CS411
Database Systems

06: SQL

Kazuhiro Minami

Constraints & Triggers

Foreign Keys
Local and Global Constraints
Triggers

Integrity Constraints in SQL

- New information added to a database could be wrong in a variety of ways
  - Typographical or transcription errors in manually entered data
- Difficult to write application programs to check the integrity (correctness) of data on every insertion, deletion, and update command.
- SQL provides a variety of techniques for expressing integrity constraints as part of the database schema

Constraints and Triggers

- A *constraint* is a relationship among data elements that the DBMS is required to enforce.
  - Example: key constraints.
- *Triggers* are only executed when a specified condition occurs, e.g., insertion of a tuple.
  - Easier to implement than many constraints.

Kinds of Constraints

- Keys
- Foreign-key, or referential-integrity
- Value-based constraints
  - Constrain values of a particular attribute
- Tuple-based constraints
  - Relationship among different attribute values
- Assertions: any SQL boolean expression

Foreign Keys

Sells(bar, beer, price)

- We might expect that a beer value is a real beer --- something appearing in Beers.name .
- A constraint that requires a beer in Sells to be a beer in Beers is called a *foreign-key* (referential integrity) constraint.

Foreign-key Constraints Corresponds to Referential Integrity Constraints in E/R modeling

Example

Sells

bar	beer	price
Blind pig	Super Dry	$3
Blind pig	Samuel Adams	$4

Beers

name	manf
Super Dry	Asahi

Violation of the foreign-key constraint

Expressing Foreign Keys

- Use the keyword REFERENCES, either:
  1. Within the declaration of an attribute, when only one attribute is involved.

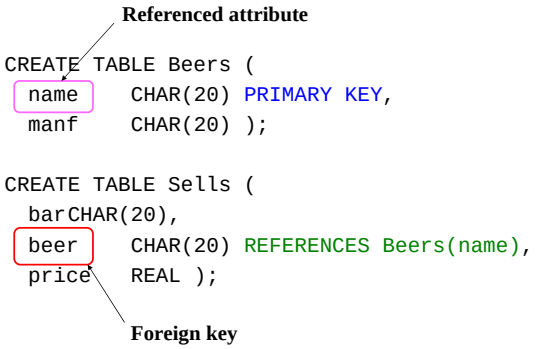
     REFERENCES <relation> (<attributes>)

  1. As an element of the schema, as:

     FOREIGN KEY (<list of attributes>)

     REFERENCES <relation> (<attributes>)
- Referenced attributes must be declared PRIMARY KEY or UNIQUE

Example: With Attribute



Example: As Element

```
CREATE TABLE Beers (
  name   CHAR(20) PRIMARY KEY,
  manf   CHAR(20) );

CREATE TABLE Sells (
  bar    CHAR(20),
  beer   CHAR(20),
  price  REAL,
  FOREIGN KEY(beer) REFERENCES
     Beers(name));
```

Foreign-key definition

Enforcing Foreign-Key Constraints

If there is a foreign-key constraint from attributes of relation *R* to the primary key of relation *S*, two violations are possible:

1. An insert or update to *R* introduces values not found in *S*.
2. A deletion or update to S causes some tuples of *R* to "dangle."

Case 1: Insertion or Update to R

Sells (= R)

bar	beer	price
Blind pig	Super Dry	$3
Blind pig	Samuel Adams	$4

Dangling tuple

Beers (= S)

name	manf
Super Dry	Asahi

Actions Taken -- 1

- An insert or update to Sells that introduces a nonexistent beer must be rejected.

Case 2: Deletion or Update to S

Sells (= R)

bar	beer	price
Blind pig	Super Dry	$3
Blind pig	Samuel Adams	$4

The second tuple in Sells has become dangle.

Beers (= S)

name	manf
Super Dry	Asahi
Samuel Adams	The Boston Beer Company

Actions Taken -- 2

The three possible ways to handle beers that suddenly cease to exist are:

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

Cascade Strategy

- Suppose we delete the Bud tuple from Beers.
  - Then delete all tuples from Sells that have beer = 'Bud'.
- Suppose we update the Bud tuple by changing 'Bud' to 'Budweiser'.
  - Then change all Sells tuples with beer = 'Bud' so that beer = 'Budweiser'.

Example: Cascade

Sells

bar	beer	price
Blind pig	Super Bitter	$3
Blind pig	Samuel Adams	$4

Beers

name	manf
Super Bitter	Asahi
Samuel Adams	The Boston Beer Company

Example: Set NULL

- Suppose we delete the Bud tuple from Beers.
  - Change all tuples of Sells that have beer = 'Bud' to have beer = NULL.
- Suppose we update the Bud tuple by changing 'Bud' to 'Budweiser'.
  - Same change.

Example: Set NULL

Sells

bar	beer	price
Blind pig	NULL	$3
Blind pig	NULL	$4

Beers

name	manf
Super Bitter	Asahi
Samuel Adams	The Boston Beer Company

When you create the table, specify which of the three options you want to use.

```
CREATE TABLE Customers (
        customerName CHAR(20) REFERENCES MasterList(name)
        ON DELETE CASCADE,
        city CHAR(20),
        state CHAR(2),
        zip CHAR (5),
        FOREIGN KEY (city, state, zip)
                REFERENCES GoodAddresses (city, state, zip)
        ON UPDATE CASCADE ON DELETE SET NULL
);
```

Default: reject all UPDATEs to MasterList that violate referential integrity

Attribute-Based Checks:

You can also check an attribute value at INSERT/UPDATE time

Use a subquery if you need to mention other attributes or relations

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

CHECK is never equivalent to a foreign key constraint.

Employees

Employee Name	Department	Hourly Wage
Winslett	Toy	10.00

Departments

Name
Toy
Complaint
Development

Name
Complaint
Development

With a **FOREIGN KEY** *constraint, the change in Departments will be reflected in Employees.*

With **CHECK**, *the change in Departments will not be reflected in Employees.*

Tuple-Based Checks:

You can also check a combination of attribute values at INSERT/UPDATE time

- Only Joe's Bar can sell beer for more than $5:

```
CREATE TABLE Sells (
    bar       CHAR(20),
    beer      CHAR(20),
    priceREAL,
    CHECK (bar = 'Joe''s Bar' OR
                 price <= 5.00)
);
```

For more complex constraints, declare standalone ASSERTIONs.

CREATE ASSERTION *assertionName*
CHECK (*condition*);

No bar can charge more than $5 on average for beer.

```
CREATE ASSERTION NoExpensiveBars CHECK (
        NOT EXISTS (
                SELECT bar
                FROM Sells
                GROUP BY bar
                HAVING 5.00 < AVG(price)
        ));
```

Bars with an average price above $5

There cannot be more bars than drinkers.

Drinkers(name, addr, phone)
Bars(name, addr, license)

```
CREATE ASSERTION FewBars CHECK (
        (SELECT COUNT (*) FROM Bars) <=
        (SELECT COUNT (*) FROM DRINKERS)
);
```

In theory, every ASSERTION is checked after every INSERT/ DELETE/ UPDATE.

In practice, the DBMS only has to check sometimes:

- Adding a drinker can't violate FewBars.
- Removing a bar can't violate NoExpensiveBars.
- Lowering a beer price can't violate NoExpensiveBars.

But is the DBMS smart enough to figure this out?

You can help your not-so-smart DBMS by using TRIGGERs instead of ASSERTIONs

A trigger is an ***ECA*** rule:

When ***Event*** occurs
If ***Condition*** doesn't hold
Then do ***Action***

E.g., an INSERT / DELETE / UPDATE to relation *R*

Any SQL Boolean condition

Any SQL statements

You can use triggers to code very complex stuff.

- You can allow your users to update their views --- but you catch their updates and rewrite them to behave the way you want, avoiding view anomalies.
- You can encode new strategies for handling violations of constraints, different from what the DBMS offers.

If someone inserts an unknown beer into Sells(bar, beer, price), add it to Beers with a NULL manufacturer.

```
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

Syntax for naming the trigger

CREATE TRIGGER *name*

CREATE OR REPLACE TRIGGER *name*

Useful when there is a trigger with that name and you want to modify the trigger.

Syntax for describing the condition

Take one element from each of the three columns:

BEFORE AFTER INSTEAD OF	INSERT DELETE UPDATE UPDATE ON *attribute*	ON *relationName*

Only if the relation is a view

You can execute a trigger once per modified tuple, or once per triggering statement

The default

- Statement-level triggers execute once for each SQL statement that triggers them, regardless of how many tuples are modified.
- Row level triggers are executed once for each modified tuple.

Request explicitly by including FOR EACH ROW

Your condition & action can refer to the tuples being inserted/deleted/updated

- INSERT statements imply a new tuple (for row-level) or new set of tuples (for statement-level).
- DELETE implies an old tuple (row-level) or table (statement-level).
- UPDATE implies both.

"No raise over 10%."

Syntax:

REFERENCING [NEW OLD][ROW TABLE] AS *name*

Pick one

Pick one

Any boolean-valued *Condition* is ok in WHEN *Condition*.

Evaluate the condition on the instance *before* the event

Evaluate the condition on the instance *after* the event

BEFORE
AFTER
INSTEAD OF

INSERT
DELETE
UPDATE
UPDATE OF *attribute*

ON *relationName*

No condition

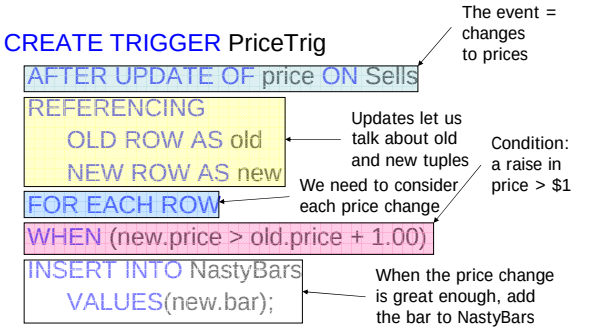
The *Action* is a sequence of SQL statements (modifications).

Surround them by BEGIN . . . END if there is more than one.

But queries make no sense in an action, so we are really limited to modifications.

Remember bars that raise the price of a beer by > $1.

Sells(bar, beer, price) **NastyBars(bar)**



Triggers are great for implementing view updates.

- We cannot insert into Developers --- it is a view.
- But we can use an INSTEAD OF trigger to turn a (name, project) triple into an insertion of a tuple (name, `Development', project) to Employee.

Example: Updating Views

How can I insert a tuple into a table that doesn't exist?

Employee(ssn, name, department, project, salary)

```
CREATE VIEW Developers AS
  SELECT name, project
  FROM Employee
  WHERE department = "Development"
```

If we make the following insertion:

```
INSERT INTO Developers
VALUES("Joe", "Optimizer")
```

This must be "Development"

It becomes:

```
INSERT INTO Employee
VALUES(NULL, "Joe", NULL, "Optimizer", NULL)
```

Allow insertions into Developers

```
CREATE TRIGGER AllowInsert
  INSTEAD OF INSERT ON Developers
  REFERENCING NEW ROW AS new
  FOR EACH ROW
  BEGIN
      INSERT INTO Empolyees(name, department, project)
  VALUES(new.name, `Development', new.project);
  END;
```