**Database Systems I**

**SQL Constraints and Triggers**

---

**Integrity Constraints**

- An *integrity constraint* (IC) describes conditions that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC’s are disallowed.
  - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200).

- **Types of IC’s:**
  - domain constraints and NOT NULL constraints,
  - primary key constraints and foreign key constraints,
  - general constraints.
**Not-Null Constraints**

- The IC **NOT NULL** disallows NULL values for a specified attribute.

```
CREATE TABLE Students
    (sid CHAR(20) PRIMARY KEY,
     name CHAR(20) NOT NULL,
     login CHAR(10) NOT NULL,
     age INTEGER,
     gpa REAL);
```

What about specifying gpa as NOT NULL?
- Primary key attributes are implicitly NOT NULL.

---

**General Constraints**

- Attribute-based CHECK
  - defined in the declaration of an attribute,
  - activated on insertion to the corresponding table or update of attribute.
- Tuple-based CHECK
  - defined in the declaration of a table,
  - activated on insertion to the corresponding table or update of tuple.
- Assertion
  - defined independently from any table,
  - activated on any modification of any table mentioned in the assertion.
**General Constraints**

- Can use general SQL queries to express constraints.
- Much more powerful than domain and key constraints.
- Constraints can be named.
- Test of constraints can be deferred until the end of the corresponding transaction of the activating SQL statement.

**Attribute-based CHECK**

- Attribute-based CHECK constraint is part of an attribute definition.
- Is checked whenever a tuple gets a new value for that attribute (INSERT or UPDATE). Violating modifications are rejected.
- CHECK constraint can contain an SQL query referencing other attributes (of the same or other tables), if their relations are mentioned in the FROM clause.
- CHECK constraint is not activated if other attributes mentioned get new values.
Attribute-based CHECK

- Attribute-based CHECK constraints are most often used to restrict allowable attribute values.

```
CREATE TABLE Sailors
  ( sid INTEGER PRIMARY KEY,  
    sname CHAR(10),  
    rating INTEGER  
      CHECK ( rating >= 1  
          AND rating <= 10),  
    age REAL);
```

Tuple-based CHECK

- Tuple-based CHECK constraints can be used to constrain multiple attribute values within a table.
- Condition can be anything that can appear in a WHERE clause.
- Same activation and enforcement rules as for attribute-based CHECK.

```
CREATE TABLE Sailors
  ( sid INTEGER PRIMARY KEY,  
    sname CHAR(10),  
    previousRating INTEGER,  
    currentRating INTEGER,  
    age REAL,  
    CHECK (currentRating >= previousRating) )
```
**Tuple-based CHECK**

- CHECK constraint that refers to other table:
  ```
  CREATE TABLE Reserves
  ( sname CHAR(10),
  bid INTEGER,
  day DATE,
  PRIMARY KEY (bid,day),
  CHECK ('Interlake' <>
  ( SELECT B.bname
  FROM Boats B
  WHERE B.bid=bid)));
  ```
- But: these constraints are *invisible* to other tables, i.e. are not checked upon modification of other tables.
- What happens if the name of a boat is updated?

**Assertions**

- *Number of boats plus number of sailors is < 100.*
- Tuple-based CHECK awkward and wrong!
- If Sailors is empty, the number of Boats tuples can be anything!
  ```
  CREATE TABLE Sailors
  ( sid INTEGER,
  sname CHAR(10),
  rating INTEGER,
  age REAL,
  PRIMARY KEY (sid),
  CHECK
  ( (SELECT COUNT (S.sid) FROM Sailors S)
  + (SELECT COUNT (B.bid) FROM Boats B) < 100 )
  );
  ```
**Assertions**

- **ASSERTION** is the right solution; not associated with either table.
- Condition can be anything allowed in a WHERE clause.
- Constraint is tested whenever any (!) of the referenced tables is modified.
- Violating modifications are rejected.
- Different from CHECK constraints, ICs expressed as assertion are always enforced (unless they are deferred until the end of the transaction).
- CHECK constraints are more efficient to implement than ASSERTIONS.

---

**Assertions**

- **Number of boats plus number of sailors is < 100.**
  
  CREATE ASSERTION smallClub
  
  CHECK
  
  ( (SELECT COUNT (S.sid) FROM Sailors S)
  
  + (SELECT COUNT (B.bid) FROM Boats B) < 100 )

- **Number of reservations per sailor is < 10.**
  
  CREATE ASSERTION notTooManyReservations
  
  CHECK ( 10 > ALL
  
  (SELECT COUNT (*)
  
  FROM Reserves
  
  GROUP BY sid)

  );
Triggers

- **Trigger**: procedure that starts automatically if specified changes occur to the DB.

- Three parts of a trigger:
  - **Event** (activates the trigger)
    insert, delete or update of the database.
  - **Condition** (tests whether the trigger should run)
    a Boolean statement or a query (nonempty answer set = true, empty answer set = false).
  - **Action** (what happens if the trigger runs)
    wide variety of options.

Triggers

- Synchronization of the Trigger with the activating statement (DB modification)
  - Before
  - After
  - Instead of
  - Deferred (at end of transaction).

- Number of Activations of the Trigger
  - Once per modified tuple
    (FOR EACH ROW)
  - Once per activating statement
    (default).
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.

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 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 necessary.
- It has a non-empty condition (**WHEN**).

**Triggers vs. General Constraints**

- Triggers can be harder to understand.
  - Several triggers can be activated by one SQL statement (arbitrary order!).
  - A trigger may activate other triggers (chain activation).
- Triggers are procedural.
  - Assertions react on any database modification, trigger only on specified event.
  - Trigger execution cannot be optimized by DBMS.
- Triggers have more applications than constraints.
  - Monitor integrity constraints,
  - Construct a log,
  - Gather database statistics, etc.
Summary

- SQL allows specification of rich integrity constraints (ICs): attribute-based, tuple-based CHECK and assertions (table-independent).
- CHECK constraints are activated only by modifications of the table they are based on, ASSERTIONs are activated by any modification that can possibly violate them.
- Choice of the most appropriate method for a particular IC is up to the DBA.
- Triggers respond to changes in the database. Can also be used to represent ICs.