Міністерство освіти і науки України

Національний університет “Львівська Політехніка”

Кафедра інформаційних систем та мереж



ЗВІТ

про виконання лабораторної роботи № 6

**“Створення таблиць бази даних засобами SQL”**

з дисципліни “Організація баз даних та знань”

Виконав студент групи УП-21

Кісіль Н.А.

Прийняла викладачка

Нич Л.Я.

2024

**Частина 1**

**Мета:** Навчитися розробляти та виконувати збережені процедури та функції у MySQL.

**Теоретична частина**

Більшість СУБД підтримують використання збережених послідовностей команд для виконання часто повторюваних, однотипних дій над даними. Такі збережені процедури дають змогу спростити опрацювання даних, а також підвищити безпеку при роботі з базою даних, оскільки в цьому випадку прикладні програми не потребують прямого доступу до таблиць, а отримують потрібну інформацію через процедури.

СУБД MySQL підтримує збережені процедури і збережені функції. Аналогічно до вбудованих функцій (типу COUNT), збережену функцію викликають з деякого виразу і вона повертає цьому виразу обчислене значення. Збережену процедуру викликають за допомогою команди CALL. Процедура повертає значення через вихідні параметри, або генерує набір даних, який передається у прикладну програму.

Синтаксис команд для створення збережених підпрограм такий:

CREATE

[DEFINER = { користувач | CURRENT\_USER }]

FUNCTION назва\_функції ([параметри\_функції ...])

RETURNS тип

[характеристика ...] тіло\_функції

CREATE

[DEFINER = { користувач | CURRENT\_USER }]

PROCEDURE назва\_процедури ([параметри\_процедури ...])

[характеристика ...] тіло\_процедури

**Описання виконаної роботи**

Напишемо функції, які будуть обгортками стандартних функцій шифрування, та процедуру, яка буде обчислювати кількість написаних автором повідомлень у кожній категорії за вказаний проміжок часу.

1). Функції шифрування/дешифрування із заданим ключем.

CREATE DEFINER=`root`@`localhost` FUNCTION `decrypt\_with\_pass`(data VARCHAR(30)) RETURNS varchar(32) CHARSET utf8mb4

BEGIN

DECLARE decrypted\_data VARCHAR(32);

SET decrypted\_data = CONCAT('pass', data);

RETURN decrypted\_data;

END

CREATE DEFINER=`root`@`localhost` FUNCTION `encrypt\_with\_pass`(data VARCHAR(32)) RETURNS varchar(30) CHARSET utf8mb4

BEGIN

DECLARE encrypted\_data VARCHAR(30);

SET encrypted\_data = replace(data,"pass", '');

RETURN encrypted\_data;

END

2). Процедура має рахувати кількість замовленнь зроблених користувачем за певний проміжок часу. Для цього треба відібрати всі замовлення за даною датою. Потім з даних замовлень відібрати усі замовлення з прізвищем та ID клієнта. У процедуру потрібно передати Прізвище та ID клієнта, а також першу і другу дату.

DELIMITER |

CREATE PROCEDURE luxurycar\_count(in x varchar(100), in ID\_x int, in date1 date, in date2 date)

Begin

select client.surname, client.clientID as ID, count(client.clientID) as Amount

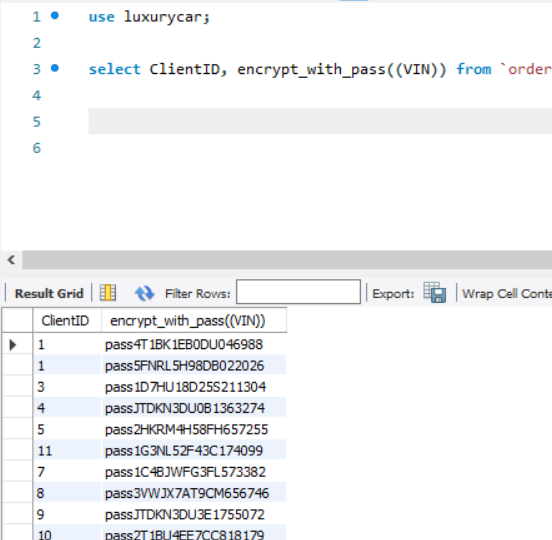
from client left join `order` on client.clientID = `order`.ClientID

where (`order`.Date\_of\_sale between date1 and date2) and (client.Surname = x) and (`order`.clientID = ID\_x);

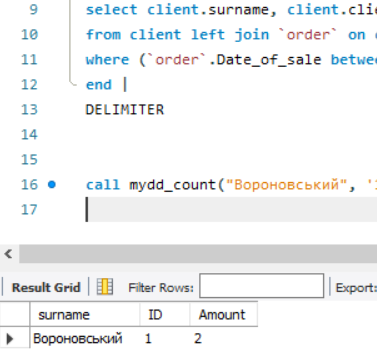
end |

DELIMITER

3). Після створення функцій і процедури перевіримо їхню роботу



**Рис.1.** *Результат шифрування*

call mydd\_count("Вороновський", '1', '2021-04-20', '2029-11-20'); 

**Рис.2.** *Результат Процедури*

**Висновок:** У цій лабораторній роботі ми навчилися створювати й виконувати збережені процедури й функції в MySQL. Це важливі навички, які дозволяють оптимізувати роботу з базою даних, зменшуючи повторюваний код і підвищуючи швидкодію операцій.

**Частина 2**

**Мета:**Навчитися використовувати механізм транзакцій у СУБД MySQL. Розробити SQL запити, які виконуються як єдине ціле в рамках однієї транзакції.

**Теоретична частина**

Транзакція – це сукупність директив SQL, які виконуються як єдине ціле з можливістю відміни результатів їхнього виконання. Зміни в таблицях записуються у базу даних лише після успішного виконання всіх директив транзакції. Інакше, всі зроблені зміни ігноруються. Це дає змогу уникати помилок при маніпулюванні великими обсягами записів, зберігати цілісність даних при помилках під час додавання, видалення, модифікації значень у різних таблицях і полях тощо. СУБД MySQL також підтримує глобальні розподілені транзакції, які виконуються на декількох базах даних, або на різних серверах баз даних (ХА-транзакції)

Для організації транзакцій в MySQL використовують такі директиви, як SET autocommit,

START TRANSACTION, COMMIT і ROLLBACK. START TRANSACTION

Вказує на початок транзакції. Директива вимикає автоматичне збереження змін для всіх подальших запитів, поки не буде виконано команду COMMIT, або ROLLBACK.

COMMIT

Зберегти зміни, зроблені даною транзакцією.

ROLLBACK

Відмінити дану транзакцію і зроблені нею зміни у базі даних. Слід зауважити, що зміни у схемі бази даних не можна відмінити, тобто результат видалення, зміни або створення таблиці завжди зберігається.

SET autocommit=0

Вимикає автоматичне збереження змін для поточної сесії зв’язку з сервером БД. За замовчуванням, зміни зберігаються автоматично, тобто результат виконання запиту, який змінює таблицю, одразу записується на диск без можливості відміни операції.

AND CHAIN

Одразу після завершення даної транзакції розпочати виконання наступної.

RELEASE

Одразу після виконання даної транзакції завершити поточну сесію зв’язку з сервером. Транзакції можна розбивати на окремі логічні частини, оголошуючи так звані точки збереження. Це дає змогу відміняти результати виконання не всієї транзакції, а лише тих запитів, які виконувались після оголошеної точки збереження (SAVEPOINT).

SAVEPOINT мітка

Оголошує точку збереження всередині транзакції та задає її назву.

ROLLBACK TO [SAVEPOINT] мітка

Відміняє результати виконання запитів, вказаних після даної точки збереження.

RELEASE SAVEPOINT мітка

Видаляє точку збереження.

**Описання виконаної роботи**

У ході роботи, потрібно продемонструвати успішне і неуспішне виконання транзакції. Розробимо транзакцію, яка буде вносити дані в таблицю Author i Role. Транзакція буде відміняти всі зміни у таблицях при виникненні помилки чи іншої суперечливості.

1) Відміна транзакції

Start transaction;

insert into auctions values

('6', "lapux", "Lviv", '8', '2'),

('7', "lapux\_2", "kyiv", '4','3');

rollback;

**Відповідь сервера:**



**Рис.3.** *Результат сервера*

2) Успішна транзакція:

Start transaction;

Insert into performer value

(Null, 'LP', '1981-03-18', 'United States', 2);

Insert into disk values

(null, 'Evolve', 689000, 160, '2017-06-19', 7, '00:34:42', 12),

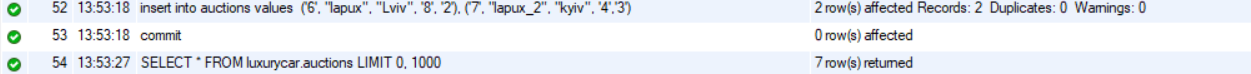
(null, 'Night Visions', 432000, 125, '2012-06-23', 7, '00:43:14', 20),

(null, 'Live in Moscow', 245000, 115, '2020-07-26', 8, '00:45:16', 17),

(null, 'Lost on You', 368000, 130, '2016-12-22', 8, '00:36:44', 10);

Commit;

Результат успішного виконаня:



**Рис.4.** *Результат сервера*

**Висновок:** У лабораторній роботі ми вивчили використання транзакцій в MySQL. Ми розробили SQL запити, які виконуються як одне ціле в рамках однієї транзакції. Це важливо для забезпечення консистентності даних та уникнення проблем з їх цілісністю.

**Частина 3**

**Мета:** Розробити SQL запити, які моделюють роботу тригерів: каскадне знищення, зміна та доповнення записів у зв’язаних таблицях.

**Теоретичні відомості**

Тригер – це спеціальний вид користувацької процедури, який виконується автоматично при певних діях над таблицею, наприклад, при додаванні чи оновленні даних. Кожен тригер асоційований з конкретною таблицею і подією. Найчастіше тригери використовуються для перевірки коректності введення нових даних та підтримки складних обмежень цілісності. Крім цього їх використовують для автоматичного обчислення значень полів таблиць, організації перевірок для захисту даних, збирання статистики доступу до таблиць баз даних чи реєстрації інших подій.

Синтаксис

Для створення тригерів використовують директиву CREATE TRIGGER.

CREATE

[DEFINER = { користувач | CURRENT\_USER }]

TRIGGER ім’я\_тригера час\_виконання подія\_виконання ON назва\_таблиці FOR EACH ROW тіло\_тригера

Можлива подія – це внесення (INSERT), оновлення (UPDATE), або видалення (DELETE) рядка з таблиці. Один тригер може бути пов’язаний лише з однією подією. Команда AFTER INSERT, AFTER UPDATE, AFTER DELETE визначає виконання тіла тригера відповідно після внесення, оновлення, або видалення даних з таблиці. Команда BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE визначає виконання тіла тригера відповідно до внесення, оновлення, або видалення даних з таблиці.

Задає набір SQL директив, які виконує тригер. Тригер викликається і виконується для кожного зміненого рядка. Директиви можуть об’єднуватись командами BEGIN … END та містити спеціальні команди OLD та NEW для доступу до попереднього та нового значення поля у зміненому рядку відповідно. У тілі тригера дозволено викликати збережені процедури, але заборонено використовувати транзакції, оскільки тіло тригера автоматично виконується як одна транзакція.

**Описання виконаної роботи**

1). Потрібно створити тригер, який буде зберігати інформацію про зміну цін у таблиці price\_changes;

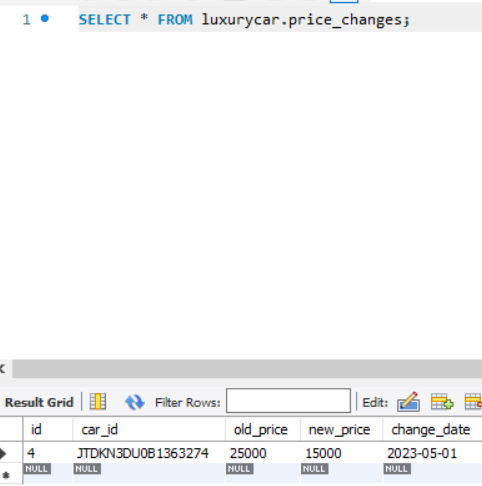
CREATE DEFINER=`root`@`localhost` TRIGGER `update\_price\_trigger` AFTER UPDATE ON `order` FOR EACH ROW BEGIN

IF NEW.Price <> OLD.Price THEN

INSERT INTO price\_changes VALUES (NEW.ID, OLD.VIN, OLD.price, NEW.price, curdate());

END IF;

END



**Рис.5.** *Результат збереження*

2). Створимо тригер, який буде шифрувати прізвище користувача функцією decrypt\_with\_pass перед тим як внести його у таблицю client.

use luxurycar;

Create trigger trigger\_on\_update before insert on client

for each row

set new.surname = decrypt\_with\_pass(new.surname);





**Рис.6-7.** *Результати праці тригера*

3). Створимо тригер, який буде видаляти запис з таблиці клієнтів при видаленні запису з таблиці замовлень:

CREATE TRIGGER delete\_order\_trigger

AFTER DELETE ON orders

FOR EACH ROW

BEGIN

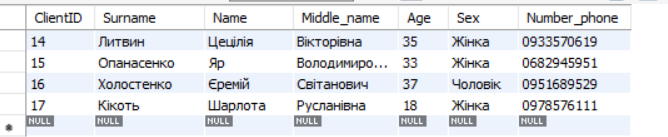
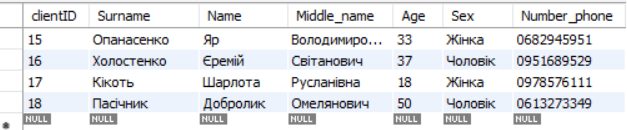
DELETE FROM customers WHERE id = OLD.customer\_id;

END;

SET foreign\_key\_checks = 0;

delete from `order` where ClientID = '18';

SET foreign\_key\_checks = 1;



**Рис.8-9.** *Результат до та після видалення*

**Висновок*:*** У даній лабораторній роботі ми створили SQL запити, які моделюють роботу тригерів для каскадного видалення, зміни та додавання записів у зв'язаних таблицях. Ці тригери відтворюють поведінку бази даних у відповідь на певні події, що стаються в таблицях, забезпечуючи консистентність та цілісність даних.