Relational Databases, SQL and ADO.NET in 75 minutes

Relational Databases
- Data is organized into tables with rows and columns
- A row is a single instance of a record
- Columns are the attributes of a record
- Tables can be linked in relationships

Keys / Indexes
- Keys are columns or groups of columns that are “Indexed” to make find / sorting them faster
- Index can be unique or allow duplicates
- One key (one or more columns) can be “primary,” must be unique

Organizing data (schema)
- How data (tables, rows, columns) are organized in a database is its “schema”
- Data is organized best when it is organized in a “normal form”
  - You will be given existing tables so understanding normal forms is not necessary
  - Please take CS 564 for more information

Relationships
- Only type of relationship discussed here is a “link” where rows / records in two tables share a common column / attribute
- Table 1: UID, Name
- Table 2: UID, Grade1
  - UID, Grade2 etc.
- Find Joe’s name and grades where the UID in both tables refers to Joe.

SQL
- Structured Query Language
- A few words that impact your life every day
- We will focus on 4 commands
  - Select
  - Insert
  - Update
  - Delete
Quotation
- Specifying data in SQL commands are very fragile with respect to use of quotation marks
- If specifying SQL commands from a program use “parameterized” arguments to avoid the problem
- Parameterized arguments are discussed later

select
- Select columns from tables where certain conditions are true plus some options
- Select all columns, all records:
  - `select * from t;`
- Select all columns, some records
  - `select * from t where age > 21;`
- Select all columns, some records, w/ options
  - `select * from t where age > 21 order by lastname;`

select
- Select some columns
  - `select firstname, lastname from t;`
- Select on more than one condition
  - `select * from t where age > 21 and age < 75;`
- Usual logical operators for conditions
- String columns can be pattern matched
  - `select firstname where firstname like '%th%';`

select
- Select (and summarize) by group
  - `select count(state),state from t group by state;`
- Select unique values
  - `select distinct state from t order by state;`
- See:

Selecting from more than one table - Join
- There are several types of joins. We only look at the “inner join” (simply use "," between table names)
- Cross product of two tables (hopefully) limited by some constraint
  - `select id, name, ordernumber from customers, orders where customers.id = orders.customerid order by id;`
- If there is a column with the same name in two tables, you must disambiguated explicitly

insert
- `insert into tbl set columnname=value;`
- Multiple columns can be set separated by ",".
- Value can be “default” if column has a default
- If there is a collision of a “unique” key, an error results
- Use “ignore” syntax if you don’t care
  - `insert ignore into t set id=29;`
- See
update
- update [ignore] tbl set id=9 where id=6
- Multiple columns may be set separated by "," 
- Compound "where" conditions may be used
- Note the optional "ignore" if you are changing
  a key value that is supposed to be unique
  and a collision occurs
- See

delete
- delete [ignore] from tbl where id=9;
- Don’t leave out the where condition unless
  you want to delete all records (not in this
  class)
- Note optional "ignore" to ignore errors
- Multiple where conditions may be specified
- See

ADO.NET
- Active Data Objects for .NET
- Object oriented wrapper to database
  methods and data structures
- We will use ODBC version of methods
  - Open Database Connectivity
  - Independent of database backend

Typical flow
- Define connection – the connection string
- Open the connection
- Issue commands, receive / transmit data
- Close the connection

Connection string
- MySQL version
  - DRIVER= {MySQL ODBC 3.51 Driver};
  - SERVER= oberon.cs.wisc.edu;
  - PORT= 3400;
  - DATABASE= databaseName;
  - USER= userName;
  - PASSWORD= myPassword;
  - OPTION= 3;
- One long string

{MySQL ODBC 3.51 Driver};
- Refers to the MySQL connector which must
  be installed on your system
- Will be preloaded on instructional machines
- Found here:
Connection object
- Instantiate an OdbcConnection
- Pass connection string to constructor
- Will use:
  - Methods
    - Open – open the connection
    - CreateCommand – create command objects
    - Close – close the connection
  - Attributes
    - State

Remember to close an open connection
- Nice use of “finally”
- Or webpage’s “Unload” function – discussed in future lecture
- Use open / close judiciously as operation is high overhead

Command object
- Instantiate OdbcCommand object either by constructor or connection object
  CreateCommand
- If you use the constructor, you need to specify the connection object to the Connection attribute
- Specify command in CommandText
- Use parameterized queries!
  - Command.Parameters.AddWithValue()

Parameterized queries
- If any part of a SQL command can come from user input, avoid SQL injection attacks by using parameterized queries
- Example:
  select c2 from t where c1 = ____;
  substitute 1; drop table t
- What happens?

Parameterized queries
- Cleaner looking code
- Eliminates the headache of proper quoting

command.CommandText = "select * from students where id = @id";
command.Parameters.AddWithValue("@id", id);

command.CommandText = "select * from students where id = @id"
command.Parameters.AddWithValue("@id", id);

ExecuteNonQuery()
- Used for executing SQL commands which do not return a row or rows of data such as:
  - insert
  - delete
  - update

command.CommandText = "update t set c2= @c2 where c1= @c1";
command.Parameters.AddWithValue("@c2", c2);
command.Parameters.AddWithValue("@c1", c1);
command.ExecuteNonQuery();
ExecuteScalar()

- Used to return exactly one column of one row
- Frequent example:
  - Getting a count of some set of rows
- Returns instance of Object
  - Return value must be converted to the appropriate type

```csharp
command.CommandText = "select count(*) from t where c1 = @v1";
command.Parameters.AddWithValue("@v1", v1);
int count = (int)command.ExecuteScalar();
```

ExecuteReader()

- Used to retrieve a row or rows one at a time
- Uses little memory
- Sequential forward-only access
- Ties up connection as long as it is open

```csharp
try
{
    reader = command.ExecuteReader();
    while (reader.Read())
    {
    }
}
finally
{
    if (reader != null && reader.IsClosed == false)
    {
        reader.Close();
    }
}
```

DataTables, DataSets

- You may use this paradigm if you wish
- See "hidden slides"
- Glossed over here

Which do you think is higher performance?

DataAdapter

- Bridge between the data source and a DataSet object (discussed next)
- Two methods most important:
  - Fill – connects to database, fills a DataSet from the database, then disconnects
  - Update – computes update needs, connects to a database, updates the database, then disconnects
**DataSet**

- Memory-resident object oriented representation of a database
- Being memory-resident:
  - is fast
  - can be randomly accessed
  - potentially large memory cost
- Perhaps overkill for many web applications?

**DataTable**

- Made up of DataRows (Rows) and DataColumns (Columns)
- Rows can be accessed by index
- Column data can be accessed by index or name
- Column data are of type Object – must be converted with System.Convert
- Has select capability (but different syntax)

**CommandBuilder**

- Generates single table SQL commands for use with DataAdapter.Update( )
  - Insert, Delete, Update
- Now for some examples

**Filling a DataTable**

```csharp
DataSet ds = new DataSet();
OracleConnection connection = new OracleConnection("connection string");
OracleDataAdapter adapter = new OracleDataAdapter("sql select statement", connection);
// Do SQL - give explanation here
adapter.Fill(ds, "MyTable");
DataTable dt = new DataTable("MyTable");
try
{
    connection.Open();
    adapter.Fill(dt);
    dt.TableName = ds.Tables[0].TableName;
    // Do something here
}
finally
{
    if (connection.State == ConnectionState.Open)
    {
        connection.Close();
    }
}
```

**Accessing Row Data**

```csharp
foreach (DataRow row in dt.Select("column = value"))
{
    flag = System.Convert.ToBoolean(row[1]);
    name = System.Convert.ToString(row["name"]);
}
```

**Updating, Deleting, Inserting**

```csharp
dt.Rows[0]["column name"] = 13;
dt.Rows[0].Delete();
```

// Makes row with proper schema
// for table - ie: establishes the
// columns.
DataRow r = dt.NewRow();
r["column name"] = "some value";
dt.Rows.Add(r);
Changes Not Committed Yet!

```csharp
OleDbDataAdapter adapter = new OleDbDataAdapter("select statement", connection);
OleDbCommandBuilder commandBuilder = new OleDbCommandBuilder(adapter);
adapter.UpdateSchemaAction = MissingSchemaAction.AddWithKey;
adapter.Update(dt);
dt.AcceptChanges();
```

Summary Reader Vs. Set

- **DataReader**
  - One way, sequential
  - Memory efficient
- **DataSet**
  - Memory resident view of database
  - Fast random access
  - Can be expensive
  - Permits definition of table relationships (not covered in this course)

Important subject not covered!

- I have not covered concurrency issues at all
- Should not be an issue because each of you get your own database