CS157A: Introduction to DataBase Management Systems

Chapter 7. Constraints and Triggers

Suneuy Kim

Constraints and Triggers

- A constraint describe allowable database states.

Example: Key constraints, referential integrity constraints (also called foreign key constraints)

- A trigger checks conditions when database is changed (by insert, delete, update) and takes an action when it is triggered.

[Q] Who is going to check the correctness of any update command ? Application or DBMS ?

[A] It’s better to save checks with database so that DBMS administer them.

[Because]

- Checks won’t be forgotten
- Can avoid duplication of work (modular)

Kinds of Constraints

1. Non-null
2. Key constraints
3. Referential integrity constraints (foreign key)
4. Attribute-based constraint

   Constrain values of a particular attribute.
5. Tuple-based constraint

   Relationship among components
6. General assertions

Non null constraint

```
CREATE TABLE USER
(uID INT,
 uNAME VARCHAR(30),
 age INT not null,
 loaned INT,
 PRIMARY KEY (uID)
);
```

Key Constraints

```
CREATE TABLE USER
(uID INT;
 uNAME VARCHAR(30),
 age INT,
 loaned INT,
 PRIMARY KEY (uID)
);
```

Referential Integrity Constraints (Foreign key constraints)

- There should not be any dangling pointers
- Referential integrity from R.A to S. B
  - The attribute B must be the PRIMARY KEY or UNIQUE in relation S.
  - Each value in column A of relation R must appear in column B of relation S.
- R.A $\rightarrow$ S.B does not mean S.B$\rightarrow$ R.A

Referential Integrity Constraints Declaration with Attributes

```
CREATE TABLE LOAN
(uID INT,
 title VARCHAR(50) REFERENCES Book(title),
 loanDate DATE DEFAULT DATE '0000-00-00',
 overdue BOOLEAN DEFAULT FALSE,
 PRIMARY KEY(uID,title,loanDate)
);
```

Referential Integrity Constraints Declaration as Schema Element

```
CREATE TABLE LOAN
(uID INT,
 title VARCHAR(50),
 loanDate DATE DEFAULT DATE '0000-00-00',
 overdue BOOLEAN DEFAULT FALSE,
 PRIMARY KEY(uID,title,loanDate),
 FOREIGN KEY(title) REFERENCES Book(title)
);
```

Null in a foreign key

Does not required to check if the existence of any value in the referenced column.

Example:

```
insert into Loan(uID, overdue)
values(777, false);
```

will set the title of the Loan to Null and this change avoids the constraint violation.

Enforcing foreign key constraints

Consider Loan.title → Book.title

Possible violation cases

- Case 1: inserting a Loan tuple of which title is not null and is not the title of any Book tuple.
- Case 2: updating a Loan tuple with a title which is not null and is not the title of any Book tuple.
- Case 3: deleting a Book tuple of which non-Null title appears as the title of a Loan tuple.
- Case 4: updating a Book tuple with a new title and the old title is the title of a Loan tuple.

Enforcing foreign key constraints

- Cases 1 and 2: Simply reject it !
- Cases 3 and 4: when a change in the referenced relation affects a foreign key value → It is possible for DBMS to modify it in away that doesn’t violate the constraint
  - The Default Policy : Reject violating modifications
  - The Cascade Policy: Make the same change in R.A
    - Delete a Book with title ‘Bambi’ → delete Loans with title ‘Bambi’
    - Update the title Bambi with Bambi II in a Book relation → update the titles of Loans whose title is ‘Bambi’ with ‘Bambi II’.
  - The Set-Null Policy: Set the title of involved Loans 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]

- Otherwise, the default (reject) is used

Example

```
CREATE TABLE LOAN
(uID INT,
 title VARCHAR(50),
 loanDate DATE DEFAULT DATE '0000-00-00',
 overdue BOOLEAN DEFAULT FALSE,
 PRIMARY KEY(uID,title,loanDate),
 FOREIGN KEY(title) REFERENCES Book(title)
 ON DELETE SET NULL
 ON UPDATE CASCADE
);
```

Circular Constraints

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

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

→ Error ! Why ?

Way around

```
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);

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

However, you can’t insert any tuple to these tables!

```
insert chicken values (1,2); will fail!
insert chicken values (2,1); will fail!
```

Way around

```
insert chicken values (1,null);
insert egg values(2, null);
update chicken
    set eID = 2
    where cID =1;
update egg
   set cID = 1
   where eID = 2;
```

Deferred Constraints (Oracle)

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

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

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

Deferred Constraints

```
INSERT INTO chicken VALUES(1, 2);
INSERT INTO egg VALUES(2, 1);
COMMIT;
```

The foreign key constraints are declared as "deferred“ and only checked at the commit point.

Deferred Constraint Options

- NOT DEFERRABLE: The constraint will be checked immediately after each statement.
- DEFERRABLE INITALLY DEFERRED: The constraint check will be deferred until the commit point.
- DEFERRABLE INITALLY IMMEDIATE: The constraint will be checked immediately after each statement
- You can change DEFERRED to IMMEDIATE and vice versa using SET CONSTRAINT command.

```
SET CONSTRAINT chickenREFegg DEFERRED;
```

Deferred Constraints

- To drop the tables with foreign key constraints, we have to drop the constraints first.

```
ALTER TABLE egg DROP FOREIGN KEY eggREFchicken;
ALTER TABLE chicken DROP FOREIGN KEY
                                  chickenREFegg;
DROP TABLE egg;
DROP TABLE chicken;
```

MySQL doesn’t support deferred constraint checking.

Attribute-Based Checks

- Constraints on the value of a particular attribute.
- Add CHECK(condition) to the declaration for the attribute. The condition is anything that can appear in WHERE clause in SQL.
- The condition may use the name of the attribute being constrained.
- If the condition refers to any other relations or attributes of relations , the relation must be introduced in the FROM clause of a subquery.
- Checked if any tuple gets a new value for this attribute by insert or update.

Example

```
CREATE TABLE USER
(uID INT,
 uNAME VARCHAR(30),
 age INT CHECK (age >=10),
 loaned INT,
 PRIMARY KEY (uID)
);
```

Example: Erroneous attempt to simulate foreign key constraint

```
CREATE TABLE LOAN
(uID INT,
 title VARCHAR(50) CHECK (title IN(SELECT
title  from BOOK)),
 loanDate DATE DEFAULT DATE '0000-00-00',
 overdue BOOLEAN DEFAULT FALSE,
 PRIMARY KEY(uID,title,loanDate));

insert into LOAN values (123,'Web Server
Programming',CURRENT_DATE(), false);
```

Timing of Checks

- Important: an attribute-based constraint is checked only when the constrained attribute is updated.
- Example: CHECK (age >= 10)

  checks every new age and rejects the modification (for that tuple) if the age is less than 10.
- Example: CHECK (title IN (SELECT title from BOOK))

  not checked if a title is deleted from Book

  (erroneous attempt to simulate the foreign-key constraint).

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 relations or attributes of relations require a subquery.
- Checked for the new or updated tuple.
- Use De Morgan’s law to find the condition that violates the check constraint.

Example

```
CREATE TABLE LOAN
(uID INT,
 title VARCHAR(50),
 loanDate DATE DEFAULT '0000-00-00',
 overdue BOOLEAN DEFAULT FALSE,
 PRIMARY KEY(uID,title,loanDate),
 CHECK (uID <> 123 or title <>'Bambi')
);
```

Example: Subquery in Check

```
CREATE TABLE LOAN
(uID INT,
title VARCHAR(50),
loanDate DATE DEFAULT '0000-00-00',
overdue BOOLEAN DEFAULT FALSE,
PRIMARY KEY(uID,title,loanDate),
CHECK (title IN (SELECT title from Book)));
```

Note: Although a change in Book causes the condition to be false, the check can’t inhibit the change.

Attribute-based vs. Tuple-based Constrains

- If more than one attributes are involved in a constraint, use tuple-based constraints.
- If one attribute is involved, use either tuple- or attribute-based constraint; The condition checked is the same, but tuple based constraint will be checked more frequently since it is checked whenever any attribute of the tuple is updated.

MySQL

- In MySQL, check is accepted, but not enforced.

Assertion

- Interrelation constraints
- These are database-schema elements, like relations or views.
- Defined by:

  CREATE ASSERTION name CHECK (condition);
- We name it so that we can delete the assertion by name.
- Condition may refer to any relation or attribute in the database schema.
- The assertion must be always true for the entire database.

Example: Assertion

CREATE ASSERTION ReferentialIntegrity

CHECK (not exists (select * from Loan where uID not in (select uID from User));

Note: It is very common to write a condition in a negative form and use not exists.

Example: Assertion

Suppose there cannot be more number of users than the total number of copies of books in the library.

```
CREATE ASSERTION FewUser CHECK (
  (select count(*) from User) <=
  (select sum(copies) from Book)
);
```

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 Loan can affect FewUser. Neither can an insertion to Books.

MySQL: Assertion

- No SQL implementation supports Assertion yet.

Triggers

"Event-Condition-Action Rules"- When event occurs, check condition, if true, take an action.

Event: data base modification, e.g., insert

Condition: Any SQL boolean-valued expression.

Action: Any SQL statements

Motivation: Triggers

- To move logic from application into DB
- To enforce integrity constraints beyond what constraint system supports – sometimes constraint system is limited. Triggers can be more expressive.
- Automatic constraint “repair” by specifying repair in the action part.

Triggers

```
CREATE TRIGGER name
BEFORE|AFTER|INSTEAD OF events  ON R
[referencing-variables]
[FOR EACH ROW| FOR EACH STATEMENT]
When (condition)
Action
```

Trigger Options

- [FOR EACH ROW]

  The trigger is activated at row level for each tuple affected by the event.
- [FOR EACH STATEMENT]
  - The trigger is activated at statement level.
- Example: Suppose a delete statement deletes 10 tuples.
  - With for each row option, trigger is activated 10 times: one for each deleted tuple
  - With for each statement, trigger is activated once for the delete statement.

Trigger Options

- [REFERENCING variable]

OLD ROW AS|NEW ROW AS|OLD TABLE AS|NEW TABLE AS `var`

- Depending on the event
  - Insert: only NEW
  - Delete: only OLD
  - Update: both OLD and NEW

Trigger Options

- Old row in delete means specific deleted row
- Old table in delete means all deleted tuples, not referring old state of data base.
- If the trigger is FOR EACH STATEMENT, row level variable (OLD ROW AS or NEW ROW AS) is not available. Only table-level of variable is available.
- Both row-level and statement-level triggers can use table-level variables (OLD TABLE AS or NEW TABLE AS)

Example: Triggers

To fail any attempt to lower the net worth of a movie executive.

```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec
REFERENCING
     OLD ROW AS OldTuple
     NEW ROW AS NewTuple
FOR EACH ROW
WHEN (OldTuple.netWorth > NewTuple.netWorth)
  UPDATE MovieExec
  SET netWorth = OldTuple.netWorth
  WHERE cert# = NewTuple.cert#;
```

Trigger Time: Before vs. After

- After trigger is more common.
- In a BEFORE trigger, you can change the NEW value with SET newtuple.*col_name* = *value* .
- Such a SET statement has no effect in an AFTER trigger because the row change will have already occurred
- A column named with OLD is read only.

Example: Before SQL Trigger (not MySQL)

```
CREATE TRIGGER FixYearTrigger
BEFORE INSERT ON Movies
REFERENCING
   NEW ROW AS NewRow
   NEW TABLE AS NewStuff
FOR EACH ROW
WHEN NewRow.year IS NULL
UPDATE NewStuff SET year = 1915;
```

NOTE: NewStuff is a relation consisting of only the new row being inserted. We need a relation to write update statement on

Example: Before SQL Trigger (not MySQL)

```
CREATE TRIGGER TransactionBeforeTrigger
BEFORE INSERT ON TransactionTable
REFERENCING NEW AS new_row
FOR EACH ROW
BEGIN
 DECLARE newmonth SMALLINT;
 SET newmonth=MONTH(new_row.DateOfTransaction);
 IF newmonth < 4 THEN  SET new_row.FiscalQuarter=3;
 ELSEIF newmonth < 7 THEN SET new_row.FiscalQuarter=4;
 ELSEIF newmonth < 10 THEN SET new_row.FiscalQuarter=1;
 ELSE SET new_row.FiscalQuarter=2;
 END IF;
END
```

Before SQL Trigger

```
INSERT INTO
TransactionTable(DateOfTransaction)
VALUES(CURRENT DATE) ;
```

For the SQL insert statement above, the "FiscalQuarter" column is set to 1, if the current date is September 24, 2013.

MySQL Triggers

```
CREATE TRIGGER trigger_name
BEFORE|AFTER   INSERT|DELETE|UPDATE
ON table_name
FOR EACH ROW
BEGIN ... END
```

MySQL Triggers

Notes:

- A trigger only can be invoked by one event.
- A trigger is immediately activated when the event occurs.
- There cannot be multiple triggers for a given table that have the same trigger event and action time. For example, two BEFORE UPDATE triggers for a table are not allowed.

MySQL Triggers

- To work around this, you can define a trigger that executes multiple statements by using the BEGIN … END compound statement.

Example: Triggers

```
CREATE TRIGGER T1
AFTER INSERT ON User
FOR EACH ROW
WHEN NEW.age > 10 and NEW.age <= 50
BEGIN
insert into Loan values (New.uID, ‘Bambi’, false);
END;
```

Example: MySQL version

```
DROP TRIGGER IF EXISTS InsertTrigger;
delimiter //
CREATE TRIGGER InsertTrigger
AFTER INSERT ON User
FOR EACH ROW
  BEGIN
  IF NEW.age > 10 and NEW.age <= 50 THEN
    insert into Loan values (New.uID, 'Bambi', 
CURRENT_DATE(), false);
  END IF;
END;
//
delimiter ;
```

Example: Triggers

Create TRIGGER T2

After delete on User

For each row

Begin

 delete from Loan where uID = Old.sID

End;

Example: MySQL version

```
DROP TRIGGER IF EXISTS DeleteCascadeTrigger;
delimiter //
CREATE TRIGGER DeleteCascadeTrigger
AFTER DELETE ON User FOR EACH ROW
BEGIN
  delete from Loan where uID =Old.uID;
END;//
delimiter ;
```

Example: Triggers

```
CREATE Trigger T3
After update of title on Book
For each row
Begin
 update Loan
 set title = New.title;
 where title = Old.title;
End;
```

Example: MySQL version

```
DROP TRIGGER IF EXISTS UpdateTrigger;
delimiter //
CREATE TRIGGER UpdateTrigger
AFTER UPDATE ON Book
FOR EACH ROW
  BEGIN
    UPDATE Loan SET title = NEW.title
    WHERE title = OLD.title;
  END//
delimiter ;
```