(Web) Application Development – With Ian
Week 3
SQL with Multiple Tables

Join

The join operation allows you to combine related rows of data found in two tables into a single result set.

It works similarly to a query we did last week:

```
SELECT hr.EmployeeID, hr.Title, pc.FirstName, pc.LastName
FROM HumanResources.Employee hr, Person.Contact pc
WHERE hr.EmployeeID = pc.ContactID
```

This query grabs the **EmployeeID** and **Title** columns from **HumanResources.Employee**, and the **FirstName** & **LastName** columns from **Person.Contact**, where the EmployeeID and ContactID are equal to 5.

Note the `SELECT hr.EmployeeID` short form. I'm using "pc" to refer to Person.Contact. That alias is set: `FROM HumanResources.Employee hr`.

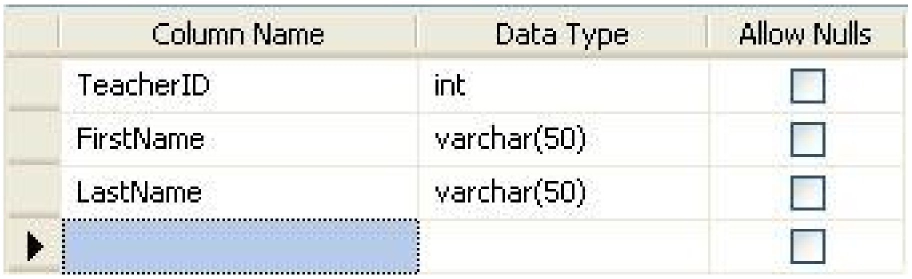
Using a join would make this query easier to understand:

```
SELECT EmployeeID, hr.Title, FirstName, LastName
FROM HumanResources.Employee hr JOIN Person.Contact pc
ON hr.EmployeeID = pc.ContactID
```

Joining two tables returns rows where the ON condition exists.

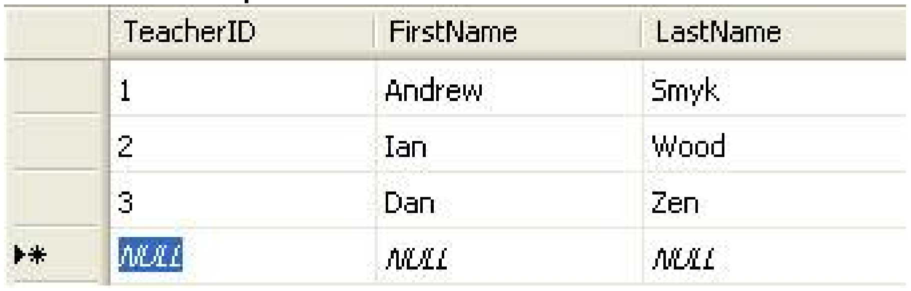
Before continuing, create the **School** database then add the tables as you see them on the next two pages.

Table Structure: Teachers



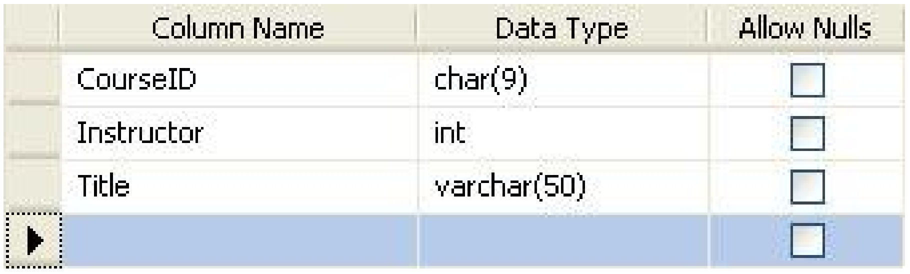
	Column Name	Data Type	Allow Nulls
	TeacherID	int	☐
	FirstName	varchar(50)	☐
	LastName	varchar(50)	☐
▶			☐

Table Teachers Populated:



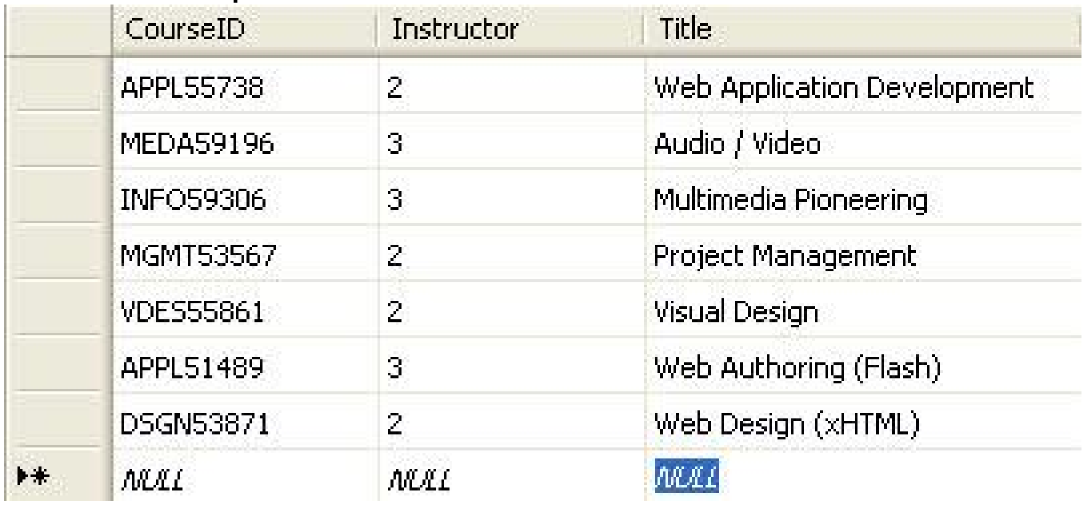
	TeacherID	FirstName	LastName
	1	Andrew	Smyk
	2	Ian	Wood
	3	Dan	Zen
▶*	NULL	NULL	NULL

Table Structure: Courses



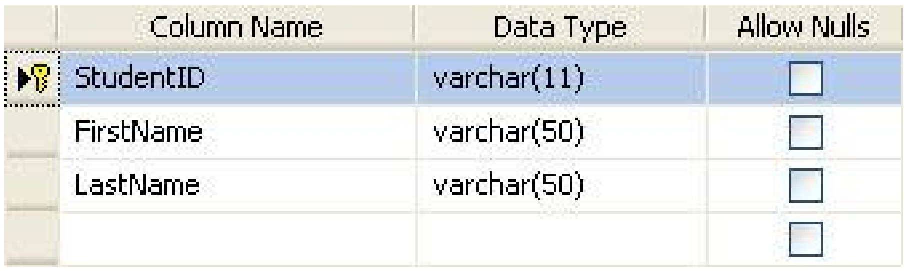
	Column Name	Data Type	Allow Nulls
	CourseID	char(9)	☐
	Instructor	int	☐
	Title	varchar(50)	☐
▶			☐

Table Courses Populated



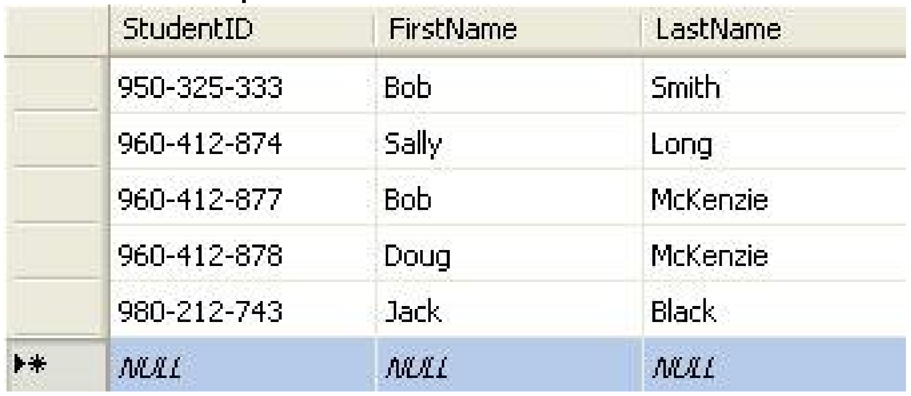
	CourseID	Instructor	Title
	APPL55738	2	Web Application Development
	MEDA59196	3	Audio / Video
	INFO59306	3	Multimedia Pioneering
	MGMT53567	2	Project Management
	VDES55861	2	Visual Design
	APPL51489	3	Web Authoring (Flash)
	DSGN53871	2	Web Design (xHTML)
▶*	NULL	NULL	NULL

Table Structure: Students



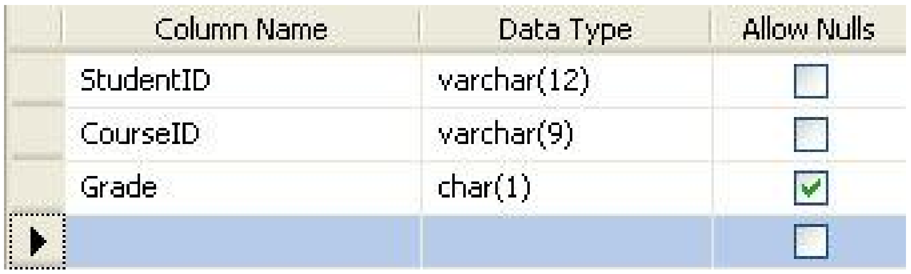
Column Name	Data Type	Allow Nulls
StudentID	varchar(11)	☐
FirstName	varchar(50)	☐
LastName	varchar(50)	☐
		☐

Table Students Populated



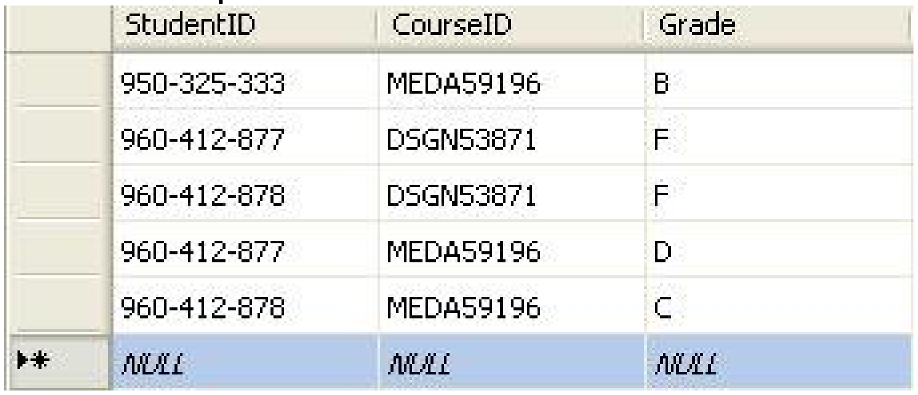
StudentID	FirstName	LastName
950-325-333	Bob	Smith
960-412-874	Sally	Long
960-412-877	Bob	McKenzie
960-412-878	Doug	McKenzie
980-212-743	Jack	Black
NULL	*NULL*	*NULL*

Table Structure: Grades



Column Name	Data Type	Allow Nulls
StudentID	varchar(12)	☐
CourseID	varchar(9)	☐
Grade	char(1)	✓
		☐

Table Grades Populated:



StudentID	CourseID	Grade
950-325-333	MEDA59196	B
960-412-877	DSGN53871	F
960-412-878	DSGN53871	F
960-412-877	MEDA59196	D
960-412-878	MEDA59196	C
NULL	*NULL*	*NULL*

Review:

1)
```
SELECT * FROM Teachers
```

2)
```
SELECT FirstName FROM Teachers
```

3)
```
SELECT FirstName, LastName FROM Teachers ORDER BY LastName Desc
```

4)
```
SELECT CourseID, Title
FROM Courses
WHERE Instructor = 2
```

New:

1)
Using a join to get the list of class codes, instructors, and course titles:

```
SELECT CourseID, FirstName, LastName, Title
FROM Teachers t JOIN Courses c
ON t.TeacherID = c.Instructor
```

Notice poor Andrew doesn't show up. He's not actively teaching this year, so he's not assigned to any courses. Thus he is excluded in the join.

2)
Lets put some order to that list:
```
SELECT CourseID, FirstName, LastName, Title
FROM Teachers t JOIN Courses c
ON t.TeacherID = c.Instructor
ORDER BY LastName
```

3)
This query will get the same information as, except only returning courses that I teach:

```
SELECT CourseID, FirstName, LastName, Title
FROM Teachers t JOIN Courses c
ON t.TeacherID = c.Instructor
WHERE FirstName = 'Ian'
```

Both of these joins are examples of ***equi-joins***, joins where the ON clause uses equals. Unless otherwise specified, such as an ***outer-join***, joins are generally assumed to be equi-joins.

4)
In SQL Server 2005 you'll get the same results using INNER JOIN as with regular JOIN

```
SELECT CourseID, FirstName, LastName, Title
FROM Teachers t INNER JOIN Courses c
ON t.TeacherID = c.Instructor
```

---

5)
Self joins allow you to join a table to itself. This is useful when you need to find information that's based on more than one column (note: this can be a hard concept to get used to!):

```
SELECT 'GradeRank' = x.StudentID + ' got a better grade than ' +
y.StudentID
FROM Grades AS x, Grades AS y
WHERE y.CourseID = 'MEDA59196'
AND x.Grade < y.Grade
```

6)
This can then be ordered by the grade:

```
SELECT 'GradeRank' = x.StudentID + ' got a better grade than ' +
y.StudentID
FROM Grades AS x, Grades AS y
WHERE y.CourseID = 'MEDA59196'
AND x.Grade < y.Grade
ORDER BY y.Grade
```

** If 5,6 are unclear don't worry. We'll be doing more examples of this type of join in the coming weeks with data it's easier to see.

---

7)
Nesting joins allows you to join multiple tables.

Say you wanted to know the <u>name</u> of any students who <u>failed</u> AND the <u>title of the course</u> they failed:

Lets start by joining the Students with their Grades:

```
SELECT FirstName, LastName, Grade
FROM Students stu JOIN Grades g
ON stu.StudentID = g.StudentID
```

```
Now add a where clause to restrict it to people who failed:
    SELECT FirstName, LastName, Grade
    FROM Students stu JOIN Grades g
    ON stu.StudentID = g.StudentID
    WHERE Grade = 'F'
```

Now do the 3 table join:

```
SELECT FirstName, LastName, Grade, Title
FROM Students stu JOIN (Grades grd JOIN Courses c
ON grd.CourseID = c.CourseID)
ON stu.StudentID = grd.StudentID
```

- Brackets set precedence
- So the Grades are joined to the Course FIRST
- THEN Students are then joined to the Grades

Note: building up your queries in steps will let you find mistakes easier.

8a)
What if we want to join two tables together, but not all records in table A have a match in B? Remember when we joined Courses and Teachers in example 1 we lost Andrew. He isn't listed in Courses so he didn't show up.

```
SELECT FirstName, LastName, Grade
FROM Students stu LEFT OUTER JOIN Grades grd
ON stu.StudentID = grd.StudentID
```

Students is the "left" table, Grades the "right". So precedence is given to the Left in this case, meaning all rows from the left will be returned even if they don't have a match in the right.

8b)

```
SELECT FirstName, LastName, Grade
FROM Students stu RIGHT OUTER JOIN Grades grd
ON stu.StudentID = grd.StudentID
```

Does it the other way around.

9) Counting

If you want to know the total number of records, or some subset of records, you can use the ***count* function**.

```
SELECT COUNT(*)
FROM Courses

SELECT COUNT(*) AS [Number of Courses]
FROM Courses
```

To Try On Your Own:

1) Get a list of all the student names and ID #'s.

2) Make the list is #1 display alphabetically by last name.

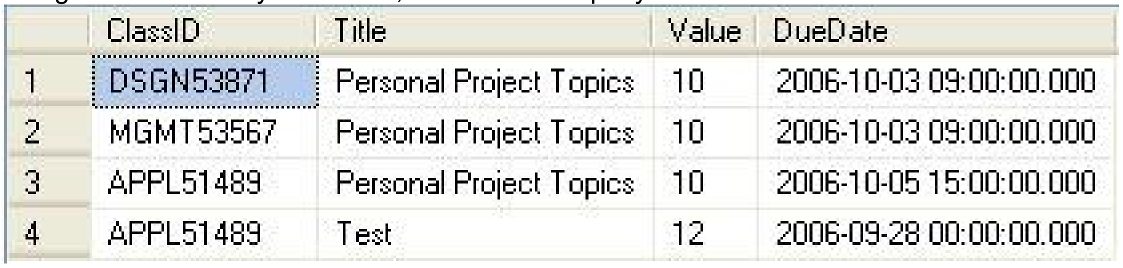
3a) Create and Assignments table with AssignmentID, Title, and Value columns.

3b) Create a Deliverables table with DeliverableID, ClassID, AssignmentID, and DueDate columns.

3c) Populate some values into both tables. Make sure you use valid ClassID's, etc. (If the data relates to the tables you've already created, use matching data)

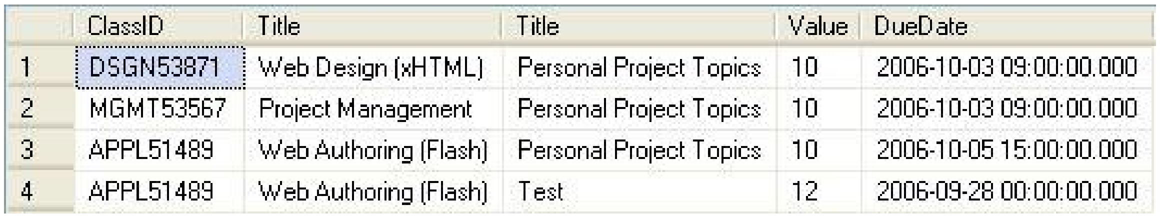
(Note: it could be argued that these 2 new tables are not as normalized as they could be. That's fine, it'll make life easier to see what's going on right now.)

4) Using the new tables you created, write the SQL query that will create this table:



	ClassID	Title	Value	DueDate
1	DSGN53871	Personal Project Topics	10	2006-10-03 09:00:00.000
2	MGMT53567	Personal Project Topics	10	2006-10-03 09:00:00.000
3	APPL51489	Personal Project Topics	10	2006-10-05 15:00:00.000
4	APPL51489	Test	12	2006-09-28 00:00:00.000

5) Using nested joins, add on to the above to create this result:



	ClassID	Title	Title	Value	DueDate
1	DSGN53871	Web Design (xHTML)	Personal Project Topics	10	2006-10-03 09:00:00.000
2	MGMT53567	Project Management	Personal Project Topics	10	2006-10-03 09:00:00.000
3	APPL51489	Web Authoring (Flash)	Personal Project Topics	10	2006-10-05 15:00:00.000
4	APPL51489	Web Authoring (Flash)	Test	12	2006-09-28 00:00:00.000