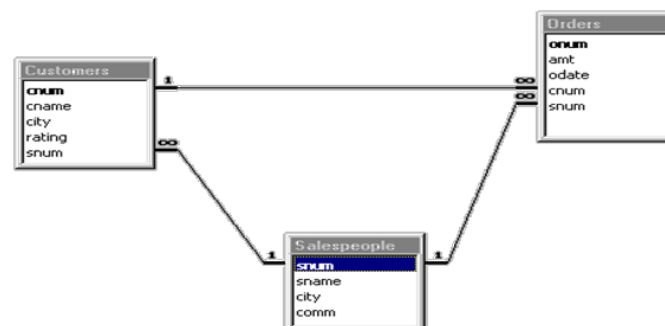


Бази даних

частина 6

Проектування і перевірка збережених
процедур (Stored Procedure)



Навчальний курс
Валько Н.В.

SQL

Мартин Грабер

МАРТИН ГРУБЕР

Понимание SQL

Перевод Лебедева В.Н.

Под редакцией Булычева В.Н.

МОСКВА, 1993

Джеймс Р. Грофф
Пол Н. Вайнберг

SQL

Полное руководство

Второе издание,
переработанное и дополненное

Перевод с английского
под редакцией В.Р. Гинзбурга

Линн Бейли

Изучаем SQL

Приведи
в порядок
свои
отношения
с данными



Прекрати путать
термины



Освой концепцию
и синтаксис SQL
максимально
эффективно



Перестань
смушаться





Зміст

- Визначення процедур
- Процедури
 - без параметрів,
 - з вхідними параметрами,
 - з вхідними параметрами за замовчуванням,
 - з вихідними параметрами
- Складні оператори
- Керування потоками даних

Збережені процедури

Stored Procedure

- ❑ Це підпрограми, які зберігаються в БД
- ❑ Створюються для часто виконуваних дій, або групи дій, які не потребують взаємодії з користувачем (працюють в фоновому режимі)
- ❑ Містять звичайні запити

Види: Збережені процедури

- це підпрограми, які викликає користувач.
Результат – кілька значень

Збережені функції

- це підпрограми, які викликає користувач.
Результат – одне значення

Приклад: $P_i()$

Тригери

- це підпрограми, які виконуються системою
автоматично **до/після** події

Типи Stored Procedure

- ❑ **Системні** - призначені для адміністрування дій. Є інтерфейсом, що забезпечує зміну, додавання, видалення і вибірку даних з системних таблиць як для користувача. Мають префікс `sp_`.
- ❑ **Користувацькі** – повноцінний об'єкт БД, зберігаються в конкретній БД
- ❑ **Тимчасові** (локальні, глобальні) – автоматично знищуються сервером при відключенні клієнта.

Команди

| | |
|--|--|
| CREATE PROCEDURE | Створення процедури |
| CREATE FUNCTION | Створення функції |
| ALTER PROCEDURE | Зміна процедури |
| ALTER FUNCTION | Зміна функції |
| DROP PROCEDURE | Видачення процедури |
| DROP FUNCTION | Видалення функції |
| SHOW CREATE PROCEDURE proc_name | Показати текст процедури proc_name |
| SHOW CREATE FUNCTION func_name | Показати текст функції func_name |
| SHOW PROCEDURE STATUS LIKE 'proc_name' | Показати характеристики процедури proc_name |
| SHOW FUNCTION STATUS LIKE 'func_name' | Показати характеристики функції func_name |
| CALL proc_name() | Викликати процедуру proc_name |
| DECLARE | Визначення локальних змінних |
| SET | Зміна значень локальних і глобальних змінних |
| SELECT ... INTO | Збереження значення вказаного стовпця в змінну |
| IF | Умовний оператор if-then-else-end |
| CASE ... WHEN | Оператор вибору |
| LOOP, REPEAT, WHILE | Цикли |
| RETURNS | Повернення значення з функції |

Переваги процедур

Недоліки процедур



- ☐ Швидко працюють – використовують кешування
- ☐ Зменшують час обробки даних
- ☐ Прості дії автоматизуються, їх виконує БД
- ☐ Є універсальними – працюють на будь-якій платформі, що використовує SQL
- ☐ Код завжди доступний БД

- ☐ «Прихована» бізнес-логіка, важко відслідкувати роботу
- ☐ Мала функціональність мови
- ☐ «Непереносність» - різні процедурні «діалекти»

Створення Stored Procedure

- ❑ **визначення типу SP:** тимчасова або користувачка
- ❑ **планування прав доступу.** При створенні SP слід враховувати, що вона буде мати ті ж права доступу до об'єктів бази даних, що і користувач який її створив
- ❑ **визначення вхідних і вихідних параметрів SP**
- ❑ **розробка коду SP.** Код процедури може містити послідовність будь-яких команд SQL, включаючи виклик інших процедур

Синтаксис загальний

{ CREATE | ALTER } PROC[EDURE] *p_name*
[; *num*] [{ @ *proc_parameter* *proc_type* }
[VARYING] [=default][OUTPUT]] [...n] [WITH
{ RECOMPILE | ENCRYPTION | RECOMPILE,
ENCRYPTION }]
[FOR REPLICATION] AS sql_operator [...n]

Варіанти процедур з/без параметрів

`CREATE PROCEDURE proc1 ()`: пустий список параметрів

`CREATE PROCEDURE proc1 (IN varname DATA-TYPE)`:
один вхідний параметр. Слово **IN** необов'язкове, тому що параметри за змовченням - **IN** (вхідні).

`CREATE PROCEDURE proc1 (OUT varname DATA-TYPE)`:
один вихідний параметр.

`CREATE PROCEDURE proc1 (INOUT varname DATA-TYPE)`:
один параметр, одночасно вхідний і вихідний.

Синтаксис створення процедури

CREATE [DEFINER = *user*]

PROCEDURE *p_name* ([*proc_parameter*[,...]])

[*characteristic* ...]

routine_body

proc_parameter:

[[IN | OUT | INOUT] *param_name* *type*]

Створення процедури

Синтаксис виклику процедури

CALL p_name([parameter[,...]])

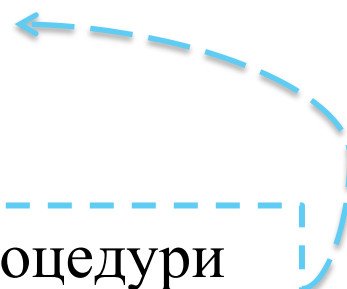
CALL p_name[()]



Виклик процедури
з параметрами і без

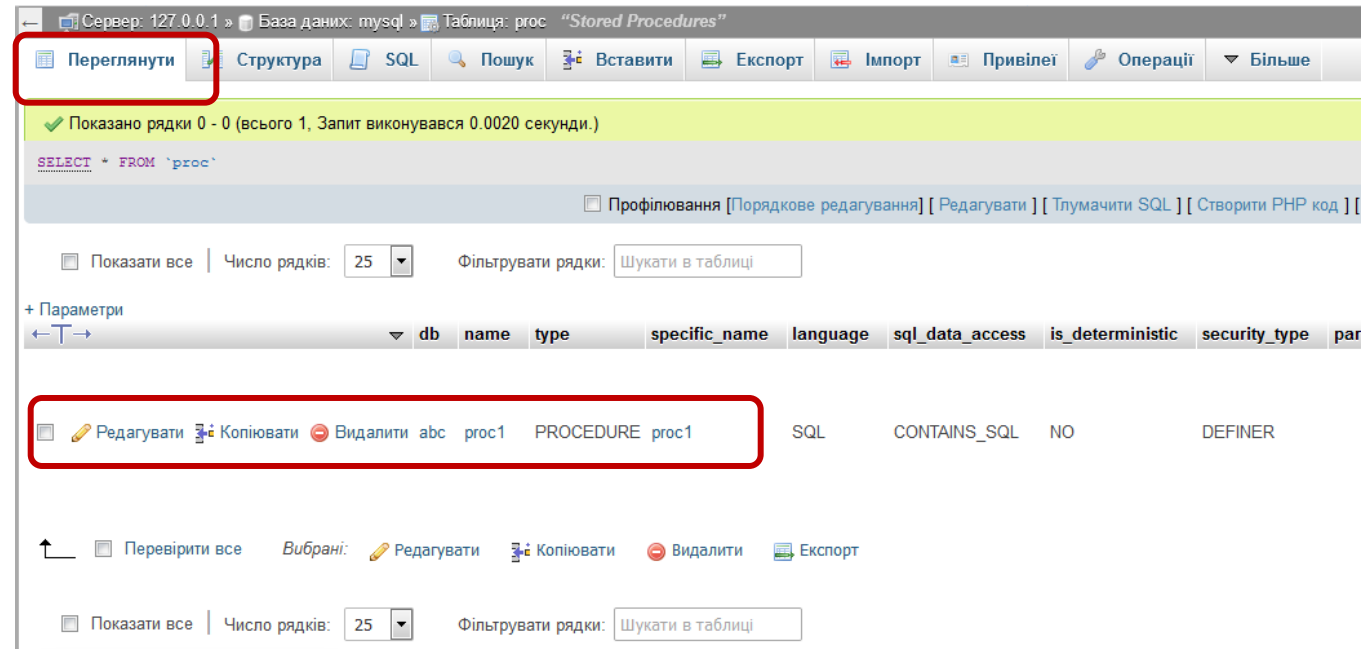
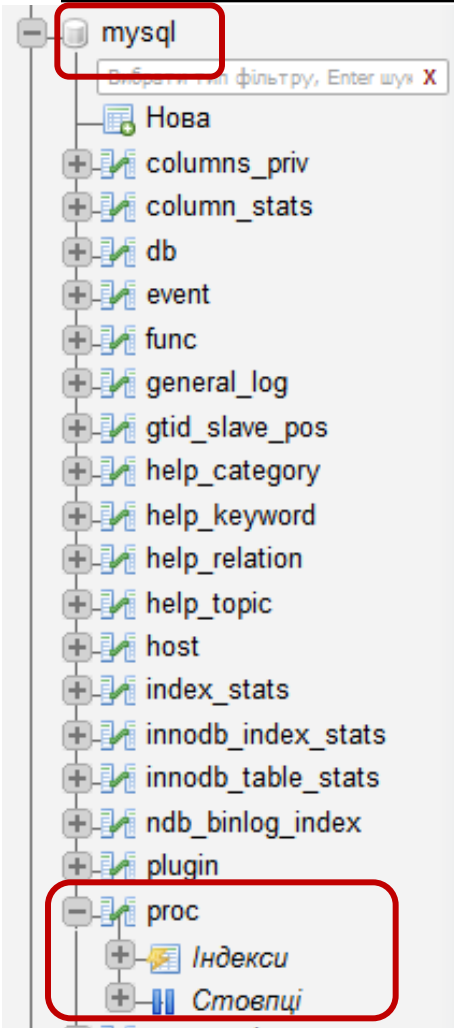
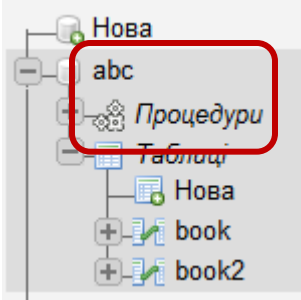
Приклад створення процедури

```
CREATE PROCEDURE SelectBooks ()  
SELECT * FROM Book;
```



Створення процедури
без параметрів

Розташування



Шукаємо тут...

Приклад виклику створеної процедури

CALL SelectBooks;

Виклик процедури
без параметрів

Викличте її ще раз

CALL SelectBooks;

... і ще

CALL SelectBooks;

... і ще ...

без параметрів

Приклад

```
CREATE PROCEDURE proc1()  
SELECT `name`,`neww` FROM `Book`  
WHERE (`neww`=1) order by (`name`) ASC ;
```

Виклик процедури

```
CALL proc1()
```

| name | neww |
|--|------|
| Автоматизация инженерно- графических работ | 1 |
| Аппаратные средства мультимедия. Видеосистема PC | 1 |
| Защита информации и безопасность компьютерных сист | 1 |

без параметрів

Приклад

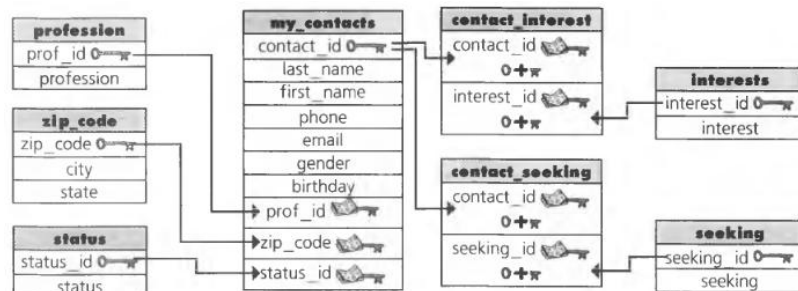
```
CREATE PROCEDURE proc3()
```

```
SELECT mc.email , p.profession
```

```
FROM my_contacts mc
```

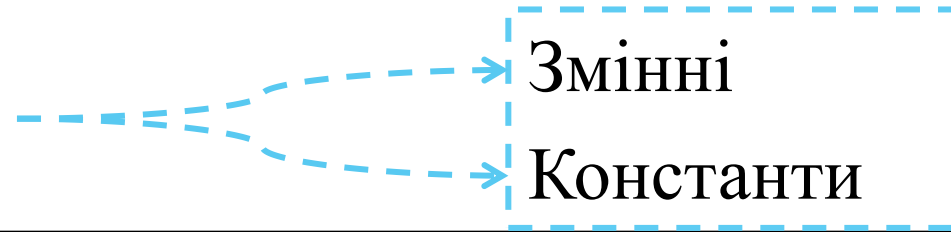
```
INNER JOIN profession p;
```

```
CALL proc3()
```



Запит, який повертає адреси електронної пошти (email) і професії (profession) кожної людини в my_contacts

Параметри



- Для передачі вхідних і вихідних даних у збереженій процедурі
- Збережена процедура може мати не більше 2100 параметрів
- Параметрам можна задавати значення за замовчуванням
- Змінні параметри починаються з символу @

Використання змінних

Або

```
SELECT @start := 1, @finish := 10;
```

```
SET @start = 1, @finish = 10;
```

```
SELECT * FROM places
```

```
WHERE place
```

```
BETWEEN @start AND @finish;
```

Процедури з параметрами

```
CREATE PROCEDURE nameBook  
(IN nameB VARCHAR(255))
```

```
SELECT * FROM book  
WHERE name LIKE nameB;
```

Виклик

```
CALL nameBook('%Win%');
```

Вхідний параметр

| nu | name |
|-----|---|
| 175 | Windows ME. Новейшие версии программ |
| 176 | Windows 2000 Professional шаг за шагом с CD |
| 206 | Windows Me. Спутник пользователя |

Процедура з двома параметрами

```
CREATE PROCEDURE proc2 (in n text(100), in y int)
SELECT `nu`, `pub` as 'Видавництво', `Dat`
FROM `Book`
WHERE year(`Dat`)>=y AND `pub` LIKE n
ORDER by `pub` AND `nu` ;
```

Вхідний параметр

```
CALL proc2('%BHV%', 2000)
```

| nu | name |
|-----|---|
| 175 | Windows ME. Новейшие версии программ |
| 176 | Windows 2000 Professional шаг за шагом с CD |
| 206 | Windows Me. Спутник пользователя |

Процедура OUT(вивід даних)

```
CREATE PROCEDURE simple1 (OUT param1 INT)  
SELECT COUNT(*) INTO param1 FROM book;
```

```
CALL simple1(@a);  
SELECT @a;
```

Рахує кількість записів у таблиці


@a

20

Процедура OUT(вивід даних)

```
CREATE PROCEDURE my_proc  
    (OUT var1 VARCHAR(15))  
SET var1 = 'This is a test';
```

Виклик

```
CALL my_proc (@var1);  
SELECT @var1;
```


Виводить текст на екран

Процедура OUT(вивід даних)

```
CREATE PROCEDURE my_proc (OUT price INT)
SET price = (SELECT MAX(book.price) FROM `book`);
```

Виклик

```
CALL my_proc_OUT(@price);
SELECT @price;
```



@price

38

Виводить на екран максимальне значення ціни

Процедура INOUT

```
CREATE PROCEDURE my_proc (INOUT var1 INT)  
SET var1 = var1 * 2;
```

Виклик

```
SET @var1 = 10;  
CALL proc_INOUT(@var1);  
SELECT @var1;
```

Виводить змінене значення на екран



Видалення процедури

DROP PROCEDURE IF EXISTS proc2;



Типи запитів

- Запит на вибірку
- Запит на зміну
 - Запит на оновлення
 - Запит на додавання
 - Запит на видалення
 - Запит на створення таблиці
- Запити з параметрами
- Перехресний запит

Процедура оновлення даних

UPDATE

```
CREATE PROCEDURE my_proc3()
```

```
UPDATE book SET Tir=Tir+1000
```

```
WHERE neww=1;
```

Збільшення тиражу
для нових книжок

Виклик процедури

```
CALL my_proc3;
```

!!! ЗМІНЮЄ таблицю

| nu | name | neww | code | price | pub | page | form | Dat | Tir |
|----|--|------|------|-------|------------------|------|-----------|------------|------|
| 4 | Аппаратные средства мультимедия. Видеосистема PC | 0 | 5110 | 15.51 | БНВ С.-Петербург | 400 | 70x100/16 | 2000-07-24 | 5000 |
| 8 | Освой самостоятельно модернизацию и ремонт ПК за 2 | 0 | 4985 | 18.9 | Вильямс | 288 | 70x100/16 | 2000-07-07 | 5000 |
| 17 | Аппаратные средства мультимедия. Видеосистема PC | 1 | 5110 | 15.51 | БНВ С.-Петербург | 400 | 70x100/16 | 2000-07-24 | 7000 |
| 18 | Освой самостоятельно модернизацию и ремонт ПК за 2 | 0 | 4985 | 18.51 | Вильямс | 288 | 70x100/16 | 2000-07-07 | 5000 |



Зміст

- Складений оператор **BEGIN ... END**
- Зміна роздільника **Delimiter**
- Створення змінної **DECLARE**
- Оператори контролю потоку даних
- Обробники умов



Створення блоку операторів

BEGIN

... список операторів, розділених ;

END

Операторні дужки – використовується для написання складених висловлювань

Як правило, використовуються на початку та в кінці збереженої процедури, але це не обов'язково. Це необхідно для циклів, операторів IF тощо, де потрібно більше одного кроку

Зміна роздільника Delimiter

delimiter //

CREATE PROCEDURE

dorepeat(p1 INT)

BEGIN

SET @x = 0; ✓

REPEAT

SET @x = @x + 1; ✓

UNTIL @x > p1

END REPEAT; ✓

END //

delimiter ;

Роздільник - символ або рядок символів, яка використовується для закриття оператора SQL. За замовчуванням як роздільник крапка з комою (;). Але це викликає проблеми в збережених процедурах і тригерах MySQL, оскільки вона може мати багато операторів, і кожен повинен закінчуватися крапкою з комою. Тому як роздільник можна використовувати, наприклад, подвійний знак долара - \$\$, або «//».

Щоб пізніше знову використовувати як роздільник «;» виконайте команду «DELIMITER; \$\$» або «DELIMITER; //»

Створення змінної в SP

DECLARE *var_name* [, *var_name*] ... *type*
[DEFAULT *value*]

- DECLARE дозволено лише всередині BEGIN ... END складеного твердження і має бути на його початку, перед будь-якими іншими твердженнями

!!! Ім'я змінної не повинне співпадати з назвою поля

```
CREATE PROCEDURE sp1 (x VARCHAR(5))  
BEGIN
```

```
    DECLARE xname VARCHAR(5) DEFAULT 'bob';
```

```
    DECLARE newname VARCHAR(5);
```

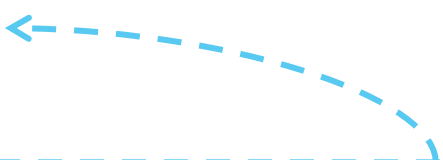
```
    DECLARE xid INT;
```

```
    SELECT xname, id INTO newname, xid FROM table1
```

```
    WHERE xname = xname;
```

```
    SELECT newname;
```

```
END;
```



newname змінна повертає
значення 'bob' незалежно від
значення table1.xname стовпця

Створення змінної в SP

❑ `DECLARE @myvar = 1 + 2`

Створює змінну, що називається, @myvar і присвоює їй значення 3

❑ `DECLARE @num =
(SELECT COUNT(*) FROM mytable)`

Створює названу змінну @num і встановлює її на кількість рядків у mytable

Створення змінної в SP

- DECLARE @foo;

Створює названу змінну @foo та встановлює їй NULL.

- DECLARE PARAMETER @requiredParam;

Створює змінну параметра @requiredParam. Оскільки не вказано *початкове значення*, потрібно вказати значення для цього параметра

Створення змінної в SP

□ DECLARE PARAMETER @optionalParam = 5;

Створює змінну параметра, яку називають @optionalParam. Оскільки вказано *початкове значення 5*, не потрібно вказувати значення для цього параметра, але це можна зробити, якщо потрібно замінити стандартне значення



Приклад

```
CREATE PROCEDURE p(increment INT)
BEGIN
    DECLARE counter INT DEFAULT 0;
    WHILE counter < 10 DO
        -- ... do work ...
        SET counter = counter + increment;
    END WHILE;
END;
```

Присвоєння значення змінній

SET *variable* = *expr* [, *variable* = *expr*] ... *variable* :

{ *user_var_name*

| *param_name*

| *local_var_name*

| {GLOBAL | @@GLOBAL.} *system_var_name*

| {PERSIST | @@PERSIST.} *system_var_name*

| {PERSIST_ONLY | @@PERSIST_ONLY.} *system_var_name*

| [SESSION | @@SESSION. | @@] *system_var_name*

}

Приклад змінної користувача

- ❑ SET @name = 43;
- ❑ SET @total_tax = (SELECT SUM(tax)
FROM taxable_transactions);



Оператори контролю потоку даних

- IF
- CASE
- ITERATE
- LEAVE
- LOOP
- WHILE
- REPEAT



IF ... THEN ...

```
IF умова THEN оператор(и)
    [ELSEIF умова
        THEN оператор(и)] ...
[ELSE оператор(и)]
END IF
```



CASE ...

CASE значення

WHEN значення THEN список_операторів
[WHEN значення THEN список_операторів]

...

[ELSE список_операторів]

END CASE



REPEAT ...

REPEAT список_операторів

UNTIL умова_пошуку

END REPEAT



LOOP ...

LOOP список_операторів
END LOOP

DO ...

DO список_операторів
END WHILE



Питання

- Чим відрізняються процедури від функцій?
- Які переваги процедур?
- Які недоліки процедур?
- Як використати змінну у збереженій процедурі?