

МІНІСТЕРСТВО ОСВІТИ І НАУКИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА  
ПОЛІТЕХНІКА»

Кафедра систем штучного інтелекту



Лабораторна робота №10  
З курсу “Організація баз даних та знань”

Виконав:  
студент групи КН-210  
Марій Павло

Викладач:  
Мельникова Наталя Іванівна

**Тема:** Написання збережених процедур на мові SQL.

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

### 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.

## **Хід роботи**

Напишемо функції, які будуть обгортками стандартних функцій шифрування, та процедуру, яка буде обчислювати кількість опублікованих користувачами оголошень для кожної марки за вказаний проміжок часу.

### 1. Функції шифрування/дешифрування із заданим ключем.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `encode`(pass
VARCHAR(200)) RETURNS tinyblob DETERMINISTIC
BEGIN
    RETURN AES_ENCRYPT(pass, 'key-key');
END
```

```
CREATE DEFINER=`root`@`localhost` FUNCTION `decode`(pass
tinyblob) RETURNS VARCHAR(200) DETERMINISTIC
BEGIN
```

```

RETURN AES_DECRYPT(pass, 'key-key');
END

```

Це функції-обгортки стандартних функцій шифрування чи дешифрування. Їх можна використовувати наступним чином:

```

SELECT id, price, mileage, encode(color)
FROM mydb.advertisement
WHERE mileage >= 200000;

```

Результат:

Result Grid				
	id	price	mileage	encode(color)
▶	8	800	200000	BLOB
	9	5500	250000	BLOB
	10	10800	280000	BLOB
	18	6800	400000	BLOB
	19	24000	320000	BLOB

```

SELECT id, price, mileage, decode(tbl.code) FROM
(SELECT id, price, mileage, encode(color) AS code
FROM mydb.advertisement
WHERE mileage >= 200000)
AS tbl;

```

Тут я спершу кодую поле color, а потім декодую його.

Результат:

	id	price	mileage	decode(tbl.code)
▶	8	800	200000	red
	9	5500	250000	black
	10	10800	280000	yellow
	18	6800	400000	red
	19	24000	320000	green

2. Я написав наступну процедуру `get_marks_by_dates`, вона приймає як аргументи дві дати як проміжок часу, і рахує кількість оголошень за цей проміжок часу відносно марок автомобілів.

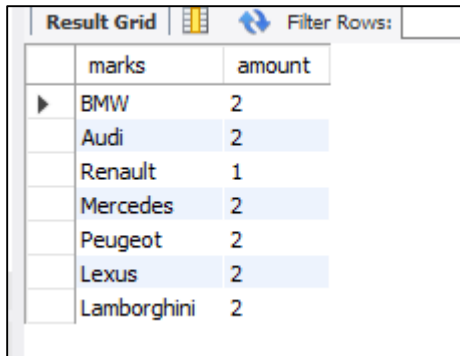
Код процедури:

```
CREATE PROCEDURE `get_marks_by_dates` (IN date1 DATETIME, IN date2
DATETIME)
BEGIN
    DECLARE error CHAR;
    SET error = 'Dates are not valid';
    IF (date1<=date2) THEN
    BEGIN
        CREATE TABLE IF NOT EXISTS mydb.marks_by_dates
        (marks VARCHAR(45), amount INT UNSIGNED);
        TRUNCATE mydb.marks_by_dates;
        INSERT INTO mydb.marks_by_dates
        SELECT MA.name, COUNT(AD.id) AS amount
        FROM mydb.advertisement AD
        INNER JOIN mydb.auto AU
            ON AD.auto_id = AU.id
        INNER JOIN mydb.model MO
            ON AU.model_id = MO.id
        INNER JOIN mydb.mark MA
            ON MO.mark_id = MA.id
        WHERE AD.pubdate BETWEEN date1 AND date2
        GROUP BY MA.id
        ORDER BY MA.id;
    END;
    ELSE SELECT error;
    END IF;
END
```

Далі спробуємо викликати цю процедуру:

```
CALL get_marks_by_dates('2020-01-01 10:10:10', '2020-04-04 21:21:24');
SELECT * FROM mydb.marks_by_dates;
```

Результат:

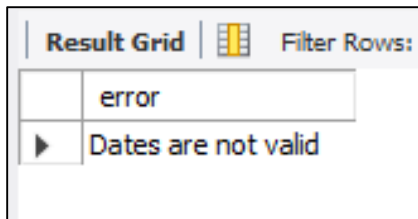


	marks	amount
▶	BMW	2
	Audi	2
	Renault	1
	Mercedes	2
	Peugeot	2
	Lexus	2
	Lamborghini	2

Якщо перша дата буде більшою, ніж друга, тоді проміжок часу буде від'ємним, і це некоректні дані, для цього я передбачив виклик помилки з текстом: «Dates are not valid». Перевіримо це:

```
CALL get_marks_by_dates('2020-04-04 21:21:24', '2020-01-01 10:10:10');  
SELECT * FROM mydb.marks_by_dates;
```

Результат:



	error
▶	Dates are not valid

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