Introduction to Database Systems
CSE 414

Lecture 16: Constraints
Announcements

• Reminders:
  – Web quiz due Monday night
  – XML homework due Wednesday night

• Today: Constraints (7.1, 7.2, 7.4)
Where We Are?

We are learning about database design
• How to design a database schema?
• Last time: Real world -> E/R Diagrams -> Relations

Next, we will learn more about **good** schemas
• Today: Constraints and data integrity
• Next time: Schema normalization, then Views
Integrity Constraints Motivation

An integrity constraint is a condition specified on a database schema that restricts the data that can be stored in an instance of the database.

- ICs help prevent entry of incorrect information
- How? DBMS enforces integrity constraints
  - Allows only legal database instances (i.e., those that satisfy all constraints) to exist
  - Ensures that all necessary checks are always performed and avoids duplicating the verification logic in each application
Constraints in E/R Diagrams

Finding constraints is part of the modeling process. Commonly used constraints:

**Keys:** social security number uniquely identifies a person.

**Single-value constraints:** a person can have only one father.

**Referential integrity constraints:** if you work for a company, it must exist in the database.

**Other constraints:** peoples’ ages are between 0 and 150.
No formal way to specify multiple keys in E/R diagrams.
Single Value Constraints

makes

vs.

makes
Referential Integrity Constraints

Each product made by at most one company.
Some products made by no company

Each product made by exactly one company.
Q: What does this mean?
A: A Company entity cannot be connected by relationship to more than 99 Product entities.
Constraints in SQL:

- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

- The more complex the constraint, the harder it is to check and to enforce
Key Constraints

Product(name, category)

CREATE TABLE Product (  
    name CHAR(30) PRIMARY KEY,  
    category VARCHAR(20))

OR:

CREATE TABLE Product (  
    name CHAR(30),  
    category VARCHAR(20),  
    PRIMARY KEY (name))
Keys with Multiple Attributes

Product(name, category, price)

CREATE TABLE Product (  
    name CHAR(30),  
    category VARCHAR(20),  
    price INT,  
    PRIMARY KEY (name, category))

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>10</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>20</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Photo</td>
<td>30</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>40</td>
</tr>
</tbody>
</table>
CREATE TABLE Product (  
productID CHAR(10),  
name CHAR(30),  
category VARCHAR(20),  
price INT,  
PRIMARY KEY (productID),  
UNIQUE (name, category))

There is at most one PRIMARY KEY; there can be many UNIQUE
FOREIGN KEY 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.

**Referential integrity constraints**

May write just Product if name is PK.
Foreign Key Constraints

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

• Example with multi-attribute primary key

```
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
What happens during updates?

Types of updates:
- In Purchase: insert/update
- In Product: delete/update

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Category</td>
</tr>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

CSE 414 - Spring 2015
What happens during updates?

- SQL has three policies for maintaining referential integrity:
- **Reject** violating modifications (default)
- **Cascade**: after delete/update do delete/update
- **Set-null** set foreign-key field to NULL
Maintaining Referential Integrity

CREATE TABLE Purchase (  
prodName CHAR(30),  
category VARCHAR(20),  
date DATETIME,  
FOREIGN KEY (prodName, category)  
REFERENCES Product(name, category)  
ON UPDATE CASCADE  
ON DELETE SET NULL )
Constraints on Attributes and Tuples

• Constraints on attributes:
  NOT NULL -- obvious meaning...
  CHECK condition -- any condition!

• Constraints on tuples
  CHECK condition
Constraints on Attributes and Tuples

```
CREATE TABLE R (
    A int NOT NULL,
    B int CHECK (B > 50 and B < 100),
    C varchar(20),
    D int,
    CHECK (C >= 'd' or D > 0)
)
```
Constraints on Attributes and Tuples

CREATE TABLE Product (  
  productID CHAR(10),  
  name CHAR(30),  
  category VARCHAR(20),  
  price INT CHECK (price > 0),  
  PRIMARY KEY (productID),  
  UNIQUE (name, category))
CREATE TABLE Purchase (prodName CHAR(30) CHECK (prodName IN (SELECT Product.name FROM Product)), date DATETIME NOT NULL)

What does this constraint do?

What is the difference from Foreign-Key?
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 ) )

But most DBMSs do not implement assertions
Because it is hard to support them efficiently
Instead, they provide triggers
Database Triggers

• **Event-Condition-Action** rules
• **Event**
  – Can be insertion, update, or deletion to a relation
• **Condition**
  – Can be expressed on DB state before or after event
• **Action**
  – Perform additional DB modifications
More About Triggers

• **Row-level trigger**
  – Executes once for each modified tuple

• **Statement-level trigger**
  – Executes once for all tuples that are modified in a SQL statement
Database Triggers Example

When Product.price is updated, if it is decreased then set Product.category = ‘On sale’

```
CREATE TRIGGER ProductCategories
AFTER UPDATE OF price ON Product
REFERENCING
  OLD ROW AS OldTuple
  NEW ROW AS NewTuple
FOR EACH ROW
WHEN (OldTuple.price > NewTuple.price)
  UPDATE Product
  SET category = 'On sale'
WHERE productID = OldTuple.productID
```
CREATE TRIGGER ProductCategory ON Product AFTER UPDATE AS BEGIN UPDATE Product SET category='sale' WHERE productID IN (SELECT i.productID from inserted i, deleted d WHERE i.productID = d.productID AND i.price < d.price) END
Discussion

• Both constraints and triggers are tools that help us keep the database consistent
• What are their pros and cons?