C# Programming in Depth

Prof. Dr. Bertrand Meyer

March 2007 – May 2007

Lecture 10: Database

Lisa (Ling) Liu
Database and Data Representation

- **Database Management System (DBMS):**
  - provides efficient, convenient, and safe multi-user storage of persistent data
  - provides access to massive amounts of persistent data
  - provides a programming interface that allows a user or program to
    - create new database and specify their structure
    - query and modify the data

- Dominant approach: relational database and SQL
### Employee

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>Title</th>
<th>ManagerID</th>
<th>VacationHours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Production Technician</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>Marketing Assistant</td>
<td>6</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Engineering Manager</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Senior Tool Designer</td>
<td>3</td>
<td>48</td>
</tr>
</tbody>
</table>

- **EmployeeID**: Int32
- **Title**: String
- **ManagerID**: Int32
- **VacationHours**: Int16
Database and Data Representation

- A “relation” is a table of data
- The columns are known as “attributes”
- The row are called “tuples”
- It is allowable for some values to be missing
- We can add, remove, or update tuples
- Each attribute has an underlying domain, or data type
SQL Database

- We will generally refer to the relations, attributes, and tuples as tables, columns, and rows.

- The structure of a table is referred to as its schema.

<table>
<thead>
<tr>
<th>Employee (HumanResources)</th>
<th>Column Name</th>
<th>Condensed Type</th>
<th>Nullable</th>
</tr>
</thead>
<tbody>
<tr>
<td>EmployeeID</td>
<td>int</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>NationalID</td>
<td>nvarchar(15)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ContactID</td>
<td>int</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>LoginID</td>
<td>nvarchar(256)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ManagerID</td>
<td>int</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>nvarchar(50)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>BirthDate</td>
<td>datetime</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>MaritalStatus</td>
<td>nchar(1)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>nchar(1)</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>HireDate</td>
<td>datetime</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>SalariedFlag</td>
<td>Flag:bit</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>VacationHours</td>
<td>smallint</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>SickLeaveHours</td>
<td>smallint</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>CurrentFlag</td>
<td>Flag:bit</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>rowguid</td>
<td>uniqueidentifier</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>ModifiedDate</td>
<td>datetime</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
### Employee

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>Title</th>
<th>ManagerID</th>
<th>VacationHours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Production Technician</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>Marketing Assistant</td>
<td>6</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>Engineering Manager</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Senior Tool Designer</td>
<td>3</td>
<td>48</td>
</tr>
</tbody>
</table>

**Primary key**

**Primary key**: no two rows can have the same EmployeeID. EmployeeID cannot be null.
- Assume that we want to add data about employees' salary
  - Assume an employee’s salary is changeable, we need to add following two columns.
    - RateChangeDate
    - Rate
  - We can’t add additional columns for the same employee without violating the primary key constraint. So we use another table.
### Employee

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>Title</th>
<th>ManagerID</th>
<th>VacationHours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PT</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>MA</td>
<td>6</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>EM</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>STD</td>
<td>3</td>
<td>48</td>
</tr>
</tbody>
</table>

### EmployeePayHistory

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>RateChangeDate</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31.07.1996</td>
<td>12.4500</td>
</tr>
<tr>
<td>2</td>
<td>26.02.1997</td>
<td>13.4615</td>
</tr>
<tr>
<td>3</td>
<td>12.12.1997</td>
<td>43.2692</td>
</tr>
<tr>
<td>4</td>
<td>05.01.1998</td>
<td>8.6200</td>
</tr>
<tr>
<td>4</td>
<td>01.07.2000</td>
<td>23.7200</td>
</tr>
<tr>
<td>4</td>
<td>15.01.2002</td>
<td>29.8462</td>
</tr>
</tbody>
</table>

EmployeeID establishes a relationship between the tables.
We say that there is a “foreign key constraints” between the tables.

The column referenced in the parent table must be a primary key.

Every value in the foreign column must actually appear in the parent table.
Simple SQL Queries

- SELECT

  SELECT *  — what columns to output
  FROM Employee  — what tables are involved
  WHERE VacationHours > 20  — what rows are of interest
### Employee

<table>
<thead>
<tr>
<th>EmployeeID</th>
<th>Title</th>
<th>ManagerID</th>
<th>VacationHours</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PT</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>MA</td>
<td>6</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>EM</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>STD</td>
<td>3</td>
<td>48</td>
</tr>
</tbody>
</table>

```sql
SELECT EmployeeID, ManagerID
FROM Employee
WHERE VacationHours > 20
```
How to interact with data stores?

- **ADO.NET**
  a set of namespaces defined on .NET platform that understand how to interact with data stores.
  - native support for SQL Server and Oracle
  - support for other databases via older *OleDb* technology
  - requires a knowledge of SQL
ADO.NET-centric Namespaces

- **Core namespaces:**
  - general:
    - System.Data
  - SQL Server:
    - System.Data.SqlClient
  - Oracle:
    - System.Data.OracleClient
  - OleDb:
    - System.Data.OleDb
Two manners of accessing database

- **Connected manner**
  - explicitly connected to and disconnected from the underlying data store
- **Disconnected manner**
  - using **DataSet** - a local copy of external data to interact with data stores
Data Providers

ADO.NET providers provide access to a given DBMS.
Overview of database access

- General steps:
  - open connection to database
  - execute SQL to retrieve records / update DB
  - close connection
Five steps:

1. Allocate, configure, and open your connection object
2. Allocate and configure a command object
3. Acquire DataReader object
4. Process each record using DataReader object
5. Close connection
Step 1: open connection

- Connections are opened based on connection string info
  - here we open a connection to a SQL Server database
  - “AdventureWorks” database must be installed on the local machine.

```csharp
using System.Data;
using System.Data.SqlClient;
...

SqlConnection cn = new SqlConnection();
cn.ConnectionString = "server=(local);database=AdventureWorks;integrated security=true";
cn.Open();

MessageBox.Show( cn.State.ToString() );
```
Building connection strings

- *Connection strings are vendor-specific, not well-documented*

- **Where to turn for help?**
  - [www.connectionstrings.com](http://www.connectionstrings.com)
Step 2-4: retrieve records

- Retrieve records via SQL Select query
  - read-only access by database field names

```csharp
string strSQL = "SELECT * FROM HumanResources.Employee";
SqlCommand myCommand = new SqlCommand(strSQL, cn);

SqlDataReader myDataReader;
myDataReader = myCommand.ExecuteReader(CommandBehavior.CloseConnection);

while (myDataReader.Read())
{
    Console.WriteLine("EmployeeID: {0}, Title: {1}, ManagerID: {2}, VacationHours: {3}",
                       myDataReader["EmployeeID"].ToString().Trim(),
                       myDataReader["Title"].ToString().Trim(),
                       myDataReader["ManagerID"].ToString().Trim(),
                       myDataReader["VacationHours"].ToString().Trim());
}
```
Step 5: close connection

- Be sure to close connection...
  - to flush pending updates
  - so others can access DB (connections are limited resources)

```csharp
    cn.Close();
```
Guaranteed close?

```csharp
IDbConnection dbConn = null;

try {
    cn.Open();
    ...
}
catch(Exception ex) {
    System.Diagnostics.EventLog.WriteEntry("MyApp", ex.StackTrace);
    throw ex;
}
finally {
    if ((cn != null) && (cn.State != ConnectionState.Closed))
        cn.Close();
}
```
Updating a database

To update database, execute an SQL Action query

Example:

➢ delete employee by their id number

```
string sql = string.Format("DELETE FROM Employee WHERE EmployeeID = '{0}'", employeeID);
SqlCommand cmd = new SqlCommand(sql, cn);

try
{
    cmd.ExecuteNonQuery();
}
catch
{
    Console.WriteLine("Sorry! That employ cannot be deleted.");
}
```
Example of action queries

Insert, update and delete:

Insert Into Customers(CID, FirstName, LastName, CreditLimit, Balance) Values(118, 'Jia', 'Zhang', 10000.0, 0.0);

Update Customers Set CreditLimit = 40000000000.0, Balance = 0.0 Where LastName = 'Gates' and FirstName = 'Bill';

Delete From Customers Where CID = 666;
**Database Access (Disconnect Manner)**

- **SqlCommand**
  - "SELECT * FROM Table1"
  - Sql Connection
- **SqlDataAdapter**
  - Fill (myDataSet, "myTable")

---

**Diagram:**

- **SQL Server**
- **DataSet**
- **Forms**
- **Client Application**
DataSets are an in-memory, read-write data structure
- easily filled with data from a database
- easily displayed in a GUI app
DataAdapter

- make use of **DataSet** objects to move data between client and data store.
- is used to fill **DataSet** with **DataTable** objects
- send modified **DataTables** back to the database for processing
- take care of connection, hence client don’t need to explicitly open and close the connection with DBMS
Steps

1. construct data adapter with a valid connection or connection string and a command object
2. fill DataSet using the internal command within the data adapter
3. operate on the DataSet
4. using data adapter to update data store with the DataSet
Example

Retrieve product info and display in a DataGrid:

```csharp
string sql = "SELECT * FROM Employee";
SqlCommand myCmd = new SqlCommand(sql, cn);
SqlDataAdapter myAdapter = new SqlDataAdapter(myCmd);

DataSet myDS = new DataSet("HumanResources");
myAdapter.Fill(myDS, "Employee");

PrintDataSet(myDS);
```
sql = string.Format("INSERT INTO Employee" + 
    "(Title, ManagerID, VacationHours) VALUES" + 
    "('{0}', '{1}', '{2}')", title, managerID, vacationHours);

SqlCommand insertCmd = new SqlCommand(sql, cn);

myAdapter.InsertCommand = insertCmd;

//Update Employee table with new row
DataRow newEmployee = myDS.Tables["Employee"].NewRow();
newEmployee["Title"] = title;
newEmployee["ManagerID"] = managerID;
newEmployee["VacationHours"] = vacationHours;
myDS.Tables["Employee"].Rows.Add(newEmployee);
myAdapter.Update(myDS.Tables["Employee"]);
Untyped DataSets

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

```
carName = myDS.Tables["Inventory"].Rows[0]["PetName"]
```

- To use relations between tables in memory, we must write code that builds the relations
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 are known by names, reducing coding time and errors

```csharp
    carName = myDs.Inventory[0].PetName;
```
Generating a typed DataSet

1. Right-click project
2. Add | Add new item.. | Data Set
3. Select a name
4. Find the tables of interest in the Server Explorer and drag them onto the design surface
5. Drag relations onto the child tables and verify the settings
6. Save
   - A file with extension .xsd that represents the tables and schemas
   - A class derived from DataSet in a .cs file
Questions