Coveo Platform 7.0

Database Connector Guide

Notice

Document part number:	PM-120517-EN
Publication date:	1/3/2019

Table of Contents

1. Database Connector

The Coveo connector for databases allows to index database content directly through an ODBC connection and SQL queries.

1.1 Supported databases

The Database connector can index the following databases:

- Microsoft SQL Server
- Any database that has an OLE DB provider
- Any database that has an ODBC provider, including Oracle and DB2

1.2 Connector Features

The features of the Database connector are:

Database indexing

Fully customizable database indexing, either by specifying SQL queries or item types (table names).

Incremental Refresh

The connector uses one column to track changes and only fetch the updated rows (see "Enabling Incremental Refresh on a Database Source" on page 31).

Note: A source full refresh or rebuild is required to take deleted items into account.

Authentication and identity replacement

The connector supports standard authentication to databases (username and password), however you can hide the password and username by introducing the `@pwd` and `@uid` tokens. By doing so, it is not possible to see passwords in clear text, even if one has access to CES Administration (see "Replacing the Identity in Database Connection Strings" on page 30).

Complement information retrieval using subqueries

The connector indexes documents through a query against a database. Subqueries can run on every document to complete the information with more complex queries (see "Complement Information Retrieval with Subqueries for the Database Connector" on page 35).

Support running in 32-bit

On a 64-bit platform, the 64-bit version of the connector can use ODBC providers in both 32-bit and 64-bit.

Query-based security provider

You can expand external groups and users using a security provider when the database contains information allowing groups to be expanded to external users and/or external users to be mapped to Active Directory users (see "Enabling a Query-Based Security Provider for the Database Connector" on page 34).

Paged query execution

The connector mapping file can be configured to support executing queries in smaller subsets of data, instead of one big result; therefore the time required to execute a query can considerably be reduced (see "Enabling Paged Query Execution for the Database Connector" on page 12).

Multithreading

The connector can run multiple threads, which can improve performances considerably (see "Enabling Multithreading for the Database Connector" on page 29).

Field mapping

Fully customizable field mapping, from your database fields to CES custom and system fields.

Pause/Resume

You have the possibility to pause and resume while indexing a database (see "Enabling Pause/Resume on a Database Source" on page 32).

Indexing custom permissions

The connector can index custom permissions predefined for each document indexed.

Redirecting hyperlinks

Capability to redirect the hyperlink of a document to an existing Web interface or its Quick View of a respective HTML cached version.

What's Next?

Get familiar with the steps to deploy the Database connector (see "Database Connector Deployment Overview" on page 3).

2. Database Connector Deployment Overview

The following procedure outlines the steps required to deploy the Database connector. The steps indicate the order in which to perform required and optional configuration tasks.

1. Validate that your environment meets the requirements (see "Database Connector Requirements" on page 5).
2. In your DBMS, select or create a crawling account (user) that will be used by the CES connector. The CES connector requires a Full Read DBMS account that is able to read all tables referred to in the mapping file. Refer to the documentation specific to your DBMS.

   A best practice is to create a fixed password account that is used exclusively by the Coveo connector.
3. Create a configuration file

   You must create a configuration file that instructs the connector how to crawl the database for a given source. You can either create a configuration file from scratch or customize the provided sample file (see "Indexing a Database Using a Configuration File" on page 6).
4. On the Coveo server, in the Coveo Administration Tool:

   a. Optionally create a CES database user identity

      When you want to hide the account credentials in the database connection string, you can configure a user identity using the credentials of the crawling account that you selected or created in Step 2 (see "Adding a User Identity" on page 14) and use tokens in the database connection string (see "Replacing the Identity in Database Connection Strings" on page 30).

   b. Optionally create security providers

      When you want to index database permissions, you must create two security providers to get database group definition permissions (if any) and resolve and expand groups.

      i. Start by selecting or creating a security provider that the Database security provider will use to resolve and expand groups. The security provider type to use depends on how users are authenticated when they access your Coveo search interface:

         Note: CES 7.0.7.0.8850+ (March 2017) You may require to also use a REGEX Transform Member Name security provider in between the two following security providers to map member types (see Configuring a REGEX Transformation Security Provider). Contact Coveo Support for assistance.

         - CES 7.0.7.0.8850+ (March 2017) When authenticated with their email address, use an Email security provider (see Configuring an Email Security Provider).
         - When authenticated with an Active Directory account, use an Active Directory security provider (see Configuring an Active Directory Security Provider).

Notes:

- CES comes with an Active Directory security provider that you can configure to connect to the default domain. When your environment contains more than one domain, you can select an Active Directory security provider that you created for other domains.
- An Active Directory security provider is appropriate only when the User Principal Name (UPN) matches the email address for all users.

ii. Then create a Database security provider that the connector uses to resolve indexed permissions (see "Configuring a Database Security Provider" on page 15).

c. Configure and index the Database source

The connector must know details about the database to index its content (see "Configuring and Indexing a Database Source" on page 17).

d. Optionally, use paged query execution

If you encounter time out or performance issues, consider using paged queries to try resolving the issues (see "Enabling Paged Query Execution for the Database Connector" on page 12).

e. Optionally, customize the configuration file to fine-tune indexed content

Once your Database source is up and running, you can customize the configuration file of the connector to fine-tune the indexed content or index other tables in your DBMS (see "Example of a Configuration File for the Database Connector" on page 6 and "Additional XML Attributes" on page 11).

3. Database Connector Requirements

Your environment must meet the following requirements in order to use the Database connector:

- Coveo license for the Database connector

  Your Coveo license must include support for the Database connector to be able to use this connector.

- A database with ODBC or OLE DB provider
- Open ports

  When the access to communication ports between the Coveo Master server and the database server(s) is restricted, the appropriate database ports must be opened in the network infrastructure such as in firewalls to allow the Coveo connector to access the content (see Common Database Server Port Numbers or Configure the Windows Firewall to Allow SQL Server Access or Ports That Must Be Open to Make an ODBC or Data Transfer Connection through a Firewall).

4. Indexing a Database Using a Configuration File

You can use a configuration file to index data from a database. This file instructs the connector on the way to retrieve and copy the data from the record fields to the Coveo system and custom fields.

Considering that the required `URI` field is used as the underlying link of a search result, it is important to determine where a user is redirected when a result is opened.

- Best case scenario, a Web interface displays the results of the search, making it easy to recreate the address of a record page with fields from your database.
- Worst case scenario, the users are not redirected when they click on a search result.

Note: If the **Open results with cached version** option is selected, the document automatically opens in the Quick View (see "Configuring and Indexing a Database Source" on page 17).

To setup a configuration file

1. Create an XML-formatted configuration file and save it on the Coveo server. It is possible to create one from scratch or from the provided sample (see "Example of a Configuration File for the Database Connector" on page 6).

   Note: It is recommended to save it in the CES configuration folder (`[Index_Path]\Config`, by default `C:\CES7\Config`).

2. Create a database source and specify the path of the configuration file using the **Configuration File Path** source parameter (see "Configuring and Indexing a Database Source" on page 17).

4.1 Example of a Configuration File for the Database Connector

You can use the following configuration file with the Database connector to index data from the sample Northwind database that can be used with Microsoft Access and Microsoft SQL Server.

Notes:

- The following configuration file example will work with the Northwind database available from Microsoft Access 2013. When using other versions, validate that schema of your Northwind database matches with the parameter names in the following configuration file example.
- In the following example, you must replace `[PARAMETER]` by `@LastRefresh` in an SqlClient scenario or by `?` otherwise.
- A Database source can regularly run incremental refreshes to re-index only the latest changed information from the database when you configure incremental refresh (see "Enabling Incremental Refresh on a Database Source" on page 31).

```
<?xml version="1.0" encoding="utf-8" ?>
<ODBC>
  <CommonMapping excludedItems="Customers">
    <AllowedUsers>
      <AllowedUser type="Windows" allowed="true">
        <Name>everyone</Name>
        <Server></Server>
      </AllowedUser>
    </AllowedUsers>
  </CommonMapping>
  <Mapping type="Orders">
    <Accessor type="query">
      SELECT Shippers.Company AS ShipperName,
             Orders.[Order ID] AS ID,
             Orders.[Customer ID],
             Orders.[Order Date],
             Orders.[Shipped Date],
             Customers.Company,
             Employees.[Last Name],
             Employees.[First Name],
             Products.[Product Name],
             Products.[List Price]
      FROM   Orders, [Order Details], Shippers, Customers, Employees, Products
      WHERE  Orders.[Shipper ID] = Shippers.ID AND
             Orders.[Customer ID] = Customers.ID AND
             Orders.[Employee ID] = Employees.ID AND
             [Order Details].[Order ID] = Orders.[Order ID] AND
             [Order Details].[Product ID] = Products.ID AND
             Orders.[Order Date] >= [PARAMETER]
    </Accessor>
    <Fields>
      <Uri>http://www.coveo.com/Orders/details.aspx?Id=%[ID]</Uri>
      <ClickableUri>http://www.coveo.com</ClickableUri>
      <FileName>%[ID].txt</FileName>
      <Title>Order ID: %[ID]: %[Product Name]</Title>
      <ModifiedDate>%[OrderDate]</ModifiedDate>
      <Body>
        Customer: %[Company]
        OrderDate: %[Order Date]<br/>
        ShippedDate: %[Shipped Date]<br/>
        Shipped via: %[ShipperName]<br/>
        %[Product Name], $%[List Price]
      </Body>
      <CustomFields>
        <CustomField name="Type">Order</CustomField>
        <CustomField name="OrderDate">%[Order Date]</CustomField>
        <CustomField name="Shipper">%[ShipperName]</CustomField>
        <CustomField name="sysAuthor">%[First Name] %[Last Name]</CustomField>
      </CustomFields>
    </Fields>
    <AllowedUsers>
      <AllowedUser type="CustomGroup" allowed="true">
        <Name>%[First Name] %[Last Name]</Name>
        <Server></Server>
      </AllowedUser>
    </AllowedUsers>
  </Mapping>
  <Mapping type="Customers">
    <Accessor type="object">Customers</Accessor>
    <Fields>
      <Uri> http://www.coveo.com/Customers/details.aspx?Id= %[ID]</Uri>
      <ClickableUri>http://www.coveo.com</ClickableUri>
      <ContentType>text/html</ContentType>
      <Title>%[Company] (%[ID])</Title>
      <Body>%[Company]
          %[Job Title] %[First Name] %[Last Name]
          %[Business Phone]</Body>
      <CustomFields>
        <CustomField name="Type">Customer</CustomField>
        <CustomField name="ID">%[ID]</CustomField>
      </CustomFields>
    </Fields>
    <AllowedUsers>
      <AllowedUser type="Windows" allowed="true">
        <Name>everyone</Name>
```

```
        <Server></Server>
      </AllowedUser>
    </AllowedUsers>
  </Mapping>
</ODBC>
```

The following list presents the different attributes found in the sample configuration file:

<CommonMapping>

You can specify multiple settings common for all or several of the mappings in the configuration file. Unless specified otherwise, `<CustomFields>` and `<AllowedUsers>` nodes used for all tables can be added (see "<Mapping>" on page 8). The `excludedItems` attribute lists all the objects from the `ItemType` source parameter that does not use the settings specified in `CommonMapping`. The following represents the entire `CommonMapping` nodes where the table `Customers` is excluded.

Refer to the following example:

```
<CommonMapping excludedItems="Customers">
    <Fields>
      <CustomFields>
        <CustomField name="ID">%[ID]</CustomField>
      </CustomFields>
    </Fields>
    <AllowedUsers>
      <AllowedUser type="Windows" allowed="true">
        <Name>everyone</Name>
        <Server></Server>
      </AllowedUser>
    </AllowedUsers>
  </CommonMapping>
```

<Mapping>

Defines how the connector retrieves data from the tables, how the indexed data is stored, and who has access to it. It has an attribute called `type`. It is important to add the name of the table used for mapping. The mapping for the table `Orders` is displayed and two tables are indexed, however one contains the most complex configuration.

Refer to the following example:

Note: All node values can be defined using the following syntax: `%[odbcField]`. When the indexing process starts, this value is replaced by the actual value from the Database source. For example, in `<FileName>%[CustomerID].txt</FileName>`, the term `%[CustomerID]` is dynamic and replaced by the source value; however, `.txt` is static.

<Accessor>

The `<Accessor>` node contains the query string used to extract the data from the table. This node is mandatory in order to have a valid mapping. It defines the method used to access the specified table. There is an attribute called `type` that can either be equal to `object` or `query`. In the example above, the `query` type is used, meaning that the information stored in the database is accessed using a SQL query and can be stored in multiple tables and views.

Refer to the following example where you must replace `[PARAMETER]` by `@LastRefresh` in an SqlClient scenario or by `?` otherwise:

```
SELECT Shippers.Company AS ShipperName,
             Orders.[Order ID] AS ID,
             Orders.[Customer ID],
             Orders.[Order Date],
             Orders.[Shipped Date],
             Customers.Company,
             Employees.[Last Name],
             Employees.[First Name],
             Products.[Product Name],
             Products.[List Price]
      FROM   Orders, [Order Details], Shippers, Customers, Employees, Products
      WHERE  Orders.[Shipper ID] = Shippers.ID AND
             Orders.[Customer ID] = Customers.ID AND
             Orders.[Employee ID] = Employees.ID AND
             [Order Details].[Order ID] = Orders.[Order ID] AND
             [Order Details].[Product ID] = Products.ID AND
             Orders.[Order Date] >= [PARAMETER]
```

Note: You can use the paged query execution feature to return results in small batch number and decrease the likelihood of execution timeout (see "Enabling Paged Query Execution for the Database Connector" on page 12).

<Fields>

Collection of fields to be mapped when a document is indexed.

<Uri>

Address to which the user is redirected when clicking on the title of a search result. It is also used by the index to identify documents.

Refer to the following example:

```
<Uri>http://www.coveo.com/Orders/details.aspx?Id=%[ID]</Uri>
```

Note: Even though the indexed database records do not have an actual Web address where they can be viewed by users, you must provide one. It is possible to generate an URI based on any dummy address as long as it is unique. You can create one formed with a primary key provided from your database.

<ClickableUri>

URI opened when trying to open documents from the user interface of CES.

Refer to the following example:

```
<ClickableUri>http://www.coveo.com</ClickableUri>
```

<FileName>

Name of the file indexed. The extension of the filename is used to select the appropriate converter.

Refer to the following example:

```
<FileName>%[ID].txt</FileName>
```

<Title>

Name displayed on the search result page representing the title of the document indexed.

Refer to the following example:

```
<Title>Order ID: %[ID]: %[Product Name]</Title>
```

Note: `%[ID]` and `%[Product Name]` are used to give each entry its own name based on the ID and product name fields in the `Orders` table.

<Body>

Body of the document indexed. It can be a mix of static and dynamic content taken from the table.

Refer to the following example:

```
<Body>
        Customer: %[Company]
        OrderDate: %[Order Date]<br/>
        ShippedDate: %[Shipped Date]<br/>
        Shipped via: %[ShipperName]<br/>
        %[Product Name], $%[List Price]
      </Body>
```

```
<Body>OrderDate: %[OrderDate] \n RequiredDate: %[Required Date] \n ShippedDate: %[ShippedDate]
\nShipped via: %[ShipperName]
</Body>
```

The fields (`%[field_name]`) are all dynamic content types that change with each entry taken from the table.

Note: You cannot use both `Body` and `BinaryBody` nodes at the same time.

<CustomFields>

Mapping of ODBC fields to CES custom fields. It is made of a collection of `<CustomField>` nodes.

<CustomField>

Each `<CustomField>` node represents a custom field in CES and the data contained therein. In the example above, several `<CustomField>` nodes like the following are displayed:

```
<CustomField name="OrderDate">%[Order Date]</CustomField>
```

The `name` attribute is mandatory and represents the name of the CES custom field to bind data to. The value of this node should use the mapping syntax. The `%[Order Date]` expression instructs the connector to copy the information from the `OrderDate` database field to the `OrderDate` custom field in CES.

<AllowedUsers>

Mapping of ODBC fields to CES security. The connector does not index the permissions of the database automatically, therefore this field can be used to protect the data taken from the database.

Note: If you do not insert this section, everybody will have access to all the documents indexed.

<AllowedUser>

Rights given to users or groups regarding indexed documents. In the following example, there are two mandatory attributes:

Refer to the following example:

```
<AllowedUser type="CustomGroup" allowed="true">
```

- `Type`: It specifies the type of user to which rights are given (`Windows`, `CustomGroup` or `CustomUser`).
- `Allowed`: It can be set to `true` or `false`.

Note: You can define multiple allowed users within a single `AllowedUser` node by separating them by a semicolon (;).

<Name>

Name of a user or group to which you want to grant permissions. The `FirstName` and `LastName` values are used and stored in the table. This type of setup is useful only if these fields correspond to users that have access to the database.

Refer to the following example:

```
<Name>%[First Name] %[Last Name]</Name>
```

You should have one or more tables listing the users as well as their respective permissions. Also, provide the name of a `Windows` group that contains all the users that should have access. Otherwise, this field can be set to `Everyone`, meaning everyone has access to the information taken from the table.

<Server>

Domain name for the current group or user. In the above example, no server name is specified. However, in an everyday context, it needs to be set to a specific value.

4.2 Additional XML Attributes

This section presents additional XML attributes necessary to index data from a database. Basically, a single mandatory section must be included in a configuration file: the `Mapping` node, which must be placed under a parent ODBC node.

The following displays this basic structure:

```
<?xml version="1.0" encoding="utf-8" ?>
<ODBC>
  <CommonMapping>
  </CommonMapping>
  <Mapping>
  </Mapping>
</ODBC>
```

Note: The `CommonMapping` node is optional.

<CommonFields>

As in the `Mapping` section, it is a `Fields` node that is added to all mappings except the ones specified in the `ExcludedItems` list. This list is made of comma (,) separated values.

Refer to the following example:

```
<CommonFields excludedItems="ItemType">
</CommonFields>
```

<Mapping type=" ">

Each `Mapping` node must have a `type` attribute. This attribute represents the name of this mapping and is the value used to reference a mapping.

<BinaryBody>

This node is used to map a BLOB field from the database to the body of the document. Exceptionally, the value specified for this node must not use the standard syntax `%[odbcField]`. Instead, use the syntax `odbcField`. Once the mapping is resolved, an appropriate converter is called depending on the type of file inside the BLOB field of the database.

Note: You cannot use both `BinaryBody` and `Body` nodes at the same time.

<PrintableUri>

URI displayed in the user interface of CES.

<ContentType>

Type of data in the body of the document. Either this node or `<FileName>` must be specified, as it is used to find the appropriate converter. If you are not sure of the type of data being indexed, use the `binarydata` value and CES will find the right converter for your document data. The mapping syntax should be used to define the mapping of this field.

Note: For database records without an actual filename, do not specify the `<FileName>` node; however, do specify the `<ContentType>text/html</ContentType>` node.

<ModifiedDate>

Last modification date on a document. The value of this field must be a date, and the mapping syntax should be used to define the mapping of this field.

4.3 Enabling Paged Query Execution for the Database Connector

The amount of data to retrieve and the time it takes to execute a query can sometimes be a lengthy process. This is why the Database connector can execute queries in smaller subsets of data - called page result - allowing the retrieval of, for example, 5,000 rows per call. By doing so, timeouts can easily be avoided.

The paged query execution takes advantage of a feature found on most DBMS, which offers the possibility to retrieve the subset of query results by specifying the range of rows to crawl.

Note: For the initial release of the paged query execution feature, SQL Server and Oracle are the officially supported DBMS supporting this mode of operation.

To take advantage of the paged query execution feature, you must properly configure your mapping file. You have to provide a query that takes advantage of the paging feature and supply two tokens (`@startRow` and `@endRow`) that will dynamically be replaced at run time by the connector with the actual range of data to retrieve. When these tokens are detected, the connector automatically enables paging execution.

Refer to the following for an excerpt of a mapping file with a query configured to run in paged mode against SQL Server:

```
SELECT * FROM (
  SELECT MESSAGE.MID,
    MESSAGE.SENDER,
    MESSAGE.DATE_,
    MESSAGE.MESSAGE_ID,
    MESSAGE.SUBJECT,
    MESSAGE.BODY,
    MESSAGE.FOLDER,
    ROW_NUMBER() OVER (Order By MESSAGE.MID) as LINE
  FROM MESSAGE
  WHERE MESSAGE.DATE_ like '2001-04-07%') RESULTS
WHERE RESULTS.LINE between @startRow and @endRow
```

All queries in a configuration file are independent. You can decide to use the paged query execution feature or not.

Example: A query can be written as a paged query, while its associated subqueries can be written without this feature.

Hidden parameter for page size (Integer)

By default, the connector is configured to query the database 5,000 records at a time when the paged query execution feature is enabled. However, this value can be overridden by configuring the `QueryPageSize` hidden parameter in the source configuration. The value should be positive and different from 0.

To configure the QueryPageSize hidden parameter

1. On the Coveo server, access the Administration Tool.
2. Add the `QueryPageSize` hidden parameter to the database connector for all database sources by specifying an `Integer` type and `5000` for the default value (see "Adding an Explicit Connector Parameter" on page 25).
3. Select the **Index** tab, and then select the **Sources and Collections** menu.
4. For each database source for which you want to change the default value for the `QueryPageSize` parameter:
    a. Under **Collections**, select the collection containing the database source.
    b. Under **Sources**, select the desired database source.
    c. In the navigation panel on the left, select **General**.
    d. In the **Query Page Size** box that now appears in the page, enter the desired page size value for this database source. The value should be positive and different from 0.
    e. Click **Apply Changes**.

5. CES Configuration for the Database Connector

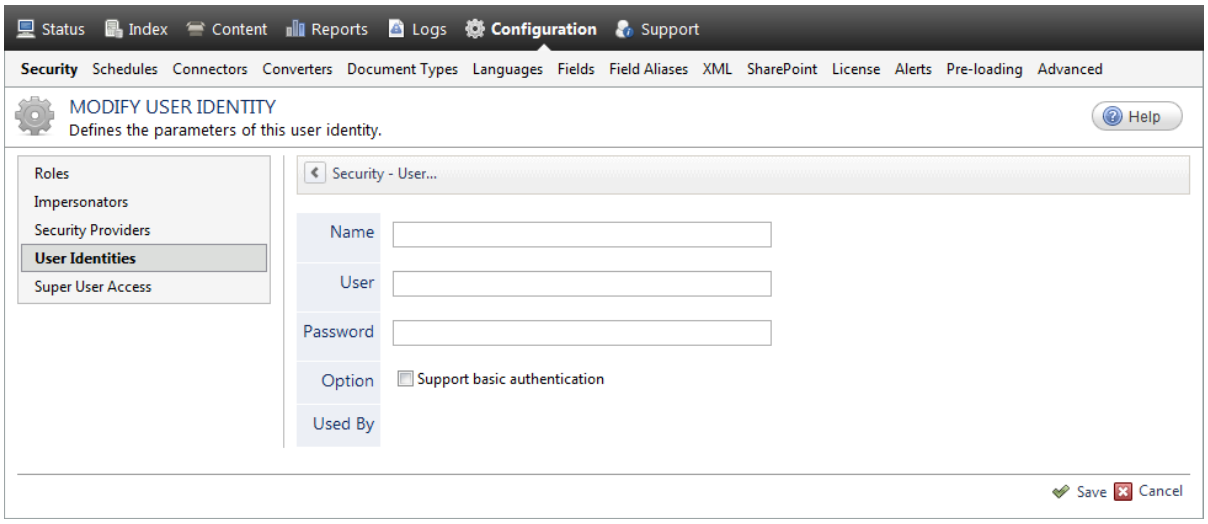
5.1 Adding a User Identity

A user identity is a set of credentials for a given repository or system that you enter once in CES and can then associate with one or more sources or security providers.

A user identity typically holds the credentials of an account that has read access to all the repository items that you want to index. It is a best practice to create an account to be used exclusively by the Coveo processes and for which the password does not change. If the password of this account changes in the repository, you must also change it in the CES user identity.

To add a user identity

1. On the Coveo server, access the Administration Tool.
2. In the Administration Tool, select **Configuration** > **Security**.
3. In the navigation panel on the left, click **User Identities**.
4. In the **User Identities** page, click **Add**.
5. In the **Modify User Identity** page:



   a. In the **Name** box, enter a name of your choice to describe the account that you selected or created in the repository to allow CES to access the repository.

   Note: This name appears only in the Coveo Administration Tool, in the **Authentication** or **User Identity** drop-down lists, when you respectively define a source or a security provider.

   b. In the **User** box, enter the username for the account that you selected or created to crawl the repository content that you want to index.

c. In the **Password** box, enter the password for the account.

d. In the **Options** section, the **Support basic authentication** check box is deprecated and not applicable for most types of repositories. You should select it only when you need to allow CES to send the username and password as unencrypted text.

e. Click **Save**.

Important: When you use Firefox to access the Administration Tool and it proposes to remember the password for the user identity that you just created, select to never remember the password for this site to prevent issues with automatic filling of username and password fields within the Coveo Administration Tool.

5.2 Configuring a Database Security Provider

When your database contains permission information and user group definitions, you can use a security provider to expand the groups and index the permissions stored in the database. Otherwise, you do not need a security provider.

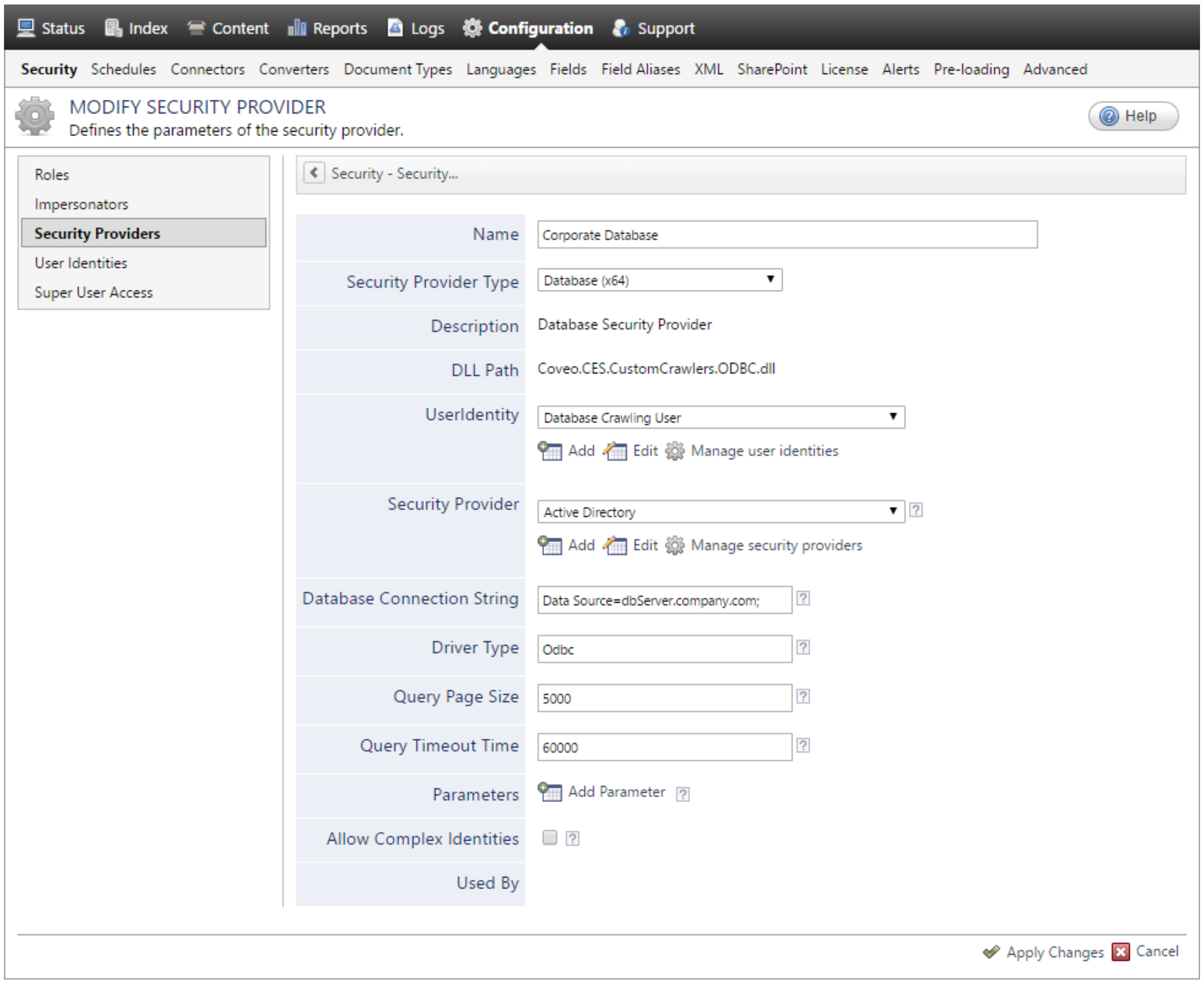
Note: You can get familiar with how Coveo components deal with permissions on documents both at indexing and query time.

To create or modify a database security provider

1. On the Coveo server, access the Administration Tool.
2. Select **Configuration** > **Security**.
3. In the navigation panel on the left, select **Security Providers**.
4. In the **Security Providers** page:
    - Click **Add** to create a new security provider.

      OR

    - Click an existing security provider to modify it.
5. In the **Modify Security Provider** page:



MODIFY SECURITY PROVIDER

Defines the parameters of the security provider.

Name	Corporate Database
Security Provider Type	Database (x64)
Description	Database Security Provider
DLL Path	Coveo.CES.CustomCrawlers.ODBC.dll
UserIdentity	Database Crawling User
Security Provider	Active Directory
Database Connection String	Data Source=dbServer.company.com;
Driver Type	Odbc
Query Page Size	5000
Query Timeout Time	60000
Parameters	
Allow Complex Identities	☐
Used By	

a. In the **Name** box, enter a name to identify this security provider.

b. In the **Security Provider Type** drop-down list:

   i. On a 32-bit server, select **Database (x86)**.

   ii. On a 64-bit server, select **Database (x64)**, or when the database driver used is a 32-bit process, select **Database (x86)**.

c. In the **User Identity** section, select a user identity only when you want to hide the account credentials in the **Database Connection String**:

   i. In the drop-down list, select the user identity that you selected or created to crawl your databases.

   ii. When this is not already done, click **Add**, **Edit**, or **Manage user identities** respectively to create, modify, or manage user identities.

d. In the **Security Provider** drop-down list:

> **Note:** CES 7.0.8691– (December 2016) The parameter was labeled **Active Directory Security Provider**.

   i. Select the security provider that you selected or created to allow this security provider to resolve and expand the groups (see Database Connector Deployment Overview).

   ii. When an appropriate security provider is missing, click **Add**, **Edit**, or **Manage security providers** respectively to create, modify, or manage security providers.

e. In the **Database Connection String** box, enter the connection string used to connect to the database. The connection string syntax differs from one database type to another. Refer to the appropriate documentation for the format of the connection string specific to your database (see www.connectionstrings.com).

   Note: When you assign a user identity to the security provider, you can hide the password and the user ID by replacing them with tokens in the connection string (see "Replacing the Identity in Database Connection Strings" on page 30).

f. In the **Drive Type** box, enter the type of the driver used to connect to the database:

   - Enter `Odbc` for Open Database Connectivity.
   - Enter `OleDb` for Object Linking and Embedding, Database.
   - Enter `SqlClient` for an SQL client.

g. In the **Query Page Size** box, enter the desired page size value for an ODBC query when executing in paged mode. The value should be positive and different from 0. The default value is `5000`.

h. In the **Query Timeout Time** box, enter the maximum time (in milliseconds) allowed to perform a query on the database. The default value is `60000` ms (60 seconds).

i. In the **Parameters** section, in rare cases, the Coveo Support could instruct you to click **Add Parameters** to specify other security provider parameter names and values that could help to troubleshoot security provider issues.

j. Leave the **Allow Complex Identities** option cleared as it does not apply to this type of security provider.

k. Click **Save** or **Apply Changes**, depending whether you are creating or modifying a security provider.

What's Next?

Create an index a source ("Configuring and Indexing a Database Source" on page 17).

5.3 Configuring and Indexing a Database Source

A source defines a set of configuration parameters for the databases indexed, including all the information required to access and authenticate.

To configure and index a database source

1. On the Coveo server, access the Administration Tool.
2. Select **Index** > **Sources and Collections**.
3. In the **Collections** section:

a. Select an existing collection in which to add the new source.

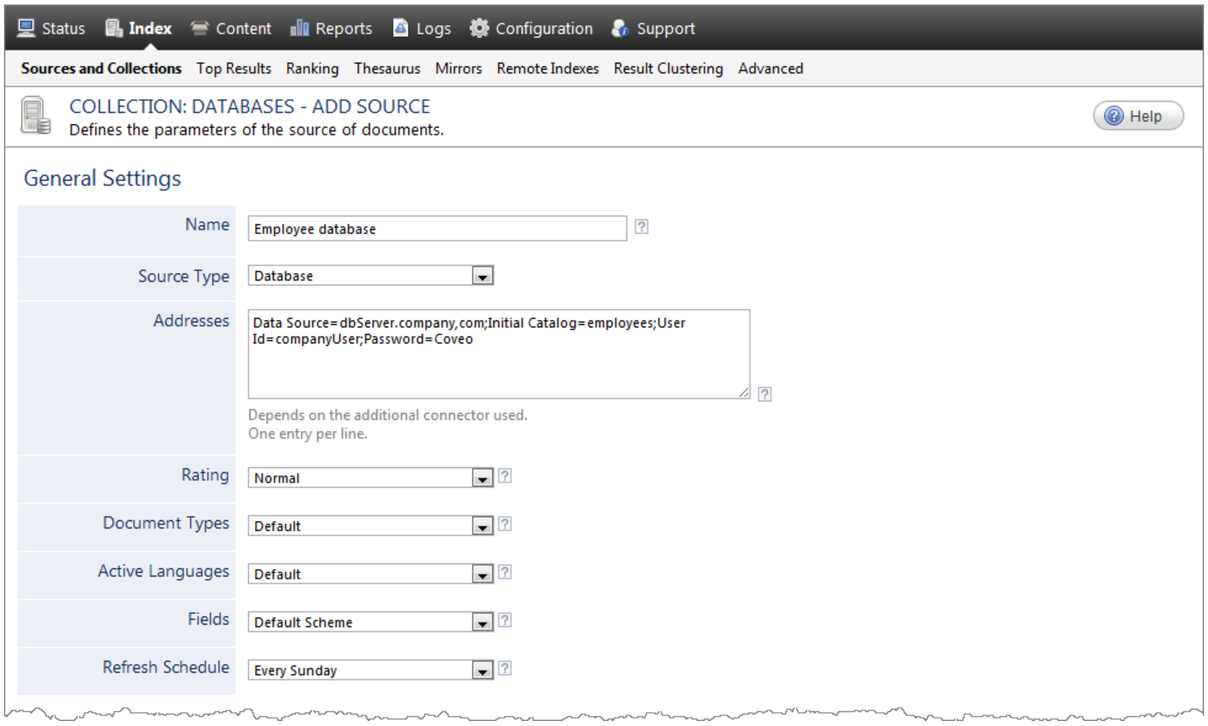
OR

b. Click **Add** to create a new collection.

4. In the **Sources** section, click **Add** to create a new database source.

The **Add Source** page that appears is organized in three sections.

5. In the **General Settings** section of the **Add Source** page:



COLLECTION: DATABASES - ADD SOURCE
Defines the parameters of the source of documents.

General Settings

Name	Employee database
Source Type	Database
Addresses	Data Source=dbServer.company,com;Initial Catalog=employees;User Id=companyUser;Password=Coveo Depends on the additional connector used. One entry per line.
Rating	Normal
Document Types	Default
Active Languages	Default
Fields	Default Scheme
Refresh Schedule	Every Sunday

a. Enter the appropriate value for the following required parameters:

Name

Enter a descriptive name of your choice for the connector source.

Example: `Employee database`

Source Type

Select the connector used by this source. In this case, select **Database**.

Note: If you do not see **Database**, your environment does not meet the requirements (see "Database Connector Requirements" on page 5).

Addresses

Enter the connection string used to connect to the database. The connection string syntax differs from one database type to another. Refer to the appropriate documentation for the format of the connection string specific to your database (see www.ConnectionStrings.com).

The same connection string can be used for different sources. However, there can only be one connection string per source.

Example: `Data Source=dbServer.company.com;Initial Catalog=employees;User Id=companyUser;Password=MyPassword`

You can hide the password and the user ID in the connection string (see "Replacing the Identity in Database Connection Strings" on page 30).

Note: When the ODBC connection string includes SSPI security, CES uses the CES Service logon account to connect to the database. Ensure that the CES service of your Coveo instance uses a domain account that can crawl the database, not a local system account.

Refresh Schedule

Time interval at which the source is automatically refreshed to keep the index content up-to-date. By default, the recommended **Every day** option instructs CES to refresh the source everyday at 12 AM.

Note: You can create new or modify existing source refresh schedules.

b. Review the value for the following parameters that often do not need to be modified:

Rating

Change this value only when you want to globally change the rating associated with all items in this source relative to the rating of elements in other sources.

Document Types

If you created a custom document type set for this source, select it. Otherwise, select **Default**.

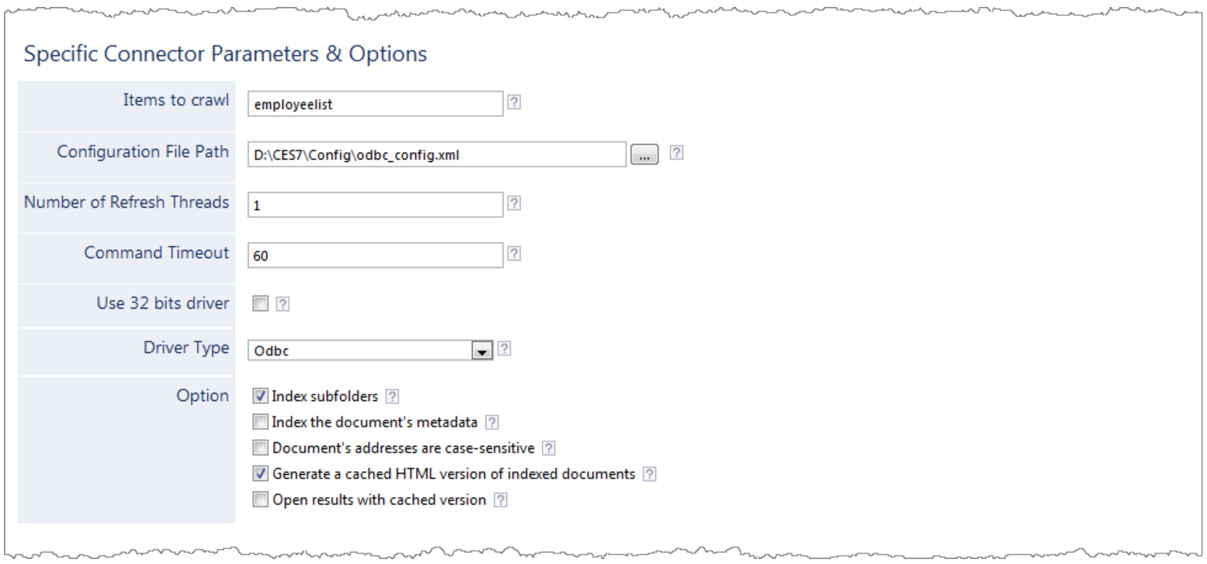
Active Languages

If you defined custom active language sets, ensure to select the most appropriate for this source.

Fields

If you defined custom field sets, ensure to select the most appropriate for this source.

6. In the **Specific Connector Parameters & Options** section of the **Add Source** page:



Specific Connector Parameters & Options

Items to crawl	employeelist
Configuration File Path	D:\CES7\Config\odbc_config.xml
Number of Refresh Threads	1
Command Timeout	60
Use 32 bits driver	☐
Driver Type	Odbc
Option	☑ Index subfolders ☐ Index the document's metadata ☐ Document's addresses are case-sensitive ☑ Generate a cached HTML version of indexed documents ☐ Open results with cached version

a. Enter the appropriate value for the following required parameters:

Items to crawl

Enter a comma-separated (,) list of table or view object names to crawl as they are defined in the configuration file. You can use this parameter to easily crawl a subset of objects defined in the configuration file rather than commenting out objects in the configuration file.

Configuration File Path

Enter the full path to the configuration file that you created to instruct the connector what to index (see "Indexing a Database Using a Configuration File" on page 6).

Example: `C:\CES7\Config\odbc_config.xml`

b. Review the value for the following parameters that often do not need to be modified:

Number of Refresh Threads

Determines the number of simultaneous downloads handled by the connector.

Command Timeout

Maximum time allowed to perform a query on the database. The default value is 60 seconds. Once elapsed, the query times out.

Note: You can use the paged query execution feature to decrease the execution time (see "Enabling Paged Query Execution for the Database Connector" on page 12).

Use 32 bits driver

On a 64-bit server, select this check box when you use a 32-bit driver to connect to the database.

Note: Selecting the **Use 32 bits driver** option may resolve issues causing the `Arithmetic operation resulted in an overflow` error that appears to be linked to the 64-bit driver inability to convert a database field. However, updating to the ODBC driver 5.3+ also can resolve the problem.

Driver Type

In the drop-down box, select the software driver that provides access to your database:

- Select **Odbc** when using Open Database Connectivity.
- Select **OleDb** when using Object Linking and Embedding, Database.
- Select **SqlClient** when using an SQL client.

In the **Option** section:

Index Subfolders

This parameter does not apply to a database source.

Index the document's metadata

When selected, CES indexes all the document metadata, even metadata that are not associated with a field. The orphan metadata are added to the body of the document so that they can be searched using free text queries.

When cleared (default), only the values of system and custom fields that have the **Free Text Queries** attribute selected will be searchable without using a field query.

Example: A document has two metadata:

- `LastEditedBy` containing the value `Hector Smith`
- `Department` containing the value `RH`

In CES, the custom field `CorpDepartment` is bound to the metadata `Department` and its **Free Text Queries** attribute is selected.

When the **Index the document's metadata** option is cleared, searching for `RH` returns the document because a field is indexing this value. Searching for `hector` does not return the document because no field is indexing this value.

When the **Index the document's metadata** option is selected, searching for `hector` also returns the document because CES indexed orphan metadata.

Document's addresses are case-sensitive

Leave this check box cleared. This parameter needs to be checked only in rare cases for systems in which distinct documents may have the same name but different casing.

Generate a cached HTML version of indexed documents

Keep this check box selected. When indexing, CES creates HTML versions of indexed documents. In the search interfaces, users can review the content more rapidly by clicking the **Quick View** link.

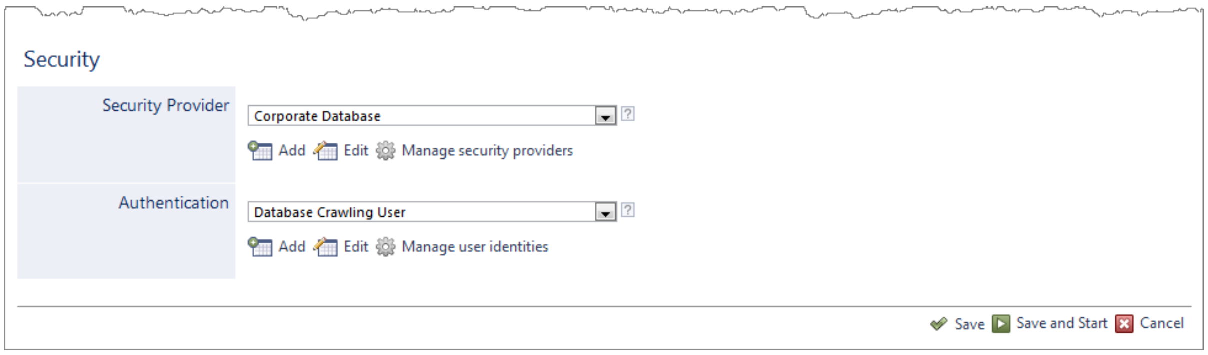
Consider clearing this check box only when you do not want to use **Quick View** links or save resources when building the source.

Open results with cached version

Select this check box to view the cached HTML version of the database content when the end-user clicks the main search result link. In this case, you must also select **Generate a cached HTML version of indexed documents**.

Clear this check box only when you defined a clickable URI in the configuration file to open the database content in a specific application. do not want users to be able to open the original document but only see the HTML version of the document as a Quick View.

7. In the **Security** section of the **Add Source** page:



Security

Security Provider: Corporate Database

Authentication: Database Crawling User

   a. In the **Security Provider** drop-down list, when you chose to use a security provider, select the security provider that you created for this source (see "Configuring a Database Security Provider" on page 15).

   b. In the **Authentication** drop-down list, when you chose to hide the database account credentials in the database connection string, select the user identity that you created for this source (see "Adding a User Identity" on page 14).

   c. Click **Save and Start** to save the source configuration and start indexing this source.

8. Validate that the source building process is executed without errors:

   - In the navigation panel on the left, click **Status**, and then validate that the indexing proceeds without errors.

     OR

   - Open the CES Console to monitor the source building activities.

5.4 Creating a View in a Database

You can exploit the principle of embedded views (queries) in a database, therefore not having to rewrite long, complex queries in a configuration file.

Note: If you have to use a configuration file, your query expression can be used to filter out objects you do not want from an existing View.

To create a view in a database

1. Log on to your database with a database editor/client.

   Note: Make sure you have the appropriate read/write rights or administrator privileges for the database.

2. On the command line of the client, create your view. If you are using an access-like database editor, create a view only using the `SELECT` and `FROM` parts of the queries provided below.

   - CES_Orders

     Security permissions for documents should be defined using one of the methods explained in "Indexing a Database Using a View" on page 24.

```
CREATE view "CES_Orders" AS
SELECT
'http://www.coveo.com/Orders/details.aspx?Id=' + convert(varchar, Orders.OrderID) AS Uri,
'http://www.coveo.com' AS ClickableUri,
Orders.OrderID AS Title,
'OrderDate: ' + convert(varchar, OrderDate) + CHAR(10) + 'RequiredDate: ' + convert(varchar,
RequiredDate) + CHAR(10) + 'ShippedDate: ' + convert(varchar, ShippedDate) + CHAR(10) +
'Shipped via: ' + Shippers.CompanyName AS Body,
'Order' AS Type,
OrderDate,
RequiredDate,
ShippedDate,
Shippers.CompanyName AS Shipper,
FirstName + ' ' + LastName AS sysAuthor
FROM
 Orders
INNER JOIN Shippers ON Orders.ShipVia = Shippers.ShipperID
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID

CES_Products
```

   - CES_Products

     Permissions are defined directly as fields inside the view (allowed for all documents is `coveo/hford`).

```
CREATE view "CES_Products" AS
SELECT
'http://www.coveo.com/Products/details.aspx?Id=' + convert(varchar, Products.ProductID) AS
Uri,
 'http://www.coveo.com' AS ClickableUri,
ProductName + ' (' + convert(varchar, Products.ProductID )+ ')' AS Title,
'Name: ' + ProductName + CHAR(10) + 'Category: ' + CategoryName + CHAR(10) + 'Supplier: ' +
Suppliers.CompanyName AS Body,
'Product' AS Type,
ProductName AS Product,
Discontinued,
UnitPrice,
QuantityPerUnit,
CategoryName AS Category,
Suppliers.CompanyName AS Supplier
'hford' AS AllowedWindowsName,
'coveo' AS AllowedWindowsServer
FROM
Categories
INNER JOIN Products ON Categories.CategoryID = Products.CategoryID
INNER JOIN Suppliers ON Products.SupplierID = Suppliers.SupplierID
```

5.5 Indexing a Database Using a View

You can index data from a database using embedded views (queries), which are defined directly in your database.

To use embedded views

1. Create a database source (see "Configuring and Indexing a Database Source" on page 17).
2. In your database, create a view that returns at least the `URI` field (see "Creating a View in a Database" on page 22).

   Note: The `URI` field value must be a valid URI, meaning it must have any scheme prefix followed by a unique identifier (ex.: `odbc://id=4`).

   The fields that can be mapped to a document are:

   - Uri
   - UriCaption
   - ClickableUri
   - PrintableUri
   - Title
   - FileName
   - ContentType
   - BinaryBody
   - Body
   - ModifiedDate
   - All security-related fields.
3. Define the required security permissions on your documents using one of the following methods:
   - In the Administration Tool:

     Note: No configuration file is required when using this method.

     a. On the Coveo server, access the Administration Tool.
     b. Access the **Sources and Collections** page (**Index** > **Sources and Collection**).
     c. In the **Source** section, expand the drop-down list of the appropriate source.
     d. Select **Edit Permissions**. The **Permissions** page of the selected source is displayed.
     e. Select **Specify the security permissions to index**.
     f. Specify the necessary users in the **Allowed Users** and **Denied Users** lists.

Note: To add a new user, enter the username in the corresponding box and click **Add**. By default the **Allowed Users** list contains `everybody` that has access to the indexed documents.

g. Click **Apply Changes**.

- From the created view:

Note: No configuration file is required when using this method.

a. In the created view, add new fields containing the required permissions for every document.

Security type	User type	Required field names	
		User-related	Server-related
Windows	Allowed users	AllowedWindowsName	AllowedWindowsServer
	Denied users	DeniedWindowsName	DeniedWindowsServer
Custom group	Allowed users	AllowedCGName	AllowedCGServer
	Denied users	DeniedCGName	DeniedCGServer
Custom user	Allowed users	AllowedCUName	AllowedCUServer
	Denied users	DeniedCUName	DeniedCUServer

b. On the Coveo server, access the Administration Tool.

c. In the **Source** section, expand the drop-down list of the appropriate source.

d. Select **Edit Permissions**. The **Permissions** page corresponding to the source is displayed.

e. In the **Permissions** section, select **Index security permissions**.

- From a configuration file:

a. Create a configuration file (see "Example of a Configuration File for the Database Connector" on page 6 and "Indexing a Database Using a Configuration File" on page 6).

b. Add an `AllowedUsers` node in which it is possible to define allowed and denied users for every document.

Note: Creating a configuration file implies that query expressions have to select the entire view or the required fields.

5.6 Adding an Explicit Connector Parameter

Connector parameters applying to all sources indexed using this connector are called explicit parameters.

When you create or configure a source, the Coveo Enterprise Search (CES) 7.0 Administration Tool presents parameters with which you can configure the connector for most setups. For many connectors, more advanced and more rarely used parameters also exist but are hidden by default. CES then uses the default value associated with each of these hidden parameters.

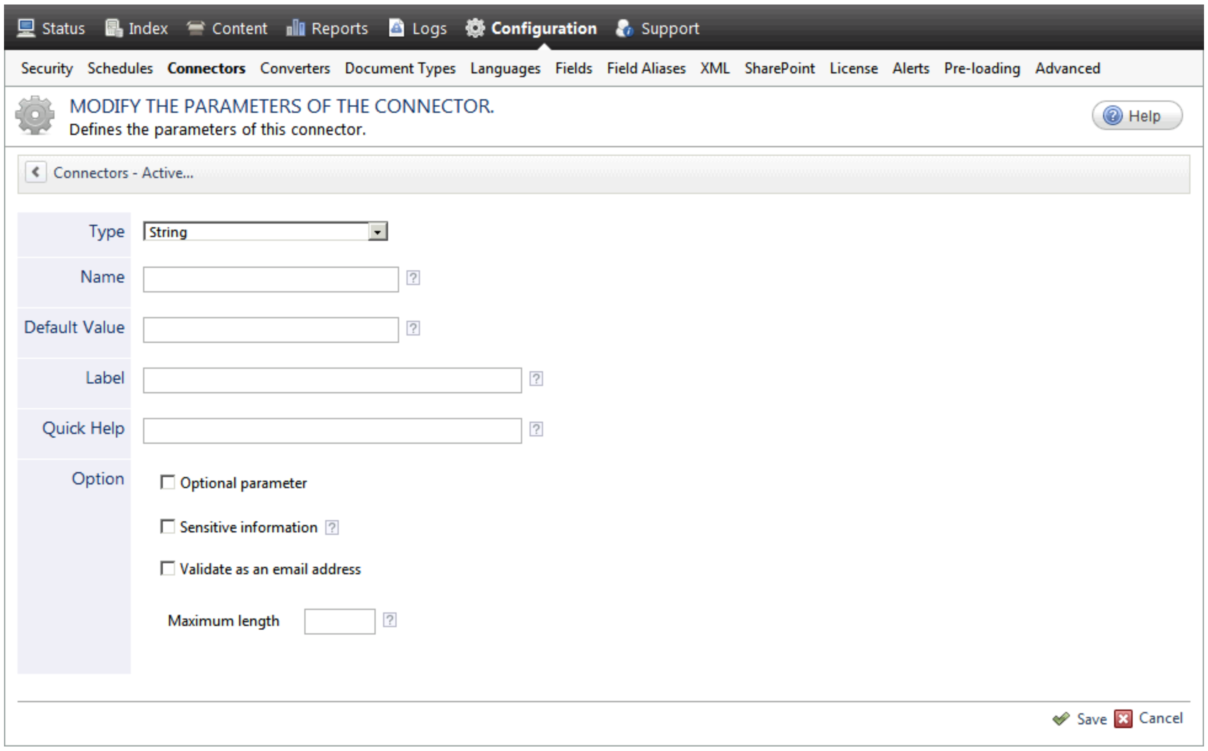
You can however choose to make one or more of these parameters appear in the **Add Source** and **Source: ... General** pages of the Administration Tool so that you can change their default value.

To add an explicit connector parameter

1. On the Coveo server, access the Administration Tool.
2. Select **Configuration** > **Connectors**.
3. In the list on the **Connectors** page, select the connector for which you want to show advanced hidden parameters.
4. In the **Parameters** section of the selected connector page, click **Add Parameter** for each hidden parameter that you want to modify.

   Note: The **Add Parameter** button is present only when hidden parameters are available for the selected connector.

5. In the **Modify the parameters of the connector** page:



MODIFY THE PARAMETERS OF THE CONNECTOR.
Defines the parameters of this connector.

Type	String
Name	
Default Value	
Label	
Quick Help	
Option	☐ Optional parameter ☐ Sensitive information ☐ Validate as an email address Maximum length

   a. In the **Type** list, select the parameter type as specified in the parameter description.

   b. In the **Name** box, type the parameter name exactly as it appears in the parameter description. Parameter names are case sensitive.

c. In the **Default Value** box, enter the default value specified in the parameter description.

Important: Do not set the value that you want to use for a specific source. The value that you enter here will be used for all sources defined using this connector so it must be set to the recommended default value. You will be able to change the value for each source later, in the **Add Source** and **Source: ... General** pages of the Administration Tool.

d. In the **Label** box, enter the label that you want to see for this parameter.

Example: To easily link the label to the hidden parameter, you can simply use the parameter name, and if applicable, insert spaces between concatenated words. For the **BatchSize** hidden parameter, enter `Batch Size` for the label.

Note: To create multilingual labels and quick help messages, use the following syntax: `<@ln>text</@>,` where *ln* is replaced by the language initials—the languages of the Administration Tool are English (en) and French (fr).

Example: `<@fr>Chemin d'accès du fichier de configuration</@><@en>Configuration File Path</@>` is a label which is displayed differently in the French and English versions of the Administration Tool.

Tip: The language of the Administration Tool can be modified by pressing the following key combination: `Ctrl+Alt+Page Up`.

e. Optionally, in **Quick Help**, enter the help text that you want to see for this parameter when clicking the question mark button ? that will appear beside the parameter value.

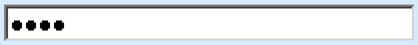
Tip: Copy and paste key elements of the parameter description.

f. When **Predefined values** is selected in the **Type** parameter, in the **Value** box that appears, enter the parameter values that you want to see available in the drop-down parameter that will appear in the Administration Tool interface. Enter one value per line. The entered values must exactly match the values listed in the hidden parameter description.

g. Select the **Optional parameter** check box when you want to identify this parameter as an optional parameter. When cleared, CES does not allow you to save changes when the parameter is empty. This parameter does not appear for **Boolean** and **Predefined values** parameter types.

h. Select the **Sensitive information** check box for password or other sensitive parameter so that, in the Administration Tool pages where the parameter appears, the typed characters appear as dots to mask them. This parameter appears only for the **String** type.

Example: When you select the **Sensitive information** check box for a parameter, the characters typed appear as follows in the text box:


●●●●

i. Select the **Validate as an email address** check box when you want CES to validate that the text string that a user enters in this parameter respects the format of a valid email address. This parameter appears only for the **String** type.

j. In the **Maximum length** box, enter the maximum number of characters for the string. This parameter appears only for the **String** type. When you enter `0`, the length of the string is not limited.

k. Click **Save**.

6. Back in the **Connector** page, click **Apply Changes**.

   The hidden parameter now appears in the **Add Source** and **Source: ... General** pages of the Administration Tool for the selected source. You can change the parameter value from these pages. Refer to the documentation for each connector for details.

Note: When you want to modify a hidden source parameter, you must first delete it, and then redefine it with the modified values.

6. Enabling Multithreading for the Database Connector

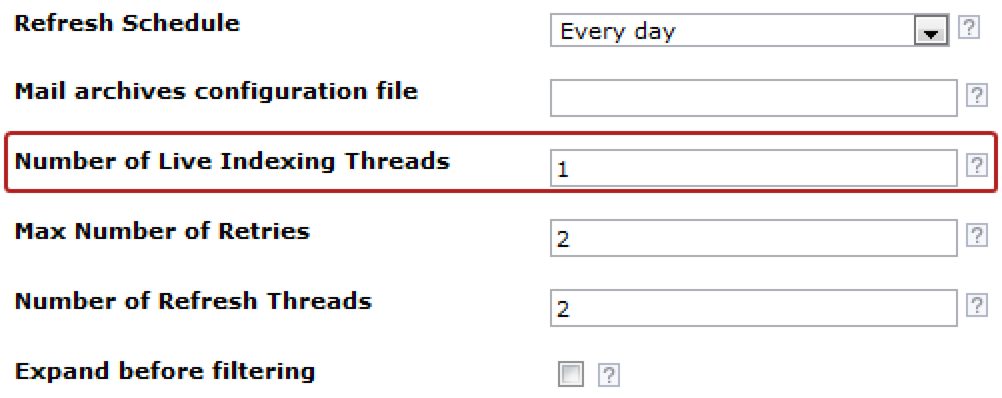
The Database connector supports multithreading. This feature can considerably improve performances with subqueries.

To enable multithreading

Specify the number of crawling threads using one of the following methods:

- When configuring a new source

  In the Coveo Administration Tool, specify the number of threads in the **Add Sources** page:



Refresh Schedule	Every day
Mail archives configuration file	
Number of Live Indexing Threads	1
Max Number of Retries	2
Number of Refresh Threads	2
Expand before filtering	☐

Note: The peak of performance is between 4 and 6 threads; therefore, setting the value to 200 will have no time benefit.

- Manually (for testing purpose)

  When you define the configuration values, add the number of threads as a parameter.

```
m_ConfigValues = new Dictionary<string, string>()
{
 {"ItemType", p_ItemType},
 {"ConfigFile", m_ConfigPath},
 {"DriverType", p_DriverType},
 {"NbRefreshThreads", "1"}
};
```

Note: To enable multithreading, `NbRefreshThreads` must be written as is.

7. Replacing the Identity in Database Connection Strings

In CES, you can create a user identity (see "Adding a User Identity" on page 14) and assign this user identity as well as the connection string to a database source (see "Configuring and Indexing a Database Source" on page 17). When this is done, you can hide the password and user ID in the connection string by respectively introducing the `@uid` and `@pwd` tokens. The Database connector internally replaces the tokens with the information provided in the user identity.

Examples:
For a basic connection string:

```
Data Source=dbServer.company.com;Initial Catalog=employees;User Id=companyUser;Password=MyPassword
```

Hiding password and user ID using tokens:

```
Data Source=dbServer.company.com;Initial Catalog=employees;User Id=@uid;Password=@pwd
```

Important: You must provide either both tokens or none at all. If you do not provide tokens, but add a user identity, the behavior will be the same as before. The user identity will be used to impersonate the process running the queries.

8. Enabling Incremental Refresh on a Database Source

Incremental refresh keeps documents up-to-date by scanning repositories and re-indexing modified documents at short intervals. To enable this feature on a database source, database items that have to be indexed, such as tables or views, must contain a `Date type` field to indicate their latest modification date. Fields can be given any name as long as it has the right data type. Furthermore, whenever a record is modified, it is important to update the latest modification date field (see "Configuring and Indexing a Database Source" on page 17 and "Additional XML Attributes" on page 11).

Note: The incremental refresh does not take into account deleted documents. A source full refresh or rebuild is required.

In the SQL query, the `SELECT` statement must have a `WHERE` clause with a criterion on the last modification date field.

Example: The following simple example should work with common database engines such as Microsoft SQL Server 2012, PostgreSQL, and MySQL. You must replace `[PARAMETER]` by `@LastRefresh` in an SqlClient scenario (MSSQL/SQLServer databases) or by `?` otherwise (e.g., NorthWind databases). The `[PARAMETER]` field is sent by the crawler to the query to indicate when the last increment refresh was performed.

```
<Accessor type="query"
  OrderByFieldName="dateCreated"
  OrderByFieldType="DateTime"
  IncrementalRefreshFieldName="dateModified">
  <![CDATA[
    Select
    id,
    title,
    dateModified,
    content,
    author
    FROM blog
    WHERE dateModified>=[PARAMETER]
    order by dateModified
    OFFSET @startRow ROWS FETCH NEXT (@endRow-@startRow) ROWS ONLY;
  ]]>
</Accessor>
```

The example also includes support for pagination (see OFFSET FETCH Clause (SQL Server Compact)).

What's Next?

Ensure that you also created an incremental refresh schedule on your database source in CES.

9. Enabling Pause/Resume on a Database Source

The Database connector can support the `Pause/Resume` parameter if the following is added to your configuration file:

1. In the SQL query, add an `ORDER BY` on the chronological fields of the `SELECT` statement:

   If the source supports Incremental Refresh

   The same chronological fields must also be used in the `ORDER BY`.

   If the source does not support Incremental Refresh

   - Field that uniquely identifies each record:

     If the primary key contains only one field, select that field. However, if there is a column that contains a sequential number incremented by the DBMS each time a new record is added, select that column instead.

   - No field that uniquely identifies each record:

     In this case, any field can be used. However, during a resume, records that have been indexed before pausing could be re-indexed.

     Example: You have a table containing the 12 months of the year and you select the month column as the reference. The values for this column are not unique, as many rows can be associated to the same month. While crawling the 6th month, you pause, therefore not indexing all the rows for the 6th month. When you resume, the connector will start crawling at the first row of the 6th month, re-indexing the rows that have already been indexed.

2. Add XML attributes on the `Accessor` element in the configuration file:

   OrderByFieldName

   Specifies the name of the column on which the `ORDER BY` is applied. This attribute must be present to enable Pause/Resume.

   OrderByFieldType

   Specifies the .NET data type of that column. This attribute is not normally required. The connector automatically tries to determine the data type by preparing the SQL query - without however executing it - and looking at the schema of the results. However, if a specific DBMS does not handle that process correctly, you can manually specify the data type with this attribute. The following lists the allowed types:

   - short (16-bit signed integer)
   - ushort (16-bit unsigned integer)
   - int (32-bit signed integer)
   - uint (32-bit unsigned integer)

- long (64-bit signed integer)
- ulong (64-bit unsigned integer)
- float (single-precision floating point number)
- double (double-precision floating point number)
- string (String)

IncrementalRefreshFieldName

To support both Incremental Refresh and Pause/Resume, `IncrementalRefreshFieldName` must contain the same name as `OrderByFieldName`.

Refer to this example for an excerpt of a configuration file for a source with Pause/Resume and Incremental Refresh enabled:

Note: You must replace `[PARAMETER]` by `@LastRefresh` in an SqlClient scenario or by `?` otherwise.

```
<Mapping type="Orders">
    <Accessor type="query"
              OrderByFieldName="OrderDate"
              OrderByFieldType="DateTime"
              IncrementalRefreshFieldName="OrderDate">
      SELECT Shippers.CompanyName AS ShipperName,
             Orders.OrderID AS ID,
             Orders.CustomerID,
             Orders.OrderDate,
             Orders.RequiredDate,
             Orders.ShippedDate,
             Customers.CompanyName,
             Employees.LastName,
             Employees.FirstName
      FROM   Orders, Shippers, Customers, Employees
      WHERE  Orders.ShipVia = Shippers.ShipperID AND
             Orders.CustomerID = Customers.CustomerID AND
             Orders.EmployeeID = Employees.EmployeeID AND
             Orders.OrderDate >= [PARAMETER]
      ORDER BY Orders.OrderDate
    </Accessor>
</Mapping>
```

Note: For backward compatibility, the equivalent `LiveIndexingFieldName` parameter from previous CES versions is still supported in CES 7.

10. Enabling a Query-Based Security Provider for the Database Connector

You can expand external groups and users using a security provider when the database contains information allowing groups to be expanded to external users and/or external users to be mapped to Active Directory users.

To expand external groups and users using a security provider

1. In the source XML mapping file, define `<AllowedUser>` entries with the following properties:

    - `Type` (mandatory): `ExternalGroup` or `ExternalUser`.
    - `Name` (mandatory): The name of the external group or user.
        - Supports multiple semicolon (;) separated names.
        - Supports the `%[column]` syntax to use values returned by the `Accessor` query.
    - `Server/ExpandGroup` (mandatory with the `ExternalGroup` type): SQL query used by the security provider to expand external groups to external users.
    - `Server/ExpandUser`: SQL query used by the security provider to map external users to Active Directory users.

```
<AllowedUsers>
  <AllowedUser type="ExternalGroup" allowed="true">
    <Name>%[column_allowed_groups]</Name>
    <Server>
      <ExpandGroup>
           select distinct column_user from membership where column_group = '@GroupName'
      </ExpandGroup>
      <ExpandUser>
           select distinct column_user_nt from nt_account where column_user = '@UserName'
      </ExpandUser>
    </Server>
  </AllowedUser>
</AllowedUsers>
```

2. Create a security provider for your database source (see "Configuring a Database Security Provider" on page 15).

    Match the values for the Security provider and Source parameters listed in the following table.

Security provider parameter	Source parameter
Driver Type	**Driver Type**
Database Connection String	**Addresses**

3. Associate this new security provider to your database source by selecting it in the **Security Provider** drop-down list (see Configuring and Indexing a Database Source).

11. Complement Information Retrieval with Subqueries for the Database Connector

The Database connector acquires information about each indexed document through a query performed against a database. For each query, it is possible to associate one or more subqueries to be executed and used to complement information.

Example: You can run a main query, and for each row, run a subquery that crawls more/different information. All the results of a single row from the main query, along with everything from the subquery, are merged into a single document.

The connector requires a mapping file to execute properly. For each mapping type, it is necessary to specify an `Accessor` representing the SQL query to execute.

11.1 Specifying Subqueries

To specify subqueries, you must set the type of the `Accessor` to `query`.

```
<Accessor type="query">
```

Following the `Accessor` definition, add an `AccessorSubQueries` node with all subqueries:

Note: The master key (value following SELECT) in the AccessorSubQuery node must match exactly the one returned by the server. The key you include must also have the same casing. The following error is thrown when the key could not be found:

```
Unable to index document : There is a formatting error in a sub query. Cannot find
master key %[key].
```

```
<AccessorSubQueries>
  <AccessorSubQuery name="FirstNameLastName" separator=";" behaviorOnMultiRows="join" allowDuplicates
= "false">
    SELECT  firstName,
    lastName
    FROM employeelist
    WHERE Email_id = %[sender]
  </AccessorSubQuery>
</AccessorSubQueries>
```

11.2 Subquery Attributes

name

Subquery name referred to in section Fields of the mapping (see "Example of a Configuration File for the Database Connector" on page 6).

separator

Separator used when concatenating multiple rows.

behaviorOnMultiRows

Action to take when a subquery returns more than one row. The only supported behavior is `join`, which concatenates values with the provided separator.

allowDuplicates (optional)

This attribute is mainly used when your subquery returns multiple rows. If set to `False`, duplicates in the results are ignored in the concatenation of the results. If set to `true`, duplicates are present.

singleQuoteEscapeSequence (optional) CES 7.0.5425+ (May 2013)

When the value of the returned field contains single quotes, these single quotes must be escaped. By default when you omit this attribute, the connector escapes the single quotes by doubling them (ex.: ''). This escaping mechanism should work in most cases. However, some database types require a different escaping sequence for single quotes. In such cases, use this attribute to specify the single quote escape sequence.

Example: For the MySQL database, the single quote escaping sequence is `\'`. In this case, in the `AccessorSubQuery` tag, include the `singleQuoteEscapeSequence` attribute as follows:

```
<AccessorSubQuery name="FirstNameLastName" separator=";" behaviorOnMultiRows="join"
allowDuplicates = "false" singleQuoteEscapeSequence="\'">
```

11.3 Subquery Master Key

In a subquery, a master key used in the `WHERE` clause must respect the format `%[fieldName]`, which corresponds to metadata acquired from the main accessor. The master key is used to make the join between the main query and subqueries.

11.4 Specifying subquery metadata for fields

The `<Fields>` section of the mapping file is used to specify the metadata to use for indexing.

Refer to the following example for a typical `<Fields>` section of a mapping file:

```
<Fields>
  <Uri>http://www.coveo.com/Emails/details.aspx?Id=%[mid]</Uri>
  <ClickableUri>http://www.coveo.com</ClickableUri>
  <FileName>Message_%[mid].txt</FileName>
  <Title>Message_%[mid]</Title>
  <ModifiedDate>%[date]</ModifiedDate>
  <Body>%[body]</Body>
  <CustomFields>
    <CustomField name="sysAuthor">%[sender]</CustomField>
    <CustomField name="firstName">%[FirstNameLastName.firstName]</CustomField>
    <CustomField name="lastName">%[FirstNameLastName.lastName]</CustomField>
  </CustomFields>
</Fields>
```

The metadata of a subquery can be specified for a field or a custom field. The way to specify is similar to the way it is done when referring a field coming from the main accessor: `%[subQueryName.fieldName]`. In the above example, custom field `firstName` is referring subquery named `FirstNameLastName` and uses the `firstName` metadata.

Refer to the following for a complete mapping file, used in our unit tests:

```
<?xml version="1.0" encoding="utf-8" ?>
<ODBC>
  <CommonMapping excludedItems="employeelist">
    <AllowedUsers>
      <AllowedUser type="Windows" allowed="true">
        <Name>everyone</Name>
        <Server></Server>
      </AllowedUser>
    </AllowedUsers>
  </CommonMapping>
  <Mapping type="message">
    <Accessor type="query">
      SELECT message.mid,
      message.sender,
      message.date,
      message.message_id,
      message.subject,
      message.body,
      message.folder
      FROM message
      WHERE DATE like '2001-04-07%'
    </Accessor>
      <AccessorSubQueries>
        <AccessorSubQuery name="FirstNameLastName" separator=";" behaviorOnMultiRows="join">
          SELECT  firstName, lastName
          FROM employeelist
          WHERE Email_id = %[sender]
        </AccessorSubQuery>
      </AccessorSubQueries>
    <Fields>
      <Uri>http://www.coveo.com/Emails/details.aspx?Id=%[mid]</Uri>
      <ClickableUri>http://www.coveo.com</ClickableUri>
      <FileName>Message_%[mid].txt</FileName>
      <Title>Message_%[mid]</Title>
      <ModifiedDate>%[date]</ModifiedDate>
      <Body>%[body]</Body>
      <CustomFields>
        <CustomField name="sysAuthor">%[sender]</CustomField>
        <CustomField name="firstName">%[FirstNameLastName.firstName]</CustomField>
        <CustomField name="lastName">%[FirstNameLastName.lastName]</CustomField>
      </CustomFields>
    </Fields>
    <AllowedUsers>
      <AllowedUser type="CustomGroup" allowed="true">
        <Name>everyone</Name>
        <Server></Server>
      </AllowedUser>
    </AllowedUsers>
  </Mapping>
</ODBC>
```