

4.2 Створення і використання збережених процедур / Creating and using stored procedures

На практиці часто потрібно повторювати послідовність однакових запитів
On practice it is often required to repeat a sequence of similar queries

Збережені процедури дозволяють об'єднати послідовності таких запитів і зберегти їх на сервері

Stored procedures allow to merge sequences of such queries and to store them on a server

Після цього клієнтам достатньо надіслати один запит на виконання збереженої процедури

Then clients only have to execute a single query in order to run a stored procedure

4.2 Створення і використання збережених процедур / Creating and using stored procedures

Переваги збережених процедур / Advantages of stored procedures:

1 Повторне використання коду / Code reuse

після створення збереженої процедури її можна викликати з будь-яких застосувань і SQL запитів

after a stored procedure is created it can be executed by any applications and SQL queries

2 Скорочення мережевого трафіку / Reduce a network traffic

замість декількох запитів на сервер можна надіслати запит на виконання збереженої процедури і відразу отримати відповідь

instead of running several queries a single query can be sent to a server in order to execute a stored procedure and take the complete answer

4.2 Створення і використання збережених процедур / Creating and using stored procedures

3 Безпека / Security

для виконання процедури користувач повинен володіти привілеєм
a user should have a privilege in order to execute a stored procedure

4 Простота доступу / Ease of access

збережені процедури дозволяють інкапсулювати складний код і
оформити його у вигляді простого виклику

stored procedures allow to encapsulate a complex code and run it by using a
simple call

5 Виконання бізнес-логіки / Execution of business logic

бізнес-логіка в вигляді збережених процедур не залежить від мови
розробки програми

business logic provided by stored procedures does not depend on application
programming language

4.2 Створення і використання збережених процедур / Creating and using stored procedures

```
CREATE PROCEDURE procedure_name ( [ parameter [, ...] ] )  
[ characteristic ... ] procedure_body
```

У дужках передається необов'язковий список параметрів, перерахованих через кому

The unnecessary list of parameters is passed in parentheses

Кожен параметр дозволяє передати в процедуру (з процедури) вхідні дані (результат роботи)

Each parameter allows to pass input data (or the result of execution) into procedure (or retrieve from a procedure)

4.2 Створення і використання збережених процедур / Creating and using stored procedures

[IN | OUT | INOUT] parameter_name type

IN

дані передаються всередину збереженої процедури
data passed into a stored procedure

при виході з процедури нове значення для такого параметру
не зберігається

a new value for such parameter will not be stored after the
procedure is completed

4.2 Створення і використання збережених процедур / Creating and using stored procedures

OUT

дані передаються зі збереженої процедури

data retrieved from a stored procedure

початкове значення такого параметра не береться до уваги всередині процедури

initial value of such parameter will not be used in a stored procedure

INOUT

береться до уваги всередині процедури, зберігає значення

such parameter is used within a stored procedure and its value will be stored after the procedure is completed, as well

4.2 Створення і використання збережених процедур / Creating and using stored procedures

Список аргументів, укладених в круглі дужки, необхідно вказувати завжди

It is always required to provide a list of arguments within parentheses

Якщо аргументи відсутні, слід використовувати порожній список
You should use the empty list if there are no arguments required

Якщо жоден з модифікаторів не вказано, вважається, що параметр оголошений з ключовим словом IN

If there is no modifier (parameter's type) provided, the parameter will be treated as if this parameter is provided with the modifier IN

4.2 Створення і використання збережених процедур / Creating and using stored procedures

Тілом процедури є складовою оператор BEGIN ... END, всередині якого можуть розташовуватися інші оператори

The composite operator BEGIN ... END is considered as the procedure's body in which another operators can be placed

[label:] BEGIN
statements
END [label]

```
CREATE PROCEDURE myProc()  
  outer_block: BEGIN  
    DECLARE l_status int;  
    SET l_status=1;  
    inner_block: BEGIN  
      IF (l_status=1) THEN  
        LEAVE inner_block;  
      END IF;  
      SELECT 'This statement will never be executed';  
    END inner_block;  
    SELECT 'End of program';  
  END outer_block$$
```

Оператор, що починається з необов'язкової мітки label (будь-яке унікальне ім'я) може закінчуватися виразом END label.

Operator that starts with the optional label *label* (any unique name) can be ended with the statement END *label*.

4.2 Створення і використання збережених процедур / Creating and using stored procedures

При роботі з збереженими процедурами символ крапки з комою в кінці запиту сприймається консольним клієнтом як сигнал до відправлення запиту на сервер

When working with stored procedures, the semicolon at the end of the query is considered by the console client as a signal to send a query to the server

Тому слід перевизначити роздільник запитів – наприклад, замість крапки з комою використовувати послідовність //

Therefore, you should override the query separator – for example, instead of a semicolon, use the sequence //

```
mysql> DELIMITER //
```

<pre>mysql> SELECT VERSION< >//</pre>	<pre>mysql> CREATE PROCEDURE my_version()</pre>
<pre>+-----+</pre>	<pre>-> BEGIN</pre>
<pre> VERSION< > </pre>	<pre>-> SELECT VERSION<>;</pre>
<pre>+-----+</pre>	<pre>-> END //</pre>
<pre> 5.0.51b-community-nt </pre>	<pre>Query OK, 0 rows affected (0.00 sec)</pre>
<pre>+-----+</pre>	
<pre>1 row in set (0.00 sec)</pre>	

4.2 Створення і використання збережених процедур / Creating and using stored procedures

Щоб викликати збережену процедуру, необхідно застосувати оператор CALL, після якого передається ім'я процедури і її параметри в круглих дужках

To call a stored procedure, use the CALL statement, followed by the name of the procedure and its parameters in parentheses

```
mysql> CALL my_version()
```

```
+-----+  
! VERSION(  
+-----+  
! 5.0.51b-community-nt !  
+-----+  
1 row in set (0.00 sec)
```

```
Query OK, 0 rows affected (0.00 sec)
```

Рекомендується уникати використання назв збережених процедур, які збігаються з іменами вбудованих функцій MySQL

It is recommended to avoid using names of stored procedures that match the names of MySQL built-in functions

4.2 Створення і використання збережених процедур / Creating and using stored procedures

У тілі процедури, що можна використовувати багаторядковий коментар, який починається з послідовності `/ *` і закінчується послідовністю `* /`

In the body of the stored procedure, you can use a multi-line comment that starts with the sequence `/ *` and ends with the sequence `* /`

Процедура привласнює користувачській змінній `@x` нове значення

The procedure assigns the user variable `@x` a new value

```
mysql> CREATE PROCEDURE set_x<IN value INT>
-> BEGIN
-> SET @x = value;
-> END//
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>
mysql> CALL set_x(123456)//
Query OK, 0 rows affected (0.00 sec)
```

4.2 Створення і використання збережених процедур / Creating and using stored procedures

Через параметр value процедурі передається числове значення 123456, яке вона привласнює користувачській змінній @x

Through the value parameter, the procedure takes a numeric value 123456, which it assigns to the user variable @x

```
mysql> SELECT @x//
+-----+
| @x    |
+-----+
| 123456 |
+-----+
1 row in set (0.00 sec)
```

Користувачька змінна @x є глобальною, вона доступна як всередині процедури set_x (), так і поза нею

The @x user variable is global, it is available both inside the set_x() stored procedure and outside of it

Параметри збереженої процедури є локальними

Parameters of the stored procedure are local

4.2 Створення і використання збережених процедур / Creating and using stored procedures

Процедура `numcatalogs()` має один цілочисельний параметр `total`, в якому зберігається число записів в таблиці `catalogs`

The `numcatalogs()` stored procedure has one integer parameter *total*, which stores the number of entries in the *catalogs* table

```
mysql> CREATE PROCEDURE numcatalogs(OUT total INT)
-> BEGIN
-> SELECT COUNT(*) INTO total FROM catalogs;
-> END //
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> CALL numcatalogs(@a) //
```

Query OK, 0 rows affected (0.01 sec)

```
mysql> SELECT @a //
```

@a
5

1 row in set (0.00 sec)

4.2 Створення і використання збережених процедур / Creating and using stored procedures

Процедура `catalogname()` повертає по первинному ключу `catID` назву каталогу `cat_name`

The stored procedure *catalogname()* returns the catalog name *cat_name* by the primary key *catID*

```
mysql> CREATE PROCEDURE catalogname(IN id INT, OUT catalog TINYTEXT)
-> BEGIN
-> SELECT cat_name INTO catalog FROM catalogs
-> WHERE catID = id;
-> END //
```

Query OK, 0 rows affected (0.01 sec)

```
mysql> SET @id = 5//
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> CALL catalogname(@id, @name)//
```

Query OK, 0 rows affected (0.02 sec)

```
mysql> SELECT @id, @name//
```

+	-----+	-----+
	@id	@name
+	-----+	-----+
	5	Мультимедиа
+	-----+	-----+

1 row in set (0.00 sec)

4.2 Створення і використання збережених процедур / Creating and using stored procedures

Збережені процедури дозволяють реалізувати складну логіку за допомогою операторів розгалуження і циклів

Stored procedures allow to implement complex logic using branching statements and loops

IF – оператор розгалуження / branching statement

CASE – множинний вибір / multiple choice

WHILE – оператор циклу / loop statement

LEAVE – достроковий вихід з циклу / early exit from the cycle (= break)

ITERATE – дострокове завершення ітерації / early exit from the iteration (= continue)

REPEAT – оператор циклу / loop statement

4.2 Створення і використання збережених процедур / Creating and using stored procedures

IF condition **THEN** statement

[**ELSEIF** condition **THEN** statement] ...

[**ELSE** statement]

END IF ;

Логічні вирази можна комбінувати за допомогою операторів **&&** (И), а також **||** (ИЛИ)

Logical expressions can be combined with the help of the operators **&&** (AND), as well as **||** (OR)

Якщо в блоках **IF**, **ELSEIF** і **ELSE** – два або більше операторів, необхідно використовувати складовий оператор **BEGIN ... END**

If there are two or more statements in the **IF**, **ELSEIF** and **ELSE** blocks, you must use the composite **BEGIN ... END** statement

4.2 Створення і використання збережених процедур / Creating and using stored procedures

CASE expression

WHEN value **THEN** statement

[**WHEN** value **THEN** statement] ...

[**ELSE** statement]

END CASE ;

Вираз порівнюється зі значеннями

Як тільки знайдено відповідність, виконується відповідний оператор або ELSE, якщо відповідники не знайдені

The expression is compared with the values

Once a match is found, the corresponding statement or ELSE (if no match is found) is executed

4.2 Створення і використання збережених процедур / Creating and using stored procedures

```
[ label: ] WHILE condition DO  
    statements  
END WHILE [ label ] ;
```

Оператори виконуються в циклі, поки істинна умова

Operators are executed in a loop while the condition is true

Якщо в циклі виконується більше одного оператора, не обов'язково укласти їх в блок BEGIN ... END, оскільки цю функцію виконує сам оператор WHILE

If more than one statement is executed in a loop, it is not necessary to enclose them in a BEGIN ... END block, since this role is played by the WHILE statement itself

4.2 Створення і використання збережених процедур / Creating and using stored procedures

[label:] **REPEAT**

statements

UNTIL condition **END REPEAT** [label] ;

Умова перевіряється не на початку, а в кінці оператора циклу

The condition is not checked at the beginning, but at the end of the cycle operator

Слід зазначити, що цикл виконується, поки умова помилкова

It should be noted that the loop is executed while the condition is false

[label :] **LOOP**

statements

END LOOP [label];

4.2 Створення і використання збережених процедур / Creating and using stored procedures

```
DELIMITER //
```

```
CREATE PROCEDURE sp_contract_ops(IN op CHAR(1), IN c_num INT, IN c_date TIMESTAMP,  
                                IN s_id INT, IN c_note VARCHAR(100))  
BEGIN  
    IF op = 'i' THEN  
        INSERT INTO contract(contract_date, supplier_id, contract_note)  
            VALUES(CURRENT_TIMESTAMP(), s_id, c_note);  
    ELSEIF op = 'u' THEN  
        UPDATE contract SET contract_date = c_date,  
                            supplier_id = s_id,  
                            contract_note = c_note  
        WHERE contract_number = c_num;  
    ELSE  
        DELETE FROM contract WHERE contract_number = c_num;  
    END IF;  
END //
```



```
CALL sp_contract_ops('i', 0, '2018-12-16', 2, 'contract inserted');  
CALL sp_contract_ops('u', 6, '2018-12-31', 2, 'contract updated');  
CALL sp_contract_ops('d', 6, '2018-12-31', 0, '');
```

4.2 Створення і використання збережених процедур / Creating and using stored procedures

```
drop table if exists m2_products;
```

```
create table m2_products (  
    product_id int not null,  
    product_name varchar(50) not null,  
    product_price decimal(8,2) not null,  
    primary key (product_id)  
);
```

```
insert into m2_products (product_id, product_name, product_price) values  
(1, 'iPhone X', 999),  
(2, 'Samsung S10', 1099),  
(3, 'Honor 8X', 299),  
(4, 'Huawei P Smart', 199),  
(5, 'Xiaomi Mi8', 399);
```

test.m2_products: 5 rows total (approximately)

 product_id	product_name	product_price
1	iPhone X	999.00
2	Samsung S10	1,099.00
3	Honor 8X	299.00
4	Huawei P Smart	199.00
5	Xiaomi Mi8	399.00

4.2 Створення і використання збережених процедур / Creating and using stored procedures

```
delimiter $$
create or replace procedure m2_cart_ops(in op_id char(1), in p_id int, in p_amount int)
begin
    create temporary table if not exists m2_cart (
        product_id int not null,
        product_amount int not null,
        primary key (product_id)
    );

    if op_id = 'a' then
        begin
            declare p_count int;
            select count(*) into p_count from m2_cart where product_id = p_id;

            if p_count < 1 then
                insert into m2_cart (product_id, product_amount) values (p_id, p_amount);
            else
                update m2_cart set product_amount = product_amount + p_amount where product_id = p_id;
            end if;

            select concat('Product [' , p_id, ' ] x [' , p_amount, ' ] added to the cart!');
        end;
    end if;

    if op_id = 'c' then
        begin
            select 'Check out';
            select m2_cart.product_id, product_name, product_amount, product_amount * product_price as total
            from m2_cart, m2_products
            where m2_cart.product_id = m2_products.product_id;
        end;
    end if;
end $$
```

4.2 Створення і використання збережених процедур / Creating and using stored procedures


```
drop table if exists m2_cart;
```

```
call m2_cart_ops('a', 1, 2);  
call m2_cart_ops('a', 3, 1);  
call m2_cart_ops('a', 4, 4);  
call m2_cart_ops('c', 0, 0);
```

Result #1 (1×1)	Result #2 (1×1)	Result #3 (1×1)	Result #4 (1×1)
concat('Product [, p_id,] x [, p_amount,] added to the...')			
Product [1] x [2] added to the cart!			

Result #1 (1×1)	Result #2 (1×1)	Result #3 (1×1)	Result #4 (1×1)
concat('Product [, p_id,] x [, p_amount,] added to the...')			
Product [3] x [1] added to the cart!			

Result #1 (1×1)	Result #2 (1×1)	Result #3 (1×1)	Result #4 (1×1)
concat('Product [, p_id,] x [, p_amount,] added to the...')			
Product [4] x [4] added to the cart!			

Result #1 (1×1)	Result #2 (1×1)	Result #3 (1×1)	Result #4 (1×1)	Result #5 (4×3)
 product_id	product_name	product_amount	total	
1	iPhone X	2	1,998.00	
3	Honor 8X	1	299.00	
4	Huawei P Smart	4	796.00	

4.2 Створення і використання збережених процедур / Creating and using stored procedures

DROP PROCEDURE [IF EXISTS] procedure_name;

використовується для видалення збережених процедур
is used to remove stored procedures

Якщо процедура, що видаляється, з таким ім'ям не існує, оператор повертає помилку, яку можна уникнути, якщо використовувати необов'язкове ключове слово IF EXISTS

If a deleted procedure with that name does not exist, the statement returns an error that can be suppressed by using the optional keyword IF EXISTS

4.3 Створення і використання тригерів / Creating and using triggers

Тригер – збережена процедура, прив'язана до події зміни вмісту конкретної таблиці

Trigger is a stored procedure associated with the event of the contents change of a specific table

Тригер можна прив'язати до трьох подій, пов'язаних зі зміною вмісту таблиці

The trigger can be tied to three events associated with changing the contents of the table

INSERT

DELETE

UPDATE

4.3 Створення і використання тригерів / Creating and using triggers

Наприклад, при оформленні нового замовлення, тобто при додаванні нового запису в таблицю *orders*, можна створити тригер, який автоматично віднімає число замовлених товарних позицій в таблиці *books*

For example, when placing a new order, that is, when adding a new entry to the *orders* table, you can create a trigger that automatically subtracts the number of ordered items in the *books* table

```
CREATE TRIGGER trigger_name trigger_time trigger_event  
ON table_name FOR EACH ROW  
BEGIN  
    statements  
END ;
```

4.3 Створення і використання тригерів / Creating and using triggers

Оператор створює тригер з ім'ям `trigger_name`, прив'язаний до таблиці `table_name`

The operator creates a trigger named *trigger_name* associated with the table *table_name*

Не допускається прив'язка тригера до тимчасової таблиці або уявлення

Binding a trigger to a temporary table or view is not allowed

Конструкція `trigger_time` вказує момент виконання тригера

The *trigger_time* construction specifies the time at which the trigger is executed

4.3 Створення і використання тригерів / Creating and using triggers

trigger_time

може приймати два значення / can take two values:

BEFORE

дії тригера виконуються до виконання операції зміни таблиці

trigger actions are performed before performing a table change operation

AFTER

дії тригера виконуються після виконання операції зміни таблиці

trigger actions are performed after the change table operation

4.3 Створення і використання тригерів / Creating and using triggers

Конструкція `trigger_event` показує, на яку подію повинен реагувати тригер, і може приймати три значення

The *trigger_event* construct indicates which event the trigger should respond to, and can take three values

INSERT, UPDATE, DELETE

Для таблиці `table_name` може бути створений тільки один тригер для кожного з подій `trigger_event` і моменту `trigger_time`

For table *table_name*, only one trigger can be created for each of the *trigger_event* event and *trigger_time* time

Таким чином, для кожної з таблиць може бути створено всього шість тригерів

Thus, for each of the tables, only six triggers can be created

4.3 Створення і використання тригерів / Creating and using triggers

BEGIN

statements

END ;

Тіло тригера – оператор, який необхідно виконати при виникненні події *trigger_event* в таблиці *table_name*

A trigger body is an operator that must be executed when a *trigger_event* event occurs in a *table_name* table

Якщо потрібно виконати декілька операторів, то необхідно використовувати складовою оператор BEGIN ... END

If several statements are required, then the composite statement BEGIN ... END must be used

4.3 Створення і використання тригерів / Creating and using triggers

Усередині складеного оператора BEGIN ... END допускаються всі специфічні для збережених процедур оператори і конструкції:

Inside a BEGIN ... END composite statement, all operators and structures specific to stored procedures are allowed:

- інші складові оператори BEGIN ... END
- another composite operators BEGIN ... END
- оператори управління потоком (IF, CASE, WHILE, LOOP, REPEAT, LEAVE, ITERATE)
- control flow statements (IF, CASE, WHILE, LOOP, REPEAT, LEAVE, ITERATE)
- оголошення локальних змінних за допомогою оператора DECLARE і призначення їм значень за допомогою оператора SET
- local variable declarations using the DECLARE operator and assigning values to them using the SET operator

4.3 Створення і використання тригерів / Creating and using triggers

Тригери складно використовувати, не маючи доступу до нових записів, які вставляються в таблицю, або старих записів, які оновлюються або видаляються

Triggers are difficult to use without access to new records that are inserted into a table, or old records that are updated or deleted

Для доступу до нових і старих записів використовуються префікси NEW і OLD відповідно

To access new and old records, the prefixes NEW and OLD are used, respectively

Якщо в таблиці оновлюється поле *total*, то отримати доступ до старого значення можна по імені *OLD.total*, а до нового – *NEW.total*

If the *total* field is updated in the table, then the old value can be accessed by the name *OLD.total*, and the new value – *NEW.total*

4.3 Створення і використання тригерів / Creating and using triggers

Розглянемо тригер, який буде включатися до вставки нових записів в таблицю *orders* і обмежує число товарів, що замовляються до 1

Let's consider a trigger that will be called before inserting new entries into the *orders* table and limits the number of items to be ordered to 1

```
mysql> CREATE TRIGGER restrict_count BEFORE INSERT ON orders
-> FOR EACH ROW
-> BEGIN
-> SET NEW.o_number=1;
-> END//
```

Query OK, 0 rows affected (0.05 sec)

```
mysql> INSERT INTO orders VALUES (NULL,1,2,NOW(),10)//
```

Query OK, 1 row affected (0.01 sec)

```
mysql> SELECT * FROM orders WHERE orderID = LAST_INSERT_ID()//
```

orderID	o_userID	o_bookID	o_time	o_number
16	1	2	2009-10-23 20:26:19	1

1 row in set (0.00 sec)

4.3 Створення і використання тригерів / Creating and using triggers

Створимо тригер, який при оформленні нового замовлення (при додаванні нового запису в таблицю *orders*) буде збільшувати на 1 значення користувацької змінної *@tot*

Create a trigger that, when placing a new order (when adding a new entry to the *orders* table), will increase by 1 the value of the user variable *@tot*

```
mysql> delimiter //
mysql> CREATE TRIGGER sub_count AFTER INSERT ON orders
      -> FOR EACH ROW
      -> BEGIN
      -> SET @tot =@tot+1;
      -> END//
```

Query OK, 0 rows affected (0.02 sec)

```
mysql> SELECT @tot //
+-----+
| @tot |
+-----+
| NULL |
+-----+
1 row in set (0.00 sec)
```

4.3 Створення і використання тригерів / Creating and using triggers

Для коректної роботи тригера необхідно, щоб користувацька змінна `@tot` мала значення, відмінне від `NULL`, оскільки операція складання з `NULL` також призводить до `NULL`

For the trigger to work correctly, the `@tot` user variable must have a value other than `NULL`, since the addition operation with `NULL` also results in `NULL`

```
mysql> SET @tot=5//
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders VALUES (NULL,1,5,NOW(),10)//
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @tot//
+-----+
| @tot |
+-----+
| 6    |
+-----+
1 row in set (0.00 sec)
```

4.3 Створення і використання тригерів / Creating and using triggers

Створимо тригер, який при додаванні нових покупців перетворює імена та по батькові покупців в ініціали

Create a trigger that when adding new customers converts the names and patronymic of customers into initials

```
mysql> CREATE TRIGGER restrict_user BEFORE INSERT ON users
-> FOR EACH ROW
-> BEGIN
-> SET NEW.u_name = LEFT(NEW.u_name,1);
-> SET NEW.u_patronymic = LEFT(NEW.u_patronymic,1);
-> END//
```

Query OK, 0 rows affected (0.06 sec)

```
mysql> INSERT INTO users VALUES (NULL, 'Светлана', 'Петровна', 'Титова',
-> '83-89-00', NULL, 'active')//
```

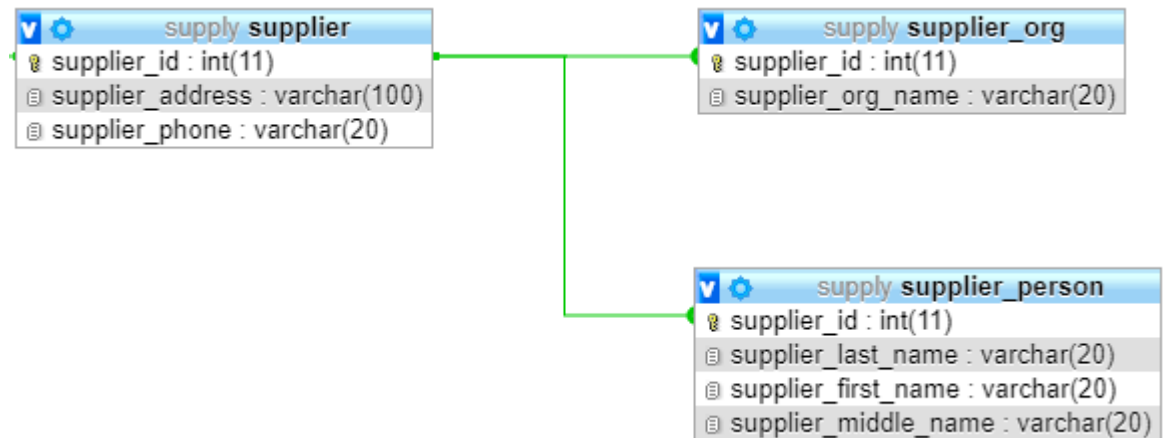
Query OK, 1 row affected (0.03 sec)

```
mysql> SELECT u_surname, u_name, u_patronymic FROM users
-> WHERE userID = LAST_INSERT_ID()//
```

u_surname	u_name	u_patronymic
Титова	С	П

1 row in set (0.00 sec)

4.3 Створення і використання тригерів / Creating and using triggers



```
DELIMITER //
```

```
CREATE TRIGGER check_supplier_org BEFORE INSERT ON supplier_person
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    IF NEW.supplier_id IN (SELECT supplier_id FROM supplier_org) THEN
```

```
        SET @message = CONCAT('The person with id ', NEW.supplier_id,
```

```
                                ' is already stored as the organization!');
```

```
        SIGNAL SQLSTATE '45001'
```

```
        SET MESSAGE_TEXT = @message;
```

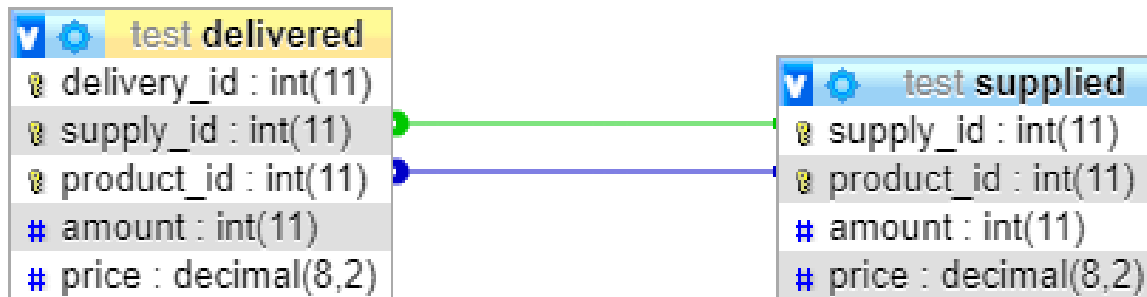
```
    END IF;
```

```
END //
```



```
INSERT INTO supplier_person VALUES (2, 'Makarov', 'Oleg', 'Petrovych');
```

4.3 Створення і використання тригерів / Creating and using triggers



```
create table delivered (  
  delivery_id int not null,  
  supply_id int not null,  
  product_id int not null,  
  amount int not null,  
  price decimal(8,2) not null,  
  primary key (delivery_id, supply_id, product_id)  
) engine=innodb;
```

```
create table supplied (  
  supply_id int not null,  
  product_id int not null,  
  amount int not null,  
  price decimal(8,2) not null,  
  primary key (supply_id, product_id)  
) engine=innodb;
```

```
alter table delivered  
add constraint foreign key (supply_id, product_id) references supplied(supply_id, product_id);
```

4.3 Створення і використання тригерів / Creating and using triggers

```
delimiter $$
create trigger tr_dlv_r_amount before insert on delivered
for each row
begin
    DECLARE available int;
    SELECT amount INTO available FROM supplied
        WHERE supplied.supply_id = NEW.supply_id AND
            supplied.product_id = NEW.product_id;
    IF available < NEW.amount THEN
        SET @message = CONCAT('Product ', NEW.product_id, ' is out of stock! Only ',
            available, ' items available. ');
        SIGNAL SQLSTATE '45001'
        SET MESSAGE_TEXT = @message;
    END IF;
end $$

insert into supplied (supply_id, product_id, amount, price)
values (1, 1, 15, 5), (1, 2, 50, 10), (1, 3, 25, 15);

insert into delivered (delivery_id, supply_id, product_id, amount, price)
values (1, 1, 1, 25, 5);
```

SQL Error (1644): Product 1 is out of stock! Only 15 items available.

4.3 Створення і використання тригерів / Creating and using triggers

```
delimiter $$
create or replace trigger tr_m2_emp_dates before insert on employee
for each row
begin
    insert into t_emp values (new.employee_id, new.first_name, new.last_name,
        new.birth_date, new.onboarding_date);

    if new.onboarding_date <= new.birth_date then
        set @inv_emp_id = new.employee_id;
    end if;
end $$
delimiter ;

drop table if exists t_emp;
create temporary table if not exists t_emp like employee;

insert into employee values (2, 'Adam', 'Lee', '1990-01-01', '1989-01-01');

delete from employee where employee_id = @inv_emp_id;

select * from employee;

select * from t_emp;
```

employee (5x1)				
employee_id	first_name	last_name	birth_date	onboarding_date
1	John	Smith	1993-04-11 00:00:00	2016-01-12 00:00:00

employee (5x1)				
employee_id	first_name	last_name	birth_date	onboarding_date
2	Adam	Lee	1990-01-01 00:00:00	1989-01-01 00:00:00