МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Звіт до лабораторної роботи №10

З ПРЕДМЕТУ “ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ”

Виконав:

ст. гр. КН-211

Шебеко Андрій

Викладач:

Якимишин Х.М.

Львів – 2020

Лабораторна робота №10
з курсу "ОБДЗ"
на тему:
"Написання збережених процедур на мові SQL"

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

Короткі теоретичні відомості.

Більшість СУБД підтримують використання збережених послідовностей команд для виконання часто повторюваних, однотипних дій над даними. Такі збережені процедури дозволяють спростити оброблення 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 процедур описано нижче.

```
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
```

Аргументи:

`DEFINER`
Задає автора процедури чи функції. За замовчуванням – це `CURRENT_USER`.

`RETURNS`
Вказує тип значення, яке повертає функція.

`тіло_функції, тіло_процедури`
Послідовність директив SQL. В тілі процедур і функцій можна оголошувати локальні змінні, використовувати директиви `BEGIN ... END`, `CASE`, цикли тощо. В тілі процедур також можна виконувати транзакії. Тіло функції обов'язково повинно містити команду `RETURN` і повертати значення.

***параметри_процедури*:**

`[ IN | OUT | INOUT ] ім'я_параметру тип`

Параметр, позначений як IN, передає значення у процедуру. OUT-параметр передає значення у точку виклику процедури. Параметр, позначений як INOUT, задається при виклику, може бути змінений всередині процедури і зчитаний після її завершення. Типом параметру може бути будь-який із типів даних, що підтримується MySQL.

***параметри_функції*:**

ім'я_параметру тип

У випадку функцій параметри використовують лише для передачі значень у функцію.

При створенні процедур і функцій можна вказувати їхні додаткові характеристики.

***характеристика*:**

```
LANGUAGE SQL
| [NOT] DETERMINISTIC
| {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
| SQL SECURITY {DEFINER | INVOKER}
| COMMENT 'короткий опис процедури'
```

`DETERMINISTIC`

Вказує на те, що процедура обробляє дані строго визначеним (детермінованим) чином. Тобто, залежно від вхідних даних, процедура повертає один і той самий результат. Недетерміновані процедури містять функції типу `NOW()` або `RAND()`, і результат їх виконання не можна передбачити. За замовчуванням всі процедури і функції є недетермінованими.

`CONTAINS SQL | NO SQL`

Вказує на те, що процедура містить (за замовчуванням), або не містить директиви SQL.

`READS SQL DATA`

Вказує на те, що процедура містить директиви, які тільки зчитують дані з таблиць.

`MODIFIES SQL DATA`

Вказує на те, що процедура містить директиви, які можуть змінювати дані в таблицях.

`SQL SECURITY`

Задає рівень прав доступу, під яким буде виконуватись процедура. `DEFINER` – з правами автора процедури (задано за замовчуванням), `INVOKER` – з правами користувача, який викликає процедуру. Щоб запускати збережені процедури і функції, користувач повинен мати права `EXECUTE`.

При створенні процедур і функцій у командному рядку клієнта MySQL, потрібно перевизначити стандартний символ завершення вводу директив ";", щоб мати можливість ввести всі директиви процедури. Це робиться за допомогою команди `DELIMITER`. Наприклад,

```
DELIMITER |
```

означає, що завершення вводу процедури буде позначатись символом "|".

Хід роботи

1. Створюємо процедуру get_rating, яка приймає 1 аргумент – ім’я користувача, обраховує рейтинг довіри до користувача опираючись на його активність, покупки, надійність паролю та коли він був зареєстрований та вертає цей рейтинг.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `get_rating`(in name varchar(40))
BEGIN
    declare err varchar(22) DEFAULT "No such name in table!";

    if length(name) = 0 then
        select err;
    else
        select `first name`, count(distinct comment) as comments,
        ifnull(sum(distinct amount), 0) as products,
        @r:= (count(distinct comment) + ifnull(sum(distinct amount), 0)*2) as rating
        from client left join comment
        on client.`first name` = comment.username
        left join basket
        on client.client_id = basket.client_id
        where `first name` = name
        group by  `first name`
        order by rating DESC limit 3;

        set @p = (select password from client where `first name` = name);
        set @d = (select register_date from client where `first name` = name);

        if length(@p) > 5 then
            set @r = @r + 2;
            select 'Good password!';
        end if;

        if TIMESTAMPDIFF(Month, @d, CURDATE()) >= 2 then
            set @r = @r + 2;
            select 'Old user';
        end if;

        select @r;
    end if;
END
```

Викликаємо цю процедуру для різних користувачів:

```
call get_rating('Andrii');
call get_rating('Vitya');
call get_rating('');
```

Перших два запити виконуються коректно, оскільки такі користувачі присутні в таблиці.

@r
14

@r
3

А 3й запит містить некоректно введені дані, на що ми отримуємо:

err
No such name in table!

2. Створюємо функцію max_order_size, яка приймає 2 аргументи – ціна та кількість продуктів, вертає або загальну вартість або (-1) у разі неправильно введених даних.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `max_order_size`(capicity int, value int) RETURNS int
    DETERMINISTIC
BEGIN
    if capicity = 0 or value = 0 then
        return -1;
    end if;

    return capicity * value;
END
```

Запускаю дану функцію двічі, передаючи різні аргументи.

```
select max_order_size(110,30) as ord;
select max_order_size(0,30) as ord;
```

Отримуємо загальну ціну товару у 1му випадку

ord
3300

А у 2му випадку отримуємо -1, що значить неправильно введені данні.

ord
-1

Висновок: я навчився створювати та використовувати функції та збережені процедури на мові SQL.