SQL Lecture II

G51DBI – Databases and Interfaces

Yorgos Tzimiropoulos
yorgos.tzimiropoulos@nottingham.ac.uk

This Lecture

- SQL SELECT
  - WHERE Clauses
  - SELECT from multiple tables
- More SQL SELECT
  - Aliases
  - ‘Self-Joins’
  - Subqueries
  - IN, EXISTS, ANY, ALL

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*)

SELECT from Multiple Tables

- Often you need to combine information from two or more tables
- You can produce the effect of a Cartesian product using:

```
SELECT * FROM Table1,
  Table2
```

- If the tables have columns with the same name, ambiguity will result
- This can be resolved by referencing columns with the table name:

```
TableName.ColumnName
```

SELECT from Multiple Tables

```
SELECT
   First, Last, Mark
FROM
   Student, Grade
```

Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	
S106	Mar	
S107	John	

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

SELECT from Multiple Tables

```
SELECT ... FROM Student, Grade WHERE ...
```

ID	First	Last	ID	Code	Mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S103	John	Smith	S104	PR1	68
S103	John	Smith	S104	IAI	65
S103	John	Smith	S106	PR2	43
S103	John	Smith	S107	PR1	76
S103	John	Smith	S107	PR2	60
S103	John	Smith	S107	IAI	35
S104	Mary	Jones	S103	DBS	72
S104	Mary	Jones	S103	IAI	58
S104	Mary	Jones	S104	PR1	68
S104	Mary	Jones	S104	IAI	65

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]
  new-col-name
```

  - Table alias

```
SELECT * FROM table [AS]
  new-table-name
```

([] *optional*)

Subqueries

- Use the result of a query as input to a new query
- The results of the subquery are passed back to the containing query
- Reminiscent of ??

- *"Find the name and gpa of the student with the highest sid" who has enrolled to some module*
- Find the highest sid from grade table
- Use that as input to a second query involving the student table

Subqueries

```
SELECT sName, gpa
  FROM Student
 WHERE sID =
   (SELECT MAX(sID)
    from Grade );
```

- First the subquery is evaluated, returning 6
- This value is passed to the main query

```
SELECT sName, gpa
  FROM Student
 WHERE sID = 6;
```

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;
```

EXISTS

- Using EXISTS we can see whether there is at least one element in a given 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;
```

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

This Lecture

- SQL SUBQUERIES
  - More examples
  - SUBQUERIES in the FROM clause
  - SUBQUERIES in the SELECT clause

Subqueries

/* “Find all students with at least one mark > 60 “ */

/* Find all students whose marks for all modules > 55 */

- Using IN
- Using Exists

Subqueries

/* find the student name and sid with the highest mark */

Subqueries

/* find the student name and sid with the highest mark */

- First find highest mark (from grade)
- Then find sid for that mark (from grade)
- Then find student name (from Student)
- **Take home message**: multiple nested subqueries are allowed

Subqueries

/* find the student with the highest mark (and actually return that mark) */

Subqueries

/* find the student with the highest mark (and actually return that mark) */

- **Join back** with Grade
- **Take home message**: subqueries can be combined with cross product
- Subqueries in the From clause as an alternative

Subqueries

/* list all marks along with the average mark */

Subqueries

/* list all marks along with the average mark */

- Find average mark first
- Select mark from grade and use subquery in the Select Clause
- **Take home message**: subqueries in the Select clause must return a single value

Subqueries

/* List the student names and number of modules that each student has registered to */

Subqueries

/* List the student names and number of modules that each student has registered to */

- Find the number of students that each student has registered to

Thanks for your attention!

Any questions??