Lecture 11
Views in SQL & More

Views

- A view in SQL is a single table derived from other tables
- A view does not necessarily exist in physical form — **virtual table**
- **Base table**: tuples are actually stored in the database
- Restrict possible update operations on views
- No limitation on querying a view

Specification of Views

- View is the result of a SELECT statement over other views or base relations
- Example:

  Students (sid, name, login, age, gpa)

  Enrolled (cid, grade, sid)
- Suppose that we are often interested in finding names and login of students who get a grade B in some course with the cid of the course

CREATE VIEW

```
CREATE VIEW B-students (name, login, course)
AS SELECT s.name, s.login, e.cid
FROM Students S, Enrolled E
Where S.sid=E.sid AND E.grade=‘B’;
```

- Whenever B-students is used in a query, the view definition is evaluated to obtain the corresponding instance of B-students
- The rest of the query is evaluated treating B-students like any other relation referred to in the query

View Benefits

- Access control: users not granted access to base tables. Instead they are granted access to the view of the database appropriate to their needs
  - External schema is composed of views
  - Allows owner to provide SELECT access to a subset of columns

Example

```
CREATE VIEW   PartOfTranscript (StudID, CrsCode, Semester)
AS     SELECT   T. StudID, T.CrsCode, T. Semester       --limit columns
       FROM     Transcript T
       WHERE    T. Semester='S2017';                    --limit rows

Grant SELECT ON PartOfTranscript TO Joe;
```

- The SELECT privilege gives Joe the capability to retrieve information from the PartOfTranscript table

View Benefits (Cont.)

- Customization: Users need not see full complexity of database. View creates the illusion of a simpler database customized to the needs of a particular category of users
- Simplify the specification of certain queries

```
Select name, course
From B-student
Where course=‘database’;
```

View Implementation

- Two main approaches:
  - Query modification: modifying the view query into a query on the underlying base tables
    - Inefficient for views defined via complex queries
  - View materialization: physically creating a temporary view table when the view is first queried. When a query is posed on the view, the (unmodified) query is executed on the precomputed view
    - Much faster than query modifications
    - Maintain the consistency of the precomputed view whenever the underlying tables are updated

View Implementation (Cont.)

- **Incremental update** techniques have been developed where it is determined what new tuples must be inserted, deleted or modified in a materialized view table when a change is applied to one of the defining base tables
- The view is kept as long as it is being queried
- If the view is not queried for a certain period of time, the system may automatically remove the physical view table and recompute it from scratch when future queries reference the view

Specifying Updates in SQL

INSERT

- Add a single tuple to a relation
  - Omit the attribute list if it is the same as in CREATE TABLE
  - Allows to specify explicit attribute names if only part of the attribute list are assigned values in the new tuple
- Insert multiple tuples into a relation in conjunction with creating the relation and loading with the result of a query

Bulk Insertion

```
CREATE TABLE Dept_Info
(Dept_name        VARCHAR(15),
No_of_emps        INTEGER,
Total_sal         INTEGER);

INSERT INTO Dept_Info(Dept_name, No_of_emps,
                              Total_sal)
SELECT Dname, COUNT(*), SUM(Salary)
FROM (DEPARTMENT JOIN EMPOLYEE ON
          Dnumber=Dno)
GROUP BY Dname;
```

DELETE

- A missing WHERE clause: all tuples in the relation are to be deleted
- Includes a WHERE clause
  - No project list in DELETE clause
  - No Cartesian product in FROM clause
  - Tuples satisfying WHERE clause (general form or including subqueries) are deleted

DELETE Examples

```
DELETE FROM     EMPLOYEE;

DELETE FROM     EMPLOYEE
WHERE           SSN='123456789';

DELETE FROM     EMPLOYEE
WHERE           Dno IN (SELECT Dnumber
                        FROM DEPARTMENT
                        WHERE Dname='Research');
```

UPDATE

- Update tuples in a single table
- All tuples satisfying WHERE clause (general form or including subqueries) are updated

UPDATE Examples

```
UPDATE   PROJECT
SET      Plocation=‘fairfax’, Dnum=5
WHERE    Pnumber=10;

UPDATE   EMPLOYEE
SET      Salary = Salary*1.1
WHERE    Dno IN (SELECT  Dnumber
                 FROM    DEPARTMENT
                 WHERE   Dname=‘Research’);
```

Updating Views

- Question: Since views look like tables to users, can they be updated?
- Answer: Yes – a view update changes the underlying base table to produce the requested change to the view
- Example:

```
CREATE VIEW    CSReg (StudID, CrsCode, Semester)
AS      SELECT T. StudID, T.CrsCode, T. Semester
          FROM Transcript T
         WHERE T.CrsCode LIKE ‘CS%’ AND
               T.Semester=‘S2017’;
```

Problem 1

```
INSERT INTO CSReg (StudID, CrsCode, Semester)
Values (1111, ‘CSE305’, ‘S2017’)
```

- Question: What value should be placed in the attributes of the underlying table that have been left out (e.g., Grade)?
- Answer: NULL (assuming null allowed in the missing attribute) or DEFAULT

Problem 2

INSERT INTO CSReg (StudID, CrsCode, Semester)
Values (1111, ‘ECO105’, ‘S2017’)

- Problem: new tuple should not be in the view
- Solutions
  - Allow insertion by default
  - Disallow it by adding the clause WITH CHECK OPTION to the definition of the view

View Example

Student (Sid, Name, Login, Age, GPA)

Club (Cname, Jyear, Mname)

- Suppose we are often interested in finding the name and login of students with a GPA greater than 3 who belong to at least one club along with the club name and the date they joined the club

```
CREATE VIEW   ActiveStudents (Name, Login, Club, Since)
AS     SELECT  S.Name, S.Login, C.Name, C.Jyear
         FROM  Student S, Clubs C
        WHERE  S.Name=C.Mname AND S.GPA>3;
```

Instances

Students

Sid	Name	Login	Age	GPA
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8

Clubs

Cname	Jyear	Mname
Sailing	2016	Dave
Hiking	2017	Smith
Rowing	2018	Smith

ActiveStudents

Name	Login	Club	Since
Dave	dave@cs	Sailing	2016
Smith	smith@ee	Hiking	2017
Smith	smith@ee	Rowing	2018
Smith	smith@math	Hiking	2017
Smith	smith@math	Rowing	2018

Problems

- What if delete the tuple (Smith, smith@ee, hiking, 2017) from ActiveStudents?
- Change either Students or Clubs (or both) in such a way that evaluating the view definition on the modified instance does not produce this change
- Two ways:
  - Delete tuple (53668, Smith, smith@ee, 18, 3.2) from Students
  - Delete tuple (Hiking, 2017, Smith) from Clubs
- Neither is satisfactory!
- Disallow such updates on views

Restrictions

- Updatable views are restricted to those in which
  - No Cartesian product in FROM clause
  - No aggregates, Group by, Having
- A view update is feasible when only one possible update on the base relations can accomplish the desired update effect on the view

Specifying Constraints as Assertions and Triggers

Assertions and Triggers

- **Assertions**: simply offer users to abort an operation that causes a violation
- **Triggers**: specify the type of action to be taken when certain events occur and when certain conditions are satisfied
- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch

```
create assertion sum_constraint check
   (not exists (select * from branch
           where (select sum(amount) from loan
                   where loan.branch_name = branch.branch_name )
                >= (select sum (amount ) from account
                   where account.branch_name =
                   branch.branch_name )));
```

Triggers

- A **trigger** is a procedure that is automatically invoked by the DBMS in response to specified changes to the database
- A database that has a set of associated triggers is called an **active database**
- A trigger description contains three parts
  - Event: A change to the database that activates the trigger
  - Condition: A query or test that is run when the trigger is activated
  - Action: A procedure that is executed when the trigger is activated and its condition is true

Triggers (Cont.)

- An insert, delete, or update statement could activate a trigger
- A *condition* in a trigger can be a true/false statement or a query
  - A query is interpreted as true if the answer set is nonempty and false if the query has no answers
- If the condition part evaluates to true, the action associated with the trigger is executed

Example

- A statement that inserts records into the Student table may activate a trigger that is used to maintain statistics on how many students younger than 18 are inserted at a time by a typical insert statement
- Two triggers are defined:
  - A trigger that initializes a variable used to count the number of qualifying insertions should be executed before changes are made to the Students table
  - A trigger that executes once per qualifying inserted record and increments the variable should be executed after each record is inserted

Example (Cont.)

```
CREATE TRIGGER init-count BEFORE INSERT ON Students  /*Event*/
    DECLARE
           count INTEGER;
    BEGIN                                                /*Action*/
           count:=0;
    END

CREATE TRIGGER incr_count AFTER INSERT ON Students  /*Event*/
    WHEN (new.age<18)                                  /*Condition*/
    FOR EACH ROW
    BEGIN                                              /*Action*/
           count:=count+1;
    END
```