Introduction to SQL – 2

Debapriyo Majumdar
DBMS – Fall 2016
Indian Statistical Institute Kolkata

Slides re-used, with minor modification, from
Silberschatz, Korth and Sudarshan

www.db-book.com
Outline

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database
A **subquery** is a *select-from-where* expression that is nested within another query.

The nesting can be done in the SQL query:

\[
\text{select } A_1, A_2, \ldots, A_n \\
\text{from } r_1, r_2, \ldots, r_m \\
\text{where } P
\]

as follows:

- \( A_i \) can be replaced by a subquery that generates a single value
- \( r_i \) can be replaced by any valid subquery
- \( P \) can be replaced with an expression of the form:
  \[ B \text{ <operation> (subquery) } \]
  Where \( B \) is an attribute and <operation> to be defined later
A common use of subqueries is to perform tests
- For set membership
- For set comparisons
- For set cardinality
Set Membership

- 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);
  ```
Set Membership (Continued)

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

```sql
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:** The above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.
Set Comparison – “some” Clause

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

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

- Same query using > some clause

```
select name
from instructor
where salary > some (select salary
    from instructor
    where dept name = 'Biology');
```
Definition of “some” Clause

- $F \ <\text{comp}\ > \ some \ r \iff \exists \ t \in r \text{ such that } (F <\text{comp}> t)$

Where $<\text{comp}>$ can be: $<, \leq, >, =, \neq$

$(5 < some \ 5) = true \ (\text{read: } 5 < \text{some tuple in the relation})$

$(5 < some \ 5) = false$

$(5 = some \ 5) = true$

$(5 \neq some \ 5) = true \ (\text{since } 0 \neq 5)$

$(\equiv some) \equiv in$

However, $(\neq some) \neq not in$
Set Comparison – “all” Clause

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

```sql
select name
from instructor
where salary > all (select salary
    from instructor
    where dept name = ’Biology’);
```
Definition of “all” Clause

- $F \text{ <comp> all } r \iff \forall t \in r \ (F \text{ <comp> } t)$

<table>
<thead>
<tr>
<th>0</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$(5 < \text{ all } 5) = \text{false}$

<table>
<thead>
<tr>
<th>6</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

$(5 < \text{ all } 10) = \text{true}$

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

$(5 = \text{ all } 5) = \text{false}$

<table>
<thead>
<tr>
<th>4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

$(5 \neq \text{ all } 6) = \text{true \ (since } 5 \neq 4 \text{ and } 5 \neq 6)$

$(\neq \text{ all}) = \text{not in}$

However, $(= \text{ all}) \neq \text{ in}$
Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- \( \text{exists } r \iff r \neq \emptyset \)
- \( \text{not exists } r \iff r = \emptyset \)
Use of “exists” Clause

- 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 as S
where semester = 'Fall' and year = 2009 and
exists (select *
    from section as T
    where semester = 'Spring' and year = 2010
    and S.course_id = T.course_id);
```

- **Correlation name** – variable S in the outer query
- **Correlated subquery** – the inner query
Use of “not exists” Clause

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

\[
\text{select distinct } S.ID, S.name \text{ from student as } S
\text{ where not exists ( (select course_id from course}
\text{ where dept_name = ’Biology’) }
\text{ except}
\text{ (select T.course_id from takes as T}
\text{ where S.ID = T.ID));}
\]

- First nested query lists all courses offered in Biology
- Second nested query lists all courses a particular student took

- Note that \( X – Y = \emptyset \iff X \subseteq Y \)
- \textbf{Note:} Cannot write this query using \( = \) all and its variants
Test for Absence of Duplicate Tuples

- The `unique` construct tests whether a subquery has any duplicate tuples in its result.
- The `unique` construct evaluates to “true” if a given subquery contains no duplicates.
- Find all courses that were offered at most once in 2009.

```sql
select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id = R.course_id
              and R.year = 2009);
```
Subqueries in the From Clause

- SQL allows a subquery expression to be used in the `from` clause.
- Find the average instructors’ salaries of those departments were the average salary is greater than $42,000
  
  ```sql
  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.
- Another way to write above query
  
  ```sql
  select dept_name, avg_salary
  from (select dept_name, avg(salary) from instructor
        group by dept_name) as dept_avg (dept_name, avg_salary)
  where avg_salary > 42000;
  ```
The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs.

Find all departments with the maximum budget

```sql
with max_budget (value) as
  (select max(budget)
   from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

MySQL does not support **WITH** clause
Complex Queries using With Clause

- Find all departments where the total salary is greater than the average of the total salary at all departments

```sql
with dept_total (dept_name, value) as
    (select dept_name, sum(salary) 
     from instructor 
     group by dept_name),
dept_total_avg(value) as 
    (select avg(value) 
     from dept_total) 
select dept_name 
from dept_total, dept_total_avg 
where dept_total.value > dept_total_avg.value;
```

MySQL does not support WITH clause
Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- List all departments along with the number of instructors in each department

```sql
select dept_name,
    (select count(*)
        from instructor
        where department.dept_name =
            instructor.dept_name)
    as num_instructors
from department;
```

- Runtime error if subquery returns more than one result tuple
Modification of the Database

- Deletion of tuples from a given relation
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation
Deletion

- Delete all instructors
  
  ```sql
  delete from instructor
  ```

- Delete all instructors from the Finance department
  
  ```sql
  delete from instructor
  where dept_name= 'Finance';
  ```

- Delete all tuples in the `instructor` relation for those instructors associated with a department located in the Watson building
  
  ```sql
  delete from instructor
  where dept_name in
  (select dept_name from department
   where building = 'Watson');
  ```
Deletion (Continued)

- Delete all instructors whose salary is less than the average salary of instructors

  \[
  \text{delete from instructor}
  \text{where salary < (select avg (salary) from instructor);} \]

- Problem: as we delete tuples from deposit, the average salary changes

- Solution used in SQL:
  1. First, compute \text{avg (salary)} and find all tuples to delete
  2. Next, delete all tuples found above (without recomputing \text{avg} or retesting the tuples)
Insertion

- Add a new tuple to course
  \[\text{insert into course} \]
  \[\text{values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);}\]

- or equivalently
  \[\text{insert into course (course_id, title, dept_name, credits)}\]
  \[\text{values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);}\]

- Add a new tuple to student with tot_creds set to null
  \[\text{insert into student} \]
  \[\text{values ('3003', 'Green', 'Finance', null);}\]
Add all instructors to the student relation with tot_creds set to 0

\[
\text{insert into student }
\text{select ID, name, dept_name, 0}
\text{from} \quad \text{instructor}
\]

The select from where statement is evaluated fully before any of its results are inserted into the relation.

Otherwise queries like

\[
\text{insert into table1 select * from table1}
\]

would cause problem
Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two `update` statements:
    
    ```sql
    update instructor
    set salary = salary * 1.03
    where salary > 100000;
    
    update instructor
    set salary = salary * 1.05
    where salary <= 100000;
    ```
  - The order is important
  - Can be done better using the `case` statement
Case Statement for Conditional Updates

- Same query as before but with case statement

```sql
update instructor
set salary = case
    when salary <= 100000 then salary * 1.05
    else salary * 1.03
end
```
Updates with Scalar Subqueries

- Recompute and update tot_creds value for all students

  \[
  \text{update student } S \\
  \text{set tot_cred} = (\text{select sum(credits)} \\
  \text{from takes, course} \\
  \text{where takes.course_id} = \text{course.course_id} \text{ and} \\
  S.ID = \text{takes.ID} \text{ and takes.grade } \neq 'F' \text{ and} \\
  \text{takes.grade is not null});
  \]

- Sets tot_creds to null for students who have not taken any course

- Instead of sum(credits), use:

  \[
  \text{case} \\
  \text{when sum(credits) is not null then sum(credits)} \\
  \text{else 0} \\
  \text{end}
  \]