

Lecture 9

ADO.NET - I

Objectives

In this lecture you will learn the following

- + Knowing about DBMS
- + Knowing about SQL
- + Manipulating of database in ASP.NET
- + Advantages of ADO.NET over ADO

Coverage Plan

Lecture 9
9.1 Snap Shot
9.2 Database
9.3 DBMS
9.4 DBMS Standardization
9.5 Structured Query Language
9.6 Using SQL as a DataQuery Language
9.7 Manipulation of database in ASP.NET
9.8 Introduction to ADO.NET
9.9 Short Summary
9.10 Brain Storm

9.1 Snap Shot

This session is designed to introduce the user to the concepts of database and DBMS. It acquaints the user with various commands in SQL. Further, the user is introduced to ADO.NET and its advantages over ADO.

Every organization has a pool of resources that it must manage effectively to achieve its objectives. Although their rules differ, all resources human, financial and material share a common characteristic.

The organization that fails to treat data or information as resource and to manage it effectively will be handicapped in how it manages its manpower, material and financial resources. In order to satisfy the information requirements of management, the data should be stored in an organized form.

Data verses Information

Data are facts concerning people, places, events or other objects or concepts. Data are often relatively useless to decision-makers until they have been processed or refined in some manner. Information is data that have been processed and refined and then given in the format that is convenient for decision making or other organizational activities. For example, report about student fee paid details is useful information for finance section.

Actually data are the facts stored in the record of a database. But the processed facts are presented in a form for usage is information.

9.2 Database

Radiant™
RAY OF HOPE

ASP.NET

Database

- ⌘ Shared collection of inter-related data designed to meet varied information needs
- ⌘ Acts as a media to store data in an organized way
- ⌘ Two properties
 - Integration
 - Sharing

A database is a shared collection of interrelated data designed to meet the varied information needs of an organization. For example, in a payroll application each person's record has Name, Age, Designation, Basic Pay etc as columns. So payroll database has a collection of all employee records, that are interrelated. From the database, various reports like payslip, persons with particular designation, service report etc can be obtained. The database acts as a media to store the data in an organized way so that it can be managed effectively. A database has two important properties integration and sharing.

Typical examples of **information stored** for some practical purpose are:

Information collected for the sake of making a statistical analysis, e.g. the national census. Operational and administrative information required for running an organization or a commercial concern will take the form of stock records, personnel records, customer records etc.

Because of the investment involved in setting up a database, the expectation must be that it will continue to be useful, over years rather than months. But the relationship with time varies from one type of

information to another. An organizational database may not change very drastically in size, but it will be subject to frequent updating (deletions, amendments, insertions) following relevant actions within the organization itself. Ensuring the accuracy, efficiency and security of this process is the main concern of many database designers and administrators.

Benefits of the Database approach

Radiant™
RAY OF HOPE

ASP.NET

Benefits of Database approach

- ⌘ Minimal data redundancy
- ⌘ Consistency of data Integration of data
- ⌘ Integration of data
- ⌘ Sharing of data
- ⌘ Ease of application development
- ⌘ Data independence
- ⌘ Reduced program maintenance

The database approach offers a number of important advantages compared to the file system. These benefits include minimal data redundancy, consistency of data, integration of data, sharing of data, enforcement of standards, ease of application development, uniform security, privacy and integrity controls, data accessibility and responsiveness, data independence, and reduced program maintenance.

Minimum data redundancy

With the database approach, data files that were separate are integrated into a single, logical structure. So each item is ideally recorded in only one place in the database. Hence, in a database system, the data redundancy is controlled.

Consistency of data

By eliminating data redundancy, the consistency of data is greatly improved. If there is any change in the data, it can be incorporated in one place unlike the traditional file system where the change has to be incorporated manually in each and every place.

Integration of data

In a database, data are organized into a single, logical structure, with logical relationships defined between associated data entities.

Sharing of data

The database is intended to be shared by all authorized users in the organization.

Database Concepts

The database stores information in tables that comprises of columns and rows. The columns separate data into fields. Each column contains a different kind of information. Each row constitutes a record. Let us consider an organization that needs to store the information about its employees. The information might include employee name, age, department, designation, salary, etc. Each of these is a column (or field). The details of each employee form a row (or record).

9.3 DBMS

Radiant™
RAY OF HOPE

ASP.NET

DBMS

- ⌘ Computerized record-keeping system that stores, maintains and provides access to information
- ⌘ Functions include Data entry and validation

A DBMS (Database Management System) is a computerized record-keeping system that stores, maintains and provides access to information. The function of the DBMS is to store, maintain and retrieve information as required by applications, programs or users. The functions of the DBMS are listed below:

Data entry and validation

Validation may include:

- Type Checking
- Range Checking
- Consistency Checking

In an interactive data entry system, errors should be detected immediately and recovery and re-entry should be permitted. If the database is error bound then it will be the main cause to make the program error prone.

Updating

Updating of data is very important otherwise it will be fruitless in future. Updating involves:

- Record Insertion
- Record Modification
- Record Deletion

Updating may take place interactively, or by submission of a file of transaction records; handling these may require a program of some kind to be written, either in a conventional programming language or in a language supplied by the DBMS for constructing command files.

Data retrieval on the basis of selection criteria

For this purpose most systems provide a **Query Language** through which the characteristics of the required records may be specified. Query languages differ enormously in power and sophistication but a standard, which is becoming increasingly common, is based on the so-called RELATIONAL operations.

These allow:

- Selection of records on the basis of particular field values
- Selection of particular fields from records to be displayed
- Linking together records from two different files on the basis of matching field values

Arbitrary combinations of these operations on the files making up a database can answer a very large number of queries without requiring users to go into one record at a time processing.

Report definition

Most systems provide facilities for describing how summary reports from the database are to be created and laid out on paper. These may include obtaining:

- COUNTS
- TOTALS
- AVERAGES
- MAXIMUM and MINIMUM values

On a table over a particular **Control Field** the above layouts have to be found out. Also specification of **Page and Line Layout, Headings, Page-Numbering**, and other narrative are required to make the report comprehensible.

Security

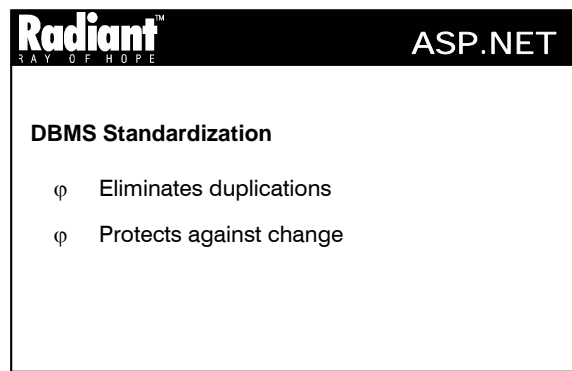
The security consists of several aspects:

- Ensuring that only those authorized can see and modify the data, generally by some extension of the password principle
- Ensuring the consistency of the database where many users are accessing and up-dating it simultaneously
- Ensuring the existence and INTEGRITY of the database after hardware or software failure. At the very least this involves making provision for back-up and re-loading

Importance of Databases and DBMS

An organization uses a computer to store and process information because it expects speed, accuracy, efficiency, economy etc. beyond what could be achieved by using clerical methods. The objectives of using a DBMS must in essence be the same although the justifications may be more indirect.

9.4 DBMS Standardization



Early computer applications were based on existing clerical methods and stored information was partitioned in much the same way as manual files. But the computer's processing speed gave a potential for relating data from different sources to produce valuable management information, provided some standardization could be imposed over departmental boundaries. The idea emerged of the integrated database as a central resource emerged. Here, data is captured as close as possible to its point of origin and transmitted to the database, then extracted by anyone within the organization who requires it. However, many provisos have become attached to this idea in practice, it still provides possibly the strongest motivation for the introduction of a DBMS in large organizations. The idea is that any piece of information is entered and stored just once, eliminating duplications of effort and the possibility of inconsistency between different departmental records. Data redundancy has to be removed as much as possible.

Advantages

Organizational requirements change over time, and applications laboriously developed need to be periodically adjusted. A DBMS gives some protection against change by taking care of basic storage and retrieval functions in a standard way, leaving the application developer to concentrate on specific

organizational requirements. Changes in one of these areas need not affect elsewhere. In general, a DBMS is a substantial piece of software, the result of many man-years of effort.

The points discussed above are probably most relevant to larger organizations using a DBMS for its administrative functions - the environment in which the idea of databases first originated. In other words the convenience of a DBMS may be the primary consideration. The purchaser of a small business computer needs all the software to run it in a package form, written so that the minimum of expertise is required to use it. The same applies to departments (e.g. Research & Development) with special needs that cannot be satisfied by a large centralized system. When comparing database management systems it is obvious that some are designed in the expectation that professional staff will be available to run them, while others are aimed at the total novice. Actual monetary costs vary widely from, for instance, a large multi-user Oracle system to a small PC-based filing system.

9.5 Structured Query Language

Radiant™
RAY OF HOPE

ASP.NET

SQL

- ⌘ Basic language that allows users to access data in database management systems
- ⌘ Mainly classified into DDL and DML

DDL

- ⌘ Create Database Statement
- ⌘ Create Table Statement
- ⌘ Alter Table Statement
- ⌘ Drop Table Statement

Structured Query Language (SQL) is a basic language that allows users to access data in database management systems, such as Oracle, Sybase, Informix, Microsoft SQL Server, Access by allowing users to describe the data the user wishes to see. SQL also allows users to define the data in a database, and manipulate that data.

SQL is mainly classified into Data Definition Language (DDL) and Data Manipulation Language (DML).

Using SQL as a Data Definition Language

Using SQL to define a database means creating a database, creating tables and adding them to a database, updating the design of existing tables, and removing tables from a database.

Create Database Statement

The **Create Database** Statement can be used to create a database:

```
CREATE DATABASE databaseName
```

where *databaseName* is the name of the database to be created. For example, the following statement creates a database named **Salesreps**

```
Create Database Salesreps
```

Note: The **Create Database** statement is not supported by all SQL implementations

Create Table Statement

The **Create Table** statement creates a table and adds it to the database:

```
CREATE TABLE tableName (columnDefinition,... , columnDefinition)
```

Each columnDefinition is of the form: ColumnName columnType

The **columnName** is unique to a table. The **columnType** identifies the type of data that will be stored in the table. Common data types are

- Char(n) - An n character text string
- Int - An integer value
- Float - A floating point value
- Bit - A boolean (1 or 0) value
- Datetime - A date value
- Money - A money value



Practice 9.1

The following is an example of a **Create Table** statement which creates a table named **Customers** containing columns CustName, Company, Cust_rep, Credit_limit:

```
CREATE TABLE Customers (CustName char(30), Company char(50), Cust_rep  
integer, Credit_limit money)
```



The command(s) completed successfully.

The example creates a **Customers** table with the following columns:

- CustName - A 30-character-wide text field
- Company - A 20-character-wide text field
- Cust_rep - A integer type
- Credit_limit - A money data type to represent currency values

ALTER TABLE Statement

The **Alter Table** statement is used to add a row to an existing table or to change the table definitions.

```
ALTER TABLE tableName ADD (columnDefinition... columnDefinition)
```

The values of the newly added columns are set to NULL. Columns are defined as described in the previous section.



Practice 9.2

The following is an example of the ALTER TABLE statement

```
ALTER TABLE CUSTOMERS ADD CONTACT_NAME VARCHAR(20)
```



The command(s) completed successfully.

The preceding SQL statement of the ALTER TABLE adds a column named Contact_Name to the Customers table

DROP TABLE Statement

The DROP TABLE statement deletes a table from the database:

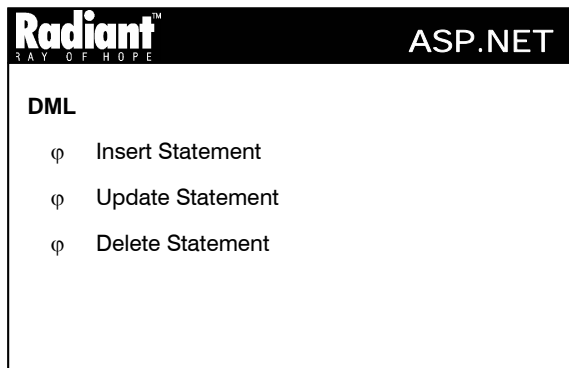
```
DROP TABLE tableName
```

The dropped table is permanently removed from the database. The following is an example of the DROP TABLE statement:

```
DROP TABLE CUSTOMERS
```

The preceding statement removes the CUSTOMERS table from the database. Tables once dropped cannot be retrieved.

Using SQL as a Data Manipulation Language



One of the primary uses of SQL is to update the data contained in a database. There are SQL statements for inserting new rows into a database, deleting rows from a database, and updating existing rows.

INSERT Statement

The INSERT statement inserts a row into a table:

```
INSERT INTO tableName VALUES (value 1, ..., value n )
```

In the preceding form of the INSERT statement, **value 1** through **value n** identify all the column values of a row. Character values should be enclosed within single quotes.



Practice 9.3

The following is an example of the preceding form of the INSERT statement:

```
INSERT INTO Customers VALUES ('Joy Anand', 'Radiant', 29, 500000)
```



(1 row(s) affected)

The preceding statement adds a row to the Customer table. All columns of this row are filled in.

An alternative form of the INSERT statement may be used to insert a partial row into a table. The following is an example of this alternative form of the INSERT statement:

```
INSERT INTO tableName (columnName 1,...,columnName m) VALUE          (value  
1,...,value m)
```

The values of **columnName 1** through **columnName m** are set to **value 1** through **value m**. The value of the other columns of the row are set to NULL.



Practice 9.4

The following is an example of the preceding form of the INSERT statement:

```
INSERT INTO Customers (CustName, Company)  
VALUES ('Smith','Radiant')
```



(1 row(s) affected)

The preceding statement adds a row to the Customer table with CustName - Smith and Company - Radiant. The other columns of the table are set to null.

DELETE Statement

The DELETE statement deletes a row or a set of rows from a table:

```
DELETE FROM tableName [WHERE condition]
```

All rows of the table that meet the condition of the WHERE clause are deleted from the table. The WHERE clause is covered in a subsequent section of this session.



If **WHERE** clause is omitted, all the rows of the table are deleted.



Practice 9.5

The following is an example of the DELETE statement:

```
DELETE FROM Customers WHERE CustName = 'Smith'
```



(1 row(s) affected)

The preceding statement deletes the row which has CustName as Smith from the Customers table.

UPDATE Statement

The UPDATE Statement is used to update an existing row of a table:

```
UPDATE tableName SET columnName1 = 'value1', ..., columnName = 'value'
[WHERE condition]
```

All the rows of the table that satisfy the condition of the WHERE clause are updated by setting the columns with the specified values. If the WHERE clause is omitted, all rows of the table are updated.



Practice 9.6

The following statement is an example of the UPDATE statement:

```
UPDATE Customers SET Credit_Limit = 1200000 WHERE Company = 'Radiant'
```



(1 row(s) affected)

The preceding statement changes the Credit Limit of the company Radiant with the new Credit Limit of Rs12,00,000.

9.6 Using SQL as a DataQuery Language

Radiant™
RAY OF HOPE

ASP.NET

Select Statement

- ☐ Specifies a database query
- ☐ Used to retrieve data from a database Where Clause
 - Boolean expression consisting of column names, column values, relational operators and logical operators

The most important use of SQL for many users is for retrieving data contained in a database. The SELECT statement specifies a database query:

```
SELECT columnList1 FROM table1 ,..., tablem [WHERE condition]
```

Note: An asterisk (*) may replace columnList1 to indicate that all columns of the table(s) are to be returned.



Practice 9.7

The following is an example for select statement.

```
SELECT * FROM CUSTOMERS
```



CustName	Company	Cust_rep	Credit_limit
----------	---------	----------	--------------

```
Joy anand    radiant    29          100000.0000
```

The preceding example selects all the rows and columns from the Customers table.

Where Clause

The WHERE clause is a Boolean expression consisting of column names, column values, relational operators, and logical operators. For example, suppose there are columns such as Department, Salary, and Bonus. The following WHERE clause could be used to find all employees in the Engineering department whose salaries are over 100,000 and bonus are less than 5,000.

```
WHERE Department = 'Engineering ' AND Salary >'100000' AND Bonus <'5000'
```

Radiant™
RAY OF HOPE

ASP.NET

WHERE clause can be based on some relational and logical comparisons

- ☐ Comparison Test
- ☐ Range Test
- ☐ Set Membership Test
- ☐ Pattern Matching Test
- ☐ Null Value Test
- ☐ Compound Search Conditions

Comparison Test

Comparison test makes use of the relational operators.

They are listed below:

- = equal to
- <> not equal to
- < less than
- <= less than or equal to
- > greater than
- >= greater than or equal to

Range Test

This test is used to test whether the value lies within the range or not. The keyword are:

- BETWEEN ... AND
- NOT BETWEEN ... AND



Practice 9.8

The following statement selects rows from the **Customers** table subject to the condition that the credit limit lies between 10000 and 50000.

```
SELECT * FROM CUSTOMERS
```

WHERE CREDIT_LIMIT BETWEEN 10000 AND 50000

CustName	Company	Cust_rep	Credit_limit
Smith	radiant	29	40000.0000

The preceeding example selects rows from **Customers** table that satisfy the condition that the Credit_limit is between 10000 and 50000.

Set Membership Test

This is to test whether the value is present in the list of values. The keyword is IN.

Example

```
SELECT * FROM CUSTOMERS WHERE CUST_REP IN (10, 15, 20, 25)
```

The above query selects the rows for which the Cust_Rep takes one of the values 10, 15, 20 or 25.

Pattern Matching Test

The Pattern matching test is employed to select rows from a table based on a field whose values match a given pattern. The keywords LIKE, %, _ are the wildcard characters used in the pattern matching test. % is equivalent to * in DOS. _(Underscore) is equivalent to ? in DOS. The \$ is used to remove the above special meaning of % and _.

Example

```
SELECT * FROM CUSTOMERS WHERE CUSTNAME LIKE 'S%H'
```

The above query searches for the customers whose first character is S and the last character is H.

```
SELECT * FROM CUSTOMERS WHERE CUSTNAME LIKE 'S$_C'
```

This will search for customers whose names start with S, end with C and contain the character _.

Null Value Test

This is used to test whether the column contains null value. The keyword is IS NULL.

Compound Search Conditions

This is used to test for more than one condition. The keyword AND/OR/NOT is used.

Radiant™
3 A Y O F H O P E

ASP.NET

- φ **Order by clause**
 - Used to order result set by specified column(s) of a table
- φ **Union**
 - Used to combine two or more tables
- φ **Joins**
 - Used when results of two or more queries have to be combined

Order By

The ORDER BY clause is used to order the result set by the specified column(s) of a table. Each of the column names in the column list may be followed by the **ASC** or **DESC** keywords. If **DESC** is specified, the result set is ordered in descending order. Otherwise, the result set is ordered in ascending order. The following query retrieves the rows of the **Customers** table ordered by the **CustName** field.

```
SELECT * FROM CUSTOMERS ORDER BY CUSTNAME
```

Union

It is used to combine two or more tables. The necessary criteria are:

- The tables must contain the same number of columns
- Data type of each of the columns of one table should match that of the other

When the keyword **UNION ALL** is used, all the rows of both the tables are displayed. When the keyword **UNION** is used, only distinct rows from both the tables are displayed.

Example:

```
SELECT * FROM A
UNION (SELECT * FROM B
UNION SELECT * FROM C)
ORDER BY NAME
```

The preceding statement produces a combined output from tables A,B and C. The output is arranged in the ascending order of Names.

Joins

Joins are used when the results of two or more queries have to be combined. This session discusses two of the joins namely, Equi joins and Non-Equi joins.

Equi-join

An equi-join is a join with a join condition containing an equality operator. An equi-join combines rows that have equivalent values for the specified columns.



Practice 9.9

The following example performs an equi-join with **Order** and **Customer** tables where both the tables are described below:

Order table contains the following columns

```
Ordernum
order_date
cust_rep
qty
amount
```

Customer Table contains the following columns

```
cust_num
company
cust_rep
credit limit
```

The query is as follows:

```
select ordernum, amount, credit_limit
from order, customer
where order.cust_rep = customer.cust_rep
```



Ordernum	amount	credit_limit
1	30000.0000	100000.0000
3	20000.0000	50000.0000

This **equi-join** returns the order number, amount and the credit limit of the Customer table where **cust_rep** of Order table is equal to **cust_rep** of Customer table.

Non-Equi-join

A non-equi-join is a join with a join condition containing any operator other than the equality operator.

Example

The following query selects the order number, amount and credit limit of the Customer table where **cust_rep** of Order table is not equal to **cust_rep** of Customer table.

```
select ordernum, amount, credit_limit
from order, customer
where order.cust_rep <> customer.cust_rep
```

Radiant™
3 A Y O F H O P E

ASP.NET

- φ **Functions**
 - Used to compute against a column or columns of data returned from a query
- φ **Group by**
 - Gather all rows that contain data in specified column(s)
- φ **Having**
 - Allows to specify conditions on rows for each group
- φ **Distinct**
 - Returns information from table columns without repetition

Functions

Functions are used to compute against a column or columns of data returned from a query. Some of the important functions are

SUM()	totals the column
AVG()	returns the average of the column
MIN()	returns the minimum value of the column
MAX()	returns the maximum value of the column
COUNT()	counts the number of rows for the specified column
COUNT(*)	counts the number rows for all the columns

Example

```
SELECT CUST_REP, SUM(AMOUNT) FROM ORDERS ORDER BY CUST_REP
```

The above query displays the number of each salesperson and the total sum of orders for each of them.

Group by

The GROUP BY clause will gather all of the rows together that contain data in the specified column(s) and will allow aggregate functions to be performed on the one or more columns.

Example:

```
SELECT company, SUM(credit_limit)
FROM customer
GROUP BY company
```

This query select the sum of credit_limit from customer table for each company.

Having

The HAVING clause allows to specify conditions on the rows for each group - in other words, which rows should be selected will be based on the conditions specified in the Having clause. The HAVING clause should follow the GROUP BY clause if used.

Example:

```
SELECT company, SUM(credit_limit)
FROM customer
GROUP BY company having sum(credit_limit) > 5000
```

This query select the sum of credit_limit from customer table for each company where sum of credit_limit is greater than 5000.

Distinct

The SQL SELECT statement with Distinct keyword returns information from table columns without repetition.

9.7 Manipulation of database in ASP.Net

Client server computing

The term client/server computing means connecting a client machine to a server machine and sharing the load of processing between the two. The client requests services and the server provides the requested services. The client does not request data at a file level, but sends a request to the server to execute a query and return specific records.

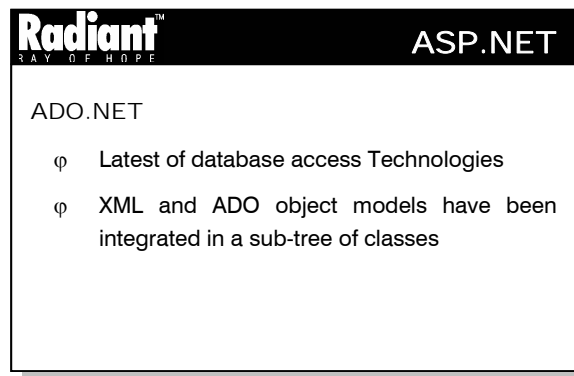
A special type of client/server architecture consisting of three well-defined and separate processes, each running on a different platform is as follows:

- The user interface, which runs on the user's computer (the *client*)
- The functional modules that actually process data. This *middle tier* runs on a server and is often called the *application server*
- A Database Management System (DBMS) that stores the data required by the middle tier. This tier runs on a second server called the *database server*

To accomplish the task of connecting to data sources that are open to many applications, the application and the database must agree on a common method of accessing the database. This agreement is implemented using a complete set of API calls and a complete SQL syntax set.

The database side of this open connectivity is provided by drivers. These drivers will transform the API functions into function calls supported by the particular data source being used. Similarly, the drivers transform the SQL syntax into syntax accepted by the data source. These drivers were originally produced by the manufacturers of the databases, but now there are many third-party vendors as well. The main benefit of this API is that many database formats can now be accessed by applications as their own native databases.

9.8 Introduction to ADO.Net



ADO.NET is the latest of the database access technologies that began with the Open Database Connectivity (ODBC) application programming interface (API). Microsoft introduced open data base connectivity with the promise of creating a singular common access methodology for databases. ODBC has come a long way since those early days. Almost every major database in use today supports ODBC drivers, and third party developers provide optimized driver versions. The primary focus of the ODBC is to provide a consistent interface to database data sources.

As time passed by COM landed at the database territory and started a colonization process that culminated with OLE DB. OLE DB is a set of COM-based interfaces that exposes data from a variety of sources. OLE DB interfaces provide applications with uniform access to data stored in diverse information sources, or data stores. These interfaces support the amount of DBMS functionality appropriate to the data store, enabling the data store to share its data.

With .NET, Microsoft is offering a general-purpose framework—the Framework Class Library—that will cover all the existing Windows API and more. In particular, it will include a number of frequently used libraries now available through separate COM objects. The XML and ADO object models have been integrated in a subtree of classes called ADO.NET

Advantages of ADO.Net over ADO

The advantages of ADO.NET over ADO are as follows:

- ADO.NET is the base that will form the foundation of data-aware .NET applications
- Unlike ADO, ADO.NET has been purposely designed for more general, and less database-oriented, guidelines
- ADO.NET gathers all the classes that allow data handling. Such classes represent data container objects that feature typical database capabilities—indexing, sorting, views

- ADO.NET is the definitive solution for .NET database applications, it shows off an overall design that is not as database-centric as the ADO model
- ADO.NET is quite different from ADO. It is a new data access programming model that requires full understanding, commitment, and a different mindset
- ADO.NET is not ADO adapted to fit into the .NET infrastructure. This is apparent when looking at ADO.NET in terms of syntax, code design, and migration

9.9 Short Summary

- Data are facts concerning people, places, events or other objects or concepts
- Information is data that have been processed and refined and then given in the format that is convenient for decision making or other organizational activities
- A database is a shared collection of interrelated data designed to meet the varied information needs of an organization
- A DBMS (Database Management System) is a computerized record-keeping system that stores, maintains and provides access to information
- SQL also allows users to define the data in a database, and manipulate that data
- SQL is mainly classified into Data Definition Language (DDL) and Data Manipulation Language (DML)
- One of the primary uses of SQL is to update the data contained in a database
- Functions are used to compute against a column or columns of data returned from a query
- ADO.NET is the latest among of the database access technologies that started with the ODBC API

9.10 Brain Storm

1. What is a database?
2. What are the advantages of using a database to store data?
3. What is a DBMS?
4. What is SQL?
5. What are DDL and DML statements?
6. What is a function?
7. List the advantages of ADO over ADO.NET.



Lecture 10

ADO.NET - II

Objectives

In this lecture you will learn the following

- + What is meant by Managed Provider?
- + Learning about ADO connection object
- + What is the user of command object?

Coverage Plan

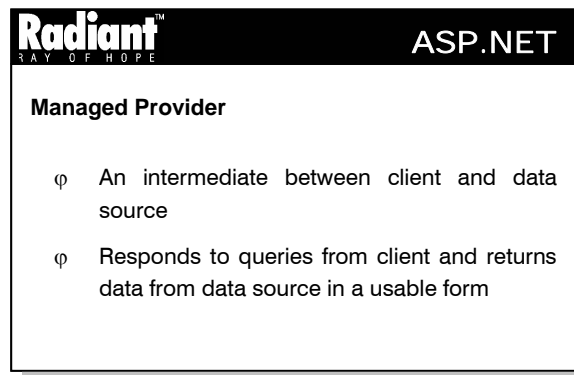
Lecture 10
10.1 Snap Shot
10.2 Managed Providers
10.3 Connection Object
10.4 Command Object
10.5 DataReader
10.6 Short Summary
10.7 Brain Storm

10.1 Snap Shot

This session is designed to introduce the user to Manager Providers, Command and Connection objects. It also introduces the DataReader, a forward-only stream used to store the retrieved data.

The Managed Provider acts as an intermediate between the client and the data source. The Connection object is used to connect to the data source. The Command object is used to create and execute commands against the data source so that the data is retrieved in an useful form to the client. The DataReader is a forward-only stream which is used to store the retrieved data, for use in the application.

10.2 Managed Providers



The **Managed Provider** is an intermediate between the client and the data source. It responds to queries from the client and returns data from the data source in a usable form. The .NET Framework provides two types of Providers:

- ADO
- SQL

The ADO Provider is used to connect to any data source, whereas the SQL Provider is specially designed for use with the SQL database. The support for the ADO Provider is given by the **System.Data.ADO** Namespace. The support for the SQL Provider is given by the **System.Data.SQL** Namespace.

The important classes contained in the **System.Data.ADO** Namespace are:

- ADOCommand
- ADOConnection
- ADODataReader
- ADODataSetCommand
- ADOError
- ADOErrors
- ADOException
- ADOParameter
- ADOParameters
- ADOProperty
- ADOProperties

The important classes contained in the **System.Data.SQL** Namespace are:

- SQLCommand

- SqlConnection
- SqlDataReader
- SQLDataSetCommand
- SQLError
- SQLErrors
- SQLException
- SqlParameter
- SQLParameters

The usage of the above classes will be dealt with while discussing the corresponding objects.

The following figure (Figure 10.1) shows the object model of ADO.NET

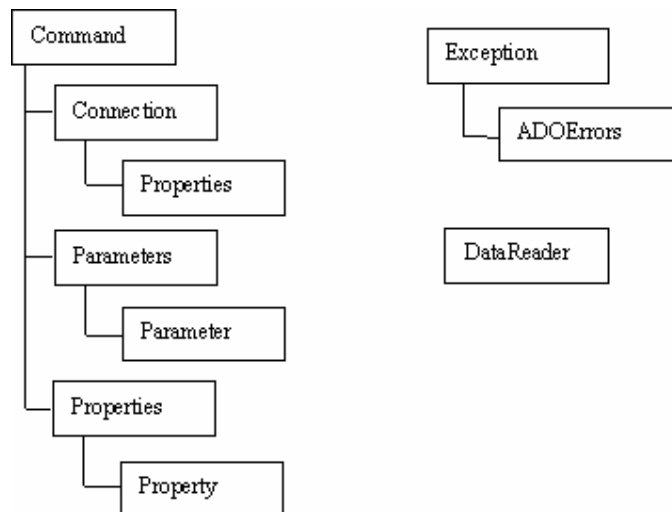


Figure 10.1

In the above figure, Properties, Parameters and ADOErrors represent collections.

Radiant™
3 A Y O F H O P E

ASP.NET

Parameter object

- ☐ Represents a parameter or argument associated with a Command object
- ☐ Enables reuse of command

Property object

- ☐ Contains metadata and setting for a property in an ADO data source

ADOException

- ☐ Instance created when ADO encounters a situation that it cannot handle

The **parameter** object represents a parameter or argument associated with a Command object based on a parameterized query or stored procedure. The parameter object enables reuse of the same command, by altering only the values passed to it. For example, by using the parameter object, an insert statement can be reused by changing the values of the fields passed as parameters to the command. The **property** object

contains the metadata and settings for a property in an ADO data source. When ADO encounters a situation that it cannot handle, it creates an instance of the **ADOException** class. Any operation involving the objects in ADO.NET can generate one or more provider errors. Whenever such errors occur, the ADO creates an instance of the **ADOError** class. These instances are created and managed by the **ADOErrors** class which in turn is created by the **ADOException** class.

The following paragraphs deal with the Connection, Command and DataReader objects.

10.3 Connection Object

Radiant™
3 A Y O F H O P E

ASP.NET

ADO Connection object

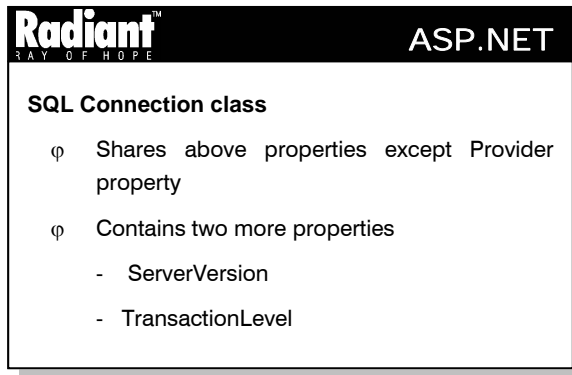
- φ Used to connect to database
- φ Represents a unique session with a data source
- φ Used to
 - Define information needed by ADO
 - Define transactional capabilities of session
 - Allow creation and execution of commands
 - Provide information about design of underlying data source

There are four basic operations involved in creating an ADO application. They are: retrieving data, examining data, editing data and updating data. To get the data from the database, a connection has to be established. The connection to the database is established using the **ADOConnection** object. A connection object represents a unique session with a data source. It is used to:

- Define the information needed by the ADO to communicate with data sources and create sessions
- Define the transactional capabilities of the session
- Allow the creation and execution of commands against the data source
- Provide information about the design of the underlying data source

The properties of the ADOConnection class are:

- **ConnectionString** – Indicates the string to be used to open a data store which includes the username, password etc.
- **ConnectionTimeout** – Denotes the time to wait to establish a connection, before terminating the attempt and generating an error
- **Database** – Represents the database to be used for the connection
- **DataSource** – Represents the source of the data to which the connection has to be established
- **IsolationLevel** – Represents the isolation level used for local transactions
- **Password** – Represents the password to be used while establishing the connection
- **Provider** – Indicates the name of the provider to be used for connecting to the data source
- **Site** – Denotes the site of the Component
- **State** – Denotes the current state of the connection
- **UserID** – Specifies the user ID to be used while connecting



The SqlConnection class also shares the above properties except the **Provider** property. The SQL Server Managed Provider uses the private protocol called *tabulardata stream* to communicate with SQL Server. It does not use OLEDB, ADO or ODBC. So, the Provider property is not present in this class.

Instead, the SqlConnection class contains two more properties listed below:

- **ServerVersion** – Denotes the version of the SQL Server to which the connection has been established
- **TransactionLevel** – Specifies the transaction level

An instance of the ADOConnection class can be created using one of the following two constructors:

- **ADOConnection()** – Creates a new instance of ADOConnection with no parameters
- **ADOConnection(string connectionString)** – Creates a new instance of ADOConnection with the specified connection string that indicates the server in which the data is stored, user name, password etc.

An instance of the SqlConnection class can be created using one of the following four constructors:

- **SqlConnection()** – Creates a new instance of the SqlConnection class with no parameters
- **SqlConnection(string connectionString)** – Creates a new instance of the SqlConnection class using the specified connection string that indicates the server in which the data is stored, user name, password etc.
- **SqlConnection(string dataSource, string userID, string password)** – Creates a new instance of the SqlConnection to the specified dataSource, using the specified user ID and password
- **SqlConnection(string dataSource, string userID, string password, string database)** – Creates a new instance of the SqlConnection to the specified database in the specified dataSource using the specified user ID and password

Some of the important methods common to both the classes are listed below:

- **Open** – Opens a database connection with the current settings
- **BeginTransaction** – Begins a database transaction
- **SaveTransaction** – Saves a point within the transaction
- **RollbackTransaction** – Rolls back a transaction from a pending state
- **CommitTransaction** – Commits the transaction

- **Dispose** – Disposes of the current SqlConnection object
- **Close** – Closes the connection to the database

The following part of code illustrates how to connect to a database using the SqlConnection class:

```
String connStr = "server = localhost; uid = swathy ; pwd = radiant;
    database = acctmaster";
SqlConnection conn = new SqlConnection(connStr);
conn.Open();
```

In the above lines, the SQL Server resides in the same machine as the client. So, the server is mentioned as *localhost*. The user id (uid) is *swathy* and password (pwd) is *radiant*. The database to which the connection has to be established is *acctmaster*.

Note: While using SqlConnection, the provider need not be specified since the default SQL Provider will be used. But, while using ADOConnection, the provider needs to be specified.



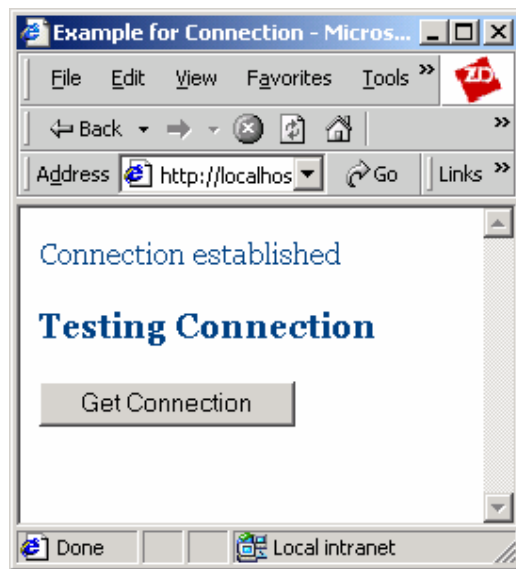
Practice 10.1

The following example uses the SqlConnection object to connect to the **master** database.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>
<title> Example for Connection </title>
<script language="C#" runat="server">
SqlConnection myConnection;
void Connect(Object Src, EventArgs E)
{
    myConnection = new SqlConnection(
        "server=localhost;uid=sa;pwd=;database=master");
    myConnection.Open();
    Response.Write("Connection established");
    myConnection.Close();
}
</script>
<body>
    <form runat="server">
        <h3>Testing Connection</h3>
        <asp:button OnClick="Connect" Text="Get Connection" runat="server" />
    </form>
</body>
</html>
```





In the above code, when the user clicks on the “Get Connection” button, its **OnClick** event is fired. The event calls the **Connect** method that in turn establishes the connection with the SQL Server database named **master** that is present in the same system as the client as the user **sa**. All these are specified in the connection string when an instance of **SqlConnection** is created. The **Open** method of the connection object actually establishes the connection.

10.4 Command Object

Radiant™
RAY OF HOPE

ASP.NET

Command Object

- ⌘ Used to request any type of operation from the provider
- ⌘ Very useful when a command is to be reused or needs to receive input parameters
- ⌘ Properties common to ADOCommand and SqlCommand classes

Once a connection to a database has been established, the queries have to be sent and the results retrieved. This is achieved through the Command object. The Command object can be used to request any type of operation from the provider, if the provider understands the command string properly. It is not necessary for a command to be executed only through the Command object. The command object is very useful when a command is to be reused or needs to receive input parameters.

The following are the properties of the Command object common to both the ADOCommand and SqlCommand classes:

- **ActiveConnection** – Represents the ADOConnection / SqlConnection used by the current instance of ADOCommand / SqlCommand

- **CommandBehavior** – Represents the behavior that the ADOCommand / SQLCommand should exhibit when executed
- **CommandText** – Denotes the SQL text command to run against the data source
- **CommandTimeout** – Denotes the time to wait before which the attempt to execute the command is terminated and an error is generated
- **CommandType** – Represents how the CommandText property is interpreted
- **Parameters** – Represents the collection of ADOParameters / SQLParameters
- **RecordsAffected** – Represents the number of records affected by the execution of the command
- **UpdatedRowSource** – Represents how the command results are applied when they are used by the Update method

The methods of the SQLCommand class are:

- **Clone** – Creates a duplicate of the current instance
- **Equals** – Checks if the specified object is the same instance as the current object
- **Execute** – Executes the specified CommandText against the ActiveConnection and builds a DataReader
- **ExecuteNonQuery** – Executes a SQL command that does not return any rows against the ActiveConnection
- **GetHashCode** – Serves as a hash function for a particular type
- **GetType** – Gets the Type of the object
- **ResetParameters** – Resets the parameters within the parameters collection
- **ToString** – Returns a string representing the current object

In addition to the above methods, the ADOCommand class contains the following methods:

- **Cancel** – Cancels the execution of a command
- **Prepare** – Creates a compiled version of the command on the data source
- **ResetCommandTimeout** – Resets the CommandTimeout property to the default value
- **ShouldPersistCommandTimeout** – Determines whether the CommandTimeout property should remain

There are five constructors in the SQLCommand class that can be used to create a command object. They are:

- **SQLCommand()** – Creates a new instance of SQLCommand without any parameters
- **SQLCommand(string cmd)** – Creates a new instance of SQLCommand using the specified command text
- **SQLCommand(string cmd, SqlConnection conn)** – Creates a new instance of SQLCommand using the specified connection object and command text
- **SQLCommand(string cmd, string connectionString)** – Creates a new instance of SQLCommand using the specified connection string and command text
- **SQLCommand(SqlConnection activeConnection, string cmd, CommandType type, SqlParameter[] param, UpdateRowSource src)** – Creates a new instance of SQLCommand using the specified connection, command text, parameters, source for updaterow and of the specified command type.

The ADOCommand class contains five similar constructors to create an ADOCommand object. The fifth constructor alone contains an additional boolean parameter that determines whether or not the command uses named parameters. The constructor is shown below:

ADOCommand(ADOConnection activeConnection, string cmd, CommandType type, bool namedparams, SQLParameter[] param, UpdateRowSource src)

The following lines show how to create a command object:

```
String st = "select * from emp";
SqlCommand cmd = new SqlCommand(st, conn);
```

The lines above create a command with the query string "select * from emp" and with the connection object "conn".



Practice 10.2

The following example uses the SqlCommand object to insert a record into the **emp** table of the **master** database.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

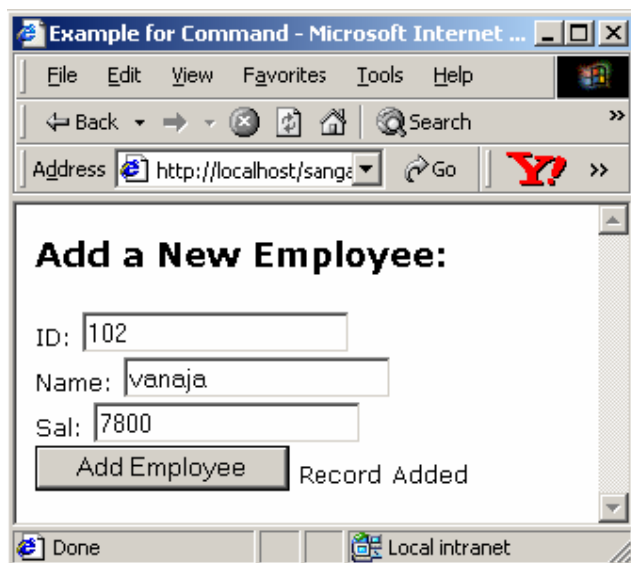
<html>
<title> Example for Command </title>
<script language="C#" runat="server">
SqlConnection myConnection;
protected void Page_Load(Object Src, EventArgs E)
{
    myConnection = new SqlConnection
        ("server=localhost;uid=sa;pwd=;database=master");
}

public void AddEmp(Object sender, EventArgs E)
{
    if(eid.Value == "" || ename.Value == "" || esal.Value == "")
    {
        lb.Text = "ERROR: Null values not allowed for
            employee ID, Name or salary";
        return;
    }
    String insertCmd = "insert into emp values (@eno, @eName, @sal)";
    SqlCommand myCommand = new SqlCommand(insertCmd, myConnection);
    myCommand.Parameters.Add(new SqlParameter("@eno",
        SqlDbType.SmallInt, 5));
    myCommand.Parameters["@eno"].Value = eid.Value;
    myCommand.Parameters.Add(new SqlParameter("@eName",
        SqlDbType.VarChar, 40));
    myCommand.Parameters["@eName"].Value = ename.Value;
    myCommand.Parameters.Add(new SqlParameter("@sal",
        SqlDbType.SmallMoney, 20));
    myCommand.Parameters["@sal"].Value = esal.Value;
    myCommand.ActiveConnection.Open();
}
```

```

        try
        {
            myCommand.ExecuteNonQuery();
            lb.Text = "Record Added";
        }
        catch (SQLException e)
        {
            if (e.Number == 2627)
                lb.Text = "ERROR: A record already exists with
                           the same primary key";
            else
                lb.Text = "ERROR: Could not add record, please ensure
                           the fields are correctly filled out";
        }
        myCommand.ActiveConnection.Close();
    }
</script>
<body style="font: 10pt verdana">
    <form runat="server">
        <h3>Add a New Employee:</h3>
        ID: <input type="text" id="eid" value="0" runat="server"> <br>
        Name: <input type="text" id="ename" value="" runat="server"> <br>
        Sal: <input type="text" id="esal" value="0" runat="server"> <br>
        <input type="submit" OnServerClick="AddEmp" value="Add Employee"
            runat="server">
        <asp:label id="lb" runat="server" />
    </form>
</body>
</html>

```



The above example receives the employee ID, name and salary from the user to add a new record to the **emp** table of the **master** database. The connection object is created when the page is loaded. When the user enters the required information and clicks the “Add Employee” button, its **OnServerClick** event is fired. This event invokes the **AddEmp** method. It checks if any of the input values is empty. If so, it prints out an appropriate message and returns without inserting the record. If not, a command object is created using the connection defined in the **Page_Load** event. Values are passed to the object using the **Parameters** collection. Then the command is executed and the record is added to the table.

10.5 DataReader

Radiant™
RAY OF HOPE

ASP.NET

DataReader

- ☐ A read-only, forward-only stream returned from the database
- ☐ At a time, only one record is held in memory
- ☐ Every time a record is read, internal cursor points to next record
- ☐ A series of Get methods enable user to access field values
- ☐ An instance is created through Execute method
- ☐ Fully utilizes the corresponding connection object

Data can be retrieved from the database and held in memory for usage by the application. But when a large amount of data is retrieved, maintaining the memory used by the data for a long time will be inefficient. In this case it is useful to make use of the DataReader. The DataReader can also be used when the user has to iterate over the stream of data retrieved from the database.

The DataReader is a read-only, forward-only stream returned from the database. At a time, only one record is held in memory. Every time a record is read, the internal cursor will automatically point to the next record. A series of Get methods provided by the DataReader enable the user to access the field values.

An instance is created for **ADODataReader** through the **Execute** method of the **ADOCommand** class and not through its constructor. Further, when the ADODataReader is being used, the corresponding ADOConnection is fully utilized by it. So, no other operation (except the Close operation) can be performed using the connection until the ADODataReader is closed. The same holds true for **SQLDataReader**.

The properties of both the classes are as follows:

- **FieldCount** – Represents the number of fields in the current record
- **HasMoreResults** – Indicates whether or not there are more results to be read
- **HasMoreRows** – Indicates whether or not there are more rows to be retrieved
- **IsClosed** – Indicates whether or not the DataReader is closed
- **Item** – Indicates the column value in its native format

The ADODataReader class contains one more property named **RowFetchCount** that indicates the number of rows to be retrieved at one time.

The following methods are common to both `ADODataReader` and `SQLDataReader`:

- **Close** - Closes the data reader object
- **Equals** - Determines whether the specified Object is the same instance as the current Object
- **GetBoolean** - Returns the value of the specified column as a boolean
- **GetByte** - Returns the value of the specified column as a byte
- **GetBytes** - Returns the value of the specified column as a byte array
- **GetChar** - Returns the value of the specified column as a character
- **GetChars** - Returns the value of the specified column as a character array
- **GetDataTypeName** - Returns the name of the back-end data type
- **GetDateTime** - Returns the value of the specified column as a `DateTime` object
- **GetDecimal** - Returns the value of the specified column as a `Decimal` object
- **GetDouble** - Returns the value of the specified column as a double-precision floating point number
- **GetFieldType** - Returns the data type of the object
- **GetFloat** - Returns the value of the specified column as a single-precision floating point number
- **GetHashCode** - Serves as a hash function for a particular type
- **GetInt16** - Returns the value of the specified column as a 16-bit signed integer
- **GetInt32** - Returns the value of the specified column as a 32-bit signed integer
- **GetInt64** - Returns the value of the specified column as a 64-bit signed integer
- **GetName** - Returns the name of the specified column
- **GetOrdinal** - Returns the column ordinal, given the name of the column
- **GetString** - Returns the value of the specified column as a string
- **GetType** - Returns the type of the object
- **GetValue** - Returns the value of the specified column in its native format
- **GetValues** - Returns all the attribute fields in the collection for the current record
- **IsNull** - Checks for non-existent values
- **NextResult** - When reading the results of batch SQL statements, this method advances the data reader to the next result
- **Read** - Advances the data reader to the next record
- **ToString** - Returns the String representing the current Object

The important methods specific to the `SQLDataReader` class are:

- **GetSQLBinary** - Returns the `SQLBinary` value of the specified column
- **GetSQLBit** - Returns the `SQLBit` value of the specified column
- **GetSQLGuid** - Returns the `SQLGuid` value of the specified column
- **GetSQLMoney** - Returns the `SQLMoney` value of the specified column
- **GetSQLNumeric** - Returns the `SQLNumeric` value of the specified column
- **GetSQLSingle** - Returns the `SQLSingle` value of the specified column

The important methods specific to **ADODataReader** class are:

- **GetSByte** - Returns the value of the specified column as a SByte
- **GetGuid** - Returns the value of the specified column as a globally-unique identifier.



Practice 10.3

The following example uses DataReader to retrieve and display the records of the table **emp** in the **master** database.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

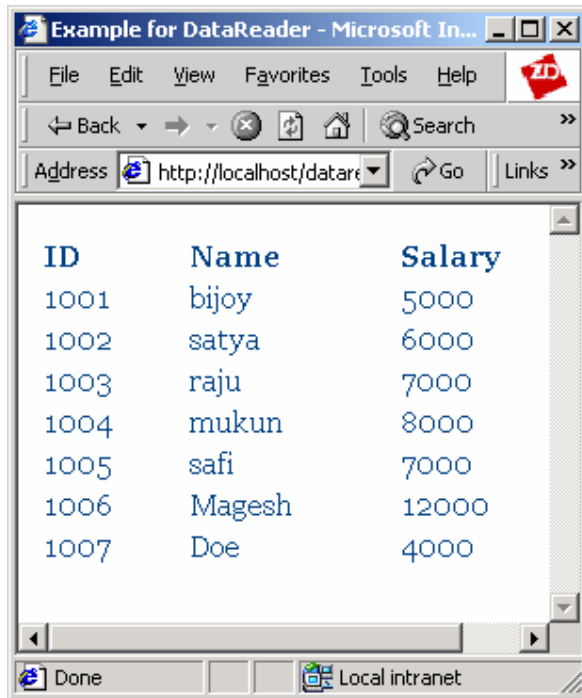
<html>
<title> Example for DataReader </title>
<script language="C#" runat="server">

SQLConnection myConnection;
protected void Page_Load(Object Src, EventArgs E)
{
    myConnection = new SQLConnection(
        "server=localhost;uid=sa;pwd=;database=master");
    SqlCommand myCommand = new SqlCommand("select distinct * from
        emp", myConnection);
    myConnection.Open();
    SQLDataReader dr;
    myCommand.Execute(out dr);
    Response.Write("<Table width=300>");
    Response.Write("<TR>");
    Response.Write("<TD>");
    Response.Write("<B>ID </B>");
    Response.Write("</TD>");
    Response.Write("<TD>");
    Response.Write("<B>Name </B>");
    Response.Write("</TD>");
    Response.Write("<TD>");
    Response.Write("<B>Salary </B>");
    Response.Write("</TD>");
    Response.Write("</TR>");

    while (dr.Read())
    {
        Response.Write("<TR>");
        Response.Write("<TD>");
        Response.Write(dr["eno"].ToString() + " ");
        Response.Write("</TD>");
        Response.Write("<TD>");
        Response.Write(dr["ename"].ToString() + " ");
        Response.Write("</TD>");
        Response.Write("<TD>");
        Response.Write(dr["sal"].ToString() + " ");
        Response.Write("</TD>");
        Response.Write("</TR>");
    }
    myConnection.Close();
}
</script>
```



```
</html>
```



The above example establishes a connection with the **master** database. It then executes a select command to retrieve rows from the **emp** table and stores the retrieved rows in the SqlDataReader **dr**. Then it reads each row from it and displays in a table.

10.6 Short Summary

- The Managed Provider acts as an intermediate between the client and the data source
- ADO.NET provides two types of providers: ADO and SQL
- The parameter object represents a parameter or argument associated with a Command object based on a parameterized query or stored procedure
- The property object contains the metadata and settings for a property in an ADO data source
- When the ADO encounters a situation that it cannot handle, it creates an instance of the ADOException class
- A connection object represents a unique session with a data source
- The Command object can be used to request any type of operation from the provider, if the provider understands the command string properly
- The DataReader is a read-only, forward-only stream returned from the database. It is useful when the user has to iterate over the stream of data retrieved from the database

10.7 Brain Storm

1. What is a Managed Provider?
2. What is the role of the property object in ADO?
3. What are the kinds of commands that can be executed using the Command object?
4. What is a DataReader?
5. When an ADODataReader is open, can the corresponding Connection object be used to execute some other command?

❧❧❧

Lecture 11

ADO.NET - III

Objectives

In this lecture you will learn the following

- + What is meant by DataSet?
- + Learning about DataTable
- + Knowing about DataView

Coverage Plan

Lecture 11
11.1 Snap Shot
11.2 DataSet
11.3 DataTable
11.4 DataView
11.5 Short Summary
11.6 Brain Storm

11.1 Snap Shot

This session introduces the user to DataSet, DataRelation, DataTable and DataView.

The DataSet is designed to handle the actual data from a data store and provides access to multiple tables, rows and columns. Each DataSet can contain multiple tables and maintain the relationship between them. The advantage of the DataSet is that since it is designed for disconnected data, multiple tables can be simultaneously passed around the tiers of a Web application, along with the relationships.

DataView is a custom view of data table, whose prime purpose is to aid in data binding. DataView is the equivalent of ADO RecordSet.

Note: A DataSet is not a RecordSet. In terms of analogies, a DataView is more like a RecordSet.

11.2 DataSet

Radiant™
RAY OF HOPE

ASP.NET

Data set

- ⌘ Contains any number of data tables
- ⌘ Constitutes a “disconnected” view of database data
- ⌘ Enables greater scalability
- ⌘ Resides in System.Data namespace

The centerpiece of any software solution using ADO.NET is the DataSet. A DataSet is an in-memory copy of the database data. A DataSet contains any number of data tables, each of which typically corresponds to a database table or view. A DataSet constitutes a “disconnected” view of the database data. That is, it exists in memory without an active connection to a database containing the corresponding tables or views. This disconnected architecture enables greater scalability by only using database server resources when reading or writing from the database.

The DataSet class resides in **System.Data** namespace. The DataSet object model is shown in Figure 11.1.

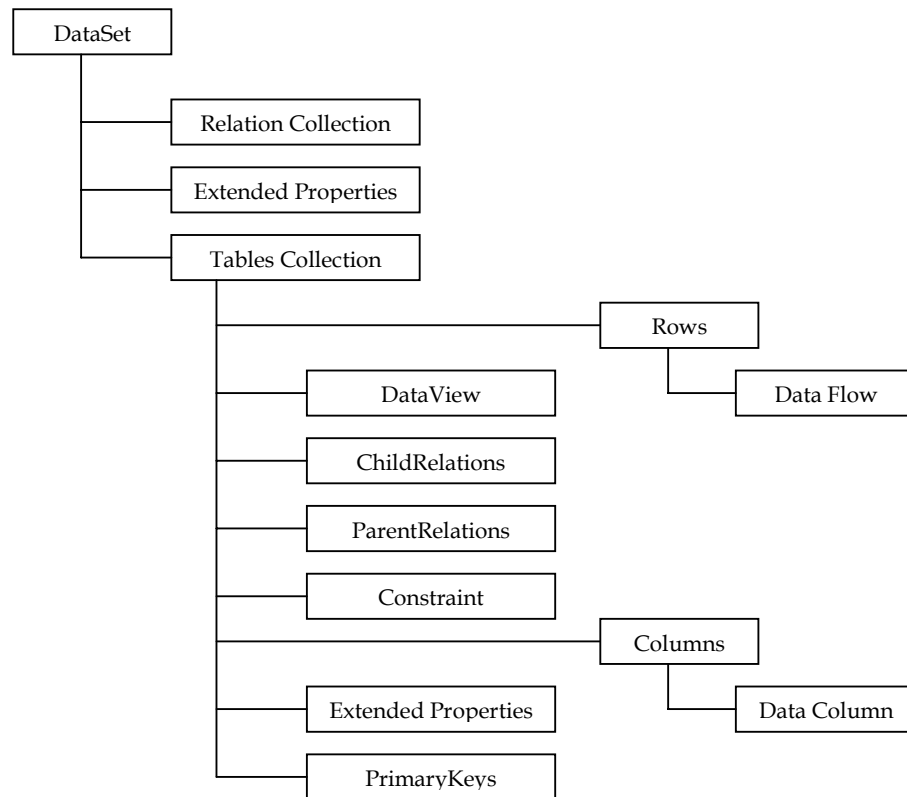


Figure 11.1

TablesCollection object

An ADO.NET DataSet is a collection of one or more tables represented by DataTable objects. The TablesCollection object contains all the DataTable objects in a DataSet. A DataTable is defined by System.Data and represents a single table of memory-resident data. It contains a collection of columns represented by the ColumnsCollection object, which defines the schema and rows of the table. It also contains a collection of rows represented by the RowsCollection object, which orders the data in the table. Along with the current state, a DataTable object retains its original state and tracks all changes that occur to the data. The DataSet is able to persist and reload its contents through XML.

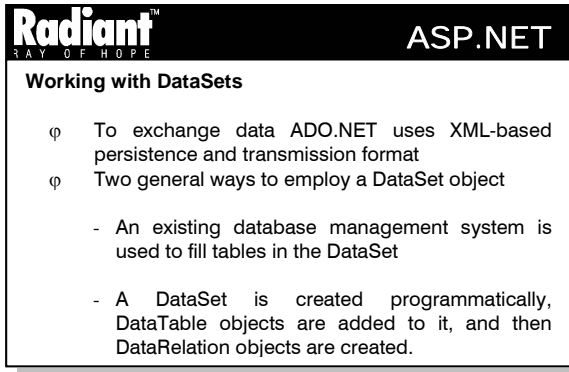
RelationsCollection object

A typical DataSet contains relationships contained by the RelationsCollection object. A relationship, represented by the DataRelation object, associates rows in one data table with rows in another data table. It is analogous to the foreign-key relationship in a relational database. A DataRelation identifies matching columns in two tables of a DataSet. Relationships that enable navigation from one table to another within a DataSet. The essential elements of a DataRelation are the name of the relationship, the two tables being related, and the primary key and foreign key columns in the tables. Relationships can also be built with more than one column per table, with an array of DataColumn objects for the primary and foreign keys. When a DataRelation is created, ADO.NET verifies if the relationship can be established. Once ADO.NET adds a relationship to the RelationsCollection, it disallows any changes that would invalidate the relationship.

ExtendedProperties

The ExtendedProperties object contains customized user information, such as a password or the time when data should be refreshed for a DataSet object.

Working with DataSets



At run time, data will be passed from the database to a middle-tier business object and then down to the user interface. To accommodate the exchange of data, ADO.NET uses an XML-based persistence and transmission format. That is, to transmit data from one tier to another, an ADO.NET solution expresses the in-memory data (the DataSet) as XML and then sends the XML to the other component.

There are two general ways to employ a DataSet object.

- An existing database management system, such as SQL Server can be used to fill tables in the DataSet. In this method, one SQLDataSetCommand can be used per table to fill the DataTable object with data. However, the DataRelation objects should be created between each table
- A DataSet can be programmatically created, DataTable objects can be added to it, and then DataRelation objects can be created to link each table

On the client application, the data is displayed using a combination of controls, such as the DataGrid. The user can add, delete, or edit the data. After the user finishes working with the data, the DataSet is again converted into an XML document for transmitting back to the server component. DataSet, the middle tier component, is created using an adapter. The DataSet is then converted into an XML document for transport back to the requester.

The DataSet stores data using COM+ types. The COM+ decimal type allows a maximum of 28 significant digits, while the SQL decimal type allows 38 significant digits. Currently, the FillDataSet method throws an exception if it encounters a valid SQL decimal with more than 28 significant digits. Currently there are currently no ways to get a SQL decimal of greater than 28 significant digits into a DataSet object. If the application requires the added precision, a SQLDataReader object can be used. The GetSQLNumeric method can then be called to get the SQL decimal value.

Figure 11.2 shows the working of DataSet in ADO.NET solution.

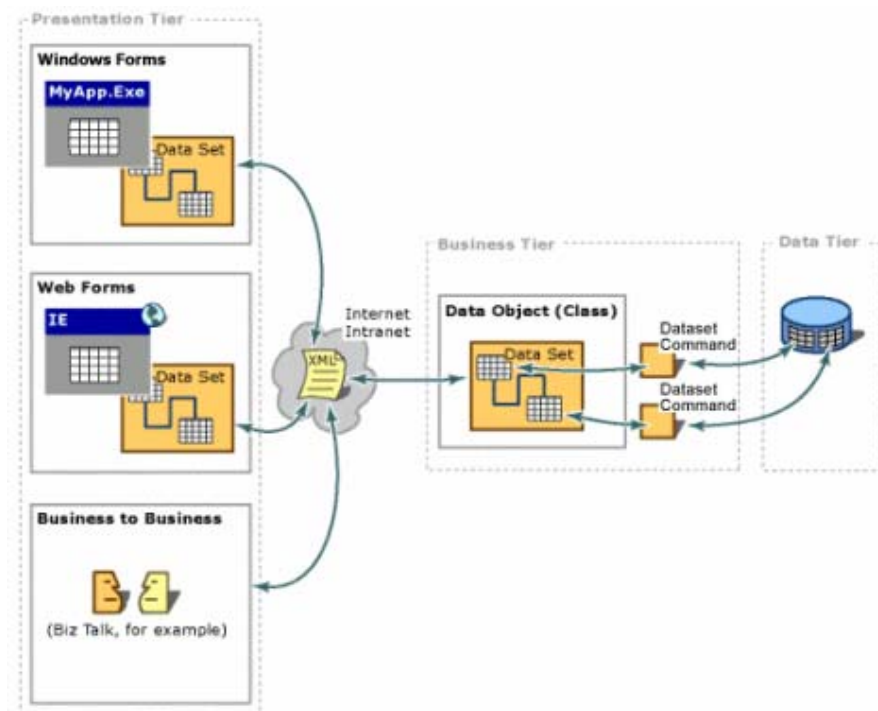


Figure 11.2

Constructors of DataSet

There are two constructors for a DataSet.

The first constructor initializes a new instance of the DataSet class, as shown below:

```
public DataSet();
```

The second constructor initializes a new instance of a DataSet class with the given name, as given below:

```
public DataSet(string);
```



Practice 11.1

The following application performs a select query from a SQL database bounded to a DataGrid.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object Src, EventArgs E)
{
    SqlConnection con = new SqlConnection
        ("server=localhost;uid=sa;pwd=");
```



```

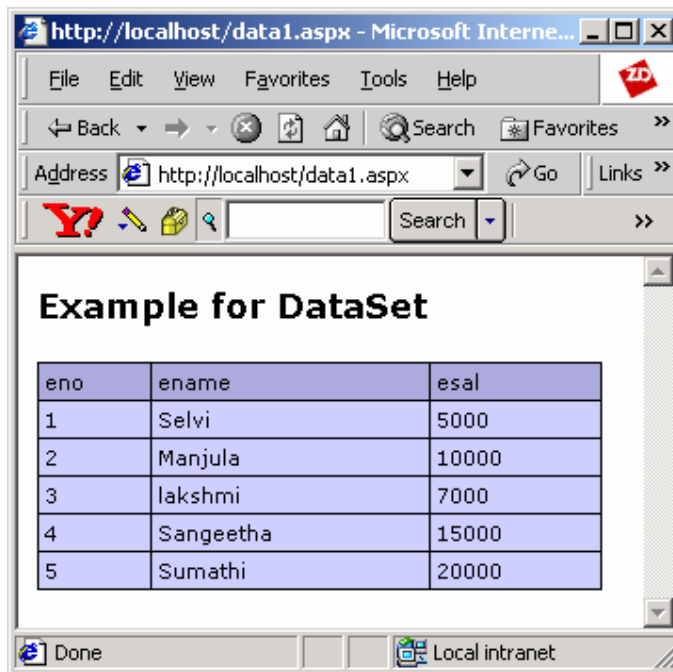
SQLDataSetCommand com = new SQLDataSetCommand("select * from
        emptab", con);
DataSet ds = new DataSet();
com.FillDataSet(ds, "emptab");
d.DataSource=ds.Tables["emptab"].DefaultView;
d.DataBind();
}
</script>
<body>

<h3><font face="Verdana">Example for DataSet</font></h3>

<ASP:DataGrid id="d" runat="server"
    Width="300"
    BackColor="#ccccff"
    BorderColor="black"
    CellPadding=3
    CellSpacing="0"
    Font-Name="Verdana"
    Font-Size="8pt"
    HeaderStyle-BackColor="#aaaadd"
    />

</body>
</html>

```



In the above example, the connection object **con** is created in the **Page_Load** event. A **SQLDataSetCommand** object **com** is also created with the query to retrieve all the records from the **emptab** table. The command is executed and the **DataSet ds** is filled with the resulting rows using the **FillDataSet()** method. Then, the **DataSource** property of the **DataGrid** is set to the **DataSet**. Finally, the **DataBind()** method is called to populate the **DataGrid** with the values.



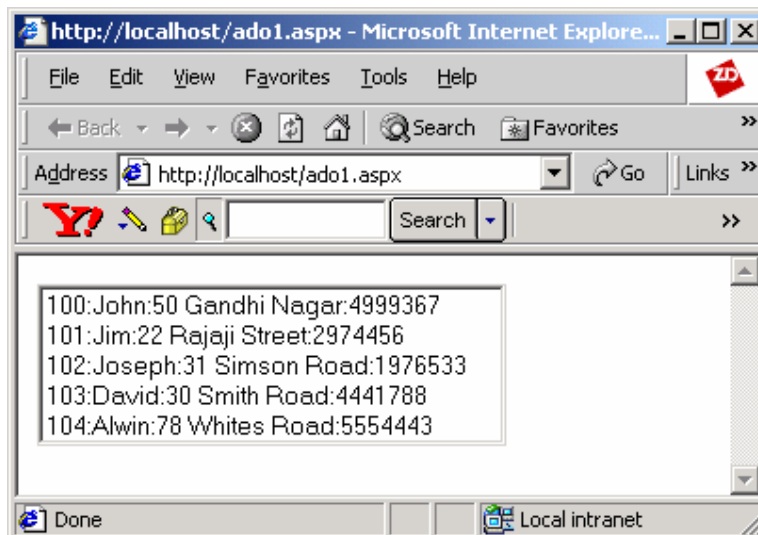
Practice 11.2

The following application executes a select query from a database using ADOConnection and ADODatasetCommand classes and displays the data in a listbox.

```
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

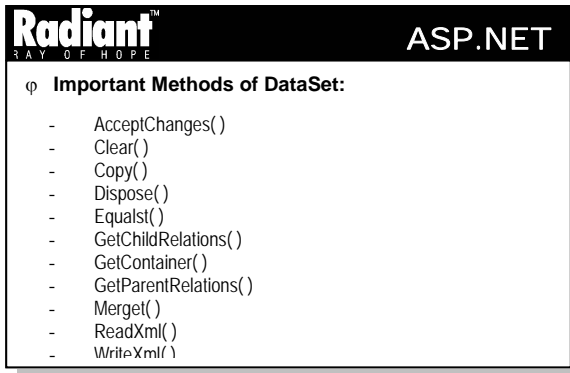
<script language="C#" runat="server">
void Page_Load(Object Src, EventArgs E )
{
    string s= "Provider=Microsoft.Jet.OLEDB.4.0;Data Source =
                E:/Inetpub/wwwroot/emp.mdb ";
    ADOConnection con = new ADOConnection(s);
    ADODatasetCommand com = new ADODatasetCommand("select * from
                customer", con);
    DataSet ds = new DataSet();
    com.FillDataSet(ds,"customer");
    DataTable dt = ds.Tables["customer"];
    foreach(DataRow dr in dt.Rows)
    {
        l.Items.Add(dr["CusID"] + ":" + dr["CusName"] + ":" +
                    +dr["CusAdd"]+"::"+dr["CusPho"]);
    }
}

</script>
<asp:Listbox id=l runat=server/>
</html>
```



In the above example, a connection with the database has been created using `ADOConnection`. The Query has been passed to `ADOSetCommand`. The `DataSet` is populated with the results from the query using the **FillDataSet** method. The result is then viewed in a listbox using the **foreach** statement. The result can also be viewed by binding it to a `DataGrid`.

Methods of DataSet



The various methods of `DataSet` are discussed below:

- **AcceptChanges()** - Commits all the changes made to this `DataSet` since it was loaded or the last time `AcceptChanges` was called
- **Clear()** - Clears the `DataSet` of any data by removing all rows in all tables
- **Clone()** - Clones the structure of the `DataSet`, including all `DataTable` schemas, relations, and constraints
- **Copy()** - Copies both the structure and data for this `DataSet`
- **Dispose()** - Disposes of the Component
- **Equals()** - Determines whether the specified Object is the same instance as the current Object
- **GetChanges()** - Returns a copy of the `DataSet` containing all changes made to it since it was last loaded, or since `AcceptChanges` was called
- **GetChildRelations()** - Gets the collection of child relations which belong to a specified table
- **GetContainer()** - Returns the `IContainer` that contains the Component
- **GetHashCode()** - Serves as a hash function for a particular type, suitable for use in hashing algorithms and data structures like a hash table
- **GetParentRelations()** - Gets the collection of parent relations which belong to a specified table
- **GetType()** - Gets the Type of the Object
- **Merge()** - Merges this `DataSet` with a specified `DataSet`
- **ReadXml()** - Reads XML schema and data into the `DataSet`
- **ReadXmlData()** - Reads data from the specified XML source and maps the data into the `DataSet` object's schema
- **ReadXmlSchema()** - Reads an XML schema into the `DataSet`
- **ToString()** - Returns a String that represents the current Object
- **WriteXml()** - Writes the current schema and data for the `DataSet` to an XML document
- **WriteXmlData()** - Overloaded. Writes the data contained in the `DataSet` to an XML document

- **WriteXmlSchema()** - Overloaded. Writes the DataSet structure as an XML schema

Properties of DataSet

Radiant™
3 A Y O F H O P E
ASP.NET

φ **Important properties of DataSet:**

- DataSetName
- DefaultView
- ExtendedProperties
- NameSpace
- Relations
- Tables
- Xml
- XmlData

The properties of DataSet are as follows:

- **CaseSensitive** - gets or sets a value indicating whether string comparisons within datatable objects are case-sensitive
- **Datasetname** - gets or sets the name of the dataset
- **Defaultview** - gets a custom view of the data contained by the dataset, one that allows filtering, searching, and navigating through the custom data view
- **Extendedproperties** - gets the collection of custom user information
- **Haserrors** - gets a value indicating whether there are errors in any of the rows in any of the tables of the dataset
- **Locale** - gets or sets the locale information used to compare strings within the table
- **Namespace** - gets or sets the namespace of the dataset
- **Relations** - get the collection of relations that link tables and allow navigation from parent tables to child tables
- **Tables** - gets the collection of tables contained in the dataset
- **Xml** - gets or sets the xml data for the dataset
- **Xmldata** - gets or sets the xml data for the dataset
- **Xmlschema** - gets or sets the xml schema of the DataSet

Events of DataSet

Radiant™
3 A Y O F H O P E
ASP.NET

φ **Events of the DataSet**

MergeFailed

- Public event MergeFailedEventHandler MergeFailed;

PropertyChanged

- public event PropertyChanged EventHandler PropertyChanged

The various events of the DataSet are given below:

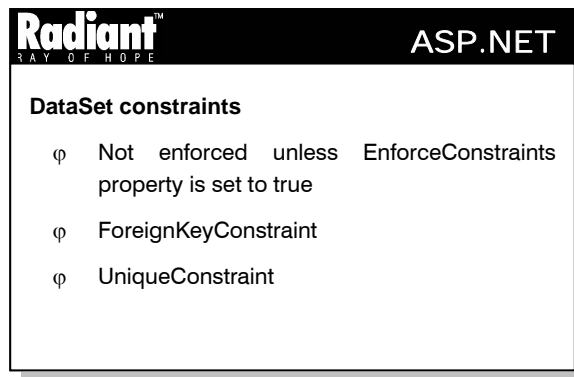
MergeFailed - Occurs when a target and source DataRow have the same primary key value, and EnforceConstraints is set to true. Its syntax is as follows:

```
public event MergeFailedEventHandler MergeFailed;
```

PropertyChanged - Occurs when a property value changes. Its syntax is as follows:

```
public event PropertyChangedEventHandler PropertyChanged;
```

DataSet Constraints



The DataSet constraints are as follows:

ForeignKeyConstraint

The ForeignKeyConstraint represents an action restriction enforced on a set of columns in a primary key/foreign key relationship when a value or row is either deleted or updated. In a parent/child relationship between two tables, deleting a value from the parent table can affect the child rows in one of the following ways:


- The child rows can also be deleted (a cascading action)
- The values in the child column (or columns) can be set to null values
- The values in the child column (or columns) can be set to default values
- An exception can be thrown

UniqueConstraint

The UniqueConstraint can be enforced to a DataTable when it is essential that all the values of a column must be unique. It can be added either to a single column or to an array of columns of the same DataTable.

Constraints are not enforced unless the EnforceConstraints property is set to true. When a DataSet is merged with a second DataSet, constraints are not enforced until all merges are completed.

DataRelation



Radiant™
RAY OF HOPE

ASP.NET

DataRelation

- ☞ Represents a parent/child relationship between two tables
- ☞ Associates rows in one table to rows in another
- ☞ Identifies matching columns in two tables of a DataSet
- ☞ DataSet can contain either related or unrelated data
- ☞ DataSet can handle both hierarchical and foreign key relationship

The DataRelation represents a parent/child relationship between two tables. The DataSet represents a complete set of data including related tables, constraints, and relationships among the tables. A typical dataset contains relationships contained by the RelationsCollection object. A relationship, represented by the DataRelation object, associates rows in one table to the rows in another table. It is similar to the foreign-key relationship in a relational database. A DataRelation identifies matching columns in two tables of a DataSet.

A DataSet can contain either related or unrelated data. It can be thought of as a document of Data. In fact, an XML data document is similar to it except for the fact that it is based on a hierarchical paradigm. Since data is often stored in relational databases, the DataSet can handle both hierarchical relationships and foreign key relationships. Relationships can also have different types of enforcement. By default, deletions and updates are cascaded. A DataSet contains a Relations collection. It is easy to add a relationship to this collection using the column or columns (in a multi-column key) of the related tables.



Practice 11.3

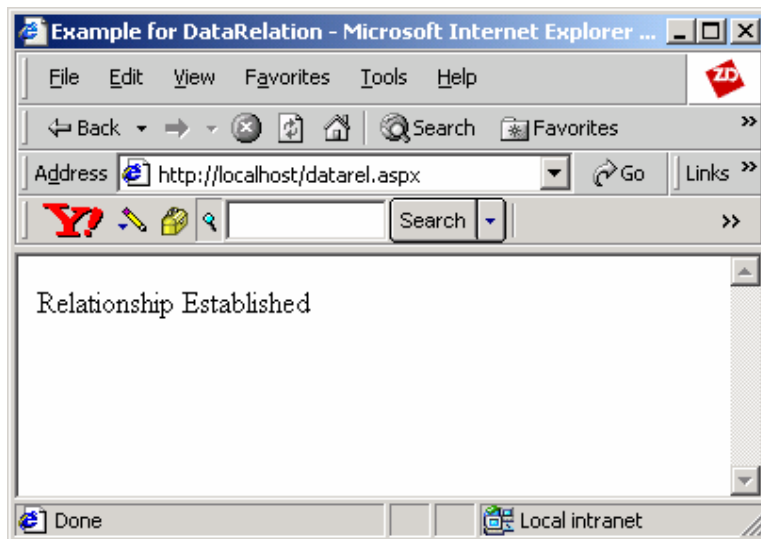
The example given below presumes that two DataTable objects exist in the DataSet. Both tables have a column named "CustomerID" which serves as the link between the two tables. The example adds a single DataRelation to the Relations collection of the DataSet object. The first argument ("CustOrders") specifies the name of the relationship. The second and third arguments are the DataColumn objects that link the two tables.

```
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<html>
<title> Example for DataRelation </title>
<script language="C#" runat="server">
public DataSet ds;
void Page_Load(Object Src, EventArgs E )
{
    string s="server=localhost;uid=sa;pwd='';database=master";
    SqlConnection con = new SqlConnection(s);
    SQLDataSetCommand com1 = new SQLDataSetCommand("select * from
        customers", con);
    SQLDataSetCommand com2 = new SQLDataSetCommand("select * from
```

```

        orders", con);
ds = new DataSet();
com1.FillDataSet(ds,"Customers");
com2.FillDataSet(ds,"Orders");
DataRelation dr;
dr = new DataRelation( "CustOrders",ds.Tables["Customers"].
        Columns["CustomerId"],
        ds.Tables["Orders"].Columns["CustomerId"]);
ds.Relations.Add(dr);
Response.Write("Relationship Established");
}
</script>
</html>

```



The example above constructs an instance **ds** of a DataSet and fills data into it from two SQLDataSetCommand objects **com1** and **com2**. Then a DataRelation **dr** is added to the Relations collection specifying its name, and the appropriate DataColumn objects as arguments. It adds a relation between the **CustomerId** Key on the **Customers** table and the **CustomerId** foreign key on the **Orders** table in the DataSet.

Data can be retrieved from the relationship defined in the example above using the lines:


```

foreach (DataRow Customer in ds.Tables["Customers"].Rows) {
Response.Write("Customer: " + Customer["ContactName"].ToString());
foreach (DataRow Order in Customer.GetChildRows(
        ds.Relations["CustOrders"])) {
    Response.Write("Order #" + Order["OrderId"].ToString());
}
}

```

A primary function of the `DataRelation` is to allow navigation from one table to another within the `DataSet`. In practice, this allows us to retrieve all the related `DataRow` objects in one table when given a single `DataRow` from a related table. The above example gives a `DataRow` from the `Customers` table. All the orders for a particular customer can be retrieved from the **Orders** table.

11.3 DataTable



3 A Y O F H O P E

ASP.NET

DataTable class

- ❖ Present in `System.Data` namespace
- ❖ Represents a table of in-memory data
- ❖ Central object in ADO.NET library
- ❖ Contains a collection of Constraint object

The `DataTable` class is present in the **System.Data** namespace. It represents a table of in-memory data. The `DataTable` is a central object in the ADO.NET library. Other objects which use the `DataTable` include the `DataSet` and the `DataRow`. The `DataTable` contains a collection of Constraint objects that can be used to ensure the integrity of the data.

Syntax

Public Class DataTable Inherits Component Implements IListSource, ISupportInitialize



Practice 11.4

This program displays the properties of a table such as its column names, types and column properties. It displays the result in a listbox.

```
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

<script language="C#" runat="server">

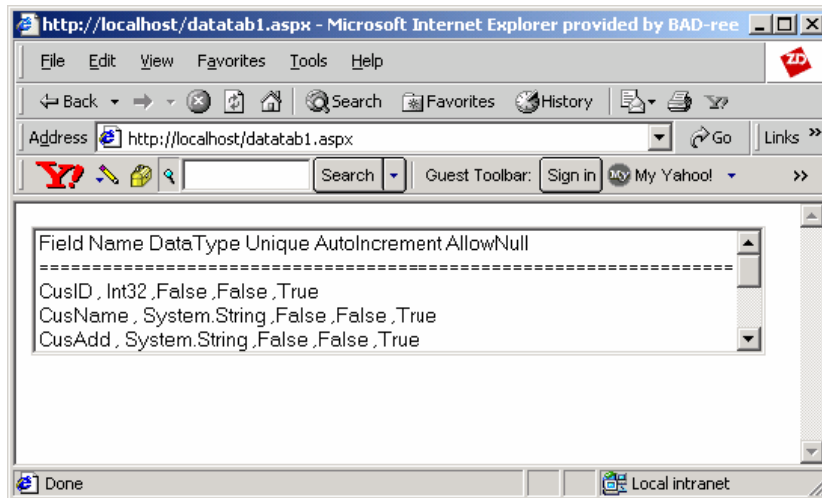
void Page_Load(Object Src, EventArgs E )
{
    string s= "Provider=Microsoft.Jet.OLEDB.4.0;Data Source =
                E:/Inetpub/wwwroot/emp.mdb " ;
    ADOConnection con = new ADOConnection(s);
    ADODataSetCommand com = new ADODataSetCommand("select * from customer", con);
    DataSet ds = new DataSet();
    com.FillDataSet(ds, "customer");
    DataTable dt = ds.Tables[0];
    l.Items.Add("Field Name DataType Unique AutoIncrement AllowNull");
    l.Items.Add("=====");
    foreach( DataColumn dc in dt.Columns )
    {
```



```


        l.Items.Add(dc.ColumnName+" , "+dc.DataType + " ,"+dc.Unique
                    +" ,"+dc.AutoIncrement+" ,"+dc.AllowNull );
    }
    foreach(DataRow dr in dt.Rows)
    {
        l.Items.Add(dr["CusID"] + ":" + dr["CusName"] + ":"
                    +dr["CusAdd"]+" ":"+dr["CusPho"]);
    }
}
</script>
<asp:Listbox id=l runat=server/>
</html>

```



The example above creates a connection with database using the `ADOConnection` object `con` and passes a query in an `ADOSetCommand` object `com`. It then populates the `DataSet ds` with the results from the query using the `FillDataSet` method. Then it displays the table properties such as its column names, types and column properties in the listbox `l` using the `foreach` statement. The `Unique`, `AutoIncrement` and `AllowNull` properties return a boolean value.

11.4 DataView



Radiant™
3 A Y O F H O P E

ASP.NET

DataView

- ☐ Represents a databindable, customized view of a DataTable
- ☐ Allows data binding on both Windows forms and Web Forms
- ☐ Can be customized to present a subset of data from DataTable
- ☐ Components of DataRow of a DataTable within a given DataView

The DataView represents a databindable, customized view of a DataTable for sorting, filtering, searching, editing, and navigation. A major function of DataView is to allow data binding on both Windows Forms and Web Forms. A DataView can be customized to present a subset of data from the DataTable. This capability allows to have two controls bound to the same DataTable, but showing different versions of the data. For example, one control may be bound to a DataView showing all the rows in the table, while the second control may be configured to display only the rows that have been deleted from the DataTable. The DataTable also has a DefaultView property, which returns the default DataView for the table.

The DataRow of a DataTable within a given DataView have three components: what the data was (the original state), what the data is now (the current state), and what the value of the data is going to be (the proposed state). When changes are made to the data, it affects the current state or proposed state, but its original state is left unaffected until it is changed explicitly by using the DataRow.AcceptChanges method.

Syntax

```
Public Class DataView Inherits Component Implements IList,
ICollection, IEnumerable, _ ISortedList, IIndexedlist, ITypedList
```



Practice 11.5

The following example creates a single DataTable with three columns and five rows using DataView.

```
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>
<script language="C#" runat="server">
void Page_Load(Object Src, EventArgs E)
{
    SqlConnection con = new SqlConnection(
        "server=localhost;uid=sa;pwd=");
    SQLDataSetCommand com = new SQLDataSetCommand("select * from
        emptab", con);
    DataSet ds = new DataSet();
    com.FillDataSet(ds, "emptab");
    DataView dv = new DataView(ds.Tables["emptab"]);
    s.InnerHtml = dv.Table.TableName;
}
```

```

        d.DataSource = dv;
        d.DataBind();
    }

</script>

<body>

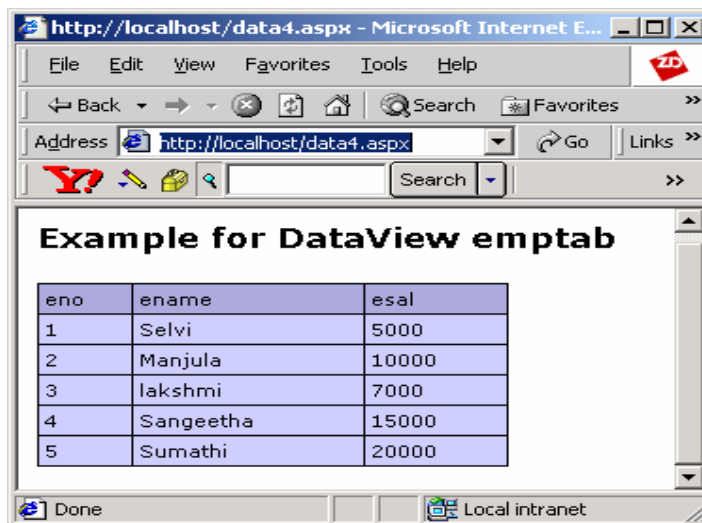
    <h3><font face="Verdana">Example for DataView <span runat="server"
        id="s" /></font></h3>

    <ASP:DataGrid id="d" runat="server"
        Width="250"
        BackColor="#ccccff"
        BorderColor="black"
        CellPadding=3
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd"
    />

</body>
</html>

```

A `SqlConnection` object `con` is created to perform a select query on the **emptab** table of the SQL database. A `SqlCommand` object `com` that contains the query statement is then constructed. A `DataSet` object is created and populated with the results from the query through the **FillDataSet** method. The `DataView dv` represents a subset of data from the `DataTable`. Finally, the output is displayed by binding the `DataView` with the `DataGrid d`.



11.5 Short Summary

- The DataSet is designed to handle the actual data from a data store and provides access to multiple tables, rows and columns
- The DataSet class resides in System.Data package
- The TableCollection object, RelationsCollection object and ExtendedProperties are the three major parts of the DataSet object model
- The DataRelation represents a parent/child relationship between two tables
- The DataTable contains a collection of Constraint objects that can be used to ensure the integrity of the data
- The DataView represents a databindable, customized view of a DataTable for sorting, filtering, searching, editing, and navigation. A major function of the DataView is to allow data binding on both Windows Forms and Web Forms

11.6 Brain Storm

1. What is a DataSet?
2. Name the two events of DataSet.
3. How can two DataTables be related?
4. Differentiate between DataTable and DataView.
5. What is the main use of DataView?

END

Lecture 12

Data Aware Control

Objectives

In this lecture you will learn the following

- + Learning about Databinder
- + Knowing about Repeater Control

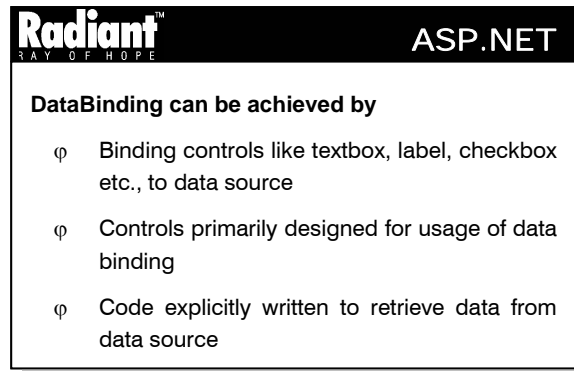
Coverage Plan

Lecture 12
12.1 Snap Shot
12.2 Databinder
12.3 Repeater Control
12.4 Short Summary
12.5 Brain Storm

12.1 Snap Shot

This session teaches the techniques that can be employed to bind ASP.NET controls to any data source. It also introduces the Repeater control that is a data-bound control.

In the previous sessions, we have learnt how to use ASP.NET controls. This session deals with binding the controls to the data source, that is, associating the control with the datasource so that the data for the control is supplied by the data source. In ASP.NET, the data source can be a database, an XML file, a control etc. The Web Forms page enables the user to bind any control's property to the information in any kind of data store.



DataBinding can be achieved in one of the following ways:

- Controls like textbox, label, checkbox etc., can be bound to the data source
- Controls like DataGrid, DataList etc., that are primarily designed for the usage of data binding can be used
- Programmer can write code explicitly to retrieve the data from the data source and display it in the controls

The Web Forms pages access data in the form of classes that expose data as properties and define methods for updating the underlying data source. The simple controls like radio button, checkbox etc., bind to a single value. They can bind to the public property of the page or any of its controls. Complex controls like Repeater, DataList and DataGrid can bind to any structure that implements the ICollection interface.

A control can be bound to the data by using either of the following steps:

- DataSource property of a complex list control is set to refer to a data class
- A property of the control is bound using a data binding expression which when evaluated provides a value that is loaded into the bound property

The databinding expression has the following syntax:

```
<% expression %>
```

where *expression* is the data binding expression which is a reference to any publicly-available property. The data binding expression is delimited by the <% and %> tags. Suppose the **Text** property of a Label has to be bound to the **deptName** field of the table **emp** then the data binding expression for it appears as follows:

```
<asp: Label id="dept" Text='<% emp(0).deptName %>' runat=server>
```

The expression is evaluated only when the **DataBind** method is called. This method is generally called at two instances: when the page is loaded or when an event-handling method has changed the data set. The DataBind method can be called for each control or for the whole page. When it is called for the whole page, it is cascaded down to each control on the page. Similarly, when it is called on a parent control, it is cascaded to all of its child controls.

12.2 Databinder

Radiant™
RAY OF HOPE

ASP.NET

Databinder

- ⌘ Makes code simpler and more readable
- ⌘ Eval()
 - Static method that evaluates data binding expressions against Container object at runtime
 - Optionally formats result as a string
 - Useful when data binding against controls in a templated list

Syntax = Eval(Object *container*, string *expression*, string *format*) n not be used

The DataBinder class is provided by ASP.NET to make code simpler and more readable. The static method Eval() of this class evaluates data binding expressions against the **Container** object at runtime and optionally formats the result as a string. This method is particularly useful when data binding against controls in a templated list. It can take one of the two forms:

```
Eval(object container, string expression)
Eval(object container, string expression, string format)
```

In the above forms, the first argument is the object reference against which the expression is evaluated. For the controls Repeater, DataList and DataGrid, the value of this parameter should be **container.dataitem**. If the binding is done against the page then the parameter should take the value **page**. The second argument is the navigation path from the *container* to the property value to be placed in the bound control property. It should be a string of property or field names separated by dots. The third argument is a format string used to convert the result of evaluating the expression into a string type to be displayed by the requesting browser.

Binding to Non-DataBase Datasources

Radiant™
RAY OF HOPE

ASP.NET

ASP.NET data binding syntax supports binding to

- ⌘ Public variables
- ⌘ Properties of the page
- ⌘ Properties of other controls on the page

The ASP.NET data binding syntax supports binding to public variables, properties of the page and properties of other controls on the page. The following example illustrates binding to public variables and simple properties on the page.



Practice 12.1

The following example binds four properties to display the details of an employee.

```
<html>
<head>
<script language="C#" runat="server">
void Page_Load(Object sender, EventArgs e) {
    Page.DataBind();
}

int EmpID {
    get {
        return 101;
    }
}

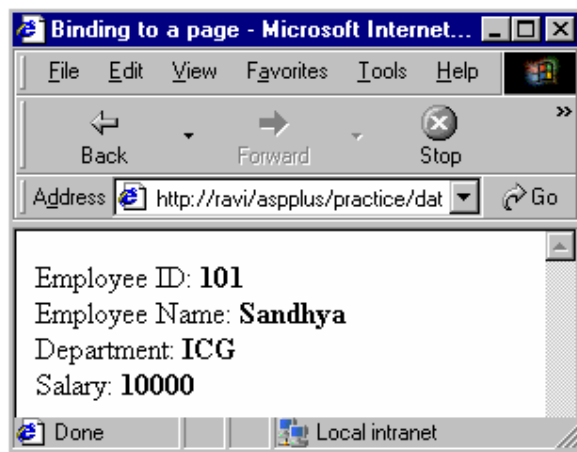
string EmpName{
    get {
        return "Sandhya";
    }
}

string Dept{
    get {
        return "ICG";
    }
}

int Salary{
    get {
        return 10000;
    }
}

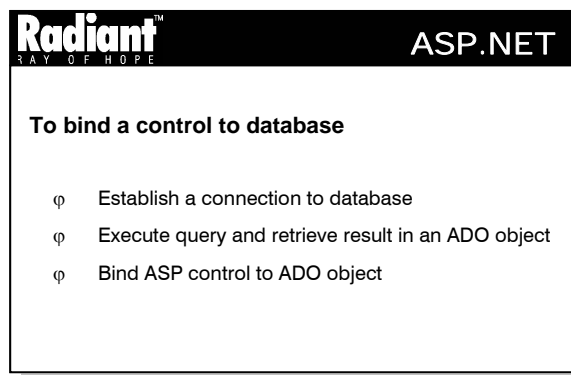
</script>
</head>
<body>
    <form runat=server>
        Employee ID: <b><%# EmpID %></b><br>
        Employee Name: <b><%# EmpName %></b><br>
        Department: <b><%# Dept %></b><br>
        Salary: <b><%# Salary %></b>
    </form>
</body>
</html>
```





The above example uses the databinding expression to bind four properties namely, EmpID, EmpName, Dept and Salary to the corresponding elements of the page. The **DataBind()** method is present in the Page_Load event so that the data is displayed as soon as the page is loaded.

Binding to a DataBase



The controls like ListBox, DropDownList, CheckBoxList and RadioButtonList can be used to display data from a database. They can display one field from the source and use another field as the item value. The **DataSource** property of the control should be set to the source of the data. The **DataTextField** property denotes the name of the field whose value is displayed in the list. The **DataValueField** property is the name of the field to be used for the **Value** property of each list item.



Practice 12.2

The following example uses the **ListBox** control to display the list of publisher names from the **Publishers** table.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<title> Binding a List to the database </title>

<script language="C#" runat="server">
void Page_Load ( Object src, EventArgs e)
{
```

```

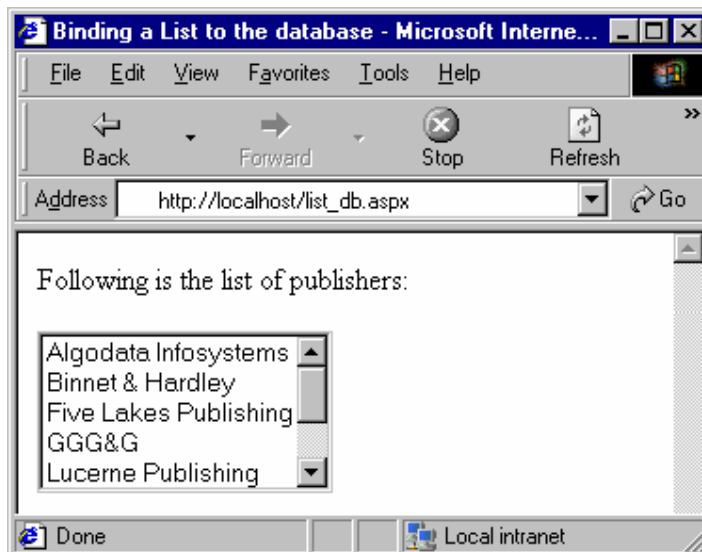
DataSet ds;
SqlConnection conn;
SQLDataSetCommand cmd;
    string connStr, query;

connStr="server=localhost;uid=sa;database=pubs";
    conn = new SqlConnection(connStr);

query="select distinct pub_name from Publishers";
cmd = new SQLDataSetCommand(query, conn);
ds = new DataSet();
cmd.FillDataSet(ds, "publishers");
Lb.DataSource= ds.Tables["publishers"].DefaultView;
Lb.DataTextField="pub_name";
Lb.DataBind();
}
</script>
<body>
<form runat=server>
<asp:Label id="lbl" text="Following is the list of publishers: "
    runat="server" />

<br><br>
    <asp:ListBox id="Lb" runat="server"/>
</form>
</body>
</html>

```



The above example binds the listbox to the database in the **Page_Load** event. It creates a connection object named **conn** using the connection string **connStr**. A command object **cmd** is then created using this connection and the query string **query**. The command is executed and the results are retrieved in the

DataSet **ds**. Then the DataSet is linked to the listbox through the properties **DataSource** and **DataTextField**.

12.3 Repeater Control

Radiant™
RAY OF HOPE

ASP.NET

Repeater Control

- ❖ Used to display data items in a repeating list
- ❖ Layout of content is defined using templates
- ❖ Bound to data source through its DataSource property
- ❖ Can bind to any class that supports ICollection or IEnumerable interface

The Repeater control is used to display the data items in a repeating list. The layout of the control's content is defined using templates.

A template is a set of HTML elements and controls that make up the layout for a particular portion of a control. Templates allow the programmer to specify both the contents and appearance of controls like the Repeater, DataGrid and DataList. Each of these controls support a slightly different set of templates that specify layouts for different portions of the control such as the header, footer, item and selected item. The type of template is specified as the NAME attribute of the template element. Templates can be created directly in the .aspx file containing the control which uses the template. Since the template also specifies the content to be displayed, the databinding expressions can be specified inside the template.

The Repeater control supports the following templates:

- **HeaderTemplate** – This defines the layout of the elements to be rendered once before all the data-bound rows are displayed
- **ItemTemplate** – This defines the layout of the elements to be rendered once for each row in the Repeater control. This template is used to bind one or more ASP.NET server controls to a data source. This template is mandatory
- **AlternatingItemTemplate** – This is similar to the ItemTemplate, but it is rendered for every alternate row in the Repeater control
- **SeparatorTemplate** – This defines the layout of the elements to be rendered between each row, such as line breaks, lines etc
- **FooterTemplate** – This defines the layout of the elements to be rendered once after all the data-bound rows are displayed

The Repeater control is bound to the data source through its **DataSource** property. It can bind to any class that supports the ICollection or IEnumerable interface.



Practice 12.3

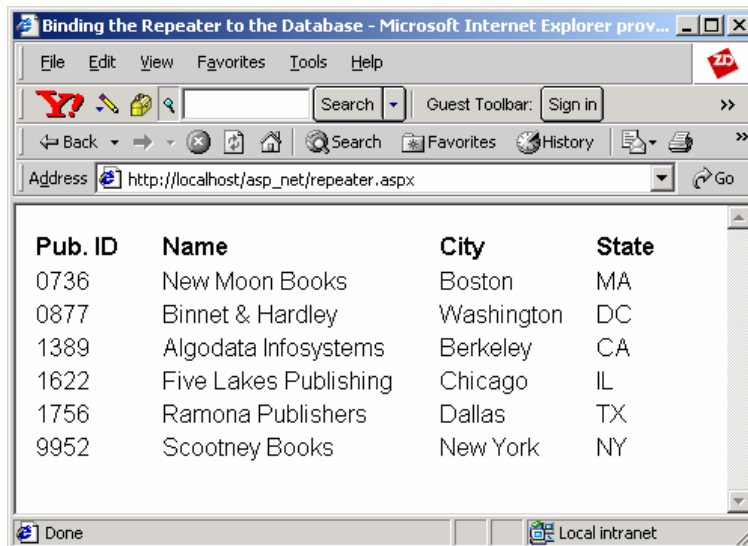
The following example uses the **Repeater** control to retrieve the rows whose country is **USA**, from the **Publishers** table.

```

<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<html><head>
<title>Binding the Repeater to the Database</title>
<script language="C#" runat="server">
void Page_Load ( Object src, EventArgs e)
{
    DataSet DS;
    SqlConnection conn;
    SQLDataSetCommand cmd;
    string connStr, query;
    connStr="server=localhost;uid=sa;database=pubs";
    conn = new SqlConnection(connStr);
    query = "select * from Publishers where country='USA'";
    cmd = new SQLDataSetCommand(query,conn);
    DS = new DataSet();
    cmd.FillDataSet(DS, "publishers");
    rep.DataSource=DS.Tables["Publishers"].DefaultView;
    rep.DataBind();
}
</script>
</head>
<body bgcolor="#FFFFFF">
    <ASP:Repeater id="rep" runat="server">
        <template name="headertemplate">
            <table >
                <tr align=left >
                    <th width=80> Pub. ID </th>
                    <th width=180> Name </th>
                    <th width=100> City </th>
                    <th> State </th>
                </tr>
            </template>
            <template name="itemtemplate">
                <tr>
                    <td><%# DataBinder.Eval(Container.DataItem, "pub_id")%> </td>
                    <td> <%# DataBinder.Eval(Container.DataItem, "pub_name")%></td>
                    <td> <%# DataBinder.Eval(Container.DataItem, "city")%> </td>
                    <td> <%#DataBinder.Eval(Container.DataItem, "state")%></td>
                </tr>
            </template>
            <template name="footertemplate">
                </table>
            </template>
        </ASP:Repeater>
    </body>
</html>

```





The screenshot shows a Microsoft Internet Explorer window titled "Binding the Repeater to the Database - Microsoft Internet Explorer prov...". The address bar displays "http://localhost/asp_net/repeater.aspx". The main content area shows a table with four columns: Pub. ID, Name, City, and State. The table contains six rows of data. The status bar at the bottom indicates "Done" and "Local intranet".

Pub. ID	Name	City	State
0736	New Moon Books	Boston	MA
0877	Binnet & Hardley	Washington	DC
1389	Algodata Infosystems	Berkeley	CA
1622	Five Lakes Publishing	Chicago	IL
1756	Ramona Publishers	Dallas	TX
9952	Scootney Books	New York	NY

The above example binds the Repeater control to the database in the **Page_Load** event. It creates a connection object named **conn** using the connection string **connStr**. A command object **cmd** is then created using this connection and the query string **query**. The command is executed and the results are retrieved in the DataSet **DS**. Then the DataSet is linked to the repeater through the properties **DataSource**. The **itemtemplate** of the repeater defines how the data of each row has to be displayed.

12.4 Short Summary

- The Web Forms page enables the user to bind any control's property to the information in any kind of data store
- The Web Forms pages access data in the form of classes that expose data as properties and define methods for updating the underlying data source
- The property of a control can be bound to a datasource using the databinding expression. The databinding expression is evaluated only when the DataBind method is called
- The ASP.NET data binding syntax supports binding to public variables, properties of the page and properties of other controls on the page
- List server controls like ListBox, DropDownList etc., use a collection as a datasource
- The controls like ListBox, DropDownList, CheckBoxList and RadioButtonList can be used to display a single field from a table
- The Repeater control is used to display data items in a repeating list

12.5 Brain Storm

1. What are the ways in which Databinding can be achieved?
2. What is the importance of the DataBind method?

3. List the steps involved in binding a control to the database.
4. List the templates supported by the Repeater control.
5. Differentiate between AlternatingItemTemplate and ItemTemplate.



Lecture 13

DataGrid and DataList Server Controls

Objectives

In this lecture you will learn the following

- + Knowing about DataGrid Server Control
- + What is meant by paging in DataGrid

Coverage Plan

Lecture 13
13.1 Snap Shot
13.2 DataGrid Server Control
13.3 DataList Server Control
13.4 Short Summary
13.5 Brain Storm

13.1 Snap Shot

This session is planned to throw light on DataGrid Server control and various types of columns that are allowed in it. Further, the focus is on DataList control and selecting items in it. It also aimed at teaching methods to access database using DataGrid and DataList Server controls.

DataGrid Server control displays information in tabular form with columns. It Provides mechanisms that allow editing and sorting. DataList Server control is similar to Repeater control, but with more formatting and layout options, including the ability to display information in a table. The DataList control also allows to specify editing behavior.

13.2 DataGrid Server Control

Radiant™
RAY OF HOPE

ASP.NET

- ☞ Allows to add specific functionality, including selecting, editing, sorting, and paging data
- ☞ Must be bound to a data source through its DataSource property
- ☞ The grid displays one row, one item for every row in the data source
- ☞ 0

By default, the DataGrid control generates a bound column for each field in the data

The DataGrid ASP.NET sever control is a multi-column, data-bound grid that displays tabular data. The control allows to define various types of columns, both to lay out the contents of the grid and to add specific functionality, including selecting, editing, sorting, and paging the data.

The DataGrid ASP.NET server control must be bound to a data source via its DataSource property or it will not be accomplished on the page.

When data binding, a data source is specified for the DataGrid control as a whole. The grid displays one row, one item for every row in the data source. By default, the DataGrid control generates a bound column for each field in the data source. When the page runs, the controls DataBind method is called to load the grid with data. The method can be called each time the page makes a round trip to the server (for example, in an event-handling method) to refresh the grid.

A simple datagrid



Practice 13.1

The following program shows a simple DataGrid with some values.

```
<%@ Import Namespace="System.Data" %>

<html>

<title> Simple DataGrid </title>

<script language="C#" runat="server">
```

```

ICollection CreateDataSource() {
    DataTable dtbl = new DataTable();
    DataRow drow;

    dtbl.Columns.Add(new DataColumn("Serial No", typeof(Int32)));
    dtbl.Columns.Add(new DataColumn("Product No", typeof(string)));

    for (int i = 1; i < 6; i++) {
        drow = dtbl.NewRow();

        drow[0] = i;
        drow[1] = "prd 00" + Int32.ToString(i);
        dtbl.Rows.Add(drow);
    }

    DataView dview = new DataView(dtbl);
    return dview;
}

void Page_Load(Object sender, EventArgs e) {
    MyDataGrid.DataSource = CreateDataSource();
    MyDataGrid.DataBind();
}

</script>

<body>

<h3><font face="Verdana">A Simple DataGrid </font></h3>

<form runat=server>

    <ASP:DataGrid id="MyDataGrid" runat="server"
        BorderWidthblh="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Courier"
        Font-Size="14pt"
        HeaderStyle-BackColor="#aaaadd"

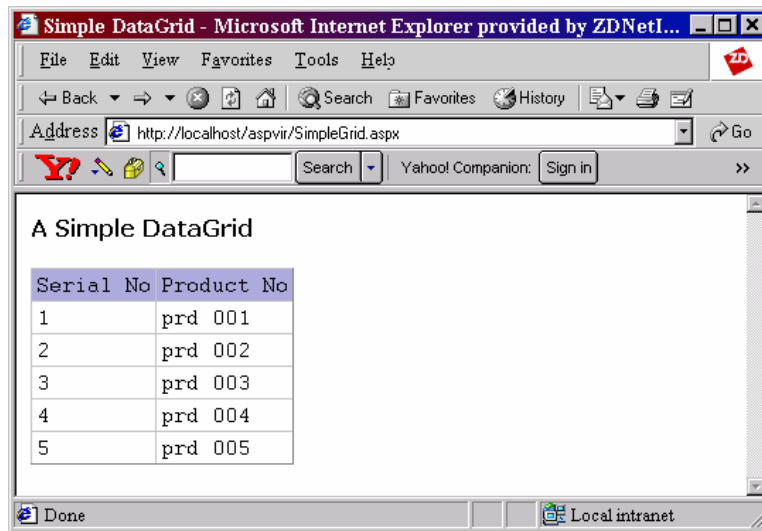
    />

</form>

</body>
</html>

```





The above example shows a simple DataGrid control which shows Serial No and Product No. These two Columns has been created using DataColumn object in Datatable. Similarly rows has been inserted using DataRow object in Datatable.

DataGrid Columns

The **DataGrid** control allows to specify the columns it displays in a variety of ways. By default, the columns are generated automatically based on the fields in the data source. However, in order to manipulate the content and layout of columns more precisely, the following types of columns can be defined:

- **Bound columns**

This columns allow to specify which database fields to be displayed and in what order it must be displayed and allow to specify the format and style of the display.

- **Hyperlink columns**

This columns display information as hyperlinks. A typical use is to display data (such as Area number or Area name) as a hyperlink that users can click to navigate to a separate page that provides details about that information.

- **Button columns**

These columns allow to display buttons next to each item in the grid and define custom functionality for the buttons. For example, a button might be created and labeled as "Add to Shopping Cart" that runs the custom logic when users click it.

- **Edit command columns**

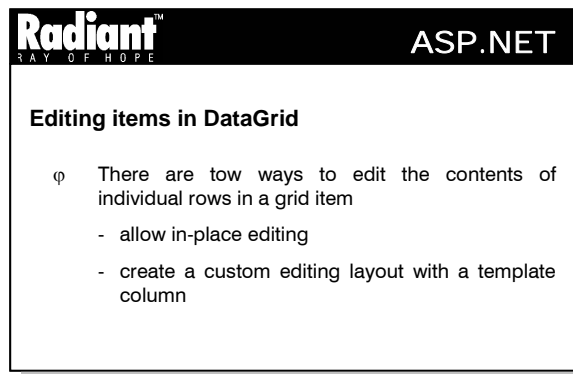
This column displays Edit, Update, and Cancel links in response to changes in the DataGrid's EditItemIndex property.

- **Template columns**

These columns allow to create combinations of HTML text and server controls to design a custom layout for a column. The controls within a template column can be data bound. Template columns provide the ultimate flexibility in defining the layout and functionality of the grid contents.

Note: To execute the following program, an MSAccess database named **sample.mdb** with a table Employee has to be created with the fields **EmployeeID, FirstName, Salary**.

Editing Items In DataGrid



There are two ways to edit the contents of individual rows in a grid item. They are:

- Allow in-place editing
- Create a custom editing layout with a template column

Allow in-place editing

This is the easiest method of allowing editing. A special column can be included in the grid with buttons labeled "Edit". When users click the edit button, the control automatically redisplay the current row with editable fields for all columns (for example, text boxes for characters and numbers and checkboxes for Boolean data). The column with the edit button is redisplayed with update and cancel buttons.

Create a custom editing layout with a template column

It is possible to create columns that display the data in editable controls. This approach allows to manipulate very precisely the columns that can be edited and allows to choose how the user can edit the data. When a template column is used, the items in the grid typically do not switch between edit and display modes. Instead, the items are always in edit mode. In either approach, a custom logic must be added to the page to update the data source with the user's edits.

Sorting Columns in DataGrid

The **DataGrid** ASP.NET server control provides a way to add sorting to the grid using the following methods:

- **Default sorting**

All columns in the grid are sortable. The header for each column contains a **LinkButton** control (hyperlink) that users click to sort by that column.

- **Custom sorting**

It is possible to define which column to support sorting and what type of button in the column heading user click to sort.

In either case, the grid does not sort the rows. Instead, it notifies the sort request by raising an event. Then sorting is performed in the code, usually by passing rebinding to the data source with new sort parameters.

Data in a grid is commonly sorted by clicking the header of the column. This is enabled by setting **AllowSorting=true**. When enabled, the grid will render LinkButtons in the header for each column. When the button is clicked, the grid's OnSortCommand event is thrown. Since DataGrid always displays the data in the same order it occurs in the datasource, the typical logic sorts the datasource, then rebinds the data to the grid.



Practice 13.2

The following example illustrates the sorting of columns in a DataGrid .

```
<% @ Import Namespace="System.Data" %>
<% @ Import Namespace="System.Data.ADO" %>
<html>
<head>
<script language="c#" runat="server" >
ADOConnection myconnection;
String conn;
void Page_Load(object source,EventArgs e)
{
    conn="provider=microsoft.jet.oledb.4.0;data source=C:/sample.mdb";
    myconnection=new ADOConnection(conn);

    if (!IsPostBack)
        BindGrid("EmployeeID");
}

void dbsort(object src,DataGridSortCommandEventArgs e)
{
    BindGrid(e.SortField);
}
void BindGrid(String sortfield)
{
    //For datasetcommand connection will be established implicitly
    ADODatasetCommand mycommand=new ADODatasetCommand("select
    EmployeeID,FirstName,Salary from Employees",conn);

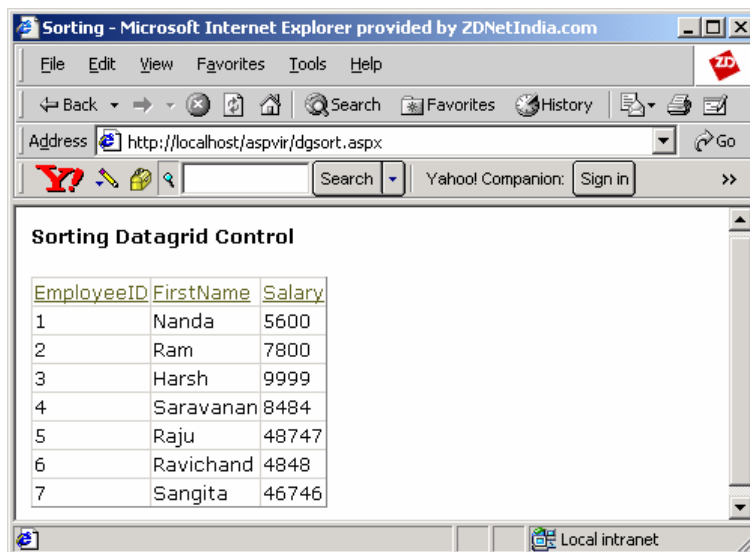
    DataSet ds=new DataSet();
    mycommand.FillDataSet(ds,"employees");

    DataView dv= ds.Tables["employees"].DefaultView;
    dv.Sort = sortfield;
```

```

        DataGrid1.DataSource = dv;
        DataGrid1.DataBind();
    }
</script>
</head>
<form runat="server">
    <h4> Sorting Datagrid Control </h4>
    <asp:DataGrid id="DataGrid1" onsortcommand="dbsort" allowsorting="true"
runat="server" autogeneratecolumns="true"/>
</form>
</body>
</html>

```



The above example shows how to do sorting in datagrid. When we click any one of the header column the rows will be sorted in ascending order. The sortfield has been used to set the field to be sorted. Set the DataView's sort property to the sortfield so that dataview will sort it out.

Paging in DataGrid

Radiant™
RAY OF HOPE

ASP.NET

- ☞ Provides the means to display a group of records from the datasource, then navigate to the "page" of next 10 records, and so on through the data.
- ☞ Enable in DataGrid by setting AllowPaging=true
- ☞ Grid displays page navigation buttons either "next/previous" buttons or as numeric buttons

The **DataGrid** ASP.NET server control provides a way to add paging to the grid using the following methods:

- **Using built-in paging controls**

The grid displays navigation buttons (either Next and Previous buttons or numeric page numbers). When the user clicks these, the grid updates the current page number and raises an event so that the data can be refreshed.

- **Providing custom navigation controls**

It is possible to provide own paging controls and manually set which page is to be displayed. This option allow to move any number of pages at a time, jump to a specific page, and so on.

Automatic versus Manual Paging

The page displayed by the grid is determined by its **CurrentPageIndex** property. The built-in controls set this property automatically; if custom navigation controls are provided it has to be set. After the **CurrentPageIndex** is set, the grid should be re-bound to the data source. The grid will recreate the entire data set and automatically move to the appropriate place in the data set. It then displays enough rows to make up one page of the grid.

In case of working with a large data set, recreating the entire data set each time users navigate to a new page can require too many overheads. In that case, we might prefer to get data in page-size “chunks”, that is, retrieving just a page worth of records at a time. To allow that, turn off the automatic paging feature of the grid so that it doesn’t assume that it is working with the entire data set. Then correct number of rows should be retrieved manually to fill the grid.

Accessing a Database

Radiant™
3 A Y O F H O P E

ASP.NET

- φ SQL Connection class gets or sets the name of the current database or the database to be used once a connection is open
- φ Accessing a Database using DataGrid is achieved with the following steps:
 - Establish a connection
 - Execute a query and retrieve the results in a DataView, DataReader or DataSet
 - Bind the DataView, DataReader or DataSet to the DataGrid through the DataSource property

SqlConnection class gets or sets the name of the current database or the database to be used once a connection is open.

Accessing a Database using DataGrid is achieved with the following steps:

- Establish a connection
- Execute a query and retrieve the results in a DataView, DataReader or DataSet
- Bind the DataView, DataReader or DataSet to the DataGrid through the DataSource property



The following example shows how a DataGrid is used to access the data in the SQL Server database.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>
<script language="C#" runat="server">
void Page_Load ( Object src, EventArgs e)
{
    DataSet DS;
    SqlConnection MyConnection;
    SQLDataSetCommand MyCommand;
    string strconn, strSQL;

    strconn="server=saravanan;uid=sa;database=pubs";
    MyConnection = new SqlConnection(strconn);
    strSQL="select * from Publishers";
    MyCommand = new SQLDataSetCommand(strSQL,MyConnection);
    DS = new DataSet();
    MyCommand.FillDataSet(DS, "publishers");
    MyDataGrid.DataSource = DS.Tables["Publishers"].DefaultView;
    MyDataGrid.DataBind();
}

</script>

<body>
<form runat=server>
    <ASP:DataGrid id="MyDataGrid" runat="server"
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd" />

    </form>
</body>
</html>
```



The screenshot shows a web browser window titled "Publishers Table - Microsoft Internet Explorer provided by ZDNetIndi...". The address bar shows "http://localhost/aspvnr/dbase.aspx". The main content area displays a table with the following data:

pub_id	pub_name	city	state	country
0736	New Moon Books	Boston	MA	USA
0877	Binnet & Hardley	Washington	DC	USA
1389	Algodata Infosystems	Berkeley	CA	USA
1622	Five Lakes Publishing	Chicago	IL	USA
1756	Ramona Publishers	Dallas	TX	USA
9901	GGG&G	München		Germany
9952	Scootney Books	New York	NY	USA
9999	Lucerne Publishing	Paris		France

In the above example, data from the Publishers table are displayed in the DataGrid. When the "Show Values" button is clicked, the records are displayed as shown.

13.3 DataList Server Control

Radiant™
3 A Y O F H O P E

ASP.NET

- ⌘ Displays database information in a format that can be controlled very precisely using templates and styles
- ⌘ Useful for displaying rows of database information displays rows of data as *items* in the list
- ⌘ It is possible to use templates to define the layout of the items by including HTML text and controls.

The DataList control displays database information in a format that can be controlled very precisely using templates and styles. The DataList control is useful for displaying rows of database information. The DataList control displays rows of data as *items* in the list. It is possible to use templates to define the layout of the items by including HTML text and controls.

Templates for Control Layout

Templates define the HTML elements and controls that should be displayed for an item, as well as the layout of these elements. Style formatting such as font, color and border attributes — is set using the **styles** property. Each template has its own style property. For example, styles for the EditItemTemplate are set via the **EditItemStyle** property.

The following templates are supported by DataList control.

- **ItemTemplate**

This template defines the content and layout of items within the list.

- **AlternatingItemTemplate**
If this template is defined, this determines the content and layout of alternating items. If it is not defined, ItemTemplate is used.
- **SelectedItemTemplate**
If this template is defined, this determines the content and layout of the selected item. If it is not defined, ItemTemplate is used.
- **EditItemTemplate**
This defines the layout of an item when it is in edit mode. This template typically contains editing controls, such as **TextBox** controls.
- **HeaderTemplate and FooterTemplate**
This text defines the text and controls to render at the beginning and end of the list.
- **SeparatorTemplate**
This defines the elements to render between each item. A typical example might be a line (using an **<HR>** element).

A Simple DataList

DataList supports directional rendering through the **RepeatDirection** property. This means that it can render its items horizontally or vertically. Since page width is the dimension that the developer must control in Web UI, DataList permits the developer to control the number of “columns” that are rendered (**RepeatColumns**), regardless of whether the items are rendered horizontally or vertically.



Practice 13.4

This following example displays a simple DataList control.

```
<%@ Import Namespace="System.Data" %>
<html>
<script language = "C#" runat="server">
ICollection CreateDataSource() {
    DataTable dt = new DataTable();
    DataRow dr;
    dt.Columns.Add(new DataColumn("StringValue", typeof(string)));
    for (int i = 0; i < 5; i++) {
        dr = dt.NewRow();
        dr[0] = "Item " + Int32.ToString(i);
        dt.Rows.Add(dr);
    }
    DataView dv = new DataView(dt);
    return dv;
}
```

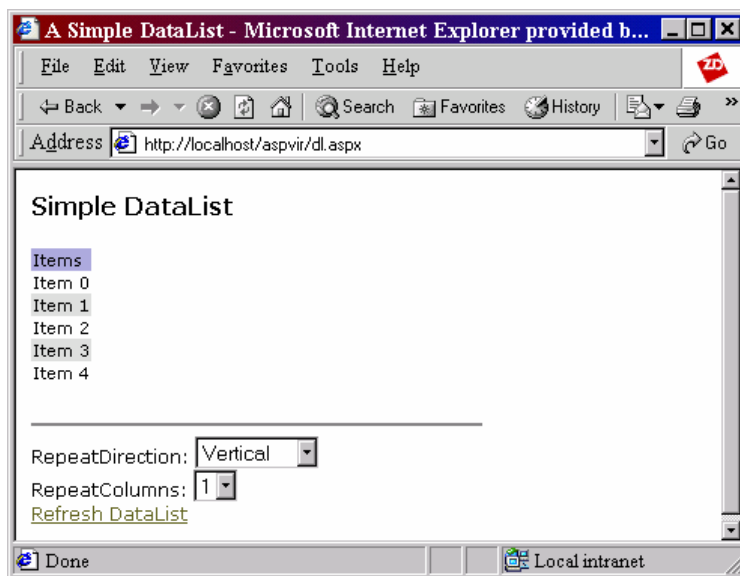
```
void Page_Load(Object Sender, EventArgs E) {
    if (!IsPostBack) {
        DataList1.DataSource = CreateDataSource();
        DataList1.DataBind();
    }
}

void Button1_Click(Object Sender, EventArgs E) {
    if (DropDown1.SelectedIndex == 0)
        DataList1.RepeatDirection = RepeatDirection.Horizontal;
    else
        DataList1.RepeatDirection = RepeatDirection.Vertical;
}
</script>
<body>
    <h3><font face="Verdana">Simple DataList Sample</font></h3>
    <form runat=server>
        <font face="Verdana" size="-1">
            <asp:DataList id="DataList1" runat="server"
                BorderColor="black"
                Font-Name="Verdana"
                Font-Size="8pt"
                HeaderStyle-BackColor="#aaaadd"
                AlternatingItemStyle-BackColor="Gainsboro"
            >
                <template name="HeaderTemplate">
                    Items
                </template>
                <template name="ItemTemplate">
                    <%# DataBinder.Eval(Container.DataItem, "StringValue") %>
                </template>
            </asp:DataList>
        </font>
        <p>
            <hr noshade align="left" width="300px">
                RepeatDirection:
                <asp:DropDownList id=DropDown1 runat="server">
                    <asp:ListItem>Horizontal</asp:ListItem>
                    <asp:ListItem>Vertical</asp:ListItem>
                </asp:DropDownList><br>
                RepeatColumns:
                <asp:DropDownList id=DropDown3 runat="server">
                    <asp:ListItem>1</asp:ListItem>
                    <asp:ListItem>2</asp:ListItem>
                    <asp:ListItem>3</asp:ListItem>
                    <asp:ListItem>4</asp:ListItem>
                </asp:DropDownList>
            </p>
        </form>
    </body>
```

```

        </asp:DropDownList><br>
        <asp:LinkButton id=Button1 Text="Refresh DataList"
        OnClick="Button1_Click" runat="server" />
    </font>
</form>
</body>
</html>

```



Above example shows a simple DataList control. As it can be seen from the above program a table is created using DataTable class and the rows are added using NewRow method. RepeatDirection property is used to render the items horizontally or vertically. With the help of RepeatColumns property, the number of columns to be repeated can be shown.

Selecting Items in DataList

It is possible to customize the content and appearance of the selected item via the **SelectedItemTemplate** property. The SelectedItemTemplate is controlled by the **SelectedIndex** property. By default the value of SelectedIndex is -1, meaning none of the items in the list is selected. When SelectedIndex is set to a particular item, that item is displayed using the SelectedItemTemplate.

Accessing a Database

SqlConnection class represents an open connection to a SQL Server data source. SQLDataSetCommand represents a set of data commands and a database connection which are used to fill the DataSet and

update the data source. DataBinder Class provides design-time support for Rapid Application Designers (RAD) designers to generate and parse DataBinding expression syntax.

Accessing a Database using DataList is achieved with the following steps:

- Establish a connection
- Execute a query and retrieve the results in a DataView, DataReader or DataSet
- Bind the DataView, DataReader or DataSet to the DataList through the DataSource property



Practice 13.5

The following example shows how a DataGrid is used to access the data in the database.

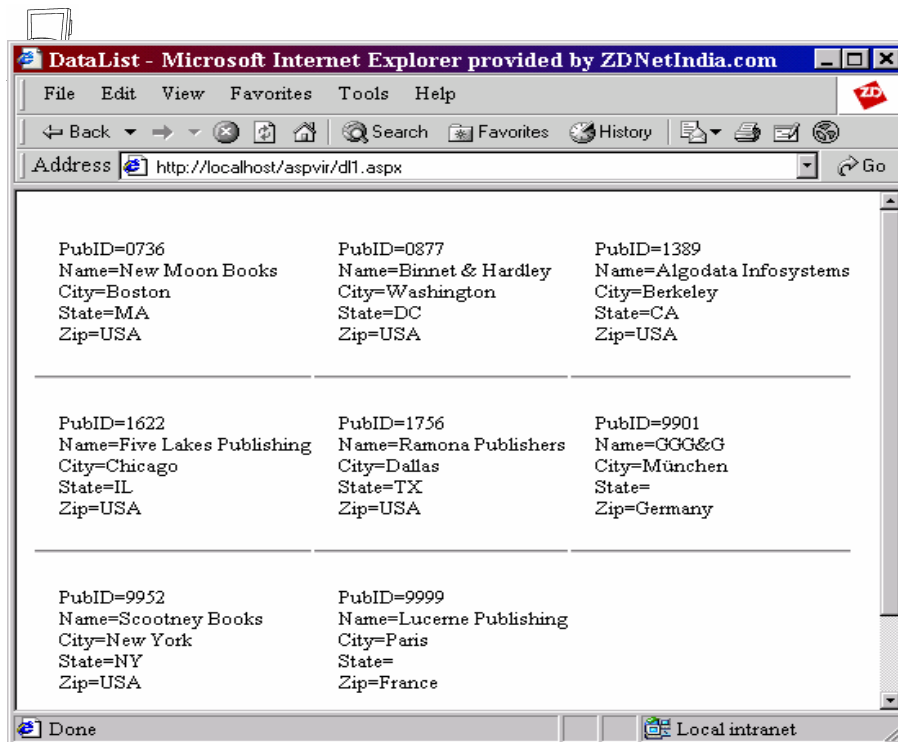
```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<script language="C#" runat="server">
void Page_Load ( Object src, EventArgs e)
{
    DataSet DS;
    SqlConnection Mycon;
    SQLDataSetCommand Mycomm;
        string strconn, strSQL;
    strconn="server=myserver;uid=sa;database=pubs";
        Mycon = new SqlConnection(strconn);
    strSQL="select * from Publishers";
    Mycomm = new SQLDataSetCommand(strSQL,Mycon);
    DS = new DataSet();
    Mycomm.FillDataSet(DS, "publishers");
    datlis.DataSource=DS.Tables["Publishers"].DefaultView;
    datlis.DataBind();
}
</script>
<html><head>
    <title>DataList</title>
</head>
<body bgcolor="#FFFFFF">
    <ASP:DataList id="datlis" repeatcolumns="3" repeatdirection="horizontal"
        repeatlayout="table" runat="server">
    <template name="headertemplate">
        </template>
    <template name="itemtemplate">
        PubID=<%# DataBinder.Eval(Container.DataItem, "pub_id")%><br>
        Name=<%# DataBinder.Eval(Container.DataItem, "pub_name")%><br>
        City=<%# DataBinder.Eval(Container.DataItem, "city")%><br>
        State=<%# DataBinder.Eval(Container.DataItem, "state")%><br>
```

```

        Zip=<%# DataBinder.Eval(Container.DataItem, "country")%><br>
    <hr>
</template>

<template name="footertemplate">
</template>
</ASP:DataList>
</body>
</html>

```



As seen from the above example, a connection to the database is created using a Connection object. SQL query is passed to the SQLDataSetCommand object mycom. The FillDataSet method executes the command and populates the DataSet DS with the result. The DataSource property of the DataList is used to bind the DataList to the DataSet. Then the DataBind method is used to list out the values in the DataList.

13.4 Short Summary

- The DataGrid ASP.NET server control is a multi-column, data-bound grid that displays tabular data
- The DataGrid ASP.NET server control must be bound to a data source via its DataSource property
- The DataGrid control allows to specify the columns it displays in a variety of ways
- The types of columns that can be defined in DataGrid are Bound columns, Hyperlink columns, Button columns, Edit command columns and Template columns

- The two ways to edit the contents of individual rows in a grid item are Allow in-place editing and Create a custom editing layout with a template column
- Data in a grid is commonly sorted by clicking the header of the column and is enabled by setting AllowSorting=true
- The DataGrid provides the means to display a group of records from the datasource
- SqlConnection class gets or sets the name of the current database or the database to be used once a connection is open
- The DataList control displays database information in a format that can be controlled very precisely using templates and styles
- DataList supports directional rendering through the RepeatDirection property
- It is possible to customize the content and appearance of the selected item through the SelectedItemTemplate property
- SqlConnection class represents an open connection to a SQL Server data source

13.5 Brain Storm

1. What is a DataGrid Control?
2. Explain various possible columns in a DataGrid.
3. What are the two ways to edit the contents of individual rows in a grid item?
4. What is a DataList control?
5. What are the Templates supported by DataList control?

