CS 235: Introduction to Databases
Svetlozar Nestorov

Lecture Notes #13

Outline

• Active elements
  – Maintain database integrity and consistency.
  – Part of database schema.
• Constraints

Constraints

• Restrictions on the data in your database.
• 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.

Constraint Types

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.
   – Not found in MySQL.
5. MySQL Triggers.
   – A substitute for assertions.

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 $S$.
• $B$ must be declared the primary key (or unique) for $S$.
  – Why is this restriction necessary?

Example

```
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 
); 
```
Alternative Declaration

- Add another element declaring the foreign key, as:
  ```sql
  CREATE TABLE Sells (
      bar CHAR(20),
      beer CHAR(20),
      price REAL,
      FOREIGN KEY (beer) REFERENCES Beers(name)
  );
  ```
- Extra element essential if the foreign key is more than one attribute.
- MySQL recognizes only this declaration.

Foreign Keys in MySQL

- Both the referenced and referencing tables must be of type InnoDB.
  - Default type is MyISAM (indexed sequential access method)
- The FOREIGN KEY syntax must be used.
- In the referenced table, there must be an index on the referenced columns
  - PRIMARY KEY or UNIQUE create one automatically.

MySQL Example

```sql
CREATE TABLE Beers (
    name CHAR(20) PRIMARY KEY,
    manf CHAR(20)
) TYPE = InnoDB;

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

Foreign Key Constraint Violations

1. Insert or update a Sells tuple so it refers to a nonexistent beer.
   - Always rejected.
2. Delete or update a Beers tuple that has a beer value some Sells tuples refer to:
   a) Default: reject the modification.
   b) Cascade: Ripple changes to referring Sells tuple.
   c) Set Null: Change referring tuples to have NULL in referring components.

Example (Cascade)

- Delete Bud.
- Cascade deletes all Sells tuples that mention Bud.
- Update Bud to Budweiser.
- Change all Sells tuples with Bud in beer column to be Budweiser.

Example (Set-Null)

- Delete Bud.
- Set-null makes all Sells tuples with Bud in the beer component have NULL there.
- Update Bud to Budweiser.
- Set-null makes all Sells tuples with Bud in the beer component have NULL there.
Selecting a Policy

- Add ON DELETE, UPDATE [CASCADE, SET NULL] to foreign key declaration.

```sql
CREATE TABLE Sells (  
    bar CHAR(20),  
    beer CHAR(20),  
    price REAL,  
    FOREIGN KEY (beer) REFERENCES Beers(name)  
    ON DELETE SET NULL  
    ON UPDATE CASCADE
);
```

- Correct policy is a design decision.
  - E.g., what does it mean if a beer goes away? What if a beer changes its name?

Attribute-Based Checks

- Follow an attribute by a condition that must hold for that attribute in each tuple of its relation.
- **CHECK** (condition).
  - Condition may involve the checked attribute.
  - Other attributes and relations may be involved, but only in subqueries.
  - MySQL: CHECK parsed but ignored.
- Condition is checked only when the associated attribute changes (i.e., an insert or update occurs).

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

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

Tuple-Based Checks

- Separate element of table declaration.
- Form: like attribute-based check.
- But condition can refer to any attribute of the relation.
  - Or to other relations/attributes in subqueries.
  - Again: MySQL parses but ignores checks.
- Checked whenever a tuple is inserted or updated.

```sql
CREATE TABLE Sells (  
    bar CHAR(20),  
    beer CHAR(20),  
    price REAL,  
    CHECK(bar = 'Ripoff' OR price <= 10.00)
);
```

Example

Example

- Only Ripoff bar can sell beer for more than $10.

SQL Assertions

- Database-schema constraint.
- Not present in MySQL.
- Checked whenever a mentioned relation changes.
- Syntax:
  - CREATE ASSERTION <name>  
    CHECK(<condition>);
<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| • No bar may charge an average of more than $5 for beer. `Sells(bar, beer, price)`  
CREATE ASSERTION NoRipoffBars  
CHECK(NOT EXISTS(  
    SELECT bar  
    FROM Sells  
    GROUP BY bar  
    HAVING 5.0 < AVG(price)  
)  
);  
• Checked whenever Sells changes. |

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| • There cannot be more bars than drinkers.  
`Bars(name, addr, license)` `Drinkers(name, addr, phone)`  
CREATE ASSERTION FewBars  
CHECK(  
    (SELECT COUNT(*) FROM Bars) <=  
    (SELECT COUNT(*) FROM Drinkers)  
);  
• Checked whenever Bars or Drinkers changes. |

<table>
<thead>
<tr>
<th>Example Aggregation Queries</th>
</tr>
</thead>
</table>
| • Find the person who likes the most beers.  
• Find the most likely pairing of a person and a beer.  
  – Most bars, frequented by a person, that serve the beer.  
  – Another condition?  
• Find the most likely couple: drinkers that frequent the most bars and like the most beers in common.  
  – Can we weigh number of bars and beers differently? |