SQL Constraints and Triggers

Dr Paolo Guagliardo

`dbs-lecturer@ed.ac.uk`

THE UNIVERSITY *of* EDINBURGH
informatics

Fall 2018

This page is intentionally left blank

Check constraints (1)

Syntax: **CHECK** (conditional-expression)
Update/insertion is rejected if the condition evaluates to **false**

Example

```
CREATE TABLE Products (
    pcode    INTEGER PRIMARY KEY,
    pname    VARCHAR(10),
    pdesc    VARCHAR(20),
    ptype    VARCHAR(20),
    price    NUMERIC(6,2) CHECK ( price > 0 ),
    CHECK ( ptype IN ('BOOK','MOVIE','MUSIC') )
);
```

Check constraints (2)

Another example

```
CREATE TABLE Invoices (
    invid    INTEGER PRIMARY KEY,
    ordid    INTEGER NOT NULL UNIQUE,
    amount   NUMERIC(8,2) CHECK ( amount > 0 ),
    issued   DATE,
    due      DATE,
    CHECK ( ordid IN SELECT ordid FROM Orders ),
    CHECK ( due >= issued )
);
```

The check on ordid is similar to a foreign key, but not the same

SQL allows queries in **CHECK** (not implemented in PostgreSQL)

Domain constraints (1)

A domain is essentially a data type with optional constraints

Syntax

CREATE DOMAIN *name datatype* [**DEFAULT** *value*] [*constraint*]

where *constraint* is **NOT NULL** | **CHECK** (*expression*)

In **CHECK** expression, **VALUE** refers to the value being tested

Example

```
CREATE DOMAIN posnumber NUMERIC(10,2)
    CHECK ( VALUE > 0 );

CREATE DOMAIN category VARCHAR(20)
    CHECK ( VALUE IN ('BOOK, 'MUSIC', 'MOVIE') );
```

Domain constraints (2)

```
CREATE TABLE Products (
    pcode    INTEGER PRIMARY KEY,
    pname    VARCHAR(10),
    pdesc    VARCHAR(20),
    ptype    category,
    price    posnumber
);

CREATE TABLE Invoices (
    invid    INTEGER PRIMARY KEY,
    ordid    INTEGER NOT NULL UNIQUE,
    amount   posnumber,
    issued   DATE,
    due      DATE,
    CHECK ( ordid IN SELECT ordid FROM Orders ),
    CHECK ( due >= issued )
);
```

Assertions

Essentially a **CHECK** constraint not bound to a specific table

Syntax: **CREATE ASSERTION** *name* **CHECK** (*condition*)

Example

```
CREATE ASSERTION too_many_customers
    CHECK ( ( SELECT COUNT(*)
              FROM   customers ) <= 1000 ) ;
```

- Standard SQL
- Not implemented in any of the currently available DBMSs
- The problem is allowing queries in **CHECK**

Triggers

Specify an action to execute if certain events took place

Event: a change to the database that **activates** the trigger (an insertion, a deletion, or an update)

Condition: a query or test checked when the trigger is activated (for a query: empty is false, non-empty is true)

Action: a procedure executed when the condition is true

- can refer to old/new values of modified tuples
- can examine answers to the condition query
- can execute new queries
- can make changes to the database (both data **and** schema)
- can be executed before/after the event for each row or for each statement

Triggers: Example 1

Suppose we have

Products	:	pcode, pname, price
Orders	:	ordid, odate, ocust, final (bool)
Details	:	ordid, pcode, qty
Prices	:	ordid, pcode, price

Whenever a new detail for an order is inserted
we want to save the price of the corresponding products

Triggers: Example 1

```
CREATE TRIGGER save_price AFTER INSERT ON details
    REFERENCING NEW TABLE AS inserted
    FOR EACH STATEMENT
    WHEN TRUE
    BEGIN
        INSERT INTO prices(ordid,pcode,price)
        SELECT I.ordid, I.pcode, P.price
        FROM   inserted I JOIN products P
               ON I.pcode = P.pcode
    END ;
```

Triggers: Example 2

Suppose we have

Products	:	pcode, pname, price
Orders	:	ordid, odate, ocust, final (bool)
Details	:	ordid, pcode, qty
Prices	:	ordid, pcode, price
Invoices	:	invid (serial), ordid, amount, issued, due

Whenever an order becomes **final**
we want to generate an invoice for it

Triggers: Example 2

```
CREATE TRIGGER invoice_order
  AFTER UPDATE OF final ON orders
  REFERENCING OLD ROW AS oldrow
              NEW ROW AS newrow
  FOR EACH ROW
  WHEN oldrow.final = FALSE AND newrow.final = TRUE
  BEGIN
    INSERT INTO invoices(ordid,amount,issued,due)
    SELECT O.ordid, SUM(D.qty * P.price),
           O.odate, O.odate+7d
    FROM   orders O, details D, prices P
    WHERE  O.ordid = newrow.ordid
      AND  O.ordid = D.ordid
      AND  D.ordid = P.ordid
      AND  D.pcode = P.pcode
  END ;
```

Triggers in real systems

In PostgreSQL (and similarly for other DBMSs):

```
CREATE TRIGGER name
  { BEFORE | AFTER } event ON table_name
  FOR EACH { ROW | STATEMENT }
  WHEN ( condition )
  EXECUTE PROCEDURE function_name ( arguments )
```

where *event* can be one of:

- **INSERT**
- **UPDATE** [**OF** *column* [, ...]]
- **DELETE**

and *condition* cannot contain queries

Triggers for database consistency

Constraints	**Triggers**
Protection against any statement	Activated by specific statement
Defined declaratively	Defined operationally
▶ easier to understand	▶ effect may be obscure
▶ easier to optimize	▶ more flexibility

Other uses of triggers

- Alert users
- Logging events
- Gather statistics
- Replication
- Workflow management
- Business rules enforcement

Caution with triggers

- An event may activate more than one trigger
- Activated triggers are processed in some **arbitrary** order
- Actions can activate other triggers: we get a chain

Recursive trigger

The action directly/indirectly activates the same trigger

$\Longrightarrow$ collections of triggers can have **unpredictable effects**