МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота 10

з організації баз даних та знать

Виконав:

Студент групи КН-208

Воробель Адріан

Викладач:

Якимишин Х. М.

Львів – 2020р.

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 "|".

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.

1. Створимо функцію, яка буде обгортати стандартну бібліотечну

```
create function autoshop_uppercase(str nvarchar(50))
returns nvarchar(50)
return upper(str);


create function autoshop_lowercase(str nvarchar(50))
returns nvarchar(50)
return lower(str);
```

А тепер протестуємо на прикладі:

```
select autoshop_uppercase(login), autoshop_lowercase(pass) from users;
```

	autoshop_uppercase(login)	autoshop_lowercase(pass)
▶	ADMIN	admin
	ALEXANDER1983	12345678
	THEBESTCUSTOMER	best123
	USERBUYCAR	iwantbmw
	MERCEDESLOVER	amgislife
	USER1	pass1
	USER2	pass2
	USER3	pass3
	USER4	pass4

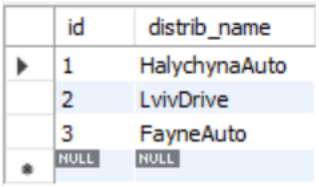
2. Створимо процедуру, яка буде рахувати кількість автомобілів в заданого дистрибютора та повертати ці результати в якості нової таблиці.

```
DELIMITER //

drop procedure if exists distrib_car_count;

create procedure distrib_car_count(in name nvarchar(50))

begin

  declare error_msg nvarchar(50);

  set error_msg = "дистрибютор відсутній в списку";

  if (name in (select distrib_name from distributor)) then

      begin

      create table if not exists autoshop.distrib_car_procedure(distrib_name nvarchar(50), amount int);

        truncate autoshop.distrib_car_procedure;

        insert into autoshop.distrib_car_procedure

        select distributor.distrib_name as distrib_name,count(cardistributor.car_id) as amount

        from distributor inner join cardistributor on distributor.id = cardistributor.distrib_id

        where distributor.distrib_name = name

        group by distrib_name;

      end;

        else select error;

        end if;
```

```
end//
  DELIMITER ;
```

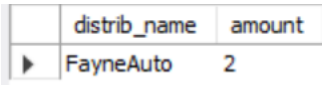
А тепер протестуємо на прикладі:

```
select * from distributor;
```



	id	distrib_name
▶	1	HalychynaAuto
	2	LvivDrive
	3	FayneAuto
*	NULL	NULL

```
call distrib_car_count('FayneAuto');
select * from distrib_car_procedure;
```



	distrib_name	amount
▶	FayneAuto	2

Висновок.

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