Database Constraints AND Triggers

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

Learning Objectives

After this lecture, you should be able to:

- Define Database Constraints, including
  - *Referential integrity constraints*
  - *Attribute-level constraints*
  - *Tuple-level constraints*
  - *Assertions*
- Define database triggers

Outline

- Constraints
  - Foreign-key, or referential-integrity constraints.
  - Value-based constraints.
  - Tuple-based constraints.
  - Assertions.
- Triggers

Constraints

- Constraints are used to make sure that the data in the database "makes sense", that is: important real-world properties are kept valid
  - Via continuous maintenance of "assertions" (i.e. **Constraints**)
- A *constraint* is a relationship among data elements that the DBMS is required to enforce.
  - Example: key constraints.

Kinds of Constraints

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

Foreign Keys

- Consider Relation Sells(cafe, drink, price).
- We might expect that a drink value is a real drink --- something appearing in Drink.name
- A constraint that requires a drink in Sells to be a drink in Drinks is called a *foreign-key* constraint.

Expressing Foreign Keys

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

  FOREIGN KEY (<list of attributes>)

  REFERENCES <relation> (<attributes>)

- Referenced attributes must be declared PRIMARY KEY or UNIQUE. *Why?*

Example: With Attribute

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

CREATE TABLE Sells (
 cafe CHAR(20),
 drink CHAR(20) REFERENCES Drinks(name),
 price REAL );
```

Example: As Element

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

CREATE TABLE Sells (
 cafe  CHAR(20),
 drink CHAR(20),
 price REAL,
 FOREIGN KEY(drink) REFERENCES Drink(name));
```

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."

Why are the other two cases (insert to S and delete of R) not important?

Actions Taken for Changes

- Suppose R = Sells, S = Drinks.
- Referencing relation changes:
  - An insert or update to Sells that introduces a nonexistent drink must be rejected.
- Referenced relation changes:
  - A deletion or update to Drinks that removes a drink value found in some tuples of Sells can be handled in three ways.

Actions Taken for Updates to Beers

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

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

Example: Cascade

- Suppose we delete the Mocha tuple from Drinks.
  - Then delete all tuples from Sells that have drink = 'Mocha'.
- Suppose we update the Mocha tuple by changing 'Mocha' to 'Latte'.
  - Then change all Sells tuples with drink = 'Mocha' so that drink= 'Latte'.

Example: Set NULL

- Suppose we delete the Mocha tuple from Drinks.
  - Change all tuples of Sells that have drink = 'Mocha' to have drink = NULL.
- Suppose we update the Mocha tuple by changing 'Mocha' to 'Latte'.
  - Same change.

Choosing a Policy

- When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates to S (the referenced relation).

- Follow the foreign-key declaration by:

  ON [UPDATE, DELETE][SET NULL, CASCADE]

- Two such clauses may be used.
- Otherwise, the default (reject) is used.

Example

```
CREATE TABLE Sells (
  cafe  CHAR(20),
  drink CHAR(20),
  price REAL,
  FOREIGN KEY(drink)
   REFERENCES Drinks(name)
   ON DELETE SET NULL
   ON UPDATE CASCADE );
```

Outline

- Constraints
  - ✓ Foreign-key, or referential-integrity constraints.
  - Value-based constraints.
  - Tuple-based constraints.
  - Assertions.
- Triggers

Checks

- Attribute-based
  - NOT NULL is one of them …
  - We'll see other more general types of checks
- Tuple-based

Attribute-Based Checks

- Place 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.

Example

```
CREATE TABLE Sells (
  cafe    CHAR(20),
  drink   CHAR(20) CHECK ( drink IN
                    (SELECT name FROM Drinks)),
  price   REAL CHECK ( price <= 5.00 )
);
```

How is Check different from Foreign Key?

```
… drink CHAR(20) CHECK ( drink IN
      (SELECT name FROM Drinks))
```

The drink check seems similar to Foreign Key constraints.

However, the timing of enforcement is different.

Timing of Checks

- An attribute-based check is checked only when a value for that attribute is inserted or updated.
  - Example: CHECK (price <= 5.00) checks every new price and rejects it if it is more than $5.
  - Example: CHECK (drink IN (SELECT name FROM Drinks)) not checked if a drink is deleted from Drinks or updated (unlike foreign-keys).
    - Only checked during inserts/updates **of that attribute**

Outline

- Constraints
  - ✓ Foreign-key, or referential-integrity constraints.
  - ✓ Value-based constraints.
  - Tuple-based constraints.
  - Assertions.
- Triggers

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: Tuple-Based Check

- Only Abdu's Café can sell Mocha for more than $5:

```
CREATE TABLE Sells (
 cafe   CHAR(20),
 drink  CHAR(20),
 price  REAL,
 CHECK (cafe = 'Abdu's Café' OR
       price <= 5.00)
);
```

Q: Why do we need tuple-level check?

- We can do attribute-based check, why tuple level?
- Reason 1: If the check involves more than one attribute of the tuple, we need the tuple-level check.
- Reason 2: Tuple-level constraints are checked more frequently.
  - Whenever there are any inserts or updates to any of the concerned attributes

Outline

- Constraints
  - ✓ Foreign-key, or referential-integrity constraints.
  - ✓ Value-based constraints.
  - ✓ Tuple-based constraints.
  - Assertions.
- Triggers

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.

- Must be true at all times

Example: Assertion

- In Customers(name, addr, phone) and Cafes(name, addr, license), there cannot be more cafes than customers.

```
CREATE ASSERTION FewCafe CHECK (
  (SELECT COUNT(*) FROM Cafes) <=
  (SELECT COUNT(*) FROM Customers)
);
```

Timing of Assertion Checks

- In principle, we must check every assertion after every modification to any relation of the database.

```
CREATE ASSERTION FewCafe CHECK (
  (SELECT COUNT(*) FROM Cafes) <=
  (SELECT COUNT(*) FROM Customers)
);
```

Outline

- Constraints
  - ✓ Foreign-key, or referential-integrity constraints.
  - ✓ Value-based constraints.
  - ✓ Tuple-based 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, but can also 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 Sells."
  - *Condition* : Any SQL Boolean-valued expression.
  - *Action* : Any SQL statements.

Example: A Trigger

- There are many details to learn about triggers.
- Here is an example to set the stage.
  - Recall that with a foreign-key constraint, we ended up rejecting any insertions into Sells(cafe, drink, price) with unknown drinks
  - Here, a trigger can add that drink to Drinks, with a NULL manufacturer.

Example: Trigger Definition

```
CREATE TRIGGER DrinkTrig
   BEFORE INSERT ON Sells
    FOR EACH ROW
   BEGIN
      SET @drink =   (SELECT name FROM Drinks
                          WHERE name = new.drink);
     IF @drink IS NULL THEN
            INSERT INTO Drinks(name)
            VALUES(new.drink);
     END IF;
   END;
```

The event

The condition

The action

Options: The Event

```
CREATE TRIGGER DrinkTrig
    BEFOE INSERT ON Sells
        FOR EACH ROW
    BEGIN
        SET @drink =(SELECT name
            FROM Drinks
            WHERE name = new.drink);
        IF @drink IS NULL THEN
            INSERT INTO Drinks(name)
            VALUES(new.drink);
      END IF;
    END;
```

- BEFORE can be AFTER.
- INSERT can be DELETE or UPDATE.

Options: FOR EACH ROW

```
CREATE TRIGGER DrinkTrig
      BEFORE INSERT ON Sells
            FOR EACH ROW
     BEGIN
          SET @drink =(SELECT name
                FROM Drinks
                WHERE name = new.drink);
          IF @drink IS NULL THEN
              INSERT INTO Drinks(name,manf)
              VALUES(new.drink);
       END IF;
     END;
```

- FOR EACH ROW indicates row-level;
- Row level triggers are executed once for each modified tuple.

Options: Reference

```
CREATE TRIGGER DrinkTrig
      BEFORE INSERT ON Sells
            FOR EACH ROW
      BEGIN
          SET @drink =(SELECT name
                FROM Drinks
                WHERE name = new.drink);
          IF @drink IS NULL THEN
              INSERT INTO Drinks(name,manf)
              VALUES(new.drink);
        END IF;
      END;
```

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

  [NEW/OLD]

Options: Condition

```
CREATE TRIGGER DrinkTrig
      BEFORE INSERT ON Sells
            FOR EACH ROW
      BEGIN
            SET @drink =(SELECT name
                  FROM Drinks
                  WHERE name = new.drink);
           IF @drink IS NULL THEN
              INSERT INTO Drinks(name,manf)
              VALUES(new.drink);
        END IF;
      END;
```

- 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.
- Use new/old to access the new/old tuple (row).

Options: The Action

```
CREATE TRIGGER DrinkTrig
      BEFORE INSERT ON Sells
            FOR EACH ROW
      BEGIN
          SET @drink =(SELECT name
                FROM Drinks
                WHERE name = new.drink);
          IF @drink IS NULL THEN
              INSERT INTO Drinks(name,manf)
              VALUES(new.drink);
        END IF;
      END;
```

- There can be more than one SQL statement in the action.
- But queries make no sense in an action, so we are really limited to modifications.

Another Example

- Using Sells(cafe, drink, price) and a unary relation RipoffCafes(drink) created for the purpose of maintaining a list of cafes that raise the price of any drink by more than $1.

The Trigger

Updates let us talk about old and new tuples

The event

```
CREATE TRIGGER PriceTrig
AFTER UPDATE ON Sells
FOR EACH ROW
BEGIN
IF new.price > old.price + 1.00 THEN
    INSERT INTO RipoffCafes VALUES(new.cafe);
END IF;
END;
```

We need to consider each price change

Condition: a raise in price > $1

When the price change is great enough, add the cafe to RipoffCafes

Using Sells(cafe, drink, price) and a unary relation RipoffCafes(cafe) created for the purpose, maintain a list of cafes that raise the price of any drink by more than $1.