SQL: Modifications, Constraints & Triggers

CS 564- Fall 2015

ACKs: *Dan Suciu, Jignesh Patel, AnHai Doan*

NULL VALUES

NULL VALUES

- Tuples in SQL relations can have **NULL** as a value for one or more attributes
- The meaning depends on context:
  - *Missing value* : e.g. we know that Greece has some population, but we don't know what it is
  - *Inapplicable* : e.g., the value of attribute *spouse* for an unmarried person

COMPLICATIONS

- The logic of conditions in SQL is 3-valued logic: **TRUE**, **FALSE**, **UNKNOWN**
- When any value is compared with NULL, the truth value is **UNKNOWN**
- A query produces a tuple in the answer only if its truth value for the **WHERE** clause is only **TRUE**

COMPLICATIONS

- What happens for the condition IndepYear>1990 if it is NULL?
  - answer is UNKNOWN!
- What about the following?

```
SELECT COUNT(*)
FROM Country
WHERE IndepYear > 1990 OR IndepYear <= 1990 ;
```

Testing For Null

We can test for **NULL** explicitly:

- x IS NULL
- x IS NOT NULL

LEFT OUTER JOINS

- Include the tuple from the left relation even if there's no match on the right!

```
SELECT C.Name AS Country, MAX(T.Population)
FROM Country C LEFT OUTER JOIN City T
   ON C.Code = T.CountryCode
GROUP BY C.Name
```

Other Outer Joins

- Left outer join:
  - include the left tuple even if there is no match
- Right outer join:
  - include the right tuple even if there is no match
- Full outer join:
  - include the both left and right tuples even if there is no match

Database Modifications

MODIFYING THE DB

- A modification command does not return a result, but it changes the database
- There are 3 kinds of modifications:
  1. **Insert** tuple(s)
  2. **Delete** tuple(s)
  3. **Update** the value(s) of existing tuple(s)

INSERT

- To insert a single tuple:

  INSERT INTO <relation>

  VALUES (<list of values>);

- We may add to the relation name a list of attributes (if we forget the order)
- We may insert the entire result of a query into a relation:

  INSERT INTO <relation>

  (<subquery>);

DELETE

- To delete tuples:

  DELETE FROM <relation>

  WHERE <condition> ;

- How do we delete everything?

  DELETE FROM <relation> ;

- Be careful! *All* tuples that satisfy the WHERE clause are deleted!

UPDATE

- To change certain attributes in certain tuples of a relation:

UPDATE <relation>

SET <list of attribute assignments>

WHERE <condition> ;

VIEWS

VIEW DEFINITION

- A view is a virtual table, a relation that is defined in terms of the contents of other tables and views
- To create one:

CREATE VIEW <name> **AS** <query> ;

- In contrast, a relation whose value is really stored in the database is called a base table

EXAMPLE

```
CREATE VIEW OfficialCountryLanguage AS
SELECT C.Name AS CountryName,
       L.Language AS Language
FROM CountryLanguage L, Country C
WHERE L.CountryCode = C.Code
     AND L.IsOfficial = 'T' ;
```

How To Use Views

- You may query a view as if it were a base table
- **BUT** there is a limited ability to modify views!
- The DBMS interprets the query as if the view were a base table
- The queries defining any views used by the query are replaced by their algebraic equivalents, and added to the expression tree for the query

CONSTRAINTS & TRIGGERS

CONSTRAINTS & TRIGGERS

- An integrity constraint is a relationship among data elements that the DBMS is required to enforce
  - Example: keys, foreign keys

- A trigger is a procedure that is executed when a specified condition occurs (e.g. tuple insertion)

INTEGRITY CONSTRAINTS (IC)

- key
- foreign-key, or referential-integrity
- domain constraints
  - e.g. NOT NULL
- tuple-based constraints
- assertions: any SQL boolean expression

FOREIGN KEY

- Use the keyword **REFERENCES**, as:

  FOREIGN KEY (<list of attributes>)
  REFERENCES <relation> (<attributes>)

- Referenced attributes must be declared **PRIMARY KEY** or **UNIQUE**

Foreign Key

```
CREATE TABLE Author(
   authorid INTEGER PRIMARY KEY,
   name TEXT) ;

CREATE TABLE Book(
   bookid INTEGER PRIMARY KEY,
   title TEXT,
   author INTEGER,
   FOREIGN KEY (author) REFERENCES
   Author(authorid));
```

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

There are 3 ways to enforce foreign key constraints!

Action 1: Reject

- This is the default action if a foreign key is declared
- The insertion/deletion/update is rejected and not executed

Action 2: Cascade Update

- When a tuple referenced is updated, the update propagates to the tuples that reference it

```
CREATE TABLE Book(
    bookid INTEGER PRIMARY KEY,
    title TEXT,
    author INTEGER,
    FOREIGN KEY (author) REFERENCES
    Author(authorid)
    ON UPDATE CASCADE);
```

Action 2: Cascade Delete

- When a tuple referenced is deleted, the deletion propagates to the tuples that reference it

```
CREATE TABLE Book(
    bookid INTEGER PRIMARY KEY,
    title TEXT,
    author INTEGER,
    FOREIGN KEY (author) REFERENCES
    Author(authorid)
    ON DELETE CASCADE);
```

ACTION 3: SET NULL

- When a delete/update occurs, the values that reference the deleted tuple are set to **NULL**

```
CREATE TABLE Book(
   bookid INTEGER PRIMARY KEY,
   title TEXT,
   author INTEGER,
   FOREIGN KEY (author) REFERENCES
   Author(authorid)
   ON UPDATE SET NULL);
```

What to Choose

- When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates

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

- Otherwise, the default (reject) is used

DOMAIN CONSTRAINTS

- A constraint on the value of a particular attribute:

  CHECK (<condition>)

- We can use the attribute, but any other relation or attribute name must be in a subquery

```
CREATE TABLE Book(
    bookid INTEGER PRIMARY KEY CHECK(bookid >= 0),
    title TEXT,
    author INTEGER,
    FOREIGN KEY (author) REFERENCES Author(authorid));
```

DOMAIN CONSTRAINTS

- A check is **checked** only when a value for that attribute is inserted or updated
- We can also add more complex constraints:

```
CREATE TABLE Book(
    bookid INTEGER PRIMARY KEY,
    title TEXT,
    author INTEGER,
    FOREIGN KEY (author) REFERENCES Author(authorid)
    CHECK (bookid >= 0 or title IS NOT NULL)
);
```

ASSERTIONS

- Defined by:

```
CREATE ASSERTION <name>
CHECK ( <condition> );
```

- The condition may refer to any relation or attribute in the database schema

```
CREATE ASSERTION LowPrice CHECK (
   NOT EXISTS (
      SELECT * FROM Book
      WHERE price <= 20 AND authorid = 111)
) ;
```

ASSERTIONS

- 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 and check only these

TRIGGERS: MOTIVATION

- Checks have limited capabilities
- Assertions are sufficiently general for most constraint applications, but they are hard to implement efficiently
- A trigger allows the user to specify when the check occurs

TRIGGERS

Procedure that starts automatically if specified changes occur to the DBMS

- Three parts:
  - **Event** (activates the trigger)
  - **Condition** (tests whether the triggers should run)
  - **Action** (what happens if the trigger runs)

EXAMPLE

```
CREATE TRIGGER addAuthor
AFTER INSERT ON Book
FOR EACH ROW
 WHEN (NEW.author NOT IN
   (SELECT authorid FROM Author))
BEGIN
INSERT INTO Author
   VALUES (NEW.author, 'NewAuthor') ;
END ;
```

TRIGGER: CONDITION

- **AFTER / BEFORE.**
  - Also **INSTEAD OF** if the relation is a view
- **INSERT / DELETE / UPDATE**
  - **UPDATE** can be **UPDATE . . . ON** a particular attribute!

TRIGGER: 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

TRIGGER: 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>

TRIGGER: ACTION

- There can be more than one SQL statement in the action
  - Surround by **BEGIN . . . END**

- But queries make no sense in an action, so we are essentially limited to modifications