CS 235: Introduction to Databases

Svetlozar Nestorov

Lecture Notes #14

Triggers (MySQL Version)

```
CREATE TRIGGER <trigger name>
    {BEFORE | AFTER}
    {INSERT | UPDATE | DELETE}
    ON <table name>
    FOR EACH ROW
    <SQL statements>
```

Example

- Whenever we insert a new tuple into Sells, make sure the beer mentioned is also mentioned in Beers, and insert it (with a null manufacturer) if not.

```
CREATE TRIGGER BeerTrig
AFTER INSERT ON Sells
FOR EACH ROW
    BEGIN
        INSERT IGNORE INTO Beers(name)
        VALUES(new.beer);
    END;
```

Options

- AFTER triggers cannot change the value of the inserted/updated tuple.
- BEFORE triggers can change the value of the inserted/updated tuple.

More Options

- INSERT can be DELETE or UPDATE
- FOR EACH ROW can be omitted, with an important effect: the action is done once for the relation(s) consisting of all changes.
  - MySQL recognized only “FOR EACH ROW”

Explanation

- There are two special (transition) variables *new* and *old*, representing the new and old tuple in the change.
  - old makes no sense in an insert, and new makes no sense in a delete.

More Explanations

- The action is any statement allowed in a MySQL function
  - Simplest form: surround one or more SQL statements with BEGIN and END.
  - However, select-from-where has a limited form.
- Need to (temporarily) redefine default delimiter (;) to another character, e.g. ($)
- MySQL triggers are part of the database schema, like tables or views.

Even More Explanations

- Important MySQL constraint: the action cannot change the relation that triggers the action.
- MySQL returns an error only at run time.

Example

- Maintain a list of all the bars that raise their price for some beer by more than $1. *RipoffBars(bar)*

```
DELIMITER //

CREATE TRIGGER PriceTrig
AFTER UPDATE ON Sells
FOR EACH ROW
    BEGIN
        IF (NEW.price > OLD.price + 1) THEN
            INSERT INTO RipoffBars VALUES(NEW.bar);
        END IF;
    END; //

DELIMITER ;
```

Attribute Checks with Triggers

- Create two triggers BEFORE INSERT and BEFORE UPDATE
  - What about BEFORE DELETE?
- The triggers check attribute constraint and if not satisfied make a modification that will be rejected, so the triggering INSERT or UPDATE will fail.

Example

```
CREATE TABLE Sells (
    bar CHAR(20) NOT NULL,
    beer CHAR(20),
    price REAL;
);
```

- Check that the price is not more than $12.

Example

```
CREATE TRIGGER PriceInsTrig
BEFORE INSERT ON Sells
FOR EACH ROW
  BEGIN
      IF (NEW.price > 12) THEN
          SET NEW.bar = NULL;
      END IF;
  END; //
```

Example

```
CREATE TRIGGER PriceUpdTrig
BEFORE UPDATE ON Sells
FOR EACH ROW
   BEGIN
        IF (NEW.price > 12) THEN
             SET NEW.bar = NULL;
        END IF;
   END; //
```

SQL Triggers

- Covered in the book.
- Some differences, including:
  1. The MySQL restriction about not modifying the relation of the trigger or other relations linked to it by constraints is not present in SQL.
  2. The action in SQL is a list of (restricted) SQL statements.

DB Application Programming

- Application is written in general-purpose programming language: C, C++, Java…
  - Not in SQL!
- Application-driven database queries.
  - E.g., user registers, sends a message.
- Impedance mismatch:
  - Sets (relations) are first class objects in DBMS, but not in C, Java…
  - Vice versa for pointers, conditional statements.

Interface Solutions

1. Extend SQL with general-purpose programming: PSM.
2. Execute DB queries within application code: embedded SQL.
3. Call function from DB library: call-level interface (CLI), ODBC, JDBC.

Persistent Stored Modules

- Stored procedures as DB elements.
- Combine general-purpose programming with SQL.
- Extends functionality of DBMS.

Basic PSM Form: Procedures

```
CREATE PROCEDURE <name> (
      <parameters>)
  <declarations>
  <body>;
```

Basic PSM Form: Functions

CREATE FUNCTION <name> (
<parameters>) RETURNS <type>
<declarations>
<body>;

Parameters in PSM

- For each parameter:
  - Mode: IN, OUT, INOUT
  - Name: as usual
  - Type: as usual
- Examples:
  IN newprice NUMBER
  OUT oldprice NUMBER
  INOUT drinker VARCHAR[30]

Example

- A procedure to add a beer and price to Spoon's menu:

```
CREATE PROCEDURE spoonMenu(
    IN beer VARCHAR[30],
    IN price NUMBER
)
INSERT INTO Sells
  VALUES('Spoon', beer, price);
```

Invoking Procedures

- Using SQL/PSM command CALL
  CALL spoonMenu('BudHeavy', '7.50')
- Functions can be used in SQL expressions, provided that the return type is appropriate.

PSM Statements

- DECLARE <name> <type>;
- SET <variable> = <expression>
- BEGIN <statements> END
- RETURN <expression>
  - Does not terminate execution!

IF Statements

- Simplest form:
  IF <condition> THEN
    <statements> END IF;
- With ELSE:
  IF…THEN…ELSE…END IF;
- Nested:
  IF…THEN…ELSEIF…ELSEIF…ELSE…END IF;

Loops

- Basic form:
  LOOP <statements> END LOOP;
- Exiting loops:
  <loop name>: LOOP
  …LEAVE <loop name>…
  END LOOP;
- Other forms:
  WHILE <cond> DO <stmts> END WHILE;
  REPEAT <stmts> UNTIL <cond> END REPEAT;