Intermediate SQL

Views

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database)

- Consider a person who needs to know an instructors name and department, but not the salary
- This person should see a relation described, in SQL, by

select *ID*, *name*, *dept_name*
from *instructor*

- A **view** provides a mechanism to hide certain data from the view of certain users

- Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a **view**

View Definition

- A view is defined using the **create view** statement which has the form

  create view *v* **as** < query expression >

  where <query expression> is any legal SQL expression; the view name is represented by *v*

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates

- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view

Example Views

- A view of instructors without their salary

create view *faculty* as
select *ID*, *name*, *dept_name*
from *instructor*

- Find all instructors in the Biology department

select *name*
from *faculty*
where *dept_name* = 'Biology'

- Create a view of department salary totals

create view *departments_total_salary(dept_name, total_salary)* as
select *dept_name*, **sum** (*salary*)
from *instructor*
group by *dept_name*;

Views Defined Using Other Views

- One view may be used in the expression defining another view

- A view relation v_1 is said to ***depend directly*** on a view relation v_2 if v_2 is used in the expression defining v_1

- A view relation v_1 is said to ***depend on*** view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2

- A view relation v is said to be ***recursive*** if it depends on itself

Views Defined Using Other Views

```
create view physics_fall_2017 as
    select course.course_id, sec_id, building, room_number
    from course, section
    where course.course_id = section.course_id
            and course.dept_name = 'Physics'
            and section.semester = 'Fall'
            and section.year = '2017';

create view physics_fall_2017_watson as
    select course_id, room_number
    from physics_fall_2017
    where building = 'Watson';
```

View Expansion

- Expand use of a view in a query/another view:

```
create view physics_fall_2017_watson  as
    select course_id, room_number
    from physics_fall_2017
    where building= 'Watson'
```

- To:

```
create view physics_fall_2017_watson as
    select course_id, room_number
    from (select course.course_id, building, room_number
        from course, section
        where course.course_id = section.course_id
                and course.dept_name = 'Physics'
                and section.semester = 'Fall'
                and section.year = '2017')
    where building= 'Watson';
```

View Expansion

- A way to define the meaning of views defined in terms of other views
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations
- View expansion of an expression repeats the following replacement step:

  repeat

  Find any view relation v_i in e_1

  Replace the view relation v_i by the expression defining v_i

  until no more view relations are present in e_1

- As long as the view definitions are not recursive, this loop will terminate

Recursive View

- In SQL, recursive queries are typically built using these components:
  - A non-recursive seed statement
  - A recursive statement
  - A connection operator
    - The only valid set connection operator in a recursive view definition is UNION ALL
  - A terminal condition to prevent infinite recursion

Recursive View Example

- In the context of a relation *flights*:

```
create table flights (
    source varchar(40),
    destination varchar(40),
    carrier varchar(40),
    cost decimal(5,0));
```

source	destination	carrier	cost
Paris	Detroit	KLM	7
Paris	New York	KLM	6
Paris	Boston	American Airlines	8
New York	Chicago	American Airlines	2
Boston	Chicago	American Airlines	6
Detroit	San Jose	American Airlines	4
Chicago	San Jose	American Airlines	2

- Find all the destinations that can be reached from ‘Paris’

Recursive View Example

```
create recursive view reachable_from (source,destination,depth) as (
        select root.source, root.destination, 0 as depth
        from flights as root
        where root.source = 'Paris'
union all
        select in1.source, out1.destination, in1.depth + 1
        from reachable_from as in1, flights as out1
        where in1.destination = out1.source and
        in1.depth <= 100);
```

- A non-recursive seed statement
- A recursive statement
- A connection operator
- A terminal condition to prevent infinite recursion

source	destination	carrier	cost
Paris	Detroit	KLM	7
Paris	New York	KLM	6
Paris	Boston	American Airlines	8
New York	Chicago	American Airlines	2
Boston	Chicago	American Airlines	6
Detroit	San Jose	American Airlines	4
Chicago	San Jose	American Airlines	2

- Get the result by simple selection on the view:

```
select distinct source, destination
        from reachable_from;
```

source	destination
Paris	Detroit
Paris	New York
Paris	Boston
Paris	Chicago
Paris	San Jose

This example view, *reachable_from*, is called the *transitive closure* of the *flights* relation

Source: https://info.teradata.com/HTMLPubs/DB_TTU_16_00/index.html#page/SQL_Reference%2FB035-1184-160K%2Fsme1472241335807.html%23wwID0EJ23T

The Power of Recursion

- Recursive views make it possible to write queries, such as transitive closure queries, that cannot be written without recursion or iteration
  - Intuition: Without recursion, a non-recursive non-iterative program can perform only a fixed number of joins of *flights* with itself
    - This can give only a fixed number of levels of reachable destinations
    - Given a fixed non-recursive query, we can construct a database with a greater number of levels of reachable destinations on which the query will not work

The Power of Recursion

- Computing transitive closure using iteration, adding successive tuples to *reachable_from*
  - The next slide shows a *flights* relation
  - Each step of the iterative process constructs an extended version of *reachable_from* from its recursive definition
  - The final result is called the *fixed point* of the recursive view definition

- Recursive views are required to be **monotonic**
  - That is, if we add tuples to flights the view reachable_from contains all of the tuples it contained before, plus possibly more

Example of Fixed-Point Computation

source	destination	carrier	cost
Paris	Detroit	KLM	7
Paris	New York	KLM	6
Paris	Boston	American Airlines	8
New York	Chicago	American Airlines	2
Boston	Chicago	American Airlines	6
Detroit	San Jose	American Airlines	4
Chicago	San Jose	American Airlines	2

Iteration #	Tuples in Closure
0	Detroit, New York, Boston
1	Detroit, New York, Boston, San Jose, Chicago
2	Detroit, New York, Boston, San Jose, Chicago

Update of a View

- Add a new tuple to *faculty* view which we defined earlier

  insert into *faculty*
  values ('30765', 'Green', 'Music');

- This insertion must be represented by the insertion into the *instructor* relation
  - Must have a value for salary

- Two approaches
  - Reject the insert
  - Insert the tuple into the *instructor* relation

    ('30765', 'Green', 'Music', null)

Some Updates Cannot be Translated Uniquely

```
create view instructor_info as
    select ID, name, building
    from instructor, department
    where instructor.dept_name= department.dept_name;

insert into instructor_info
    values ('69987', 'White', 'Taylor');
```

- Issues
  - Which department, if multiple departments in Taylor?
  - What if no department is in Taylor?

And Some Not at All

```
create view history_instructors as
    select *
    from instructor
    where dept_name= 'History';
```

- What happens if we insert into *history_instructors?*

  ('25566', 'Brown', 'Biology', 100000)

View Updates in SQL

- Most SQL implementations allow updates only on simple views
  - The **from** clause has only one database relation
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification
  - Any attribute not listed in the **select** clause can be set to null
  - The query does not have a **group** by or **having** clause

Materialized Views

- Certain database systems allow view relations to be physically stored
  - Physical copy created when the view is defined
  - Such views are called **Materialized view**

- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated

Next Lecture

Intermediate SQL

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in