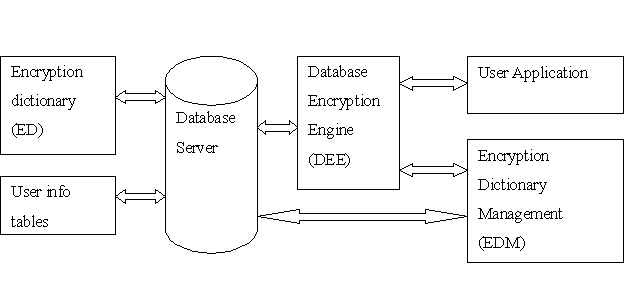
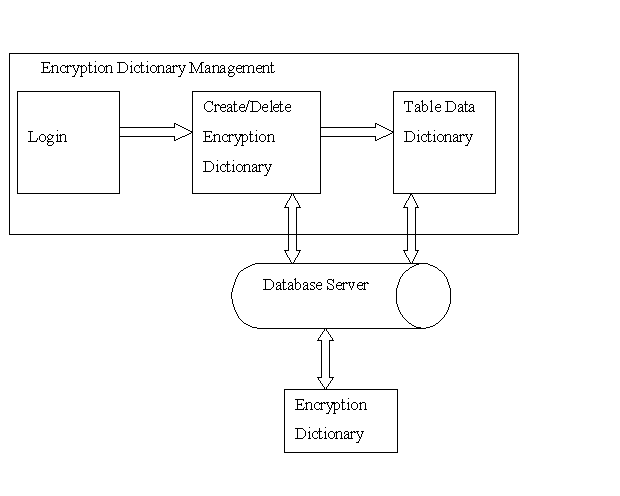
Architecture of system



Step 1: Encryption Dictionary Management (EDM)



Login -> Choose table and columns to encrypt -> Enter passphrase and usb key

1. Create the symmetric key (AES key)
2. Create a pair of public/private key
3. Encrypt the symmetric key with the public key. Store the public key in the database. Store the encrypted symmetric key in database
4. The private key will be stored in the USB key. Encrypt the private key with a password-derived key.

**数据库加脱密引擎**

该引擎设计采用三大模块组成：加脱密处理、语法分析和数据库接口. 加脱密处理该模块是数据库加脱密引擎的核心模块 其主要功能为加脱密引擎的初始化、 命令的加密转换、查询结果的脱密处理以及加脱密算法实现等等 其关键点在于截获提交的 语句，若存在加密要求则对提交的数据进行加密处理后替换原 语句，重新提交给数据库管理系统；在查询时，若存在加密字段，将加密表脱密后提交给客户端 具体的加密算法可采用新发布的高级加密算法 标 准（ ）［ ，］

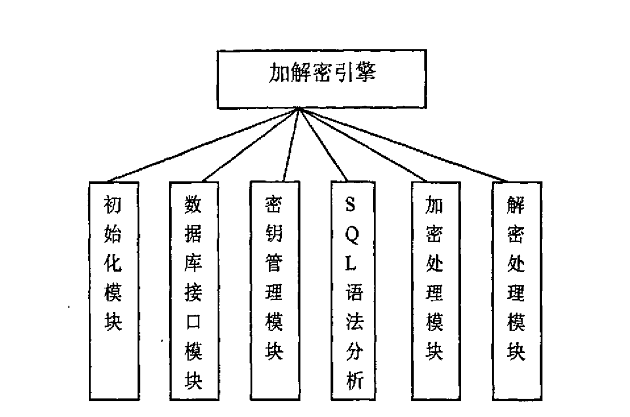
，这 种 对 称 密 钥 算 法 较 之

以前采用的 算法具有更高的安全保密性 具体处理过程如图 所示图 系统加脱密处理流程该模块与数据库接口的联系：客户端应用程序提交 命令，调用用户接口模块时被调用；加脱密处理中在需要访问后台数据库时，调用后台数据库接口模块提供的数据库服务该模块还用到了 语法分析模块提供的功能 其 过 程 为 将 用 户 提 交 的 命 令 传 递 给“ 语法分析模块”返回一棵树，用于加脱密语意分析 得到 语句中的表名、字段名以及需要加密处理的输入数据为提高引擎的处理速度，设置加密字典缓冲区，将存放最近访问的数据库表的字典信息，不加密的表虽然不存在与之对应的加密字典数据，但也登记在加密字典缓冲区中，从而便于系统快速判断一个表是否已加密

**语法分析**

本模块功能类似于该编译器的功能，将命令转换成易处理的树形式，从树中提取出数据库加脱密处理模块所需的参数（表名、字段名、插入数据等等）除此主要功能外，还包括由语法树转换成 命令的功能 主要子模块包括：词法分析器 用 于 对 命 令 进 行 词 法分析，分割成各个词法单位；语法树生成器 用于将分析得到的词法单位生成一棵语法树；语法树反向生成 命令，用于将经过加密变换后的语法树转换成新的 命令；语法错误处理，生的错误.

**数据库接口**

模块主要包括两部分：前端数据库客户应用程序访问数据库加脱密引擎的接口函数和数据库加脱密引擎访问后台数据库服务器的接口函数客户端访问数据库加脱密引擎的入口点是前端数据库接口模块，用户通过建立的连接和命令调用数据库接口模块提供的用户接口函数，对于不同的数据库应用编程接口，用户接口函数的定义是不同的当进行“加脱密处理模块”在需要数据库服务时，调用“后台数据库接口”提供的通用数据库驱动函数连接到数据源，取出用户所需的信息返回客户端 所以，数据库接口模块的工作实际上就是完成未加密前 驱动程序的工作，只是在中间插入加脱密处理模块，增加与加脱密模块的连接函数 处理过程为接受客户端的操作请求，传递给“加脱密处理模块”，代替“加脱密模块”去访问数据库服务器 

1. 初始化模块主要负责获取加、解密引擎在进行加解密时所必须的信息，这些信息通常被放在加密字典中，主要包括字段数据类型、大小、精度、小数位数、是否加密等。

Initialization module: Retrieve information related to the encryption and decryption, i.e, retrieve information from the data encryption dictionary.

2、数据库接口模块将所有访问数据库的操作封装在一起，屏蔽了各类数据库的特性，使得加解密处理模块不必关心实际使用的是哪种数据库。该模块包含两部分接口，一是前端数据库客户访问数据库加解密引擎的接口函数；二是数据库加解密引擎访问后台数据库服务器的接口函数。

Database interface module: Receives the SQL queries from the user application and sends to the SQL statement analysis module. Gets back the analyzed sql statement directly from the SQL analysis module or the encryption/decryption module. Sends the sql statement to the database and obtains the results. It then returns the result information to the user application.

3、本系统实行二级密钥管理。一级密钥为主密钥，二级密钥为工作密钥。主密钥的作用是对二级密钥信息加密生成工作密钥。工作密钥用于对数据库数据的加／解密。数据库的每条记录录入的时间信息都是唯一和确定的，所以可以在每条记录录入时增加时间字段，并对时间进行处理，把时间信息中的年、月、日、小时分、秒、毫秒信息进行处理得到一个整数型，把这个数作为函数void srand(unsigneA int s。ed)中的伪随机数种子，得到一个伪随机数。再把伪随机数和用户密钥进行异或处理，结果作为算法密钥，以此可以增加密钥的安全性，使算法更加难以破解。

本系统中，主密钥保护了工作密钥，工作密钥保护敏感信息。整个系统的安全依赖于主密钥的安全。主密钥的安全需要解决以下几个问题：

主密钥的生成：采用投币法产生128位二进制数据。

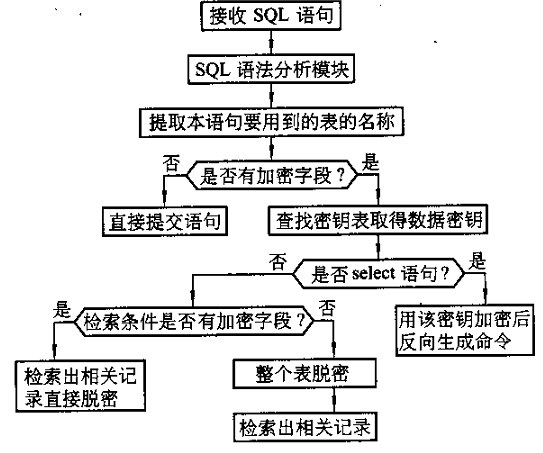
主密钥的存储：主密钥经加密并存放于安全区域内，使用时由系统自动获取并解密。本系统也可将主密钥注入密卡中以保证安全。主密钥的更换：在数据库加密系统中，主密钥的一旦更换是一个比较棘手的问题。主密钥更换以后，会造成工作密钥的全部更换。数据库中存储有巨大数量的数据，已经加密的数据需要用原来的密钥系统解密，而且解密时间将会很长。为安全起见，密钥更换前，要对数据系统进行全库备份，然后利用系统所提供的工具完成数据库中密文数据向明文数据的转换。更换主密钥后，再根据数据库加密要求进行明文数据向密文数据的转换。

4、SQL'语法分析模块功能是将用户提交的SQL命令转换成易处理的树形式的语法结构，用于加解密语意分析，得SQL语句中的表名、字段名以及需要加解密处理的数据。

SQL statement analysis module: Analyses the SQL statement and breaks it into a tree-structure. Retrieves the SQL statement’s table, fields. If there are any fields that need to be encrypted or decrypted, send to the Encryption/Decryption module. Else send back to database module.

5、加密模块块是加解密引擎的核心，它主要负责数据的加密。在该模块中，加密算法是其核心，如上所述我采用的是AES加密标准。该模块除了要对数据进行加密外．还要负责数据的拆分和组装。任何一种加密算法一次加密的数据长度都是有限的，而需加密数据的长度往往会超过这个长度．因此必须对需加密的数据以加密算法一次能加密的数据的长度为单位进行拆分；而加密以后又需要将加密后的数据组装在一起。当数据的长度不够时，该模块会在数据后面添O以达到加密的基本长度：解密后，解密模块还要负责去掉添加的0以还数据本来面目。

6、解密模块和加密模块一起构成了加解密引擎的核心，它的作用和加密模块相反，主要负责数据的解密。在该模块中解密算法是其核心，解密算法与所使用的加密算法应一致。与加密模块一样，解密模块也具有对数据进行拆分和组装的功能。



**Human Resources Management System Architecture**

Department Management

Human Resources Management System

System Management

Change Password

User Manage-ment

Modify Salary Table

Search Salary Table

Create Salary Table

Employee Salary Management

Info Management

Employee Management

SQL Injection Attacks- use of parametrized queries

**Securing Connection Strings**

<http://msdn.microsoft.com/en-us/library/89211k9b(v=vs.80).aspx>

1. **Prevent Connection String Injection Attacks**

A connection string injection attack can occur when dynamic string concatenation is used to build connection strings based on user input. If the user input is not validated and malicious text or characters not escaped, an attacker can potentially access sensitive data or other resources on the server. To address this problem, ADO.NET 2.0 introduces new connection string builder classes to validate connection string syntax and ensure that additional parameters are not introduced.

When applied to connections strings, the rule for the majority of database providers is the "last one wins" algorithm. If a KEYWORD=VALUE pair occurs more than once in the connection string, the value associated with the LAST occurrence is used. This opens the door to some serious attacks.

By way of example, in a web application, a user enters username and password; a subsequent connection string is generated to connect to the back end database.

*Data Source = myDataSource; Initial Catalog = db; Integrated Security = no; User ID = myUsername; Password = XXX;*

In the password field, if the attacker enters "xxx; Integrated Security = true", the connection string becomes,

*Data Source = myDataSource; Initial Catalog = db; Integrated Security = no; User ID = myUsername; Password = XXX; Intergrated Security = true;*

Under the "last one wins" principle, the web application will then try to connect to the database using the operating system account under which the application is running to bypass normal authentication.

Integrated Security: When false, User ID and Password are specified in the connection. When true, the current Windows account credentials are used for authentication.

1. **Use the Default for Persist Security Info**

The default value for Persist Security Info is false; we recommend using this default in all connection strings. Setting Persist Security Info to true or yes allows security-sensitive information, including the user ID and password, to be obtained from a connection after it has been opened. When Persist Security Info is set to false or no, security information is discarded after it is used to open the connection, ensuring that an untrusted source does not have access to security-sensitive information.

1. **Storing Connection Strings in Configuration Files**

You can also store connection strings in configuration files, which eliminates the need to embed them in your application's code. Configuration files are standard XML files for which the .NET Framework has defined a common set of elements. Connection strings in configuration files are typically stored inside the <connectionStrings> element in the app.config for a Windows application, or the web.config file for an ASP.NET application. The [System.Configuration](http://msdn.microsoft.com/en-us/library/system.configuration(v=vs.80).aspx) namespace allows programmatic access to configuration files using the [WebConfigurationManager](http://msdn.microsoft.com/en-us/library/system.web.configuration.webconfigurationmanager(v=vs.80).aspx) for Web applications and the WebConfigurationManager for Windows applications.

<connectionStrings>

<add name="partialConnectString"

connectionString="Initial Catalog=Northwind;"

providerName="System.Data.SqlClient" />

</connectionStrings>

/\*To use the connection string into the code\*/

string connectString =

ConfigurationManager.ConnectionStrings["partialConnectString"].ConnectionString;

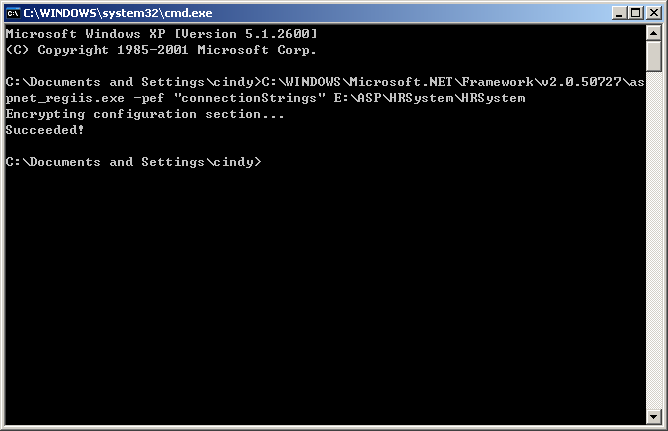
1. **Encrypting Configuration File Sections Using Protected Configuration**

Part of securing an application involves ensuring that highly sensitive information is not stored in a readable or easily decodable format. Examples of sensitive information include user names, passwords, connection strings, and encryption keys. Storing sensitive information in a non-readable format improves the security of your application by making it difficult for an attacker to gain access to the sensitive information, even if an attacker gains access to the file, database, or other storage location.

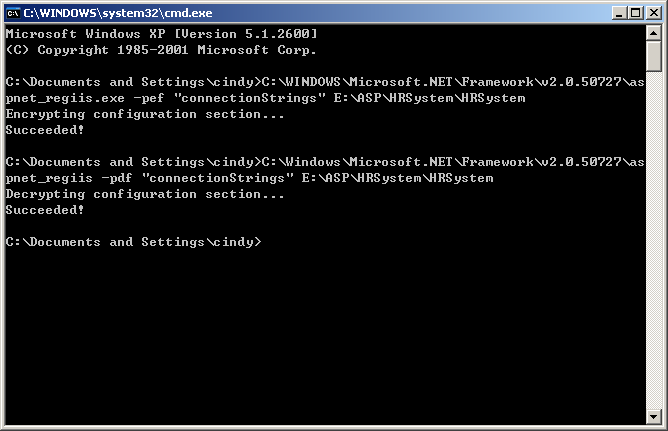
One of the primary places that sensitive information is stored in an ASP.NET application is the Web.config file. To help secure information in configuration files, ASP.NET provides a feature called protected configuration, which enables you to encrypt sensitive information in a configuration file.

Encrypt:

C:\Windows\Microsoft.NET\Framework\v2.0.50727\aspnet\_regiis -pef "connectionStrings" folder where web.config is found



Decrypt:



<configuration>

<system.web>

<compilation debug="true" targetFramework="4.0" />

</system.web>

<connectionStrings>

<add name="CnStr" connectionString="Server=CINDY-AK\SQLEXPRESS; Database=HrSys; User Id=hradmin; password= Hr\_Admin123; Integrated Security=SSPI;"

providerName="System.Data.SqlClient" />

</connectionStrings>

</configuration>

to

<configuration>

<system.web>

<compilation debug="true" targetFramework="4.0" />

</system.web>

<connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">

<EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"

xmlns="http://www.w3.org/2001/04/xmlenc#">

<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />

<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">

<EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">

<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1\_5" />

<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">

<KeyName>Rsa Key</KeyName>

</KeyInfo>

<CipherData>

<CipherValue>o00VmE47z9eWsOV8HeRGKgfYL20xMYITNcCWZrXY/Jg3nUFCSdoB+wk/PVfrBeRpHiIRI4QWgcyWHmsInGfWtKLmGpKC1SEeOctNbiF8NKHyUTmWGJgsJ+dZyepCVPJMiVnl/w3ph7xOj0zmTIaa73Kz7MTEhQtzvucPW9ongQg=</CipherValue>

</CipherData>

</EncryptedKey>

</KeyInfo>

<CipherData>

<CipherValue>yKycDakP6ydbVnWem1aiTJhHI+/tZpKtBywn6EkBDlD08S+4gvvZvykjFPkQpq6iFd5RRLNhpxN1lsXF/sDSXjT3ecBmJGO47CsvjBDmBm+60gFX7ZwItsLRE+s8vD7Ueh9S20td1Locj4rEWFojpqIpinJurWkjYvkUVctkm25B51Dpi36GfMIMTHcns8l7E9ohK4QvCvKwNcj8MDQ6oEc3Gj4tDVuJnkITcKK98UggL/UM1mf0Rb12abl/uZU9IbPucf/vaaEwFHNtqSSjsIonNaB0mRh7F40D5TnZGz9O/fPdNz7kGTNWzd2VkWk4mMp2DM6pv+E=</CipherValue>

</CipherData>

</EncryptedData>

</connectionStrings>

</configuration>

**Stored Procedures**

A parameterized query is the most secure against SQL Injection attacks. But what about stored procedures? Let’s take a closer look at these in this article.

A stored procedure is a database object just like table. It is a group of SQL statements that form a logical unit and perform a particular task. It is called using the name of the stored procedure and the parameter list. Stored procedures are widely used due to the benefits like encapsulation of business logic in a single entity, strong validation, faster execution and  exception handling. But are they safe against SQL injection attacks? Not always. SQL injection is possible if the dynamic SQL inside the stored procedure is not handled properly.

--WRONG WAY

Create Procedure spTest(@UserName varchar(20),@UserPwd varchar(max))

AS

Begin

select UserName,UserPwd from Users where UserName='''+@UserName+''' And UserPwd = '''+@UserPwd+'''

End

--RIGHT WAY

Create Procedure spTest(@UserName varchar(20),@UserPwd varchar(max))

AS

Begin

select UserName,UserPwd from Users where UserName=@UserName And UserPwd = @UserPwd

End

In ASP .NET code:

string connString = System.Configuration.ConfigurationManager.ConnectionStrings["connStr"].ConnectionString;

using (SqlConnection con = new SqlConnection(connString))

{

SqlCommand cmd = new SqlCommand("spTest", con);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.AddWithValue("@UserName",txtUserName.Text);

cmd.Parameters.AddWithValue("@UserPwd", txtPassword.Text);

con.Open();

SqlDataReader dr = cmd.ExecuteReader();

dr.Read();

if (dr.HasRows)

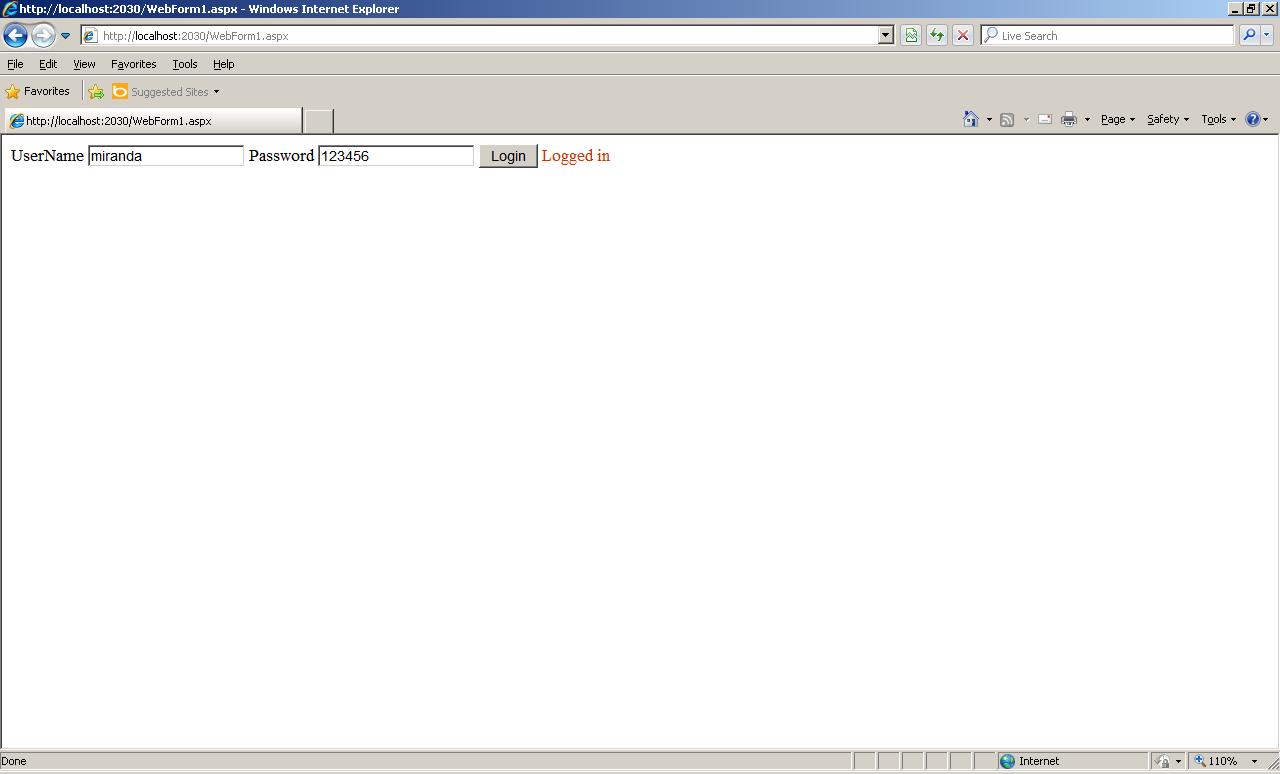
{

labelLogin.Text = "Logged in";

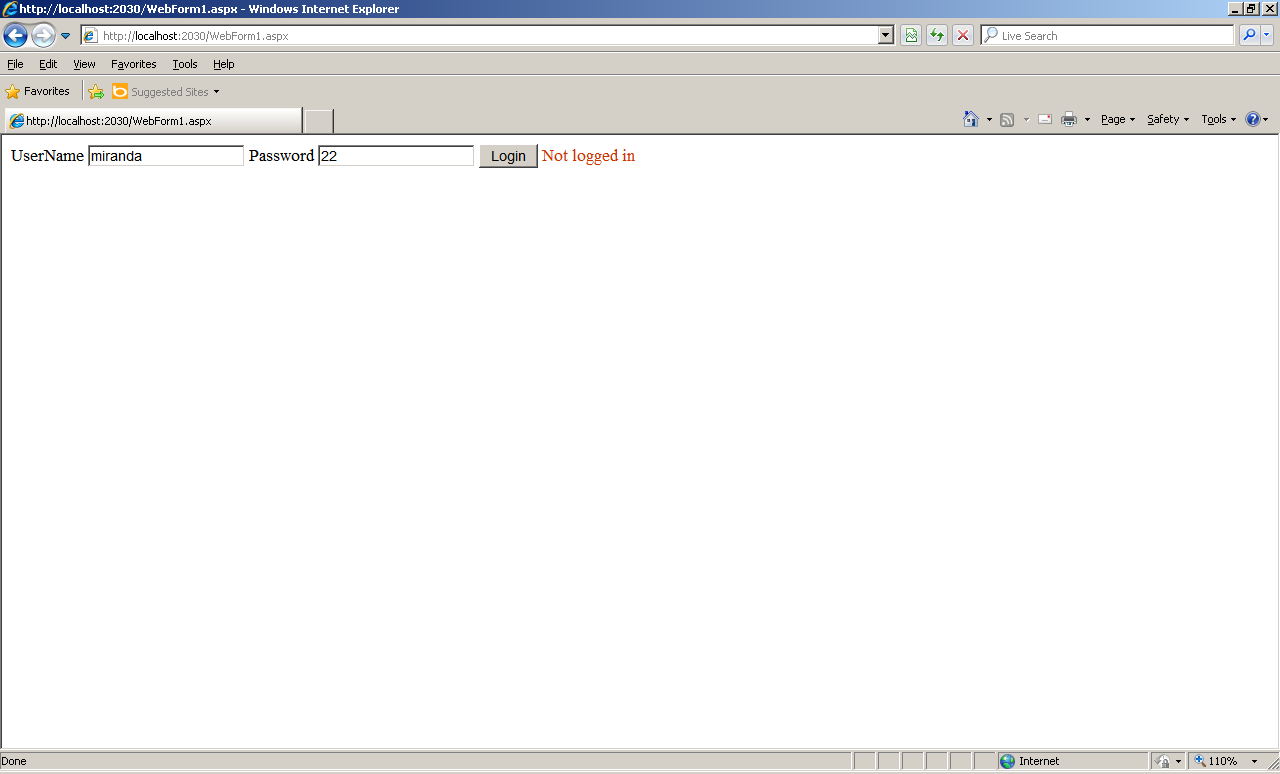
}

else { labelLogin.Text = "Not logged in"; }

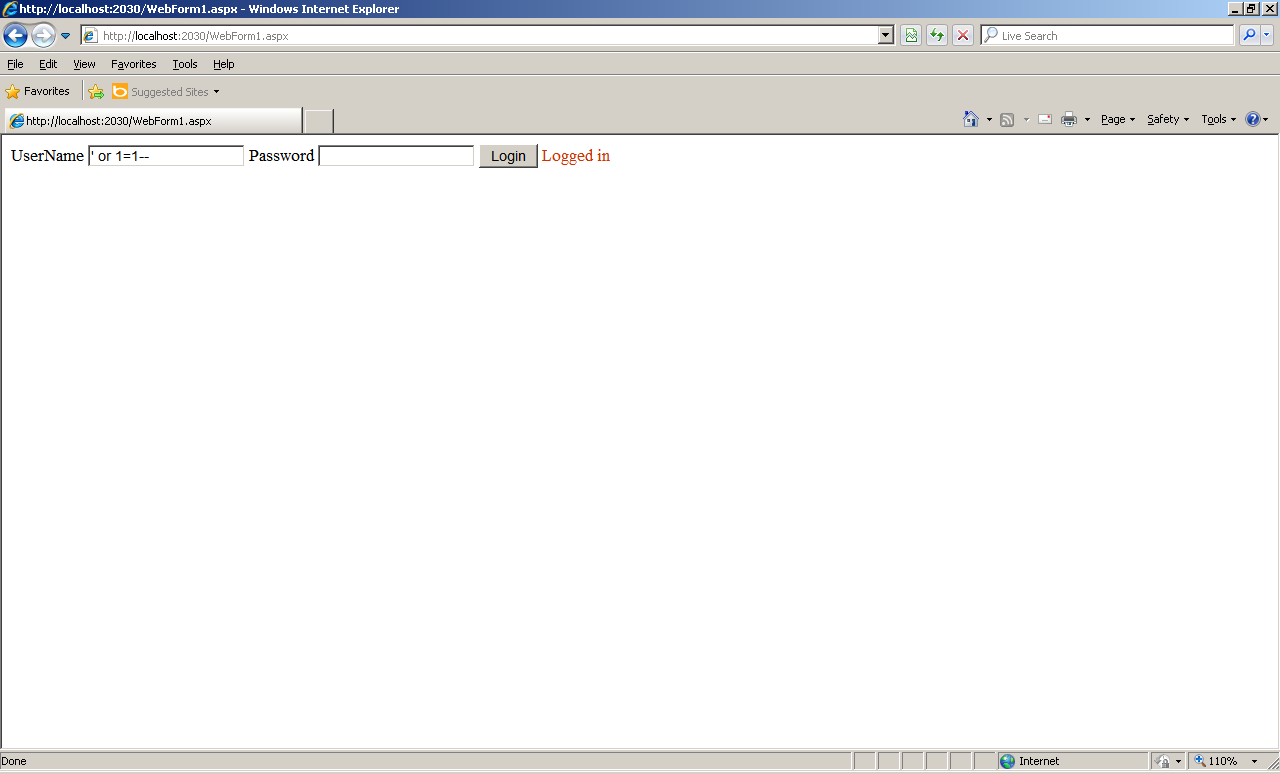
Valid Username/Password



Invalid Username/Password



Sql Injection



In spite of this, stored procedures have two big advantages for larger enterprises:

* They allow you to define an application interface for the database, so that the system can be shared between multiple applications without having to duplicate logic in those applications.
* They move the sql code to the db, where you can easily have an experienced DBA tune, update, and otherwise maintain it, rather than application developers who often don't know exactly what they're doing with database code.

Of course, these advantages aren't without cost:

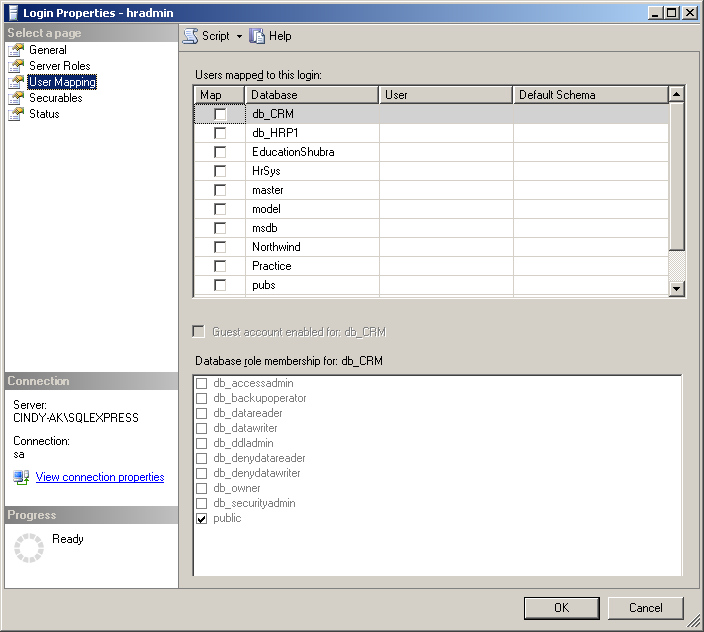
* It's harder to track changes in source control
* The database code is far separated from the code that uses it
* Developer tools for managing many stored procedures are less than ideal (if you've ever open the stored procedures folder in management studio to find 200 procedures for a database, you know what I'm talking about here).

**Login and Users**

A login is a security principal, or an entity that can be authenticated by a secure system. Users need a login to connect to SQL Server. You can create a login based on a Windows principal (such as a domain user or a Windows domain group) or you can create a login that is not based on a Windows principal (such as an SQL Server login).

As a security principal, permissions can be granted to logins. The scope of a login is the whole Database Engine. To connect to a specific database on the instance of SQL Server, a login must be mapped to a database user. **Permissions inside the database are granted and denied to the database user, not the login.**By default, the database user has the same name as the login.

Create a login hradmin. If the login is not mapped to a user within a database HrSys, then the web app will fail to connect to the database. The web app can only connect to the server and not to the database.



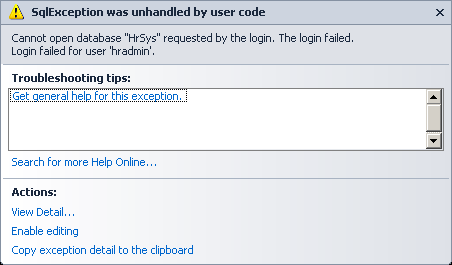
<connectionStrings>

<add name="connStr"

connectionString="Server=CINDY-AK\SQLEXPRESS;database=HrSys;uid=hradmin;password=Hr\_Admin123"

providerName="System.Data.SqlClient"/>

</connectionStrings>



Create Role hrAccessDB

grant execute on spTest to hrAccessDB

deny select on dbo.Users to hrAccessDB

exec sp\_addrolemember 'hrAccessDB', 'hradminUser'

Because every time you pass a query string to SQL Server the code has to be compiled etc, stored procedures are already compiled and ready to run on the server.

Also you are sending less data over the network although this is generally a minimal impact anyway.

EDIT: As a side note stored procedures have other benefits.

1) Security - Since the actual query is stored on the server you are not transmitting this over the network which means anyone intercepting your network traffic does not gain any insight into your table structure. Also a well designed SP will prevent injection attacks.

2) Code seperation, you keep your database code in your database and your application code in your application, there is very little crossover and I find this makes bug fixing a lot nicer.

3) Maintainability and Code Reuse, you can reuse a procedure many times without having to copy paste the query, also if you wish to update the query you just have to update it in one place.

4) Decreased network traffic. As mentioned above this may not be an issue for most people but with a large application you can significantly reduce the ammount of data being transferred via your network by switching to using stored procedures.

You can create a stored procedure once, store it in the database, and call it any number of times in your program. Someone who specializes in database programming may create stored procedures; this allows the application developer to concentrate on the code instead of SQL. You can modify stored procedures independently of the program source code--the application doesn't have to be recompiled when/if the SQL is altered.

If the operation requires a large amount of T-SQL code or is performed repetitively, stored procedures can be faster than batches of T-SQL code. During creation, stored procedures are parsed and optimized, and you can use an in-memory version of the procedure after the procedure executes the first time. Each time a T-SQL statement from the client is run, SQL Server compiles, optimizes, and executes the statement.

You can perform an operation that requires hundreds of lines of T-SQL code through a single statement that executes the code in a procedure, rather than by sending hundreds of lines of code over the network.

**ConfirmButtonExtender**

<http://www.asp.net/web-forms/videos/ajax-control-toolkit/how-do-i-use-the-aspnet-ajax-confirmbutton-extender>

**Add imagebutton to gridview**

<Columns>

<asp:TemplateField>

<ItemTemplate>

<asp:ImageButton ID="btnImgDelete" runat="server" ImageUrl="~/images/delete.gif"

ImageAlign="Middle" OnClientClick="deleteDept()" />

</ItemTemplate>

</asp:TemplateField>

</Columns>

# [ASP.NET Gridview delete row only on confirmation](http://stackoverflow.com/questions/9026884/asp-net-gridview-delete-row-only-on-confirmation)

http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.rowdeleting.aspx

http://stackoverflow.com/questions/9026884/asp-net-gridview-delete-row-only-on-confirmation

Step 1: Add the OnClientClick to the Image Button/Link Button

Step 2:Add the CommandName. When it is “Delete”, it will automatically call RowDeleting of the gridview.

Step 3: Define the RowDeleting for the GridView.

<asp:GridView ID="GridViewDept" runat="server" Width="75%" OnRowDeleting="GridViewDept\_RowDeleting">

<Columns>

<asp:TemplateField>

<ItemTemplate>

<asp:ImageButton ID="btnImgDelete" runat="server" ImageUrl="~/images/delete.gif" ImageAlign="Middle" CommandName="Delete" OnClientClick="return confirm('Are you certain you want to delete this questionnaire?');" />

</ItemTemplate>

</asp:TemplateField>

</Columns>

<HeaderStyle BackColor="Gainsboro" Font-Names="Segoe UI" Font-Size="13px" Height="35px" HorizontalAlign="Justify" />

<RowStyle Font-Names="Segoe UI" Font-Size="12px" ForeColor="#333333" />

</asp:GridView>

<!--ModalPopup for "Done" message-->

<asp:ModalPopupExtender ID="ModalPopupExtDone" runat="server" PopupControlID="donePanel" TargetControlID="donePanel" CancelControlID="donePanelBtn" BackgroundCssClass="modalBackground">

</asp:ModalPopupExtender>

<div class="panel">

<asp:Panel ID="donePanel" runat="server" Width="200px" Height="41px"

BackColor="#FF9900" EnableViewState="False">

<asp:Label ID="donePanelLabel" runat="server" ForeColor="#FF3300"></asp:Label>

<br />

<asp:Button ID="donePanelBtn" runat="server" Text="OK" />

</asp:Panel>

</div>

**Stored Procedures used**

Create Procedure spGetEmp (@EmpID int=null,@FamilyName varchar(50)=null)

AS

BEGIN

Declare @DeptID int

select @DeptID = DeptID from Departments where Department = @Department

select e.[EmpID], e.[FamilyName], e.[FirstName], e.[Gender], e.[Nationality], CONVERT(VARCHAR(8), e.DateOfBirth, 1) AS [DateOfBirth],

e.[MaritalStatus],

e.[Address],CONVERT(VARCHAR(8), e.HireDate, 1) AS [HireDate],dept.[Department],e.[Salary],

e.[SocialSecurityNum],p.[Position]

from Employees as e

LEFT JOIN Departments as dept ON e.DeptID = dept.DeptID

LEFT JOIN Positions as p ON e.PositionID = p.PositionID

where e.EmpID=@EmpID or e.FamilyName=@FamilyName or [e.DeptID=@DeptID](mailto:e.DeptID=@DeptID);

END

Hashtable ht = new Hashtable();

switch(DropDownListSearch.SelectedItem.Value)

{

case "Employee ID":ht.Add("@EmpID", txtSearchEmp.Text); break;

case "Family Name":ht.Add("@FamilyName", txtSearchEmp.Text); break;

}

SqlDataReader dr = dbsql.GetDataReader("spGetEmp",ht);

dr.Read();

if (dr.HasRows)

{

GridViewEmp.DataSource = dr;

GridViewEmp.DataBind();

}

**Notes:**

1. Remove the time part from the datetime data and keep only the date part.

From 3/10/1981 12:00:00 AM to 03/10/81

<http://msdn.microsoft.com/en-us/library/ms187928.aspx>

[2] Giving optional parameters to stored procedures: give default values to the parameters.

create Procedure spInsertEmp

(@EmpID int,@FamilyName varchar(50),@FirstName varchar(50),@Gender char(1),@Nationality varchar(50),

@Birth smalldatetime,@MaritalStatus varchar(20),@Address varchar(50),@HireDate smalldatetime,

@Department varchar(40),@Salary int,@SocialSecurityNum varchar(MAX),@Position varchar(50))

AS

BEGIN

Declare @DeptID int

Declare @PositionID int

select @DeptID=DeptID from Departments where Department=@Department;

select @PositionID=PositionID from Positions where Position=@Position;

INSERT INTO Employees Values(@EmpID,@FamilyName,@FirstName,@Gender,@Nationality,@Birth,@MaritalStatus,@Address,

@HireDate,@DeptID,@Salary,@SocialSecurityNum,@PositionID)

END

Create procedure spUpdateEmp

(@EmpID int,@FamilyName varchar(50),@FirstName varchar(50),@Gender char(1),@Nationality varchar(50),

@Birth smalldatetime,@MaritalStatus varchar(20),@Address varchar(50),@HireDate smalldatetime,

@Department varchar(40),@Salary int,@SocialSecurityNum varchar(MAX),@Position varchar(50))

AS

BEGIN

Declare @DeptID int

Declare @PositionID int

select @DeptID=DeptID from Departments where Department=@Department;

select @PositionID=PositionID from Positions where Position=@Position;

Update Employees

Set FamilyName = @FamilyName, FirstName = @FirstName,

Gender = @Gender, Nationality = @Nationality, DateOfBirth = @Birth, MaritalStatus= @MaritalStatus,

Address = @Address, HireDate = @HireDate, DeptID = @DeptID, PositionID = @PositionID,

Salary = @Salary, SocialSecurityNum = @SocialSecurityNum

Where EmpID = @EmpID;

END

create procedure spInsertUser

(@UserName varchar(20),@EmpID int,@Pwd\_hash char(32),@RandomKey nvarchar(50),@BaseSet bit,@EmployeeSet bit,

@SalarySet bit,@SystemSet bit)

AS

BEGIN

Insert Into Users(UserName,EmpID,Pwd\_hash,randomKey,BaseSet,EmployeeSet,SalarySet,SystemSet)

Values(@UserName,@EmpID,@Pwd\_hash,@RandomKey,@BaseSet,@EmployeeSet,@SalarySet,@SystemSet)

END

Create procedure spUpdateUser

(@EmpID int,@UserName varchar(20),@BaseSet bit,@EmployeeSet bit,

@SalarySet bit,@SystemSet bit)

AS

BEGIN

Update Users

Set UserName = @UserName, BaseSet = @BaseSet,EmployeeSet = @EmployeeSet,

SalarySet=@SalarySet,SystemSet=@SystemSet

where EmpID=@EmpID;

END

Create procedure spGetUser

(@EmpID int=null,@UserName varchar(20)=null)

AS

BEGIN

select u.[UserName],u.[EmpID],e.[FamilyName],e.[FirstName],

u.[BaseSet],u.[EmployeeSet],u.[SalarySet],u.[SystemSet] from Users as u

Left Join Employees as e

ON u.[EmpID]=e.[EmpID]

where u.UserName=@UserName or u.EmpID=@EmpID

END

create procedure spResetPwd

(@UserName varchar(20),@Pwd\_hash char(32),@RandomKey nvarchar(50)=null)

AS

BEGIN

Update Users

Set Pwd\_hash=@Pwd\_hash, randomKey = @RandomKey

where UserName = @UserName

END

^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\W).{10,}$

1) Search for at least one digit in any position  
2) Search for at least one upper or lower case in any position  
3) Enforce password to consist of 4-10 characters