triggers для mysql

Что есть триггер?

Триггер - это хранимая процедура особого типа, исполнение которой обусловлено наступлением определенного события (event).

Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики.

Все производимые ими модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Все триггеры в Вашей базе можно рассматривать, как event-manager связанный с событиями изменения данных (или их количества) в одних таблицах и вызывающий изменение данных в других.

Event’s

BEFORE – до начала

AFTER – после выполнения

Эти события можно привязывать к операциям:

- INSERT
- UPDATE
- DELETE

Т.е. вы можете определить действия:

- BEFORE INSERT или AFTER INSERT
- BEFORE UPDATE или AFTER UPDATE
- BEFORE DELETE или AFTER DELETE

Event’s

BEFORE – используется для «перехвата» входящего запроса, и изменения данных в самом запросе или/и упреждающего изменения данных в другой таблице базы.

AFTER – используется только для принятия решения об изменении данных в какой либо из таблиц базы на основании уже «случившегося» события.

Trigger = «слушатель»

Как уже понятно, триггеры привязаны к таблицам, в которых «слушают события».

```
TRIGGER `first_trigger` AFTER INSERT ON `user` FOR EACH ROW
BEGIN
….
    обычный SQL ;)
….
END
```

Триггер с именем first_trigger «слушает» события INSERT в таблице user и, после выполнения операции, для каждой изменённой строки выполняет некое SQL – выражение.

«Болванка» для триггера

Пример запроса на создание триггера на событие BEFORE UPDATE для таблицы user:

```
DELIMITER $$

    DROP TRIGGER first_trigger$$
    CREATE TRIGGER first_trigger
    BEFORE UPDATE ON user FOR EACH ROW
    BEGIN
      ….
          обычный SQL ;)
      ….
    END;$$

DELIMITER ;
```

Боевой пример

Пусть у нас будут 2 таблицы:

User

user_id (идентификатор юзера в проекте)

status (0 – для не подтверждённых юзеров, 1 – для подтверждённых)

user_id - primary key, autoincrement.

- - - - -

Confirm

user_id (идентификатор юзера)

activ_key (идентификатор действия)

user_id - foreign key

Боевой пример

Допустим, речь идёт о стандартной регистрации в проекте.

Юзер регистрируется:

- в таблице user создаётся запись с данными юзера и status=0

INSERT INTO user(status, …) VALUES (0, ….)

В этот момент user получает свой user_id. Пусть для примера, user_id будет 123456.

- в таблице confirm создаётся запись с активационным ключом для подтвреждения персоны

INSERT INTO confirm(user_id, activ_key) VALUES (123456, ‘some_value’)

И нашему юзеру на почту падает письмо со ссылкой типа:

```
<a href=“http://mysite.ru/confirm?user_id=123456&a_key=some_value”>
  Подтвердить регистрацию
</a>
```

Боевой пример

Юзер переходит по ссылке, мы вынуждены сделать запросы:

1) SELECT activ_key FROM confirm WHERE user_id=123456;
*проверка: совпадает ли activ_key в базе и из ссылки
2) UPDATE user SET status=1 WHERE user_id=123456;
* обновление: выставляем статус «подтверждён» для персоны
3) DELETE FROM confirm WHERE user_id=123456;
* удалем запись из базы confirm, т.к. человек себя уже подтвердил
4) SELECT * FROM user WHERE user_id=123456;
* забираем данные персоны. Т.к. после подтверждения, нам нужно отрисовать страницу, где мы поздравляем юзера с успешной регистрацией, и (может) обращаемся по имени и пр.

Конечно, можно сделать и проще, но суть останется прежней.

Боевой пример

```
DELIMITER $$

    DROP TRIGGER confirm$$
    CREATE TRIGGER confirm
    AFTER DELETE ON confirm FOR EACH ROW
    BEGIN
      UPDATE user SET status=1 WHERE user_id=OLD.user_id;
    END;$$

DELIMITER ;
```

Теперь, мы можем поступить так (вместо варианта на предыдущей странице):

1) DELETE FROM confirm WHERE user_id=123456 AND activ_key='some_value';
2) SELECT * FROM user WHERE user_id=123456;

И просто проверить user.status, если он равен 1, то юзер себя уже подтвердил.

OLD и NEW

Как было указано в предыдущем примере, в триггерах используются две парадигмы OLD и NEW.

OLD – значение в таблице до события.

NEW – значение в таблице после события.

Например, для события DELETE существует только OLD.

Для события INSERT только NEW.

А для UPDATE и то и другое.

Всем этим арсеналом пользуются для весьма гибкого переопределения значений.

Т.к. триггеры в «теле» содержат sql, который поддерживает условия IF THEN ELSE (см доки по mysql), то вполне допустимо использование:

```
IF (NEW.status = 1 AND NEW.status != OLD.status)
THEN SET NEW.some_field='some_value';
END IF;
```

Несколько примеров

```
DELIMITER $$
DROP TRIGGER news_counter$$
CREATE TRIGGER news_counter
AFTER INSERT ON `news_comments` FOR EACH ROW
BEGIN
    UPDATE
      news
    SET
      comments_ count = comments_count + 1
    WHERE
      news_id = NEW.news_id
END;$$
DELIMITER ;
```

Счётчик кол-ва комментариев хранится в таблице news (с самими новостями), а комментарии - в таблице news_comments. Триггер выполнит инкремент счётчика, при добавлении нового комментария.

NEW.news_id относится к новой записи в таблице news_comments.

Несколько примеров

```
DELIMITER $$
DROP TRIGGER category_flag_changer$$
CREATE TRIGGER category_flag_changer
BEFORE INSERT ON `category_news` FOR EACH ROW
BEGIN
    IF
     NEW.category_id = 12
    THEN
     SET NEW.some_flag = 1;
    ELSE
     SET NEW.some_flag = 0;
    END IF;
END;$$
DELIMITER ;
```

Здесь видно, что для категории 12 выставляется флаг «some_flag» вне зависимости от того значения, что пришло в запросе.

Заключение

Эта презентация не раскрывает всей силы использования триггеров в проекте, но хотя бы избавит от «страха» использовать этот инструмент в работе.

Удачи.