Chapter 7 Constraints and Triggers

- Keys and foreign keys
- Constraints on attributes and tuples
- Modification of constraints
- Assertions
- **triggers**

Triggers: Motivation

- **Attribute- and tuple-based checks have limited capabilities.**
- **Assertions are sufficiently general for most constraint applications, but they are hard to implement efficiently.**
  - **The DBMS must have real intelligence to avoid checking assertions that couldn't possibly have been violated.**

Triggers: Solution

- **A trigger allows the user to specify when the check occurs.**
- **Like an assertion, a trigger has a general-purpose condition and also can perform any sequence of SQL database modifications.**

Triggers

Often called event-condition-action rules

- **Event= a class of changes in the DB, e.g.: insert, delete**
- **Condition= a test as in a where-clause for whether or not the trigger applies.**
- **Action=one or more SQL statements**

Triggers

Differ from checks, assertions:

- **Triggers are invoked by certain events specified by the database programmer.**
- **Once awakened, the trigger tests a condition.**
- **Only the condition is satisfied, the actions are performed. The action could be any sequence of database operations.**

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:**

REPLACE TRIGGER <name>

  - **Useful if there is a trigger with that name and you want to modify the trigger.**

Options: The Condition

- **AFTER can be BEFORE.**
  - **Also, INSTEAD OF, if the relation is a view.**
    - **A great 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 are executed once for each modified tuple.**
- **Statement-level triggers execute once for an SQL statement, regardless of how many tuples are modified.**

Options: REFERENCING

- **INSERT statements imply a new tuple (for row-level) or new set of tuples (for statement-level).**
- **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 is appropriate.**
- **It is evaluated before or after the triggering event, depending on whether BEFORE or AFTER is used in the event.**
- **Access the new/old tuple or set of tuples through the names declared in the REFERENCING clause.**

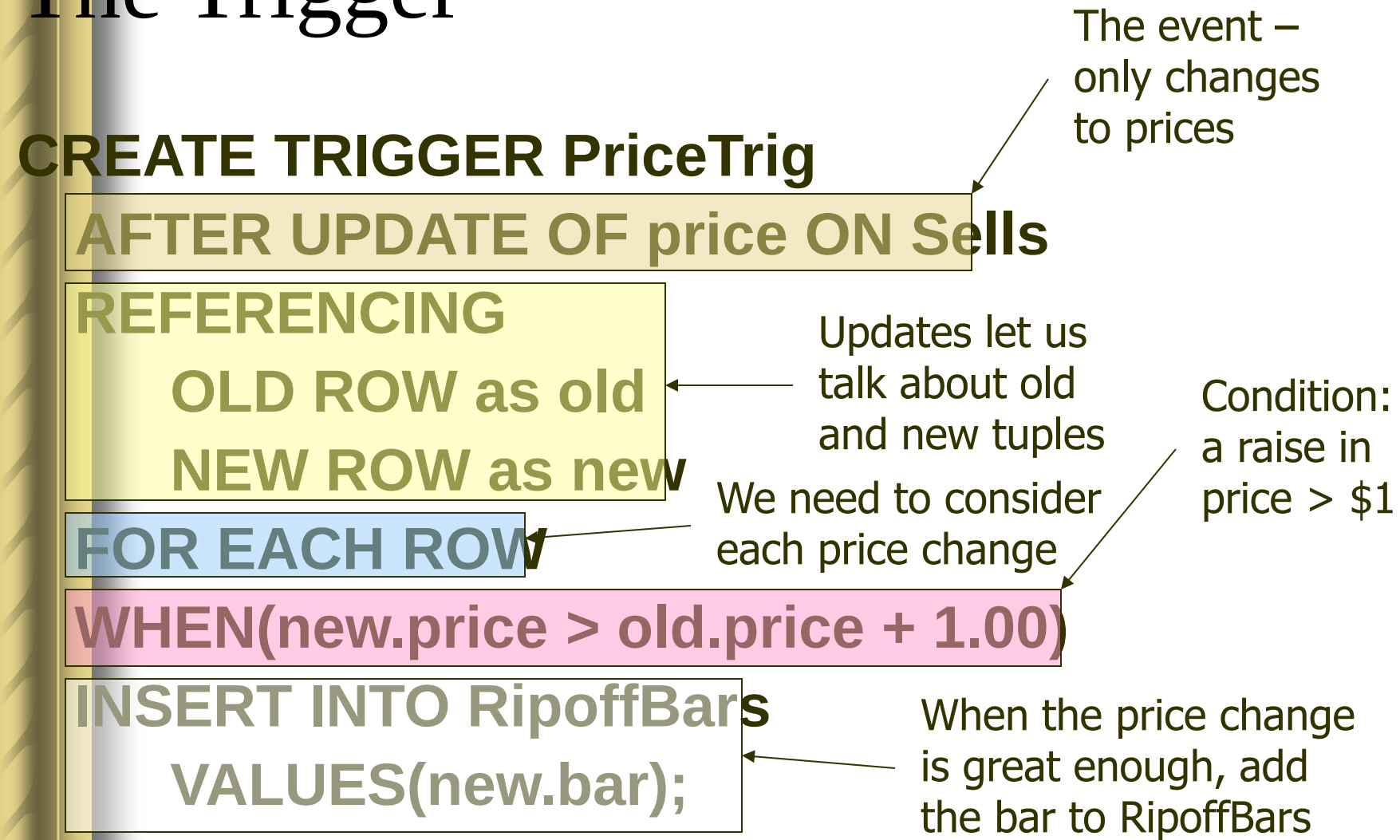
Options: The Action

- **There can be more than one SQL statement in the action.**
  - **Surround by BEGIN . . . END if there is more than one.**
- **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) created for the purpose, maintain a list of bars that raise the price of any beer by more than $1.**

The Trigger


CREATE TRIGGER PriceTrig
AFTER UPDATE OF price ON Sells
REFERENCING
OLD ROW as old
NEW ROW as new
FOR EACH ROW
WHEN(new.price > old.price + 1.00)
INSERT INTO RipoffBars
VALUES(new.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 > $1
When the price change is great enough, add the bar to RipoffBars

Event vs. Triggers

Event will come → wake the trigger

Steps for After trigger:

- **Event happens → test the condition: if true do action otherwise nothing.**

Steps for before trigger:

- **Test the condition: if true do action otherwise nothing → event happens**

Steps for instead of:

- **Test the condition: if true do action otherwise nothing**

Example

- **create table R(x int,y int);**
- **create table S(u int,v int);**
- **insert into R values(1,10);**
- **insert into S values(2,20);**
- **insert into S values(3,30);**

After vs. Before Trigger

```
create trigger beforetrig
before insert on R
for each row
when (3 > (select count(*) from R))
begin
update R set y=y+New.y;
end;

insert into R select * from S;
select * from R;
```

After vs. Before Trigger (cont.)

```
create trigger aftertrig
after insert on R
for each row
when (3 > (select count(*) from R))
begin
update R set y=y+New.y;
end;
insert into R select * from S;
select * from R;
```

Self Triggering

- create table T1(A int);
- pragma recursive_triggers = on;
- create trigger R1

```
after insert on T1
for each row
when (select count(*) from T1) < 10
begin
  insert into T1 values (New.A+1);
end;
insert into T1 values (1);
select * from T1;
```

Row-level Trigger

- Create table T1 (a float);
- create table T2 (a float);

- insert into T1 values (1);
- insert into T1 values (1);
- insert into T1 values (1);
- insert into T1 values (1);

- create trigger R1

```
after insert on T1
for each row
begin
  insert into T2 select avg(A) from T1;
end;
```

- insert into T1 select A+1 from T1;
- select * from T1;
- select * from T2;

Classroom Exercises & Demo

Database schema:

Students(sid,name,dept,age)

Courses(cid,cname,spring,teacher)

SC(sid,cid,semester,cname,grade)

Use triggers to implement Foreign key declaration

CREATE TABLE SC (

sid char(9) REFERENCES students (sid) ON DELETE CASCADE ON UPDATE CASCADE,

…)

Cases to violate:

Delete, update(sid) on students

Insert, update(sid) on sc

Implement: sc(sid) references students(sid)
needs four triggers (R1 ~ R4)
R1: Cascaded delete (students)

```
create trigger R1
after delete on Students
for each row
begin
  delete from sc where sid = Old.sID;
end;
```

R2:Cascaded update when students (sid) update

```
create trigger R2
after update of sid on students
for each row
begin
  update sc
  set sid = new.sid
  where sid = old.sid;
end;
```

R3: insert into sc

```
Create trigger R3
Before insert on sc
For each row
When not exists (select * from students
where sid=new.sid)
Begin
 select raise (rollback, studentNotExists);
End;
```

R4: update sc(sid)

```
create trigger R4
before update of sid on sc
for each row
when not exists (select * from students
where sid=new.sid)
begin
 select raise (rollback,studentNotExists);
end;
```

Test

- **Select * from students;**
- **Insert into sc(sid,cid) values(11,1);**
- **Select * from sc where sid=1;**
- **Delete from students where sid=1;**
- **Select * from sc where sid=1;**

Trigger R5: New cs students will be automatically chosen database courses.

```
create trigger R5
after insert on students
for each row
when new.dept='cs'
begin
  insert into sc(sid,cid,cname) values
(new.sid, 1,'database');
end;
Insert into students(sid,name,dept)
values(11,’wangdong’,’cs’);
Select * from sc;
```

Trigger R6: when the no. of database students is great than 5, it is not allowed.

```
create trigger R6
after insert on sc
for each row
when new.cid=1 and 4 < (select count(*) from sc where cid=1)
begin
  select raise(rollback, greaterThan5);
end;
Insert into sc(sid,cid) values(5,1);
```

Summary

- **Key constraints**
- **Referential Integrity Constraints**
- **Value-based ,Tuple-based Check Constraints**
- **Assertions**
- **Triggers**
- **Invoking time**