SQL Queries and Subqueries

Zaki Malik

September 04, 2008
Basic SQL Query

\[
SELECT \textbf{[DISTINCT]} \ target-list \\
FROM \ relatin-list \\
WHERE \ qualification;
\]

- **Relation-list**: A list of relation names (possibly with range-variable after each name).
- **Target-list**: A list of attributes of relations in relation-list
- **Qualification**: conditions on attributes
- **DISTINCT**: optional keyword for duplicate removal.
  - Default = no duplicate removal!
# SQL Comparison Operators

<table>
<thead>
<tr>
<th>Comparison Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to (used by most implementations of SQL)</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to (used by some implementations of SQL)</td>
</tr>
</tbody>
</table>
How to evaluate a query?

\[
\text{SELECT [DISTINCT] target-list} \\
\text{FROM relation-list} \\
\text{WHERE qualification;}
\]

- **Conceptual query evaluation** using relational operators:
  1) Compute the cross-product of relation-list.
  2) Discard resulting tuples if they fail qualifications.
  3) Delete attributes that are not in target-list. (called column-list)
  4) If DISTINCT is specified, eliminate duplicate rows.

\[
\text{SELECT S.sname} \\
\text{FROM Sailors S, Reserves R} \\
\text{WHERE S.sid=R.sid AND R.bid=103;}
\]
Example of Conceptual Evaluation (1)

Compute the cross-product of relation-list.

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103;
```

Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Example of Conceptual Evaluation (2)

SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103;

<table>
<thead>
<tr>
<th>S.sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>R.sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

(2) Discard tuples if they fail qualifications.
Example of Conceptual Evaluation (3)

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103;
```

(3) Delete attribute columns that are not in target-list.

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Renaming / Aliasing

Consider the following SALESREPS relation

<table>
<thead>
<tr>
<th>Empl_num</th>
<th>name</th>
<th>age</th>
<th>Rep_office</th>
<th>manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>Bill</td>
<td>37</td>
<td>13</td>
<td>104</td>
</tr>
<tr>
<td>104</td>
<td>Bob</td>
<td>33</td>
<td>12</td>
<td>106</td>
</tr>
<tr>
<td>106</td>
<td>Sam</td>
<td>52</td>
<td>11</td>
<td>NULL</td>
</tr>
</tbody>
</table>

How do we determine the name of Bob’s manager?
SELECT s2.name
FROM SALESREPS s1, SALESREPS s2
WHERE s1.name='Bob' AND
    s1.manager=s2.empl_num;

• Aliases must be used here.
• The row referenced by s1 is intended to be Bob…
• …while s2 will be his manager’s.
• Remember, first FROM, then WHERE, then SELECT
Relational Design Example

- Students (PID: string, Name: string, Address: string)
- Professors (PID: string, Name: string, Office: string, Age: integer, DepartmentName: string)
- Courses (Number: integer, DeptName: string, CourseName: string, Classroom: string, Enrollment: integer)
- Teach (ProfessorPID: string, Number: integer, DeptName: string)
- Take (StudentPID: string, Number: integer, DeptName: string, Grade: string, ProfessorEvaluation: integer)
- Departments (Name: string, ChairmanPID: string)
- PreReq (Number: integer, DeptName: string, PreReqNumber: integer, PreReqDeptName: string)
Motivation for Subqueries

• Find the name of the professor who teaches “CS 4604.”
  
  SELECT Name 
  FROM Professors, Teach 
  WHERE (PID = ProfessorPID) AND (Number = ‘4604’) 
    AND(DeptName = ‘CS’); 

• Do we need to take the natural join of two big relations 
  just to get a relation with one tuple?

• Can we rewrite the query without using a join?
Nesting

- A query can be put inside another query
- Most commonly in the WHERE clause
- Sometimes in the FROM clause (depending on the software)
- This subquery is executed first (if possible)
Subquery Example

• Find the name of the professor who teaches “CS 4604.”

```sql
SELECT Name
FROM Professors
WHERE PID =
    (SELECT ProfessorPID
     FROM Teach
     WHERE (Number = 4604) AND (DeptName = 'CS')
    );
```

• When using =, the subquery must return a single tuple
Conditions Involving Relations

• SQL includes a number of operators that apply to a relation and produce a boolean result.

• These operators are very useful to apply on results of sub-queries.

• Let R be a relation and t be a tuple with the same set of attributes.
  – EXISTS R is true if and only if R contains at least one tuple.
  – t IN R is true if and only if t equals a tuple in R.
  – t > ALL R is true if and only if R is unary (has one attribute) and t is greater than every value in R.
    • Can use any of the other five comparison operators.
    • If we use <> , R need not be unary.
  – t > ANY R (which is unary) is true if and only if t is greater than at least one value in R.

• We can use NOT to negate EXISTS, ALL, and ANY.
Subqueries Using Conditions

- Find the departments of the courses taken by the student with name ‘Suri’.

SELECT DeptName
FROM Take
WHERE StudentPID IN
  ( SELECT PID
    FROM Students
    WHERE (Name = 'Suri')
  );
Correlated vs Uncorrelated

• The previous subqueries did not depend on anything outside the subquery
  – ...and thus need to be executed just once.
  – These are called uncorrelated.

• A correlated subquery depends on data from the outer query
  – ... and thus has to be executed for each row of the outer table(s)
Correlated Subqueries

• Find course names that have been used for two or more courses.

```sql
SELECT CourseName
FROM Courses AS First
WHERE CourseName IN
  (SELECT CourseName
   FROM Courses
   WHERE (Number <> First.Number)
   AND (DeptName <> First.DeptName)
  );
```
Evaluating Correlated Subqueries

SELECT CourseName
FROM Courses AS First
WHERE CourseName IN
    (SELECT CourseName
     FROM Courses
     WHERE (Number <> First.Number)
     AND (DeptName <> First.DeptName)
    );

• Evaluate query by looping over tuples of First, and for each tuple evaluate the subquery.

• Scoping rules: an attribute in a subquery belongs to one of the tuple variables in that subquery’s FROM clause, or to the immediately surrounding subquery, and so on.
Subqueries in FROM clauses

• Can use a subquery as a relation in a FROM clause.
• We must give such a relation an alias using the AS keyword.
• Let us find different ways of writing the query “Find the names of Professors who have taught the student whose first name is ‘Suri’.”

• The old way:
SELECT Professors.Name
FROM Professors, Take, Teach, Students
WHERE (Professors.PID = Teach.ProfessorPID)
   AND (Teach.CourseNumber = Take.CourseNumber)
   AND (Teach.DeptName = Take.DeptName)
   AND (Take.StudentPID = Student.PID)
   AND (Student.Name = 'Suri %');
• “Find the names of (Professors who have taught (courses taken by (student with first name ‘Suri’)))).”

```
SELECT Name
FROM Professors
WHERE PID IN
    (SELECT ProfessorPID
     FROM Teach
     WHERE (Number, DeptName) IN
         ( SELECT Number, DeptName
             FROM Take
             WHERE StudentPID IN
                 (SELECT PID
                  FROM Students
                  WHERE Name = ‘Suri %’)
         )
    );
```
Aggregate Operators

- \texttt{COUNT (*)}
- \texttt{COUNT ([DISTINCT] A)}
  - \( A \) is a column
- \texttt{SUM ([DISTINCT] A)}
- \texttt{AVG ([DISTINCT] A)}
- \texttt{MAX (A)}
- \texttt{MIN (A)}
- Count the number of sailors
  \begin{verbatim}
  SELECT COUNT (*)
  FROM Sailors S
  \end{verbatim}
Find the average age of sailors with rating = 10

Sailors(sid: integer, sname: string, rating: integer, age: real)

SELECT AVG (S.age) FROM Sailors S WHERE S.rating=10
Count the number of different sailor names

Sailors(sid: integer, sname: string, rating: integer, age: real)

SELECT COUNT (DISTINCT S.sname)
FROM Sailors S
Find the age of the oldest sailor

Sailors(sid: integer, sname: string, rating: integer, age: real)

SELECT MAX(S.AGE)
FROM Sailors S
Find name and age of the oldest sailor(s)

SELECT S.sname, MAX (S.age)
FROM Sailors S

• This is illegal, but why?
  – Cannot combine a column with a value

SELECT S.sname, S.age
FROM Sailors S
WHERE S.age = (SELECT MAX (S2.age) FROM Sailors S2)
GROUP BY and HAVING

• So far, aggregate operators are applied to all (qualifying) tuples.
  – Can we apply them to each of several groups of tuples?
• Example: find the age of the youngest sailor for each rating level.
  – In general, we don’t know how many rating levels exist, and what the rating values for these levels are!
  – Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this:

For \( i = 1, 2, \ldots, 10 \):  

\[
\begin{align*}
  \text{SELECT} & \quad \text{MIN} \ (S.\text{age}) \\
  \text{FROM} & \quad \text{Sailors} \ S \\
  \text{WHERE} & \quad S.\text{rating} = i
\end{align*}
\]
Find the age of the youngest sailor for each rating level

SELECT S.rating, MIN (S.age) as age
FROM Sailors S
GROUP BY S.rating

(1) The sailors tuples are put into “same rating” groups.
(2) Compute the Minimum age for each rating group.
Find the age of the youngest sailor for each rating level that has at least 2 members

SELECT S.rating, MIN (S.age) as minage
FROM Sailors S
GROUP BY S.rating
HAVING COUNT(*) > 1

1. The sailors tuples are put into “same rating” groups.

2. Eliminate groups that have < 2 members.

3. Compute the Minimum age for each rating group.

<table>
<thead>
<tr>
<th>Sid</th>
<th>Sname</th>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>
Queries With *GROUP BY* and *HAVING*

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification
```

- The *target-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., AVG (*S.age*)).
- The attribute list (e.g., *S.rating*) in *target-list* must be in *grouping-list*.
- The attributes in group-qualification must be in *grouping-list*.

```
SELECT S.rating, MIN (S.age) as age
FROM Sailors S
GROUP BY S.rating
HAVING S.rating > 5
```
Starwars Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• Which planet does Princess Leia go to in movie3?

SELECT distinct pname
FROM timetable
WHERE cname = 'Princess Leia' and movie=3;
Starwars Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• How many humans stay on Dagobah in movie 3?

SELECT count(*)
FROM timetable, characters
WHERE movie=3 and pname =‘Dagobah’ and
   timetable.cname=characters.name and
   characters.race=‘Human’;
Starwars Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• Who has been to his/her homeworld in movie 2?

SELECT distinct c.name
FROM characters c, timetable t
WHERE c.name=t.cname and t.pname=c.homeworld and
      movie=2;
Starwars Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• Find distinct names of the planets visited by those of race “droid”.

SELECT distinct t.pname
FROM char c, timetable t
WHERE c.name=t.cname and c.race=‘droid’;
Starwars Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• For each character and for each neutral planet, how much time total did the character spend on the planet?

SELECT c.name, p.name, SUM(t.departure-t.arrival) as amount
FROM characters c, timetable t, planets p
WHERE t.cname=c.name and t.pname=p.name and p.affiliation='neutral'
GROUP BY c.name, p.name;