# **Лекція 12. Реалізація реляційної алгебри засобами оператора SELECT**

# **(Реляційна повнота SQL)**

Для того, щоб продемонструвати, що мова SQL є реляційно повною, потрібно показати, що кожний реляційний оператор може бути виражений засобами SQL. Насправді достатньо показати, що засобами SQL можна виразити будь-який з *примітивних* реляційних операторів.

# **Оператор декартового добутку**

Реляційна алгебра: 

Оператор SQL:

SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, … FROM A, B;

або

SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, … FROM A CROSS JOIN B;

# **Оператор проекції**

Реляційна алгебра: 

Оператор SQL:

SELECT DISTINCT X, Y, …, Z FROM A;

# **Оператор вибірки**

Реляційна алгебра: ,

Оператор SQL:

SELECT \*

FROM A WHERE c;

# **Оператор об’єднання**

Реляційна алгебра: 

Оператор SQL:

SELECT \*

FROM A

UNION

SELECT \*

FROM B;

# **Оператор віднімання**

Реляційна алгебра: 

Оператор SQL:

SELECT \*

FROM A EXCEPT SELECT \*

FROM B

Реляційний оператор перейменування RENAME виражається за допомогою ключового слова AS у списку відібраних полів оператора SELECT. Таким чином, мова SQL є реляційно повною.

Інші оператори реляційної алгебри (з’єднання, перетину, ділення) виражаються через примітивні, і, як наслідок, можуть бути вираженими за допомогою операторів SQL. Наведено їх.

# **Оператор з’єднання**

Реляційна алгебра: 

Оператор SQL:

SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, … FROM A, B

WHERE c;

або

SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, … FROM A CROSS JOIN B

WHERE c;

# **Оператор перетину**

Реляційна алгебра: 

Оператор SQL:

SELECT \*

FROM A

INTERSECT

SELECT \*

FROM B;

# **Оператор ділення**

Реляційна алгебра: 

Оператор SQL:

SELECT DISTINCT A.X FROM A

WHERE NOT EXIST (SELECT \*

FROM B

WHERE NOT EXIST (SELECT \*

FROM A A1 WHERE

A1.X = A.X AND

A1.Y = B.Y));

Зауваження. Оператор SQL, який реалізує ділення відношень важко запам’ятати, тому дамо приклад еквівалентного перетворення виразів, які представляють сутність запиту.

Нехай відношення A містить дані про поставки деталей, відношення B містить список усіх деталей, які можуть поставлятися. Атрибут X є номером постачальника, атрибут Y є номером деталі.

Розділити відношення A на відношення B означає в даному прикладі "відібрати номера постачальників, які поставляють *усі* деталі".

Перетворимо текст виразу:

Запит "Відібрати номери постачальників, які поставляють *усі* деталі" еквівалентний такому

"Відібрати ті номери постачальників з таблиці A, для яких *не існує* деталей у таблиці B, які не поставляються", що еквівалентно

"Відібрати ті номери постачальників з таблиці A, для яких *не існує* тих номерів деталей з таблиці B, які *не поставляються* цим постачальником", що еквівалентно

"Відібрати ті номери постачальників з таблиці A, для яких *не існує* тих номерів деталей з таблиці B, для яких *не існує* записів про поставки у таблиці A для цього постачальника та цієї деталі".

Останній вираз записується мовою SQL. При переводі виразу на мову SQL потрібно врахувати, що у внутрішньому підзапиті таблиця A має бути перейменована, для того, щоб відрізняти її від екземпляра цієї ж таблиці, яка використовується у зовнішньому запиті.

Практично кожна база даних, за виключенням найпростіших, містить деяку кількість програмного коду у вигляді тригерів та збережених процедур.

***Збережені процедури*** - це процедури та функції, що зберігаються безпосередньо в базі даних у відкомпильованому вигляді та які можуть запускатися користувачами або додатками, що працюють з базою даних. Збережені процедури зазвичай пишутися, як праило, за допомогою спеціального процедурного розширення мови SQL. Основне призначення збережених процедур - реалізація бізнес-процесів предметної області.

***Тригери*** - це збережені процедури, що зв’язані з деякими подіями, які відбуваються під час роботи бази даних. В якості таких подій виступають операції вставки, оновлення та видалення рядків таблиць. Якщо у базі даних визначено деякий тригер, то він запускається *автоматично* завжди при виникненні події, з якою цей тригер зв’язаний. Важливим є те, що користувач не може обійти тригер. Тригер спрацьовує незалежно від того, хто з користувачів і в який спосіб ініціював подію, що викликала запуск тригера. Таким чином, основне призначення тригерів - автоматична підтримка цілісності бази даних. Тригери можуть бути як достатньо простими, наприклад, такими, що підтримують цілісність посилань, так і доволі складними, що реалізують деякі складні обмеження предметної області або складні дії, які мають відбутися при настанні деяких подій. Наприклад, з операциєю вставки нового товару в накладну може буть зв’язаний тригер, який виконує такі дії - перевіряє, чи є необходна кількість товару, при наявності товару додає його у накладну та зменшує дані про наявгість товару на складі, при відсутностіі товару формує замовлення на постачання недоствтньої кількості товару та надсилає замовлення за електронною поштою постачальнику. Очевидно, що чим більше програмного коду у вигляді тригерів і збережених процедур міститься у базі даних, тим складніше її розробка і супроводження.

**Синтаксис создания триггера:**

CREATE TRIGGER trigger\_name trigger\_time trigger\_event  
ON tbl\_name FOR EACH ROW trigger\_stmt

trigger\_name — назва тригеру  
trigger\_time — момент спрацьовування тригеру. BEFORE — перед подією. AFTER — після події.

trigger\_event — подія:

insert — подія, яка пов’язана з операторами insert, data load, replace  
update — подія, яка пов’язана з оператором update  
delete — подія, яка пов’язана з операторами delete, replace. Оператори DROP TABLE та TRUNCATE не активують виконання тригеру  
tbl\_name — назва таблиці  
trigger\_stmt - вираз, який виконується при активації тригеру

# **Застосування**

# ***Приклад1.*** Приклад тригеру, який підсумовує значення, які вставляються в один з стовбців таблиці:

**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;**

> SET @sum = 0;

> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);

> SELECT @sum AS 'Total amount inserted';

# ***Приклад2.***

# **таблиця, за якою будемо спостерігати**

# **CREATE TABLE `test` (`id` INT( 11 ) UNSIGNED NOT NULL AUTO\_INCREMENT**

# **PRIMARY KEY, `content` TEXT NOT NULL) ENGINE = MYISAM;**

# **таблиця журналу**

# **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;**

# **—триггер DELIMITER //**

# **CREATE TRIGGER `update\_test` AFTER INSERT ON `test`**

# **FOR EACH ROW BEGIN**

# **INSERT INTO log Set msg = 'insert', row\_id = NEW.id;**

# **END;**

# Якщо тепер додати запис у таблицю test, то у таблиці log теж з’явиться запис, а у полі row\_id зберігається id рядка, який вставляється у таблицю test.

# ***Приклад3.***

# **— видаляємо триггер**

# **DROP TRIGGER `update\_test`;**

# **— створимо ще одну таблицю, в якій будуть зберігатися резервні копії рядків з таблиці test**

# **CREATE TABLE `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**

# **DELIMITER //**

# **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 відредагувати або видалити рядок з test, він буде скопійований у *backup*.

У тригерах використовуються псевдоніми OLD та NEW. OLD.*col\_name* звертається до стовбця існуючого рядка перед тим, як його буде модифіковано або видалено. NEW.*col\_name* звертається до стовпця нового рядка, який буде вставлений, або ж до існуючого рядка після того, як він модифікується.

***Приклад збереженої процедури***

**CREATE PROCEDURE p ()**

**BEGIN**

**DECLARE i INT DEFAULT 0;**

**WHILE i < 5 DO**

**SELECT \* FROM v;**

**SET i = i + 1;**

**END WHILE;**

**END;**

**DELIMITER ;**

# **>CALL p()**

Використовуючи конструкцію BEGIN ... END, можна визначати тригер, який виконує багато інструкцій. Всередині блока BEGIN можна також застосувати інший синтаксис, який дозволяється всередині збережених процедур, типу умовних виразів і циклів. Однак, для того, щоб визначити тригер, необхідно перевизначити операторний роздільник **mysql** так, щоб використовувати ; як операторний роздільник всередині опису тригера. Наступний приклад ілюструє ці моменти. У прикладі визначено тригер UPDATE, який перевіряє нове значення, яке необхідно використати для модифікації кожного рядка, і змінює значення, щоб залишатися в діапазоні від 0 до 100. Цей тригер має бути тригером типу BEFORE, тому що значення повинно бути перевірене перед тим, як воно буде використано, щоб модифікувати рядок:

**> delimiter|**

**> 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;**

**> delimiter ;**

Можливо простіше визначити збережену процедуру і потім викликати її з тригера простою інструкцією CALL. Це вигідно, якщо потрібно викликати одну процедуру всередині декількох тригерів.

# **Висновки**

Фактично стандартом доступу до реляційних баз даних в даний час є мова SQL (Structured Query Language).

Мова SQL оперує термінами, які дещо відрізняються від термінів реляційної теорії, наприклад, замість "відношень" застосовуютьмя "таблиці", замість "кортежів" - "рядки", замість "атрибутів" - "колонки" або "стовбці".

Стандарт мови SQL, хоча й базується на реляційній теорії, але у багатьох моментах відходить від неї. Основу мови SQL складають оператори, умовно розбиті на декілька груп за функціями, що вони виконують:

* + - Оператори DDL (Data Definition Language) - оператори визначення об’єктів бази даних.
    - Оператори DML (Data Manipulation Language) - оператори маніпулювання даними.
    - Оператори захисту та управління даними, і т.д.