DAMG 6210
Database Design

Nik Bear Brown
@NikBearBrown

SQL Constraints & Triggers

Topics

- Constraints
- Triggers
- Assertions

Constraints in SQL

- Constraints on attribute values:
  - these are checked whenever there is insertion to table or attribute update
  - not null constraint
  - attribute based check constraint
  - E.g., sex char(1) CHECK (sex IN (‘F’, ‘M’))
  - domain constraint
    - E.g., Create domain gender-domain CHAR (1) CHECK (VALUE IN (‘F’, ‘M’))
    - define sex in schema defn to be of type gender-domain

Integrity Constraints

- An 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, primary key constraints, foreign key constraints, general constraints.
  - *Domain constraints*: Field values must be of right type. Always enforced.

Constraints in SQL

- Constraints on tuples
- Tuple based CHECK constraint:

```
CREATE TABLE Gamer (
name CHAR(30) UNIQUE
gender CHAR(1) CHECK (gender in ('F', 'M')
age int
plat int
CHECK (age < 100 AND age > 20)
CHECK (plat IN (SELECT plat FROM platform))
)
```

these are checked on insertion to relation or tuple update

Keys: Fundamental Constraint

- In the CREATE TABLE statement, use:
  - PRIMARY KEY, UNIQUE

```
CREATE TABLE Gamer (
  name CHAR(30) PRIMARY KEY,
  address VARCHAR(255),
  gender CHAR(1));
```

- Or, list at end of CREATE TABLE

```
PRIMARY KEY (name)
```

Keys...

- Can use the UNIQUE keyword in same way
  - …but for any number of attributes
  - foreign keys, which reference attributes of a second relation, only reference PRIMARY KEY
- Indexing Keys

```
CREATE UNIQUE INDEX UserIndex ON
  Twitter_User(screen_name)
```

- Makes insertions easier to check for key constraints

Referential Integrity Constraints

- 2 rules for Foreign Keys:

  Movies(MovieName, year)

  ActedIn(ActorName, MovieName)

1) Foreign Key must be a reference to a valid value in the referenced table.

2) … must be a PRIMARY KEY in the referenced table.

Declaring FK Constraints

- FOREIGN KEY <attributes> REFERENCES <table> (<attributes>)

```
CREATE TABLE ActedIn (
  Name CHAR(30) PRIMARY KEY,
  MovieName CHAR(30)
    REFERENCES Movies(MovieName));
```

- Or, summarize at end of CREATE TABLE

```
FOREIGN KEY MovieName REFERENCES Movies(MovieName)
```

- MovieName must be a PRIMARY KEY

Declaring FK Constraints

- FOREIGN KEY <attributes> REFERENCES <table> (<attributes>)

```
CREATE TABLE ActedIn (
  Name CHAR(30) PRIMARY KEY,
  MovieName CHAR(30)
    REFERENCES Movies(MovieName));
```

- Or, summarize at end of CREATE TABLE

```
FOREIGN KEY MovieName REFERENCES Movies(MovieName)
```

- MovieName must be a PRIMARY KEY

Declaring FK Constraints

- ```
  CREATE TABLE IF NOT EXISTS `Exam1_Twitter_Tweets` (
  `tweet_id` bigint(20) unsigned NOT NULL
  AUTO_INCREMENT,  `from_user_id` bigint(20) unsigned
  NOT NULL DEFAULT '0',  `tweet` varchar(255) DEFAULT
  NULL,  `geo` varchar(255) NOT NULL,  `created_at`
  datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  PRIMARY KEY (`tweet_id`),  KEY `tweet` (`tweet`))
  ENGINE=MyISAM  DEFAULT CHARSET=utf8
  AUTO_INCREMENT=29 ;
  ```
- ```
  ALTER TABLE Exam1_Twitter_Tweets ADD CONSTRAINT
  Tweets_User_Id FOREIGN KEY (from_user_id) REFERENCES
  Exam1_Twitter_Users (user_id);
  ```

Constraining Attribute Values

Constrain invalid values

NOT NULL

gender CHAR(1)

CHECK (gender IN (‘F’, ‘M’))

GameName CHAR(30)

CHECK (GameName IN

(SELECT GameName FROM Games))

Last one not the same as REFERENCE

The check is invisible to the Games table!

Constraining Values with User Defined ‘Types’

- Can define new domains to use as the attribute type...

  CREATE DOMAIN GenderDomain CHAR(1)

    CHECK (VALUE IN (‘F’, ‘M’));

- Then update our attribute definition...

  gender GenderDomain

More Complex Constraints…

- …Among several attributes in one table
  - Specify at the end of CREATE TABLE

CHECK (gender = ‘F’ OR name NOT LIKE ‘Ms.%’)

Giving Names to Constraints

Why give names? In order to be able to alter constraints.

Add the keyword CONSTRAINT and then a name:

```
ssn CHAR(50) CONSTRAINT  ssnIsKey PRIMARY KEY

CREATE DOMAIN ssnDomain  INT
  CONSTRAINT  ninedigits CHECK  (VALUE >= 100000000
                                 AND VALUE <= 999999999

CONSTRAINT  rightage
    CHECK  (age >= 0  OR status = "dead")
```

Altering Constraints

```
ALTER TABLE  Product  DROP CONSTRAINT positivePrice

ALTER TABLE  Product ADD CONSTRAINT
     positivePrice  CHECK  (price >= 0)

ALTER DOMAIN  ssn ADD CONSTRAINT no-leading-1s
   CHECK  (value >= 200000000)

DROP ASSERTION  assert1.
```

Integrity Constraints (Review)

- An 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, primary key constraints, foreign key constraints, general constraints.
  - *Domain constraints*: Field values must be of right type. Always enforced.
  - *Primary key and foreign key constraints*: you know them.

Triggers (Active database)

- Trigger: A procedure that starts automatically if specified changes occur to the DBMS
- Analog to a "daemon" that monitors a database for certain events to occur
- Three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run) [Optional]
  - Action (what happens if the trigger runs)
- Semantics:
  - When event occurs, and condition is satisfied, the action is performed.

Triggers – Event,Condition,Action

- Events could be :

  BEFORE|AFTER INSERT|UPDATE|DELETE ON <tableName>

  e.g.: BEFORE INSERT ON Professor

- Condition is SQL expression or even an SQL query (query with non-empty result means TRUE)

- Action can be many different choices :
  - SQL statements , body of PSM, and even DDL and transaction-oriented statements like “commit”.

Trigger Syntax

```
CREATE TRIGGER <triggerName>
BEFORE|AFTER   INSERT|DELETE|UPDATE
  [OF <columnList>] ON <tableName>|<viewName>
  [REFERENCING [OLD AS <oldName>] [NEW AS <newName>]]
[FOR EACH ROW] (default is “FOR EACH STATEMENT”)
[WHEN (<condition>)]
<PSM body>;
```

Example Trigger

Assume our DB has a relation schema :

Professor (pNum, pName, salary)

We want to write a trigger that :

Ensures that any new professor inserted has salary <= 60000

Example Trigger

```
CREATE TRIGGER minSalary BEFORE INSERT ON Professor

    FOR EACH ROW

BEGIN

   IF (:new.salary >= 60000)
      THEN RAISE_APPLICATION_ERROR (-20004,          'Violation of Minimum
   Professor Salary');
   END IF;

END;
```

Example trigger

```
CREATE TRIGGER minSalary BEFORE INSERT ON Professor        FOR
  EACH ROW


DECLARE temp int;   -- dummy variable not needed


BEGIN
  IF (:new.salary >= 60000)
    THEN RAISE_APPLICATION_ERROR (-20004,          'Violation of
  Minimum Professor Salary');
  END IF;


temp := 10;      -- to illustrate declared variables


END;
.
```

Details of Trigger Example

- BEFORE INSERT ON Professor
  - This trigger is checked before the tuple is inserted
- FOR EACH ROW
  - specifies that trigger is performed for each row inserted
- :new
  - refers to the new tuple inserted
- If (:new.salary >= 60000)
  - then an application error is raised and hence the row is not inserted; otherwise the row is inserted.
- Use error code: -20004;
  - this is in the valid range

Row vs Statement Level Trigger

- Row level: activated once per modified tuple
- Statement level: activate once per SQL statement

- Row level triggers can access new data, statement level triggers cannot always do that (depends on DBMS).

- Statement level triggers will be more efficient if we do not need to make row-specific decisions

Row vs Statement Level Trigger

- Example: Consider a relation schema

  Account (num, amount)

  where we will allow creation of new accounts only during normal business hours.

Example: Statement level trigger

```
CREATE TRIGGER MYTRIG1
BEFORE INSERT ON Account
FOR EACH STATEMENT                    --- is default
BEGIN
   IF (TO_CHAR(SYSDATE,'dy') IN ('sat','sun'))
   OR
   (TO_CHAR(SYSDATE,'hh24:mi') NOT BETWEEN '08:00' AND '17:00')
    THEN
      RAISE_APPLICATION_ERROR(-20500,'Cannot   create new account now
  !!');
   END IF;
END;
```

When to use BEFORE/AFTER

- Based on efficiency considerations or semantics.

- Suppose we perform statement-level after insert, then all the rows are inserted first, then if the condition fails, and all the inserted rows must be “rolled back”

- Not very efficient !!

Summary : Trigger Syntax

```
CREATE TRIGGER <triggerName>
BEFORE|AFTER   INSERT|DELETE|UPDATE
  [OF <columnList>] ON <tableName>|<viewName>
  [REFERENCING [OLD AS <oldName>] [NEW AS <newName>]]
[FOR EACH ROW] (default is “FOR EACH STATEMENT”)
[WHEN (<condition>)]
<PSM body>;
```

Constraints versus Triggers

- Constraints are useful for database consistency
  - Use IC when sufficient
  - More opportunity for optimization
  - Not restricted into insert/delete/update

- Triggers are flexible and powerful
  - Alerters
  - Event logging for auditing
  - Security enforcement
  - Analysis of table accesses (statistics)
  - Workflow and business intelligence …

- But can be hard to understand ……
  - Several triggers (Arbitrary order unpredictable !?)
  - Chain triggers (When to stop ?)
  - Recursive triggers (Termination?)

Triggers

Enable the database programmer to specify:

- when to check a constraint,
- what exactly to do.

A trigger has 3 parts:

- An event (e.g., update to an attribute)
- A condition (e.g., a query to check)
- An action (deletion, update, insertion)

When the event happens, the system will check the constraint, and if satisfied, will perform the action.

NOTE: triggers may cause cascading effects.

Database vendors did not wait for standards with triggers!

Elements of Triggers

- Timing of action execution: before, after or instead of triggering event
- The action can refer to both the old and new state of the database.
- Update events may specify a particular column or set of columns.
- A condition is specified with a WHEN clause.
- The action can be performed either for
  - once for every tuple, or
  - once for all the tuples that are changed by the database operation.

Assertions

- Assertions are constraints over a table as a whole or multiple tables.
- General form:

  CREATE ASSERTION <name> CHECK <cond>
- An assertion must always be true at transaction boundaries. Any modification that causes it to become false is rejected.
- Similar to tables, assertions can be dropped by a DROP command.

Example Assertion

- CREATE ASSERTION RichProf CHECK
  (NOT EXISTS
  (SELECT *
  FROM dept, emp
  WHERE emp.name = dept.mgrname AND
  emp.salary < 50000))

- This assertion correctly guarantees that each professor makes more than 50000.
- If someone made a professor whose salary is less than 50K that insertion/update to dept table will be rejected.

Declaring Assertions

- CREATE ASSERTION <name> CHECK (<condition>)

```
CREATE ASSERTION RichPres CHECK
  (NOT EXISTS
    (SELECT *
     FROM Studio, MovieExec
     WHERE presC# = cert#
         AND netWorth < 10000000))
```