Outer Join – Example

• right outer join

SELECT * FROM instructor RIGHT JOIN teaches

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>CS-101</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>FIN-201</td>
</tr>
<tr>
<td>76766</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>FIN-201</td>
</tr>
</tbody>
</table>

• full outer join

SELECT * FROM instructor FULL OUTER JOIN teaches

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>course_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>CS-101</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>FIN-201</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>FIN-201</td>
</tr>
<tr>
<td>76766</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>FIN-201</td>
</tr>
</tbody>
</table>

CMSC424: Database Design

SQL cont.

Professor: Pete Keleher
keleher@cs.umd.edu
Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A subquery is a select-from-where expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Example Query

- Find courses offered in Fall 2009 and in Spring 2010

```sql
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
  course_id in (select course_id
                 from section
                 where semester = 'Spring' and year= 2010);
```

- Find courses offered in Fall 2009 but not in Spring 2010

```sql
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
  course_id not in (select course_id
                     from section
                     where semester = 'Spring' and year= 2010);
```

Already did w/ set operations
**Example Query**

- Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
(select course_id, sec_id, semester, year
from teaches
where teaches.ID = 10101);
```

- Note: Above query could also be written more efficiently with a join. The formulation above is simply to illustrate SQL features.

**Definition of Some Clause**

- F <comp> some r ↔ ∃ t ∈ r such that (F <comp> t)
  - Where <comp> can be: <, >=, >, =, !=, <>

  - (5 < some 5) = true
  - (5 < some 6) = true (read: 5 < some tuple in the relation)
  - (5 < some 0) = false
  - (5 = some 5) = true
  - (5 != some 5) = true (the 0)

- However, (!= some) is not the same as not in
Set Comparison

• Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```sql
select distinct T.name
from instructor T, instructor S
where T.salary > S.salary and S.dept name = 'Biology';
```

• Same query using > some clause

```sql
select name
from instructor
where salary > some (select salary
from instructor
where dept name = 'Biology');
```

Definition of all clause

• F <comp> all r ↔ ∀ t ∈ r (F <comp> t)

- (5 < all 0) = false
- (5 < all 5) = false
- (5 < all 6) = true
- (5 = all 4) = false
- (5 != all 6) = true (since 5 ne 4 and 5 ne 6)
Example Query

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
    from instructor
    where dept name = 'Biology');
```

Test for Empty Relations

- The `exists` construct returns the value `true` if the argument subquery is nonempty.
- `exists` $r \iff r \neq \emptyset$
- `not exists` $r \iff r = \emptyset$


**Correlated Subqueries**

- Yet another way of specifying the query “Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester”

```sql
select course_id
from section F
where semester = 'Fall' and year = 2009 and
exists (select *
        from section S
        where semester = 'Spring' and year = 2010
        and F.course_id = S.course_id);
```

- **Correlation name** or **correlation variable**

**Not Exists**

- Find all students who have taken all courses offered in the Biology department.

```sql
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                   from course
                   where dept_name = 'Biology')
                   minus
                   (select T.course_id
                    from takes as T
                    where S.ID = T.ID));
```

- **Note**: Cannot write this query using = all and its variants
Test for Absence of Duplicate Tuples

Find all courses that were offered at most once in 2009:

- **WRONG:** unique is used to define constraints at table creation.
- select T.course_id
  from course T
  where unique (select R.course_id
    from section R
    where T.course_id = R.course_id
    and R.year = 2009);

- **RIGHT:**
  select T.course_id from course T
  where 1 = (select count(R.course_id)
    from section R
    where T.course_id = R.course_id and R.year = 2009);

Derived Relations

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors’ salaries of those departments where the average salary is greater than $42,000.”
  
  select dept_name, avg_salary
  from (select dept_name, avg(salary) as avg_salary
    from instructor
    group by dept_name)
  where avg_salary > 42000;

- Note that we do not need to use the **having** clause
Derived Relations (Cont.)

```sql
select dname, dsalary
from (select dept_name, avg(salary) as avg_salary
        from instructor
        group by dept_name)
    dept_avg(dname, dsalary)
where dsalary > 42000;
```

---

**Triggers**

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
  - 1. setting the account balance to zero
  - 2. creating a loan in the amount of the overdraft
  - 3. giving this loan a loan number identical to the account number of the overdrawn account
Trigger Example in SQL:1999

create trigger overdraft-trigger after update on account
    referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
    actions to be taken
end

create trigger overdraft-trigger after update on account
    referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
    insert into borrower
        (select customer-name, account-number
         from depositor
         where nrow.account-number = depositor.account-number);
    insert into loan values
        (nrow.account-number, nrow.branch-name, nrow.balance);
    update account set balance = 0
    where account.account-number = nrow.account-number
end
Triggers...

- External World Actions
  - How does the DB order something if the inventory is low?

- Syntax
  - Every system has its own syntax

- Careful with triggers
  - Cascading triggers, Infinite Sequences...

- More Info/Examples:
  - https://www.tutorialspoint.com/postgresql/postgresql_triggers.htm
  - Google “create trigger postgresql”

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

  ```sql
  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

  ```sql
  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
  ```sql
  create view faculty as
  select ID, name, dept_name
  from instructor
  ```

- Find all instructors in the Biology department
  ```sql
  select name
  from faculty
  where dept_name = 'Biology'
  ```

- Create a view of department salary totals
  ```sql
  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

• create view physics_fall_2009 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 = '2009';

• create view physics_fall_2009_watson as
  select course_id, room_number
  from physics_fall_2009
  where building = 'Watson';

View Expansion

• Expand use of a view in a query/another view

create view physics_fall_2009_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 = '2009')
    where building = 'Watson';
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.

View Expansion

- A way to interpret queries w/ views...
  - Let view $v_i$ be defined by an expression $e_i$ that may itself contain uses of view relations.
  - View expansion of an expression $e$ repeats the following replacement step:

  ```repeat
  Find any view relation $v_i$ in $e$
  Replace the view relation $v_i$ by expression $e_i$
  until no more view relations are present in $e$
  ```

  - As long as the view definitions are not recursive, this loop will terminate.
Update of (through) 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 of the tuple
  
  `('30765', 'Green', 'Music', null)`  
  
  into the instructor relation.

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');`
  
  - which department, if multiple departments in Taylor?
  
  - what if no department is in Taylor?

- 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.
And Some Not at All

- \texttt{create view history\_instructors as}
  \begin{verbatim}
  select * 
  from instructor 
  where dept\_name= 'History';
  \end{verbatim}
- Insert ('25566', 'Brown', 'Biology', 100000) into \textit{history\_instructors}