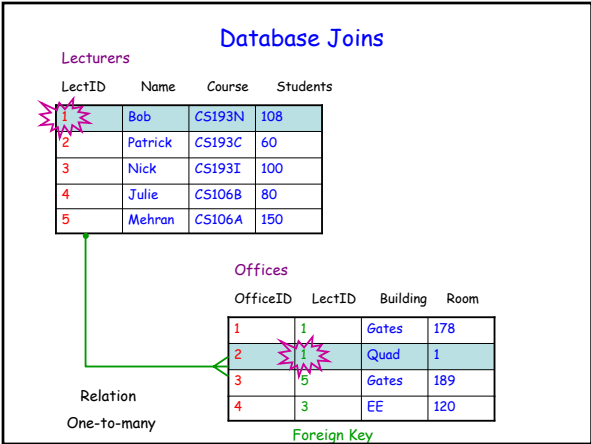
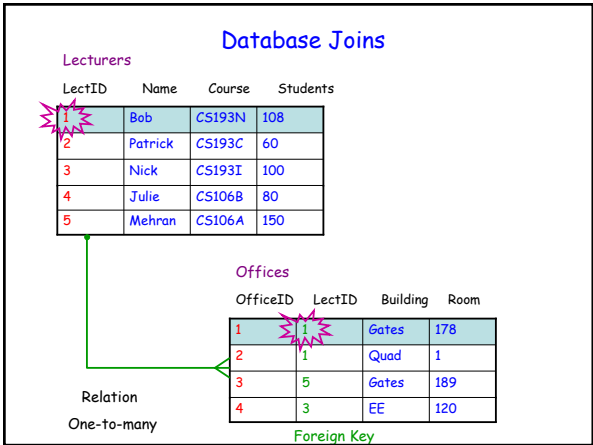
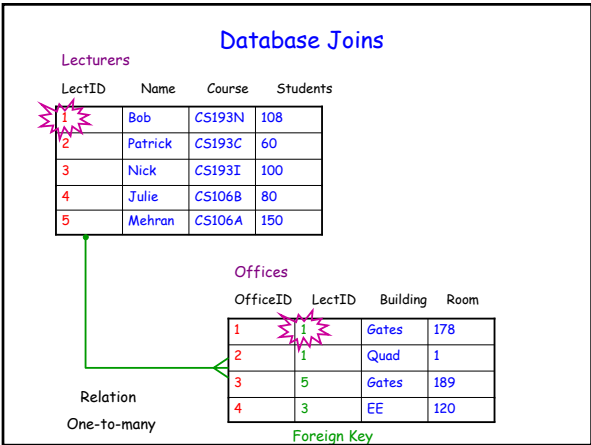


We will **NOT** have a Mid-Term or Intra-Quarter Quiz.

New grade breakdown:

Assignments: 70%
Final Exam: 30%

- Topics for today
- Database Joins
 - Typed DataSets
 - Data Binding
 - Database Updating



```
SELECT *
FROM Lecturers INNER JOIN OFFICES ON Lecturers.LectID = Offices.LectID
ORDER BY Offices.LectID
```

LectID	Name	Course	Students	OfficeID	LectID	Building	Room
1	Bob	CS193N	80	1	1	Gates	178
2	Bob	CS193N	80	11	1	Math	124
3	Bob	CS193N	80	9	1	Quad	1
4	Patrick	CS193C	60	12	2	EE	1
5	Patrick	CS193C	60	2	2	Gates	100
6	Nick	CS193I	100	3	3	EE	120
7	Nick	CS193I	100	13	3	Ariaga	86
8	Julie	CS106B	80	4	4	EE	121
9	Mehran	CS106A	150	5	5	Gates	189
10	Mehran	CS106A	150	14	5	Mudd	101
11	Maggie	CS103A	120	6	6	Gates	184
12	Jerry	CS106X	60	7	7	Chem	200
13	Jerry	CS106X	60	15	7	Gates	189
14	Gene	CS137	25	10	8	Gates	480
15	Gene	CS137	25	8	8	Gates	280

Demo of
Microsoft SQL Server Management Studio Express

Search on this term
and download

Lecturers				Name
1	Bob	CS193N	108	
2	Patrick	CS193C	60	
3	Nick	CS193I	100	
4	Julie	CS106B	80	
5	Mehran	CS106A	150	

Offices		Building	
1	1	Gates	178
2	1	Quad	1
3	5	Gates	189
4	3	EE	120

Grants		Agency	
1	1	NSF	
2	1	DARPA	
3	4	NSF	

Advisees			
1	1	Bala	Ramesh
2	1	Burr	Brandon
3	2	Srisu	Charles

```

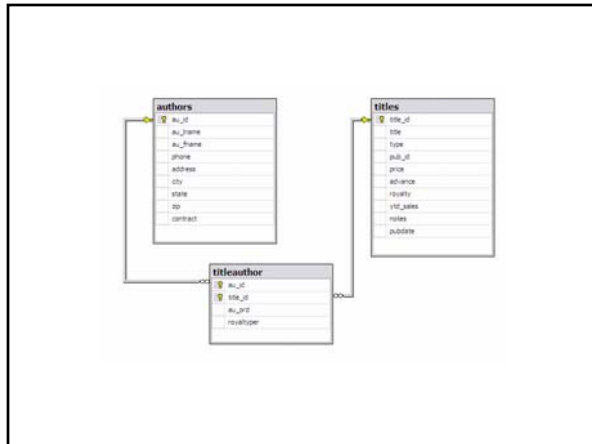
SELECT Name
FROM Lecturers
INNER JOIN Offices ON Lecturers.LectID = Offices.LectID
INNER JOIN Grants ON Lecturers.LectID = Grants.LectID
WHERE Building = 'Gates' AND Agency = 'NSF'
    
```

```

SELECT Name
FROM Lecturers
INNER JOIN Offices ON Lecturers.LectID = Offices.LectID
INNER JOIN Grants ON Lecturers.LectID = Grants.LectID
WHERE Building = 'Gates' AND Agency = 'NSF'
    
```

Name

Bob



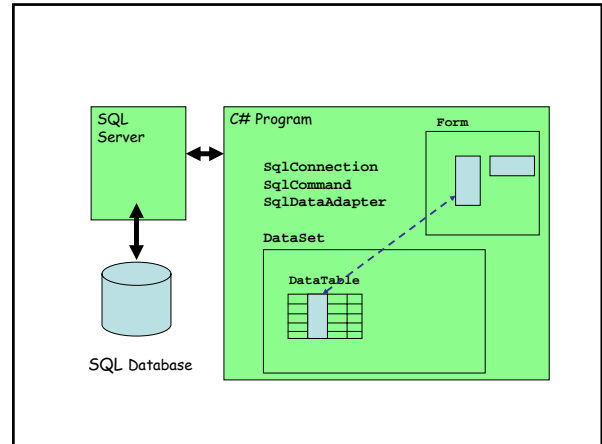
```

SELECT au_fname, au_lname, title
FROM authors
JOIN titleauthor
ON authors.au_id = titleauthor.au_id
JOIN titles
ON titleauthor.title_id = titles.title_id
ORDER BY title
    
```

```

SELECT au_fname, au_lname, title
FROM authors
JOIN titleauthor
ON authors.au_id = titleauthor.au_id
JOIN titles
ON titleauthor.title_id = titles.title_id
ORDER BY title
    
```

au_fname	au_lname	title
Cheryl	Caen	But Is User Friendly?
Cheryl	MacFether	Computer Phobic AND NonPhobic Individuals: Beha...
Lisa	Karen	Computer Phobic AND NonPhobic Individuals: Beha...
Michael	O'Leary	Cooking with Computers: Surprisingly Balance Sheets
Deanna	MacFether	Cooking with Computers: Surprisingly Balance Sheets
Chakne	Lockley	Emotional Security: A New Algorithm
Reginald	Blotcher-Hall	Fifty Years in Buckingham Palace Kitchens
Anne	Ringer	Is Anger the Enemy?
Albert	Ringer	Is Anger the Enemy?
Albert	Ringer	Like Without Fear
Chakne	Lockley	Niel Equates
Sylvia	Parley	Dinner, Drinks, and Sails: Cooking Secrets of the M...
Johnson	John	Photographing the Disasters: Five Case Studies
Ann	Dul	Secrets of Silicon Valley
Sheryl	Hunter	Secrets of Silicon Valley
Irvin	delCastillo	Silicon Valley Gastronomic Treats
Dean	Shaugh	Straight Talk About Computers
Michael	O'Leary	Suits, Anyone?
Burt	Gendelso	Suits, Anyone?
Hilko	Valjondis	Suits, Anyone?
Marissa	Green	The Busy Executive's Database Guide
Abraham	Bennet	The Busy Executive's Database Guide
Michel	Defrance	The Gourmet Microphone
Ann	Ringer	The Gourmet Microphone
Marissa	Green	You Can Conquer Computer Stress!



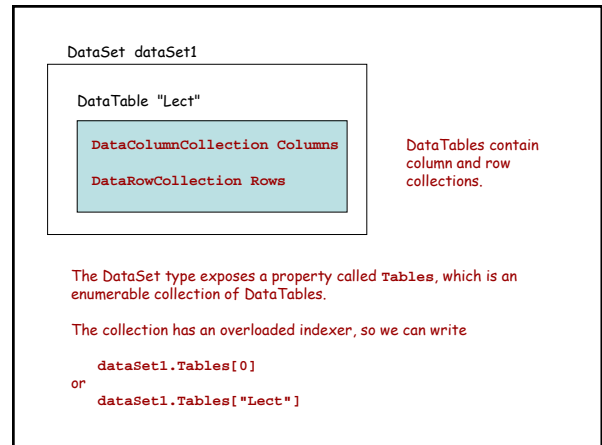
```

SQL:

SELECT LectID, Name, Course, Students
FROM Lecturers

ADO.NET:

(details omitted)
DataSet dataSet1 = new DataSet();
sqlSelectCommand1.CommandText =
"SELECT LectID, Name, Course, Students " +
"FROM Lecturers";
dataAdapter1.Fill(dataSet1, "Lect");
    
```



```

DataColumn objects hold information about the column,
not the actual data.

DataTable tbl = dataSet1.Tables[0];

foreach (DataColumn col in tbl.Columns)
    Console.WriteLine(col.ColumnName + col.DataType.ToString());

foreach(DataRow row in tbl.Rows)
    Console.WriteLine("The name is: " + row["Name"]);

foreach(DataRow row in tbl.Rows)
    Console.WriteLine("The name is: " + row[1]);

DataRow objects hold the data. Rows may be indexed
by column name or column position.
    
```

```

(details omitted)
DataSet dataSet1 = new DataSet();
sqlSelectCommand1.CommandText =
"SELECT LectID, Name, Course, Students " +
"FROM Lecturers";
dataAdapter1.Fill(dataSet1, "Lect");

DataTable tbl = dataSet1.Tables["Lect"];

foreach(DataRow row in tbl.Rows)
    Console.WriteLine("The name is: " + row[1]);
    
```

Untyped DataSets

- Collection of Tables
 - Tables are collections of Columns and Rows
 - Rows hold the data
 - Filling tables does not create relations between them

```
roomNum = dsLect.Tables[1].Rows[0][3];
roomNum = dsLect.Tables["Offices"].Rows[0]["Room"];
```

- To use relations between tables in memory, we must write code that builds the relations

Demo: DBRelations

Database Search

```

. . .
DataTable table = new DataTable();
sqlDataAdapter1.Fill(table);
DataRow[] rows;

rows = table.Select(" Country = 'USA' AND " +
                    " City <> 'Palo Alto'");

foreach (DataRow row in rows)
    Console.WriteLine(row["CompanyName"]);

```

Typed DataSets

- A class derived from DataSet
- Incorporates the schemas for the tables it contains
- Has properties and methods that allow access to tables and columns by name
- Extends DataSet, DataTable, DataRow to provide custom classes
- Table, column, and method names are known to IntelliSense, reducing coding time and errors

Typed DataSets

- A class derived from DataSet
- Incorporates the schemas for the tables it contains
- Has properties and methods that allow access to tables and columns by name
- Extends DataSet, DataTable, DataRow to provide custom classes
- Table, column, and method names are known to IntelliSense, reducing coding time and errors

```
roomNum = dsLect.Offices[0].Room;
```

↑ Property returning an OfficesDataTable
↑ Indexer operates on Rows collection, returning in OfficesRow
↑ Property returning value from Room column

Generating a Typed DataSet

- Open a Visual Studio project
- Select Data | Add new data source...
- Find the tables of interest in the Data Sources panel and choose the view and columns you want
- Drag the tables onto the design surface

Generating a Typed DataSet

- Open a Visual Studio project
- Select Data | Add new data source...
- Find the tables of interest in the Data Sources panel and choose the view and columns you want
- Drag the tables onto the design surface



- A file with extension `.xsd` that represents the tables and schemas
- A class derived from `DataSet` in a `.cs` file
- An instance of the class is added to the form
- A Binding Source and Table Adapter for each table are added to the form

```
public class LectDataSet : DataSet LectDataSet.Designer.cs
{
    public LecturersDataTable Lecturers;
    public OfficesDataTable Offices;
    ...
}

public class LecturersDataTable : DataTable, IEnumerable
{
    public LecturersRow this[int index] { get {...} }
    public LecturersRow FindByLectID(int lectID) { ... }
}

public class OfficesDataTable : DataTable, IEnumerable
{
    public OfficesRow this[int index] { get {...} }
    public OfficesRow FindByOfficeID(int officeID) { ... }
}
```

```
public class LecturersRow : DataRow LectDataSet.Designer.cs
{
    public int LectID { get {...} set {...} }
    public string Name { get {...} set {...} }
    ...
    public OfficesRow[] GetOfficesRows() {...}
}

public class OfficesRow : DataRow
{
    public string Building { get {...} set {...} }
    public string Room { get {...} set {...} }
    ...
    public LecturersRow LecturersRow
    { get {...} set {...} }
}
```

Database Search

```
...
DataTable table = new DataTable();
sqlDataAdapter1.Fill(table);
DataRow[] rows;

rows = table.Select(" Country = 'USA' AND " +
                    " City <> 'Palo Alto'");

foreach (DataRow row in rows)
    Console.WriteLine(row["CompanyName"]);
```

One problem is that you can't data bind to an array of rows.

DataView

- Represents a databindable, customized view of a `DataTable` for sorting, filtering, searching, editing, and navigation
- Associated with a single table
- Returns all columns
- Does not maintain its own copy of the data
- Can be configured in designer or programmatically
- Allows two controls to bind to the same table but show different versions of the data (e.g. one might show added rows and the other might show deleted rows)

DataView

```
DataRowView dv = new DataView(Lecturers);

dv.RowFilter = "Name = 'Bob'";

dv.RowStateFilter = DataRowViewRowState.ModifiedCurrent;


dv.Sort = "Course DESC";

foreach (DataRowView rowView in dv)
    Console.WriteLine(rowView["Course"]);
```

DataView

```
DataView dv = new DataView(Lecturers);
dv.RowFilter = "Name = 'Bob'";
dv.RowStateFilter = DataViewRowState.ModifiedCurrent;
dv.Sort = "Course DESC";
foreach (DataRowView rowView in dv)
    Console.WriteLine(rowView["Course"]);
```

Use current values of modified rows



Database Operations

- SELECT - retrieve rows from a table

Database Operations

- SELECT - retrieve rows from a table
- INSERT - add new rows to a table
- UPDATE - modify the data in existing rows
- DELETE - remove rows from a table

Database Operations

- SELECT - retrieve rows from a table
- INSERT - add new rows to a table
- UPDATE - modify the data in existing rows
- DELETE - remove rows from a table

How do we do these operations

-- to the dataset?

-- to the database?

Working with DataSets

- DataSets are in-memory copies of the data
- The **Fill** method uses a SELECT command to retrieve data
- We can modify data directly—easy with typed DataSets
- DataSets have **Add** and **Delete** methods

Working with DataSets

- Rows have a **RowState** property whose value comes from the **DataRowState** enumeration
 - Added
 - Deleted
 - Detached
 - Modified
 - Unchanged

Working with DataSets

- Rows have a **RowState** property whose value comes from the **DataRowState** enumeration
 - Added
 - Deleted
 - Detached
 - Modified
 - Unchanged

`row.RowState`

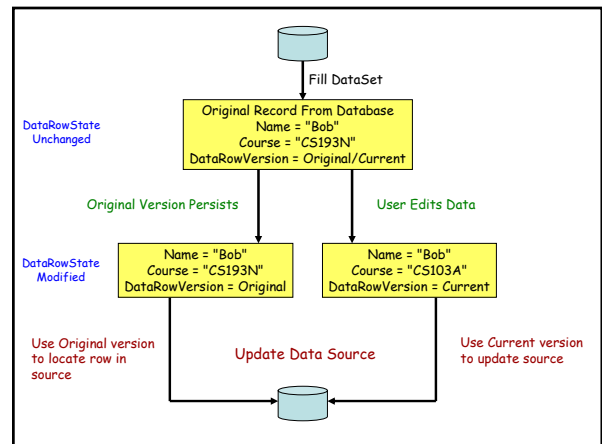
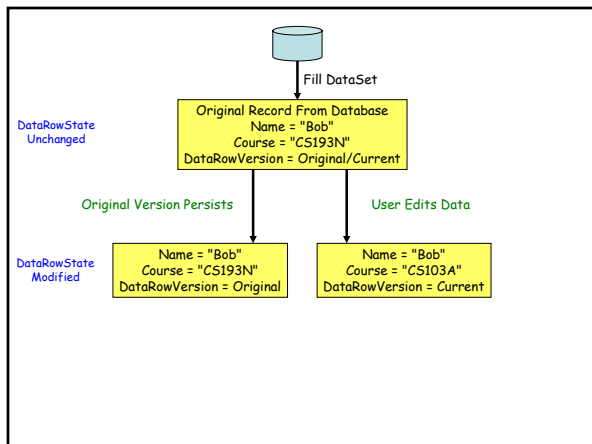
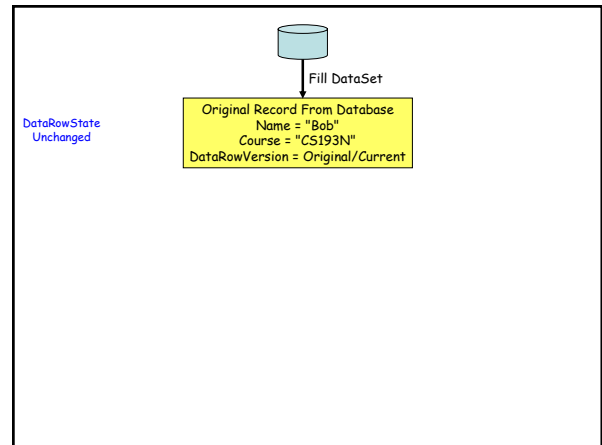
Working with DataSets

- Rows have a **RowState** property whose value comes from the **DataRowState** enumeration
 - Added
 - Deleted
 - Detached
 - Modified
 - Unchanged
- Rows maintain multiple versions, accessible using a member of the **DataRowVersion** enumeration
 - Current
 - Default
 - Original
 - Proposed

Working with DataSets

- Rows have a **RowState** property whose value comes from the **DataRowState** enumeration
 - Added
 - Deleted
 - Detached
 - Modified
 - Unchanged
- Rows maintain multiple versions, accessible using a member of the **DataRowVersion** enumeration
 - Current
 - Default
 - Original
 - Proposed

`row[3, DataRowVersion.Original]`



We can accomplish the update by using the DataAdapter's Update method:

```
sqlDataAdapter2.Update(LectDataSet1);
```

The DataAdapter examines the RowState property, and executes the required INSERT, UPDATE, or DELETE statements for each row. The state of all rows in the DataSet is then set to Unchanged.

Database Updating

In a multiuser environment, suppose:

- Two users have copies of the same data
- Both change it
- One user updates the database
- Then the other user updates the database

What happens?

Database Updating

There are three models for updating the data in the database:

- Pessimistic concurrency
- Optimistic concurrency
- "Last in wins"

Database Updating

Pessimistic concurrency—rows are locked at the data source until the lock owner releases it

Useful where

- There is high contention for the same records. The cost of placing locks may be less than the cost of dealing with conflicts.
- The nature of the user's task requires that data not be changed while the steps are carried out.

Difficult in a disconnected architecture

Not good where lock times are long and there are many users

Database Updating

Optimistic concurrency—rows are not locked when they are read. A concurrency error occurs if a user attempts to modify the database but finds that the data has been changed.

Useful where

- There is low contention for the same records.
- High performance is desired
- Code can be written to handle errors

A good fit for a disconnected architecture

Database Updating

Last in Wins—no check of the original data is made. The data is always written to the database.

User A reads a row.

User B reads the same row, modifies it, and writes it to the database.

User A modifies the row and writes it to the database.

B's changes are never seen by A and are overwritten.