

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 транзакції. 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 "|".

Нижче наведено синтаксис додаткових директив MySQL, які дозволяють розробляти нескладні програми на мові SQL.

```
DECLARE назва_змінної тип_змінної  
[DEFAULT значення_за_замовчуванням]  
Оголошення змінної заданого типу.
```

```
SET назва_змінної = вираз  
Присвоєння змінній значення.
```

```
IF умова THEN директиви  
[ELSEIF умова THEN директиви] ...  
[ELSE директиви2]  
END IF
```

Умовний оператор. Якщо виконується вказана *умова*, то виконуються відповідні їй *директиви*, в протилежному випадку виконуються *директиви2*.

```
CASE вираз  
WHEN значення1 THEN директиви1  
[WHEN значення2 THEN директиви2] ...  
[ELSE директиви3]  
END CASE
```

Оператор умовного вибору. Якщо *вираз* приймає *значення1*, виконуються *директиви1*, якщо приймає *значення2* – виконуються *директиви2*, і т.д. Якщо вираз не прийме жодного зі значень, виконуються *директиви3*.

```
[мітка:] LOOP  
директиви  
END LOOP
```

Оператор безумовного циклу. Вихід з циклу виконується командою LEAVE *мітка*.

```
REPEAT  
директиви  
UNTIL умова  
END REPEAT
```

```
WHILE умова DO  
директиви  
END WHILE
```

Оператори REPEAT і WHILE дозволяють організувати умовні цикли, які завершуються при виконанні деякої умови.

Хід 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 FUNCTION mycms_encode (pass CHAR(48))  
RETURNS TINYBLOB  
RETURN AES_ENCRYPT(pass, 'key-key');
```

```
CREATE FUNCTION mycms_decode (pass TINYBLOB)  
RETURNS CHAR(48)  
RETURN AES_DECRYPT(pass, 'key-key');
```

2. Процедура повинна рахувати кількість повідомлень автора написаних за певний проміжок часу у кожній з існуючих категорій. Для цього потрібно відібрати всі повідомлення та їх категорії за автором та часом написання. Потім згрупувати вибрані повідомлення за категоріями та порахувати кількість повідомлень. У процедуру потрібно передати ім'я автора, а також першу і другу дату.

Перед основними директивами додамо перевірку коректності задання початкової і кінцевої дати (IF date1<=date2 THEN...). Результати обчислень будуть записуватись у таблицю Stats, яку процедура завжди очищує (командою TRUNCATE mycms.stats) і заповнює з нуля.

```
DELIMITER //
```

```
CREATE PROCEDURE mycms_count (IN name CHAR(19), IN date1 DATE, IN  
date2 DATE)  
BEGIN  
  DECLARE error CHAR;  
  SET error = 'Некоректно задані дати';  
  IF (date1<=date2) THEN  
    BEGIN  
      CREATE TABLE IF NOT EXISTS mycms.stats (category CHAR(20),  
amount INT UNSIGNED);  
      TRUNCATE mycms.stats;  
      INSERT INTO mycms.stats SELECT cname AS category,  
COUNT(message.messageID) AS amount  
FROM ((author INNER JOIN message)  
INNER JOIN message_category) INNER JOIN category  
ON author.login=name  
AND author.authorID=message.authorID  
AND message.messageID=message_category.messageID  
AND message_category.categoryID=category.categoryID  
WHERE message.posted BETWEEN date1 AND date2  
      GROUP BY category;  
    END;  
    ELSE SELECT error;  
  END IF;  
END//  
DELIMITER ;
```

3. Після створення функцій і процедури перевіримо їх роботу:

```
SELECT login, mycms_decode(password) FROM author LIMIT 4;
```

login	mycms_decode(password)
admin	adminpass
user1	uuuupass
user2	user2pass
user3	user3pass

```
CALL mycms_count('user1', '2008-01-01', '2009-05-05');  
SELECT * FROM stats;
```

Результат роботи процедури – таблиця stats:

category	amount
blogentry	2
movies	1
news	1
questions	1
ua	1

```
CALL mycms_count('user1', '2010-01-01', '2009-05-05');
```

Результат виклику процедури:

error
Некоректно задані дати

Висновок: на цій лабораторній роботі я навчився розробляти та використовувати збережені процедури і функції у СУБД MySQL.