SQL Constraints and Triggers

Week 12

SQL Constraints

- Constraints
  - Primary Key (covered)
  - Foreign Key (covered)
  - General table constraints
  - Domain constraints
  - Assertions
- Triggers

Primary Key Constraints

- Every table should have a primary key
- When a primary key constraint is created it specifies that:
  - The attributes of the primary key cannot be null
  - The primary key must be unique
- Violating a primary key causes the violating update to be rejected

Foreign Key Constraints

- Represents a relationship between two tables
- If a table **`R`** contains a foreign key on attributes **`{a}`** that references table **`S`**:
  - **`{a}`** generally correspond to the primary key of **`S`**
    - Must have the same number of attributes, and
    - The same domains
  - Any value for **`{a}`** in **`R`** must also exist in **`S`** except that
    - If **`{a}`** is not part of the primary key of **`R`** it may be null
  - There may be values for **`{a}`** <u>in **`S`**</u> that are <u>not</u> in **`R`**

Foreign Key Specification

- Foreign keys specify the actions to be taken if referenced records are updated or deleted
  - For example, create a foreign key in Account that references Branch
    - Assign accounts of a deleted branch to the Fairfax branch
    - Cascade any change in branch names

Cascading Changes

- It is possible that there can be a chain of foreign key dependencies
  - e.g. branches, accounts, transactions
- A cascading deletion in one table may cause similar deletions in a table that references it
  - If any cascading deletion or update causes a violation, the entire transaction is aborted

Referencing non-Primary Keys

- By default SQL foreign keys reference the primary key (of the referenced table)
- It is possible to reference a list of (non-primary-key) attributes
  - The list must be specified after the name of the referenced table
  - The specified list of attributes must be declared as a candidate key of the referenced table

General Constraints

- A general or **table** constraint is a constraint over a single table
  - Included in a table's **`CREATE TABLE`** statement
  - Table constraints may refer to other tables
- Defined with the **`CHECK`** keyword followed by a description of the constraint
  - The constraint description is a Boolean expression, evaluating to true or false
  - If the condition evaluates to false the update is rejected

Constraint Example

- Check that a customer's age is greater than 18, and that a customer is not an employee

```
CREATE TABLE Customer
   (SSN       CHAR(11),
    …,
    income    REAL,
    PRIMARY KEY (SSN),
    CONSTRAINT CustAge CHECK (age > 18),
    CONSTRAINT notEmp CHECK (SSN NOT IN
                   (SELECT empSSN
                    FROM Employee)))
```

Domain Constraints

- New domains can be created using the **`CREATE DOMAIN`** statement
  - Each such domain must have an underlying source type (i.e. an SQL base type)
  - A domain must have a name, base type, a restriction, and a default optional value
    - The restriction is defined with a **`CHECK`** statement
- Domains are part of the DB schema but are not attached to individual table schemata

Domain Constraint Example

- Create a domain for minors, who have ages between 0 and 18
  - Make the default age 10

```
CREATE DOMAIN minorAge INTEGER DEFAULT 10
   CHECK (VALUE > 0 AND VALUE <= 18)
```

Using Domain Constraints

- A domain can be used instead of one of the base types in a **`CREATE TABLE`** statement
  - Comparisons between two domains are made in terms of the underlying base types
    - e.g. comparing an age with an account number domain simply compares two integers
- The SQL:1999 standard introduced syntax for distinct types
  - Types are distinct so that values of different types cannot be compared
- Not supported by Oracle
  - Create a table that holds the domain values instead, and reference this table

Creating Domains in Oracle (review)

- Say you want to restrict the values of GPA (0 < GPA <= 4.0)

- Approach 1: Specify constraint when defining the table

```
CREATE TABLE Students
        (sid CHAR(20),
         name CHAR(20),
         login CHAR(10),
         age INTEGER,
         gpa REAL check(gpa <= 4.0 AND gpa > 0) );
```

Creating Domains

- Approach 2: After CREATING TABLE, use ALTER TABLE

```
CREATE TABLE Students
        (sid CHAR(20),
         name CHAR(20),
         login CHAR(10),
         age INTEGER,
         gpa REAL);

ALTER TABLE Students
ADD CONSTRAINT check_gpa CHECK(gpa > 0 AND gpa <= 4.0);
```

To specify a set of allowed values, do something like this (using either approach):

… CHECK(gender='M' OR gender='F')

Creating Types

- The SQL **`CREATE TYPE`** clause defines new types
  - To create distinct age and account number types:
    - **`CREATE TYPE Ages AS INTEGER`**
    - **`CREATE TYPE Accounts AS INTEGER`**
  - Assignments, or comparisons between ages and account numbers would now be illegal
    - Although it is possible to **cast** one type to another

Deferring Constraint Checking

- For circular references, or the chicken-and-egg problems:

```
CREATE TABLE chicken (cID INT PRIMARY KEY,
            eID INT REFERENCES egg(eID));

CREATE TABLE egg(eID INT PRIMARY KEY,
         cID INT REFERENCES chicken(cID));
```

Deferring Constraint Checking

- To get around this, create tables without foreign key constraints, then alter table:

```
CREATE TABLE chicken(cID INT PRIMARY KEY,
                     eID INT);
CREATE TABLE egg(eID INT PRIMARY KEY,
                 cID INT);

ALTER TABLE chicken ADD CONSTRAINT chickenREFegg
   FOREIGN KEY (eID) REFERENCES egg(eID)
   INITIALLY DEFERRED DEFERRABLE;

ALTER TABLE egg ADD CONSTRAINT eggREFchicken
   FOREIGN KEY (cID) REFERENCES chicken(cID)
   INITIALLY DEFERRED DEFERRABLE;
```

Deferring Constraint Checking

- To drop tables, drop the constraints first.

```
ALTER TABLE egg DROP CONSTRAINT eggREFchicken;
ALTER TABLE chicken DROP CONSTRAINT chickenREFegg;

DROP TABLE egg;
DROP TABLE chicken;
```

Assertions

- Table constraints apply to only one table
- Assertions are constraints that are separate from **`CREATE TABLE`** statements
  - Similar to domain constraints, they are separate statements in the DB schema
  - Assertions are tested **whenever the DB is updated**
    - Therefore they may introduce significant overhead

Note: Not supported in Oracle

Example Assertion

- Check that a branch's assets are greater than the total account balances held in the branch

```
CREATE ASSERTION assetCoverage
CHECK (NOT EXISTS
          (SELECT *
           FROM Branch B
           WHERE assets <
               (SELECT SUM (A.balance)
                FROM Account A
                WHERE A.brName = B.brName)))
```

Assertion Limitations

- There are some constraints that cannot be modeled with table constraints or assertions
  - What if there were participation constraints between customers and accounts?
    - Every customer must have at least one account and every account must be held by at least one customer
  - An assertion *could* be created to check this situation
    - But would prevent new customers or accounts being added!

Triggers

- A trigger is a procedure that is invoked by the DBMS as a response to a specified change
- A DB that has a set of associated triggers is referred to as an **active database**
- Triggers are available in most current commercial DB products
  - And are part of the SQL 1999 standard
- Triggers carry out **actions** when their triggering conditions are met
  - Generally SQL constraints only reject transactions

Why Use Triggers?

- Triggers can implement business rules
  - e.g. creating a new loan when a customer's account is overdrawn
- Triggers may also be used to maintain data in related database tables
  - e.g. Updating derived attributes when underlying data is changed, or maintaining summary data

Trigger Components

- Event (activates the trigger)
  - A specified modification to the DB
    - May be an insert, deletion, or change
    - May be limited to specific tables
    - The trigger may **fire** before or after the transaction
- Condition
- Action

Trigger Components

- Event
- Condition (tests whether the triggers should run)
  - A Boolean expression or a query
    - If the query answer set is non-empty it evaluates to true, otherwise false
    - If the condition is true the trigger action occurs
- Action

Trigger Components

- Event
- Condition
- Action (what happens if the trigger runs)
  - A trigger's action can be very far-ranging, e.g.
    - Execute queries
    - Make modifications to the DB
    - Create new tables
    - Call host-language procedures

Triggers

- Synchronization of the Trigger with the activating statement (DB modification)
  - Before
  - After
- Number of Activations of the Trigger
  - Once per modified tuple (FOR EACH ROW)
  - Once per activating statement (default).

Two kinds of triggers

- **Statement-level trigger**: executed once for all the tuples that are changed in one SQL statement.

```
REFERENCING  NEW TABLE AS  newtuples,   // Set of new tuples
             OLD TABLE AS   oldtuples     // Set of old tuples
```

- **Row-level trigger**: executed once for each modified tuple.

```
REFERENCING  OLD AS   oldtuple,
             NEW AS  newtuple
```

newtuples, *oldtuple*, *newtuple* can be used in the CONDITION and ACTION clauses

Triggers

- Options for the REFERENCING clause:
  - NEW TABLE: the set of tuples newly inserted (INSERT).
  - OLD TABLE: the set of deleted or old versions of tuples (DELETE / UPDATE).
  - OLD ROW: the old version of the tuple (FOR EACH ROW UPDATE).
  - NEW ROW: the new version of the tuple (FOR EACH ROW UPDATE).
- The action of a trigger can consist of multiple SQL statements, surrounded by BEGIN . . . END.

Triggers

```
CREATE TRIGGER youngSailorUpdate
   AFTER INSERT ON SAILORS                               /* Event */
   REFERENCING NEW TABLE NewSailors
   FOR EACH STATEMENT
      INSERT                                             /* Action */
         INTO YoungSailors(sid, name, age, rating)
         SELECT sid, name, age, rating
         FROM NewSailors N
         WHERE N.age <= 18;
```

- This trigger inserts young sailors into a separate table.
- It has no (i.e., an empty, always true) condition.

Triggers

```
CREATE TRIGGER notTooManyReservations
   AFTER INSERT ON Reserves                                      /* Event */
   REFERENCING NEW ROW NewReservation
   FOR EACH ROW
   WHEN (10 <= (SELECT COUNT(*)
                    FROM Reserves
                    WHERE sid =NewReservation.sid))              /* Condition */
          DELETE FROM Reserves R
          WHERE R.sid= NewReservation.sid                        /* Action */
                   AND day=
                   (SELECT MIN(day) FROM Reserves R2 WHERE R2.sid=R.sid);
```

- This trigger makes sure that a sailor has less than 10 reservations, deleting the oldest reservation of a given sailor, if neccesary.
- It has a non- empty condition (WHEN).

Triggers in Oracle

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
    {BEFORE|AFTER} {INSERT|DELETE|UPDATE} ON <table_name>
    [REFERENCING [NEW AS <new_row_name>] [OLD AS <old_row_name>]]
    [FOR EACH ROW [WHEN (<trigger_condition>)]]
    <trigger_body>
```

Create a trigger that checks whether a new tuple inserted into T4 has the first attribute <= 10. If so, insert the *reverse* tuple into T5.

```
CREATE TABLE T4 (a INTEGER, b CHAR(10));
CREATE TABLE T5 (c CHAR(10), d INTEGER);

CREATE TRIGGER trig1
        AFTER INSERT ON T4
        REFERENCING NEW AS newRow
        FOR EACH ROW
        WHEN (newRow.a <= 10)
        BEGIN
            INSERT INTO T5 VALUES(:newRow.b, :newRow.a);
        END trig1;
```

Examples from http://infolab.stanford.edu/~ullman/fcdb/oracle/or-triggers.html