By Marina Barsky

Advanced features of relational databases

Lecture 11

ASSERTIONS AND TRIGGERS

Preserving data integrity: constraints revisited

- Note that in most DBMS'es (including ORACLE) only simple CHECK conditions are allowed.

- For example:
  - It is not allowed to refer to columns of other tables
  - No queries as check conditions.

Checks with sub-queries (theoretically)

Example

```
CREATE TABLE Sells (
  bar    CHAR(20),
  beer CHAR(20) CONSTRAINT beer_check
         CHECK ( beer IN (SELECT name FROM Beers)),
  price REAL CHECK ( price <= 5.00 )
);
```

Not possible in PostgreSQL, Oracle

Possible: IN ('Blue', 'Bud')

Assertions (theoretically)

- Such checks are called **assertions**
- They state what must be true at all times
- DBMS's do not generally support assertions since it is very hard to implement them efficiently

Oracle solution:
Views WITH CHECK OPTION

Example

```
CREATE VIEW SellsSafe (bar, beer, price) AS
SELECT bar, beer, price
FROM Sells
WHERE beer IN (
  SELECT beer FROM beers
)
WITH CHECK OPTION;
```

Then, we insert into this view as opposed to directly into Sells

(Note that for this example we could get away with using foreign key constraints)

View with check option: example

```
CREATE TABLE HotelStays (
  roomID INTEGER NOT NULL,
  arrival_date DATE NOT NULL,
  departure_date DATE NOT NULL,
  guest_name CHAR(15) NOT NULL,
  PRIMARY KEY (roomID, arrival_date),
  CHECK (departure_date > arrival_date)
);
```

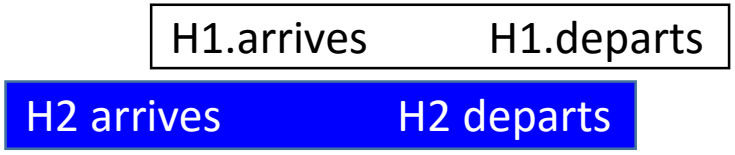
- We want to add the constraint that reservations do not overlap.

Non-overlapping hotel stays

```
CREATE TABLE HotelStays (
  roomID INT NOT NULL,
  arrival_date DATE NOT NULL,
  departure_date DATE NOT NULL,
  guest_name CHAR(15) NOT NULL,
  PRIMARY KEY (room_nbr, arrival_date),
  CHECK (departure_date > arrival_date)
);

CREATE VIEW HotelStaysSafe AS
SELECT roomID, arrival_date, departure_date, guest_name
FROM HotelStays H1
WHERE NOT EXISTS (
  SELECT *
  FROM HotelStays H2
  WHERE H1.roomID = H2.roomID
            AND
            (H2.arrival_date < H1.arrival_date
            AND H1.arrival_date < H2.departure_date)
)
WITH CHECK OPTION;
```

We want to add the constraint that reservations do not overlap.



Hotel Stays – Inserting

```
INSERT INTO HotelStaysSafe (roomID, arrival_date,
   departure_date, guest_name)
VALUES(1, '01-Jan-2009', '03-Jan-2009', 'Alex');
```

This goes Ok.

```
INSERT INTO HotelStays (roomID, arrival_date,
   departure_date, guest_name)
VALUES(1, '02-Jan-2009', '05-Jan-2009', 'Marina');
 *
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Triggers

Triggers: Motivation

- Assertions are powerful, but the DBMS often can't implement them efficiently to be checked at all times
- Attribute- and tuple-based checks are checked at known times, but are less powerful
- Triggers let the user decide when to check for any condition

Event-Condition-Action Rules

Another name for “trigger” is *event-condition-action* (ECA) rule.

- *Event* : typically a type of database modification, e.g., “insert on Sells.”
- *Condition* : Any SQL boolean-valued expression.
- *Action* : Any SQL statements.

Trigger: general syntax

```
Create Trigger name
Before|After|Instead Of events
[ referencing-variables ]
[ For Each Row ]
When ( condition )
action
```

Example 1: AFTER UPDATE trigger

- Using Sells(bar, beer, price) and a unary relation, maintain a list of bars that raise the price of any beer by more than $1.
  - Let the unary relation be RipOffBars(bar)

```
CREATE TABLE Sells(
  beer  VARCHAR(10),
  bar   VARCHAR(13),
  price FLOAT
);


CREATE TABLE RipOffBars(
  bar VARCHAR(13)
);
```

The Trigger

```
CREATE OR REPLACE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  FOR EACH ROW
  WHEN(new.price > old.price + 1.00)
   BEGIN
        INSERT INTO RipoffBars
        VALUES(new.bar);
  END;
```

Remark. This and other trigger examples are in standard SQL syntax which differs from both Oracle and PostgreSQL syntax.

The Trigger

```
CREATE OR REPLACE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  FOR EACH ROW
  WHEN(new.price > old.price + 1.00)
   BEGIN
        INSERT INTO RipoffBars
        VALUES(new.bar);
  END;
```

EVENT: only changes to prices

The Trigger

```
CREATE OR REPLACE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  FOR EACH ROW
  WHEN(new.price > old.price + 1.00)
   BEGIN
        INSERT INTO RipoffBars
        VALUES(new.bar);
  END;
```

We need to consider each price change

The Trigger

```
CREATE OR REPLACE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  FOR EACH ROW
  WHEN(new.price > old.price + 1.00)
   BEGIN
        INSERT INTO RipoffBars
        VALUES(new.bar);
  END;
```

CONDITION:
a raise in price > $1

Updates let us talk about old and new tuples.

The Trigger

```
CREATE OR REPLACE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  FOR EACH ROW
  WHEN(new.price > old.price + 1.00)
   BEGIN
        INSERT INTO RipoffBars
        VALUES(new.bar);
  END;
```

ACTION:
When the price change is big enough, add the bar to RipoffBars

Example 2: BEFORE INSERT trigger

```
CREATE TABLE sales (
   empno INT,
   deptno INT,
   sale FLOAT,
   comm FLOAT
);
```

For employees of department 30, we want to record commission into comm each time a new sale is recorded

The trigger

For employees of department 30, we want to record commission into comm each time a new sale is recorded

```
CREATE TABLE sales (
    empno INT,
    deptno INT,
    sale FLOAT,
    comm FLOAT
);
CREATE OR REPLACE TRIGGER emp_comm_trig
   BEFORE INSERT ON sales
   FOR EACH ROW
BEGIN
   IF NEW.deptno = 30 THEN
      NEW.comm := NEW.sale * .4;
   END IF;
END;
```

Example 3: INSTEAD OF trigger for updateable views

- Remember this updateable view that cannot be updated?

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movies
    WHERE studioName = 'Paramount';
```

The trigger

```
CREATE TRIGGER Paramount_Insert
   INSTEAD OF INSERT ON ParamountMovie

   FOR EACH ROW
   BEGIN
         INSERT INTO Movies (title, year, studioName)
         VALUES(new.title, new.year, 'Paramount');
    END;
```

Pg syntax with example:
https://vibhorkumar.wordpress.com/2011/10/28/instead-of-trigger/

Options: FOR EACH ROW

Two types of triggers:

- *Row level triggers* : execute once for each modified tuple.
- *Statement-level triggers* : execute once for an SQL statement, regardless of how many tuples are modified.

- FOR EACH ROW indicates row-level; its absence indicates statement-level.

Options: The Event

- AFTER can be BEFORE
- UPDATE ON can be DELETE ON or INSERT ON
- And UPDATE ON can be UPDATE …OF… ON mentioning a particular attribute in relation

Row-level triggers

- For an update trigger:
  - The old attribute value can be accessed using old.<column>
  - The new attribute value can be accessed using new.<column>

- For an insert trigger, only new.<column> can be used.
- For a delete trigger only old.<column> can be used.

- In WHEN clause of the trigger we can use old.<column>, new.<column>

Options: The Condition

- Any Boolean-valued condition

- Evaluated on the database as it would exist before or after the triggering event, depending on whether BEFORE or AFTER is used

Options: The Action

- Surround by BEGIN . . . END.

Multiple events

- You may specify up to three triggering events using the keyword OR. Here are some examples:

  ... INSERT ON *R* ...

  ... INSERT **OR** DELETE **OR** UPDATE ON *R* ...

ACTION: restrictions

- Restrictions on <trigger_body> include:
  - You can’t modify the same relation whose modification triggered the trigger.
  - You can’t modify a relation which is the “parent” of the triggering relation in a foreign-key constraint.

PostgreSQL triggers: (non-standard) syntax and examples

https://www.postgresql.org/docs/9.1/static/sql-createtrigger.html

PG defines triggers in 2 steps

- Step 1: write procedure (function) that returns trigger
- Step 2: use this procedure in the action part

- In general, PG trigger syntax is very far from standard

Step 1. Create function

```
CREATE FUNCTION my_trigger_function()
RETURNS trigger
AS
 '
BEGIN
  IF NEW.C1 IS NULL OR NEW.C1 = '''' THEN
    NEW.C1 := ''X'';
  END IF;
  RETURN NEW;
END
'
LANGUAGE 'plpgsql'
```

Step 2. Create trigger

```
CREATE TRIGGER my_trigger
BEFORE INSERT ON T
FOR EACH ROW
EXECUTE PROCEDURE my_trigger_function()
```

View triggers and functions

\dt --- shows tables

\df --- shows functions

- To see all triggers:

select * from pg_trigger;

- Pg_trigger is a normal system table, we can get its fields as

\d+ pg_trigger

select tgname from pg_trigger;

Dropping / disabling

- Dropping functions

DROP FUNCTION <function-name>**()**;

- Dropping Triggers

DROP TRIGGER <trigger_name> **ON** <table-name>;

- Disabling or Enabling Triggers

ALTER TABLE tblname

DISABLE|ENABLE TRIGGER <trigger-name>;

The rest of examples are in TRIGGERS_PG.sql

SQLite triggers (even more non-standard)

- General SQLite syntax: http://sqlite.awardspace.info/syntax/localindex.htm
- Triggers in SQLite: http://linuxgazette.net/109/chirico1.html