Views

Views, materialized views, temporary views, parameterized views, and recursive views

Views

A SQL query walks into a bar and sees two tables. It walks up to them and says “Can I join you?”

Then a waitress walks up and says “Nice view”.

The database

- A relational database is a collection of base relations

Student

Sid	Sname	Major	Byear
s1	John	CS	1990
s2	Ellen	Math	1995
s3	Eric	CS	1990
s4	Ann	Biology	2001

Course

Cno	Cname	Dept
c1	Dbs	CS
c2	Calc1	Math
c3	Calc2	Math
c4	AI	CS

Enroll

Sid	Cno	Grade
s1	c1	B
s1	c2	A
s2	c3	B
s3	c4	A
s3	c2	C

View

- CS views:

CS_Course

Cno	Cname
c1	Dbs
c4	AI

Student_enrolled_in_CS_course

Sid	Sname	Major	Byear
s1	John	CS	1990
s3	Eric	CS	1990

CS_Enroll_Info

Sid	Cno	Grade
s1	c1	B
s1	c2	A
s3	c2	C
s3	c4	A

Views

- A view is a virtual relation defined by a query
- Example view in SQL:

```
CREATE VIEW CS_Course AS
    SELECT C.Cno, C.Cname
    FROM   Course C
    WHERE C.Dept = 'CS';
```

- This view augments the database with a new relation that contain the (Cno,Cname) pairs for each course offered by the CS department

Views

- A view is a virtual relation defined by a query

```
CREATE VIEW CS_Course AS
    SELECT C.Cno, C.Cname
    FROM   Course C
    WHERE C.Dept = 'CS';
```

→

CS_Course

Cno	Cname
c1	Dbs
c4	AI

Database = Base relations + virtual relations (views)

Student

Sid	Sname	Major	Byear
s1	John	CS	1990
s2	Ellen	Math	1995
s3	Eric	CS	1990
s4	Ann	Biology	2001

Course

Cno	Cname	Dept
c1	Dbs	CS
c2	Calc1	Math
c3	Calc2	Math
c4	AI	CS

Enroll

Sid	Cno	Grade
s1	c1	B
s1	c2	A
s2	c3	B
s3	c1	A
s3	c2	C

CS_course

Cno	Dbs
c1	Dbs
c4	AI

Views (terminology)

```
CREATE VIEW CS_Course AS
    SELECT C.Cno, C.Cname
    FROM   Course C
    WHERE C.Dept = 'CS';
```

- CS_Course is the name of the view (i.e., the name of the virtual relation).
- The SELECT query is the definition of the view.

Views in queries

- Once defined, views can be used as relations in queries.
- Example:

```
SELECT C.Cname
FROM   CS_Course C;
```

returns the names of the courses offered by the CS department.

Views in queries: example

- Consider the query “Find the names of students enrolled in CS courses.”
- In SQL,

view relation

```
SELECT DISTINCT S.Sname
FROM   Student S, Enroll E, CS_Course C
WHERE S.Sid = E.Sid AND E.Cno = C.Cno;
```

Views defined using other views

- Views may also be used in the definition of other views.
- Example: create the view showing all information of students taking CS courses.
- In SQL,

```
CREATE VIEW Student_enrolled_in_CS_course AS
    SELECT S.Sid, S.Sname, S.Major, S.Byear
    FROM   Student S
    WHERE S.Sid IN (SELECT E.Sid
                    FROM Enroll E
                    WHERE E.Cno IN (SELECT C.Cno
                                    FROM    CS_Course C));
```

Views defined using other views

- Example: create a view showing all enrollment information of students who take CS courses
- In SQL,

```
CREATE VIEW CS_Enroll_Info AS
      SELECT E.Sid, E.Cno, E.Grade
      FROM   Enroll E
      WHERE E.Sid IN (SELECT C.Sid
                             FROM   Student_enrolled_in_CS_course C)
```

Importance of views

- Views can be used to modularize a complex query into simpler units. The previous examples illustrate this.
- Views provide independence from changes in the underlying database due to growth and restructuring of the base relation schemas.
- This is called logical data independence

Importance of views: Growth

- Database schemas can grow in several ways:
  - A base table can be expanded by the addition of new attributes and constraints (e.g., using ALTER).
  - New base tables or views can be added to the database.
- The definition of an existing view need not be changed in these cases!

Views unaffected by growth

- Addition of a new columns to a base relation does not affect any views defined using this base relation

```
ALTER TABLE Student
  ADD COLUMN Matriculation_Year INTEGER;
```

- The CS_course and Student_enrolled_in_CS_course views do not need to be re-defined because of this alteration to the Student relation

Importance of views: Restructuring

- It might be necessary to restructure the database in a more fundamental way.
- For example, it could become necessary to replace the relation Student(Sid, Sname, Major, Byear) by the two relations

  Student_Info (Sid, Sname, Byear)
  Student_Major(Sid, Major);

- To ensure that previously defined views remain valid, it now suffices to define Student as a view using the base relations Student_Info and Student_Major.

Restructuring (continued)

- This can easily be done as follows:

```
CREATE VIEW Student AS
    SELECT S.Sid AS Sid, S.Sname AS Sname, S.Byear AS Byear, Sm.Major AS Major
    FROM   Student_Info S, Student_Major Sm
    WHERE S.Sid = Sm. Sid;
```

- Any query or view that previously used the base relation Student does not need to be changed

Garbage collection

- As discussed, views can be defined in terms of base relations and other views.
- This has consequences on the semantics of the DROP TABLE and DROP VIEW commands;
- DROP TABLE *table_name* will only succeed if there are no views with *table_name* as base relation.
- DROP TABLE *table_name* CASCADE will succeed and drop all the views that were defined in terms of *table_name*
- The same applies to DROP VIEW *view_name*

Review of our discussion on views

- Views are defined by queries.
- Recall our example:

```
CREATE VIEW CS_Course AS
    SELECT C.Cno, C.Cname
    FROM   Course C
    WHERE C.Dept = 'CS';
```

- The following SQL query uses this view:

```
SELECT E.Sid
FROM  Enroll E
WHERE E.Cno IN (SELECT C.Cno
                FROM CS_Course C);
```

It returns the sids of students enrolled in CS courses.

Query evaluation with view expansion

- View expansion refers to the process of rewriting queries by expanding the views by their definitions in these queries

```
CREATE VIEW CS_Course AS
    SELECT C.Cno, C.Cname
    FROM   Course C
    WHERE C.Dept = 'CS';
```

```
SELECT  C.Cname                          SELECT C.Cname
FROM    CS_Course C   view expansion →   FROM   (SELECT C.Cno, C.Cname
                                                  FROM   Course C
                                                  WHERE C.Dept = 'CS') C;
```

- View expansion precedes query evaluation

View updates

- Views are essentially designed so that users can work with them as if they were base relations.
- Hence, we do not only want to query views, but also to update views.
- Updating views, however, will necessarily affect the base relations.
- Is it therefore always possible to update views?
- If it is possible, can it be done in a deterministic (algorithmic) way?

Database = Base relations + virtual relations (views)

Student

Sid	Sname	Major	Byear
s1	John	CS	1990
s2	Ellen	Math	1995
s3	Eric	CS	1990
s4	Ann	Biology	2001

Course

Cno	Cname	Dept
c1	Dbs	CS
c2	Calc1	Math
c3	Calc2	Math
c4	AI	CS

Enroll

Sid	Cno	Grade
s1	c1	B
s1	c2	A
s2	c3	B
s3	c1	A
s3	c2	C

CS_course

Cno	Dbs
c1	Dbs
c2	AI

Insertions in views

- Consider again the (very simple) view,

```
CREATE VIEW CS_Course AS
    SELECT C.Cno, C.Cname
    FROM   Course C
    WHERE C.Dept = ‘CS’;
```

- We want to do the following insertion:

```
INSERT INTO CS_Course VALUES(c6, ‘Networks’);
```

Insertions in views (continued)

- In SQL, this is “solved” by (internally) executing the following command:

  INSERT INTO Course VALUES(c6, ‘Networks’, NULL);

- Unfortunately, this will make the (c6, ‘Networks’) disappear from the view again, as the system no longer recognizes this as a CS course …
- It takes quite sophisticated update management to enforce the system to add (c6, ‘Networks’, ‘CS’) instead.
- In SQL, triggers can used for this purpose.

After INSERT of (c6, Networks) in CS_course

Student

Sid	Sname	Major	Byear
s1	John	CS	1990
s2	Ellen	Math	1995
s3	Eric	CS	1990
s4	Ann	Biology	2001

Course

Cno	Cname	Dept
c1	Dbs	CS
c2	Calc1	Math
c3	Calc2	Math
c4	AI	CS
c6	Networks	NULL

Enroll

Sid	Cno	Grade
s1	c1	B
s1	c2	A
s2	c3	B
s3	c1	A
s3	c2	C

CS_course

Cno	Dbs
c1	Dbs
c2	AI

Deletions in views

- Deletions in complex views can also be problematic
- Since 'c1' appear in the CS_course view, the following deletion will remove 'c1' from the Course relation

  DELETE FROM CS_course WHERE Cno = 'c1';

- However, since course 'c6' does not appear in the CS_course view, the following deletion will not remove 'c6' from the Course relation

  DELETE FROM CS_course WHERE Cno = 'c6';

After DELETE of course c6 from CS_course

Student

Sid	Sname	Major	Byear
s1	John	CS	1990
s2	Ellen	Math	1995
s3	Eric	CS	1990
s4	Ann	Biology	2001

Course

Cno	Cname	Dept
c1	Dbs	CS
c2	Calc1	Math
c3	Calc2	Math
c4	AI	CS
c6	Networks	NULL

Enroll

Sid	Cno	Grade
s1	c1	B
s1	c2	A
s2	c3	B
s3	c1	A
s3	c2	C

CS_course

Cno	Dbs
c1	Dbs
c2	AI

View materialization

- In general, a view is a virtual relation.
- If it is used in a query, it is computed at the time this query is evaluated.
- This can be very expensive.
- In many cases, especially if the database state is not changing, it is useful to precompute the view and then store it in the database for future use.
- A view that is precomputed is called a materialized view.

Materialized views (challenges)

- The major challenge with materialized views is that they need to be updated when updates to the base relations (i.e., the state of the basis) occur.
- For certain views (like views that are defined by join queries) this can be done efficiently using incremental changes to the materialized views.

Materialized views (challenges)

- Consider a view V as a result of (inner) join of table R and S is defined as $V = R \bowtie S$
- When table R is changed in a transaction, this can be described as $R \leftarrow R - \nabla R + \Delta R$, where
  - ∇R and ΔR denote tuples deleted from and inserted into R, respectively
- The deltas of the view are calculated as $\nabla V = \nabla R \bowtie S$ and $\Delta V = \Delta R \bowtie S$, then the view can be updated as $V \leftarrow V - \nabla V + \Delta V$

Materialized views (challenges)

- However, when views are defined by queries that involve negation with expressions involving EXCEPT, ALL, NOT IN, and NOT EXISTS, incremental view maintenance may not be possible.
- In such cases, the entire process to materialize the views need to be redone.

Materialized views (challenges)

```
CREATE VIEW foo AS
   SELECT  Cno
   FROM    Course
   WHERE NOT EXISTS (SELECT  1
                            FROM    Course
                            WHERE  cname = 'Java');
 SELECT * FROM foo;
```

Returns all cnos since no course exists with name Java.

However, after the following insertion, no cnos are returned since a course exists with name Java:

```
INSERT INTO Course VALUES ('c5', 'Java', 'CS');
```

Materialized views in PostgreSQL

```
CREATE MATERIALIZED VIEW CS_Course AS
   SELECT C.cno
   FROM   Course C
   WHERE C.dept = 'CS';
```

… other SQL statements (including INSERT/DELETE)

These statements may change the state of CS_course!

```
REFRESH MATERIALIZED VIEW CS_Course;
```

After this, we get the correct state for CS_course

Temporary views (WITH statement)

- A common application of views is to use them to modularize the solution for a query
- Such views need not always be defined and retained
- The WITH statement of SQL permits us to define temporary views which are only local to the query we wish to solve
- These temporary views are not made persistent outside the query
- WITH statement is also referred to as Common Table Expression (CTE)

Temporary views (WITH clause)

- The WITH clause provides a way to define temporary views for use in a larger query

WITH *view_1* AS (SQL query 1),
view_2 AS (SQL query 2),
…
view_n AS (SQL query n)
SQL query that can use *view_1*,…,*view_n*;

- At the end of the evaluation of the last SQL query, the temporary views *view_1* through *view_n* are discarded

Temporary views (example)

"Find the sid and name of each student who takes a Math course and who majors in CS"

```
WITH Math_Course AS (SELECT  Cno, Cname
                     FROM    Course
                     WHERE   Dept = 'Math'),
     Student_enrolled_in_Math_course AS
       (SELECT Sid, Sname, Major, Byear
        FROM   Student
        WHERE Sid IN (SELECT  E.Sid
                      FROM    Enroll E, Math_Course C
                      WHERE E.Cno = C.Cno))
SELECT  Sid, Sname
FROM    Student_enrolled_in_Math_course
WHERE   Major = 'CS';
```

Recursive views/queries

- Using RECURSIVE, a WITH query can refer to its own output

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

- The general form of a recursive WITH query is always
  - a *non-recursive term*,
  - then UNION (or UNION ALL),
  - then a *recursive term*,
    - the recursive term can contain a reference to the query's own output

Recursive Query Evaluation

- Evaluate the non-recursive term
- For UNION (but not UNION ALL), discard duplicate rows
- Store the output in a temporary working table
- Repeat till the working table is not empty:
  - Evaluate the recursive term, self-reference goes to the working table
  - For UNION (but not UNION ALL), discard duplicate rows and rows that duplicate any previous result row
  - Store the result in an intermediate table
  - Add the content of intermediate table to working table, then empty the intermediate table

Recursive views

- Consider a directed graph Graph(source, target)
- A pair (s,t) is in Graph if (s,t) is an edge in the graph
- Consider the relation Path(source,target)
- A pair (s,t) is in Path if there is a path in Graph from s to t
- A recursive definition of Path is as follows:

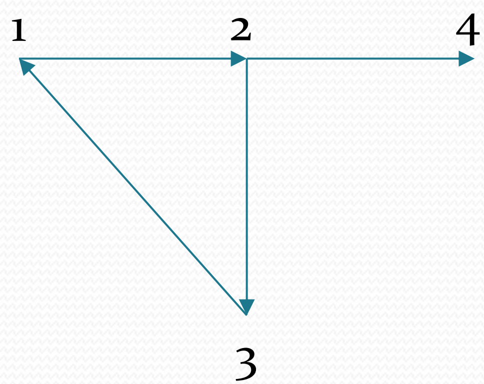
Base rule: If Graph(s,t) then Path(s,t)
Inductive rule: If Graph(s,u) and Path(u,t) then Path(s,t)

Recursively defined temporary views in SQL

```
WITH RECURSIVE Path(source, target) AS
  (
    SELECT E.source, E.target          /* Base rule */
    FROM   Graph E

    UNION

    SELECT  E.source, P.target         /* Inductive rule */
    FROM    Graph E, Path P
    WHERE E.target = P.source
  )

  SELECT * FROM Path;
```

Graph

Source	Target
1	2
2	3
3	1
2	4



1	2
2	3
3	1
2	4

1	3
1	4
2	1
3	2

1	1
2	2
3	3
3	4

Path

Source	Target
1	1
1	2
1	3
1	4
2	1
2	2
2	3
2	4
3	1
3	2
3	3
3	4

Not all subqueries define SQL views!

- Consider the following SQL query:

```
SELECT E.Sid
FROM Student S, Enroll E
WHERE S.Sid = E.Sid AND E.Cno IN (SELECT C.Cno
                                  FROM  Course C
                                  WHERE C.Dept = S.Major);
```

  It returns the sids of all students enrolled in some course offered by the department in which he or she majors.

- Observe that the subquery

```
SELECT C.Cno
FROM  Course C
WHERE C.Dept = S.Major
```

  is not an SQL view, since it has a parameter S.Major.

Parameterized views

- In standard SQL, subqueries such as on the previous slide cannot be used to define parameterized views.
- PostgreSQL, however, permits the use of user-defined functions that return tables.
- Using this feature, we can specify parameterized views.

Parameterized views: example

- In PostgreSQL, we can specify the following function returning tables:

```
CREATE FUNCTION coursesOfferedByDept(deptname TEXT)
          RETURNS TABLE(Cno TEXT) AS
          $$
               SELECT C.Cno
               FROM   Course C
               WHERE C.Dept = deptname;
          $$   LANGUAGE  SQL;
```

- The function returns a table (relation) of course numbers, i.e., of courses offered by the department specified by the value of the parameter deptname.

Parameterized views (continued)

- The query

```
SELECT E.Sid
 FROM Student S, Enroll E
  WHERE S.Sid = E.Sid AND E.Cno IN (SELECT C.Cno
                                     FROM  Course C
                                     WHERE C.Dept = S.Major);
```

can now be rewritten as

```
SELECT E.Sid
FROM Student S, Enroll E
WHERE S.Sid = E.Sid AND
        E.Cno IN (SELECT C.Cno
                  FROM coursesOfferedByDept(S.Major) C);
```

set of courses offered in the department of student S's major

Parameterized views (continued)

- Incidentally, if we only want to report the courses in a specific department, we can do this by changing the subquery as follows:

```
SELECT C.Cno
FROM  coursesOfferedByDept('Math') C
```

- This query returns the courses offered by the Math department.