SQL Part 3

Database Modifications

Integrity constraints

Peter Scheuermann

Outline

- **Database Modifications**
- **Static Integrity Constraints**
  - Domain Constraints
  - Key / Referential Constraints
  - Semantic Integrity Constraints
- **Dynamic Integrity Constraints**
  - **Triggers**

Database Modifications

- A ***modification*** command does not return a result (as a query does), but changes the database in some way.
- Three kinds of modifications:

1. ***Insert*** a tuple or tuples.
2. ***Delete*** a tuple or tuples.
3. ***Update*** the value(s) of an existing tuple or tuples.

Our Running Example

- Our SQL queries will be based on the following database schema.
  - Underline indicates key attributes.

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

Insertion

- To insert a single tuple:

  INSERT INTO <relation>

  VALUES (<list of values>);

- Example: add to Likes(drinker, beer) the fact that Sally likes Bud.

  INSERT INTO Likes

  VALUES('Sally', 'Bud');

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes.
- Two reasons to do so:
  1. We forget the standard order of attributes for the relation.
  2. We don't have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

Example: Specifying Attributes

- Another way to add the fact that Sally likes Bud to Likes(drinker, beer):

```
INSERT INTO Likes(beer, drinker)
VALUES('Bud', 'Sally');
```

Adding Default Values

- In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value.
- When an inserted tuple has no value for that attribute, the default will be used.

Example: Default Values

```
CREATE TABLE Drinkers (
      name CHAR(30) PRIMARY KEY,
      addr   CHAR(50)
               DEFAULT '123 Sesame St.',
      phone CHAR(16)
);
```

Example: Default Values

```
INSERT INTO Drinkers(name)
VALUES('Sally');
```

Resulting tuple:

name	address	phone
Sally	123 Sesame St	NULL

Inserting Many Tuples

- We may insert the entire result of a query into a relation, using the form:

```
INSERT INTO <relation>
( <subquery> );
```

Example: Insert a Subquery

- Using Frequents(drinker, bar), enter into the new relation PotBuddies(name) all of Sally's "potential buddies," i.e., those drinkers who frequent at least one bar that Sally also frequents.

Solution

The other drinker

```
INSERT INTO PotBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = 'Sally' AND
    d2.drinker <> 'Sally' AND
    d1.bar = d2.bar
);
```

Pairs of Drinker tuples where the first is for Sally, the second is for someone else, and the bars are the same.

Deletion

- To delete tuples satisfying a condition from some relation:

  DELETE FROM <relation>

  WHERE <condition>;

- Important : FROM clause can specify only ONE relation !

Example: Deletion

- Delete from Likes(drinker, beer) the fact that Sally likes Bud:

```
DELETE FROM Likes
WHERE drinker = 'Sally' AND
      beer = 'Bud';
```

Example: Delete all Tuples

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.

Example: Delete Some Tuples

- Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

```
DELETE FROM Beers b
WHERE EXISTS (
   SELECT name FROM Beers
   WHERE manf = b.manf AND
         name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

Semantics of Deletion --- (1)

- Suppose Anheuser-Busch makes only Bud and Bud Lite.
- Suppose we come to the tuple *b* for Bud first.
- The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
- Now, when *b* is the tuple for Bud Lite, do we delete that tuple too?

Semantics of Deletion --- (2)

- Answer: we *do* delete Bud Lite as well.
- The reason is that deletion proceeds in two stages:

1. Mark all tuples for which the WHERE condition is satisfied.
2. Delete the marked tuples.

Updates

- To change certain attributes in certain tuples of a relation:

  UPDATE <relation>

  SET <list of attribute assignments>

  WHERE <condition on tuples>;

Example: Update

- Change drinker Fred's phone number to 555-1212:

```
UPDATE  Drinkers
SET     phone = '555-1212'
WHERE   name = 'Fred';
```

Example: Update Several Tuples

- Make $4 the maximum price for beer:

```
UPDATE  Sells
SET     price = 4.00
WHERE   price > 4.00;
```

Semantic Integrity Constraints(ICs)

- **Objective:**
  - capture semantics of the mini-world in the database
  - ensuring that authorized changes to the database do not result in a loss of **data consistency**
  - guard against accidental damage to the database (avoid data entry errors)
- Advantages of a centralized, automatic mechanism to ensures semantic integrity constraints:
  - More effective integrity control
  - Stored data is more faithful to real-world meaning
  - Easier application development, better maintainability
- Note: DBMS allow to capture more ICs than, e.g., ER Model

Integrity Constraint (IC)

- **Integrity Constraint (IC)**:
  condition that must be true for every instance of a database
  - A **legal** instance of a relation is one that satisfies all specified ICs
    - DBMS should never allow illegal instances….
- ICs are *specified* in the database schema
  - The database designer is responsible to ensure that the integrity constraints are not contradicting each other!
- ICs are *checked* when the database is modified
  - With one degree of freedom:
    - After a SQL statement, or at the end of a transaction?
- Possible *reactions* if an IC is violated:
  - Undoing of a database operation
  - Abort of the transaction
  - Execution of “maintenance” operations to make db legal again

Types of Integrity Constraints

- **Static Integrity Constraints**
  describe conditions that every *legal instance* of a database must satisfy
  - Inserts / deletes / updates that violate ICs are disallowed
  - Three kinds:
    - *Domain Constraints*
    - *Key Constraints & Referential Integrity*
    - *Semantic Integrity Constraints; Assertions*

- **Dynamic Integrity Constraints**
  are predicates on database state changes
  - ***Triggers***

Domain (Attribute) Constraints

- The most elementary form of an integrity constraint:
- Fields must be of right data domain
  - always enforced for values inserted in the database
  - Also: queries are tested to ensure that the comparisons make sense.
- SQL DDL allows domains of attributes to be restricted in the **create table** definition with the following clauses:
  - **CHECK** (*condition on attribute*) *<- Simple semantic integrity constraint*
    all values of the attribute must always satisfy the given predicate
  - **DEFAULT** *default-value*
    default value for an attribute if its value is omitted in an insert stmnt.
  - **NOT NULL**
    attribute is not allowed to become NULL

Example of Domain Constraints

```
CREATE TABLE Student
 (
      sid         INTEGER              PRIMARY KEY,
      name        VARCHAR(20)          NOT NULL,
      semester    INTEGER              DEFAULT 1 CHECK (semester > 0),
      birthday    DATE,
      country     VARCHAR(20)
 );
```

Semantic:

sid is primary key of **Student**
name must not be NULL
semester will be 1 if not specified by an insert
all other attributes can be NULL (**birthday** and **country**)

Example:

```
INSERT INTO Student(sid,name) VALUES (123,'Peter');
```

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

- Attribute-based checks performed only when a value for that attribute is inserted or updated.
  - Example: **CHECK (price <= 5.00**） checks every new price and rejects the modification (for that tuple) if the price is more than $5.
  - Example: **CHECK (beer IN (SELECT name FROM Beers))** not checked if a beer is deleted from Beers (unlike foreign-keys).

Tuple-Based Checks

- CHECK (<condition>) may be added as a relation-schema element.
- 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''s Bar' OR
                  price <= 5.00)
);
```

Semantic Integrity Constraints

- Integrity constraints on more than one attribute
- Also, a name for integrity constraint would be very useful for administration / maintenance…
- SQL:

  CONSTRAINT *name* **CHECK** **(** *semantic-condition* **)**

- One can use subqueries to express constraint

Semantic Constraints Example

```
CREATE TABLE Enrollment
(
   sid     INTEGER    REFERENCES Student,
   c-code VARCHAR(8) REFERENCES Course ,
   empid  INTEGER    REFERENCES Lecturer,
   grade  INTEGER,
   CONSTRAINT grade CHECK (grade between 0 and 100),
   CONSTRAINT rightLecturer
        CHECK ( empid = (SELECT c.lecturer
                         FROM   Course c
                         WHERE  c.c_code=c-code))
);
```

.

SQL: Naming Integrity Constraints

- The **CONSTRAINT** clause can be used to name all types of integrity constraints
- Example:

```
CREATE TABLE Enrolled
(
  sid        INTEGER,
  c-code     VARCHAR(8),
  grade      CHAR(2),
  CONSTRAINT FK_sid_enrolled   FOREIGN KEY (sid)
                               REFERENCES Student
                               ON DELETE CASCADE,
  CONSTRAINT FK_cid_enrolled   FOREIGN KEY (c-code)
                               REFERENCES Course
                               ON DELETE CASCADE,
  CONSTRAINT CK_grade_enrolled CHECK(grade in ('F',…)),
  CONSTRAINT PK_enrolled      PRIMARY KEY (sid, c-code)
);
```

Example: Deferring Constraints

```
CREATE TABLE Course
(
   c_code        VARCHAR(8),
   title         VARCHAR(220),
   lecturer      INTEGER,
   credit_points INTEGER,
   CONSTRAINT Course_PK PRIMARY KEY (c_code),
   CONSTRAINT Course_FK FOREIGN KEY (lecturer)
     REFERENCES Lecturer DEFERABBLE INITIALLY DEFERRED
);
```

- Allows to insert a new course referencing a lecturer which is not present at that time, but who will be added later *in the same transaction*.
- Behaviour can be dynamically changed within a transaction with the SQL statement
  SET CONSTRAINT *Course_FK* **IMMEDIATE**;

Deferring Constraint Checking

- Any constraint - domain, key, foreign-key, semantic - may be declared:
  - **NOT DEFERRABLE**
    The default. It means that every time a database modification occurs, the constraint is checked immediately afterwards.
  - **DEFERRABLE**
    Gives the option to wait until a transaction is complete before checking the constraint.
    - **INITIALLY DEFERRED** wait until transaction end,
      but allow to dynamically change later

Assertions

- The integrity constraints seen so far are associated with a single table
  - Plus: they are required to hold only if the associated table is nonempty!
- Need for more general integrity constraints
  - E.g. integrity constraints over several tables
  - Always checked, independent if one table is empty

- **Assertion**: a predicate expressing a condition that we wish the database always to satisfy.
- SQL-92 syntax:
  create assertion <*assertion-name*> **check** (<*condition*>)

- Assertions are schema objects (like tables or views)
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate it
  - This testing may introduce a significant amount of overhead; hence assertions should be used with great care.

Assertion Example 1

- Asserting $\forall X: P(X)$ is achieved in a round-about fashion using not exists X such that not P(X)
- Example: For all students, the sum of all grades for a course must be less or equal than 10000

```
CREATE ASSERTION grade-constraint CHECK
(
  not exists ( select   c-code
               from     Enrollment
               group by c_code
               having   sum(grade) > 10000 )
)
```

Example Assertion 2

- 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.