



# Sitecore CMS 6.0

# Configuring Production Environments

*A Practical Guide for Sitecore Administrators and Developers*

## Table of Contents

Chapter 1	Introduction.....	3
1.1	Configuring CMS master server.....	4
1.1.1	Leaving the Web database on the CMS server.....	4
1.1.2	Sharing the Web database between CMS master and slave servers.....	5
1.2	Configuring content delivery server.....	7
1.2.1	Step 1. Choose and implement a security storage method.....	7
	Remove the Master database and share the Core database.....	7
	Remove the Core database from the content delivery environment.....	10
	Move security settings to the separate database.....	11
	Move security data to the Web.....	13
1.2.2	Step 2. Clean the solution (optional).....	13
1.3	Appendix 1.....	17
1.4	Appendix 2.....	21

# Chapter 1

## Introduction

This manual is designed to give you detailed instructions on configuring Sitecore to divide it into content creation server (CMS master server) and content delivery server(s).

**Note:** This guide describes the configuration for a default installation of Sitecore CMS 6.0.0 rev. 080623 and assumes that SQL 2005 Server is used as the database engine. If the guideline is used to reconfigure an existing setup, there may be additional steps that need to be done due to the difference in the configuration files. Also please don't forget that the following settings may be different on your installation thus may need to be adjusted if the sample `web.config` file is used: `dataFolder` and `licenseFile`.

**Attention:** This configuration has not been tested with all the modules.

**Note:** The Staging module should be installed prior to this configuration. For the detailed information on installing and configuring the Staging module, see [here](#).

## 1.1 Configuring CMS master server

There are two ways to locate Web database:

- Leaving the Web database on the CMS server. We recommend method for the purpose of having a database for the final preview of the content before it goes to production.
- Sharing the Web database between CMS master and slave servers. We recommend this method if you do not want to have a web database on the CMS server for the preview/QA purposes.

### 1.1.1 Leaving the Web database on the CMS server

We recommend this configuration for the purpose of having a database for the final preview of the content before it goes to production. This setup can also be useful for troubleshooting publishing issues.

For this configuration, you will keep the standard “web” connection string to publish to the web database in the CMS environment. You will add a reference to the content delivery web database to be able to publish to it from the CMS server.

#### Backup config files

Backup the `web.config` and `/App_config/ConnectionStrings.config` files.

#### Add a reference to the production web database in the `ConnectionStrings.config` file

Open the `/App_config/ConnectionStrings.config` file and add the connection string to point to the production web database:

```
<?xml version="1.0" encoding="utf-8"?>
<connectionStrings>
  <!--
    Sitecore connection strings.
    All database connections for Sitecore are configured here.
  -->
  <add name="core" connectionString="user id=user;password=password;Data
Source=(server);Database=Sitecore Core" />
  <add name="master" connectionString="user id=user;password=password;Data
Source=(server);Database=Sitecore Master" />
  <add name="web" connectionString="user id=user;password=password;Data
Source=(server);Database=Sitecore_Web" />
  <add name="production" connectionString="user id=user;password=password;Data
Source=(server);Database=Sitecore Production" />
</connectionStrings>
```

#### Copy and paste the web database definition and change the ID to production as in the `ConnectionStrings.config` file:

Open the `web.config` file.

```
<database id="production" singleInstance="true" type="Sitecore.Data.Database,
Sitecore.Kernel">
  <param desc="name">$(id)</param>
  <icon>Network/16x16/earth.png</icon>
  <securityEnabled>true</securityEnabled>
  <dataProviders hint="list:AddDataProvider">
    <dataProvider ref="dataProviders/main" param1="$(id)">
      <disableGroup>publishing</disableGroup>
      <prefetch hint="raw:AddPrefetch">
        <sc.include file="/App_Config/Prefetch/Common.config" />
        <sc.include file="/App_Config/Prefetch/Web.config" />
      </prefetch>
    </dataProvider>
  </dataProviders>
```

```

<proxiesEnabled>>false</proxiesEnabled>
<proxyDataProvider ref="proxyDataProviders/main" param1="$(id)" />
<archives hint="raw:AddArchive">
  <archive name="archive" />
  <archive name="recyclebin" />
</archives>
<cacheSizes hint="setting">
  <data>20MB</data>
  <items>10MB</items>
  <paths>500KB</paths>
  <standardValues>500KB</standardValues>
</cacheSizes>
</database>

```

### Add a publishing target pointing to the newly defined production database.

To add a publishing target:

- In the Content Editor, navigate to /sitecore/system/Publishing targets folder.
- Add one more Publishing target.
- The target database field must contain the database ID (in this example it is a “production”).

There are two optional steps:

### Modify the target database reference in the publishing agent.

In the web.config file perform the following modifications:

```

<agent type="Sitecore.Tasks.PublishAgent" method="Run" interval="00:00:10">
  <param desc="source database">master</param>
  <param desc="target database">production</param>
  <param desc="mode (full or incremental)">incremental</param>
  <param desc="languages">en, da</param>
</agent>

```

### Change the reference from web to production for the modules\_web and website definitions.

You may want to change the reference from web to production for the modules\_web and website definitions. This way the data will be fetched from the production database for these sites. In the web.config file perform the following modifications:

```

<site name="modules_website" virtualFolder="/sitecore modules/web"
physicalFolder="/sitecore modules/web" rootPath="/sitecore/content" startItem="/home"
language="en" database="production" domain="extranet" allowDebug="true" cacheHtml="true" />
<site name="website" virtualFolder="/" physicalFolder="/"
rootPath="/sitecore/content" startItem="/home" database="production" domain="extranet"
allowDebug="true" cacheHtml="true" htmlCacheSize="10MB" registryCacheSize="0"
viewStateCacheSize="0" xslCacheSize="5MB" filteredItemsCacheSize="2MB" enablePreview="true"
enableWebEdit="true" enableDebugger="true" disableClientData="false" />

```

## 1.1.2 Sharing the Web database between CMS master and slave servers

We recommend this configuration if you do not want to have a web database on the CMS server for the preview/QA purposes.

### Backup the /App\_config/ConnectionStrings.config file

Backup your /App\_config/ConnectionStrings.config file.

### Change the reference to the public web database in the /App\_config/ConnectionStrings.config file

Change the connection string so that it points to the public web database. In the `/App_config/ConnectionStrings.config` file perform the following modifications:

```
<add name="web" connectionString="user id=user;password=password;Data Source=(server);Database=Sitecore_Web" />
```

## 1.2 Configuring content delivery server

This section describes a setup of the content delivery server. The same approach can be used with multiple content delivery servers. In this case each additional content delivery server should be configured in the same way as a single one. In such case the content delivery server has the only web database.

To configure content delivery server:

- Step 1. Choose and implement a security storage
- Step 2. Clean the solution (optional)

### 1.2.1 Step 1. Choose and implement a security storage method

Since the security items are not processed by the Sitecore publishing operation, the decision should be made whether move the security definitions to a separate database shared between CMS master and content delivery server(s) or move the security definitions to the front-end web database which should be already configured for access from the CMS server, but some additional configurations on the CMS server must be performed.

Choose one of the following 4 methods of security storage:

- **Remove the Master database and share the Core database**  
Leave only two databases: Web and Core. Share the Core database between the CMS master server and content delivery server.
- **Remove the Core database from the content delivery environment**  
Get rid of the Core database and move security definitions to a separate database. Leave the minimum set of the databases on the content delivery side.
- **Move security settings to the separate database**  
Create separate database for storing security settings of CMS master and content delivery server. You can share Security database instead of sharing Core database.
- **Move security data to the Web**  
Configure Sitecore content delivery server to work with a single Web database

#### Important

To implement a security storage method you must implement all preceding storage methods.

### Remove the Master database and share the Core database

When making the following modifications, make sure that the configuration files in the `/App_Config/Include/` folder do not have anything that interferes with the following directions.

Backup the `web.config` and `/App_config/ConnectionStrings.config` files.

In the `/App_config/ConnectionStrings.config` file:

1. Remove the connections to the master database.
2. Modify the connection strings of the core databases so that they point to the corresponding database intended for the CMS master server.

In the `web.config` file:

1. Remove all databases under the <databases> section except for those defined in the /App\_config/ConnectionStrings.config file earlier. Note: if you do not want to share the core database, please refer to the “Move security settings to the separate database” section.
2. Remove the <archives> section for each <database>.
3. Disable archives:

```
<archives defaultProvider="sql" enabled="false">
  <providers>
  <clear />
  <add name="sql" type="Sitecore.Data.Archiving.SqlArchiveProvider, Sitecore.Kernel"
database="*" />
  <add name="switcher" type="Sitecore.Data.Archiving.SwitchingArchiveProvider,
Sitecore.Kernel" />
  </providers>
</archives>
```

4. Remove <indexes> section for each <database>.
5. Comment <indexes> section which contains the index definitions.

```
<indexes>
  <!--
  <index id="system" singleInstance="true" type="Sitecore.Data.Indexing.Index,
Sitecore.Kernel">
  <param desc="name">$(id)</param>
  <fields hint="raw:AddField">
  <field target="created"> created</field>
  <field target="updated"> __updated</field>
  <field target="author"> __updated by</field>
  <field target="published"> published</field>
  <field target="name">@name</field>
  <field storage="unstored">@name</field>
  <field target="template" storage="keyword">@tid</field>
  <field target="id" storage="unstored">@id</field>
  <type storage="unstored">memo</type>
  <type storage="unstored">text</type>
  <type storage="unstored" stripTags="true">html</type>
  <type storage="unstored" stripTags="true">rich text</type>
  </fields>
  </index>
  -->
</indexes>
```

6. In the <sites> section change references from “master” to “web”:

```
<site name="shell" ... content="web" /> <site name="testing" ... database="web" />
<site name="modules_shell" content="web" />
```

7. Remove the database agent operating the core database:

```
<!-- Agent to process schedules embedded as items in a database --> <agent
type="Sitecore.Tasks.DatabaseAgent" method="Run" interval="00:10:00"> <param
desc="database">core</param> <param desc="schedule
root"/>/sitecore/system/tasks/schedules</param> <LogActivity>true</LogActivity> </agent>
```

8. Change the reference of the database agent from the master to the web:

```
<!-- Agent to process schedules embedded as items in a database --> <agent
type="Sitecore.Tasks.DatabaseAgent" method="Run" interval="00:10:00"> <param
desc="database">master</param> <param desc="schedule
root"/>/sitecore/system/tasks/schedules</param> <LogActivity>true</LogActivity> </agent>
```

9. Change the connection of the Link database to the web:

```
<!-- LINK DATABASE -->
```

```

    <LinkDatabase type="Sitecore.Data.$(database).$(database)LinkDatabase,
Sitecore.Kernel">
      <param connectionStringName="web" />
    </LinkDatabase>

```

#### 10. Remove the Upload Watcher:

```

    <!-- <add type="Sitecore.Resources.Media.UploadWatcher, Sitecore.Kernel"
name="SitecoreUploadWatcher" /> -->

```

#### 11. Remove the publishing agent. You do not need to publish on the public servers:

```

    <!-- Agent to publish database periodically --> <agent
type="Sitecore.Tasks.PublishAgent" method="Run" interval="00:00:00"> <param desc="source
database">master</param> <param desc="target database">web</param> <param desc="mode
(full or incremental)">incremental</param> <param desc="languages">en, da</param>
</agent>

```

#### 12. Adjust the URL parameter of the URL agent so that it points to the correct URL of the public server:

```

    <agent type="Sitecore.Tasks.UrlAgent" method="Run" interval="01:00:00">
      <param desc="url">http://localhost/sitecore/keepalive.aspx</param>
    <LogActivity>true</LogActivity>
  </agent>

```

**Note:** This parameter could be configured for any site (in case of multiple web servers). But if it is load balanced, it's unknown which instance would handle request so it is possible that the second one would be unloaded due to the idle interval. If it is designed as load balanced then there is some heavy load, so keeping application alive is done by customer's requests.

#### 13. Comment search related definitions in the Search manager:

```

    <configuration type="Sitecore.Search.SearchConfiguration, Sitecore.Kernel"
singleInstance="true">
      <indexes hint="list:AddIndex">
        <index id="system" type="Sitecore.Search.Index, Sitecore.Kernel">
          <param desc="name">$(id)</param>
          <param desc="folder"> system</param>
          <Analyzer ref="search/analyzer" />
          <locations hint="list:AddCrawler">
            <core type="Sitecore.Search.Crawlers.DatabaseCrawler, Sitecore.Kernel">
              <Database>core</Database>
              <Root>/sitecore/content</Root>
              <include hint="list:IncludeTemplate">
                <application>{EB06CEC0-5E2D-4DC4-875B-
01ADCC577D13}</application>
              </include>
              <Tags>application</Tags>
              <Boost>2.0</Boost>
            </core>
            <core-controlpanel
type="Sitecore.Search.Crawlers.DatabaseCrawler, Sitecore.Kernel">
              <Database>core</Database>
              <Root>/sitecore/content/applications/control panel</Root>
              <include hint="list:IncludeTemplate">
                <taskoption>{BDB6FA46-2F76-4BDE-8138-52B56C2FC47E}</taskoption>
              </include>
              <Tags>taskoption</Tags>
              <Boost>1.9</Boost>
            </core-controlpanel>
            <!--
            <master type="Sitecore.Search.Crawlers.DatabaseCrawler,
Sitecore.Kernel">
              <Database>master</Database>
              <Tags>master content</Tags>
            </master>

```

```

-->
</locations>
</index>
</indexes>
</configuration>

```

Test the site.

## Remove the Core database from the content delivery environment

It is possible to get rid of the core database and move security definitions to a separate database (in this case log in to Sitecore will not be possible even if the license allows it), if it is required to get the minimum set of the databases on the content delivery side. Removing the Core database causes minimal surface for possible hackers' attacks (because it will not contain a back-end). To remove the Core database from the content delivery environment use the following instructions:

1. Remove the connection string for the core from the `/App_config/ConnectionStrings.config` file and the core database definition from the `web.config` file.
2. In the `web.config` file perform the following changes:
  - o Change connection of the Link database to the web:

```

<!-- LINK DATABASE -->
<LinkDatabase type="Sitecore.Data.${(database)}.${(database)}LinkDatabase,
Sitecore.Kernel">
  <param connectionStringName="web" />
</LinkDatabase>

```

- o Change connection to the Task database:

```

<!-- TASK DATABASE -->
<TaskDatabase type="Sitecore.Data.${(database)}.${(database)}TaskDatabase,
Sitecore.Kernel">
  <param connectionStringName="web" />
</TaskDatabase>

```

- o Change connection to the ID Table:

```

<!-- ID TABLE -->
<IDTable type="Sitecore.Data.${(database)}.${(database)}IDTable, Sitecore.Kernel"
singleInstance="true">
  <param connectionStringName="web" />
  <param desc="cacheSize">500KB</param>
</IDTable>

```

- o Comment search related definitions in the Search manager:

```

<configuration type="Sitecore.Search.SearchConfiguration, Sitecore.Kernel"
singleInstance="true">
  <indexes hint="list:AddIndex">
    <index id="system" type="Sitecore.Search.Index, Sitecore.Kernel">
      <param desc="name">${id}</param>
      <param desc="folder">__system</param>
      <Analyzer ref="search/analyzer" />
      <locations hint="list:AddCrawler">
        <!--
      <core type="Sitecore.Search.Crawlers.DatabaseCrawler, Sitecore.Kernel">
        <Database>core</Database>
        <Root>/sitecore/content</Root>
        <include hint="list:IncludeTemplate">
          <application>{EB06CEC0-5E2D-4DC4-875B-
01ADCC577D13}</application>
        </include>
      </core>
      </locations>
    </index>
  </indexes>
</configuration>

```

```

        <Boost>2.0</Boost>
    </core>
    <core-controlpanel
type="Sitecore.Search.Crawlers.DatabaseCrawler, Sitecore.Kernel">
        <Database>core</Database>
        <Root>/sitecore/content/applications/control panel</Root>
        <include hint="list:IncludeTemplate">
            <taskoption>{BDB6FA46-2F76-4BDE-8138-52B56C2FC47E}</taskoption>
        </include>
        <Tags>taskoption</Tags>
        <Boost>1.9</Boost>
    </core-controlpanel>
    -->
    <!--
Sitecore.Kernel">
        <Database>master</Database>
        <Tags>master content</Tags>
    </master>
    -->
</locations>
</index>
</indexes>
</configuration>

```

- Configure the CMS server so that it points to the same Security database.
- Remove all the sites from <sites> section except for “website”, “modules web” and “shell”.  
**Important:** Make sure that none of the sites database attributes point to the core database. Change such attributes to any existing database (for example, to the web database).

### 3. Test the front-end of the site.

## Move security settings to the separate database

Since Sitecore 6 supports .NET based security system, it is possible to move security settings to the separate database.

1. Create the clear aspnet\_db database (detailed information see in Appendix 1) and place to the /app\_data folder.
2. Add a new connection to \App\_Config\ConnectionStrings.config file, for example:

```

<add name="security" connectionString="user id=sa;password=XXXXX;Data
Source=.\SQLEXPRESS;initial catalog=Sitecore6.Security;Connect Timeout=30"/>
</connectionStrings>

```

3. Attach the database in your SQL 2005 server with a logical name as Sitecore6.Security.
4. Make the following modifications in the web.config file:

```

    <membership defaultProvider="sitecore">
        <providers>
            <clear />
            ...
            <add name="sql" type="System.Web.Security.SqlMembershipProvider"
connectionStringName="security" applicationName="sitecore" minRequiredPasswordLength="1"
minRequiredNonalphanumericCharacters="0" requiresQuestionAndAnswer="false"
requiresUniqueEmail="false" maxInvalidPasswordAttempts="256" />
            ...
        </providers>
    </membership>

    <roleManager defaultProvider="sitecore" enabled="true">

```

```

    <providers>
    <clear />
    ...
    <add name="sql" type="System.Web.Security.SqlRoleProvider"
connectionStringName="security" applicationName="sitecore" />
    ...
    </providers>
  </roleManager>

  <profile defaultProvider="sql" enabled="true"
inherits="Sitecore.Security.UserProfile, Sitecore.Kernel">
  <providers>
  <clear />
  <add name="sql" type="System.Web.Profile.SqlProfileProvider"
connectionStringName="security" applicationName="sitecore" />
  ...
  </providers>
  <properties>
  <clear />
  <add type="System.String" name="SC UserData" />
  </properties>
  </profile>

```

5. In the `web.config` file, change the authentication settings and the Roles manager so that they point to the security database:

```

<!-- ROLES -->
  <rolesInRolesManager defaultProvider="sql" enabled="true">
    <providers>
    <clear />
    <add name="sql"
type="Sitecore.Security.Accounts.SqlServerRolesInRolesProvider, Sitecore.Kernel"
connectionStringName="security" rolesInRolesSupported="true"
globalRolesConfigStoreName="globalRoles" raiseEvents="true" />
    </providers>
  </rolesInRolesManager>
  <!-- AUTHORIZATION -->
  <authorization defaultProvider="sql">
    <providers>
    <clear />
    <add name="sql"
type="Sitecore.Security.AccessControl.SqlServerAuthorizationProvider, Sitecore.Kernel"
connectionStringName="security" embedAclInItems="true" />
    </providers>
  </authorization>

```

6. Copy additional tables to the Security database using the following queries (see Appendix 1):

```

SELECT [Id],[MemberRoleName],[TargetRoleName],[ApplicationName],[Created] into
[Sitecore6.Security].[dbo].[RolesInRoles] FROM [Crestone Core].[dbo].[RolesInRoles]
Use [Sitecore6.Security]
ALTER TABLE [dbo].[RolesInRoles] ADD CONSTRAINT [DF_RolesInRoles_Id] DEFAULT
(newid()) FOR [Id]
SELECT [Id],[EntityId],[TypeName],[AccessRules] into
[Sitecore6.Security].[dbo].[AccessControl] FROM [MsavCrestone_Core].[dbo].[AccessControl]

```

7. In the `web.config` file, change the Client DataStore reference:

```

<!-- CLIENT DATASTORE -->
  <clientDataStore type="Sitecore.Data.$(database).$(database)ClientDataStore,
Sitecore.Kernel">
    <param connectionStringName="web" />
    <param desc="object lifetime">00:20:00</param>
  </clientDataStore>

```

## Move security data to the Web

Due to the new architecture of Sitecore security model it is possible to configure Sitecore content delivery server to work with a single database.

The following procedure describes how to move security data to the Web database:

1. In SQL Management Studio generate a script for creating Security database. In the databases context menu, select Tasks - Generate scripts, select the database and check "Script all objects in the selected database" checkbox, click **Finish**). As a result you will get a script that should be saved to the file system.
2. Apply newly created script to your production database.
3. Copy data from the security database (created on the previous step) to your production database using the Export Data tool (detailed information see in Appendix 2):
  - o In the Object Explorer right click [Security\_Base]. In the context menu select Tasks, Export Data.
  - o In a new window click Next.
  - o In the **Choose a Destination** window specify a data source "SQL Native Client", current server name, database ([Security\_Base]), user name and password and then click Next.
  - o In the **Choose a Data Source** window specify a data source "SQL Native Client", current server name, database ([Production\_Base]), user name and password and then click Next.
  - o In the **Specify Table Copy or Query** window select Copy data from one or more tables or views and then click Next.
  - o In the **Select Source Tables and Views** window select the following options:
    - (a) Select all the tables except [dbo.sysdiagrams] (don't select views, it might cause an error).
    - (b) Select "Optimize for many tables" option.
    - (c) Select "Run in a transaction" option.
    - (d) Click Next.
  - o In the **Save and Execute Package** window accept defaults and click Next.
  - o In the **Complete the Wizard** window click Finish.
4. Update definitions for membership providers both on CMS and content delivery servers. They must point to the production database. (Please see "Move security settings to the separate database" section).

The result is that the production database stores security for the CMS and content delivery servers and content delivery server works with a single database.

### 1.2.2 Step 2. Clean the solution (optional)

In order to make the content delivery instance more lightweight and secure, it is possible to remove the whole content management part of the product. Note that if this step is not executed, you may get an exception after requesting "/sitecore" on the content delivery since the core database is not available any more. To eliminate that, you can restrict the access to /sitecore on the IIS level.

#### Removing the content of the /Sitecore folder

If you remove the content of the `/sitecore` folder, please beware that you will not have access to Sitecore backend anymore on the content delivery servers. This may potentially complicate upgrading because there will be no access to the Sitecore Installation Wizard for update deployment. In this case you will need to manually update Sitecore by moving the file assets from the update packages to the file system of the content delivery servers.

Remove everything from the `/sitecore` folder except for the following folders and files:

- `/service` – can be removed completely if web.config settings (described later in this section) are adjusted.
- `/shell/sitecore.version.xml` – can be removed if the `VersionFilePath` setting (described later in this section) is adjusted.
- `/login/default.css` – the CSS file used on most of the aspx pages under `/sitecore/service` thus can be removed if the reference on the service pages is adjusted.
- `/images` – contains one blank.gif file. If you plan to leave aforementioned service pages as they are with no modification to meet the overall site styling, you can either leave it there or modify the source of the service pages.

The following settings that can be adjusted to move the service pages out from the `/sitecore` folder and also get rid of the `/shell` folder:

```
<agent type="Sitecore.Tasks.UrlAgent" method="Run" interval="01:00:00">
  <param desc="url">/sitecore/service/keepalive.aspx</param>
  <LogActivity>true</LogActivity>
</agent>
<setting name="ErrorPage" value="/sitecore/service/error.aspx" />
<setting name="ItemNotFoundUrl" value="/sitecore/service/notfound.aspx" />
<setting name="LayoutNotFoundUrl" value="/sitecore/service/nolayout.aspx" />
<setting name="LinkItemNotFoundUrl" value="/sitecore/service/notfound.aspx" />
<setting name="NoAccessUrl" value="/sitecore/service/noaccess.aspx" />
<setting name="NoLicenseUrl" value="/sitecore/service/nolicence.aspx" />
<setting name="VersionFilePath" value="/sitecore/shell/sitecore.version.xml" />
```

If you move the service pages from `/service` folder, change the `virtualFolder` and `physicalFolder` attributes of the `service` site to the new location of the pages, for example:

```
<site name="service" virtualFolder="/pages" physicalFolder="/pages" />
```

## Removing tables

If it was decided to remove the `core` database reference from the content delivery side, you can remove the following tables from the authoring core database because it is not used for hosting security schema any more:

```
drop table [AccessControl]
drop table [aspnet_PersonalizationAllUsers]
drop table [aspnet_PersonalizationPerUser]
drop table [aspnet_Profile]
drop table [aspnet_SchemaVersions]
drop table [aspnet_WebEvent_Events]
drop table [aspnet_Membership]
drop table [aspnet_UsersInRoles]
drop table [RolesInRoles]
drop table [aspnet_Roles]
drop table [aspnet_Users]
drop table [aspnet_Paths]
drop table [aspnet_Applications]
```

## Removing /Indexes folder

Remove the `/indexes` folder on the content delivery side. Note that if you use built-in Lucene search index on the content delivery side, you will need `/indexes` folder.

## Web.config cleaning

Remove or comment the following sections:

- `<events>`
- `<workflowHistoryStores>`
- `<processors>`
- `<dataviews>`
- `<pageextenders>`
- `<controlSources>`
- `<replacers>`
- `<fastCache>` for “shell” site:

```
<fastCache type="Sitecore.Resources.Media.FastMediaCache, Sitecore.Kernel">
  <param desc="Url prefix">/sitecore</param>
  <param desc="Site name">shell</param>
  <memoryCacheSize>200KB</memoryCacheSize>
</fastCache>
```

- `<watcher>`
- `<commands>`
- `<languageDefinitions>`
- `<icons>`
- `<portraits>`
- `<publishing>`
- `<watchers>` - `<media>` and `<watchers>` - `<config>`
- ConfigWatcher from `<httpModules>` and `<system.webServer>/<modules>`
- Filesystem database definition from `<databases>`
- DomainManager and `<domains>` section should be configured according to the requirements. You can remove unnecessary domains such as “sitecore” from the content delivery instance

Remove or comment the following scheduled agents:

- CleanupPublishQueue agent
- CleanupAgent’s timing intervals should be adjusted if necessary
- HtmlCacheClearAgent if not used

Additional configuration:

- Check `<httpRuntime maxRequestLength="16384" executionTimeout="600" />` if there is any resources consuming operations

- Adjust the settings in the <cacheSizes> section. See the [Sitecore Performance](#) article for more details.

### Removing unnecessary files

Delete the following files:

- /App\_Config/Commands.config
- /App\_Config/ConnectionStringsSQLite.config \*
- /App\_Config/Prefetch/Core.config
- /App\_Config/Prefetch/Master.config
- /App\_Config/Icons.config
- /App\_Config/LanguageDefinitions.config
- /App\_Config/Portraits.config
- /WebSite/web.config.sqlite \*
- /WebSite/webedit.css

\* If SQL Server is used for database storage.

## 1.3 Appendix 1

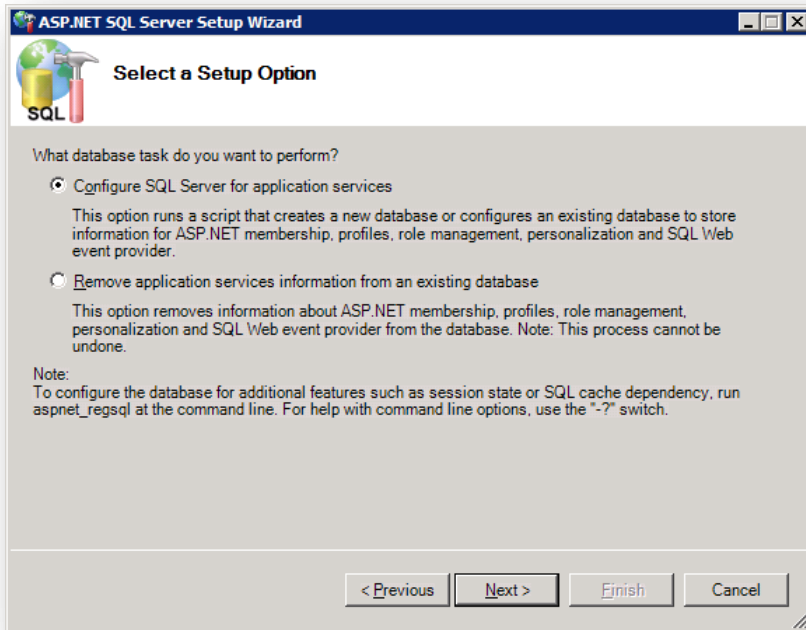
How to create the Application Services Database?

Call the wizard from the command prompt by using the following command on the machine that hosts SQL Server:

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet_regsql.exe
```

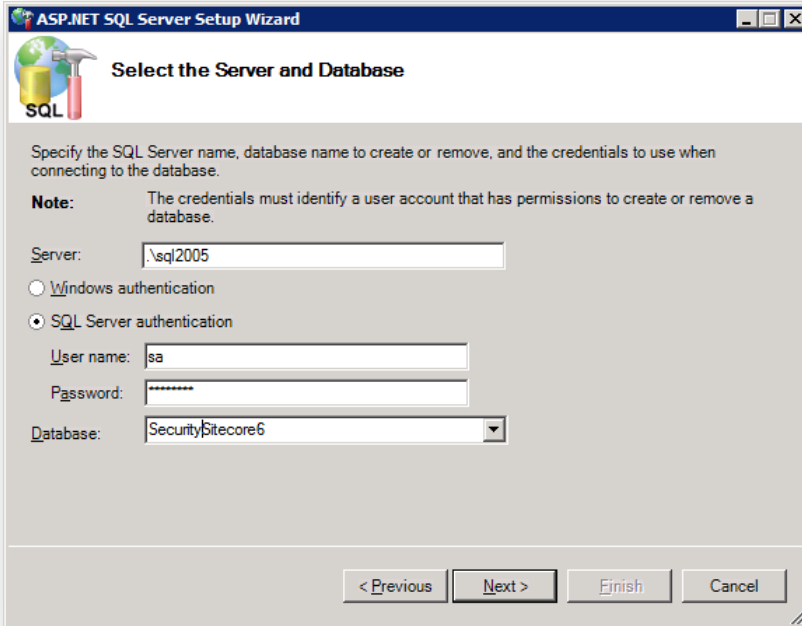
In the ASP.NET SQL Server Setup Wizard:

**Select “Configure SQL Server for application services”.**



**Specify login parameters for the SQL Server and a name for the new security database.**

Specify the SQL Server name, database name to create and the credentials to use when connecting to the database.

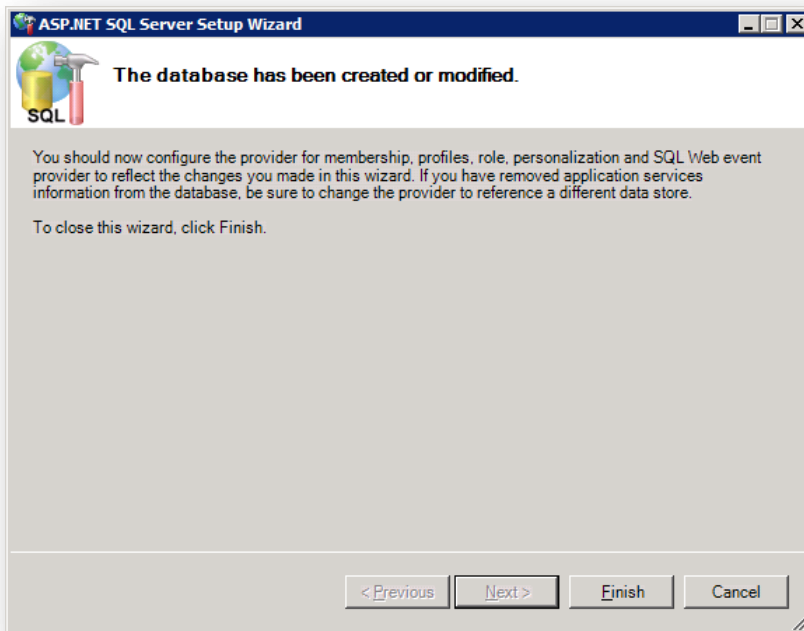


The image shows a screenshot of the "ASP.NET SQL Server Setup Wizard" window, specifically the "Select the Server and Database" step. The window title is "ASP.NET SQL Server Setup Wizard" and it has a standard Windows window icon. The main area contains the following text and controls:

- Select the Server and Database**
- Specify the SQL Server name, database name to create or remove, and the credentials to use when connecting to the database.
- Note:** The credentials must identify a user account that has permissions to create or remove a database.
- Server:** A text box containing the value "\sql2005".
- Windows authentication
- SQL Server authentication
- User name:** A text box containing the value "sa".
- Password:** A text box containing a series of asterisks "\*\*\*\*\*".
- Database:** A dropdown menu with the value "SecuritySitecore6" selected.
- Navigation buttons at the bottom: "< Previous", "Next >", "Finish", and "Cancel".

## Finish the setup.

Click Finish.



## Clean up the aspnet\_ SchemaVersions table.

When you finished the Wizard procedure, clean up the aspnet\_ SchemaVersions table using the following SQL script:

```
delete from [aspnet_SchemaVersions]
```

**Important:** If this step will cause “The 'System.Web.Security.SqlRoleProvider' requires a database schema compatible with schema version '1'” error then you should re-create Application Services Database skipping this step.

## Create RolesInRoles and AccessControl tables.

Since a valid Sitecore Security database contains two additional tables, execute the following SQL query to create them:

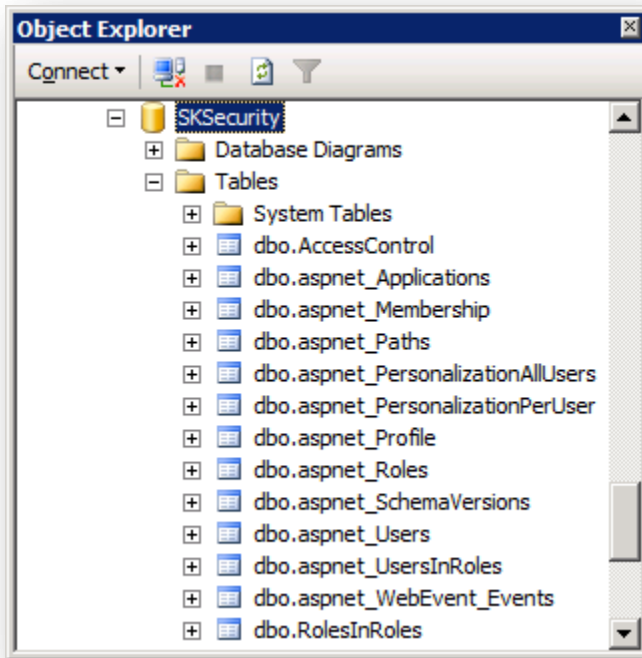
```
CREATE TABLE [dbo].[RolesInRoles] (
  [Id] [uniqueidentifier] NOT NULL CONSTRAINT [DF_RolesInRoles_Id] DEFAULT (newid()),
  [MemberRoleName] [nvarchar] (256) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
  [TargetRoleName] [nvarchar] (256) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
  [ApplicationName] [nvarchar] (256) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
  [Created] [datetime] NOT NULL
) ON [PRIMARY]

CREATE TABLE [dbo].[AccessControl] (
  [Id] [uniqueidentifier] NOT NULL,
  [EntityId] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
  [TypeName] [varchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
  [AccessRules] [nvarchar] (max) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
  CONSTRAINT [PK_AccessControl] PRIMARY KEY CLUSTERED
(
  [Id] ASC
```

```
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]  
) ON [PRIMARY]
```

**Check the set of tables.**

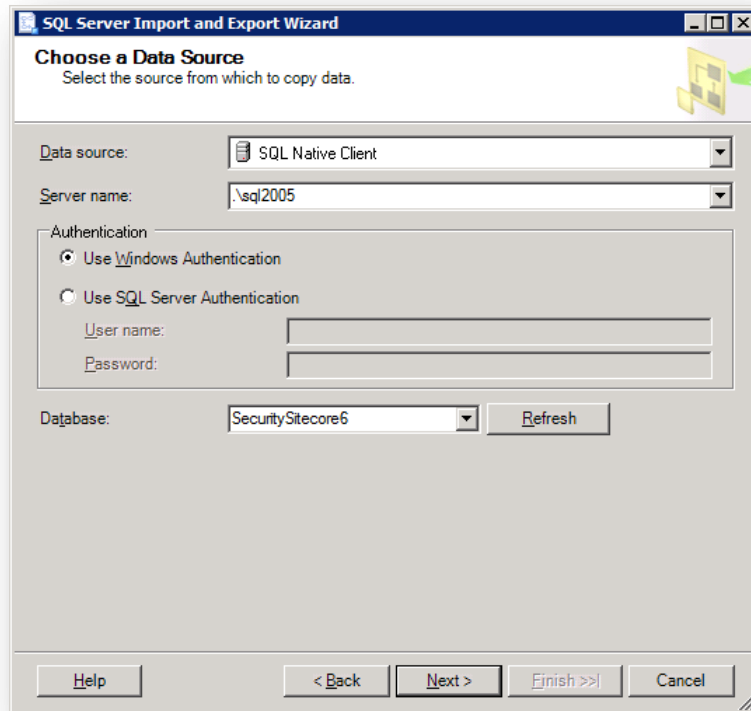
When previous two steps are accomplished, you will see the following set of tables for the new security database that you've just configured:



## 1.4 Appendix 2

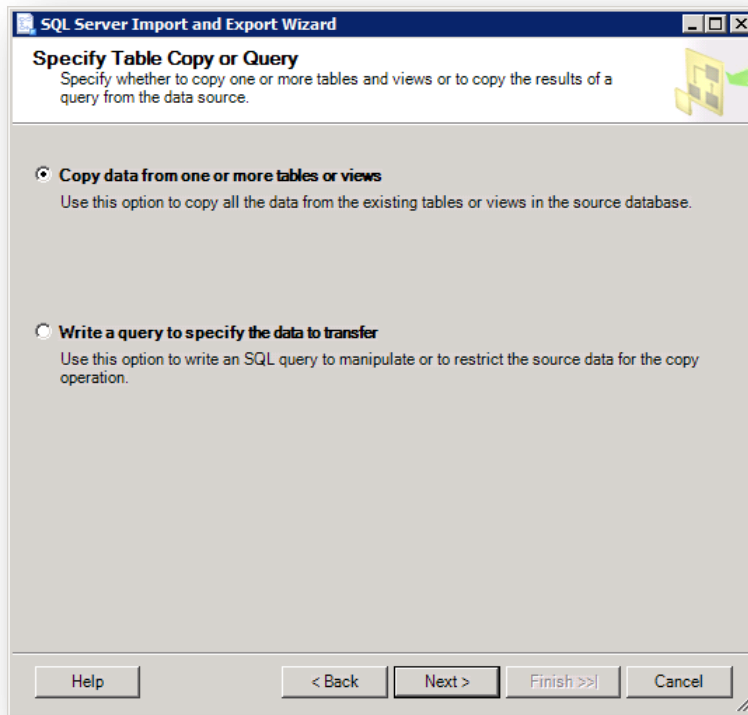
How to move the existing security definitions from one database to another?

- Open SQL Management Studio.
- Right click on the source database » Tasks » Export Data...
- In the **Welcome** window, click Next.
- In the "Choose a Data Source" step, specify the data source "SQL Native Client", current server name (".\sql2005"), specify the authentication parameters.

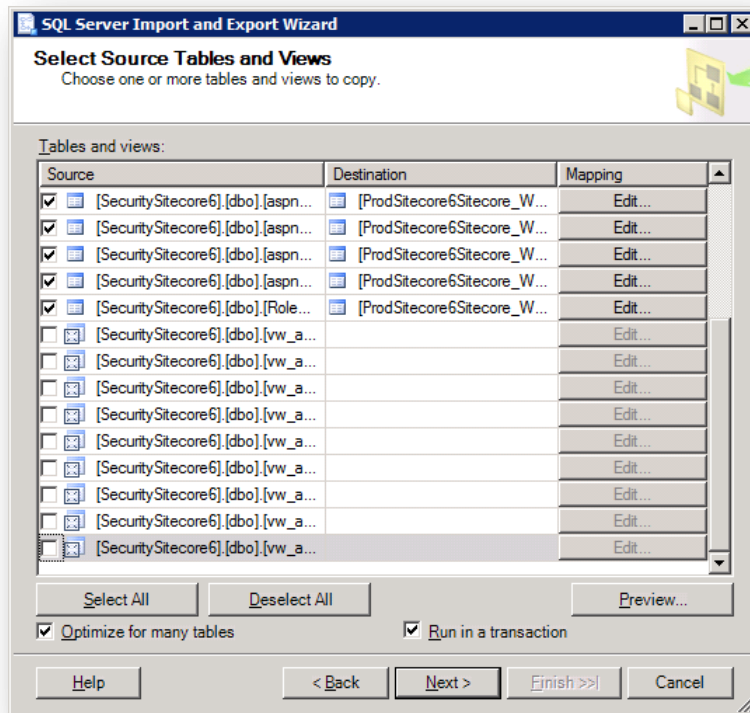


- In the "Choose a Destination" step, select the destination database and the same set of parameters as previously.

- In the "Specify Table Copy or Query" step, select "Copy data from one or more tables or views".



- In the "Select Source Tables and Views" step, check both "Optimize for many tables" and "Run in a transaction" checkboxes and select the following tables:
  - aspnet\_Applications
  - aspnet\_Membership
  - aspnet\_Paths
  - aspnet\_PersonalizationAllUsers
  - aspnet\_PersonalizationPerUser
  - aspnet\_Profile
  - aspnet\_Roles
  - aspnet\_SchemaVersions
  - aspnet\_Users
  - aspnet\_UsersInRoles
  - aspnet\_WebEvent\_Events
  - AccessControl
  - RolesInRoles



Click Next.

- In the "Save and Execute Package" step, accept the default settings ("Execute immediately" is checked).
- In the "Complete the Wizard" step click Finish.