CIS 560 - Database System Concepts

Lecture 6

SQL

September 9, 2013

Credits for slides: Suciu, Chang, Ullman.

Copyright: Caragea, 2013

Announcements

- SQL2 assignment posted due Friday
- My office hours:
 - Fridays 10:30 to 11:30am
 - Tuesdays 2:30 to 3:30pm
- GTA's office hours:
 - Wednesdays 9:30 to 10:30am
 - Fridays 12:30 to 1:30pm

Outline

Last time:

- Outer joins (Section 6.3.8)
- Views (Sections 8.1, 8.2, 8.3)
- Constraints (Sections 2.3, 7.1, 7.2)

Today:

• Constraints (Sections 2.3, 7.1, 7.2)

Next:

• E/R Diagrams (Sections 4.1-4.5)

3

Review

- Outer joins how are they useful?
- Views? Types of views? Why use them?
- Constraints versus triggers

Foreign Key Constraints

Referential integrity constraints

CREATE TABLE Purchase (
prodName CHAR(30)
REFERENCES Product(name),
date DATETIME)

prodName is a **foreign key** to Product(name) name must be a **key** in Product

5

Product

NameCategoryGizmogadgetCameraPhotoOneClickPhoto

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Foreign Key Constraints

```
CREATE TABLE Purchase (
    prodName CHAR(30),
    category VARCHAR(20),
    date DATETIME,
    FOREIGN KEY (prodName, category)
    REFERENCES Product(name, category))
```

• (name, category) must be a key in Product

7

What happens during updates?

Types of "problematic" updates:

• In Purchase: insert/update

• In Product: delete/update

Product			Purchase	
<u>Name</u>	Category		ProdName	Store
Gizmo	gadget		Gizmo	Wiz
Camera	Photo		Camera	Ritz
OneClick	Photo		Camera	Wiz
		•	•	

•

Insert/Update in Purchase

An insert or update to Purchase that introduces a nonexistent product must be rejected.

9

Delete/Update in Product

A deletion or update to Product that removes a product value found in some tuples of Purchase can be handled in three ways.

- 1. Default: Reject the modification.
- 2. Cascade: Make the same changes in Purchase.
 - Deleted product: delete Purchase tuple.
 - Updated product: change value in Purchase.
- 3. Set NULL: Change the ProdName to NULL.

```
DELETE FROM relation>
DELETE FROM Product
WHERE condition>;
WHERE name = 'Camera';
```

Example: Cascade

- Suppose we delete the Camera tuple from Product.
 - Then delete all tuples from Purchase that have ProdName= 'Camera'.
- Suppose we update the Camera tuple by changing 'Camera' to 'SonyCamera'.
 - Then change all Purchase tuples with ProdName = 'Camera' so that ProdName= 'SonyCamera'.

11

Example: Set NULL

- Suppose we delete the Camera tuple from Products.
 - Change all tuples of Purchase that have
 ProdName= 'Camera' to have ProdName= NULL.
- Suppose we update the Camera tuple by changing 'Camera' to 'SonyCamera'.
 - Same change to Purchase.

Choosing a Policy

- When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates.
- Follow the foreign-key declaration by:
 ON [UPDATE|DELETE][SET NULL|CASCADE]
- Two such clauses may be used.
- Otherwise, the default (reject) is used.

13

Example

```
create table Purchase (
prodName CHAR(30),
category VARCHAR(20),
store CHAR(30),
FOREIGN KEY (prodName)
REFERENCES Product(name)
ON DELETE SET NULL
ON UPDATE CASCADE)
```

Constraints on Attributes and Tuples

• Constraints on attributes:

NOT NULL -- obvious meaning... CHECK condition -- any condition !

• Constraints on tuples CHECK condition

CHECK constraints not supported in MySQL – silently ignored

15

Attribute-Based Checks

- Put a constraint on the value of a particular attribute.
- CHECK(<condition>) must be added to the declaration for the attribute.
- The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery.

What is the difference from Foreign-Key?

```
CREATE TABLE Purchase (
prodName CHAR(30)

CHECK (prodName IN

SELECT Product.name
FROM Product),
date DATETIME NOT NULL)
```

17

Tuple-Based Checks

- CHECK (<condition>) may be added as another element of a schema definition.
- The condition may refer to any attribute of the relation, but any other attributes or relations require a subquery.
- Checked on insert or update only.

Example

```
CREATE TABLE Purchase (
prodName CHAR(30),
category VARCHAR(20),
date DATETIME,
CHECK (Store=' Ritz' OR
date>' 2010-01-01'))
```

19

General Assertions (not supported in MySQL)

- 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: General Assertions

```
CREATE ASSERTION myAssert CHECK
NOT EXISTS(
SELECT Product.name
FROM Product, Purchase
WHERE Product.name = Purchase.prodName
GROUP BY Product.name
HAVING count(*) > 200)
```

21

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.

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.

23

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.

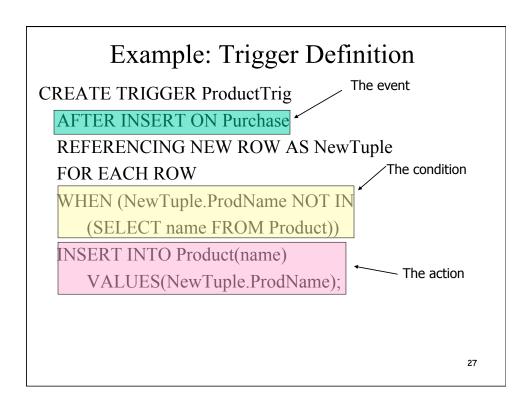
Event-Condition-Action Rules

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

25

Example: A Trigger

 Instead of using a foreign-key constraint and rejecting insertions into
 Purchase(ProdName, Store)
 with unknown products, a trigger can add that product to Product, with a NULL category.



Options: The Event

- 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.

29

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][TUPLE|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 (or fixed by "OLD", "NEW" in MySQL.)

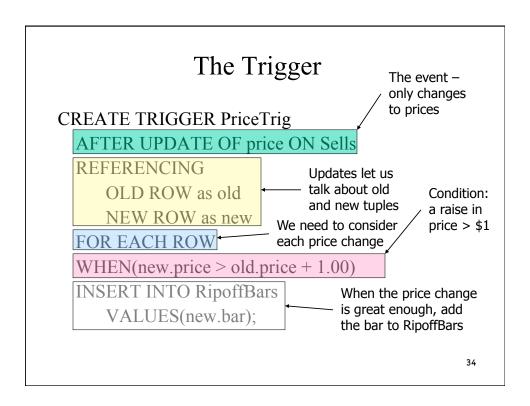
31

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.

Trigger Exercise

• Using Sells(bar, beer, price) and a unary relation RipoffBars(bar), use this unary relation to keep track of bars that raise the price of any beer by more than \$1.



Triggers on Views

- Generally, it is impossible to modify a view, because it doesn't exist physically.
- But an INSTEAD OF trigger lets us interpret view modifications in a way that makes sense.
- Example: We'll design a view Synergy that has (drinker, beer, bar) triples such that the bar serves the beer, the drinker frequents the bar and likes the beer.

35

Example: The View

CREATE VIEW Synergy AS

Pick one copy of each attribute

SELECT Likes.drinker, Likes.beer, Sells.bar

FROM Likes, Sells, Frequents

WHERE Likes.drinker = Frequents.drinker

AND Likes.beer = Sells.beer

AND Sells.bar = Frequents.bar;

Natural join of Likes, Sells, and Frequents

Interpreting a View Insertion

- We cannot insert into Synergy --- it is a view.
- But we can use an INSTEAD OF trigger to turn a (drinker, beer, bar) triple into three insertions of projected pairs, one for each of Likes, Sells, and Frequents.
 - The Sells.price will have to be NULL.

37

The Trigger

CREATE TRIGGER ViewTrig
INSTEAD OF INSERT ON Synergy
REFERENCING NEW ROW AS n
FOR EACH ROW
BEGIN

INSERT INTO LIKES VALUES(n.drinker, n.beer); INSERT INTO SELLS(bar, beer) VALUES(n.bar, n.beer); INSERT INTO FREQUENTS VALUES(n.drinker, n.bar); END;

TRIGGERS in MySQL

```
DROP TRIGGER IF EXISTS TriggerName;
DELIMITER $
PUT TRIGGER CODE HERE...
DELIMITER;
```

The usual delimiter for MySQL triggers is "; " but we will be using them in the trigger body. So it is recommended to set the delimiter to a different symbol before we start the trigger and then set it back to ";" after we are done with the trigger.

39

40

Example

```
DROP TRIGGER IF EXISTS SetPending;
DELIMITER $
CREATE TRIGGER SetPending
BEFORE INSERT ON genre
FOR EACH ROW
 BEGIN
   IF NEW.genre IS NULL THEN
     SET NEW.genre='pending';
   END IF;
END;
DELIMITER;
```

Steps in Building a DB Application

Suppose you are working on your CIS560 project

- · Step 0: pick an application domain
- · Step 1: conceptual design
 - agree on the structure of the database before deciding on a particular implementation
 - discuss with your team mates what to model in the application domain (what entities, relationships between entities, constrains in your domain)
 - need a modeling language to express what you want
 - ER model is the most popular such language
 - output: an ER diagram of the app. domain

Steps in Building a DB Application

- Step 2: pick a type of DBMS
 - relational DBMS is most popular and is our focus
- Step 3: translate ER design to a relational schema
 - use a set of rules to translate from ER to relational schema
 - use a set of schema refinement rules to transform the above relational schema into a good relational schema
- · At this point
 - you have a good relational schema on paper

Steps in Building a DB Application

- Subsequent steps include
 - implement your relational DBMS using a "database programming language" called SQL
 - ordinary users cannot interact with the database directly
 - and the database also cannot do everything you want
 - hence write your application program in C++, Java, Perl, etc to handle the interaction and take care of things that the database cannot do
- So, the first thing we should start with is to learn ER model ...