МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт

«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»

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», 2022. – 26 с.

Укладачі: Д.Л. Орловський

А.М. Копп

Рецензент: В.В. Москаленко

Кафедра програмної інженерії та інтелектуальних технологій управління

ЗМІСТ

ВСТУП

Система управління базами даних (СУБД) MySQL є вільно розповсюджуваною реляційною СУБД, розробку та підтримку якої здійснює корпорація Oracle. Від самого початку MySQL розроблялася шведською компанією MySQL AB, яка потім була придбана компанією Sun Microsystems, яка, в свою чергу, згодом була поглинена Oracle.

СУБД MySQL розповсюджується як під ліцензією вільного програмного забезпечення GNU General Public Licence (GPL), так і під власною комерційною ліцензією. За умовами GPL, програмне забезпечення, що використовує бібліотеки MySQL, також повинне розповсюджуватися за ліцензією GPL. Для випадків, коли розробники не бажають відкривати вихідний код свого програмного забезпечення, передбачена комерційна ліцензія. Перевагою комерційної ліцензії є якісна сервісна підтримка. У противагу політиці ліцензування MySQL компанією Oracle та для забезпечення вільного статусу СУБД, було створено відгалуження від MySQL, яке отримало назву MariaDB. Ця СУБД підтримує високу сумісність з MySQL, забезпечуючи точну відповідність інтерфейсу програмування, так званого API (Application Programming Interface), та команд MySQL.

СУБД MySQL є чудовим рішенням для малих, середніх, та інколи навіть великих програмних систем. Також MySQL є складовою частиною стеків розробки веб-застосувань WAMP (Windows, Apache, MySQL, PHP/Perl/Python) та LAMP (Linux, Apache, MySQL, PHP/Perl/Python). Дана СУБД включена до складу багатьох готових збірок серверів, призначених для розробки веб-застосувань, таких як XAMPP (який пропонується використовувати у даному лабораторному практикумі), OpenServer, Denwer тощо. Однак останнім часом, саме через підтримку відкритості, розробники збірок серверів та хостинг провайдери все частіше включають MariaDB у WAMP та LAMP стеки.

Зазвичай MySQL застосовується у ролі сервера, до якого звертаються локальні або віддалені клієнти. Проте дистрибутив містить і бібліотеку, що забезпечує розгортання внутрішнього сервера для автономних застосувань. У даному лабораторному практикумі розглядається знайомство з основними можливостями MySQL щодо реалізації та підтримки бізнес-логіки у базі даних.

ЛАБОРАТОРНА РОБОТА 1. СТВОРЕННЯ ТА ВИКОРИСТАННЯ ЗБЕРЕЖЕНИХ ПРОЦЕДУР ТА ТРИГЕРІВ

Мета роботи: навчитися створювати та застосовувати програмні об’єкти бази даних – збережені процедури та тригери, на прикладі СУБД MySQL.

Хід роботи

1. Створення та використання збережених процедур

Створення збережених процедур реалізується оператором CREATE PROCEDURE. Таким чином, створити збережену процедуру, яка реалізує вибірку даних з таблиць «contract», «supplier_org», «supplier_person», можна за допомогою наступної команди (рисунок 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 //
```

Виклик процедури здійснюється за допомогою оператора CALL.

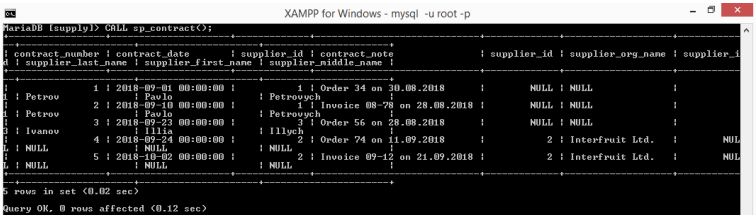

Рисунок 1.1

Для знайомства з особливостями створення та використання процедур з параметрами, необхідно створити збережену процедуру, яка забезпечує формування агрегатних даних за поставками для вказаного інтервалу календарних дат (рисунок 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 //
```

Здійснити виклик створеної процедури можна за допомогою наступного запиту.

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

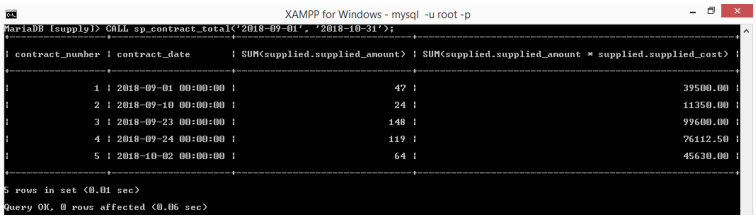

Рисунок 1.2

Наступна збережена процедура призначена для виконання різних операцій модифікації даних для таблиці «contract». Дана процедура використовує оператор умови IF, призначений для управління потоком даних.

```
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 //
```

Наступний запит дозволяє створювати договір (рисунок 1.3).

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

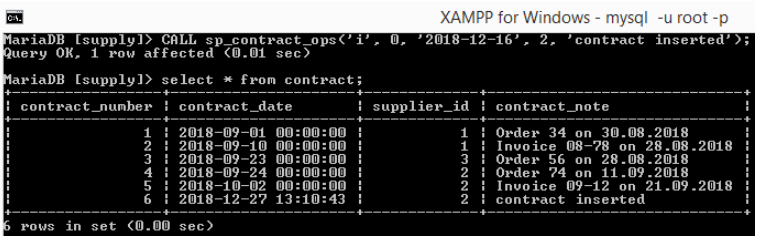

Рисунок 1.3

Наступний запит дозволяє модифікувати договір (рисунок 1.4).

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

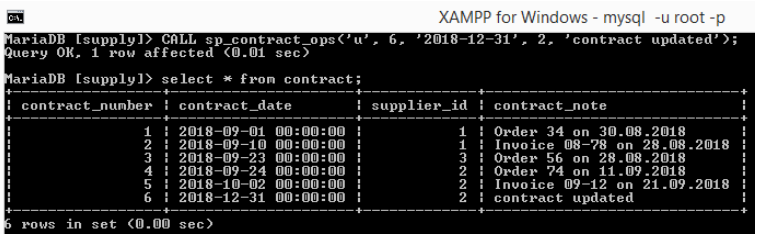

Рисунок 1.4

Наступний запит дозволяє видаляти договір (рисунок 1.5).

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

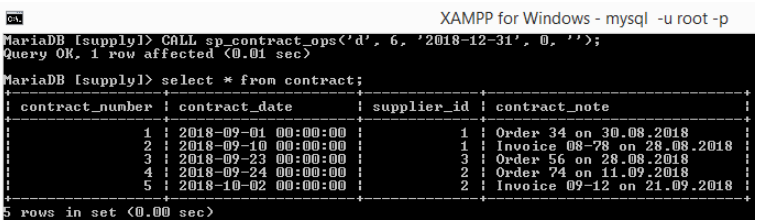

Рисунок 1.5

2. Створення та використання тригерів

Припустимо, що при вводі даних у таблицю «contract», у якій зберігається інформація про договори на постачання продукції, поле

«contract_date», у якому зберігається дата укладення договору, повинне бути обов'язково заповнене. При чому у випадку, якщо при вводі нового договору дане поле залишається незаповненим, в нього повинна бути автоматично записана поточна дата. Дану задачу можна вирішити за допомогою створення певного тригера, використовуючи відповідну команду CREATE TRIGGER (рисунок 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 //
```

Для перевірки роботи тригеру необхідно додати новий договір за допомогою наступного запиту.

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

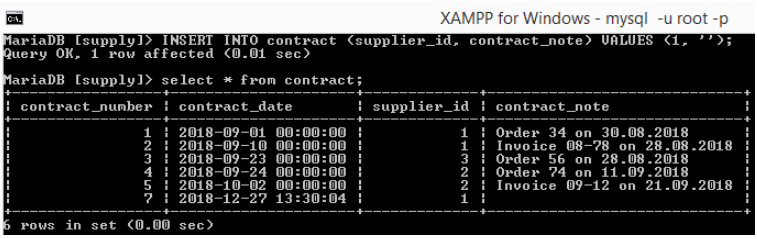

Рисунок 1.6

В базі даних зберігається як загальна інформація про постачальників, так і інформація, яка відноситься тільки до фізичних або юридичних осіб. Одночасна наявність даних про постачальника у таблицях «supplier_org» та «supplier_person» не допускається з точки зору логіки управління бізнесом. Таким чином, виникає необхідність складного контролю відношень посилкової цілісності. Для вирішення даної задачі створимо тригер, який при введенні інформації у таблицю «supplier_person» буде контролювати наявність коду відповідного постачальника у таблиці «supplier_org» та блокувати введення даних про постачальника як про фізичну особу у тому випадку, якщо вже

наявні дані про даного постачальника як про юридичну особу (рисунок 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 //
```

Для перевірки роботи тригеру необхідно спробувати додати дані про постачальника 2 (який вже зберігається у БД в якості юридичної особи) як про фізичну особу.

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

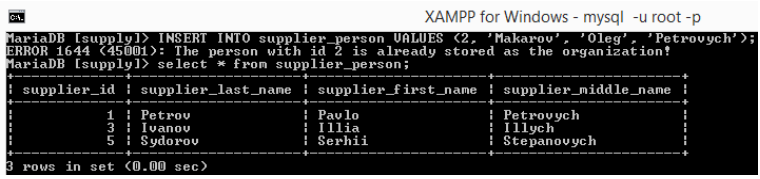

Рисунок 1.7

Для видалення збережених процедур та тригерів необхідно скористатися операторами DROP PROCEDURE та DROP TRIGGER відповідно.

3. Оформити звіт з лабораторної роботи

У звіт включити основні етапи виконання лабораторної роботи та знімки екрану, що їх демонструють.

4. Питання для самоконтролю

1. Що таке збережена процедура?
2. Назвати переваги використання збережених процедур.
3. Який оператор використовується для створення збереженої процедури?

4. Яким чином можна визначити вхідні або вихідні параметри збереженої процедури?

5. Для чого використовується оператор IF?

6. Яке призначення операторів BEGIN та END?

7. Що таке тригер?

8. Назвати переваги використання тригерів.

9. За допомогою якого оператора тригер зв’язується з таблицею?

10. До яких подій, пов’язаних зі зміною вмісту таблиці, можна прив’язати тригер?

11. Яким чином можна визначити до чи після операції зміни вмісту таблиці повинен спрацьовувати тригер?

12. Для чого використовуються префікси NEW та OLD?

13. Яке призначення оператора SET?

14. За допомогою яких операторів виконується видалення процедур та тригерів?

ЛАБОРАТОРНА РОБОТА 2. ОСНОВИ ВИКОРИСТАННЯ ЗАСОБІВ КОНТРОЛЮ ЦІЛІСНОСТІ ДАНИХ

Мета роботи: вивчити основи роботи із засобами контролю посилкової цілісності даних на прикладі СУБД MySQL.

Хід роботи

Увага! Перш ніж перейти до виконання лабораторної роботи, необхідно створити тимчасову базу даних, використовуючи запити, використані у лабораторній роботі 2. В усіх подальших пунктах даної лабораторної роботи передбачається використання тимчасової бази даних.

1. Вивчити особливості роботи механізму посилкової цілісності NO ACTION

Особливості роботи механізму посилкової цілісності NO ACTION розглянемо на прикладі відношень між таблицями «supplier» та «contract», «supplier» та «supplier_person», «supplier» та «supplier_org». Дані таблиці пов’язані між собою за полем «supplier_id». У цьому зв’язку таблиця «supplier» є батьківською, а таблиці «contract», «supplier_org», «supplier_person» – дочірніми. Для вивчення особливостей роботи механізму посилкової цілісності необхідно виконати наступну послідовність дій.

Встановити параметри ON DELETE та ON UPDATE, що визначають поведінку під час видалення та оновлення записів з таблиці-предка.

```
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;
```

Припустимо, що в силу певних причин необхідно видалили постачальника з кодом 4 (рисунок 2.1).

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

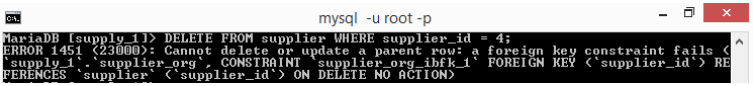

Рисунок 2.1

Таким чином, для того, щоб видалити даного постачальника, необхідно попередньо видалити усі пов'язані з ним дані. Для цього потрібно видалити відповідний запис з таблиці «supplier_org» та перевірити наявність договорів з даним постачальником у таблиці «contract». Якщо такі договори є, їх також потрібно видалити (при цьому необхідно мати на увазі, що може виникнути потреба видалення й вмісту даних договорів). Після цього необхідно спробувати видалити постачальника з кодом 4 знову. Якщо зв'язаних з ним даних немає, постачальник буде видалений.

Припустимо, що в силу певних причин виникла необхідність для постачальника з кодом 5 змінити код на 7 (рисунок 2.2).

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

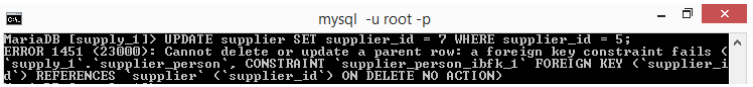

Рисунок 2.2

Оскільки договори з даним постачальником відсутні, посилання на нього є лише в таблиці «supplier_person». Видаливши цей запис, необхідно повторити зміну коду постачальника з 5 на 7. Тепер ця операція повинна пройти успішно. Після цього необхідно перевірити вміст таблиць.

2. Вивчити особливості роботи механізму посилкової цілісності CASCADE

Змінимо механізми посилкової цілісності для зв'язків між усіма розглянутими вище таблицями на 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;
```

Припустимо, що в силу певних причин виникла необхідність для постачальника з кодом 2 змінити код на 8 (рисунок 2.3).

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

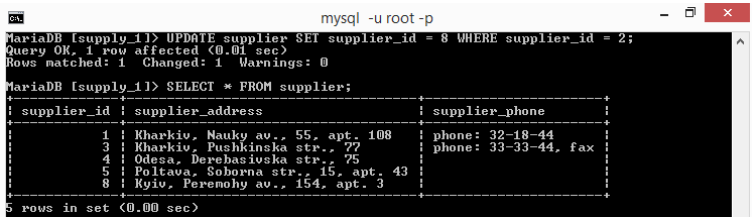

Рисунок 2.3

Перевірити наявність відповідних змін у таблиці «supplier_org».

Тепер припустимо, що даного постачальника (який зараз має код 8) треба видалити (рисунок 2.4).

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

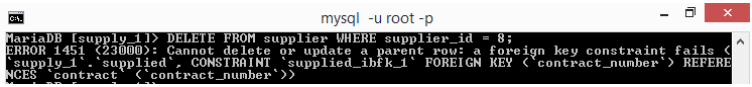

Рисунок 2.4

Визначити причину, через яку записи не були видалені. Внести необхідні зміни у механізми посилкової цілісності необхідних таблиць для того, щоб необхідні дані все ж були видалені.

3. Вивчити особливості роботи механізму посилкової цілісності SET NULL

Особливості механізму посилкової цілісності SET NULL розглянемо на прикладі таблиць «supplier» та «contract».

Змінимо механізми посилкової цілісності для зв’язків між усіма розглянутими вище таблицями на 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;
```

В таблиці «supplier» змінити код постачальника 3 на 10. Перевірити дані в таблиці «contract» (рисунок 2.5).

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

```
mysql  -u root -p
MariaDB [supply_1]> UPDATE supplier SET supplier_id = 10 WHERE supplier_id = 3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [supply_1]> 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 |        NULL | Order 56 on 28.08.2018   |
|               4 | 2018-09-24 00:00:00 |           8 | Order 74 on 11.09.2018   |
|               5 | 2018-10-02 00:00:00 |           8 | Invoice 09-12 on 21.09.2018 |
+-----------------+---------------------+-------------+--------------------------+
5 rows in set (0.00 sec)
```

Рисунок 2.5

Замість NULL встановити значення коду постачальника 10 для договору з номером 3.

4. Оформити звіт з лабораторної роботи

У звіт включити основні етапи виконання лабораторної роботи та знімки екрану, що їх демонструють.

5. Питання для самоконтролю

1. Чи є конструкції ON DELETE та ON UPDATE обов’язковими при формуванні команди CREATE TABLE або ALTER TABLE?
2. Яку поведінку СУБД задає конструкція ON DELETE?
3. Яку поведінку СУБД задає конструкція ON UPDATE?
4. Які параметри можна встановити після конструкцій ON DELETE та ON UPDATE?
5. Назвати особливості механізму посилкової цілісності CASCADE.
6. Назвати особливості механізму посилкової цілісності SET NULL.

7. Назвати особливості механізму посилкової цілісності NO ACTION.

8. Назвати особливості механізму посилкової цілісності SET DEFAULT.

9. Назвати особливості механізму посилкової цілісності RESTRICT.

10. Чому у даній лабораторній роботі не було розглянуто роботу з механізмом посилкової цілісності SET DEFAULT?

11. Яким чином можна встановити той чи інший механізм посилкової цілісності для зовнішнього ключа таблиці?

12. Навіщо перед тим, як встановити механізм SET NULL, була виконана модифікація поля supplier_id таблиці contract?

13. В яких випадках не рекомендується використання механізму посилкової цілісності CASCADE?

14. Який механізм посилкової цілісності завжди використовується за замовченням в СУБД MySQL у випадку, якщо конструкції ON DELETE та ON UPDATE не були визначені?

ЛАБОРАТОРНА РОБОТА 3. РОБОТА З ТРАНЗАКЦІЯМИ

Мета роботи: вивчити основи роботи з механізмом транзакцій на прикладі СУБД MySQL.

Хід роботи

Увага! Перш ніж перейти до виконання лабораторної роботи, необхідно створити тимчасову базу даних, використовуючи запити, використані у лабораторній роботі 2. В усіх подальших пунктах даної лабораторної роботи передбачається використання тимчасової бази даних.

1. Створити запит, що ілюструє роботу механізму транзакцій при додаванні даних в одну таблицю

Розглянемо послідовність дій при створенні та використанні запиту, за допомогою якого запускається транзакція, до таблиці «supplied» додається новий запис, а потім імітується ситуація некоректного або коректного завершення транзакції. Стан таблиці контролюється до початку транзакції, під час виконання транзакції та після її завершення. Для цього необхідно виконати наступну послідовність дій.

```
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;
```

Запити SELECT дозволяють вивести дані, які ілюструють стан таблиці до початку транзакції (рисунок 3.1), в процесі виконання транзакції та після завершення транзакції.

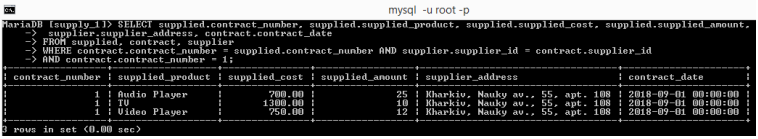

Рисунок 3.1

Як видно з наведених даних, новий запис у таблиці з'являється (рисунок 3.2), а потім зникає (рисунок 3.3).

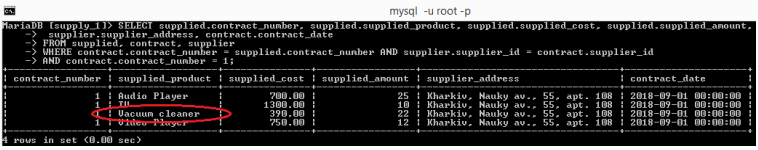

Рисунок 3.2

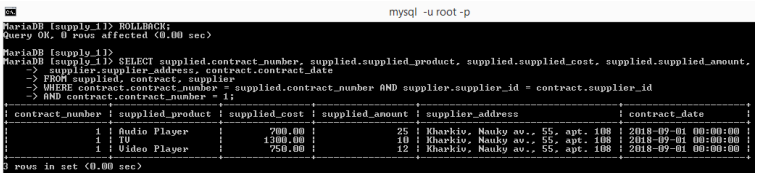

Рисунок 3.3

Тепер необхідно розглянути ситуацію коректного завершення транзакції. Для цього у наведеному тексті запиту необхідно змінити оператор ROLLBACK на COMMIT. Виконати запит та проаналізувати отримані результати.

2. Створити запит, що ілюструє роботу механізму транзакцій при додаванні даних в декілька таблиць

Розглянемо послідовність дій при створенні та використання запиту, за допомогою якого запускається транзакція, а потім створюється новий постачальник, з цим постачальником укладається договір на постачання, за цим договором поставляється продукція. Імітується ситуація некоректного або коректного завершення транзакції. Стан таблиць контролюється до початку транзакції, в процесі виконання транзакції та після завершення транзакції. Для цього необхідно виконати наступну послідовність дій.

```
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;
```

Запити SELECT дозволяють вивести дані, які ілюструють стан таблиць до початку транзакції, в процесі виконання транзакції та після завершення транзакції. Як буде видно з отриманих даних, нові записи у таблицях з'являються, а потім зникають.

Тепер необхідно розглянути ситуацію коректного завершення транзакції. Для цього у наведеному тексті запиту необхідно змінити оператор ROLLBACK на COMMIT. Виконати запит та проаналізувати отримані результати.

3. Створити запит, що ілюструє роботу механізму транзакцій при зміненні даних в декількох таблицях

Розглянемо послідовність дій при створенні та використанні запиту, за допомогою якого запускається транзакція, потім змінюються дані, введені у таблиці при виконанні попереднього запиту. Імітується ситуація некоректного або коректного завершення транзакції. Стан таблиць контролюється до початку транзакції, в процесі виконання транзакції та після завершення транзакції. Для цього необхідно виконати наступну послідовність дій.

```sql
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;
```

Запити SELECT дозволяють вивести дані, які ілюструють стан таблиць до початку транзакції, в процесі виконання транзакції та після завершення транзакції. Як буде видно з отриманих даних, оновлені записи у таблицях з'являються, а потім зникають.

Тепер необхідно розглянути ситуацію коректного завершення транзакції. Для цього у наведеному тексті запиту необхідно змінити оператор ROLLBACK на COMMIT. Виконати запит та проаналізувати отримані результати.

4. Створити запит, що ілюструє роботу механізму транзакцій при видаленні даних з декількох таблиць

Розглянемо послідовність дій при створенні та використанні запиту, за допомогою якого запускається транзакція, в рамках якої видаляється постачальник, який був створений при виконанні запиту 2 та дані якого були змінені при виконанні запиту 3. З урахуванням механізму контролю посилкової цілісності, що використовується (CASCADE), дані будуть видалені у декількох таблицях. Імітується ситуація некоректного або коректного завершення транзакції. Стан таблиць контролюється до початку транзакції, в процесі виконання транзакції та після завершення транзакції. Для цього необхідно виконати наступну послідовність дій.

```
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;
```

Запити SELECT дозволяють вивести дані, які ілюструють стан таблиць до початку транзакції, в процесі виконання транзакції та після завершення транзакції. Як буде видно з отриманих даних, видалені записи у таблицях зникають, а потім з'являються.

Тепер необхідно розглянути ситуацію коректного завершення транзакції. Для цього у наведеному тексті запиту необхідно змінити оператор ROLLBACK на COMMIT. Виконати запит та проаналізувати отримані результати.

5. Оформити звіт з лабораторної роботи

У звіт включити основні етапи виконання лабораторної роботи та знімки екрану, що їх демонструють.

6. Питання для самоконтролю

1. Що таке транзакція?
2. Таблиці яких типів у СУБД MySQL підтримують транзакції?
3. Таблиці яких типів у СУБД MySQL не підтримують транзакції?
4. Яким чином у СУБД MySQL можна відключити режим автоматичного завершення транзакцій?
5. Який оператор використовується для завершення транзакції?
6. Який оператор використовується для відкату змін, виконаних транзакцією?
7. За допомогою якої команди у СУБД MySQL можна включити режим автоматичного завершення транзакцій для окремої послідовності операторів?
8. З таблицями якого типу можуть бути використані оператори SAVEPOINT та ROLLBACK TO SAVEPOINT?
9. Яке призначення операторів SAVEPOINT та ROLLBACK TO SAVEPOINT?

10. З якими проблемами пов’язане паралельне виконання транзакцій?

11. Які існують рівні ізоляції транзакцій та які проблеми кожен з цих рівнів дозволяє вирішити?

12. Який тип таблиць використовується у MySQL за замовченням (починаючи з версії 5.5)?

13. Які рівні ізоляції транзакцій підтримує InnoDB?

14. Який рівень ізоляції транзакцій за замовченням використовується у InnoDB?

ЛАБОРАТОРНА РОБОТА 4. УПРАВЛІННЯ ПРАВАМИ КОРИСТУВАЧІВ

Мета роботи: вивчити основи роботи з обліковими записами та привілеями користувачів на прикладі СУБД MySQL.

Хід роботи

1. Створити нові облікові записи користувачів

Система управління базами даних MySQL є багатокористувацьким середовищем, тому для доступу до таблиць бази даних supply можуть бути створені різні облікові записи з різним рівнем привілеїв.

Обліковому запису менеджера із закупівель можна надати привілеї на перегляд таблиць «supplier», «supplier_org», «supplier_person» та «contract», додавання нових записів, видалення та оновлення вже існуючих записів у даних таблицях.

Адміністратору бази даних «supply» можна надати більш широкі повноваження (можливість створення таблиць, редагування та видалення вже існуючих, створення та редагування облікових записів користувачів тощо).

Для працівника складу достатньо лише перегляду таблиць «contract» та «supplied», а також додавання нових записів, видалення та оновлення вже існуючих записів у таблиці «supplied».

Розглянемо створення облікових записів для різних користувачів бази даних.

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

Даний запит дозволяє створити облікові записи для наступних користувачів:

1) адміністратора з паролем «admin123»;

2) менеджера з закупівель з паролем «manager123»;

3) працівника складу з паролем «storekeeper123».

Для видалення облікового запису використовується оператор DROP USER. Зміна імені користувача в обліковому записі виконується

за допомогою оператора RENAME USER %old_name% TO %new_name%.

Оскільки усі облікові записи користувачів зберігаються у таблиці «user» системної бази даних «mysql», перевірити створення розглянутих облікових записів можна за допомогою наступного запита (рисунок 4.1):

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

Рисунок 4.1

2. Призначити привілеї для створених облікових записів

Розглянуті вище оператори дозволяють створювати, видаляти та редагувати облікові записи, однак вони не дозволяють змінювати привілеї користувача – повідомляти MySQL, який користувач має право тільки на читання інформації, який на читання та редагування, а кому надані права змінювати структуру БД та створювати облікові записи.

Необхідно призначити привілеї для створених облікових записів.

```
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';
```

Для позбавлення облікового запису користувача певних привілеїв використовується оператор REVOKE. Даний оператор не

видаляє облікові записи, а лише відміняє надані раніше привілеї. Тому для остаточного видалення облікового запису необхідно скористатися оператором DROP USER.

Перевірити привілеї облікового запису «admin», якому були надані усі права на рівні бази даних «supply», можна за допомогою наступного запита (рисунок 4.2).

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

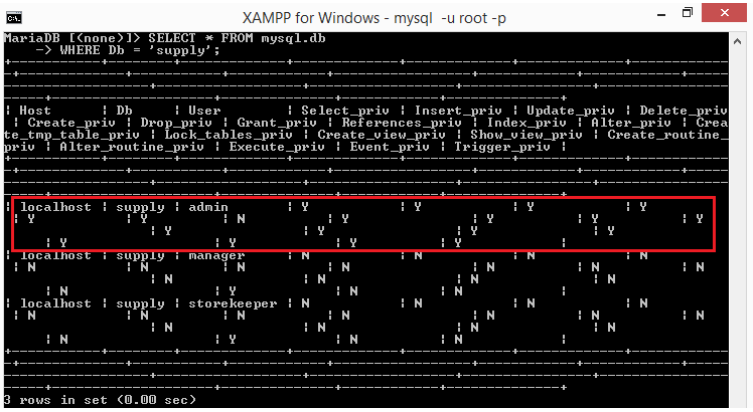

Рисунок 4.2

Аналогічно можна перевірити привілеї облікових записів «manager» та «storekeeper», для яких були визначені певні обмеження у роботі з таблицями бази даних «supply» (рисунок 4.3).

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

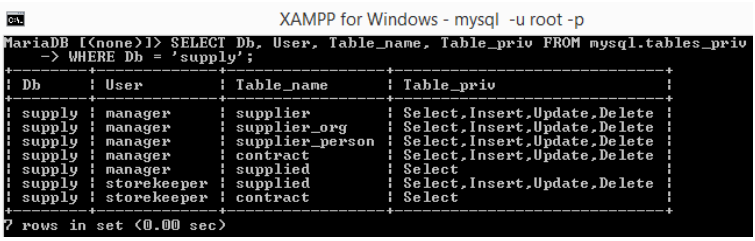

Рисунок 4.3

Крім того, певним користувачам необхідно надати також привілеї, які дозволять їм використовувати представлення, що містяться у базі даних «supply». Наприклад, користувачу manager повинні бути надані права для перегляду представлень «contract_supplier» та «supplier_info», тоді як для користувача «storekeeper» повинне бути доступним лише представлення «contract_supplier».

3. Оформити звіт з лабораторної роботи

У звіт включити основні етапи виконання лабораторної роботи та знімки екрану, що їх демонструють.

4. Питання для самоконтролю

1. Який вигляд має обліковий запис користувача у СУБД MySQL?
2. З яких складових формується обліковий запис?
3. Яке призначення у складових облікового запису?
4. Яким чином можна переглянути усі облікові записи?
5. Яка команда використовується для створення облікового запису?
6. Яка команда використовується для видалення облікового запису?
7. Яким чином можна змінити ім’я користувача в обліковому записі?
8. За допомогою якого оператора можна визначити певні привілеї для необхідного облікового запису?
9. Який оператор може бути використаний для відміни привілеїв?
10. Які привілеї можуть бути визначені для облікового запису?
11. Які існують рівні призначення привілеїв?
12. Яким чином можна перевірити глобальні привілеї, привілеї бази даних та привілеї таблиць?

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»

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. 277

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