ADO.NET

ADO.NET

- Behind every great application is a database manager
  - Amazon
  - eBay
- Programming is about managing application data
- UI code is just goo :)

ADO.NET

- Elegant, easy-to-use database API for managed applications
- Many database APIs
  - ODBC
  - DAO
  - RDO
  - OLE DB

Two Providers

- ADO.NET database access goes through two data providers
  - SQL Server .NET provider interfaces with Microsoft SQL Server
  - OLE DB .NET provider interfaces to databases through unmanaged providers

Providers

- Decide on a provider before you write any code
- Microsoft SQL Server 2000 is installed on Winter
- I have provided you all with accounts and databases
- You can connect using the SQL Server provider as described next

Namespace

- Need to use the SqlClient namespace
- using System.Data.SqlClient;

Connecting to a Database

1. Create a connection object encapsulating a connection string
2. Open the connection by calling Open on the connection object
3. Create a command object encapsulating both an SQL command and the connection the command will use
4. Call a method on the command object to execute the command
5. Close the connection by calling Close on the connection object

1. Create a connection

- Create a connection object encapsulating a connection string

```
SqlConnection conn = new
  SqlConnection

  ("server=winter.cs.pacificu.edu;
       database=cs360Test;
       uid=shereen;
       pwd=abc123");
```


1. Create a connection

- Other available parameters
  - Connect Timeout: specifies the maximum length of time in seconds that you're willing to wait when opening a connection (default = 15)
  - Integrated Security: enables and disables integrated security
    - Default is false, use is authenticated based on user name on password not Windows access

2. Open the connection

- Open the connection by calling Open on the connection object
- `conn.Open();`
- But what if a connection could not be established?
- Place within a try and catch
- Open throws a SqlException

3. Open the connection

```
try
{
  conn.Open();
}
catch (sqlException ex)
{
  // handle exception
}
finally
{
  conn.Close();
}
```


4. Create a command object

- Create a command object encapsulating both an SQL command and the connection the command will use
- We now have an open connection to the database
- We use command classes to execute commands on the database
  - Select
  - Insert

4. Create a command object

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "delete from
  titles where title_id =
  'BU1032'";
cmd.Connection = conn;
cmd.CommandTimeout = 10;
cmd.ExecuteNonQuery();
```


ExecuteNonQuery Method

- Used for executing
  - INSERT, UPDATE, DELETE
    - Return value is number of rows affected
  - CREATE DATABASE
  - CREATE TABLE
  - Other SQL commands that don't return values
    - Return value is -1

ExecuteScalar Method

- Executes an SQL command and returns the first row of the first column in the result set
- Executes functions that return single-row, single-column result sets
  - COUNT, AVG, MIN, MAX and SUM

Example of ExecuteScalar

```
SqlCommand cmd = new sqlCommand
     ("select max (advance) from titles",
       conn);
Decimal amount = (decimal)cmd.ExecuteScalar();
Console.WriteLine("ExecuteScalar returned
  {0:c}", amount);
```

- An incorrect cast throws an InvalidCastException

Example of ExecuteScalar

- BLOB (Binary Large Objects)
  - Such as images stored as bitmaps

```
MemoryStream stream = new MemoryStream();
byte[] blob = (byte[]) cmd.ExecuteScalar();
stream.Write(blob, 0, blob.Length);
Bitmap bitmap = new Bitmap(stream);
Bitmap.Dispose();
```


Writing BLOBs to a Database

```
FileStream stream = new FileStream
                ("Logo.jpg, FileMode.Open);
byte[] blob = new byte[stream.Length];
Stream.Read(blob, 0, (int) stream.Length);
Stream.Close();
SqlCommand cmd = new SqlCommand
    ("insert into pub_info (pub_id, logo)
      values ('9937', @logo)", conn);
cmd.Parameters.Add("@logo", blob);
cmd.ExecuteNonQuery();
```

ExecuteReader Method

- Performs database queries and obtains the results as quickly and efficiently as possible
- Returns an SqlDataReader object
- Inherits from DataReader
- Contains methods and properties to iterate over the result set
- Fast, forward-only, read-only mechanism

ExecuteReader Example

```
SqlCommand cmd = new SqlCommand
     ("select * from titles", conn);
SqlDataReader reader = cmd.ExecuteReader();
while(reader.Read())
  Console.WriteLine(reader["title"]);
```

- Each call to Read returns one result
- Fields can be referenced by name or index

ExecuteReader

- Do you need to know a database's schema in order to query it?
- No, example follows

GetName Method

```
SqlCommand cmd = new SqlCommand
              ("select * from titles", cmd);
SqlDataReader reader = cmd.ExecuteReader();
for(int i=0; i<reader.FieldCount; i++)
  Console.WriteLine(reader.GetName(i));
```

- GetName used to retrieve field names

GetValue Method

- GetValue is used to retrieve field values
- Returns a generic object
- Other Get methods
  - GetInt32 and GetDecimal

GetValue Example

```
SqlCommand cmd = new SqlCommand
   ("select * from titles where advance != 0,
   conn);
SqlDataReader reader = cmd.ExecuteReader();
int index = reader.GetOrdinal("advance");
while(reader.Read())
  Console.WriteLine("{0:c},
          reader.GetDecimal(index));
```

Closing Readers

- Readers need to be closed
- `reader.Close();`

Transacted Commands

- Suppose an application will transfer funds from one bank account to another
- The updates should be grouped together
- Why?

Transactions Example

```
SqlCommand cmd = new SqlCommand
   ("update accounts set balance =
    balance - 1000" +
    "where account_id = '1111'", conn);
cmd.ExecuteNonQuery();
cmd.CommandText =
    "update accounts set balance =
    balance + 1000" +
    "where account_id = '2222'";
cmd.ExecuteNonQuery();
```

- What's the problem?

Better Solution

```
SqlTransaction trans = null;
try {
 trans = conn.BeginTransaction
            (IsolationLevel.Serializable);
 SqlCommand cmd = new SqlCommand();
 cmd.Connection = conn;
 cmd.Transaction = trans;
 cmd.CommandText = "update accounts set
  balance = balance - 1000 where account_id =
  '1111'";
 cmd.ExecuteNonQuery();
```


Continued

```
 cmd.CommandText = "update accounts set balance =
   balance + 1000 where account_id = '2222'";
 cmd.ExecuteNonQuery();
 trans.Commit();
}
catch(SqlException) {
 if(trans!=null)
  trans.Rollback();
}
finally {
 conn.Close()
}
```


Parameterized Commands

- Used for executing the same command on a database with different values
- Especially useful for user input!

Example

```
SqlCommand cmd = new SqlCommand("update accounts set
  balance = balance + @amount where account_id = @id",
  conn);
cmd.Parameters.Add("@amount", SqlDbType.Money);
cmd.Parameters.Add("@id", SqlDbType.Char);
cmd.Parameters["@amount"].Value = -1000;
cmd.Parameters["@id"].Value = "1111";
cmd.ExecuteNonQuery();
cmd.Parameters["@amount"].Value = 1000;
cmd.Parameters["@id"].Value = "2222";
cmd.ExecuteNonQuery();
```