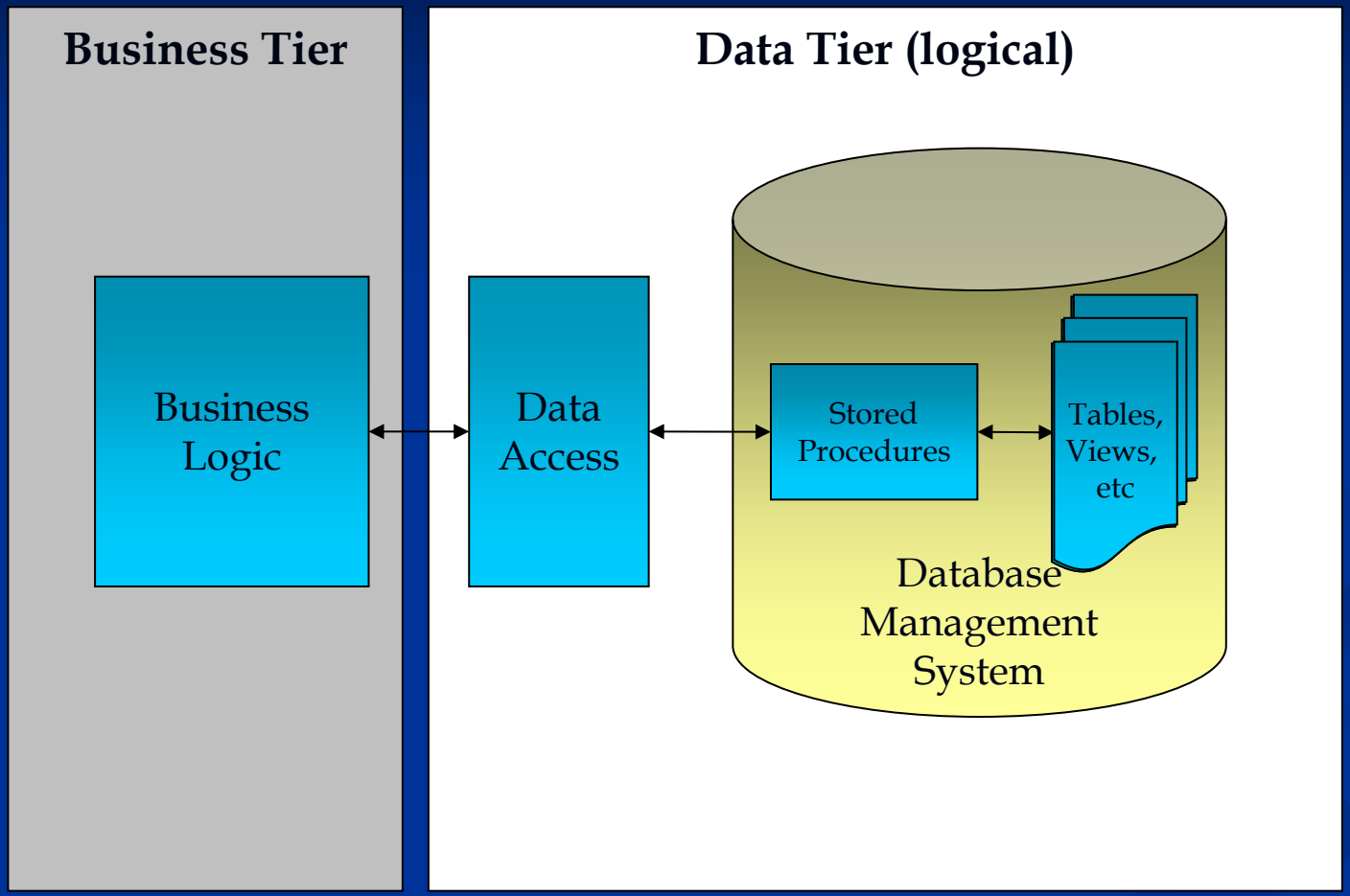
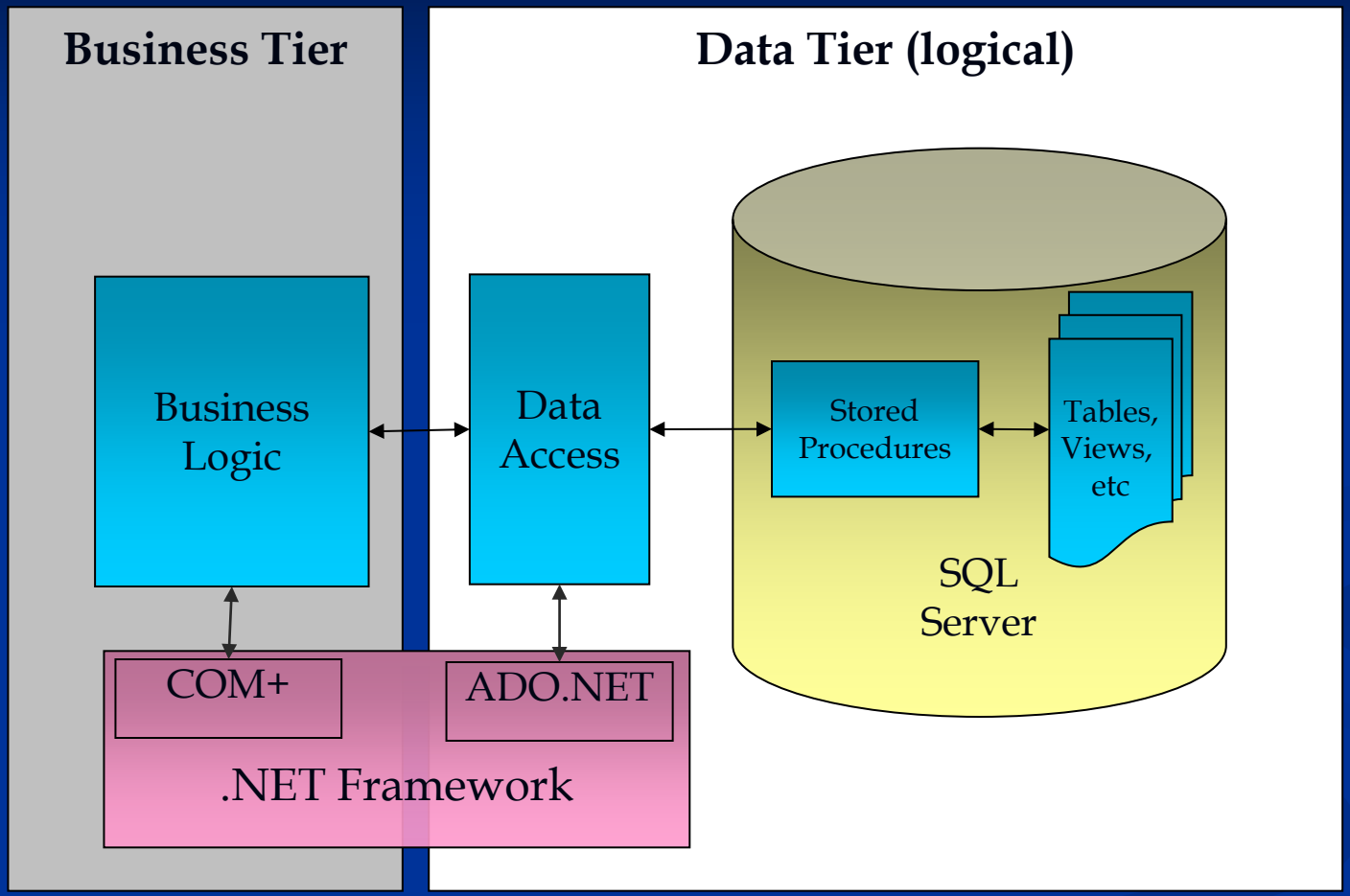
The Data Tier

ADO.NET and SQL Server

The Data Tier



The Data Tier with .NET (& SQL Server)

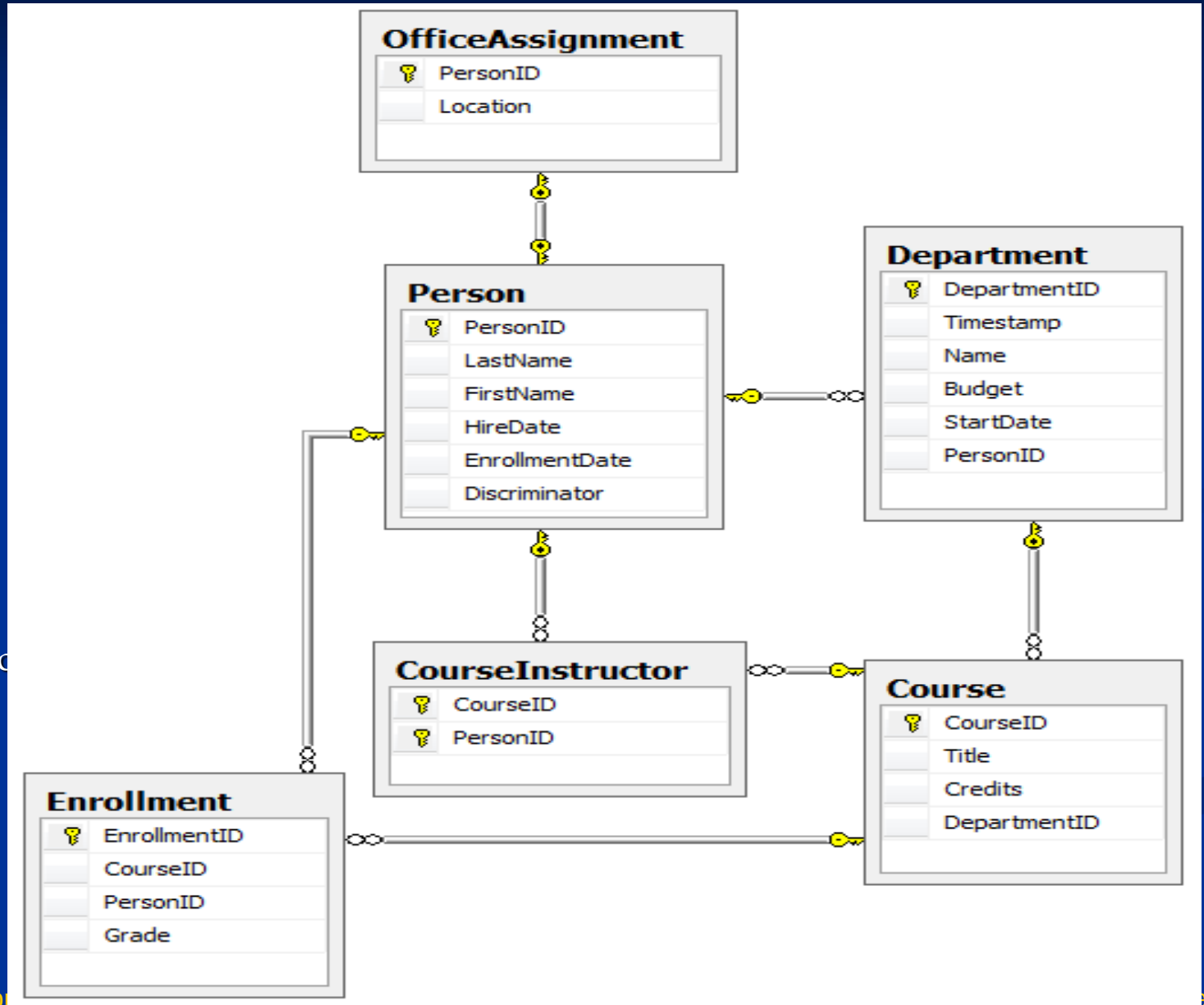


Worldwide RDBMS Software Revenue by Vendor, 2004–2006 ($M)

	2004	2005	2006	2005 Share (%)	2005–2006 Growth (%)	2006 Share (%)
Oracle	6,003	6,376	7,312	44.3	14.7	44.4
IBM	2,923	3,113	3,483	21.6	11.9	21.2
Microsoft	2,013	2,442	3,052	17.0	25.0	18.6
Sybase	471	503	524	3.5	4.3	3.2
NCR Teradata	390	423	457	2.9	8.0	2.8
Other	1,495	1,542	1,624	10.7	5.3	9.9
Total	13,296	14,398	16,451	100.0	14.3	100.0

Elements of a Database

- Databases
  - Tables
    - Columns
    - Primary Key
    - Indexes
  - Constraints
    - Primary and Foreign Keys,
    - Not null,
    - unique
    - One-to-one,
    - One-to-many etc



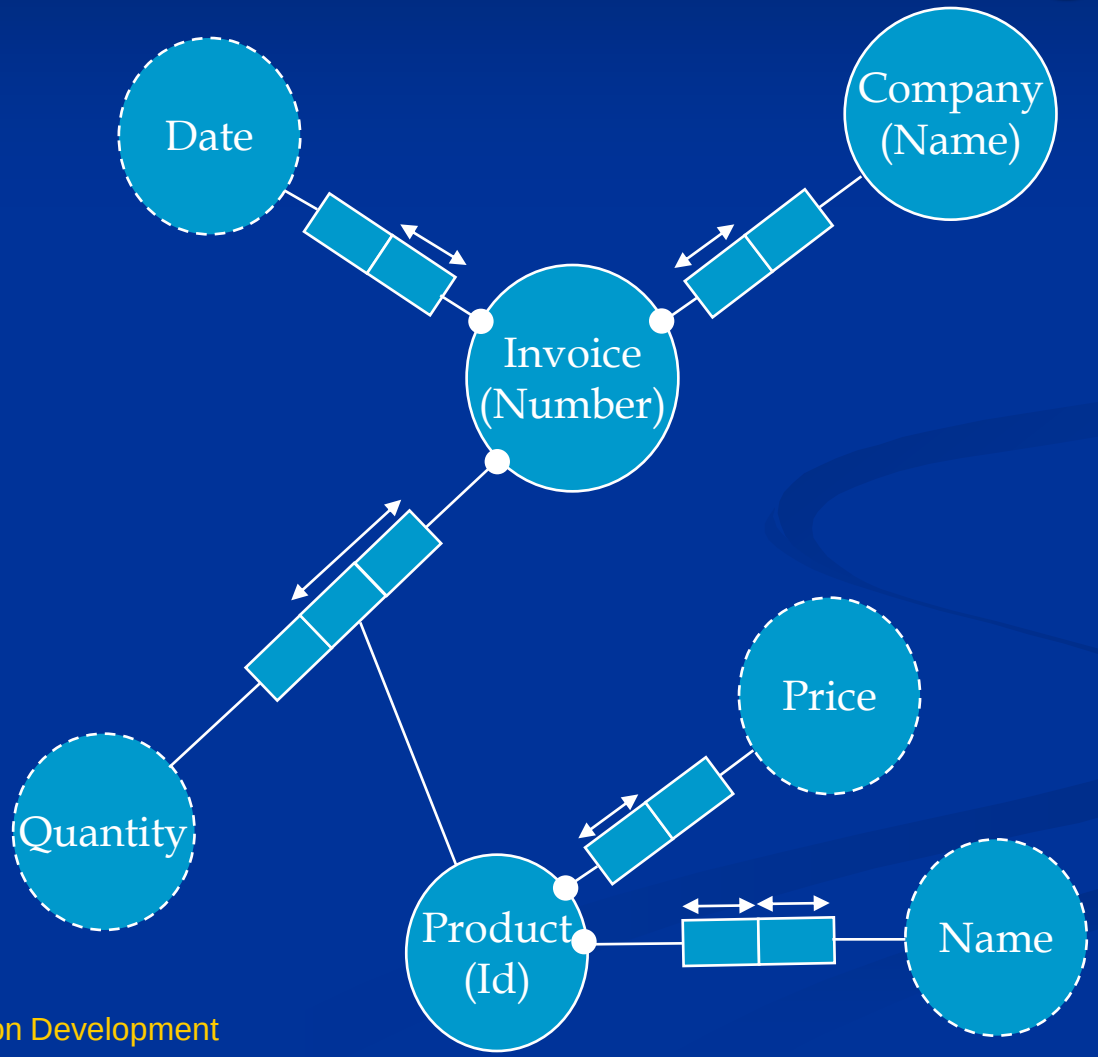
Elements of a Database

- Databases
  - Tables
    - Columns
    - Primary Key
    - Indexes
  - Constraints
    - Primary and Foreign Keys, Not null, etc
  - Views
    - Can be thought of as either a virtual table or a stored query.
  - Stored Procedures
    - A group of SQL statements compiled into a single execution plan.
  - Triggers
    - Stored procedures defined to execute automatically when an event occur.
  - Permissions
    - who can do what?

Database Schema Design

- How should the data be mapped to a set of relational tables?
- Goals:
  - Minimize redundancy of data without losing information.
  - Structure data in such a way as to avoid update anomalies.
- Requirements:
  - All column values are indivisible.
  - No column can be functionally dependent on any combination of columns, other than a super key.

Conceptual Schema Design (Entity-Relationship Modelling or Object-Role Modeling)



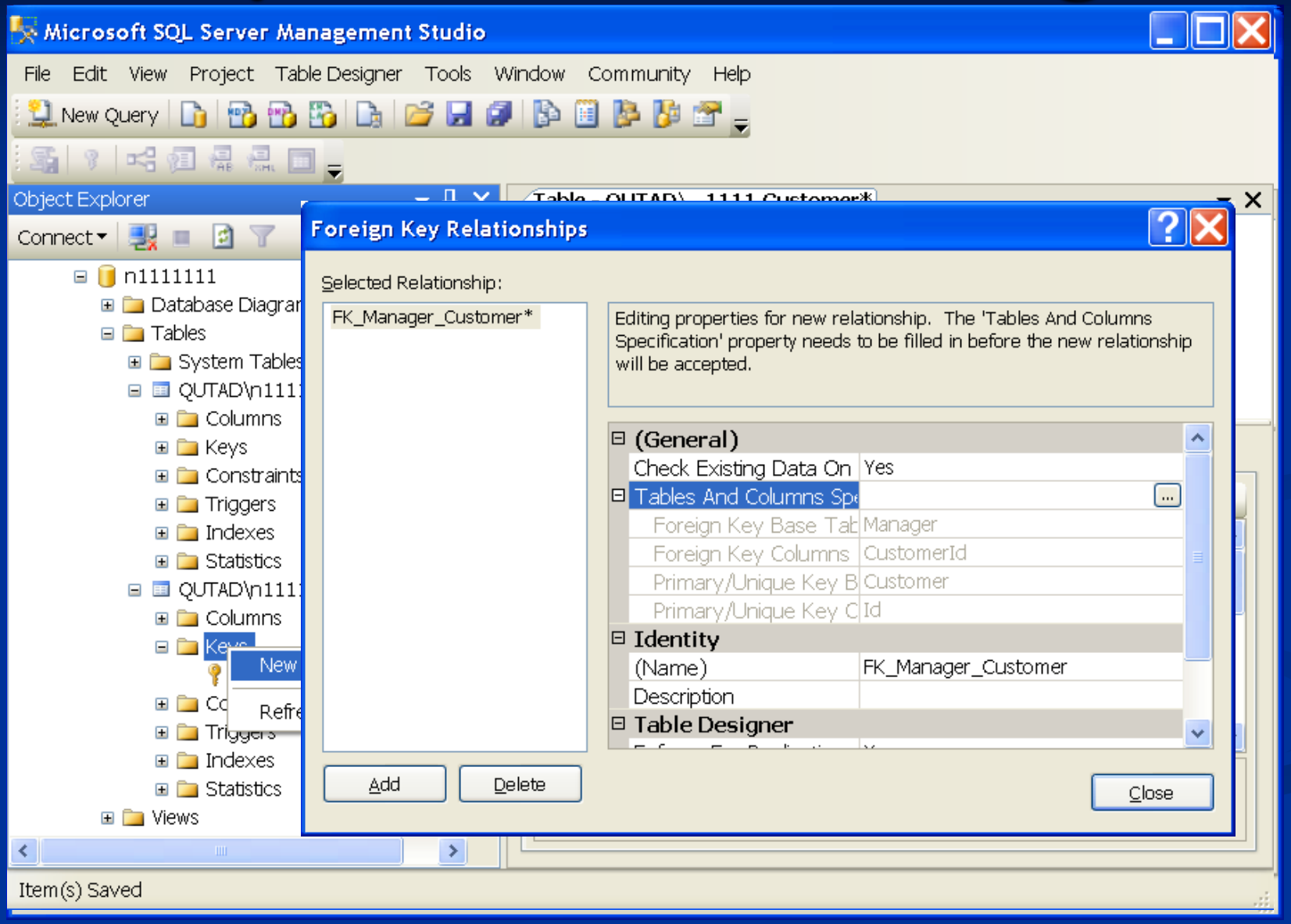
Relational Table Design

Invoice Table

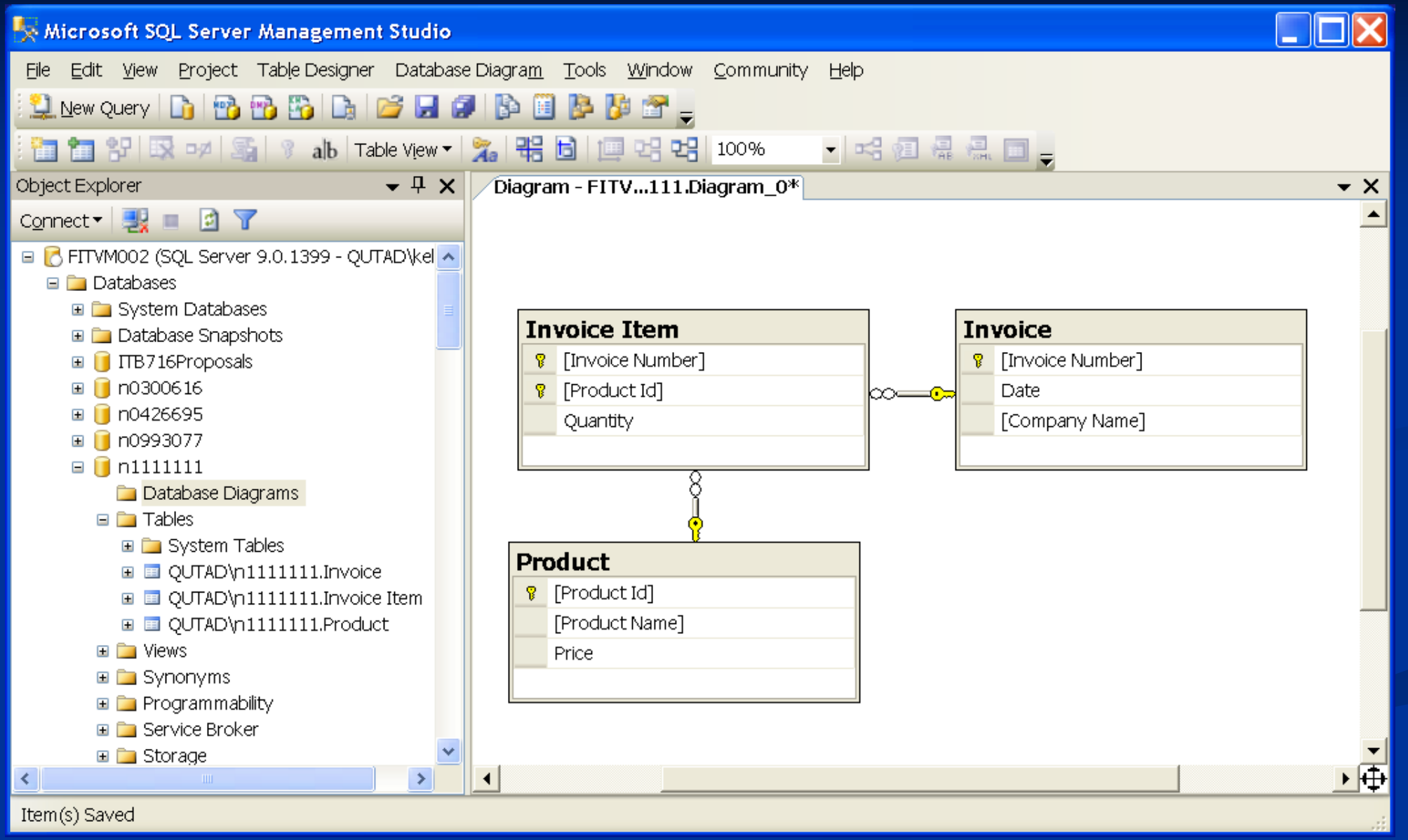
Invoice Item Table

Product Table

Physical Database Design



Creating Database Diagrams



Structured Query Language (SQL)

```
SELECT *
FROM Invoice
WHERE [Company Name] = 'QUT'
ORDER BY [Date]

SELECT [Invoice Number], [Product Name], Quantity * Price
FROM [Invoice Item], Product
WHERE [Invoice Item].[Product Id] = Product.[Product Id] AND
      [Invoice Number] = 1

SELECT [Invoice Number], SUM(Quantity)
FROM [Invoice Item]
GROUP BY [Invoice Number]
```

	Invoice Number	Date	Company Name
1	1	2004-08-18 ...	QUT
2	4	2004-08-18 ...	QUT

	Invoice Number	Product Name	(No co...
1	1	Paper	1931.3000
2	1	Staples	1029.0000

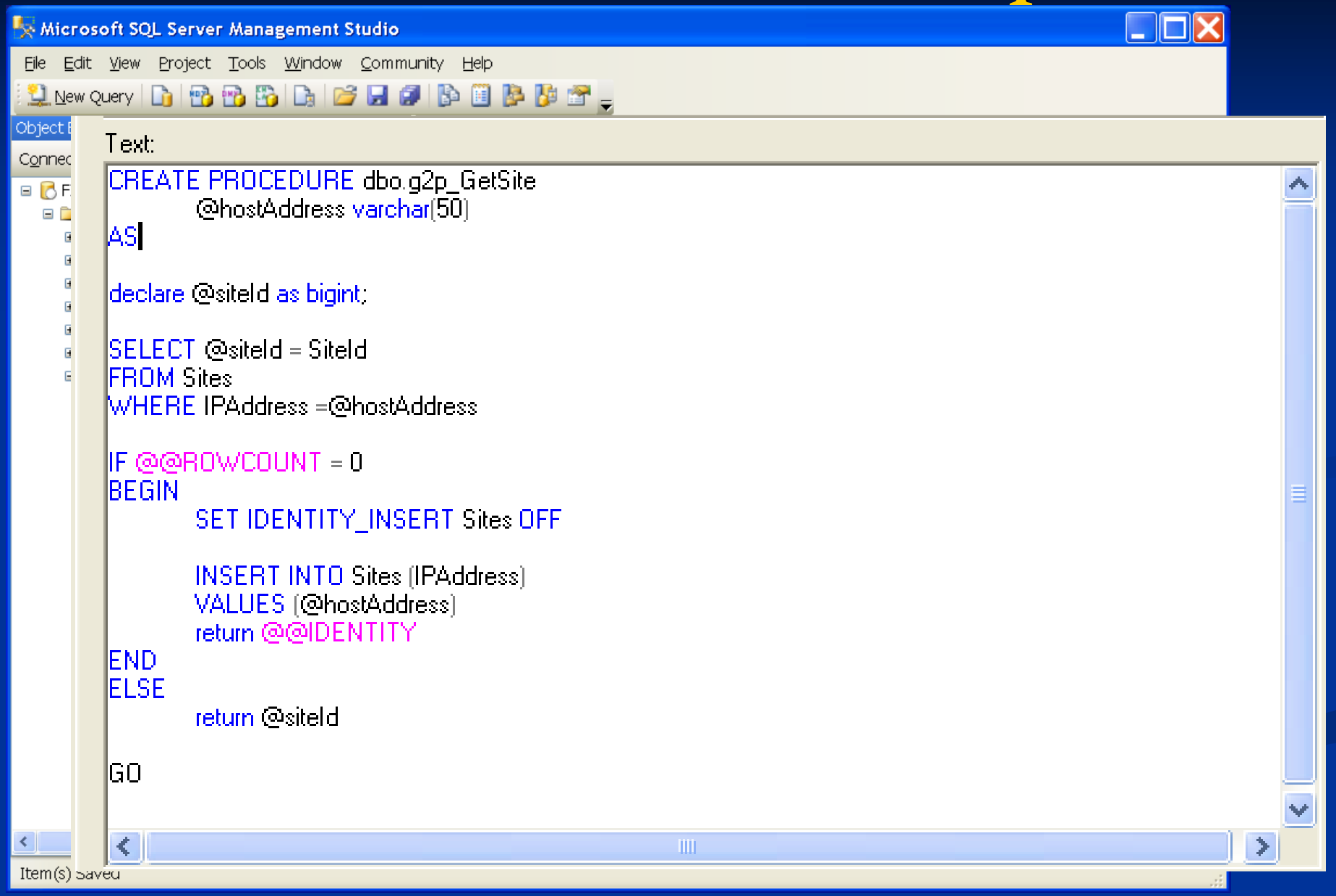
	Invoice Number	(No column name)
1	1	777
2	2	3809
3	3	247

SQL Commands

```
INSERT INTO Invoice ([Date], [Company Name])
values ('18/08/2004', 'Energex')

UPDATE [Invoice Item]
SET Quantity = 0
WHERE Quantity < 0

DELETE FROM Product
WHERE Price > 99.99
```

Ref: http://www.w3schools.com/sql/

Stored Procedures Example



Why Stored Procedures?

- Precompiled and optimized for efficiency
- Compile time syntax and semantic checking
- Save granting permissions on underlying tables.
- Easier to maintain than hard-coded queries in deployed code.
- Enforcement of database invariants
- Abstracting physical database design
- Reuse in different applications
- Executed close to the database

More Stored Procedure Examples

Text:

```
CREATE PROCEDURE dbo.g2p_FetchJob
        @volunteerId bigint,
        @classId bigint,
        @jobId bigint output,
        @method varchar(50) output,
        @clientId bigint output,
        @clientName varchar(50) output
AS

UPDATE Jobs
SET   @jobId = JobId, @clientId = ClientId, @method = MethodName,
    Status = 1, volunteerId = @volunteerId
WHERE JobId = (SELECT top 1 JobId FROM ReadyJobs WHERE classId = @classId)

SELECT UserName + '@' + HostName
FROM Sites
WHERE SiteId = (SELECT SiteId FROM Clients WHERE ClientId = @clientId)

GO
```

More Stored Procedure Examples

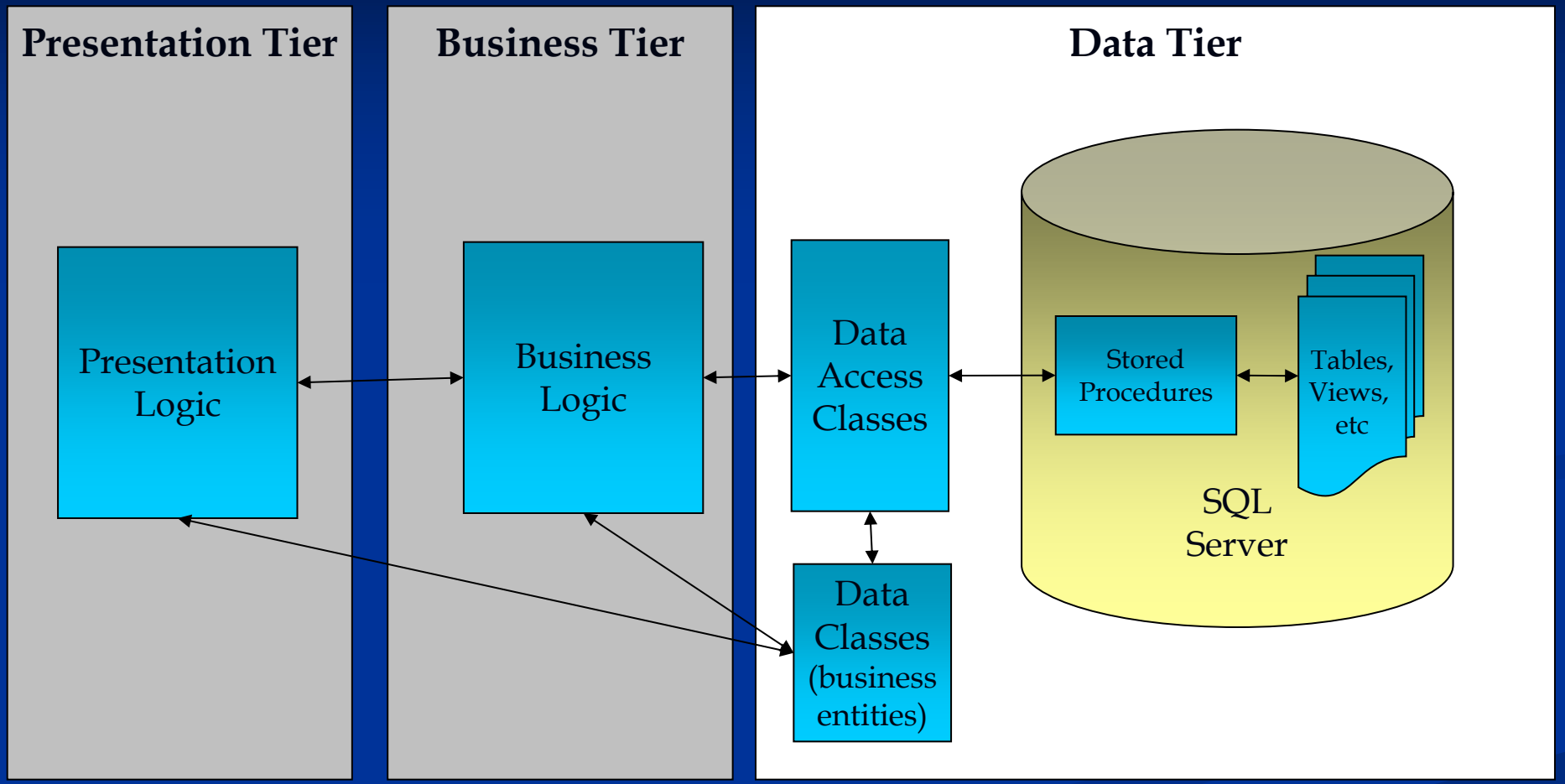
Text:

```
CREATE PROCEDURE dbo.g2p_VolunteerPulse
        @volunteerId bigint
AS

UPDATE Volunteers
SET TimeStamp = GETDATE()
WHERE VolunteerId = @volunteerId

GO
```

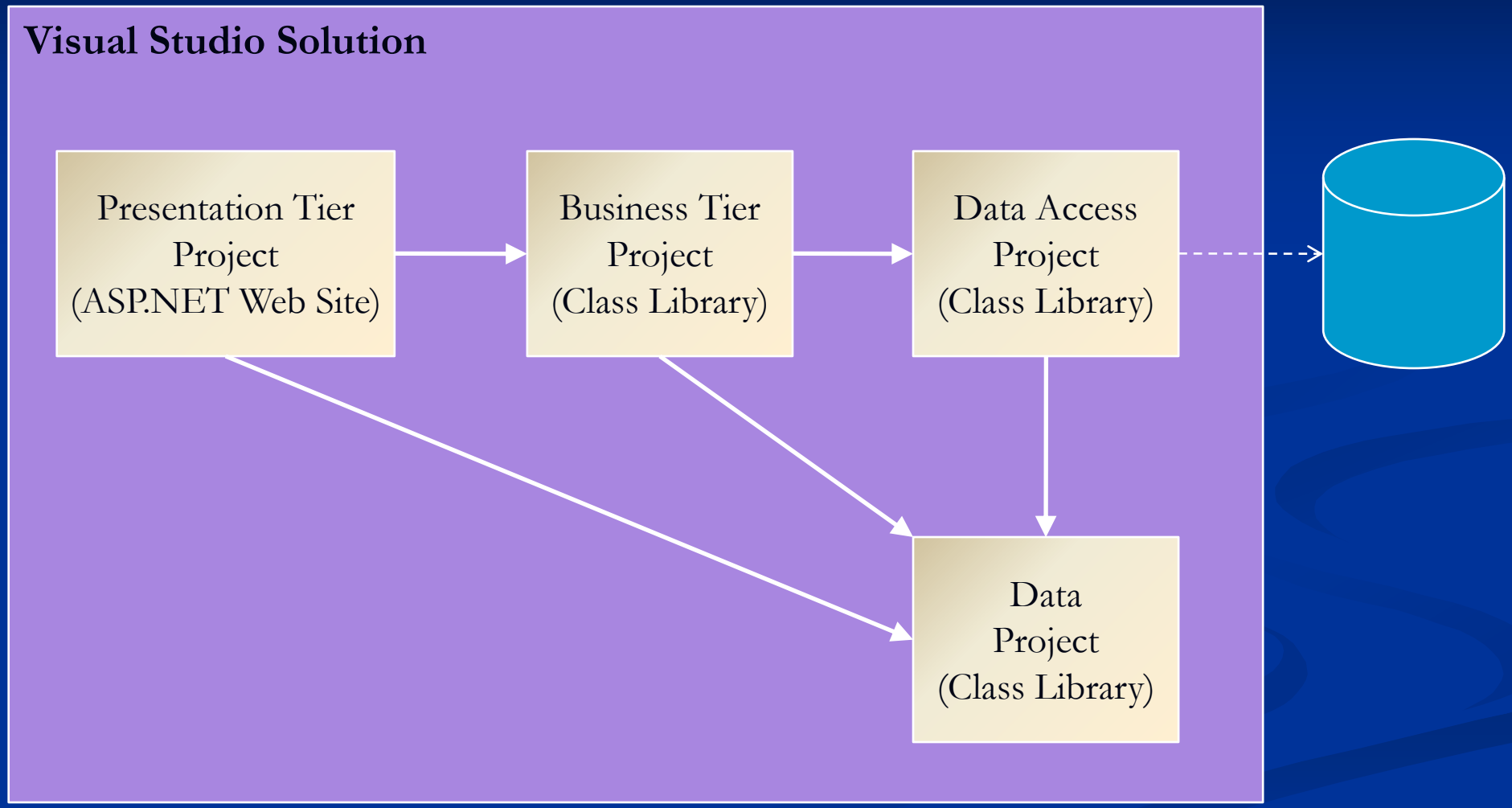
Middle Tier | Data Tier

- **Business Logic Classes**
  - Stateless methods that manipulate Data classes
- **Data Access Classes**
  - Abstracts the process of accessing databases
  - Take and return Data classes.
- **Data Classes** *(Business Entity Classes)*
  - Dumb containers
  - Just store information about a business entity
    - properties for each field
  - Don't know anything about data source
  - Don't contain any business logic methods

Data vs Data Access



Visual Studio Project References

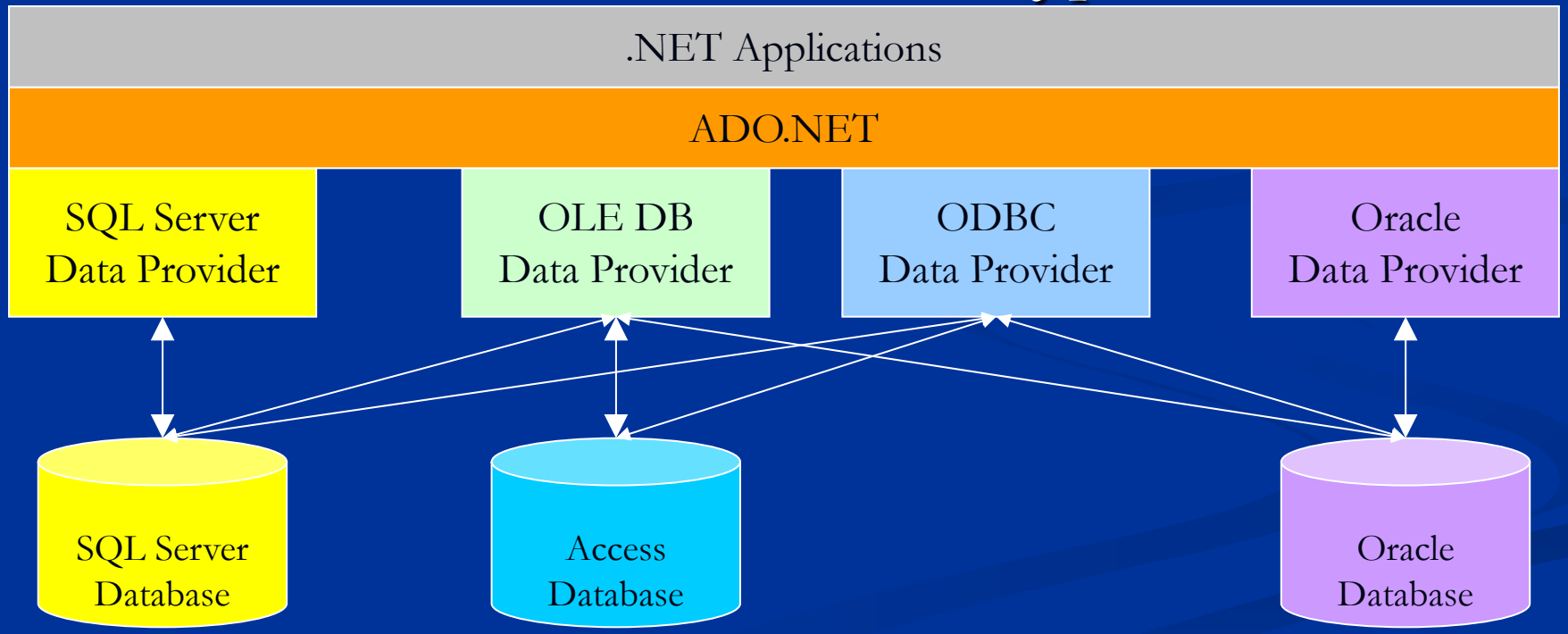


ADO.NET

Database API for .NET

ADO.NET Data Providers

- Different databases expose their own native APIs
- ADO.NET supports a number of different "Data Providers" for use with different types of databases



ADO.NET Data Providers

- SQL Server (`System.Data.SqlClient`)
  - `SqlConnection`
  - `SqlCommand`
  - `SqlDataReader`
- Oracle (`System.Data.OracleClient`)
  - `OracleConnection`
  - `OracleCommand`
  - `OracleDataReader`
- OLE DB (`System.Data.OleDb`)
  - `OleDbConnection`
  - `OleDbCommand`
  - `OleDbDataReader`
- ODBC (`System.Data.Odbc`)
  - `OdbcConnection`
  - `OdbcCommand`
  - `OdbcDataReader`

SQL Connections and Commands

```
SqlConnection cnn = new SqlConnection("server=local;uid=sa;database=pubs");

cnn.Open(); //Open the Connection

SqlCommand command1 = new SqlCommand("delete from jobs", cnn);
command1.ExecuteNonQuery(); // produces no result

SqlCommand command2 = new SqlCommand("select min(job_id) from jobs", cnn);
object scalar_result = command2.ExecuteScalar(); // produces scalar result

SqlCommand command3 = new SqlCommand("select * from jobs", cnn);
SqlDataReader myReader = command3.ExecuteReader(); // produces data reader

while (myReader.Read())
{
    int job_id = myReader.GetInt16(0);
    string job_desc = myReader.GetString(1);
    // ...
}
myReader.Close();
```

Data Access Methods (CRUD)

```
ProductList GetAllProducts()
ProductList GetLocalProducts(string region)
void Fill(ProductList list)

void Update(ProductList list)

void Update(Product p)
void Update(string name, string region, double price)

void Insert(Product p)
void InsertProduct(string name, string region, double price)

void Delete(Product p)
void DeleteProduct(string name)
```

Example Data Access Method

```
protected void g2p_InsertVolunteer(string hostaddress)
{
    SqlCommand command = new SqlCommand();

    command.CommandType = CommandType.Text;
    command.CommandText = "INSERT INTO Volunteers (HostAddress) VALUES (@HostAddress)";
    command.Connection = connection;

    SqlParameter HostAddress = new SqlParameter("@hostAddress", SqlDbType.VarChar, 50);
    HostAddress.Value = hostaddress;
    command.Parameters.Add(HostAddress);
    command.ExecuteNonQuery();
}
```

Using Stored Procedure

```
protected void g2p_InsertVolunteer(string hostaddress)
{
    SqlCommand command = new SqlCommand();

    command.CommandType = CommandType.StoredProcedure;
    command.CommandText = "InsertVolunteer";
    command.Connection = connection;

    SqlParameter HostAddress = new SqlParameter("@hostAddress", SqlDbType.VarChar, 50);
    HostAddress.Value = hostaddress;
    command.Parameters.Add(HostAddress);
    command.ExecuteNonQuery();
}
```

Implementing Data Classes

Options:

1. Create a class called `Product` with properties corresponding to each of the fields in the Products table.
   - `System.Collection.Generic.List<Product>`

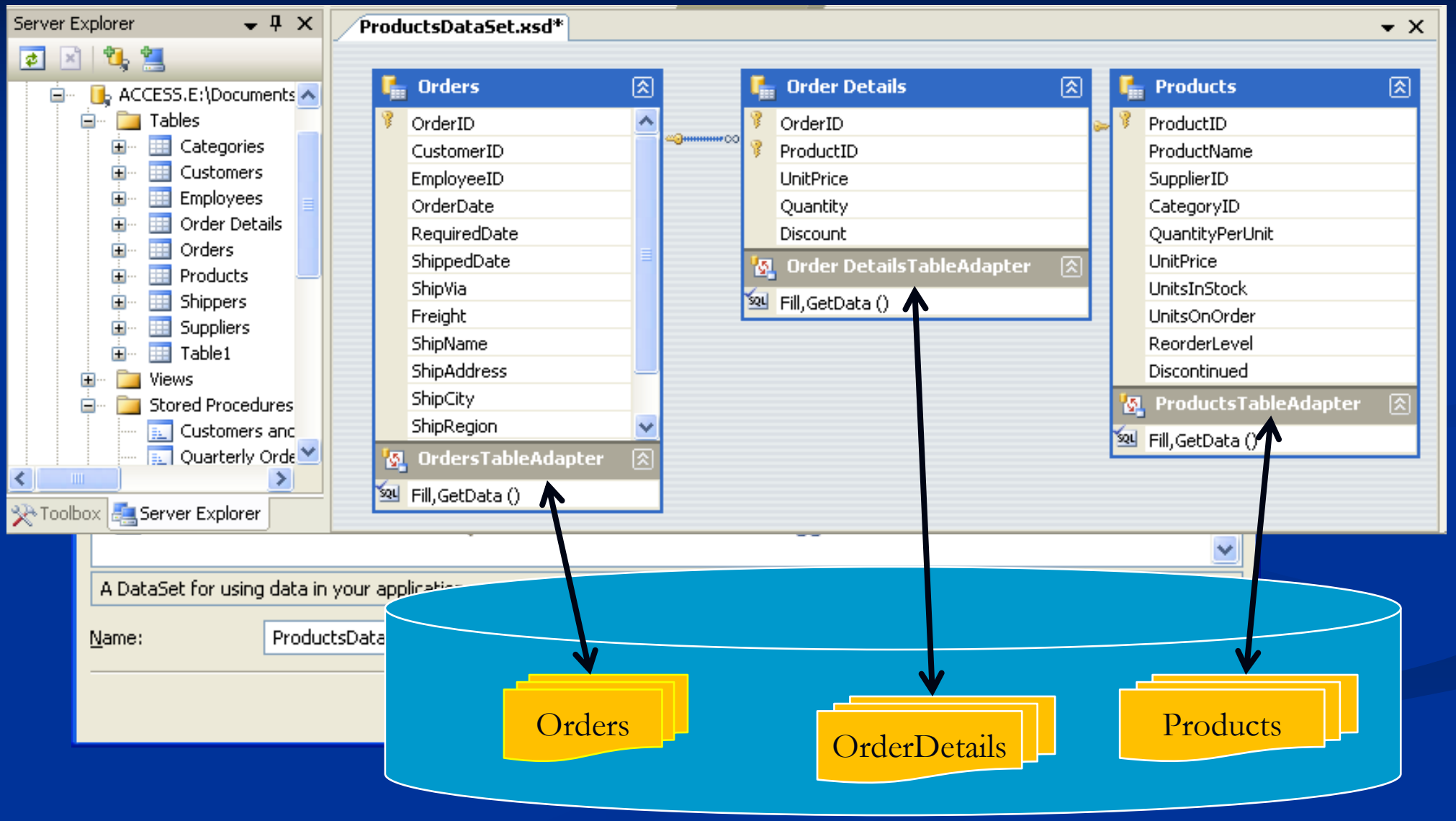
2. Use ADO.NET `DataSets` to represent sets of Products

ADO.NET DataSets

- A DataSet is a class designed to store relational data
- Each DataSet consists of one or more DataTables
  - Each DataTables contains zero or more DataRows consisting of one or more columns.
  - We can optionally define relationships and constraints between these tables
- Each DataTable is optionally associated with a TableAdapter class that can be used for syncing the DataTable with the corresponding table in with the database.

- You can use `System.Data.DataSet` as a *"vanilla"* (dynamically typed) dataset, or you can create a *Typed Dataset*

Creating Typed DataSets



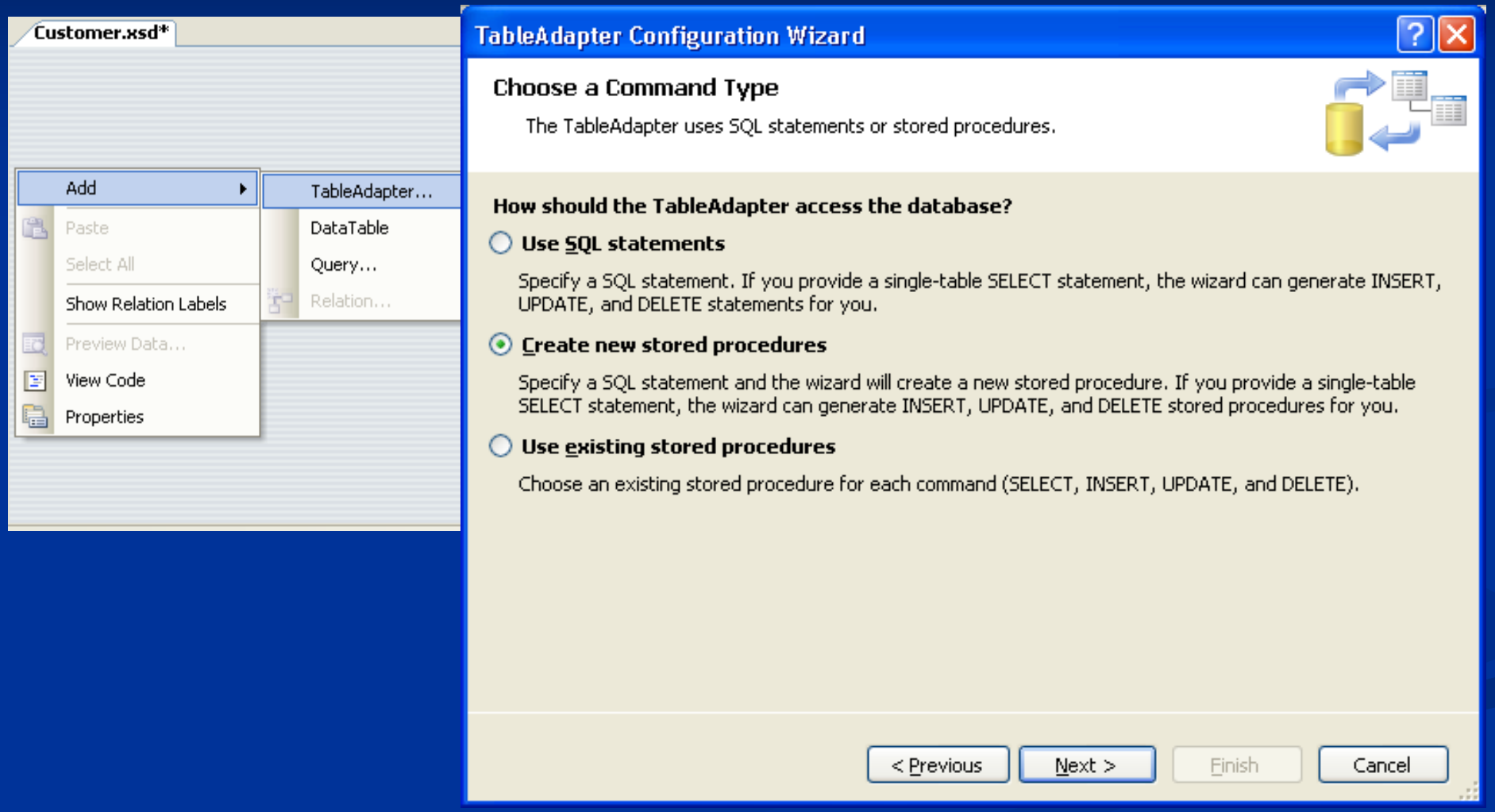
Using Typed DataSets

```
ProductsDataSetTableAdapters.ProductsTableAdapter productsAdapter =
    new ProductsDataSetTableAdapters.ProductsTableAdapter();

ProductsDataSet.ProductsDataTable products = productsAdapter.GetData();

foreach (ProductsDataSet.ProductsRow product in products)
    Console.WriteLine("{0} {1}", product.ProductName, product.UnitPrice);

productsAdapter.Insert(productName, supplierID, categoryID, QuantityPerUnit, Un

productsAdapter.Delete(45, "Bread", 43, 64, 10, 2.99, 0, 100, 10, false);

ProductsDataSetTableAdapters.OrdersTableAdapter orderAdapter =
    new ProductsDataSetTableAdapters.OrdersTableAdapter();

// ...
```

Table Adapters

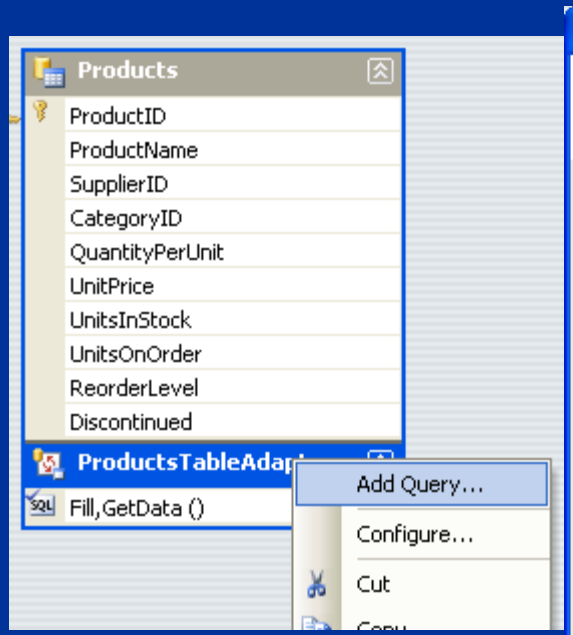
- TableAdapters encapsulate SQL commands to perform basic CRUD operations,
  - CREATE
  - READ
  - UPDATE
  - DELETE
- The SQL commands can be:
  - generated automatically:
    - Select command derived from table name
    - Update, delete and insert derived from Select command
  - or specified manually by the programmer.
    - as textual SQL commands or as stored procedures

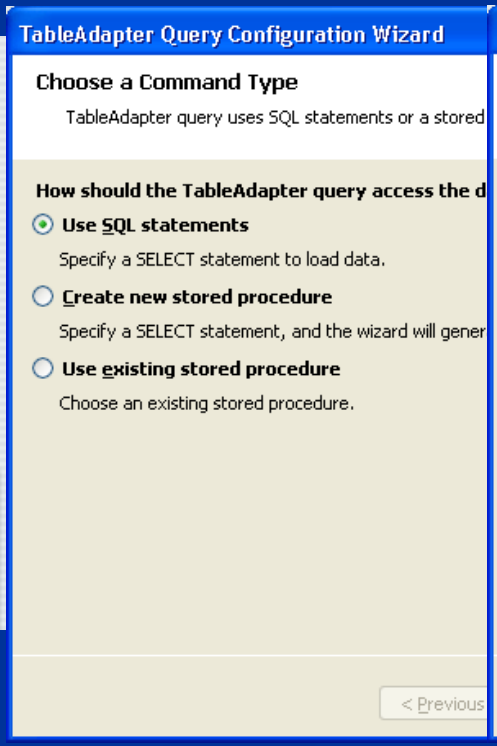
TableAdapters using Stored Procedures

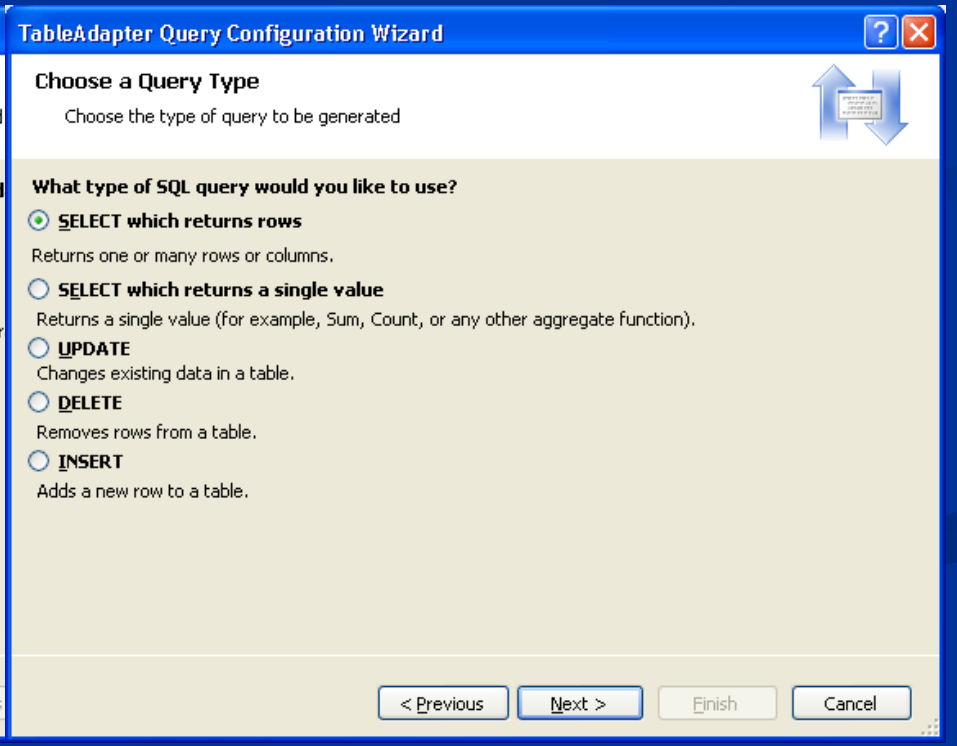


Extending TableAdapters

- Given that the TableAdapters will act as your Data Access classes, its useful to be able to customize their behaviour.
- You can do this by adding additional methods to access the database:



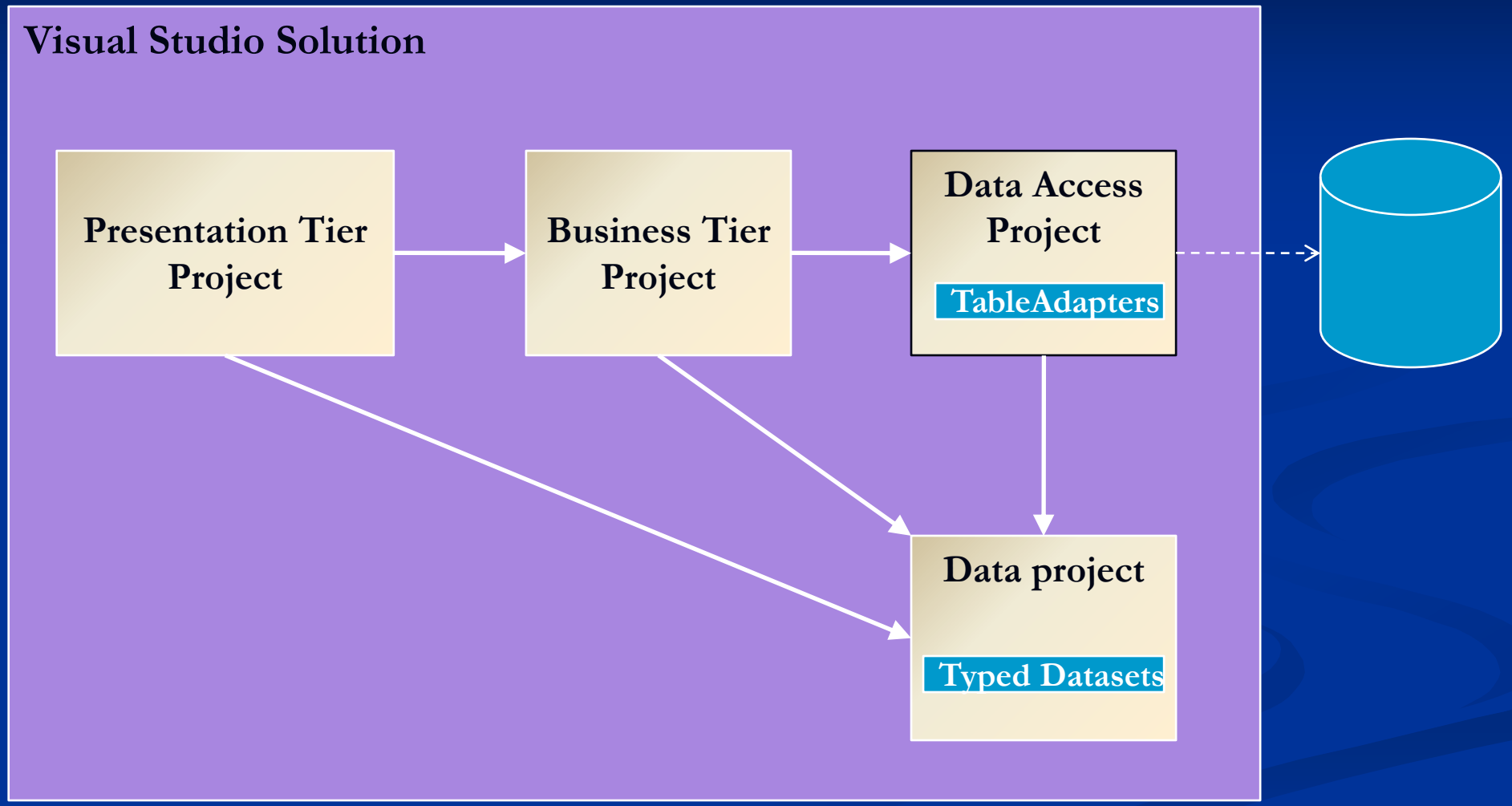




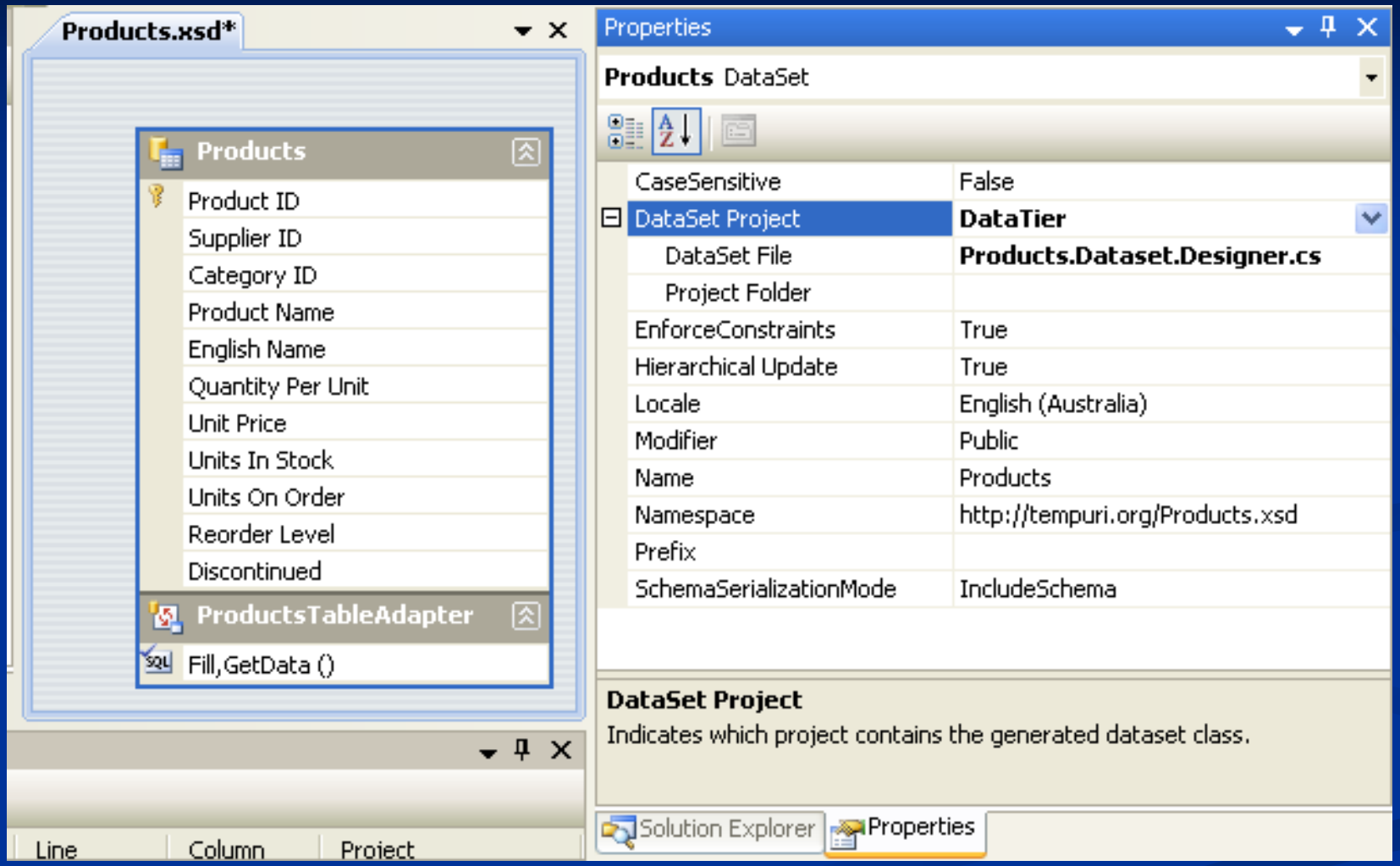
Separating DataAccess and Data classes

- The DataSets and DataTables are **Data** classes
  - they are just dumb container classes
  - it is OK to use these classes even in the presentation tier
- The TableAdapters are **Data access** classes
  - they shouldn't be contained in any component used by the presentation tier.
- So, we want to separate them and put:
  - The data sets and data tables in the Data project
  - The table adapters in the Data Access project

Separating DataSets and TableAdapters

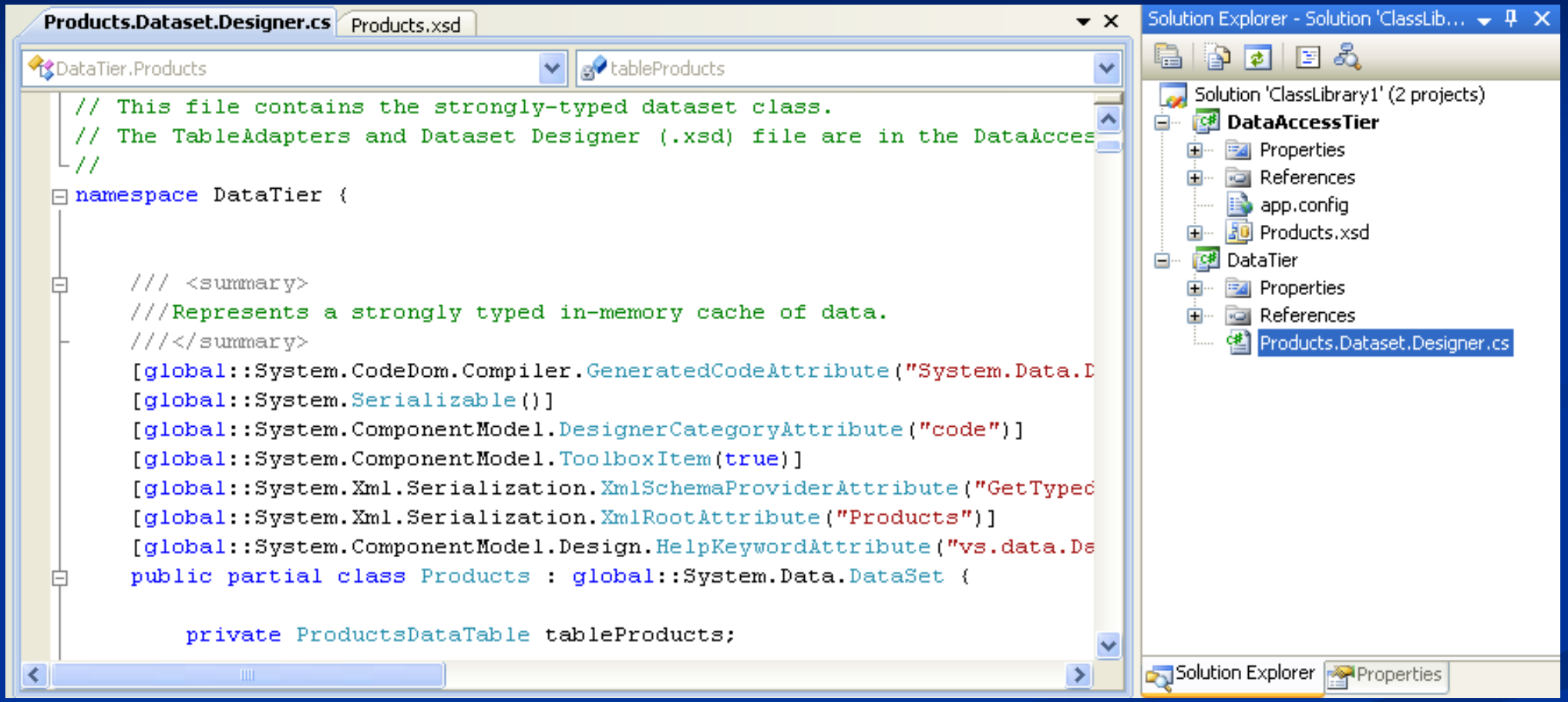


Separating DataSets and TableAdapters



http://msdn.microsoft.com/en-us/library/bb384586.aspx

Separated DataSets and TableAdapters



Object/Relational Mappers

- Object/Relational mapping systems aim to spare the programmer the tedium of having to manually:
  - create business entity classes with properties for every field in every table of the database
  - creating data access methods to perform CRUD operations on each table.
- Visual Studio's Dataset wizards are one of many such systems.
- Others include:
  - Hibernate (Java)
  - NHibernate (.NET)
  - Java Persistence API
  - ActiveRecord (Ruby)
  - LINQ to SQL (.NET Framework 3.5)

Language Integrated Query (LINQ)

- The next big challenge in programming technology is to reduce the complexity of accessing and integrating information that is not natively defined using OO technology.
  - The two most common sources are relational databases and XML.
- LINQ is a new .NET framework that supports general purpose query for all sources of information.
- Extends C# and Visual Basic with native syntax for queries.
- LINQ allows third parties to:
  - augment the set of standard query operators and to
  - replace the standard query operators with their own implementations

LINQ Example (C# Syntax)

```
// The Three Parts of a LINQ Query:
// 1. Data source.
Northwnd db = new Northwnd(@"c:\northwnd.mdf");

// 2. Query creation.
var orders =
from order in db.SalesOrders
where order.Status == "Pending Stock Verification" &&
      order.SalesPerson.State == "WA"
select order;

// 3. Query execution.
foreach(SalesOrder order in orders)
```

LINQ Frameworks

- Basic LINQ works on any IEnumerable<T> type.
- LINQ to XML works on XML data sources
- For relational data:
  - LINQ to SQL (direct to a relational database)
  - LINQ to DataSet (query an in memory dataset)
  - LINQ to Entity (abstract view of a relational database)
- Visual studio wizards exist for visually designing Business Entity classes from database schema definitions.
  - The Object Relational Designer
  - DataSet Designer
  - Entity Framework Designer

Transactions

- Transact SQL
- ADO.NET
- COM+

Transact SQL Transactions

Text:

```
CREATE PROCEDURE FetchJob
        @jobId bigint output,
        @volunteerId bigint
AS

BEGIN TRANSACTION

SET @jobId = (SELECT Top 1 JobId FROM ReadyJobs)

UPDATE Jobs
SET status = 2, volunteerId = @volunteerId
WHERE JobId = @jobId

COMMIT TRANSACTION
GO
```

ADO.NET Transactions

```
{
    SqlTransaction transaction = connection.BeginTransaction();

    SqlCommand command1 = new SqlCommand(cmdText1, connection);
    command1.Transaction = transaction;
    SqlCommand command2 = new SqlCommand(cmdText2, connection);
    command2.Transaction = transaction;

    try
    {
        command1.ExecuteNonQuery();
        command2.ExecuteNonQuery();
        transaction.Commit();
    }
    catch
    {
        transaction.Rollback();
    }
}
```

COM+ Transactions

```
[ Transaction(TransactionOption.RequiresNew)]
public class UpdateOrder : System.EnterpriseServices.ServicedComponent
{
    [ AutoComplete ]
    public void Add(OrderData order)
    {
        try
        {
            DataAccess.OrderdbOrder = new DataAccess.Order();
            OrderData.SaleDetail saleInfo = order.SaleDetails[0];
            dbOrder.AddSale(saleInfo, out saleId);
            saleInfo.SaleID = saleId;
            foreach(OrderData.SeatDetail s in order.SeatDetails)
                dbOrder.UpdateSeatPurchased(saleInfo.EventID, saleInfo.SectionID, sale

            int ccType = int.Parse(saleInfo.CreditCardTypeID.ToString());
            string ccName = saleInfo.NameOnCard;
            string ccNumber = saleInfo.CreditCardNumber;
            DateTime ccExpDate = DateTime.Parse(saleInfo.ExpirationDate.ToString());
            float purchaseAmount = float.Parse(saleInfo.TotalCharged.ToString());

            CreditCardProcessor ccProc = new CreditCardProcessor();
            string ccValidation = ccProc.ChargeCard( ccName, ccType, ccNumber, ccExpDa
            ccProc = null;
        }
        catch
        {
            throw;
        }
    }
```