available, the link to it is only in the supplier_person table. After deleting this entry, you must repeat the supplier code change from 5 to 7. Now, this operation must be successful. After that, you need to check the contents of the tables.

2. Learn the features of the referential integrity control mechanism CASCADE

Change the referential integrity mechanisms for links between all the above tables to the CASCADE.

```
ALTER TABLE contract
DROP FOREIGN KEY contract_ibfk_1;

ALTER TABLE contract
ADD CONSTRAINT contract_ibfk_1 FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE supplier_org
DROP FOREIGN KEY supplier_org_ibfk_1;

ALTER TABLE supplier_org
ADD CONSTRAINT supplier_org_ibfk_1 FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE supplier_person
DROP FOREIGN KEY supplier_person_ibfk_1;

ALTER TABLE supplier_person
ADD CONSTRAINT supplier_person_ibfk_1 FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE CASCADE ON UPDATE CASCADE;
```

Suppose that for some reason there was a need for a supplier with code 2 to change the code to 8 (figure 2.3).

```
UPDATE supplier SET supplier_id = 8 WHERE supplier_id = 2;
```

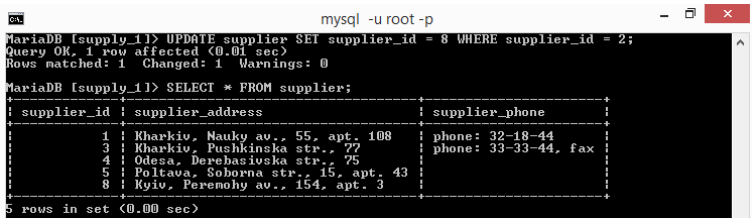

Figure 2.3

Check for the appropriate changes in the “supplier_org” table.

Now assume that this supplier (which now has code 8) must be removed (figure 2.4).

```
DELETE FROM supplier WHERE supplier_id = 8;
```

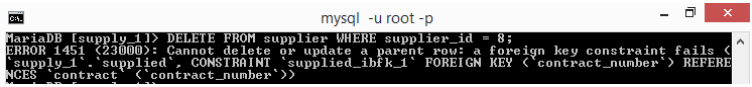

Figure 2.4

Determine the reason why entries were not deleted. Make the necessary changes in the referential integrity mechanisms of the required tables in order to ensure that the necessary data has still been deleted.

3. Learn the features of the referential integrity control mechanism SET NULL

Consider the features of the SET NULL referential integrity mechanism, e.g., for the “supplier” and “contract” tables.

Change the referential integrity mechanisms for links between all the above tables to the SET NULL.

```
ALTER TABLE contract
DROP FOREIGN KEY contract_ibfk_1;

ALTER TABLE contract
MODIFY supplier_id INT NULL;

ALTER TABLE contract
ADD CONSTRAINT contract_ibfk_1 FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE SET NULL ON UPDATE SET NULL;
```

In the “supplier” table, change supplier code 3 to 10. Check the data in the contract table (figure 2.5).

```
UPDATE supplier SET supplier_id = 10 WHERE supplier_id = 3;
```

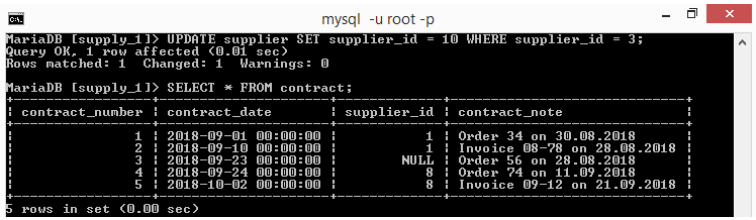

Figure 2.5

Instead of NULL set the value of the supplier code 10 for the contract with number 3.

4. Make a report for the laboratory work

The report should include the main stages of laboratory work and screenshots that demonstrate them.

5. Questions

1. Are the ON DELETE and ON UPDATE commands are necessary for the CRATE TABLE or ALTER TABLE commands?
2. What behavior of a database the ON DELETE command defines?
3. What behavior of a database the ON UPDATE command defines?
4. Which parameters might be defined after the ON DELETE and ON UPDATE statements?
5. Name features of the referential integrity mode CASCADE.
6. Name features of the referential integrity mode SET NULL.
7. Name features of the referential integrity mode NO ACTION.
8. Name features of the referential integrity mode SET DEFAULT.
9. Name features of the referential integrity mode RESTRICT.
10. Why the referential integrity mechanism SET DEFAULT has not been considered in this laboratory work?
11. How to set a certain referential integrity mechanism for a foreign key of a table?
12. Why the supplier_id field of the contract table was modified before the SET NULL mode is set?

13. In which cases it is not recommended to used the referential integrity mechanism CASCADE?

14. Which referential integrity mode is always used by default in the MySQL database in case if it was not defined using the ON DELETE and ON UPDATE statements?

LABORATORY WORK 3. WORK WITH TRANSACTIONS

Goal: learn the basics of the transactional mechanism using the MySQL database.

Progress

Warning! It is recommended to create the temporary database using the queries shown in the laboratory work 2. Use this temporary database in this laboratory work.

1. Create a query that demonstrates usage of transactions to add data into a single table

Consider the sequence of actions when creating and using a query that triggers a transaction, a new entry is added to the table, and then the situation of the incorrect or correct completion of the transaction is simulated. The table status is controlled before the transaction begins, during the execution of the transaction and after it is completed. To do this you need to do the following sequence of actions.

```
SELECT supplied.contract_number, supplied.supplied_product, supplied.supplied_cost, supplied.supplied_amount,
   supplier.supplier_address, contract.contract_date
FROM supplied, contract, supplier
WHERE contract.contract_number = supplied.contract_number AND supplier.supplier_id = contract.supplier_id
AND contract.contract_number = 1;

SET AUTOCOMMIT = 0;
START TRANSACTION;
INSERT INTO supplied VALUES (1, 'Vacuum cleaner', 22, 390);

SELECT supplied.contract_number, supplied.supplied_product, supplied.supplied_cost, supplied.supplied_amount,
   supplier.supplier_address, contract.contract_date
FROM supplied, contract, supplier
WHERE contract.contract_number = supplied.contract_number AND supplier.supplier_id = contract.supplier_id
AND contract.contract_number = 1;

ROLLBACK;

SELECT supplied.contract_number, supplied.supplied_product, supplied.supplied_cost, supplied.supplied_amount,
   supplier.supplier_address, contract.contract_date
FROM supplied, contract, supplier
WHERE contract.contract_number = supplied.contract_number AND supplier.supplier_id = contract.supplier_id
AND contract.contract_number = 1;
```

The SELECT queries can output data that illustrates the state of the table before the transaction begins (figure 3.1), during the execution of the transaction, and after the transaction is completed.

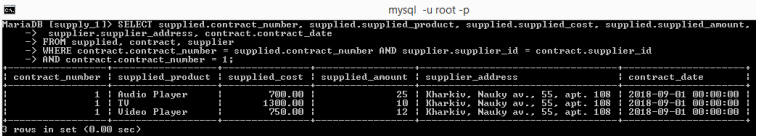

Figure 3.1

As can be seen from the data shown, a new entry in the table appears (figure 3.2), and then disappears (figure 3.3).

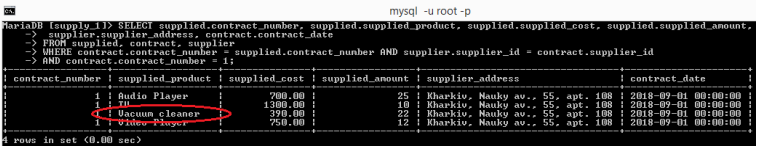

Figure 3.2

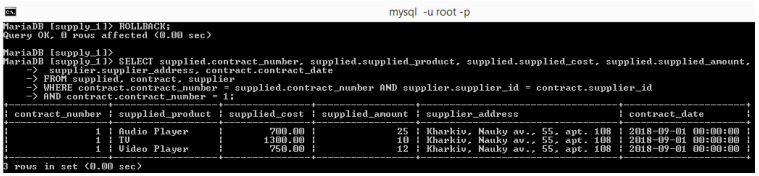

Figure 3.3

Now it is necessary to consider the situation of the correct completion of the transaction. To do this, in the text of the query, you need to change the ROLLBACK statement to COMMIT. Perform the statement and analyze the results.

2. Create a query that demonstrates usage of transactions to add data into multiple tables

Consider the sequence of actions when creating and using a query that triggers a transaction, and then creates a new supplier, a contract for the supply is concluded with that supplier, the products delivered under this contract. The situation of the incorrect or correct completion of the transaction is simulated. The status of the tables is controlled before the transaction begins, in the process of executing the transaction and after the transaction is completed. To do this you need to do the following sequence of actions.

```
SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied;

SET AUTOCOMMIT = 0;
START TRANSACTION;
INSERT INTO supplier (supplier_id, supplier_address, supplier_phone)
   VALUES (6, 'Kyiv, Velyka Vasylkivska st., 55', '');
INSERT INTO contract (contract_date, supplier_id, contract_note)
   VALUES ('2018-12-12', 6, '');
INSERT INTO supplied VALUES (6, 'Vacuum cleaner', 22, 390);
INSERT INTO supplied VALUES (6, 'Coffee machine', 33, 90);

SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied;

ROLLBACK;

SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied;
```

The SELECT queries allow you to output data that illustrates the status of the tables before the transaction begins, in the process of executing the transaction and after the transaction is completed. As will be seen from the data obtained, new entries in the tables appear and then disappear.

Now it is necessary to consider the situation of the correct completion of the transaction. To do this, change the ROLLBACK statement to COMMIT. Perform the query and analyze the results.

3. Create a query that demonstrates usage of transactions to update data in multiple tables

Consider the sequence of actions when creating and using the query that triggers the transaction, then the data entered in the table are changed when the previous request is executed. The situation of the incorrect or correct completion of the transaction is simulated. The status of the tables is controlled before the transaction begins, in the process of executing the transaction and after the transaction is completed. To do this you need to do the following sequence of actions.

```
ALTER TABLE contract
DROP FOREIGN KEY contract_ibfk_1;

ALTER TABLE contract
ADD CONSTRAINT contract_ibfk_1 FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id) ON DELETE CASCADE ON UPDATE CASCADE;

SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied;

SET AUTOCOMMIT = 0;
START TRANSACTION;
UPDATE supplier SET supplier_id = 22 WHERE supplier_id = 6;
UPDATE supplied SET supplied_cost = supplied_cost * 1.1 WHERE contract_number = 8;

SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied WHERE contract_number = 8;

ROLLBACK;

SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied WHERE contract_number = 8;
```

The SELECT queries allow you to output data that illustrates the status of the tables before the transaction begins, in the process of executing the transaction and after the transaction is completed. As will be seen from the data obtained, updated entries in the tables appear and then disappear.

Now it is necessary to consider the situation of the correct completion of the transaction. To do this, change the ROLLBACK statement to COMMIT. Perform the query and analyze the results.

4. Create a query that demonstrates usage of transactions to delete data from multiple tables

Consider the sequence of actions when creating and using a query that triggers a transaction that removes the supplier that was created when query 2 was executed and whose data was modified by query 3. Considering the CASCADE referential integrity control mechanism the data will be deleted in several tables. The situation of the incorrect or correct completion of the transaction is simulated. The status of the tables is controlled before the transaction begins, in the process of executing the transaction and after the transaction is completed. To do this you need to do the following sequence of actions.

```
ALTER TABLE supplied
DROP FOREIGN KEY supplied_ibfk_1;

ALTER TABLE supplied
ADD CONSTRAINT supplied_ibfk_1 FOREIGN KEY (contract_number) REFERENCES contract(contract_number) ON DELETE CASCADE ON UPDATE CASCADE;

SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied;

SET AUTOCOMMIT = 0;
START TRANSACTION;
DELETE FROM supplier WHERE supplier_id = 22;

SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied;

ROLLBACK;

SELECT * FROM supplier;
SELECT * FROM contract;
SELECT * FROM supplied;
```

The SELECT queries allow you to output data that illustrates the status of the tables before the transaction begins, in the process of executing the transaction and after the transaction is completed. As will be seen from the data obtained, removed entries in the tables disappear and then appear.

Now it is necessary to consider the situation of the correct completion of the transaction. To do this, change the ROLLBACK statement to COMMIT. Perform the query and analyze the results.

5. Make a report for the laboratory work

The report should include the main stages of laboratory work and screenshots that demonstrate them.

6. Questions

1. What is a transaction?
2. Which table types support transactions in the MySQL DBMS?
3. Which table types do not support transactions in the MySQL DBMS?
4. How to disable transaction auto commit in the MySQL DBMS?
5. What operator is used to complete a transaction?
6. What operator is used to cancel changes performed by a transaction?
7. Which command of the MySQL DBMS should be used to enable the transaction auto commit mode for a certain sequence of statements?
8. With which type of tables the SAVEPOINT and ROLLBACK TO SAVEPOINT operators might be used?
9. What is the purpose of the SAVEPOINT and ROLLBACK TO SAVEPOINT operators?
10. Which issues might be caused by parallel execution of transactions?
11. What are the levels of transaction isolation and what problems can each of these levels solve?
12. What table type is used in the MySQL database by default (starting from the version 5.5)?
13. Which transaction isolation levels are supported by InnoDB?
14. Which transaction isolation level is set by default in the InnoDB engine?

LABORATORY WORK 4. USER RIGHTS MANAGEMENT

Goal: learn basics of user accounts and privileges using the MySQL database.

Progress

1. Create new user accounts

The MySQL database management system is a multi-user environment, so different accounts with different privileges can be created to access the "supply" database tables.

The supply manager's account can be provided with privileges to view the "supplier", "supplier_org", "supplier_person" and "contract" tables, add new records, delete and update existing records in the data tables.

Database administrator can be given wider rights (privileges to create tables, editing and deleting existing tables, creating and editing user accounts, etc.).

For a warehouse employee it is enough just to view the "contract" and "supplied" tables, as well as add new records, delete and update already existing records in the "supplied" table.

Consider creating accounts for different database users.

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin123';
CREATE USER 'manager'@'localhost' IDENTIFIED BY 'manager123';
CREATE USER 'storekeeper'@'localhost' IDENTIFIED BY 'storekeeper123';
```

These statements allow to create accounts for the following users:

1) administrator with the password "admin123";

2) supply manager with the password "manager123";

3) warehouse employee with the password "storekeeper123".

The DROP USER statement is used to delete an account. The change of user name in the account is performed with the operator RENAME USER %old_name% TO %new_name%.

Since all user accounts are stored in the "mysql" system databases' "user" table, you can check the creation of the accounts by using the following query (figure 4.1):

```
SELECT Host, User, Password FROM mysql.user;
```

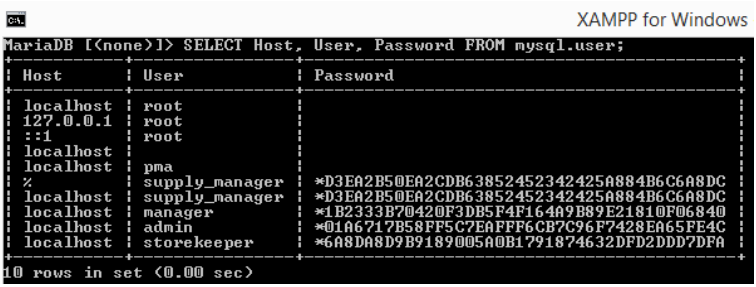

Figure 4.1

2. Assign privileges for created accounts

The above operators allow you to create, delete, and edit accounts, but they do not allow you to change user privileges – to tell the MySQL DBMS, which user is only allowed to read information or to read and edit, and who are given the rights to change the structure of the database and create accounts.

It is required to assign privileges for the created accounts.

```
GRANT ALL ON supply.* TO 'admin'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE ON supply.supplier TO 'manager'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON supply.supplier_org TO 'manager'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON supply.supplier_person TO 'manager'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON supply.contract TO 'manager'@'localhost';
GRANT SELECT ON supply.supplied TO 'manager'@'localhost';
GRANT EXECUTE ON supply.* TO 'manager'@'localhost';

GRANT SELECT, INSERT, UPDATE, DELETE ON supply.supplied TO 'storekeeper'@'localhost';
GRANT SELECT ON supply.contract TO 'storekeeper'@'localhost';
GRANT EXECUTE ON supply.* TO 'storekeeper'@'localhost';
```

The REVOKE operator is used to deprive the user of certain privileges. This operator does not delete accounts, but only cancels the previously granted privileges. Therefore, for the final removal of the account, you must use the operator DROP USER.

Check the privileges of the “admin” account granted with all rights at the “supply” database level using the following query (figure 4.2).

```
SELECT * FROM mysql.db
WHERE Db = 'supply';
```

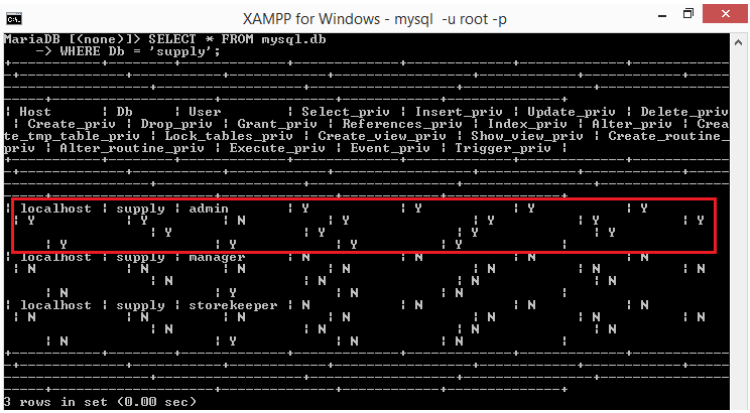

Figure 4.2

Similarly, you can check the privileges of the “manager” and “storekeeper” accounts, for which certain restrictions were encountered with the “supply” database tables (figure 4.3).

```
SELECT Db, User, Table_name, Table_priv FROM mysql.tables_priv
WHERE Db = 'supply';
```

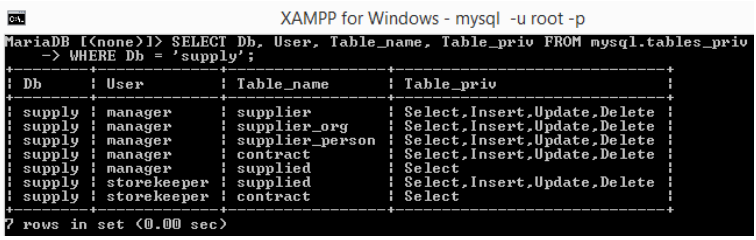

Figure 4.3

In addition, certain users must also have privileges that allow them to use the views contained in the “supply” database. For example, the “manager” user should be given permissions to view the “contract_supplier” and “supplier_info” views, whereas only the “contract_supplier” view for the user “storekeeper” should be available.

3. Make a report for the laboratory work

The report should include the main stages of laboratory work and screenshots that demonstrate them.

4. Questions

1. What is the structure of a user account in the MySQL DBMS?
2. Which components a user account is contains of?
3. What is the purpose of each component of a user account?
4. How to view all user accounts?
5. What command is used to create a user account?
6. What command is used to delete a user account?
7. How to change a name of a user?
8. What statement is used to define certain privileges for a certain user account?
9. What operator is used to cancel given privileges?
10. Which privileges might be defined for a user account?
11. What levels of privileges do you know?
12. How to check the global privileges, database privileges, and table privileges?

REFERENCES

1. SQL Tutorial. – Режим доступу : https://www.w3schools.com/sql/default.asp

2. SQL. – Режим доступу : https://www.tutorialspoint.com/sql/index.htm

3. MySQL. – Режим доступу : https://www.tutorialspoint.com/mysql/index.htm

4. MySQL Documentation. – Режим доступу : https://dev.mysql.com/doc/

5. MySQL Tutorial – Learn MySQL Fast, Easy, and Fun. – Режим доступу : https://www.mysqltutorial.org/

6. MySQL Tutorial for Beginners Learn in 7 Days. – Режим доступу : https://www.guru99.com/mysql-tutorial.html

7. MySQL by Examples for Beginners. – Режим доступу : https://www3.ntu.edu.sg/home/ehchua/programming/sql/MySQL_Beginner.html

8. Drake M. An Introduction to Queries in MySQL. – Режим доступу : https://www.digitalocean.com/community/tutorials/introduction-to-queries-mysql

9. Мулеса О.Ю. Основи мови запитів SQL / О.Ю. Мулеса. – Ужгород, 2015. – 48 с.

10. Балик Н.Р. MySQL: лабораторний практикум / Н.Р. Балик, В.І. Мандзюк. – Тернопіль : Навчальна книга–Богдан, 2008. – 88 с.

11. Балик Н.Р. Бази даних MySQL: Навчальний посібник / Н.Р. Балик, В.І. Мандзюк. – Тернопіль : Навчальна книга–Богдан, 2010. – 160 с.

12. Берко А.Ю. Системи баз даних та знань. Книга 2. Системи управління базами даних та знань: навч. посіб. / А.Ю. Берко, О.М. Верес, В.В. Пасічник. – Львів : «Магнолія-2006», 2013. – 584 с.

Навчальне видання

Методичні вказівки
до виконання лабораторних робіт за темою
«Вивчення основ роботи з СУБД MySQL:
Основні засоби реалізації та підтримки бізнес-логіки мови SQL»

для студентів спеціальностей
121 «Інженерія програмного забезпечення»,
122 «Комп'ютерні науки»,
126 «Інформаційні системи та технології»

Англійською мовою

Укладачі:
ОРЛОВСЬКИЙ Дмитро Леонідович
КОПП Андрій Михайлович

Відповідальний за випуск проф. Гамаюн І.П.
Роботу до видання рекомендував проф. Гамаюн І.П.

План 2022 р., поз. 274

Підп. до друку 26.10.2022.
Гарнітура Times New Roman.
Ум. друк. арк. 0,5.

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2

Самостійне електронне видання