Constraints

DSCI 551

Wensheng Wu

Kinds of Constraints

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

Example

Parent: department(id, name)
child: person(pid, name, did)

```
-- parent: delete, update
Create table person(pid int primary key,
          name char(10), did int, foreign key (did)
          references department(id)
          on delete set null
          on update set null);
```

Keys

- Specified using "primary key" or "unique"

```
create table Sells(
     bar varchar(100) references
          Bars(name),
     beer varchar(100) references
          Beers(name),
     price real,
     primary key(bar, beer)
);
```

Foreign Keys

- Consider Relation 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* constraint.

Expressing Foreign Keys

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

  FOREIGN KEY (<list of attributes>)

    REFERENCES <relation> (<attributes>)

- Note MySQL seems to enforce FK only when defined as an element

Example: Express FK with Attribute

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

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

Example: Express FK 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));
```

Note parentheses are necessary!

Primary Key vs. Unique

- Referenced attributes must be declared as PRIMARY KEY or UNIQUE.
  - Otherwise, MySQL does not allow creation of the table
  - Note that primary key can not be null, but unique attribute can

- Null values can be inserted into attribute of foreign key
  - Even though it refers to primary key in referenced table

Example of FKs with Unique Attributes

- create table R (a int primary key);
  - insert into R values (1);
  - Select * from R;

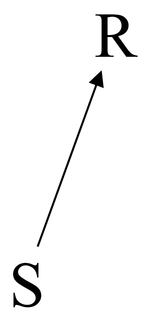
Or "a int unique"

- create table S(b int, foreign key (b) references R(a));
  - insert into S values (1);
  - insert into S values (null); // this works even though "a" is primary key in R
  - select * from S;

Enforcing Foreign-Key Constraints

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

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



Actions Taken

- Suppose R = Beers, S = Sells.
- An insert or update to Sells that introduces a nonexistent beer must be rejected.
- A deletion or update to Beers that removes a beer value found in some tuples of Sells can be handled in three ways.

$$\begin{tikzcd}
 & \text{Beers} \\
\text{Sells} \arrow[ur] &
\end{tikzcd}$$

Actions Taken (Cont'd)

- 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 beers in Sells to NULL.

Example: Cascade

- 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: 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.

Choosing a Policy

- When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates.
- 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 (
  bar   CHAR(20),
  beer  CHAR(20),
  price REAL,
  FOREIGN KEY(beer)
    REFERENCES Beers(name)
    ON DELETE SET NULL
    ON UPDATE CASCADE );
```

Multi-attribute keys (unique, PK, FK)

```
create table R(a int, b int, unique(a, b));
insert into R values(1, null);

create table P(a int, b int, primary key(a, b));
insert into P values(1, 2);
insert into P values(2, null);

create table F(a int, b int, foreign key (a, b)
          references P (a, b));
insert into F values(1, null);

create table F1 (a int, foreign key (a) references P(a));
insert into F1 values(2);
```

Are they ok?

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.

Attribute-Based Checks

- Put 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.
- Note: MySQL does not seem to support this
  - Accept definition, but does not enforce it
- Other DBMS, e.g., PostgreSQL, may support it

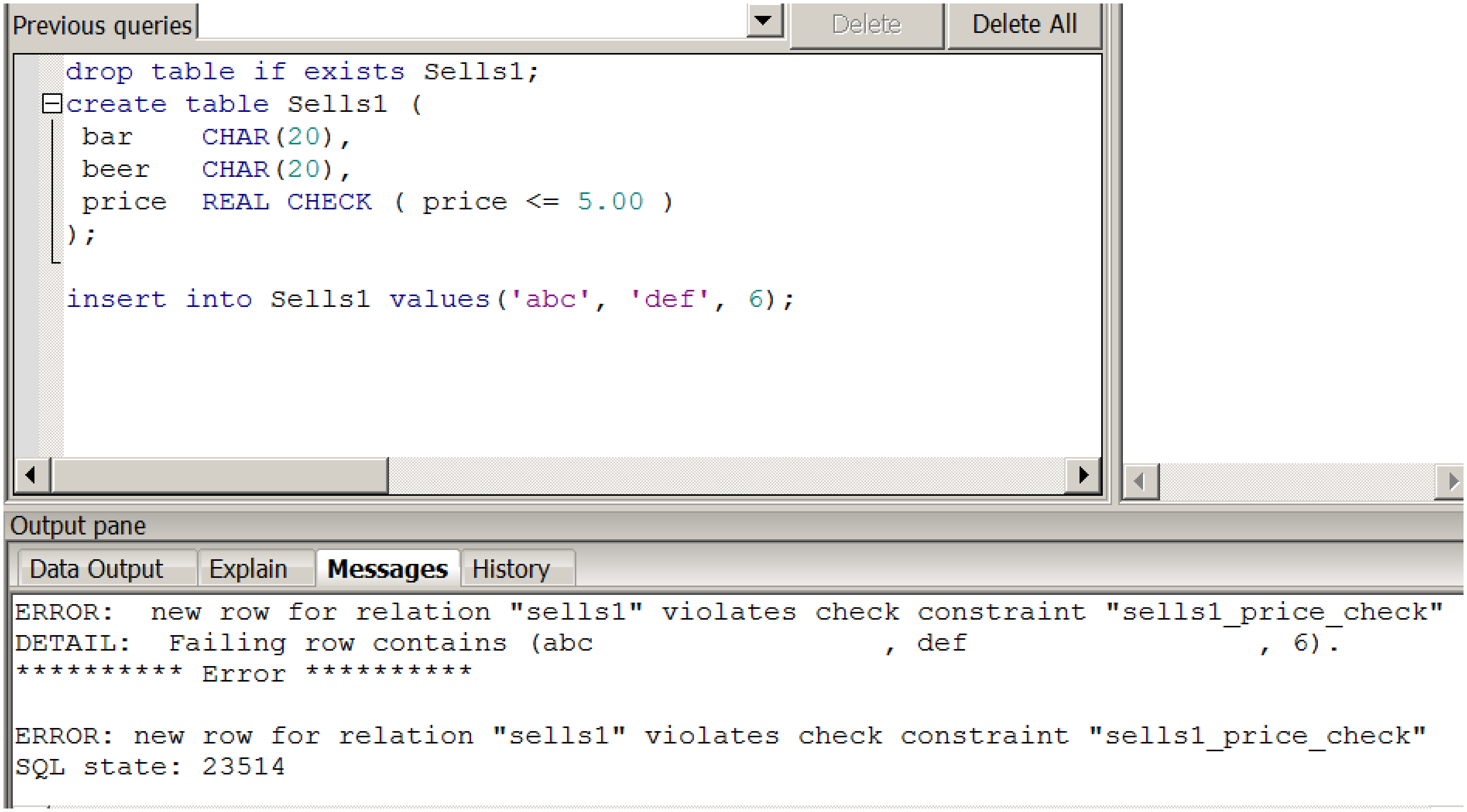
Example

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

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 (beer IN (SELECT name FROM Beers)) not checked if a beer is deleted from Beers (unlike foreign-keys).

PostgreSQL example



```
drop table if exists Sells1;
create table Sells1 (
 bar    CHAR(20),
 beer   CHAR(20),
 price  REAL CHECK ( price <= 5.00 )
);

insert into Sells1 values('abc', 'def', 6);
```

```
ERROR:  new row for relation "sells1" violates check constraint "sells1_price_check"
DETAIL:  Failing row contains (abc                 , def                 , 6).
********** Error **********

ERROR: new row for relation "sells1" violates check constraint "sells1_price_check"
SQL state: 23514
```

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 Joe’s Bar can sell beer for more than $5:

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

Example (work in PostgreSQL)

- insert into sells values('Joe', 'bud', 8);
  - This insert is ok
- update sells set bar = 'joe1'
  - This update is not ok

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.

- Very expensive to enforce
  - Neither PostgreSQL nor MySQL supports this

Example: Assertion

- In Sells(bar, beer, price), no bar may charge an average of more than $5.

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

Bars with an average price above $5

Example: Assertion

- In Drinkers(name, addr, phone) and Bars(name, addr, license), there cannot be more bars than drinkers.

```
CREATE ASSERTION FewBar CHECK (
  (SELECT COUNT(*) FROM Bars) <=
  (SELECT COUNT(*) FROM Drinkers)
);
```

Timing of Assertion Checks

- 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.
  - Example: No change to Beers can affect FewBar. Neither can an insertion to Drinkers.