ЛР6. Триггеры. Транзакции

Триггеры

https://dev.mysql.com/doc/refman/8.0/en/triggers.html

Триггер представляет собой именованный объект базы данных, который связан с таблицей. Он будет активирован, когда произойдет определенное событие (`INSERT, DELETE, UPDATE`), связанное с заданной таблицей. Триггер может быть установлен для активации до или после события. Триггеры не могут быть использованы с представлениями.

Синтаксис создания триггера:

```
CREATE
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

Данное выражение создает новый триггер. *`trigger_name`* – название триггера. *`trigger_time`* определяет время выполнения триггера: до или после наступления события *`trigger_event`*. В качестве события могут быть следующие: `INSERT, UPDATE, DELETE`.

Возможно создание нескольких триггеров для заданной таблицы, с одинаковыми значениями *`trigger_time`* и *`trigger_event`*. В данном случае по умолчанию порядок выполнения таких триггеров определяется датой их создания. Можно изменить порядок выполнения таких триггеров, используя выражения `FOLLOWS` и `PRECEDES`. С `FOLLOWS` новый триггер активируется после существующего триггера. С `PRECEDES` новый триггер активируется перед существующим триггером.

`trigger_body` представляет выражение, которое выполняется при активации триггера. Если выражение состоит из нескольких строк (операций), разделенных

«;», то данное выражение заключается внутри конструкции BEGIN…END. Подробное описание синтаксиса операций, заключенных внутри данной конструкции, представлено здесь: https://dev.mysql.com/doc/refman/8.0/en/sql-compound-statements.html.

В теле триггера ключевые слова `OLD` и `NEW` позволяют получить доступ к строкам таблицы, действие над которыми активировали триггер. В случае `INSERT` триггера может использоваться только `NEW.col_name`, так как нет старых строк. В `DELETE` триггере может использоваться только `OLD.col_name`, так как нет новых строк. В `UPDATE` триггере можно использовать `OLD.col_name` для обращения к столбцам строки перед ее обновлением и `NEW.col_name` для обращения к столбцам строки после ее обновления.

```
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
               SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'error';
           END IF;
       END;//
```

При написании кода триггеров следует использовать оператор DELIMITER, который заменяет стандартный разделитель «;» на указанный пользователем. Это нужно для того, чтобы сервер MySQL воспринимал объявление функции, процедуры или триггера как единый запрос.

Пример:

```
delimiter //
    CREATE TRIGGER upd_check BEFORE UPDATE ON account
    FOR EACH ROW
    BEGIN
      …
    END;//
delimiter ;
```

Транзакции

https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-transactions.html

Транзакция — это операция, состоящая из одного или нескольких запросов к базе данных. Суть транзакций — обеспечить корректное выполнение всех запросов в рамках одной транзакции, а также обеспечить механизм изоляции транзакций друг от друга для решения проблемы совместного доступа к данным. Любая транзакция либо выполняется полностью, либо не выполняется вообще.

В транзакционной модели есть две базовых операции: `COMMIT` и `ROLLBACK`. `COMMIT` выполняет фиксацию всех изменений в транзакции. `ROLLBACK` отменяет (откатывает) изменения, произошедшие в транзакции.

При старте транзакции все последующие изменения сохраняются во временном хранилище. В случае выполнения `COMMIT`, все изменения, выполненные в рамках одной транзакции, сохранятся в физическую БД. В случае выполнения `ROLLBACK` произойдет откат и все изменения, выполненные в рамках этой транзакции, не сохранятся.

По умолчанию MySQL работает в режиме автоматического завершения транзакций, т. е. как только выполняется оператор обновления данных, который модифицирует таблицу, изменения тут же сохраняются на диске. Чтобы объединить операторы в транзакцию, следует отключить этот режим: `set AUTOCOMMIT=0`. Также отключить режим автоматического завершения транзакций для отдельной последовательности операторов можно оператором `START TRANSACTION`.

Для некоторых операторов нельзя выполнить откат с помощью `ROLLBACK`. Это операторы языка определения данных (Data Definition Language). Сюда входят запросы `CREATE, ALTER, DROP, TRUNCATE, COMMENT, RENAME`.

Следующие операторы неявно завершают транзакцию (как если бы перед их выполнением был выдан `COMMIT`): `ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, ALTER VIEW, CREATE DATABASE, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, CREATE TRIGGER, CREATE VIEW, DROP DATABASE, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, DROP TRIGGER, DROP VIEW, RENAME TABLE, TRUNCATE TABLE, BEGIN, SET autocommit = 1, START TRANSACTION`.

Транзакция может быть разделена на точки сохранения. Оператор `SAVEPOINT identifier_name` устанавливает именованную точку сохранения транзакции с именем идентификатора. Если текущая транзакция имеет точку

сохранения с тем же именем, старая точка сохранения удаляется, а новая устанавливается. Оператор `ROLLBACK TO SAVEPOINT identifier_name` откатывает транзакцию до указанной точки сохранения без прерывания транзакции. Изменения, которые выполняются в текущей транзакции для строк после установки точки сохранения, отменяются при откате. Для удаления одной или нескольких точек сохранения используется команда `RELEASE SAVEPOINT identifier_name`.

Уровни изоляций транзакций с разной степенью обеспечивают целостность данных при их одновременной обработке множеством процессов (пользователей). В MySQL существует 4 уровня изоляции транзакций:

- `READ UNCOMMITTED` — самый низкий уровень изоляции. При этом уровне возможно чтение незафиксированных изменений параллельных транзакций.
- `READ COMMITTED` — на этом уровне возможно чтение данных только зафиксированных транзакций. Однако, возможна запись в уже прочитанные внутри транзакции данные.
- `REPEATABLE READ` — на этом уровне изоляции так же возможно чтение данных только зафиксированных транзакций. Так же на этом уровне отсутствует проблема «Неповторяемого чтения», то есть строки, которые участвуют в выборке в рамках транзакции, блокируются и не могут быть *изменены* другими параллельными транзакциями. Но таблицы целиком не блокируются и возможно фантомное чтение — одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. Другая транзакция в интервалах между этими выборками *добавляет* или *удаляет* строки, используемые в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества строк. Однако в MySQL проблема фантомного чтения решена на данном уровне изоляции: все чтения данных в пределе одной транзакции используют «снимок» данных, полученный при первом чтении внутри этой транзакции.
- `SERIALIZABLE` — максимальный уровень изоляции, гарантирует неизменяемость данных другими процессами до завершения транзакции. Но в то же время является самым медленным. В MySQL этот уровень изоляции схож с `REPEATABLE READ`, однако все простые операторы `SELECT` неявно преобразуются к виду `SELECT ... FOR SHARE`, если режим автоматического завершения транзакций выключен.

По умолчанию в MySQL установлен уровень изоляции `REPEATABLE READ`. Для смены уровня изоляции используется оператор `SET TRANSACTION`. Этот оператор устанавливает уровень изоляции следующей транзакции, глобально либо только для текущего сеанса.

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL

{READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
SERIALIZABLE }
```

ЛР6. Задание

Триггеры

1. Создать триггер, заполняющий одно из полей таблицы на основе вводимых данных (например, поле «Возраст» по вводимым данным поля «Дата рождения»).
2. Создать два триггера на одно событие: первый осуществляет проверку целостности добавляемых данных в таблицу, второй запрещает добавление данных при заданных условиях.
3. Создать триггеры, осуществляющие протоколирование операций `INSERT`, `UPDATE`, `DELETE` для заданной таблицы. Для этого создать новую таблицу, содержащую следующие поля: пользователь, операция, время, данные до выполнения операции, новые данные.

Транзакции

4. Написать процедуру, которая добавляет связанные данные в несколько таблиц. В том случае, если вставка данных в одну из таблиц по какой-либо причине невозможна, выполняется откат внесенных процедурой изменений.
5. Написать процедуру, которая добавляет в таблицу несколько строк данных. При каждом добавлении строки выполняется проверка выполнимости некоторого условия на агрегированных данных одного из столбцов данной таблицы (например, сумма всех значений по заданному столбцу не превышает заданного значения). При невыполнении данного условия выполняется откат добавления такой строки (используя оператор `ROLLBACK TO SAVEPOINT`).
6. Продемонстрировать возможность чтения незафиксированных изменений при использовании уровня изоляции `READ UNCOMMITTED` и отсутствие такой возможности при уровне изоляции `READ COMMITTED`.
7. Продемонстрировать возможность записи в уже прочитанные данные при использовании уровня изоляции `READ COMMITTED` и отсутствие такой возможности при уровне изоляции `REPEATABLE READ`.