CSE 562
Database Systems

Constraints, Triggers, Views, Indexes

Some slides are based or modified from originals by
Database Systems: The Complete Book,
Pearson Prentice Hall 2nd Edition

cse@buffalo

Example DB

Beers(name, manf)
Bars(name, addr, license)
Sells(bar, beer, price)
Drinkers(name, addr, phone)
Likes(drinker, beer)
Frequents(drinker, bar)

– Underline indicates key attributes

Constraints and Triggers

- A constraint is a relationship among data elements that the DBMS is required to enforce
  - **Example**: key constraints
- Triggers are only executed when a specified condition occurs, e.g., insertion of a tuple
  - Easier to implement than complex constraints

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

Review: Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute
- **Example**:

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


Review: Multi-Attribute Key

- The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
     bar       CHAR(20),
     beer      VARCHAR(20),
     price     REAL,
     PRIMARY KEY (bar, beer)
);
```


Foreign Keys

- Values appearing in attributes of one relation must appear together in certain attributes of another relation
- **Example**: in Sells(bar, beer, price), we might expect that a beer value also appears in Beers.name

Example: As Schema 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)
);
```

Actions Taken

- 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 (next slide)

Actions Taken (cont'd)

- Default: Reject the modification
- Cascade: Make the same changes in Sells
  - **Deleted beer**: delete Sells tuple
  - **Updated beer**: change value in Sells
- Set NULL: Change the beer to NULL

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: Setting Policy

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

Attribute-Based Checks

- Constraints on the value of a particular attribute
- Add CHECK(<condition>) 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

Example: Attribute-Based Check

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


Tuple-Based Checks

- CHECK (<condition>) may be added as a relation-schema element
- The condition may refer to any attribute of the relation
  - But 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''s Bar' OR
                  price <= 5.00)
);
```

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

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)
);
```


Triggers: Motivation

- Assertions are powerful, but the DBMS often can’t tell when they need to be checked
- Attribute- and tuple-based checks are checked at known times, but are not powerful
- Triggers let the user decide when to check for any condition

Event-Condition-Action Rules

- Another name for "trigger" is ECA rule, or **event-condition-action** rule
- Event: typically a type of database modification, e.g., "insert on Sells"
- Condition: Any SQL Boolean-valued expression
- Action: Any SQL statements

Preliminary Example: A Trigger

- Instead of using a foreign-key constraint and rejecting insertions into Sells(bar, beer, price) with unknown beers, a trigger can add that beer to Beers, with a NULL manufacturer

Example: Trigger Definition

```
CREATE TRIGGER BeerTrig
  AFTER INSERT ON Sells                  The event
  REFERENCING NEW ROW AS NewTuple
  FOR EACH ROW
  WHEN (NewTuple.beer NOT IN
      (SELECT name FROM Beers))          The condition
  INSERT INTO Beers(name)
      VALUES(NewTuple.beer);             The action
```


Options: The Event

- AFTER can be BEFORE
- INSERT can be DELETE or UPDATE
  - And UPDATE can be UPDATE … ON a particular attribute

Options: 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: execute once for each modified tuple
- Statement-level triggers: execute once for a SQL statement, regardless of how many tuples are modified

Options: REFERENCING

- INSERT statements imply a new tuple (for row-level) or new table (for statement-level)
  - The "table" is the set of inserted tuples
- DELETE implies an old tuple or table
- UPDATE implies both
- Refer to these by

  [NEW OLD][TUPLE TABLE] AS <name>

Options: The Condition

- Any Boolean-valued condition
- Evaluated on the database as it would exist before or after the triggering event, depending on whether BEFORE or AFTER is used
  - But always before the changes take effect
- Access the new/old tuple/table through the names in the REFERENCING clause

Options: The Action

- There can be more than one SQL statement in the action
  - Surround by BEGIN … END if there is more than one
- But queries make no sense in an action, so we are really limited to modifications

Another Example

- Using Sells(bar, beer, price) and a unary relation RipoffBars(bar), maintain a list of bars that raise the price of any beer by more than $1

The Trigger

```
CREATE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  REFERENCING
     OLD ROW AS ooo
     NEW ROW AS nnn
  FOR EACH ROW
  WHEN(nnn.price > ooo.price + 1.00)
  INSERT INTO RipoffBars
     VALUES(nnn.bar);
```

The event – only changes to prices

Updates let us talk about old and new tuples

We need to consider each price change

Condition: a raise in price > $1

When the price change is great enough, add the bar to RipoffBars

Views

- A view is a relation defined in terms of stored tables (called base tables) and other views
- Two kinds:
  - Virtual = not stored in the database; just a query for constructing the relation
  - Materialized = actually constructed and stored

Declaring Views

- Declare by:
  CREATE [MATERIALIZED] VIEW <name> AS
  <query>;
- Default is virtual

Example: View Definition

- CanDrink(drinker, beer) is a view "containing" the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer:

```
CREATE VIEW CanDrink AS
    SELECT drinker, beer
    FROM Frequents, Sells
    WHERE Frequents.bar = Sells.bar;
```


Example: Accessing a View

- Query a view as if it were a base table
  - Also: a limited ability to modify views if it makes sense as a modification of one underlying base table
- **Example query**:

```
SELECT beer
FROM CanDrink
WHERE drinker = 'Sally';
```


Materialized Views

- Problem: each time a base table changes, the materialized view may change
  - Cannot afford to re-compute the view with each change
- Solution: Periodic reconstruction of the materialized view, which is otherwise "out of date"

Example: A Data Warehouse

- Wal-Mart stores every sale at every store in a database
- Overnight, the sales for the day are used to update a data warehouse = materialized views of the sales
- The warehouse is used by analysts to predict trends and move goods to where they are selling best

Indexes

- Index = data structure used to speed access to tuples of a relation, given values of one or more attributes
- Could be a hash table, but in a DBMS it is always a balanced search tree with giant nodes (a full disk page) called a B-tree

Declaring Indexes

- No standard!
- Typical syntax:

```
CREATE INDEX BeerInd ON Beers(manf);
CREATE INDEX SellInd ON Sells(bar, beer);
```


Using Indexes

- Given a value **v**, the index takes us to only those tuples that have **v** in the attribute(s) of the index
- **Example**: use BeerInd and SellInd to find the prices of beers manufactured by Pete's and sold by Joe (next slide)

Using Indexes (cont'd)

```
SELECT price
FROM Beers, Sells
WHERE manf = 'Pete''s' AND
      Beers.name = Sells.beer AND
      bar = 'Joe''s Bar';
```

1. Use BeerInd to get all the beers made by Pete's
2. Then use SellInd to get prices of those beers, with bar = 'Joe''s Bar'

Database Tuning

- A major problem in making a database run fast is deciding which indexes to create
- Pro: An index speeds up queries that can use it
- Con: An index slows down all modifications on its relation because the index must be modified too

Example: Tuning

- Suppose the only things we did with our beers database was:
  - Insert new facts into a relation (10%)
  - Find the price of a given beer at a given bar (90%)
- Then SellInd on Sells(bar, beer) would be wonderful, but BeerInd on Beers(manf) would be harmful

This Time

- Constraints and Triggers
  - Chapter 7
- Views and Indexes
  - Chapter 8: 8.1, 8.3, 8.5.1, 8.5.2