More SQL Select

Database Systems Lecture 8
Natasha Alechina

In This Lecture

- More SQL Select
- Aliases
- ‘Self-joins’
- Subqueries
- IN, EXISTS, ANY, ALL
- For more information
  - Connoly and Begg Chapter 5
  - Ullman and Widom Chapter 6.3.

More SQL SELECT Overview

```
SELECT [DISTINCT | ALL] <column-list>
FROM <table-names>
[WHERE <condition>]
[ORDER BY <column-list>]
[GROUP BY <column-list>]
[HAVING <condition>]
 {} - optional, | - or
```

Aliases

- Aliases rename columns or tables to
  - Make names more meaningful
  - Make names shorter and easier to type
  - Resolve ambiguous names

- Two forms:
  - Column alias
    SELECT column AS newName...
  - Table alias
    SELECT ...
    FROM table AS newName

This ‘AS’ is optional, but Oracle doesn’t accept it at all

Example

```
Employee
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>John</td>
</tr>
<tr>
<td>124</td>
<td>Mary</td>
</tr>
</tbody>
</table>

WorksIn
<table>
<thead>
<tr>
<th>ID</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Marketing</td>
</tr>
<tr>
<td>124</td>
<td>Sales</td>
</tr>
<tr>
<td>124</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

SELECT E.ID AS empID, E.Name, W.Dept
FROM Employee E, WorksIn W
WHERE E.ID = W.ID
```

Example

```
Employee
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>John</td>
</tr>
<tr>
<td>124</td>
<td>Mary</td>
</tr>
</tbody>
</table>

WorksIn
<table>
<thead>
<tr>
<th>ID</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Marketing</td>
</tr>
<tr>
<td>124</td>
<td>Sales</td>
</tr>
<tr>
<td>124</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

SELECT E.ID AS empID, E.Name, W.Dept
FROM Employee E, WorksIn W
WHERE E.ID = W.ID
```
Aliases and 'Self-Joins'

Aliases can be used to copy a table, so that it can be combined with itself:

```sql
SELECT A.Name FROM Employee A, Employee B
WHERE A.Dept = B.Dept
AND B.Name = 'Andy'
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Marketing</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
</tr>
<tr>
<td>Peter</td>
<td>Sales</td>
</tr>
<tr>
<td>Andy</td>
<td>Marketing</td>
</tr>
<tr>
<td>Anne</td>
<td>Marketing</td>
</tr>
</tbody>
</table>

The result is the names of all employees who work in the same department as Andy.
Subqueries

- A **SELECT** statement can be nested inside another query to form a subquery.
- The results of the subquery are passed back to the containing query.
- E.g. get the names of people who are in Andy’s department:
  ```sql
  SELECT Name
  FROM Employee
  WHERE Dept =
  (SELECT Dept
   FROM Employee
   WHERE Name='Andy')
  ```

Options
- **IN** - checks to see if a value is in the set.
- **EXISTS** - checks to see if the set is empty or not.
- **ALL/ANY** - checks to see if a relationship holds for every/one member of the set.

**NOT IN**
- Using **IN** we can see if a given value is in a set of values.
- **NOT IN** checks to see if a given value is not in the set.
- The set can be given explicitly or from a subquery.

```
SELECT <columns>
FROM <tables>
WHERE <value> NOT IN <set>
```

(Using the example data:

<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Marketing</td>
<td>Chris</td>
</tr>
<tr>
<td>Mary</td>
<td>Marketing</td>
<td>Chris</td>
</tr>
<tr>
<td>Chris</td>
<td>Marketing</td>
<td>Jane</td>
</tr>
<tr>
<td>Peter</td>
<td>Sales</td>
<td>Jane</td>
</tr>
<tr>
<td>Jane</td>
<td>Management</td>
<td></td>
</tr>
</tbody>
</table>

```sql
SELECT * 
FROM Employee
WHERE Department IN ('Marketing', 'Sales')
```

```sql
SELECT * 
FROM Employee 
WHERE Name NOT IN (SELECT Manager 
                   FROM Employee)
```
More SQL SELECT

( NOT ) IN

• First the subquery
  SELECT Manager
  FROM Employee
  is evaluated giving
  Chris
  Chris
  Jane

• This gives
  SELECT *
  FROM Employee
  WHERE Name NOT IN (’Chris’, ’Jane’)

  Name  Department  Manager
  John  Marketing  Chris
  Mary  Marketing  Chris
  Peter  Sales  Jane

( NOT ) EXISTS

• Using EXISTS we see if there is at least one element in a set
  SELECT 
  FROM Employee
  WHERE NOT EXISTS (SELECT * FROM Employee E2
  WHERE E2.Name = E1.Manager)

  Name  Salary
  Mary  20,000
  John  15,000
  Jane  25,000
  Paul  30,000

( NOT ) EXISTS

• NOT EXISTS is true if the set is empty
• The set is always given by a subquery
  SELECT Columns
  FROM Tables
  WHERE NOT EXISTS (SELECT Columns
  FROM Tables
  WHERE <set>)

ANY and ALL

• ANY and ALL compare a single value to a set of values
• They are used with comparison operators like =, >, <, <=

  val = ANY (set) is true if there is at least one member of the set equal to the value
  val = ALL (set) is true if all members of the set are equal to the value

ALL

Find the names of the employee(s) who earn the highest salary
  SELECT Name
  FROM Employee
  WHERE Salary >= ALL (SELECT Salary
  FROM Employee)

ANY

Find the names of employee(s) who earn more than someone else
  SELECT Name
  FROM Employee
  WHERE Salary > ANY (SELECT Salary
  FROM Employee)
Word Searches

- Word Searches
  - Commonly used for searching product catalogues etc.
  - Want to be able to search by keyword
  - Want to be able to use word stemming for flexible searching

- For example: given a database of books,
  - Searching for “crypt” would return
    - “Cryptonomicon” by Neil Stephenson
    - “Applied Cryptography” by Bruce Schneier

To search we can use queries like

```sql
SELECT * FROM Items
WHERE itmID IN (SELECT itmID FROM ItemKey
                 WHERE keyID IN (SELECT keyID FROM Keywords
                                 WHERE keyWord LIKE 'crypt%'))
```

Word Searches

- Sometimes you need to search for a set of words
  - To find entries with all words you can link conditions with AND
  - To find entries with any of the words use OR

```sql
SELECT * FROM Items
WHERE itmID IN (SELECT itmID FROM ItemKey
                 WHERE keyID IN (SELECT keyID FROM Keywords
                                 WHERE keyWord LIKE 'word1%'))
AND
 itmID IN (SELECT itmID FROM ItemKey
             WHERE keyID IN (SELECT keyID FROM Keywords
                             WHERE keyWord LIKE 'word2%'))
```

Next Lecture

- Yet more SQL
  - ORDER BY
  - Aggregate functions
  - GROUP BY and HAVING
  - UNION etc.
- For more information
  - Connoly and Begg Chapter 5
  - Ullman and Widom Chapter 6.4