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

SQL SELECT Overview

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

( [] - optional, | - or )
```

More SQL SELECT

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

```
<table>
<thead>
<tr>
<th>empID</th>
<th>Name</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>John</td>
<td>Marketing</td>
</tr>
<tr>
<td>124</td>
<td>Mary</td>
<td>Sales</td>
</tr>
<tr>
<td>124</td>
<td>Mary</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>Employee</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A Name</td>
<td>Dept</td>
</tr>
<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')
```

More SQL SELECT

Subqueries

- Often a subquery will return a set of values rather than a single value
- You can't directly compare a single value to a set

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

More SQL SELECT

(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

Examples:

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

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

More SQL SELECT

More SQL SELECT

More SQL SELECT

More SQL SELECT
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

More SQL SELECT

(NOT) EXISTS

• Using EXISTS we see
  if there is at least
  one element in a set
• NOT EXISTS is true if
  the set is empty

  SELECT <columns>
  FROM <tables>
  WHERE EXISTS <set>

  SELECT <columns>
  FROM <tables>
  WHERE NOT EXISTS <set>

More SQL SELECT

(NOT) EXISTS

Employee

<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>Sales</td>
<td>Jane</td>
</tr>
<tr>
<td>Peter</td>
<td>Management</td>
<td>Jane</td>
</tr>
</tbody>
</table>

More SQL SELECT

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

More SQL SELECT

ALL

Find the names of the employee(s) who earn the highest salary

  SELECT Name
  FROM Employee
  WHERE Salary >= ALL
    (SELECT Salary
     FROM Employee)

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

More SQL SELECT

ANY

Find the names of employee(s) who earn more than someone else

  SELECT Name
  FROM Employee
  WHERE Salary >
    ANY
      (SELECT Salary
       FROM Employee)

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

More SQL SELECT
Word Searches

• Word Searches
  • Commonly used for searching product catalogues etc.
  • Want to be able to use word stemming for flexible searching
• For example: given a database of books,
  • Searching for “automata” should return everything with “automata” somewhere in the title

To search we can use queries like

```sql
SELECT * FROM Book
WHERE Title LIKE '%Automata%';
```

which returns all titles which have a substring `Automata`. % stands for `any other string`.

Next Lecture 4 March!

• No lectures the week of the 22nd Feb
• 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