SQL SELECT II

Database Systems

Michael Pound

This Lecture

- More SQL SELECT
  - Aliases
  - 'Self-Joins'
  - Subqueries
  - IN, EXISTS, ANY, ALL
  - LIKE
- Further reading
  - The Manga Guide to Databases, Chapter 4
  - Database Systems, Chapter 6

Last Lecture

- WHERE Clauses
- SELECT from multiple tables

```
SELECT * FROM TA, TB;
```

- JOINs
  - CROSS JOIN (Cartesian Product)

```
SELECT * FROM TA CROSS JOIN TB;
```

  - INNER JOIN (Specifies a column or condition)

```
SELECT * FROM TA INNER JOIN TB USING (Col1);
SELECT * FROM TA INNER JOIN TB ON (α);
```

  - NATURAL JOIN (Compares columns with identical names)

```
SELECT * FROM TA NATURAL JOIN TB;
```

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
  - Can make names more meaningful
  - Can shorten names, making them easier to use
  - Can resolve ambiguous names
- Two forms:
  - Column alias

```
SELECT column [AS]
  newName
```

  - Table alias

```
SELECT table [AS]
  newName
```

([] *optional*)

Alias Example

Employee

ID	First
123	John
124	Mary

WorksIn

ID	Department
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
```

Note: You cannot use a column alias in a WHERE clause

Alias Example

empID	Name	Department
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
```

Note: You normally cannot use a column alias in a WHERE clause

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 **Employee B**

A

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

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
Anne	Marketing	Mary	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	Andy	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
- For example, retrieve a list of 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 'Marketing'
- This value 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
- We cannot directly compare a single value to a set. Doing so will result in an error
- Options for handling sets
  - IN – checks to see if a value is in a set
  - EXISTS – checks to see if a set is empty
  - ALL/ANY – checks to see if a relationship holds for every/one member of a set
  - NOT can be used with any of the above

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 can be produced in a subquery

```
SELECT <columns>
  FROM <tables>
  WHERE <value>
     IN <set>

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

IN

Employee

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

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

Employee

Name	Dept	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	Deptartment	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Peter	Sales	Jane

EXISTS

- Using EXISTS we see that 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>
```

EXISTS

Employee

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

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

Name	Dept	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 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

Name
Paul

- 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

Name
Mary
Jane
Paul

- Find the names of the 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.
  - Need to search by keywords
  - Might need to use partial keywords
- For example: Given a database of books, searching for "crypt" might return
  - "*Cryptonomicon*" by Neil Stephenson
  - "*Applied Cryptographer*" by Bruce Schneier

LIKE

- We can use the **`LIKE`** keyword to perform string comparisons in queries
- Like is not the same as '=' because it allows wildcard characters
- It is not normally case sensitive

```
SELECT * FROM books
 WHERE bookName LIKE "%crypt%";
```

LIKE

- The '%' character can represent any number of characters, including none

```
bookName LIKE "crypt%"
```

- Will return "Cryptography Engineering" and "Cryptonomicon" but not "Applied Cryptography"
- The '_' character represents exactly one character

```
bookName LIKE "cloud_"
```

- Will return "Clouds" but not "Cloud" or "Cloud Computing"

LIKE

- Sometimes you might 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 words use OR

```
SELECT * FROM
  books
 WHERE bookName
  LIKE "%crypt%";
 OR bookName LIKE
  "%cloud%";
```

Examples

Student

sID	sName	sAddress	sYear
1	Smith	5 Arnold Close	2
2	Brooks	7 Holly Avenue	2
3	Anderson	15 Main Street	3
4	Evans	Flat 1a, High Street	2
5	Harrison	Newark Hall	1
6	Jones	Southwell Hall	1

Module

mCode	mCredits	mTitle
G51DBS	10	Database Systems
G51PRG	20	Programming
G51IAI	10	Artificial Intelligence
G52ADS	10	Algorithms

Enrolment

sID	mCode
1	G52ADS
2	G52ADS
5	G51DBS
5	G51PRG
5	G51IAI
4	G52ADS
6	G51PRG
6	G51IAI

Examples

- Write SQL statements to do the following:
  - Find a list of students in the 2nd or 3rd year
  - Find a list of student IDs and Names for students studying G52ADS, but without using a JOIN
  - Find a list of names of any students who are enrolled on at least one module alongside 'Evans'

Next Lecture

- More SQL SELECT
  - ORDER BY
  - Aggregate functions
  - GROUP BY and HAVING
  - UNION
- Further reading
  - The Manga Guide to Databases, Chapter 4
  - Database Systems, Chapter 6