SQL Subqueries

T. M. Murali

September 1, 2010
Relational algebra expressions can become very long.
Use linear notation to store results of intermediate expressions.

1. A relation name and a parenthesised list of attributes for that relation. Use Answer as the conventional name for the final result.
2. The assignment symbol :=.
3. Any expression in relational algebra on the right.
Example of Linear Notation

- Name pairs of students who live at the same address.
- Normal expression:

\[ \pi_{S1.\text{Name},S2.\text{Name}}(\sigma_{S1.\text{Address} = S2.\text{Address}}(\rho_{S1}(\text{Students}) \times \rho_{S2}(\text{Students}))) \]
Name pairs of students who live at the same address.

Normal expression:

\[ \pi_{S1}\text{.Name}, S2\text{.Name}(\sigma_{S1}\text{.Address} = S2\text{.Address}(\rho_{S1}(\text{Students}) \times \rho_{S2}(\text{Students}))) \]

Linear notation:

Pairs(P1, N1, A1, P2, N2, A2) := \rho_{S1}(\text{Students}) \times \rho_{S2}(\text{Students})

Matched(P1, N1, A1, P2, N2, A2) := \sigma_{A1 = A2}(\text{Pairs(P1, N1, A1, P2, N2, A2)})

Answer(\text{Name1, Name2}) := \pi_{N1, N2}(\text{Matched(P1, N1, A1, P2, N2, A2)})
Interpreting Queries Involving Multiple Relations

SELECT A, B FROM R, S WHERE C;

▶ Nested loops:
for each tuple $t_1$ in $R$
for each tuple $t_2$ in $S$
if the attributes in $t_1$ and $t_2$ satisfy $C$
output the tuples involving attributes $A$ and $B$.

▶ Conversion to relational algebra:
$\pi_{A,B}(\sigma_C(R \times S))$.

1. Compute $R \times S$.
2. Apply the selection operator $\sigma_C()$ to $R \times S$.
3. Project the resulting tuples to attributes $A$ and $B$. 
SELECT A, B FROM R, S WHERE C;

- Nested loops:
  - for each tuple $t_1$ in $R$
    - for each tuple $t_2$ in $S$
      - if the attributes in $t_1$ and $t_2$ satisfy $C$
        - output the tuples involving attributes $A$ and $B$. 

Conversion to relational algebra:

1. Compute $R \times S$.
2. Apply the selection operator $\sigma_C()$ to $R \times S$.
3. Project the resulting tuples to attributes $A$ and $B$. 

T. M. Murali September 1, 2010 CS 4604: SQL Subqueries
SELECT A, B FROM R, S WHERE C;

- Nested loops:
  for each tuple \( t_1 \) in \( R \)
    for each tuple \( t_2 \) in \( S \)
      if the attributes in \( t_1 \) and \( t_2 \) satisfy \( C \)
        output the tuples involving attributes \( A \) and \( B \).

- Conversion to relational algebra: \( \pi_{A,B}(\sigma_C(R \times S)) \).
  1. Compute \( R \times S \).
  2. Apply the selection operator \( \sigma_C() \) to \( R \times S \).
  3. Project the resulting tuples to attributes \( A \) and \( B \).
Motivation for Subqueries

- Find the name of the professor who teaches “CS 4604.”
Motivation for Subqueries

- Find the name of the professor who teaches “CS 4604.”
  
  ```sql
  SELECT Name
  FROM Professors, Teach
  WHERE (PID = ProfessorPID) AND (Number = '4604') AND (DeptName = 'CS');
  ```
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?
SQL Subquery For Example

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

$\text{EXISTS } R$ is true if and only if $R$ contains at least one tuple.

$\text{IN } R$ is true if and only if $t$ equals a tuple in $R$.

$\text{t } > \text{ ALL } R$ 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 $\neq$, $R$ need not be unary.

$\text{t } > \text{ ANY } R$ (which is unary) is true if and only if $t$ is greater than at least one value in $R$.

We can use $\text{NOT}$ to negate $\text{EXISTS}$, $\text{ALL}$, and $\text{ANY}$. 
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.
- $\text{EXISTS } R$ is true if and only if $R$ contains at least one tuple.
- $t \text{ IN } R$ is true if and only if $t$ equals a tuple in $R$. 

- $t > \text{ ALL } R$ 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.
- $t > \text{ ANY } R$ (which is unary) is true if and only if $t$ is greater than at least one value in $R$.
- We can use $\text{NOT}$ to negate $\text{EXISTS}$, $\text{ALL}$, and $\text{ANY}$. 
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.
- $\text{EXISTS } R$ is true if and only if $R$ contains at least one tuple.
- $t \text{ IN } R$ is true if and only if $t$ equals a tuple in $R$.
- $t > \text{ ALL } R$ 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 $\neq$, $R$ need not be unary.
- $t > \text{ ANY } R$ (which is unary) is true if and only if $t$ is greater than at least one value in $R$.
- We can use $\text{NOT}$ to negate $\text{EXISTS}$, $\text{ALL}$, and $\text{ANY}$. 

T. M. Murali September 1, 2010 CS 4604: SQL Subqueries
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 $R$.
- $t > \text{ALL } R$ 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.
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 $R$.
- $t > \text{ALL } R$ 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.
- $t > \text{ANY } R$ (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**.
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.
- \( \text{EXISTS} \ R \) is true if and only if \( R \) contains at least one tuple.
- \( t \ \text{IN} \ R \) is true if and only if \( t \) equals a tuple in \( R \).
- \( t > \text{ALL} \ R \) 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.
- \( t > \text{ANY} \ R \) (which is unary) is true if and only if \( t \) is greater than at least one value in \( R \).
- We can use \( \text{NOT} \) to negate EXISTS, ALL, and ANY.
Subqueries Using Conditions

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

```sql
SELECT DeptName
FROM Take
WHERE StudentPID IN
  (SELECT PID
   FROM Students
   WHERE Name = 'Suri');
```
Subqueries Using Conditions

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

```sql
SELECT DeptName
FROM Take
WHERE StudentPID IN
  (SELECT PID
   FROM Students
   WHERE Name = 'Suri')
;
```
Correlated Subqueries

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

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

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

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

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

➤ Evaluate query by looping over tuples of First and for each tuple, evaluate the subquery.
Evaluating Correlated Subqueries

```sql
SELECT CourseName
FROM Courses AS First
WHERE CourseName IN
  (SELECT CourseName
   FROM Courses
   WHERE (Number <> First.Number)
   OR (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 \textit{FROM} clauses

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

\textbf{Old way:}
\begin{verbatim}
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');
\end{verbatim}
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 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');
```
Professors who have Taught 'Suri'

“Find the (names of (Professors who have taught (courses taken by student with name 'Suri'))).”
Professors who have Taught 'Suri'

“Find the (names of (Professors who have taught (courses taken by student with name 'Suri'))).”

SELECT Name
FROM Professors
WHERE PID IN
  (SELECT ProfessorPID
   FROM Teach
   WHERE (Number, DeptName) IN
     (SELECT Number, DeptName
      FROM Take, Students
      WHERE (StudentPID = PID) AND
        (Students.Name = 'Suri')))
Professors who have Taught 'Suri'

“Find the (names of (Professors who have taught (courses taken by (student with first name 'Suri'))))).”
Professors who have Taught 'Suri'

“Find the (names of (Professors who have taught (courses taken by (student with first name 'Suri'))))).”

```sql
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')));
```