

CptS 451- Introduction to Database Systems

SQL : Constraints and Triggers (DMS Ch-5.7)

Instructor: Sakire Arslan Ay



Topics

- Database Modifications in SQL
 - INSERT, DELETE, UPDATE
- Constraints in SQL
- Triggers in SQL

Database Modifications

- A modification command does not return a result (as a select query does), but changes the database in some way.
- Three kinds of modifications:
 - Insert a tuple or tuples.
 - Delete a tuple or tuples.
 - Update the value(s) of an existing tuple or tuples.

Insertion of a Tuple

- To insert a single tuple:

**INSERT INTO relation
VALUES (list of values);**

- Example:

```
INSERT INTO Emp (ename, dno, sal)
VALUES ('Tom', 123, 45000)
```

- Can drop attribute names if we provide all values in order.

```
INSERT INTO Emp
VALUES ('Tom', 123, 45000)
```

- If we don't provide values for all attributes, they will be filled with NULL.

```
INSERT INTO Emp (ename,sal) ← List the attributes that we provide
VALUES ('Tom', 45000) ← values for
                        The values for those attributes
```

- ✓ The tuple ('Tom',NULL,45000) will be inserted to Emp.
 - Dno will be assigned NULL

Insertion of a Tuple - Default Values

- In a CREATE TABLE statement, we can define a DEFAULT value for an attribute.
- When an inserted tuple has no value for that attribute, the default will be used.
- **Example:**

```
CREATE TABLE Emp(  
    ename CHAR(30) PRIMARY KEY,  
    dno INT DEFAULT 2752,  
    sal FLOAT );
```

```
INSERT INTO Emp (ename,sal)  
VALUES ('Tom', 45000)
```

- ✓ The tuple ('Tom',2752,45000) will be inserted to Emp.
–dno will be assigned 2752

Insertion of a Query's Result

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

INSERT INTO relation
(subquery);



The result of the subquery
should have the same set of
attributes as “relation”

Insertion of a Query's Result – Example

```
CREATE TABLE LowIncomeEmp (  
    ename VARCHAR(12),  
    dno INT,  
    sal FLOAT);
```

```
INSERT INTO LowIncomeEmp  
    ( SELECT *  
      FROM emp  
      WHERE sal <= 30K AND dno = 123; );
```

```
INSERT INTO LowIncomeEmp  
    ( SELECT ename, dno, sal * 1.1 → salary increased by 10%  
      FROM emp  
      WHERE sal <= 30K AND dno = 123);
```

Insert is
performed
next

subquery is
executed first

- Note the order of querying and inserting: first subquery is executed, then insertion is performed.

Insertion of a Query's Result – Additional Examples

Emp(ename, dno, salary)

Dept(dno, dname, mgr)

Emp(dno) is a FK referencing Dept(dno)

Dept(mgr) is a FK referencing Emp(ename)

- **Query:** “Find the department dno which has the minimum average salary.”

Deletion

- To delete tuples satisfying a condition from some relation:

DELETE FROM relation
[WHERE condition];

← WHERE clause is optional

```
DELETE
FROM emp
WHERE dno = 2752;
```

```
DELETE
FROM emp;
```

← All tuples in relation
“emp” will be deleted.

- There is no way to delete only a single occurrence of a tuple that appears twice in a relation.

Delete (cont)



Emp(ename, dno, sal),
Dept(dno, dname, mgr)

- **Example:** Delete all employees working in a department with only one employee.

```
DELETE FROM emp AS E1
```

Delete is
performed
next

```
WHERE NOT EXISTS
```

```
(SELECT ename  
FROM emp  
WHERE dno = E1.dno AND ename <> E1.ename);
```

Note the relation renaming “E1”

subquery is
executed first

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 assignments

WHERE condition;

- **Example-1:** “Change employees in dept 2752 to dept 2755.”

```
UPDATE emp
SET dno = 2755
WHERE dno = 2752;
```

Updates (cont.)

- **Example-2:** “Cut the salaries that are more than 100K by 10%.”

```
UPDATE emp  
SET sal = sal * 0.9  
WHERE sal > 100000;
```

- **Example-3:** “Change employees in dept 2752 to dept 2755 and increase their salaries by 10%.”
 - Multiple assignments separated by “,”

```
UPDATE emp  
SET dno = 2755, sal = sal * 1.1  
WHERE dno = 2752;
```

Next: SQL Constraints



Data Modifications and Constraints - Revisited



- Modification, insertion, and deletion requests can lead to violations of one or more integrity constraints
- DBMS must check for violations for each operation and suitably allow or disallow the operation

Kinds of Constraints

- So far, we have seen:
 - Primary keys, UNIQUE
 - Foreign-key, or referential-integrity.
 - How to define policies in SQL DDL to handle foreign key violations
 - CHECK clause

Kinds of Constraints

- Now we will review these constraints and elaborate more on how to define policies in SQL DDL to handle violations due to an insertion, deletion, or update:
 - ✓ (1) Keys, primary keys.
 - ✓ (2) Foreign-key, or referential-integrity.
 - (3) Attribute-based checks.
 - Constrain values of a particular attribute. (CHECK on a single attribute)
 - (4) Tuple-based checks.
 - Constraints on any attribute of the relation
 - (5) Assertions: any SQL boolean expression.
 - (6) Triggers

(1) Primary Keys, UNIQUE (Review)

```
CREATE TABLE emp
( ssn      integer Primary Key,
  name     char(15),
  dno      integer,
);
```

```
CREATE TABLE emp
( ssn      integer,
  name     char(15),
  dno      integer,
  Primary Key (dno,name)
);
```

```
CREATE TABLE emp
( SSN      integer UNIQUE,
  name     char(15),
  dno      integer,
);
```

```
CREATE TABLE emp
( SSN      integer,
  name     char(15),
  dno      integer,
  UNIQUE (dno,name)
);
```

Enforcing Key Constraints (Review)

- Check constraint each time the table is modified
 - Insertion: check
 - Update: check
 - Deletion: do not check
- Enforcing key constraints efficiently:
 - Suppose “ssn” is a primary key of Emp.
 - Every time we insert a new employee, do we want to scan the whole table to check if the ssn already exists? No!
 - Using index on the key attribute(s).

| ename | dno | sal |
|-------|-----|-----|
| Jack | 111 | 81K |
| Alice | 111 | 70K |
| Lisa | 222 | 32K |
| Tom | 333 | 56K |
| Mary | 333 | 65K |

(2) Foreign-Key Constraint (Review)

```
CREATE TABLE emp
( SSN integer,
  name char(15),
  dno integer REFERENCES dept(dept#),
);
```

```
CREATE TABLE emp
( ssn integer,
  name char(15),
  dno integer,
  FOREIGN KEY (dno) REFERENCES
    dept(dept#));
```

Enforcing foreign-Key Constraints

If there is a foreign-key constraint from relation *emp* to relation *dept*, two violations are possible:

1. An insert or update to *emp* introduces values not found in *dept*.
2. A deletion or update to *dept* causes some tuples of *emp* to “dangle.”

(2) Foreign Key Constraint – Choosing a Policy (Review)

- Add “**ON [DELETE,UPDATE] [CASCADE, SET NULL]**” when declaring a foreign key
 - If no policy is specified, the default (**REJECT**) is used.
- Which policy to choose depends on the application.

```
CREATE TABLE emp
( ssn integer,
  name varchar(20),
  dno integer,
  FOREIGN KEY dno REFERENCES dept(dept#)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);
```

(3) Attribute-Based Checks

```
CREATE TABLE Emp (  
    name varchar(30),  
    dno integer,  
    gender varchar(3) CHECK (gender IN ('F', 'M', 'N/A')),  
    age integer CHECK (age > 18 AND age < 100)  
);
```

- Syntax: **CHECK (condition)**
- Constraints on attribute values.
- Checked when there is an **insertion** or **update** of the attribute.

[back](#)

Attribute-Based Checks (cont.)

- Condition may involve the checked attribute
- Other attributes and relations may be used (in a query)
 - Subqueries in CHECK statements are not allowed in PostgreSQL
- Condition checked only when that associated attribute changes

```
CREATE TABLE Emp (  
    ssn char(9),  
    name varchar(30),  
    dno integer CHECK (dno in (SELECT dno from dept))  
);
```

- Condition checked when we insert/update Emp.dno, but **NOT when we modify dept (unlike foreign-keys).**

Tuple-Based Checks

```
CREATE TABLE Emp (  
    ssn    CHAR(9),  
    age    INTEGER,  
    dno    INTEGER,  
    startdate DATE,  
    enddate DATE,  
    CHECK (age > 18 AND age < 100),  
    CHECK (enddate IS NULL OR startdate < enddate),  
    CHECK (( SELECT COUNT(E.ssn) FROM Emp E) < 1000)  
);
```

- Checked whenever a tuple is inserted or updated
- Again, other relations may be used.
 - (subqueries in CHECK statements are not supported by PostgreSQL) [back](#)

(Review) Modification of Constraints

- It is possible to add, modify, or delete constraints on tables
- **Giving Names to Constraints:**
 - In order to modify constraints later, they should be given names
 - Use keyword **CONSTRAINT** to name constraints
 - **Examples:**

1. gender CHAR(3) **CONSTRAINT** validGender CHECK (gender IN ('F','M','N/A'));
2. Ssn CHAR(9) **CONSTRAINT** SSNisKey PRIMARY KEY;
3. **CONSTRAINT** FK1 FOREIGN KEY (dno) REFERENCES dept(dept#);
4. **CONSTRAINT** CheckSal CHECK (salary>20K AND salary<300K);

Attribute-based check

Tuple-based check

(Review) Modification of Constraints (cont.)



- **Altering Constraints on Tables:**

- Use ALTER TABLE clause to add and drop named constraints

- Drop constraint:

```
ALTER TABLE emp DROP CONSTRAINT CheckSal;  
ALTER TABLE emp DROP CONSTRAINT FK1;
```

- Add constraint:

```
ALTER TABLE emp ADD CONSTRAINT CheckSal2  
    CHECK (salary>20K AND salary<300K);  
ALTER TABLE emp ADD CONSTRAINT FK2  
    FOREIGN KEY (dno) REFERENCES department(deptNo));
```

All tuple-based checks

(Review) ALTER TABLE - add/drop attributes



- Used to add/drop attributes from a relation
- Add attribute:

ALTER TABLE relationName **ADD** attribName
attribDomain

- All tuples in the relation are assigned *null as the default value of the* new attribute

- Drop attribute:

ALTER TABLE relationName **DROP** attribName

- Dropping of attributes not supported by many DBMSs

Assertions

- So far most constraints are within one table
 - Attribute-based checks
 - Tuple-based checks
 - Assertions:
 - Constraints over a table as a whole or multiple tables.
 - It is a boolean valued SQL expression that must be true at all times
- 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.
 - **DROP ASSERTION** assert1;

Assertions - Example

Dept(dept#, mgr), Emp(name, salary)

- “There can be maximum 1000 employees.”

```
CREATE ASSERTION empCount CHECK  
  (1000 > (SELECT COUNT(*)  
           FROM emp));
```

- If someone **inserts** a new employee and if there are 1000 employees in table Emp, the insertion will be rejected.

CREATE ASSERTION is not yet implemented in PostgreSQL

Assertions - Example

Dept(dept#, mgr), Emp(name, salary)

- “Each employee must make more than \$30000.”

```
CREATE ASSERTION empSALARY CHECK  
  (NOT EXISTS  
    (SELECT *  
      FROM emp  
      WHERE emp.sal <= 30000));
```

- If someone **inserts** an employee whose salary is less than or equal to 30K, the insertion/update to **emp** table will be rejected.
- Furthermore, if an employee's salary reduced to less than 30K, the corresponding **update** to **emp** table will also be rejected.

Assertions - Example

Dept(dept#, mgr), Emp(name, salary)

- “Each manager must make more than \$50000.”

```
CREATE ASSERTION mgrSALARY CHECK
  (NOT EXISTS
    (SELECT *
     FROM dept, emp
     WHERE emp.name = dept.mgr AND emp.sal < 50000));
```

- If someone **inserts** a manager whose salary is less than 50K, the insertion/update to **dept** table will be rejected.
- Furthermore, if a manager's salary reduced to less than 50K, the corresponding **update** to **emp** table will also be rejected.

Assertions – Example (cont.)

Dept(dept#, mgr), Emp(name, salary)

- “Each manager must make more than \$50000.”

```
CREATE ASSERTION mgrSALARY CHECK
  (NOT EXISTS
    (SELECT *
     FROM dept, emp
     WHERE emp.name = dept.mgr AND emp.sal < 50000));
```

- When checked?**
 - Insert/updates on the two tables.
 - update to Emp ('Mary', 333, 40K) will be rejected.

Emp (ename, dno, sal)

| ename | dno | sal |
|-------|-----|-----|
| Jack | 111 | 81K |
| Alice | 111 | 70K |
| Lisa | 222 | 51K |
| Tom | 333 | 45K |
| Mary | 333 | 65K |

Dept(dno, dname, mgr)

| dno | dname | mgr |
|-----|-------------|-------|
| 111 | Sells | Alice |
| 222 | Toys | Lisa |
| 333 | Electronics | Mary |

Different Constraint Types

| Type of Constraint | Where Declared | When activated |
|-----------------------|-----------------|---|
| Attribute-based CHECK | With attribute | Insert or update on the attribute |
| Tuple-based CHECK | Relation schema | Insert/update on the relation |
| Assertion | Database schema | Any change in any relation mentioned in assertion |

Triggers

- Motivation:
 - Assertions are powerful, but they need to be checked for all transactions.
 - Attribute- and tuple-based checks are checked at known times, but are not powerful.
 - Triggers enable database programmers to specify
 - **when** to check a constraint,
 - **what** exactly to do.
- NOTE: triggers may cause cascading effects.
- Triggers are not part of SQL2 but included in SQL3.

Triggers

- 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 **condition(constraint)**, and if satisfied, will perform the **action**.
 - Thus, also called “ECA” – Event Condition Action

Elements of Triggers

emp(ename, dno, sal)

“No update can reduce employees’ salaries.”

- (1) CREATE TRIGGER NoLowerSalary
- (2) AFTER UPDATE OF sal ON emp
- (3) REFERENCING
 OLD ROW AS OldTuple
 NEW ROW AS NewTuple
- (4) WHEN (OldTuple.sal > NewTuple.sal)
- (5) FOR EACH ROW
- (6) UPDATE Emp
 SET sal= OldTuple.sal
 WHERE name = NewTuple.ename;

1. Declaration of trigger
2. Timing of action execution: **before**, **after**, or **instead of** triggering event
3. The action can refer to both the old and new state of the database.
4. A condition is specified with a WHEN clause.
5. The action can be performed either
 - a. once for every tuple (row-level), or
 - b. once for all the tuples that are changed by the database operation (*statement level*)
6. Update events may specify a particular column or set of columns.

Row-Level Triggers

emp(ename, dno, sal)

“Handle unknown departments.”

(1) CREATE TRIGGER dnoCheck

(2) BEFORE INSERT ON emp

(3) REFERENCING

NEW ROW AS NewTuple

(4) WHEN (NewTuple.dno NOT IN
(SELECT dno FROM Dept))

(5) FOR EACH ROW

(6) INSERT INTO Dept
VALUES (NewTuple.dno);



Insert the new dno to
Dept table

Statement-Level Triggers

emp(dno...), dept(dno, ...)

“Whenever we insert employees tuples, make sure that their dno’s exist in Dept.”

CREATE TRIGGER deptExistTrig

AFTER INSERT ON emp

REFERENCING

OLD TABLE AS OldStuff

NEW TABLE AS NewStuff

WHEN (EXISTS (SELECT * FROM NewStuff
WHERE dno NOT IN
(SELECT dno FROM dept)))

FOR EACH STATEMENT

DELETE FROM emp
WHERE dno NOT IN
(SELECT dno FROM dept);

- **EVENT:** Trigger is activated when a new tuple is inserted in “emp”
- Trigger action is performed after the insert.

- References to old and new tables are created

- **CONDITION:** When there are dno s not defined in the department table.

- Action is performed on all new tuples (whose dno s are not in dept relation)

Delete the new tuples with undefined department dno s.

PostgreSQL Trigger Example

emp(ename, dno, sal)

“No update can reduce employees’ salaries.”

```
CREATE OR REPLACE FUNCTION updateSalary() RETURNS trigger AS '
BEGIN
    UPDATE Emp
        SET salary= OLD.salary
        WHERE ename = NEW.ename;
    RETURN NEW;
END
' LANGUAGE plpgsql;
```

```
CREATE TRIGGER NoLowerSalary
AFTER UPDATE OF salary ON emp
FOR EACH ROW
WHEN (OLD.salary > NEW.salary)
EXECUTE PROCEDURE updateSalary();
```

To drop the trigger:

```
DROP TRIGGER NoLowerSalary on emp;
```

Summary

- Database modifications
- Attribute-based checks
- Tuple-based checks
- Assertions
- Triggers