SQL: Databases Modification

Yannis Velegrakis

Database Modifications

Modification = insert + delete + update

e.g. to insert a tuple we use the statement

INSERT INTO <relation> **VALUES** (<list of values>)

- In the above statement the values as listed in the same order with which the attributes were declared
- If we want to ignore this order then we list the attributes as arguments of the relation

Example:

Consider *StarsIn*(*movieTitle*, *movieYear*, *starName*)

Insert the fact that Sydney Green stars in The Maltese Falcon

INSERT INTO *StarsIn*(*movieTitle, starName, movieYear)*
VALUES('The Maltese Falcon', 'Sydney Green', 1942);

Insertion of the Result of a Query

Syntax: **INSERT INTO** <relation> (<subquery>).

Example:

Consider

Studio(*name, address, presC#*)

Movie(*title*, *year*, *length*, *inColor*, *studio-Name*, *prodC#*)

Add to relation *Studio* all the studios that are mentioned in relation *Movie*

```
INSERT INTO Studio(name)
      SELECT DISTINCT studio-Name
      FROM Movie
      WHERE studio-Name NOT IN (SELECT name
                                FROM Studio)
```

Deletion

Syntax: **DELETE FROM** <relation> **WHERE** <condition>

Semantic: Deletes all tuples satisfying the condition from

the named relation

Example:

Consider *StarsIn*(*movieTitle*, *movieYear*, *starName*)

Sydney Green was not a star in The Maltese Falcon…

DELETE FROM *StarsIn*
WHERE *movieTitle* = 'The Maltese Falcon' **AND**
movieYear = 1942 **AND** *starName* = 'Sydney Green'

As another example, to make the *StarsIn* relation empty execute

DELETE FROM *StarsIn*

Updates

Syntax: **UPDATE** <relation>

SET <new-value assignments>

WHERE <condition>

Example:

Consider

MovieExec(*name*, *address*, *cert#*, *netWorth*)

Studio(*name, address, presC#*)

Prefix the name of every movie executive with the title ‘Pres. ‘

if the executive is the president of a studio.

UPDATE *MovieExec*
SET *name* = ‘Pres. ‘ || *name*
WHERE *cert#* **IN** (**SELECT** *presC#* **FROM** *Studio)*

Defining a Database Schema

Syntax: **CREATE TABLE** <name> (<list of elements>)

- Principal elements are attributes and their types, but key declarations and constraints also appear.
- Similar **CREATE X** commands for other schema elements **X**: **VIEWS**, **INDEX**, **ASSERTION**, **TRIGGER**.
- "**DROP X** *name*" deletes the created element of kind **X** with that name.

Example: *Studio*(*name*, *address*, *presC#*)

```
CREATE TABLE Studio (
    name CHAR(20),
    address VARCHAR(20),
    presC# REAL)

DROP TABLE Studio
```

Types

1. **INT** or **INTEGER**.
2. **BOOLEAN**
3. **REAL** or **FLOAT**.
4. **CHAR**(*n*) = fixed length character string, padded with "pad characters."
5. **VARCHAR**(*n*) = variable-length strings up to *n* characters.
6. **DATE** & **TIME**

Declaring Keys

Use **PRIMARY KEY** or **UNIQUE**.

- But only one primary key, many **UNIQUE**s allowed.
- SQL permits implementations to create an *index* (data structure to speed access given a key value) in response to **PRIMARY KEY** only.
- SQL does not allow nulls in primary key, but allows them in “unique” columns (which may have two or more nulls, but not repeated non-null values).

Declaring Keys

Two places to declare:

1. After an attribute's type, if the attribute is a key by itself.

2. As a separate element.
   - Essential if key is >1 attribute.

Example

CREATE TABLE *Studio* (

name **CHAR**(20),

address **VARCHAR**(20),

presC# **REAL**,

PRIMARY KEY(*name*))

Example

Find the differences…

CREATE TABLE *Studio-version1* (

name **CHAR**(20),

address **VARCHAR**(20) **UNIQUE**,

presC# **REAL UNIQUE**,

PRIMARY KEY(*name*))

CREATE TABLE *Studio-version2* (

name **CHAR**(20),

address **VARCHAR**(20),

presC# **REAL**,

UNIQUE (*address*, *presC#*),

PRIMARY KEY(*name*))

Other Properties You Can Give to Attributes

1. **NOT NULL** = every tuple must have a real value for this attribute.
2. **DEFAULT** value = a value to use whenever no other value of this attribute is known.

Example:

```
CREATE TABLE Studio (
        name CHAR(30) PRIMARY KEY,
        addr CHAR(50) DEFAULT '123 Sesame St',
        presC# REAL NOT NULL)
```

Example

Consider executing the following statement

INSERT INTO *Studio*(*name, presC#*)
VALUES('Fox', 90943)

The result is the following tuple:

name	addr	*presC#*
Sally	123 Sesame St.	90943

Primary key is by default not NULL.
This insertion is legal. It is OK to list a subset of the attributes and values for only this subset. But if we had forgot to specify a value for *presC#* then the insertion could not be made.

Changing Columns

Add an attribute to relation *R* with

ALTER TABLE *R* **ADD** <column declaration>

Example:

CREATE TABLE *MovieStar* (

name **CHAR**(30),

address **VARCHAR**(255),

gender **CHAR(1)**,

birthdate **DATE,**

PRIMARY KEY(*name*))

ALTER TABLE *MovieStar* **ADD** *phone* **CHAR**(16) **DEFAULT** 'unlisted'

Columns may also be dropped…

Example"

ALTER TABLE *MovieStar* **DROP** *birthdate*

Constraints

Commercial relational systems allow much more “fine-tuning” of constraints than do the modeling languages we learned earlier.

In essence: SQL programming is used to describe constraints.

Outline:

1. Primary key declarations (already covered).
2. Foreign-keys = referential integrity constraints.
3. Attribute- and tuple-based checks = constraints within relations.
4. SQL Assertions = global constraints

Foreign Keys

In relation *R* a clause that "attribute *A* **REFERENCES** *S*(*B*)"
says that whatever values appear in the *A* column of *R*
must also appear in the *B* column of relation *S*.

NOTE*: B* must be declared as the primary key for *S*.

Example:

Consider

MovieExec(*name*, *address*, *cert#*, *netWorth*)

Studio(*name, address, presC#*)

CREATE TABLE *Studio* (
name **CHAR**(30) **PRIMARY KEY**,
addr **CHAR**(50) **DEFAULT** '123 Sesame St',
presC# **REAL REFERENCES** *MovieExec(cert#)*)

Foreign Keys (cont.)

Alternative:

Add another element declaring the foreign key, as:

CREATE TABLE *Studio* (

name **CHAR**(30) **PRIMARY KEY**,

addr **CHAR**(50) **DEFAULT** '123 Sesame St',

presC# **REAL**.

FOREIGN KEY (presC#) **REFERENCES** *MovieExec(cert#)*)

Extra element is essential if the foreign key has

more than one attribute…

Foreign Key Constraint Violation

Two ways to violate the constraint:

1. Insert or update a *Studio* tuple so it refers to a non-existent movie executive
   - Always rejected.

2. Delete or update a *MovieExec* tuple that has a *cert#* value some *Studio* tuples refer to.
   a) Default: reject.
   b) *Cascade*: Ripple changes to referring *Studio* tuples.
   c) *Set* **Null**: Change referring tuples to have **NULL** in referring components.

Selecting a Policy

Add **ON** [**DELETE**, **UPDATE**] [**CASCADE**, **SET NULL**] to the declaration of foreign key.

Example:

CREATE TABLE *Studio* (

name **CHAR**(30) **PRIMARY KEY**,

addr **CHAR**(50) **DEFAULT** '123 Sesame St',

presC# **REAL**.

FOREIGN KEY (presC#) **REFERENCES** *MovieExec(cert#)*

ON DELETE SET NULL

ON UPDATE CASCADE)

Choosing a “correct” policy is a design decision…

Attribute-Based Checks

Follow an attribute by a condition that must hold for

that attribute in each tuple of its relation.

- Syntax: **CHECK** (<condition>).
  - Condition may involve the checked attribute.
  - Other attributes and relations may be involved, but *only* in subqueries.
- Condition is checked only when the associated attribute changes (*i.e.*, an insert or update occurs).

Example

```
CREATE TABLE Studio (
    name CHAR(30) PRIMARY KEY,
    addr CHAR(50) DEFAULT '123 Sesame St',
    presC# REAL CHECK (presC# >= 100000),
    FOREIGN KEY (presC#) REFERENCES MovieExec(cert#)
        ON DELETE SET NULL
        ON UPDATE CASCADE)
```

- Check on *presC#* is like a foreign-key constraint, except:
  - The check occurs only when we add a tuple or change the *presC#* in an existing tuple, not when we delete a tuple from *Studio*

Tuple-Based Checks

Separate element of table declaration.

- Syntax: like attribute-based check.
- But condition can refer to any attribute of the relation.
  - Or to other relations/attributes in subqueries.
- Checked whenever a tuple is inserted or updated.

Example

If a star's gender is male, then his name must not begin with 'Ms.'

CREATE TABLE *MovieStar* (

name **CHAR**(30) **PRIMARY KEY**,

address **CHAR**(255),

gender **CHAR**(1),

birthdate **DATE**,

CHECK (*gender* = 'F' **OR** *name* **NOT LIKE** 'Ms. %'**)**

SQL Assertions

- Database-schema constraints
- Checked whenever a mentioned relation changes.
- Syntax:

  CREATE ASSERTION < name>
  CHECK(<condition>)

Example

Consider

MovieExec(*name*, *address*, *cert#*, *netWorth*)

Studio(*name, address, presC#*)

What if we wish to require that no one can become the president of a studio unless their net worth is at least $10,000,000

```
CREATE ASSERTION RichPres CHECK
   (NOT EXISTS(SELECT *
                  FROM Studio, MovieExec
                  WHERE presC# = cert# AND
                        netWorth < 10000000))
```

Checked whenever *Studio* and *MovieExec* change

Triggers

Often called event-condition-action rules.

- *Event* = a class of changes in the DB, *e.g.*, “insertions into Beers.”
- *Condition* = a test as in a where-clause for whether or not the trigger applies.
- *Action* = one or more SQL statements.
- Differ from checks or SQL assertions in that:
  1. Triggers invoked by the event; the system doesn’t have to figure out when a trigger could be violated.
  2. Condition not available in checks.

Example

Consider

MovieExec(*name*, *address*, *cert#*, *netWorth*)

Whenever we update the netWorth of a movie executive we want the new value to be bigger than the old value.

```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec
REFERENCING
        OLD ROW AS OldTuple,
        NEW ROW AS NewTuple
FOR EACH ROW
WHEN(OldTuple.netWorth > NewTuple.netWorth)
        UPDATE MovieExec
        SET netWorth = OldTuple.netWorth
        WHERE cert# = NewTuple.cert#
```