

МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА
ПОЛІТЕХНІКА»

Кафедра систем штучного інтелекту



Лабораторна робота №12
З курсу “Організація баз даних та знань”

Виконав:
студент групи КН-210
Марій Павло

Викладач:
Мельникова Наталя Іванівна

Тема: Розробка та застосування тригерів.

Мета: Розробити SQL запити, які моделюють роботу тригерів: каскадне знищення, зміна та доповнення записів у зв'язаних таблицях.

Теоретичні відомості

Тригер – це спеціальний вид користувацької процедури, який виконується автоматично при певних діях над таблицею, наприклад, при додаванні чи оновленні даних. Кожен тригер асоційований з конкретною таблицею і подією. Найчастіше тригери використовуються для перевірки коректності вводу нових даних та підтримки складних обмежень цілісності. Крім цього їх використовують для автоматичного обчислення значень полів таблиць, організації перевірок для захисту даних, збирання статистики доступу до таблиць баз даних чи реєстрації інших подій.

Для створення тригерів використовують директиву CREATE TRIGGER.

Синтаксис:

CREATE

[DEFINER = { *користувач* | CURRENT_USER }]

TRIGGER *ім'я_тригера* *час_виконання подія_виконання*

ON *назва_таблиці* FOR EACH ROW *тіло_тригера*

Аргументи:

DEFINER

Задає автора процедури чи функції. За замовчуванням – це CURRENT_USER.

ім'я_тригера

Ім'я тригера повинно бути унікальним в межах однієї бази даних.

час_виконання

Час виконання тригера відносно події виконання. BEFORE – виконати тіло тригера до виконання події, AFTER – виконати тіло тригера після події.

подія_виконання

Можлива подія – це внесення (INSERT), оновлення (UPDATE), або видалення (DELETE) рядка з таблиці. Один тригер може бути пов'язаний лише з однією подією. Команда AFTER INSERT, AFTER UPDATE, AFTER DELETE визначає виконання тіла тригера відповідно після внесення, оновлення, або видалення даних з таблиці. Команда

BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE визначає виконання тіла тригера відповідно до внесення, оновлення, або видалення даних з таблиці.

ON *назва_таблиці*

Таблиця, або віртуальна таблиця (VIEW), для якої створюється даний тригер. При видаленні таблиці з бази даних, автоматично видаляються всі пов'язані з нею тригери.

FOR EACH ROW *тіло_тригера*

Задає набір SQL директив, які виконує тригер. Тригер викликається і виконується для кожного зміненого рядка. Директиви можуть об'єднуватись командами BEGIN ... END та містити спеціальні команди OLD та NEW для доступу до попереднього та нового значення поля у зміненому рядку відповідно. В тілі тригера дозволено викликати збережені процедури, але заборонено використовувати транзакції, оскільки тіло тригера автоматично виконується як одна транзакція.

NEW.*назва_поля*

Повертає нове значення поля для зміненого рядка. Працює лише при подіях INSERT та UPDATE. У тригерах, які виконуються перед (BEFORE) подією можна змінити нове значення поля командою SET NEW.*назва_поля* = *значення*.

OLD.*назва_поля*

Повертає старе значення поля для зміненого рядка. Можна використовувати лише при подіях UPDATE та DELETE. Змінити старе значення поля не можливо.

Щоб видалити створений тригер з бази даних, потрібно виконати команду **DROP TRIGGER** *назва_тригера*.

Хід роботи

Потрібно розробити тригери, які виконуватимуть наступні дії.

1. Каскадне оновлення таблиці auto при видаленні типу з таблиці type.
2. Шифрування паролю користувача під час внесення в таблицю.
3. Тригер для таблиці Session, який буде фіксувати у таблиці Author дату останнього входу користувача в систему.

1. Перевіримо записи таблиці auto:

```
SELECT * FROM mydb.auto  
LIMIT 3;
```

	id	volume_engine	volume_power	transmission	type_id	model_id
▶	1	1.6	140	Auto	2	1
	2	1.8	160	Mechanic	3	2
	3	2	180	Variator	5	3
*	NULL	NULL	NULL	NULL	NULL	NULL

Тепер видалимо 2 і 3 тип, і подивимось, що станеться з записами в auto:

```
DELETE FROM mydb.type  
WHERE id IN (2, 3);
```

Результат:

❌ 71 17:59:48 DELETE FROM mydb.typ... Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails

Не встановлено, що слід робити при видаленні поля цього ключа.

Для цього, створимо тригер, який встановлюватиме значення type_id за замовчуванням 1.

Код створення тригера:

```
CREATE TRIGGER type_delete  
BEFORE DELETE ON mydb.type  
FOR EACH ROW  
UPDATE mydb.auto SET type_id=1 WHERE type_id=OLD.id;
```

Тепер спробуємо видалити деякі типи:

```
DELETE FROM mydb.type  
WHERE id IN (2, 3);
```

Результат:

✅	73	18:06:09	CREATE TRIGGER type_...	0 row(s) affected
✅	74	18:06:41	DELETE FROM mydb.typ...	2 row(s) affected

```
SELECT * FROM mydb.auto  
LIMIT 3;
```

	id	volume_engine	volume_power	transmission	type_id	model_id
▶	1	1.6	140	Auto	1	1
	2	1.8	160	Mechanic	1	2
	3	2	180	Variator	5	3
*	NULL	NULL	NULL	NULL	NULL	NULL

type_id змінився на значення за замовчуванням 1.

2. Шифрування паролю користувача під час внесення в таблицю.

Код створення тригера:

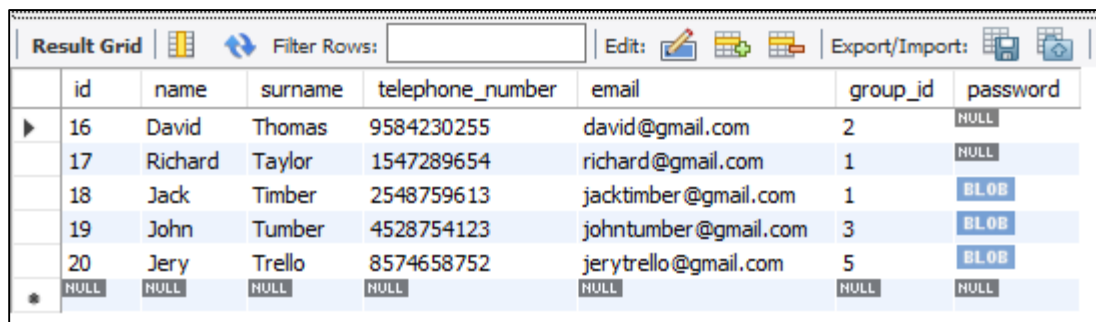
```
CREATE TRIGGER user_password
BEFORE INSERT ON mydb.user
FOR EACH ROW
SET NEW.password = AES_ENCRYPT(NEW.password, 'key-key');
```

Тепер спробуємо додати нових користувачів:

```
INSERT INTO mydb.user VALUES
(18, 'Jack', 'Timber', '2548759613', 'jacktimber@gmail.com', 1, 'mypassword1'),
(19, 'John', 'Tumber', '4528754123', 'johntumber@gmail.com', 3, 'simplepass'),
(20, 'Jery', 'Trello', '8574658752', 'jerytrello@gmail.com', 5, '1111');
```

```
SELECT * FROM mydb.user
WHERE id > 15;
```

Результат:



The screenshot shows a database result grid with columns: id, name, surname, telephone_number, email, group_id, and password. It displays five rows of user data (ids 16-20) and a summary row at the bottom. The password field for rows 18-20 is marked as BLOB.

	id	name	surname	telephone_number	email	group_id	password
▶	16	David	Thomas	9584230255	david@gmail.com	2	NULL
	17	Richard	Taylor	1547289654	richard@gmail.com	1	NULL
	18	Jack	Timber	2548759613	jacktimber@gmail.com	1	BLOB
	19	John	Tumber	4528754123	johntumber@gmail.com	3	BLOB
	20	Jery	Trello	8574658752	jerytrello@gmail.com	5	BLOB
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. Додамо поле last_activity до таблиці user, і будемо заносити туди ту дату та час, коли користувач опублікував останнє оголошення.

Спершу модифікуємо таблицю user:

```
ALTER TABLE mydb.user
ADD COLUMN last_activity DATETIME DEFAULT NULL;
```

Далі створюємо тригер:

```
CREATE TRIGGER user_last_activity
AFTER INSERT ON mydb.advertisement
FOR EACH ROW
UPDATE mydb.user SET user.last_activity=(NEW.pubdate)
WHERE user.id=NEW.user_id;
```

Тепер протестуємо роботу тригера:

INSERT INTO mydb.advertisement VALUES

(21, 10000, 70000, 'blue', 'good', '2020-03-26 21:21:25', '67512', 2019, 'Text', 2, 1),
(22, 6000, 120000, 'green', 'bad', '2020-04-27 21:21:25', '67512', 2012, 'Text', 4, 2);

Результат:

✓	96	18:42:43	ALTER TABLE mydb.user...	0 row(s) affected	Records: 0	Duplicates: 0	Warnings: 0
✓	97	18:43:16	INSERT INTO mydb.adve...	2 row(s) affected	Records: 2	Duplicates: 0	Warnings: 0

Таблиця user:

	id	name	surname	telephone_number	email	group_id	password	last_activity
▶	1	Oliver	Smith	0987654321	oliver@gmail.com	1	NULL	2020-03-26 21:21:25
	2	Jack	Johnson	0987468465	jack@gmail.com	2	NULL	2020-04-27 21:21:25
	3	Harry	Williams	0987654323	harry@gmail.com	3	NULL	NULL

Отже, як бачимо, при додаванні користувачем оголошення, дата та час подання оголошення записуватиметься в таблицю user в поле last_activity автоматично за допомогою тригера.

Висновок: на цій лабораторній роботі було розглянуто тригери, їх призначення, створення та використання.