Өгөгдлийн сангийн үндэс (CSII202 - 3 кр)
Database Systems

Lecture 5: More SQL SELECT

МУИС, ХШУИС, МКУТ-ийн багш

Док. Довдонгийн Энхзол

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.

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

ID	Name
123	John
124	Mary

WorksIn

ID	Dept
123	Marketing
124	Sales
124	Marketing

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

Example

empID	Name	Dept
123	John	Marketing
124	Mary	Sales
124	Mary	Marketing

```
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:

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

Employee

Name	Dept
John	Marketing
Mary	Sales
Peter	Sales
Andy	Marketing
Anne	Marketing

Aliases and Self-Joins

Employee A

A

Name	Dept
John	Marketing
Mary	Sales
Peter	Sales
Andy	Marketing
Anne	Marketing

Employee B

B

Name	Dept
John	Marketing
Mary	Sales
Peter	Sales
Andy	Marketing
Anne	Marketing

Aliases and Self-Joins

```
SELECT … FROM Employee A, Employee B …
```

A.Name	A.Dept	B.Name	B.Dept
John	Marketing	John	Marketing
Mary	Sales	John	Marketing
Peter	Sales	John	Marketing
Andy	Marketing	John	Marketing
Anne	Marketing	John	Marketing
John	Marketing	Mary	Sales
Mary	Sales	Mary	Sales
Peter	Sales	Mary	Sales
Andy	Marketing	Mary	Sales
M ore A			Sales

Aliases and Self-Joins

```
SELECT … FROM Employee A, Employee B
  WHERE A.Dept = B.Dept
```

A.Name	A.Dept	B.Name	B.Dept
John	Marketing	John	Marketing
Andy	Marketing	John	Marketing
Anne	Marketing	John	Marketing
Mary	Sales	Mary	Sales
Peter	Sales	Mary	Sales
Mary	Sales	Peter	Sales
Peter	Sales	Peter	Sales
John	Marketing	Andy	Marketing
Andy	Marketing	Andy	Marketing
Anne	Marketing		Marketing

Aliases and Self-Joins

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

A.Name	A.Dept	B.Name	B.Dept
John	Marketing	Andy	Marketing
Andy	Marketing	Andy	Marketing
Anne	Marketing	Andy	Marketing

Aliases and Self-Joins

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

A.Name
John
Andy
Anne

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:

```
SELECT Name
  FROM Employee
 WHERE Dept =
  (SELECT Dept
   FROM Employee
   WHERE Name='Andy')
```

Subqueries

```
SELECT Name
 FROM Employee
 WHERE Dept =
 (SELECT Dept
   FROM Employee
   WHERE
     Name='Andy')
```

- First the subquery is evaluated, returning the value 'Marketing'
- This result is passed to the main query

```
SELECT Name
   FROM Employee
  WHERE Dept =
    'Marketing'
```

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

(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>
    IN <set>
```

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

(NOT) IN

Employee

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

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

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

(NOT) IN

Employee

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

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

(NOT) IN

- First the subquery

```
SELECT Manager
   FROM Employee
```

- is evaluated giving

Manager
Chris
Chris
Jane
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
- NOT EXISTS is true if the set is empty
- The set is always given by a subquery

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

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

(NOT) EXISTS

Employee

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

```
SELECT *
 FROM  Employee E1
 WHERE EXISTS (
   SELECT * FROM
     Employee E2
     WHERE E2.Name =
           E1.Manager)
```

Name	Department	Manager
Chris	Marketing	Jane
Jane	Management	

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

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

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

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

ANY

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

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
    - “*Crypt*onomicon” by Neil Stephenson
    - “Applied *Crypt*ography” by Bruce Schneier

Word Searches

- To do a word search we can keep
  - A table of items to be searched
  - A table of keywords
  - A linking table saying which keywords belong to which items

Word Searches

To search we can use queries like

```
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

```
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