

CSPP 53001: Databases

Svetlozar Nestorov

Overview

- MySQL triggers
- DB application programming
- MySQL routines

MySQL Triggers

```
CREATE TRIGGER name
{ BEFORE | AFTER }
{ INSERT | UPDATE | DELETE }
ON rel
FOR EACH ROW
sql_statements
```

Example Trigger

Whenever we insert a new tuple into Sells, make sure the beer mentioned appears 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;
```

Timing Options

- AFTER triggers cannot change the value of the inserted or updated tuple.
- BEFORE triggers are allowed to change the value of the inserted or updated tuple.

Event Options

- Specify the event that activates the trigger: INSERT, DELETE, UPDATE of a relation
- In principle, the trigger can act after every tuple change or after all tuple changes (e.g. INSERT subquery)
- FOR EACH ROW means act for every tuple (only option in MySQL).

Special Variables

- Two special variables are available inside the trigger: **old** and **new**, representing the old and new values of the tuple that triggered the action.
- For inserts, only **new** is available.
- For deletes, only **old** is available.

Trigger Actions

- The action can be one or more MySQL statements surrounded by **BEGIN** and **END** with some restrictions.
- In order to simplify parsing of the trigger the default delimiter (“;”) needs to be redefined temporarily to another character (e.g. “|”)

Action Restrictions

- The actions cannot change the relation that triggers the action except for the tuple that triggered the action.
However, other relations can be modified.
- Restriction is in place to avoid infinite loop bugs.
- MySQL returns an error only at run time!!

Ripoff Bars

- Maintain a relation of all bars that raise the price of any 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**.
Do we need **BEFORE DELETE**?
- The triggers check the attribute constraints and if not satisfied make a modification of the tuple that will be rejected, so the triggering **INSERT** or **UPDATE** fails.

Price Check

- Check that price of beer is less than \$20!

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

Price Check Trigger 1

```
CREATE TRIGGER PriceCheckInsTrig
BEFORE INSERT ON Sells
FOR EACH ROW
BEGIN
  IF (NEW.price >= 20) THEN
    SET NEW.bar = NULL;
  END IF;
END;
```

Price Check Trigger 2

```
CREATE TRIGGER PriceCheckUpdTrig
BEFORE UPDATE ON Sells
FOR EACH ROW
BEGIN
  IF (NEW.price >= 20) THEN
    SET NEW.bar = NULL;
  END IF;
END;
```

DB Application Programming

- Application is written in general purpose language: C, Java, php (but not SQL!)
- DB queries are application driven:
user register, buys product, etc.
- Impedance mismatch:
Sets (relations) are first class objects in DBMS but not in application language.
Vice versa for pointers, data structures.

Interface Solutions

- Extend SQL with general-purpose programming
Persistent Stored Modules (PSM), called routines in MySQL
- Execute DB queries within application code: embedded SQL
- Call functions from DB library: call-level interface (CLI), ODBC, JDBC.

MySQL Routines

- MySQL's version of PSM
Stored procedures
Functions
- New feature in 5.0+ versions
Adheres to standards (similar to IBM's DB2, different from Oracle PL/SQL).
Bugs are possible (bugs.mysql.com)

Procedures

```
CREATE PROCEDURE name(arglist)
BEGIN
  declarations
  statements
END;
```

Functions

```
CREATE FUNCTION name(arglist)
RETURNS type
BEGIN
  declarations
  statements
END;
```

Arguments

- The argument list has name-mode-type triples.
Modes: IN, OUT, INOUT for read-only (input), write-only (output) and read/write, respectively
Types: standard SQL data types.

Procedure Example

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

```
CREATE addHopleafBeer(
  IN b CHAR(20),
  IN p REAL)
BEGIN
  INSERT INTO Sells
  VALUES('Hopleaf',b,p)
END;

CALL addHopleafBeer('Corona',7.50);
```

Declarations

- Variables
- Conditions
- Cursors
- Handlers
- Must be declared in this particular order!

Conditions

```
DECLARE condName
  CONDITION FOR SQLSTATE errorStr
```

```
DECLARE condName
  CONDITION FOR errorNumber
```

- The following conditions are predefined:
NOT FOUND (no more rows)
SQLEXCEPTION (run-time error)
SQLWARNING (warning)

Handlers

- Declare what to do in case of errors (or conditions)

```
DECLARE {EXIT | CONTINUE}
  HANDLER FOR
  {errorNum | SQLSTATE errorStr | condName}
  SQL statement
```

- Common practice: set a flag for CONTINUE handlers and check inside stored procedures

Body Constructs

- Assignments:
`SET var = expr;`
– variables must be declared
- Branches:
`IF cond THEN`
 `statements`
`ELSE`
 `statements`
`END IF;`

Queries in Routines

- Single-row selects allow retrieval into a variable of the result of a query that is guaranteed to produce one tuple.
- Cursors allow the retrieval of many tuples, with the cursor and a loop used to process each in turn.

Cursors in MySQL

- The cursor declaration is:
`DECLARE curName`
 `CURSOR FOR query;`
- Fetching is done with:
`FETCH curName INTO variables;`

Procedure Example 1/3

- Beer accessibility: make every beer available for \$5 in at least one bar.
`CREATE PROCEDURE BeerAvailability()`
`BEGIN`
 `DECLARE aBeer CHAR(20);`
 `DECLARE aBar CHAR(20);`
 `DECLARE aPrice REAL;`
 `DECLARE flag INT DEFAULT 0;`

Procedure Example 2/3

```
DECLARE priceyBeers CURSOR FOR
SELECT ???
FROM Sells
WHERE ???

DECLARE CONTINUE HANDLER
FOR NOT FOUND
SET flag = 1;
```

Procedure Example 3/3

```
OPEN priceyBeers;

REPEAT
    FETCH priceyBeers INTO aBeer, aBar, aPrice;
    IF aPrice > 5 THEN
        UPDATE Sells SET price = 5
        WHERE bar = aBar AND beer = aBeer;
    END IF;
UNTIL flag = 1;
END REPEAT;
CLOSE priceyBeers;
END;
```