МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Лабораторна робота №10

З дисципліни «Організація баз даних та знань»

Виконав:
студент групи КН-210
Дойков Вадим

Викладач:
Мельникова Н. І.

Львів – 2020

Тема: Написання збережених процедур на мові SQL

Мета: 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

```
REPEAT

директиви UNTIL умова END REPEAT
```

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

мітка.

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

Хід роботи

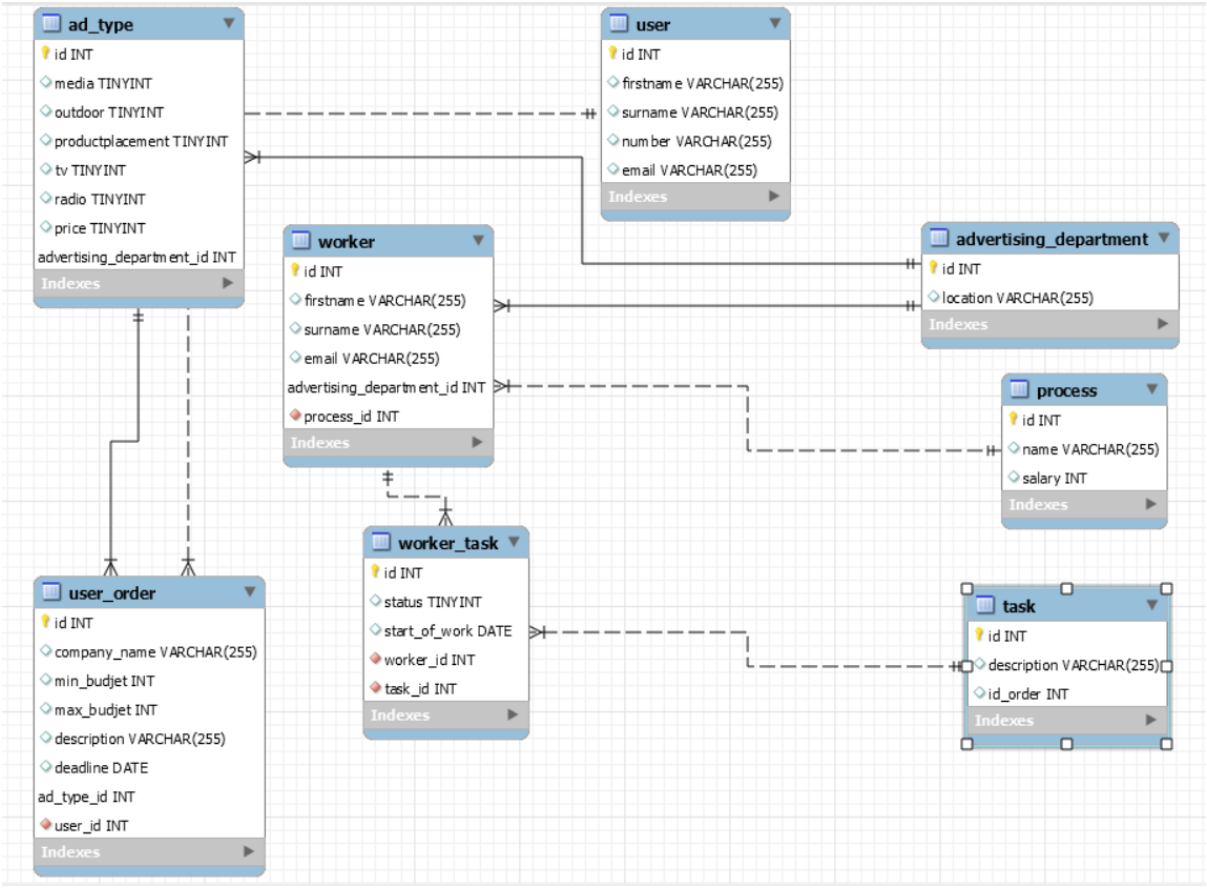


Рис 1. ER-діагрма бази даних

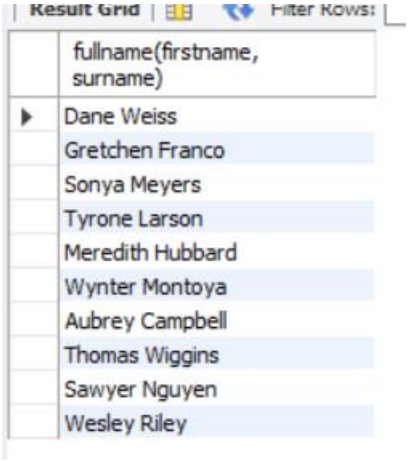
1. Функція конкатенації імені та прізвища:

```
use adcompany;


delimiter //
create function fullname(
    firstname varchar(255),
    lastname varchar(255)
)
returns varchar(510)
deterministic
begin

    return concat(firstname, ' ', lastname);
end; //
```

Виклик створеної функції:

```
17 •    select fullname(firstname, surname)
18      from worker;
```

Результат виконання запиту функції:



fullname(firstname, surname)
Dane Weiss
Gretchen Franco
Sonya Meyers
Tyrone Larson
Meredith Hubbard
Wynter Montoya
Aubrey Campbell
Thomas Wiggins
Sawyer Nguyen
Wesley Riley

2. Процедура, яка перевіряє чи хто з працівників виконав завдання або ні:

```
22      delimiter //
23 •    create procedure check_work(IN status_of_work tinyint)
24      begin
25          select w.id, fullname(firstname, surname) as fullname, wt.status, ur.company_name
26          from worker as w
27          inner join worker_task as wt on wt.worker_id = w.id
28          inner join task as t on t.id= wt.task_id
29          inner join user_order as ur on ur.id = t.id_order
30          where wt.status = status_of_work
31          order by fullname;
32      end; //
```

Виклик створеної процедури з аргументом 1:

```
35 •    call check_work(1);
```

Результат виконання даної процедури буде список працівників, які завершили певне завдання:

id	fullname	status	company_name
5	Meredith Hubbard	1	Et Eros PC
5	Meredith Hubbard	1	A Sollicitudin Orci Industries
3	Sonya Meyers	1	Sollicitudin Adipiscing Corp.
6	Wynter Montoya	1	Mattis Company

Виклик створеної процедури з аргументом 0:

```
35 •    call check_work(0);
```

Результат виконання даної процедури буде список працівників, які ще виконують певне завдання:

Result Grid | Filter Rows: | Export: | Wrap

id	fullname	status	company_name
2	Gretchen Franco	0	A Sollicitudin Orci Industries
3	Sonya Meyers	0	Mattis Company

Функція, яка перевіряє скільки працівник виконав робіт за заданий проміжок часу:

```
delimiter //
create function works_done(
  start_point date,
  end_point date,
  worker_name varchar(255)
)
returns int
deterministic
begin
declare count_of_work int default 0;
set count_of_work = (select count(*)
            from worker_task as wt
                        inner join worker as w on w.id = wt.worker_id
            where wt.start_of_work between start_point and end_point and wt.status = 1 and worker_name = w.firstname
          );
return count_of_work;
end; //
```

Результат виконання функції:

```
24 •    select works_done('2019-05-20', '2021-05-20', 'Meredith') as count_of_works;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

count_of_works
2

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