

Триггеры

Триггеры. Отличие от других хранимых процедур

- Вызываются событием, нельзя вызвать вручную
- Нельзя вызвать из внешнего интерфейса (клиентского приложения)
- Не имеют параметров
- Не могут быть функциями, т.е. возвращать значения

Ссылочная целостность

- Декларативная (create)
- Активная (триггеры)

Назначение триггеров

- Поддержание ссылочной целостности
- Передача пользователю предупреждения об ошибках или сообщений о данных
- Отладка (т.е. отслеживание ссылок на указанные переменные и/или контроль над изменениями состояния этих переменных).
- Аудит (например, регистрация информации о том, кто и когда внес те или иные изменения в определенные переменные отношения).
- Измерение производительности (например, регистрация времени наступления или трассировка указанных событий в базе данных).
- Проведение компенсирующих действий (например, каскадная организация удаления кортежа поставщика для удаления также соответствующих кортежей поставок).
- Логическое удаление

Логическое и физическое удаление

Физическое (жесткое) удаление	Логическое (мягкое) удаление
Строка удаляется из БД полностью, занятый участок памяти освобождается и становится доступным для дальнейшего использования.	Строка сохраняется в БД, но в служебной части она помечается как удаленная.
Удаленное данные не доступно	сохраняется история (удобно для аудита) и зависимые данные
Данные занимают меньше памяти	Объем памяти, занимаемый данными постоянно растет
Запросы проще	В запросах условия на актуальность
Возможность поддержания ссылочной целостности декларативно	Ссылочная целостность только активная

Из чего состоит триггер

- **событие** — операция в базе, вызвавшая триггер
- **Время** вызова триггера, относительно операции
- **условие** — это логическое выражение, которое должно принимать значение TRUE для того, чтобы было выполнен триггер
- **действие** — тело триггерной процедуры

Триггеры по времени действия

- До

- Для каскадного удаления
- Обработки ошибок
- Сохранение старых значений
- Отладка
- Шифрование данных

- Вместо

- Для каскадного удаления
- Обработки ошибок
- Сохранение старых значений
- Отладка
- Шифрование данных

- После

- Логирование изменений
- Проведение компенсирующих действий (Удаление с очисткой справочника, расчет вычисляемого поля)

Типы триггеров по способу обработки команд

- FOR EACH ROW
Для каждой обработанной строки
- FOR EACH STATEMENT
Для каждой обработанной команды

Как в триггере узнать старые и НОВЫЕ данные?

MySQL| Postgres

- OLD
- NEW

MS SQL server (transact SQL)

- DELETED
- INSERTED

	id_st	surname	name	patronym	id_gr
▶	3	Сидоров	Сидор	Сидорович	3

	id_st	surname	name	patronym	id_gr
▶	1	Иванов	Иван	Иванович	1
	2	Петров	Петр	Петрович	2
	3	Сидоров	Сидор	Сидорович	3
	4	Кузнецов	Кузьма	Кузьмич	1
	5	Иванова	Ирина	Игоревна	2

Что может использоваться в триггерах

- Команды по манипулированию и определению данных
- Процедурные расширения SQL
- Работа с транзакциями
- Сигналы (для сообщения об ошибках)

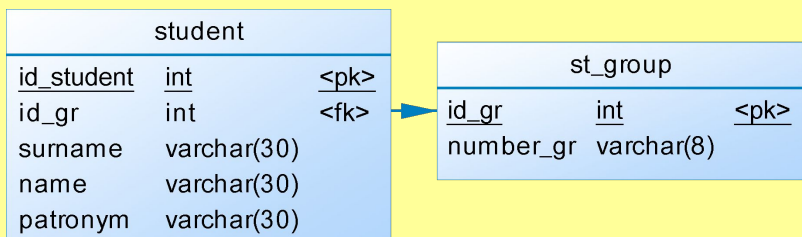
Транзакции

- *Транзакция* — это логическая единица работы; она начинается с выполнения операции BEGIN TRANSACTION и заканчивается операцией COMMIT (выполнение всех действий транзакции) или ROLLBACK(откат всех действий транзакции).

Создание

My SQL	MS SQL Server
CREATE [DEFINER = <i>user</i>] TRIGGER <i>trigger_name</i> <i>trigger_time</i> <i>trigger_event</i> ON <i>tbl_name</i> FOR EACH ROW [<i>trigger_order</i>] <i>trigger_body</i>	CREATE [OR ALTER] TRIGGER <i>trigger_name</i> ON { <i>table</i> <i>view</i> } [WITH <dml_trigger_option> [,...n]] <i>trigger_time</i> <i>trigger_event</i> [NOT FOR REPLICATION] AS <i>trigger_body</i>
<i>trigger_time</i> : { BEFORE AFTER }	<i>trigger_time</i> : { FOR AFTER INSTEAD OF }
<i>trigger_event</i> : { INSERT UPDATE DELETE }	<i>trigger_event</i> : { [INSERT] [,] [UPDATE] [,] [DELETE] }
<i>trigger_order</i> : { FOLLOWS PRECEDES } <i>other_trigger_name</i>	
<i>trigger_body</i> : BEGIN ... END	<i>trigger_body</i> : AS ... GO
OLD , NEW	DELETED , INSERTED

Пример реализации каскадного удаления



My SQL

BEFORE

```
delimiter //
Create trigger my_trigger
before delete on st_group FOR EACH
ROW
Begin
delete from student where id_gr
=OLD.id_gr;
End//
delimiter ;
```

MS SQL Server

INSTEAD OF

```
Create trigger my_trigger
instead of delete on st_group
as
begin
delete * from student where id_gr in
(select id_gr from deleted)
delete * from st_group where id_gr in
(select id_gr from deleted)
end
go
```

Пример реализации подсчёта

student		
<u>id_student</u>	int	<pk>
id_gr	int	<fk>
surname	varchar(30)	
name	varchar(30)	
patronym	varchar(30)	

st_group		
<u>id_gr</u>	int	<pk>
number_gr	varchar(8)	
stud_count	tinyint	

My SQL

```
delimiter //
Create trigger my_trigger
after update on student
FOR EACH ROW
Begin
Update st_group
Set stud_count=stud_count-1
where id_gr =OLD.id_gr;
Update st_group
Set stud_count=stud_count+1
where id_gr =NEW.id_gr;
End//
delimiter ;
```

MS SQL Server

```
Create trigger my_trigger
after update on student
as
begin
update st_group
Set stud_count=stud_count+ q.cnt
from (select count(id_student) as cnt,id_gr from
inserted group by id_gr)q inner join st_group on
st_group.id_gr=q.id_gr;
update st_group
Set stud_count=stud_count- q.cnt
from (select count(id_student) as cnt,id_gr from
deleted group by id_gr)q inner join st_group on
st_group.id_gr=q.id_gr;
end
go
```

Транзакции в триггерах

student		
<u>id_student</u>	int	<pk>
id_gr	int	<fk>
surname	varchar(30)	
name	varchar(30)	
patronym	varchar(30)	

st_group		
<u>id_gr</u>	int	<pk>
number_gr	varchar(8)	
stud_count	tinyint	



My SQL

Нельзя начинать
откатывать или завершать
транзакции в триггерах
MySQL

MS SQL Server

```
Create trigger my_trigger
after insert on student
as
begin
If exists (select * from inserted where surname ="
) rollback transaction;
end
go
```

Создание Postgres

My SQL	Postgres
CREATE [DEFINER = <i>user</i>] TRIGGER <i>trigger_name</i> <i>trigger_time</i> <i>trigger_event</i> ON <i>tbl_name</i> FOR EACH ROW [<i>trigger_order</i>] <i>trigger_body</i>	CREATE [OR REPLACE] [CONSTRAINT] TRIGGER <i>trigger_name</i> <i>trigger_time</i> <i>trigger_event</i> ON <i>table_name</i> [FROM <i>referenced_table_name</i>] [NOT DEFERRABLE [DEFERRABLE] [INITIALLY IMMEDIATE INITIALLY DEFERRED]] [REFERENCING { { OLD NEW } TABLE [AS] <i>transition_relation_name</i> } [...]] [FOR [EACH] { ROW STATEMENT }] [WHEN (<i>condition</i>)] EXECUTE <i>trigger_body</i>
<i>trigger_time</i> : { BEFORE AFTER }	<i>trigger_time</i> : { BEFORE AFTER INSTEAD OF }
<i>trigger_event</i> : { INSERT UPDATE DELETE }	<i>trigger_event</i> : { INSERT DELETE TRUNCATE UPDATE [OF <i>column_name</i> [, ...]] } <i>event</i> [OR ...]
	<i>referenced_table_name</i> Имя другой таблицы, на которую ссылается ограничение. используется для ограничений внешнего ключа и не рекомендуется для обычного применения. допускается только для триггеров ограничений.
<i>trigger_body</i> : BEGIN ... END	<i>trigger_body</i> : { FUNCTION PROCEDURE } <i>function_name</i> (<i>arguments</i>)
OLD . NEW	OLD . NEW

Создание Postgres какие триггеры есть

Когда	Событие	На уровне строк (FOR EACH ROW)	На уровне оператора (FOR EACH STATEMENT)
BEFORE	INSERT/UPDATE/DELETE	Таблицы и сторонние таблицы	Таблицы, представления и сторонние таблицы
	TRUNCATE	—	Таблицы
AFTER	INSERT/UPDATE/DELETE	Таблицы и сторонние таблицы	Таблицы, представления и сторонние таблицы
	TRUNCATE	—	Таблицы
INSTEAD OF	INSERT/UPDATE/DELETE	Представления	—
	TRUNCATE	—	—

Пример реализации подсчёта Postgres

```
CREATE OR REPLACE FUNCTION calc_stud_gr_after() RETURNS trigger
```

```
AS $$
```

```
BEGIN
```

```
update st_group set stud_count = stud_count + 1 where st_group.id_gr = new.id_gr;
```

```
update st_group set stud_count = stud_count - 1 where st_group.id_gr = old.id_gr;
```

```
RETURN NEW;
```

```
END;$$
```

```
LANGUAGE plpgsql;
```

```
CREATE TRIGGER calc_stud_gr_after AFTER UPDATE OF id_gr
```

```
ON student
```

```
FOR EACH ROW EXECUTE PROCEDURE calc_stud_gr_after()
```

Пример реализации проверки Postgres

- CREATE OR REPLACE FUNCTION insert_existing_gr() RETURNS trigger
 - AS \$\$
 - BEGIN
 - IF EXISTS (SELECT * FROM st_group WHERE st_group.num_gr = NEW.num_gr)
 - THEN RAISE EXCEPTION 'Невозможно добавить группу %, так как группа с данным номером уже существует', NEW.num_gr;
 - END IF;
 - RETURN NEW;
 - END;\$\$
 - LANGUAGE plpgsql;
-
- CREATE TRIGGER insert_existing_gr1 BEFORE INSERT ON st_group
 - FOR EACH ROW EXECUTE PROCEDURE insert_existing_gr()

Удаление

- DROP TRIGGER [IF EXISTS]
[schema_name.]trigger_name

Изменение триггера

My SQL	MS SQL Server
<pre>DROP TRIGGER my_trigger CREATE TRIGGER my_trigger ...</pre>	<pre>ALTER TRIGGER schema_name.trigger_name ON (table view) [WITH <dml_trigger_option> [,...n]] (FOR AFTER INSTEAD OF) { [DELETE] [,] [INSERT] [,] [UPDATE] } [NOT FOR REPLICATION] AS { sql_statement [;] [...n] EXTERNAL NAME <method specifier> [;] } <dml_trigger_option> ::= [ENCRYPTION] [<EXECUTE AS Clause>] <method_specifier> ::= assembly_name.class_name.method_name</pre>

Транзакции MySQL

- START TRANSACTION
[*transaction_characteristic* [, *transaction_characteristic*] ...]
- *transaction_characteristic*: { WITH CONSISTENT SNAPSHOT | READ WRITE | READ ONLY }
- BEGIN [WORK]
- COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
- ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
- SET autocommit = {0 | 1}

Сигнал

- СИГНАЛ - это способ «вернуть» ошибку из процедуры.
- SIGNAL предоставляет информацию об ошибке обработчику, внешней части приложения или клиенту. Кроме того, он обеспечивает контроль характеристик ошибки (номер ошибки, значение SQLSTATE, сообщение)

Сигнал синтаксис

- *SIGNAL condition_value*
[SET *signal_information_item* [, *signal_information_item*]
...]
- *condition_value*: { SQLSTATE [VALUE] *sqlstate_value* |
condition_name }
- *signal_information_item*:
condition_information_item_name =
simple_value_specification
- *condition_information_item_name*: { CLASS_ORIGIN |
SUBCLASS_ORIGIN | MESSAGE_TEXT | MYSQL_ERRNO |
CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA |
CONSTRAINT_NAME | CATALOG_NAME | SCHEMA_NAME
| TABLE_NAME | COLUMN_NAME | CURSOR_NAME }

SQLSTATE

- ~~Class = '00' (success)~~

- Class = '01' (warning)

Значение системной переменной `warning_count` увеличивается. `SHOW WARNINGS` показывает сигнал. Обработчики `SQLWARNING` ловят сигнал.

Предупреждения не могут быть возвращены из хранимых функций, потому что оператор `RETURN`, который вызывает возврат функции, очищает область диагностики. оператор `RETURN` очищает все предупреждения, которые могли там присутствовать (и сбрасывает `warning_count` в 0).

- Class = '02' (not found)

Обработчики `NOT FOUND` ловят сигнал. Нет влияния на курсоры. Если сигнал не обрабатывается в хранимой функции, выполнение заканчивается.

- Class = '02' (exception)

Если сигнал не обрабатывается в хранимой функции, выполнение заканчивается.

- Class = '40'

Рассматривается как обычное исключение.

Чтобы указать общее значение `SQLSTATE`, используйте `'45000'`, что означает «необработанное пользовательское исключение».

Сигнал пример

```
CREATE PROCEDURE p (pval INT)
BEGIN
    DECLARE specialty CONDITION FOR SQLSTATE '45000';
    IF pval = 0 THEN
        SIGNAL SQLSTATE '01000';
    ELSEIF pval = 1 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An error occurred';
    ELSEIF pval = 2 THEN
        SIGNAL specialty
        SET MESSAGE_TEXT = 'An error occurred';
    ELSE
        SIGNAL SQLSTATE '01000'
        SET MESSAGE_TEXT = 'A warning occurred', MYSQL_ERRNO = 1000;
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'An error occurred', MYSQL_ERRNO = 1001;
    END IF;
END;
```

Пример триггера с сигналом об ошибке

```
delimiter //
use test//
create table trigger_test
(
id int not null
)//
drop trigger if exists trg_trigger_test_ins //
create trigger trg_trigger_test_ins before insert on trigger_test
for each row
begin
declare msg varchar(255);
if new.id < 0 then
set msg = concat('MyTriggerError: Trying to insert a negative value
in trigger_test: ', cast(new.id as char));
signal sqlstate '45000' set message_text = msg;
end if;
end
//

delimiter ;
```

Вызов триггера

- insert into trigger_test values (2);
- insert into trigger_test values (-1);

Action Output				
	#	Time	Action	Message
✓	1	21:07:27	insert into trigger_test values (2)	1 row(s) affected
✗	2	21:07:29	insert into trigger_test values (-1)	Error Code: 1644. MyTriggerError: Trying to insert a negative value in trigger_test: -1

Курсоры MySQL

- Необязательный результат
- Только чтение
- Только в одном направлении
- **DECLARE** *cursor_name* CURSOR FOR *select_statement*
- **OPEN** *cursor_name*
- **FETCH** [[NEXT] FROM] *cursor_name* INTO *var_name* [, *var_name*] ...
- **CLOSE** *cursor_name*
- Если больше нет строк, возникает условие «Нет данных» со значением SQLSTATE «02000». Чтобы обнаружить это условие, можно настроить обработчик для него (или для условия NOT FOUND)

Курсоры пример

```
CREATE PROCEDURE curdemo()  
BEGIN  
    DECLARE done INT DEFAULT FALSE;  
    DECLARE a CHAR(16);  
    DECLARE b, c INT;  
    DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;  
    DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;  
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  
  
    OPEN cur1;  
    OPEN cur2;  
  
    read_loop: LOOP  
        FETCH cur1 INTO a, b;  
        FETCH cur2 INTO c;  
        IF done THEN  
            LEAVE read_loop;  
        END IF;  
        IF b < c THEN  
            INSERT INTO test.t3 VALUES (a,b);  
        ELSE  
            INSERT INTO test.t3 VALUES (a,c);  
        END IF;  
    END LOOP;  
  
    CLOSE cur1;  
    CLOSE cur2;  
END;
```