ЛАБОРАТОРНА РОБОТА №7

Розробка збережуваних процедур і функцій. Реалізація тригерів

Мета: дослідження засобів створення збережуваних процедур і функцій та реалізація тригерів для забезпечення цілісності даних.

Програмне забезпечення: СКБД на вибір студента.

ПРАКТИЧНЕ ЗАВДАННЯ

1) Розробити 3 збережувані процедури та 3 збережувані функції для бази даних з ЛР№6 (умови студент підбирає сам).

2) Реалізувати тригери на кожну з можливих ситуацій: BEFORE INSERT, AFTER INSERT, BEFORE UPDATE, AFTER UPDATE, BEFORE DELETE, AFTER DELETE для бази даних з ЛР№6 (умови студент підбирає сам).

3) Для кожної таблиці бази даних з ЛР№6 розробити журналювання операцій оновлення та видалення даних. Для цього до бази даних необхідно додати таблицю з назвою log, що має поля table – назва таблиці, operation – виконана операція, time – повний час виконання операції.

ТЕОРЕТИЧНІ ВІДОМОСТІ

SQL дозволяє використовувати різні типи збережуваних об’єктів:

– збережувана процедура – об’єкт, що створено виразом CREATE PROCEDURE та приведено у дію виразом CALL;

– збережувана функція – об’єкт, що створено виразом CREATE FUNCTION;

– тригер – об’єкт, що прив’язано до певної таблиці та активується при виконанні певної події;

– представлення – об’єкт, що є віртуальною таблицею, визначеною виразом VIEW.

Вираз для створення збережуваних процедур або функцій має наступний синтаксис:

```
CREATE
    [DEFINER = user]
    PROCEDURE [IF NOT EXISTS] sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = user]
    FUNCTION [IF NOT EXISTS] sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type
```

```
type:
    Any valid MySQL data type

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}

routine_body:
    Valid SQL routine statement
```

proc/func _parameter – це ті дані, які ми будемо передавати процедурі/функції при її виклику, а оператори – це власне запити. Зверніть увагу, як задаються параметри: необхідно дати ім'я параметру і вказати його тип, а в тілі процедури/функції ми вже використовуємо імена параметрів.

Кожна збережувана підпрограма може включати кілька виразів SQL, які розділено крапкою з комою (;).

Наприклад, нижче наведена процедура має тіло, що визначено операторами BEGIN ... END, які містять вираз SET і цикл REPEAT для повторення іншого виразу SET:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END;
```

Синтаксис виклику збереженої процедури:

```
CALL імя_процедури ('значення_параметрів');
```

Наступний приклад показує як створити та викликати процедуру:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END

CALL dorepeat(1000);

SELECT @x;
```

Результат:

```
+------+
| @x   |
+------+
| 1001 |
+------+
```

Тригер є іменованим об'єктом бази даних, який пов'язаний з таблицею, та активується, коли певна подія відбувається для цієї таблиці, а саме коли для таблиці виконується інструкція:

- додавання нового рядка (запису) в таблицю (INSERT);
- вилучення рядка (запису) в заданій таблиці (DELETE);
- зміна даних у певному стовпці заданої таблиці (UPDATE).

Тригер може бути встановлений, щоб активуватися до або після виклику інструкції. Це дуже добре, коли Ви повинні виконати перевірки значень, які будуть вставлені в таблицю або виконувати обчислення на значеннях, що включаються до модифікації. Наприклад, Ви можете мати тригер, який спрацьовує перед видаленням кожного рядка з таблиці або після кожної модифікації вже існуючого рядка в таблиці.

Вираз для створення тригеру має наступний синтаксис:

CREATE TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt

Оператор create trigger створює trigger_name, прив'язаний до таблиці tbl_name. Таблиця повинна існувати фізично, тобто не допускається прив'язка тригера до тимчасової таблиці або представлення.

Конструкція trigger_time вказує момент виконання тригера і може приймати два значення:

- BEFORE – дії тригера проводяться до виконання операції зміни таблиці.
- AFTER – дії тригера проводяться після виконання операції зміни таблиці.
- Конструкція trigger_event показує, на яке з подій повинен реагувати тригер, і може приймати три значення:
- INSERT – тригер прив'язаний до події вставки нового запису в таблицю;
- UPDATE – тригер прив'язаний до події оновлення запису таблиці;
- DELETE – тригер прив'язанй до події видалення записів таблиці.

Зауваження:

Для таблиці tbl_name може бути створений лише один тригер для кожної з подій trigger_event і моменту trigger_time. Тобто для кожної з таблиць може бути створено лише шість тригерів.

Конструкція trigger_stmt представляє тіло тригера, тобто оператор, який необхідно виконати при виникненні події trigger_event в таблиці tbl_name. Якщо потрібно виконати декілька операторів, то слід використати складений оператор BEGIN ... END, в якому розміщуються всі необхідні запити.

Взагалі синтаксис і допустимі оператори збігаються з тілом збережених процедур. Усередині складеного оператора BEGIN ... END допускаються всі специфічні для збережених процедур оператори та конструкції:

- інші складені оператори BEGIN ... END;
- оператори управління потоком (IF, CASE, WHILE, LOOP, REPEAT, LEAVE, ITERATE);
- оголошення локальних змінних за допомогою оператора DECLARE і призначення їм значень за допомогою оператора SET;
- іменовані умови і обробники помилок.

Зауваження:

Для створення тригера за допомогою оператора CREATE TRIGGER потрібна наявність привілеї SUPER. Також в СКБД MySQL ттригери не можна прив'язати до каскадного оновлення або видалення записів з таблиці типу InnoDB по зв'язку первинний ключ / зовнішній ключ.

Тригери дуже складно використовувати, не маючи доступу до нових записів, які вставляються в таблицю, або старим записам, які оновлюються або видаляються. Для доступу до нових і старих записів використовуються префікси NEW і OLD відповідно. Тобто якщо в таблиці оновлюється поле total, то отримати доступ до старого значенням можна по імені OLD.total, а до нового – NEW.total.

Приклад тригера для обчислення суми після кожної операції додавання нового рядка. Це простий приклад, який пов'язує тригер із таблицею для інструкцій INSERT.

Це діє як суматор, щоб підсумовувати значення, вставлені в один із стовпців таблиці. Наступні інструкції створюють таблицю та тригер для неї:

```
CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
CREATE TRIGGER ins_sum BEFORE INSERT ON account
FOR EACH ROW SET @sum = @sum + NEW.amount;
```

Команда CREATE TRIGGER створює тригер ins_sum, який пов'язаний із таблицею account. Це також включає пропозиції, які визначають час активації, подію виклику, і що робити з активованим тригером далі:

- ключове слово BEFORE вказує на час спрацювання. У цьому випадку тригер повинен активуватись перед кожним рядком, вставленим у таблицю. Інше допустиме ключове слово тут: AFTER;
- ключове слово INSERT вказує на подію, яка активує тригер. У цьому прикладі тригер спрацьовує від інструкції INSERT. Ви також можете створювати тригери для інструкцій DELETE і UPDATE.
- інструкція FOR EACH ROW визначає, що тригер повинен спрацювати один раз для кожного рядка, на який впливає інструкція в прикладі. Власне тригер являє собою в даному випадку простий SET, який накопичує значення, вставлені в стовпець amount. Інструкція звертається до стовпця як NEW.amount, що означає «значення стовпця amount, яке буде вставлено в новий рядок».

Щоб використовувати тригер, встановіть змінну суматора в нуль, виконайте інструкцію INSERT, а потім перегляньте, яке значення змінна має пізніше:

```
SET @sum = 0;
INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
SELECT @sum AS 'Total amount inserted';
```

Результат:

```
+----------------------+
| Total amount inserted |
+----------------------+
| 1852.48              |
+----------------------+
```

У цьому випадку значення @sum після виконання команди INSERT дорівнює 14.98 + 1937.50 – 100 або 1852.48.

Оператор DROP TRIGGER дозволяє видаляти існуючі тригери і має наступний синтаксис:

```
DROP TRIGGER tbl_name.trigger_name
```

Тут оператор видаляє тригер з ім'ям trigger_name таблиці tbl_name.

Приклад. Демонструється видалення тригера restrict_user таблиці users

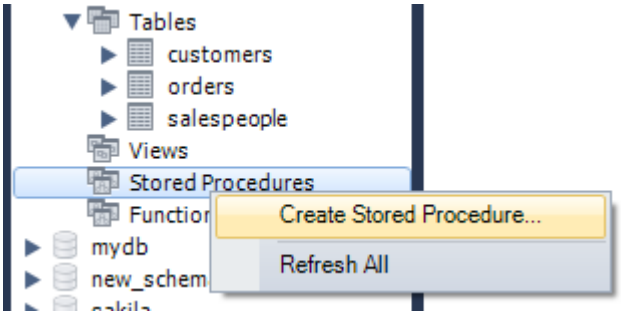
```
DROP TRIGGER users.restrict_user;
```

Тригери можна включати і вимикати за допомогою команди ALTER:

```
ALTER TABLE <ім'я таблиці>
<ENABLE|DISABLE> TRIGGER <ALL|<ім'я тригера>>
```

РОБОТА В MYSQL WORKBENCH

I. В MySQL Workbench збережувана процедура створюється натисканням правою кнопкою миші на Stored Procedure → Create Stored Procedure…

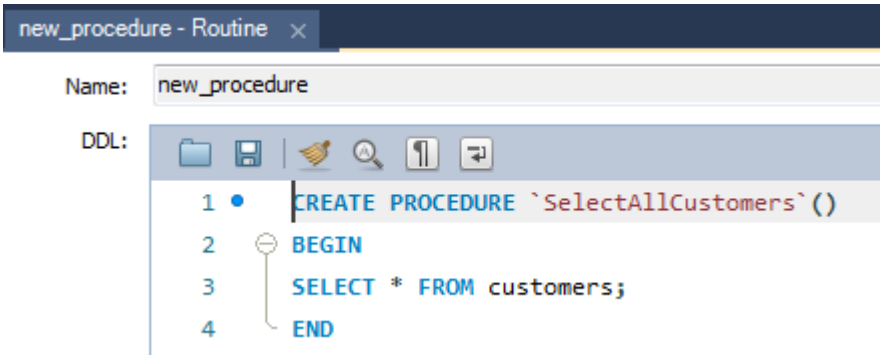


Далі вводимо назву збережуваної процедури і натискаємо кнопку Apply.

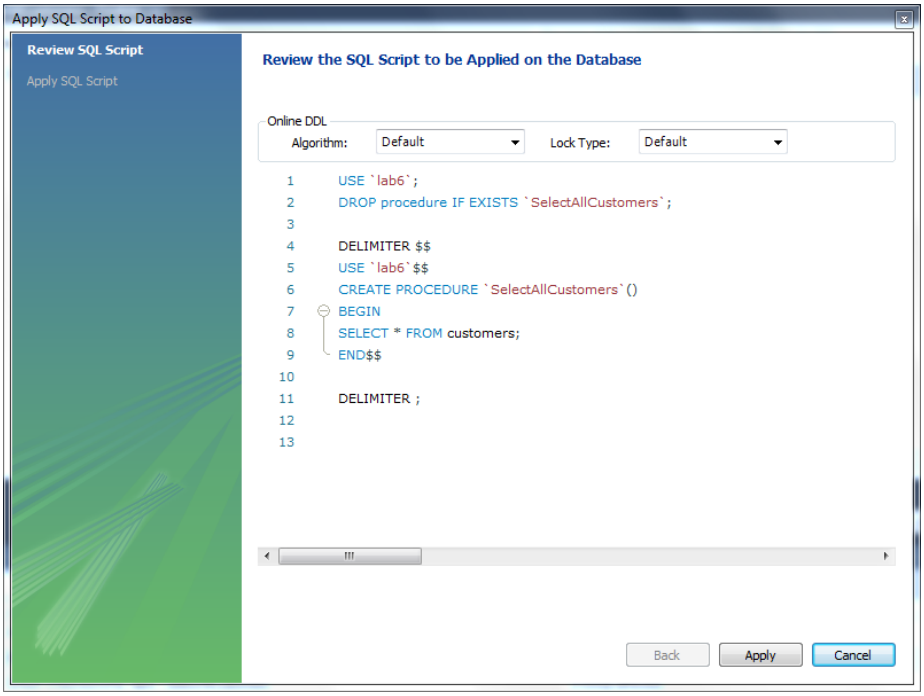
Наприклад, створимо процедуру з ім'ям SelectAllCustomers() для отримання всіх полів таблиці 1.

Таблиця 1 – Customers (Покупці)

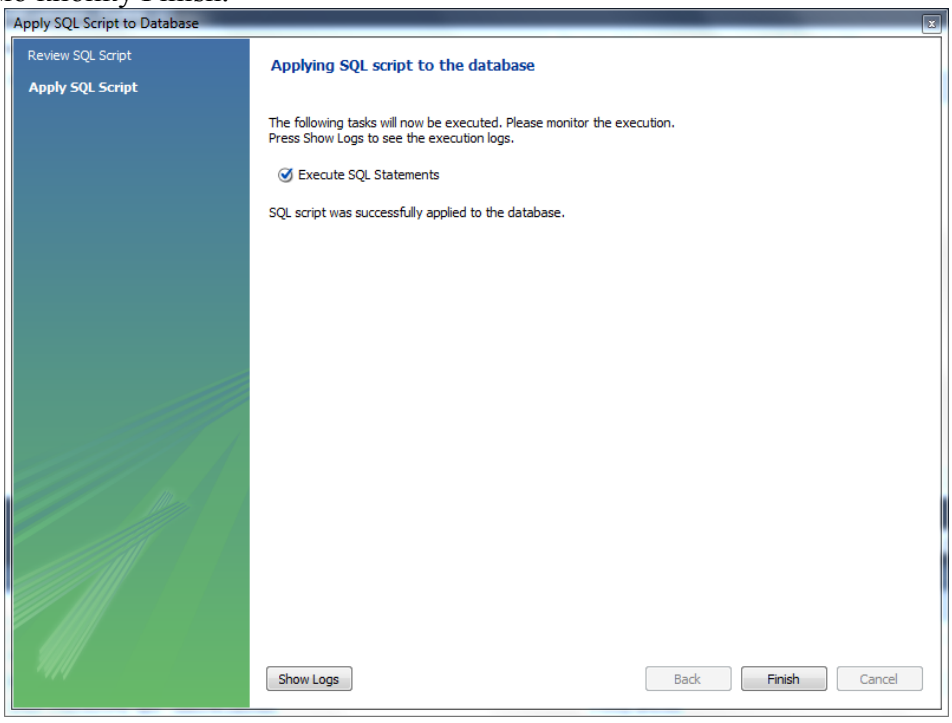
cnum	cname	city	rating	snum
2003	Liu	San Jose	200	1002
2004	Grass	Berlin	300	1002
2006	Clemens	London	100	1001
2007	Pereira	Rome	100	1004
2008	Cisneros	San Jose	300	1007



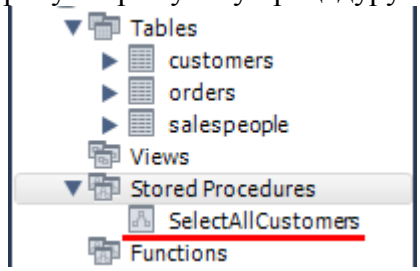
Ви можете переглянути повний код, який відправиться в MySQL, перед тим, як він запишеться в базу даних. Якщо помилок немає, натискаємо Apply.



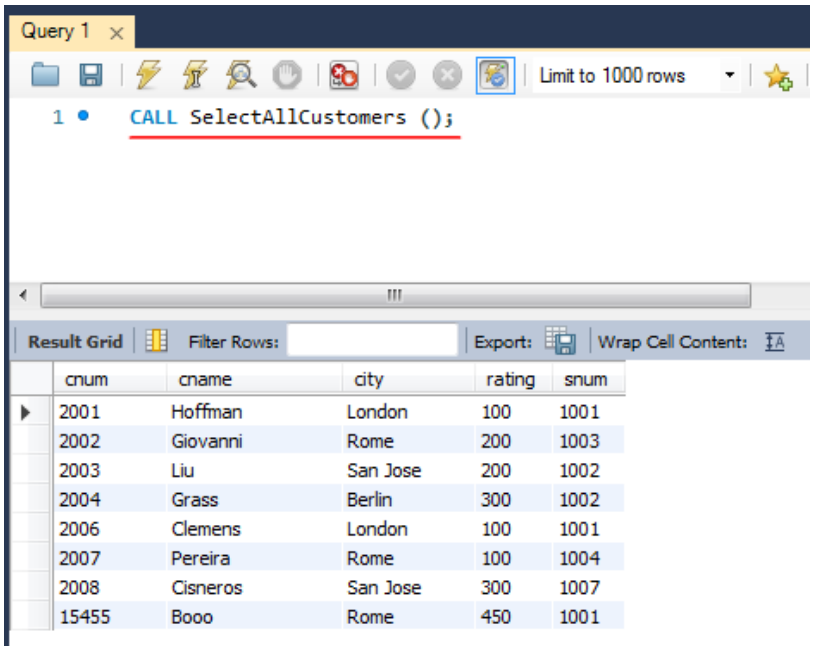
Після компіляції MySQL записує процедуру в каталог. Після завершення запису натискаємо кнопку Finish.



Можна побачити створену збережувану процедуру в списку Stored Procedures:



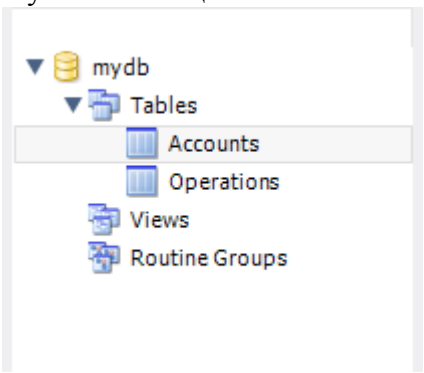
Для виклику збережуваної процедури, використовується вбудована SQL команда CALL:



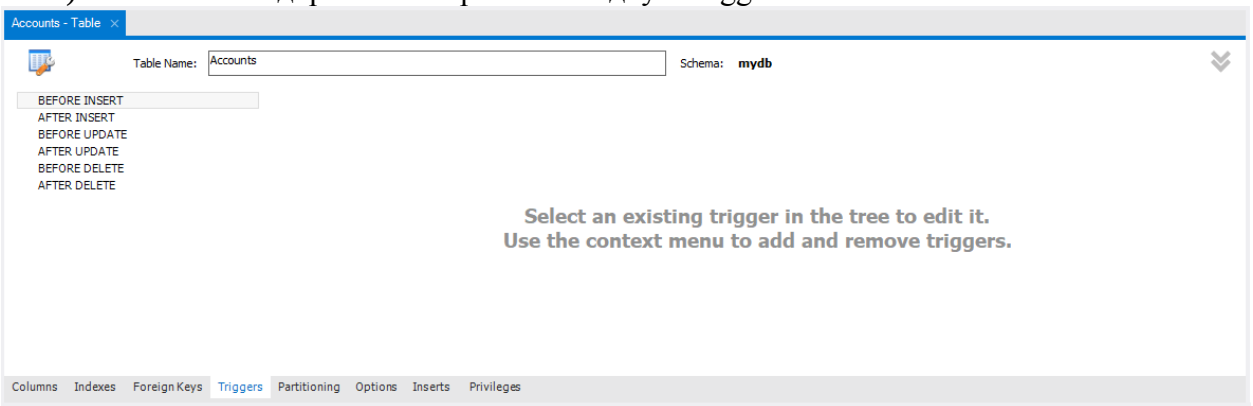
cnum	cname	city	rating	snum
2001	Hoffman	London	100	1001
2002	Giovanni	Rome	200	1003
2003	Liu	San Jose	200	1002
2004	Grass	Berlin	300	1002
2006	Clemens	London	100	1001
2007	Pereira	Rome	100	1004
2008	Cisneros	San Jose	300	1007
15455	Booo	Rome	450	1001

II. Для того, щоб створити тригер в MySql Workbench потрібно виконати наступні кроки:

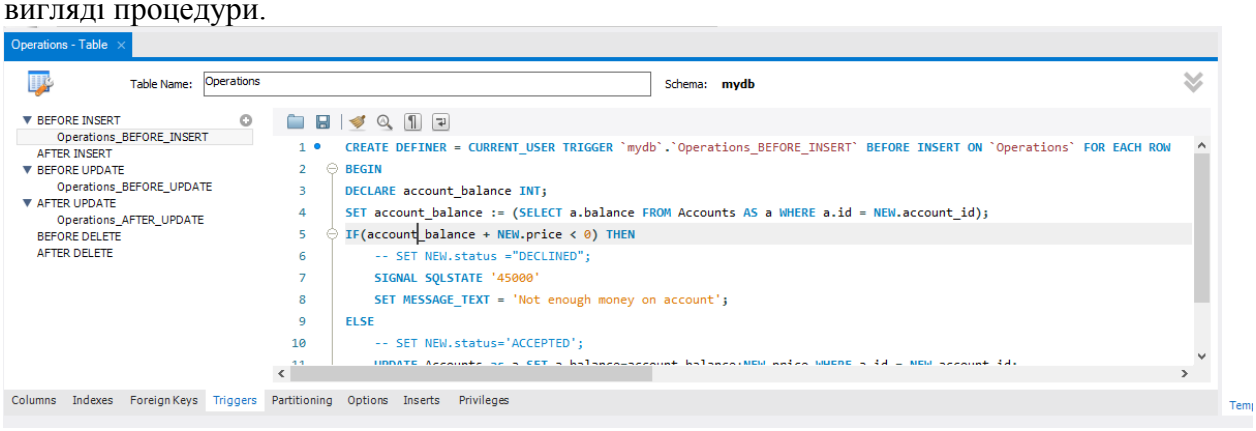
1) Натискаємо на потрібну нам таблицю



2) В вікні яке відкрилось обираємо вкладку «Triggers»



3) Обираємо потрібний вид тригера і описуємо потрібний функціонал тригера в вигляді процедури.



4) Переносимо з моделі в реальну базу даних, і перевіряємо за допомогою Insert, Update, Delete запитів.

ПРИКЛАДИ ЗБЕРЕЖУВАНИХ ПРОЦЕДУР

Приклад. Додавання нового запису в таблицю 1:

```
CREATE PROCEDURE `InsertNewCustomers` (cn int(11), nm char(10), ct char(10), rt int(11), sn int(11))
    BEGIN
        INSERT into customers (cnum, cname, city, rating, snum) value (cn, nm, ct, rt, sn);
    END
```

Виклик процедури:

```
CALL InsertNewCustomers ('2020', 'Flower', 'Paris', '150', '1003');
```

cnum	cname	city	rating	snum
2002	Giovanni	Rome	200	1003
2003	Liu	San Jose	200	1002
2004	Grass	Berlin	300	1002
2006	Clemens	London	100	1001
2007	Pereira	Rome	100	1004
2008	Cisneros	San Jose	300	1007
2020	Flower	Paris	150	1003

Приклад. Збережувана процедура, яка буде показувати нам список покупців з міста, який ми передамо в якості аргументу.

```
CREATE PROCEDURE `get_customers` (city_arg char(10))
BEGIN
    SELECT cname, city
    FROM customers
    WHERE city = city_arg;
END
```

Виклик процедури:

```
CALL get_customers ('Rome');
```

cname	city
Giovanni	Rome
Pereira	Rome
Booo	Rome

Приклад. Створимо процедуру, яка в якості параметра отримує прізвище співробітника і друкує список всіх договорів, якими він керує.

```
CREATE PROCEDURE 'show_contracts' (v_staff_name CHAR(50))
BEGIN
        SELECT contract_num, contract_date, contract_type
        FROM k_contract c JOIN k_staff s ON c.k_staff_staff_num=s.staff_num
        WHERE s.staff_name=v_staff_name;
END
```

Виклик процедури:

```
CALL show_contracts ('Іванов');
```

Приклад. Створимо процедуру, яка отримує номер місяця і номер року та друкує договори за вказаний період.

```
CREATE PROCEDURE 'find_contracts_by_month_and_year' (v_month int(11),
v_year int(11))
BEGIN
        SELECT contract_num, contract_date, contract_type
        FROM k_contract
        WHERE MONTH (contract_date) = v_month AND YEAR (contract_date) =
v_year;
END
```

Виклик процедури:

```
CALL find_contracts_by_month_and_year ('11', '2011');
```

Приклад. Створити процедуру, яка визначає кращого продавця за сумарним значенням продажів за період дат, отриманих в аргументах.

```
CREATE PROCEDURE best_salespeople (d1 DATE, d2 DATE)
BEGIN
        SELECT sname, SUM(amt) AS "Сумарні продажі"
        FROM salespeople
        INNER JOIN orders ON orders.snum = salespeople.snum AND odate
BETWEEN d1
        AND d2
        GROUP BY sname
        ORDER BY SUM(amt) DESC
        LIMIT 1;
END
```

Виклик процедури:

```
CALL best_salespeople ('2010-01-01', '2019-01-01');
```

Приклад. Створити процедуру, яка відобразить інформацію про виконанні замовлення, вартість яких вище числа, заданого в аргументі.

```
CREATE PROCEDURE info_orders (arg int(10))
BEGIN
        SELECT onum, amt, odate
         FROM orders
         WHERE amt > arg;
END
```

Виклик процедури:

```
CALL info_orders ('100');
```

ПРИКЛАДИ ТРИГЕРІВ

Приклад. Використання ключового слова AFTER

Тригер, який при оформленні нового замовлення (додавання нового запису в таблицю orders) буде присвоювати значення 1 для змінної користувача @tot.

```
CREATE TRIGGER sub_count AFTER INSERT ON orders
FOR EACH ROW
        BEGIN
                SET @tot = 1;
        END

INSERT INTO orders VALUES (NULL,1, NOW(),1,10);
SELECT @tot;
```

Як видно з прикладу, в результаті додавання нового запису в таблицю orders змінній @tot присвоюється значення 1.

Приклад.

Відредагуємо тригер sub_count таким чином, щоб до змінної @tot додавалося щоразу число замовлених товарних позицій number.

```
CREATE TRIGGER sub_count AFTER INSERT ON orders
FOR EACH ROW
        BEGIN
                SET @tot = @tot + NEW.number;
        END

INSERT INTO orders VALUES (NULL,1,NOW(),5,10);
SELECT @tot;
```

Як видно з прикладу, для того щоб отримати число товарних позицій, всередині тригера відбувається звернення до змінної NEW.number, яка пов'язана з полем number INSERT-запиту.

Приклад. Використання ключового слова BEFORE

Попередні два приклади демонстрували роботу тригерів після додавання запису в таблицю (AFTER) без втручання в запит. Розглянемо тригер, який буде викликатися до (BEFORE) вставки нових записів в таблицю orders. Основне завдання тригера полягає в обмеженні числа замовлених товарів до 1.

```
CREATE TRIGGER restrict_count BEFORE INSERT ON orders
FOR EACH ROW
        BEGIN
                SET NEW.number = 1;
        END

INSERT INTO orders VALUES (NULL,1,NOW(),2,10);
```

Як видно з прикладу, незважаючи на те, що замовлення оформлялося на дві товарні позиції, тригер restrict_count змінив значення поля number на 1.

Приклад. Контролюючий тригер

Часто при оновленні одних полів таблиці оператори баз даних забувають оновити пов'язані поля таблиці або здійснюється спроба додавання некоректних значень. Нехай при додаванні нового клієнта необхідно перетворити імена та по батькові клієнтів в ініціали. Тригер для обробки цієї ситуації може виглядати так, як це представлено в наступному прикладі.

```
CREATE TRIGGER restrict_user BEFORE INSERT ON users
FOR EACH ROW
        BEGIN
                SET NEW.name = LEFT(NEW.name,1);
                SET NEW.patronymic = LEFT(NEW.patronymic,1);
        END

INSERT INTO users VALUES (NULL, 'Ремізов', 'Олексійович', 'Сергій', '83-89-00'
f NULL, NULL, 'active')
SELECT surname, patronymic, name FROM users WHERE id_user =
LAST_INSERT_ID()
```

Як видно з прикладу, ім'я та по-батькові кожного нового відвідувача урізується до однієї літери. Для того, щоб ім'я та по-батькові не було відредаговано за допомогою оператора update, слід також створити тригер, прив'язаний до події update.

Приклад.

Написати тригер, що забороняє коректувати дані в таблиці progress між сесіями.

```
CREATE TRIGGER ProgressTerm ON progress
FOR INSERT, UPDATE, DELETE
AS
        IF EXISTS
```

```
(SELECT 'TRUE' FROM progress
        WHERE (DATEPART(mm,getDATE()) <>'01' AND NTerm%2=1)
OR (DATEPART(mm,getDATE()) <>'06' AND NTerm%2=0))

BEGIN
        RAISERROR('Не можна виправляти оцінку!!!',20,1)
        *-- Відкат транзакції в разі виникнення помилки*/
        ROLLBACK TRAN
END
```

Тепер будь-яка спроба вставити або змінити дані в період відмінний від обумовленого, наприклад спроба 14 грудня виконати дії:

```
UPDATE progress SET mark=2 WHERE NRecordBook='050001'
INSERT INTO progress VALUES ('050001',1,2,1,4,5)
```

викличе повідомлення:

Server: Msg 50000 – Не можна виправляти оцінку!!!

Приклад.

Створити тригер, що забороняє змінювати записи для непарного семестру завжди, окрім січня, для парних семестрів завжди окрім червня.

```
CREATE TRIGGER ProgressTerm ON progress
FOR INSERT, UPDATE, DELETE
AS
    IF EXISTS
    (SELECT 'TRUE' FROM progress
            WHERE (DATEPART(mm,getDATE()) <>'01' AND NTerm%2=1)
    OR (DATEPART(mm,getDATE()) <>'06' AND NTerm%2=0))

    BEGIN
            RAISERROR('Сесія завершена! Правка заборонена !!!',16,1)
            *-- Відкат транзакції в разі виникнення помилки*/
            ROLLBACK TRAN
    END
```

Тепер спроба введення або редагування даних в період між сесіями:

```
UPDATE progress SET MARK=2 WHERE NRECORDBOOK='050001'
INSERT INTO progress VALUES ('050001',1,2,1,4,5)
UPDATE progress SET mark=2 WHERE NRecordBook='050001'
```

потерпить невдачу і буде видано повідомлення:

Server: Msg 50000 – Сесія завершена! Правка заборонена !!!

Приклад.

Написати тригер, що видаляє рядки в таблиці progress що відносяться до записів student, що видаляється з відношення.

```
CREATE TRIGGER StudentProgress ON student
FOR INSERT, UPDATE, DELETE
AS
        DECLARE @COUNT int
        SELECT @COUNT=COUNT(*) FROM DELETED
```

```
*--Перевіряємо чи видалялися з головної таблиці student які-небудь
записи*/
*--Якщо так, то видалення необхідно виконати й із залежної таблиці*/
IF @COUNT>0
        BEGIN
                DELETE FROM progress
                FROM DELETED D JOIN progress P ON
                D.NRecordBook=P.NRecordBook
        END
```

Приклад.

Створити тригер, що забороняє виправляти оцінку відносно таблиці progress на вищу.

```
CREATE TRIGGER Update1Progress ON progress
FOR UPDATE
        AS
           IF EXISTS
           (SELECT 'TRUE' FROM INSERTED I LEFT JOIN DELETED D
           ON D.NRecordBook=I.NRecordBook WHERE I.mark>D.mark)
                BEGIN
                        RAISERROR('Не можна виправляти оцінку!!!',16,1)
                        *-- Відкат транзакції в разі виникнення помилки*/
                        ROLLBACK TRAN
                END
```

Тепер виконання команди, що намагається замінити оцінку 3 на оцінку 4

```
UPDATE progress
SET mark=4
WHERE NRecordBook='050001'
```

завершиться наступним повідомленням:

Server: Msg 50000 – Не можна виправляти оцінку!!!

Приклад.

Тригер, який перевіряє нове значення, яке потрібно використовувати для модифікування кожного рядка, і змінює значення, щоб залишатися всередині діапазону від 0 до 100. Використовуємо UPDATE для перевірки нового значення та BEFORE, тому що значення має бути перевірено перш, ніж воно використовується, щоб модифікувати рядок:

```
CREATE TRIGGER upd_check BEFORE UPDATE ON account
FOR EACH ROW
   BEGIN
      IF NEW.amount < 0 THEN SET NEW.amount = 0;
         ELSEIF NEW.amount > 100 THEN SET NEW.amount = 100;
      END IF;
   END;
```

Приклад. Реалізація логу.
Початкові дані:

- таблиця за якою будемо слідкувати:

```
CREATE TABLE `test` (
        `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY
        KEY,
        `content` TEXT NOT NULL)
ENGINE = MYISAM
```

- таблиця логу:

```
CREATE TABLE `log` (
        `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY
        KEY,
        `msg` VARCHAR(255) NOT NULL,
        `time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
        `row_id` INT(11) NOT NULL)
ENGINE = MYISAM
```

- тригер:

```
CREATE TRIGGER `update_test` AFTER INSERT ON `test`
FOR EACH ROW
        BEGIN
                INSERT INTO log Set msg = 'insert', row_id = NEW.id;
        END;
```

Тепер, якщо додати запис до таблиці test, то у таблиці log теж з'явиться запис. Зверніть увагу на поле row_id, у ньому зберігається id вставленого вами рядка.

Приклад. Реалізація розширеного логу.
Початкові дані:

- видаляємо тригер:

```
DROP TRIGGER `update_test`;
```

- створимо ще одну таблицю в якій будуть зберігатись копії рядків із таблиці test:

```
CREATE TABLE `testing`.`backup` (
        `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY
        KEY,
        `row_id` INT(11) UNSIGNED NOT NULL,
        `content` TEXT NOT NULL)
ENGINE = MYISAM
```

- тригери:

```
CREATE TRIGGER `update_test` before update ON `test`
FOR EACH ROW
        BEGIN
```

```
            INSERT INTO backup Set row_id = OLD.id, content = OLD.content;
        END;

    CREATE TRIGGER `delete_test` before delete ON `test`
    FOR EACH ROW
        BEGIN
            INSERT INTO backup Set row_id = OLD.id, content = OLD.content;
        END;
```

Тепер, якщо ми відредагуємо або видалимо рядок з test, вона скопіюється в backup.

Контрольні запитання

1) Які обмеження характерні для підпрограм у SQL?
2) Які відмінності між процедурами та функціями у SQL?
3) Чи дозволяють збережувані підпрограми збільшити швидкість роботи інформаційної системи?
4) Що таке тригер?
5) Які обмеження мають тригери?
6) Які переваги застосування тригерів?