DM505 Database Design and Programming
DM576 Database Systems

Panagiotis Tampakis

ptampakis@imada.sdu.dk

Assertions

Assertions

- These are database-schema elements, like relations or views
- Defined by:

```
CREATE ASSERTION <name>
    CHECK (<condition>);
```

- Condition may refer to any relation or attribute in the database schema

Example: Assertion

- In Sells(bar, beer, price), no bar may charge an average of more than 100

```
CREATE ASSERTION NoRipoffBars CHECK
  (
  NOT EXISTS (
      SELECT bar FROM Sells
      GROUP BY bar
      HAVING AVG(price) > 100)
  );
```

Bars with an average price above 100

Example: Assertion

- In Drinkers(name, addr, phone) and Bars(name, addr, license), there cannot be more bars than drinkers

```
CREATE ASSERTION LessBars CHECK (
  (SELECT COUNT(*) FROM Bars) <=
  (SELECT COUNT(*) FROM Drinkers)
);
```

Timing of Assertion Checks

- In principle, we must check every assertion after every modification to any relation of the database
- A clever system can observe that only certain changes could cause a given assertion to be violated
  - Example: No change to Beers can affect the assertion `LessBars` (from last slide)

Exercise

Exercise 7.4.2: Write the following as assertions. The database schema is from the battleships example of Exercise 2.4.3.

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

a) No class may have more than 2 ships.

Triggers

Triggers: Motivation

- Assertions are powerful, but the DBMS often cannot tell when they need to be checked
- Triggers let the user decide when to check for any condition

Event-Condition-Action Rules

- Another name for "trigger" is the *ECA rule*, or *event-condition-action* rule
- *Event:* typically a type of database modification, e.g., "insert on Sells"
- *Condition:* Any SQL boolean-valued expression
- *Action:* Any SQL statements

Preliminary Example: A Trigger

- Instead of using a foreign-key constraint and rejecting insertions into Sells(bar, beer, price) with unknown beers, a trigger can add that beer to Beers, with a NULL manufacturer

Example: Trigger Definition

```
CREATE TRIGGER BeerTrig
  AFTER INSERT ON Sells
  REFERENCING NEW ROW AS NewTuple
  FOR EACH ROW
  WHEN (NewTuple.beer NOT IN
     (SELECT name FROM Beers))
  INSERT INTO Beers(name)
     VALUES(NewTuple.beer);
```

The event

The condition

The action

Options: CREATE TRIGGER

- CREATE TRIGGER <name>
- or CREATE OR REPLACE TRIGGER <name>
  - Useful if there is a trigger with that name and you want to modify the trigger

Options: The Event

- AFTER can be BEFORE
  - Also, INSTEAD OF, if the relation is a view
    - A clever way to execute view modifications: have triggers translate them to appropriate modifications on the base tables
- INSERT can be DELETE or UPDATE
  - And UPDATE can be UPDATE . . . ON a particular attribute

Options: FOR EACH ROW

- Triggers are either "row-level" or "statement-level"
- FOR EACH ROW indicates row-level; its absence indicates statement-level
- *Row level triggers:* execute once for each modified tuple
- *Statement-level triggers:* execute once for a SQL statement, regardless of how many tuples are modified

Row vs Statement Level Triggers

- When to Use Row-Level Triggers
  - Auditing Individual Row Changes
  - Enforcing Data Validation & Business Rules
  - Automatically Updating Related Tables
  - Generating Derived or Default Values

Feature	ROW-Level Trigger	Statement-Level Trigger
Execution Count	Once per row	Once per statement
Granularity	Row-based	Statement-based
`OLD` & `NEW` Values	Available	Not available
Best Use Cases	Auditing, validation, data integrity	Bulk operations, logging schema changes

Row vs Statement Level Triggers

- When to Use Statement-Level Triggers
  - Bulk Operations:
    - Auditing or Restricting Bulk Operations
    - Enforcing Business Rules for Batch Operations
  - Performance Considerations: When you don't need a trigger to run for each row, improving efficiency.
  - DDL Monitoring: When tracking schema modifications.
  - Aggregation Updates: When recalculating summary tables after batch operations.

Options: REFERENCING

- INSERT statements imply a new tuple (for row-level) or new table (for statement-level)
  - The “table” is the set of inserted tuples
- DELETE implies an old tuple or table
- UPDATE implies both
- Refer to these by

[NEW OLD][ROW TABLE] AS <name>

Options: The Condition

- Any boolean-valued condition
- Evaluated on the database as if it would exist before or after the triggering event, depending on whether BEFORE or AFTER is used
  - But always before the changes take effect
- Access the new/old tuple/table through the names in the REFERENCING clause

Options: The Action

- There can be more than one SQL statement in the action
  - Surround by BEGIN . . . END if there is more than one
- But queries make no sense in an action, so we are really limited to modifications

Another Example

- Using Sells(bar, beer, price) and a unary relation RipoffBars(bar), maintain a list of bars that raise the price of any beer by more than 10

The Trigger

```
CREATE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  REFERENCING
     OLD ROW AS ooo
     NEW ROW AS nnn
  FOR EACH ROW
  WHEN (nnn.price > ooo.price + 10)
  INSERT INTO RipoffBars
     VALUES (nnn.bar);
```

The event – only changes to prices

Updates let us talk about old and new tuples

Condition: a raise in price > 10

We need to consider each price change

When the price change is great enough, add the bar to RipoffBars

Exercise

Exercise 7.5.2: Write the following as triggers. In each case, disallow or undo the modification if it does not satisfy the stated constraint. The database schema is from the "PC" example of Exercise 2.4.1:

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

a) When updating the price of a PC, check that there is no lower priced PC with the same speed.

SQL vs PostgreSQL

Checks in PostgreSQL

- Tuple-based checks may only refer to attributes of that relation
- Attribute-based checks may only refer to the name of the attribute
- *No subqueries allowed!*
- Use triggers for more elaborate checks

Assertions in PostgreSQL

- *Assertions are not implemented!*
- Use attribute-based or tuple-based checks where possible
- Use triggers for more elaborate checks

Triggers in PostgreSQL

- PostgreSQL does not allow events for only certain columns
- Rows and tables are called OLD and NEW (no REFERENCING ... AS)
- PostgreSQL only allows to execute a *function* as the action statement

The Trigger – SQL

```
CREATE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  REFERENCING
      OLD ROW AS ooo
      NEW ROW AS nnn
  FOR EACH ROW
  WHEN (nnn.price > ooo.price + 10)
  INSERT INTO RipoffBars
      VALUES (nnn.bar);
```

The event – only changes to prices

Updates let us talk about old and new tuples

We need to consider each price change

Condition: a raise in price > 10

When the price change is great enough, add the bar to RipoffBars

The Trigger – PostgreSQL

```
CREATE TRIGGER PriceTrigger
  AFTER UPDATE ON Sells

  FOR EACH ROW

  EXECUTE PROCEDURE
   checkRipoff();
```

The event – any changes to Sells

Updates have fixed references OLD and NEW

Conditions moved into function

We need to consider each price change

Always check for a ripoff using a function

The Function – PostgreSQL

```
CREATE FUNCTION CheckRipoff()
  RETURNS TRIGGER AS $$BEGIN
    IF NEW.price > OLD.price+10 THEN
      INSERT INTO RipoffBars
      VALUES (NEW.bar);
    END IF;
    RETURN NEW;
  END$$ LANGUAGE plpgsql;
```

Conditions moved into function

When the price change is great enough, add the bar to RipoffBars

Updates have fixed references OLD and NEW

Functions in PostgreSQL

- CREATE FUNCTION name([arguments]) RETURNS [TRIGGER type] AS $$function definition$$ LANGUAGE lang;
- Example:

```
CREATE FUNCTION add(int,int)
RETURNS int AS $$select $1+$2;$$
LANGUAGE SQL;
```

- `CREATE FUNCTION add(i1 int,i2 int)`

```
RETURNS int AS $$BEGIN RETURN
i1 + i2; END;$$ LANGUAGE plpgsql;
```

Example: Attribute-Based Check

```
CREATE TABLE Sells (
  bar    CHAR(20),
  beer   CHAR(20)    CHECK (beer IN
         (SELECT name FROM Beers)),
  price INT CHECK (price <= 100)
);
```

Example: Attribute-Based Check

```
CREATE TABLE Sells (
  bar    CHAR(20),   beer CHAR(20),
  price  INT CHECK (price <= 100));
CREATE FUNCTION CheckBeerName() RETURNS
  TRIGGER AS $$BEGIN IF NOT NEW.beer IN
  (SELECT name FROM Beers) THEN RAISE
  EXCEPTION 'no such beer in Beers';   END
  IF; RETURN NEW; END$$        LANGUAGE
  plpgsql;
CREATE TRIGGER BeerName AFTER UPDATE OR
  INSERT ON Sells FOR EACH ROW
  EXECUTE PROCEDURE CheckBeerName();
```

Example: Assertion

- In Drinkers(name, addr, phone) and Bars(name, addr, license), there cannot be more bars than drinkers

```
CREATE ASSERTION LessBars CHECK (
  (SELECT COUNT(*) FROM Bars) <=
  (SELECT COUNT(*) FROM Drinkers)
);
```

Example: Assertion

```
CREATE FUNCTION CheckNumbers()
  RETURNS TRIGGER AS $$BEGIN IF
  (SELECT COUNT(*) FROM Bars) >
  (SELECT COUNT(*) FROM Drinkers)
  THEN RAISE EXCEPTION ‘2manybars’;
  END IF; RETURN NEW; END$$
  LANGUAGE plpgsql;
CREATE TRIGGER NumberBars AFTER
  INSERT ON Bars EXECUTE PROCEDURE
  CheckNumbers();
CREATE TRIGGER NumberDrinkers AFTER
  DELETE ON Drinkers EXECUTE PROCEDURE
  CheckNumbers();
```

Summary 10

More things you should know:

- Triggers, Events, Conditions, Actions
- Assertions