Announcements

- Assignment 3 is due now!
- Assignment 4 is due next Tuesday!
- Midterm in class next Tuesday (Nov 1)
  - in class; open book/notes.

Outline

- Active elements
  - Maintain database integrity and consistency.
  - Part of database schema.
- Constraints
- Triggers

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:
  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

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).

Example

```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.

Example

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

- 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:
  
  ```sql
  CREATE ASSERTION <name>
  CHECK(<condition>);
  ```

Example
- No bar may charge an average of more than $5 for beer. `Sells(bar, beer, price)`
  
  ```sql
  CREATE ASSERTION NoRipoffBars
  CHECK(NOT EXISTS(
    SELECT bar
    FROM Sells
    GROUP BY bar
    HAVING 5.0 < AVG(price)
  ));
  ```
  - Checked whenever `Sells` changes.

Example
- There cannot be more bars than drinkers.
  
  ```sql
  CREATE ASSERTION FewBars
  CHECK(
    (SELECT COUNT(*) FROM Bars) <=
    (SELECT COUNT(*) FROM Drinkers)
  );
  ```
  - Checked whenever `Bars` or `Drinkers` changes.

Triggers (MySQL Version)
- Whenever we insert a new tuple into `Sells`, make sure the beer mentioned is also mentioned in `Beers`, and insert it (with a null manufacturer) if not.
  
  ```sql
  CREATE TRIGGER BeerTrig
  AFTER INSERT ON Sells
  FOR EACH ROW
  BEGIN
    INSERT IGNORE INTO Beers(name)
    VALUES(new.beer);
  END;
  ```

Options
- AFTER triggers cannot change the value of the inserted/updated tuple.
- BEFORE triggers can change the value of the inserted/updated tuple.
More Options

- INSERT can be DELETE or UPDATE
- FOR EACH ROW can be omitted, with an important effect: the action is done once for the relation(s) consisting of all changes.
  - MySQL recognized only "FOR EACH ROW"

Explanation

- There are two special (transition) variables new and old, representing the new and old tuple in the change.
  - old makes no sense in an insert, and new makes no sense in a delete.

More Explanations

- The action is any statement allowed in a MySQL function
  - Simplest form: surround one or more SQL statements with BEGIN and END.
  - However, select-from-where has a limited form.
- Need to (temporarily) redefine default delimiter (;) to another character, e.g. ($)
- MySQL triggers are part of the database schema, like tables or views.

Even More Explanations

- Important MySQL constraint: the action cannot change the relation that triggers the action.
- MySQL returns an error only at run time.

Example

- Maintain a list of all the bars that raise their price for some beer by more than $1. RipoffBars(bar)
  DELIMITER //
  CREATE TRIGGER PriceTrig
  AFTER UPDATE ON Sells
  FOR EACH ROW
  BEGIN
  IF (NEW.price > OLD.price + 1) THEN
    INSERT INTO RipoffBars VALUES(NEW.bar);
  END IF;
  END; //
  DELIMITER ;

Attribute Checks with Triggers

- Create two triggers BEFORE INSERT and BEFORE UPDATE
  - What about BEFORE DELETE?
- The triggers check attribute constraint and if not satisfied make a modification that will be rejected, so the triggering INSERT or UPDATE will fail.
**Example**

```sql
CREATE TABLE Sells (  
    bar CHAR(20) NOT NULL,  
    beer CHAR(20),  
    price REAL;
);

- Check that the price is not more than $12.
```

**Example**

```sql
CREATE TRIGGER PriceInsTrig  
BEFORE INSERT ON Sells  
FOR EACH ROW  
BEGIN  
    IF (NEW.price > 12) THEN  
        SET NEW.bar = NULL;
    END IF;
    END;
```

**Example**

```sql
CREATE TRIGGER PriceUpdTrig  
BEFORE UPDATE ON Sells  
FOR EACH ROW  
BEGIN  
    IF (NEW.price > 12) THEN  
        SET NEW.bar = NULL;
    END IF;
    END;
```

**SQL Triggers**

- Covered in the book.
- Some differences, including:
  1. The MySQL restriction about not modifying the relation of the trigger or other relations linked to it by constraints is not present in SQL.
  2. The action in SQL is a list of (restricted) SQL statements.