



## Cisco Info Server SQL

---

The Cisco Info Server is the core of the Cisco Info Center system. This chapter describes how the Cisco Info Server stores and manages alerts, the data structures of the Cisco Info Server, and the syntax of Cisco Info Server SQL.

It contains the following sections:

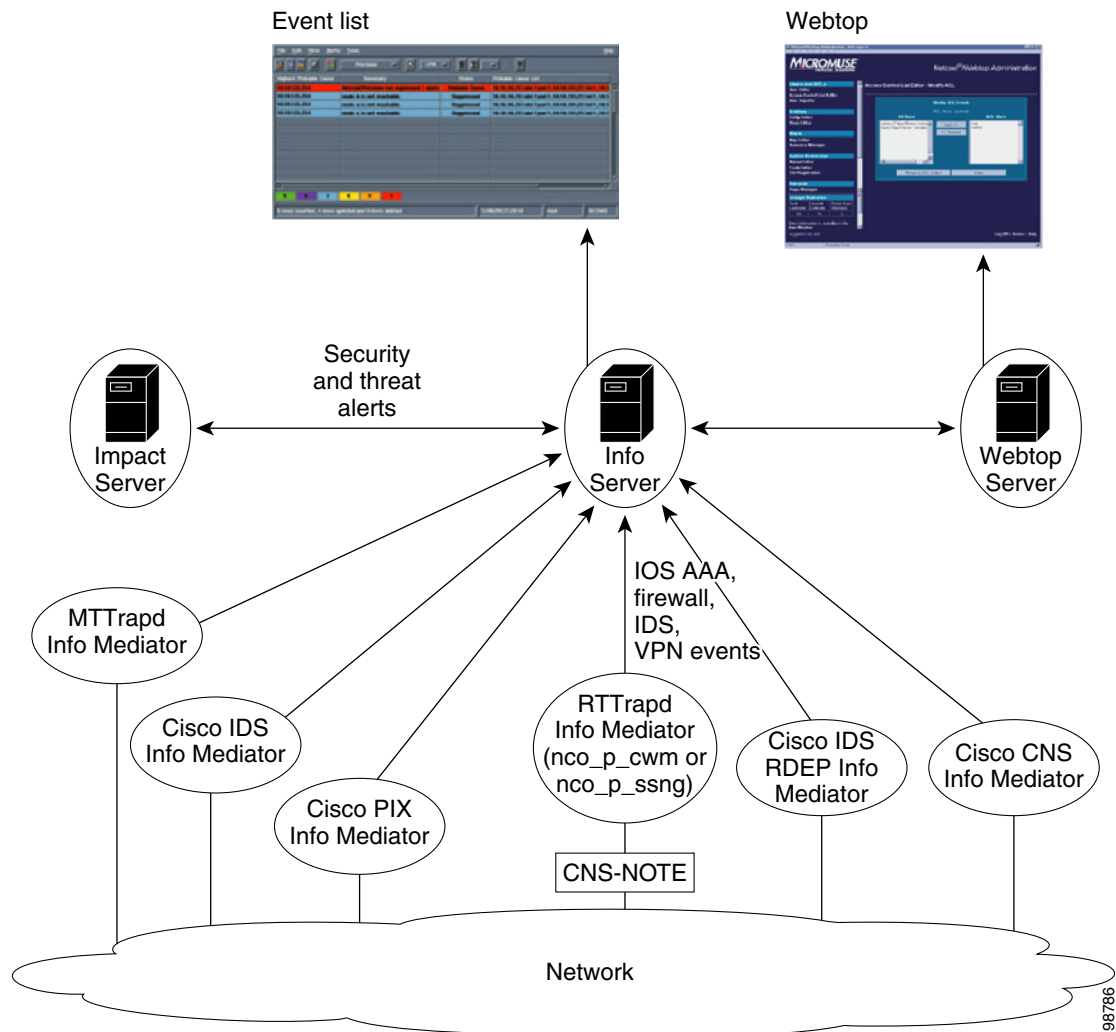
- [Alert Processing](#)
- [Cisco Info Server Databases and Tables, page 2-3](#)
- [Specifying the SQL File, page 2-4](#)
- [Direct Access Using the SQL Interactive Interface \(nco\\_sql\), page 2-4](#)
- [Cisco Info Server Data Types, page 2-6](#)
- [SQL Building Blocks: Operators, Expressions, and Conditions, page 2-8](#)
- [SQL Commands, page 2-14](#)
- [Adding and Modifying Cisco Info Server Data Structures, page 2-25](#)
- [Checkpointing - Logical and Region Storage, page 2-28.](#)

For a full description of the properties and command line options available for the Cisco Info Server, see [Chapter 1, “Configuring the Cisco Info Server and Proxy Server.”](#)

## Alert Processing

The Cisco Info Server is the database server at the core of Cisco Info Center. Alert information is forwarded to the Cisco Info Server from external programs such as Cisco Info Mediators or gateways, stored and managed in database tables, and displayed in the Event List and objective view. [Figure 2-1](#) shows an example Cisco Info Center system architecture.

Figure 2-1 Sample Cisco Info Center System Architecture



## Introduction to Deduplication

A single device may generate the same error repeatedly until the problem is dealt with. The Cisco Info Server uses a process called *deduplication* to ensure alert information generated from the same source is not duplicated. Repeated alerts are identified and stored as a single alert to reduce the amount of data in the Cisco Info Server.

## Deduplicating the alerts.status Table

Alerts are forwarded to the Cisco Info Server from Cisco Info Mediators or gateways, and stored as rows (entries) in the `alerts.status` table. Each alert has an **Identifier** field, used to uniquely identify the problem source. The identifier is generated by the Cisco Info Mediator according to the specification in Cisco Info Mediator rules file, as described in [Creating a Unique Identifier, page 5-9](#).

When the alert arrives, the **alerts.status** table is searched for a matching **Identifier** field. If no entry with the same identifier is found, a new alert entry is inserted. The entry contains detailed information about the problem. For example, the **FirstOccurrence** field stores the time the problem first occurred.

When an entry with the same identifier is found, deduplication occurs. The default behavior is to update the **LastOccurrence** field of the existing entry with the time of the new alert and increment the **Tally** field by one. If the **ReawakenClosed** property is set to **TRUE**, two additional updates can be made:

- if the existing alert is cleared (the **Severity** field is 0), the **Severity** field is updated with the severity of the incoming alert
- if the **Severity** field is updated and the **DeackOnReawaken** property is set to **TRUE**, the **Acknowledged** field is cleared (set to 0), making the alert deacknowledged.

The **DeackOnReawaken** and **ReawakenClosed** properties are described in [Specifying Cisco Info Server Properties, page 1-7](#). All of the fields in the **alerts.status** table are described in [alerts.status Table, page A-1](#).

## Forcing Updates on Deduplication

You can force any character (char), variable-length character (**varchar**), integer (**int**), and time (time) fields to be updated during deduplication using one of the following methods:

- Add the **UPDATE ON DEDUPLICATION** clause to the field name entry of the **CREATE TABLE** command in the Cisco Info Server SQL file. For more information, see [Creating a Table, page 2-14](#) and [Adding and Modifying Cisco Info Server Data Structures, page 2-25](#).
- Use the update function in the Cisco Info Mediator rules file, as described in [Update on Deduplication Function, page E-22](#)
- Use the updating clause in the Cisco Info Server SQL insert command, as described in [Adding Table Data: The INSERT Command, page 2-19](#).

## Additional Alert Processing Capabilities

You can export alert information to other applications through a gateway. A bidirectional gateway can provide failover support to another Cisco Info Server.

The Cisco Info Server can also respond automatically to specified alerts. This is called automation, and is described in [Chapter 4, “Automation.”](#)

# Cisco Info Server Databases and Tables

The Cisco Info Server is made up of the following databases:

- Alerts, which contains alerts forwarded to the Cisco Info Server from Cisco Info Mediators and gateways and related information
- Master, which contains information concerning authorization of users and groups
- Tools, which contains configuration information for Desktop tools
- Auto, which contains information about triggers and actions, and should only be modified using the Automation tool

- Service, which is used to support Cisco Info Center/Internet Service Monitors
- Custom, which can be used for tables added by users.

Each of the tables in these databases is described in [Appendix A, “Cisco Info Server Tables.”](#) The fully-qualified table name includes the database name and the table name, separated by a period (.). For example, **alerts.details**.

## Specifying the SQL File

The Cisco Info Server SQL file is read when a Cisco Info Server starts. It contains a set of **CREATE DATABASE** and **CREATE TABLE** commands which define the Cisco Info Server data structures. If you do not specify an SQL file when starting a Cisco Info Server, the **\$OMNIHOME/etc/<servername>.sql** default file is used (where *<servername>* is the name of the Cisco Info Server). Use the **-sqlfile** command line option to specify the full path and file name of an alternative SQL file.

You can modify the default Cisco Info Server template to create an alternative SQL file, as described in [Adding and Modifying Cisco Info Server Data Structures, page 2-25](#).

## Direct Access Using the SQL Interactive Interface (nco\_sql)

This section describes how to log into the SQL interactive interface (**nco\_sql**) to connect to Cisco Info Servers and use SQL commands to interact with and control the Cisco Info Server. The SQL interactive interface allows you to perform tasks such as creating a new database table or stopping the Cisco Info Server.



### Note

Only users with the Allow ISQL Access option enabled can connect. This option is described in [Creating a User, page 3-5](#).

## Logging into the SQL Interactive Interface

Use the SQL interactive interface to connect to a Cisco Info Server as a specific user. For example:

```
$OMNIHOME/bin/nco_sql -server <servername> -user <username>
```

where *<servername>* is the name of the Cisco Info Server and *<username>* is a valid user name. If you do not specify a server name, the default server name **NCOMS** is used. If you do not specify a user name, the default is the user running the command. The command line options for **nco\_sql** are described in [Table 2-1](#).

Table 2-1 nco\_sql Command Options

Option	Description
<b>-help</b>	Displays help information about the command line options and exits.
<b>-nosecure</b>	When specified, no security checks are performed on the connection request. You must use this option to connect to Cisco Info Servers and gateways for releases previous to Cisco Info Center version 3.5.
<b>-password</b> <password>	Specifies the password for the user. It is not recommended to specify the password on the command line because this makes the password visible. If not specified, you are prompted for the password.
<b>-server</b> <servername>	Specifies the name of the server to which to connect. The default is <b>NCOMS</b> .
<b>-user</b> <username>	Specifies the name of a Cisco Info Center user. The default is the user running the command.

Once logged in using **nco\_sql** or **isql**, you can enter Cisco Info Server SQL commands.

## Entering SQL Commands Using the SQL Interactive Interface

After connecting with a user name and password, a numbered prompt is displayed:

1>

You can enter any valid Cisco Info Server SQL command at the prompt. Commands are not processed until you enter the keyword **go** in lowercase at the beginning of a new line and press **Return**. Commands can be split over multiple lines. Multiple commands, separated by a semicolon, can be executed with a single **go**.

The results of the command are displayed.

For the syntax of Cisco Info Server SQL commands, see [SQL Commands, page 2-14](#).

## Exiting the SQL Interface

When you are using the SQL interface, the following applies:

- to exit from the SQL interactive interface, press **Control + D** or enter **quit**
- the SQL interactive interface exits if you shut down the Cisco Info Server.

## Using the Secure SQL Interactive Interface

When a Cisco Info Server runs in secure mode, it requires clients such as Cisco Info Mediators, Desktop clients, gateways, and the SQL interactive interface to connect using valid user names and passwords. The login information is automatically encrypted when it is transmitted between components to make snooping ineffective. Versions of Cisco Info Center previous to 3.5 do not support this transmission encryption.

The SQL interactive interface runs in secure mode unless you specify the **-nosecure** option. You can use the **-nosecure** option to connect to versions of the Cisco Info Server previous to 3.5.

When you run the SQL interactive interface in secure mode, it makes use of the **nco\_get\_login\_token** utility to encrypt its login data for transmission. The utility produces a token that can be used only once to log in to the Cisco Info Server. The token has a time limit after which it expires and becomes invalid.

## Encrypting Passwords in nco\_sql Scripts

You can use the **nco\_sql\_crypt** utility to encrypt plain text login passwords so they are not exposed in any scripts that run **nco\_sql**. Passwords encrypted using **nco\_sql\_crypt** are decrypted by the Cisco Info Server when the connection is made.

To encrypt a plain text password:

---

**Step 1** Enter the following:

```
$OMNIHOME/bin/nco_sql_crypt <password>
```

where *<password>* is the unencrypted form of the password. The **nco\_sql\_crypt** utility displays an encrypted version of the password.

**Step 2** Copy the encrypted password into the script.

---

## Cisco Info Server Data Types

Each column value in a Cisco Info Server table has an associated data type. The data type determines how the Cisco Info Server processes the data in the column. For example, the plus operator (+) adds integer values, but concatenates string values. The data types supported by the Cisco Info Server are listed in [Table 2-2](#).

Table 2-2 Cisco Info Server Data Types

Data Type	Description	Cisco Info Server ID for Data Type
<b>char</b> ( <i>&lt;integer&gt;</i> )	A fixed size character string, up to <i>&lt;integer&gt;</i> characters long (8192 bytes is the maximum). The default value is ''.  The <i>&lt;char&gt;</i> type is identical in operation to <i>&lt;varchar&gt;</i> , but performance is better for mass updates that change the length of the string.	10
<b>varchar</b> ( <i>&lt;integer&gt;</i> )	A variable-sized character string, up to <i>&lt;integer&gt;</i> characters long (8192 bytes is the maximum). The default value is ''.  The <i>&lt;varchar&gt;</i> type uses less storage space than the <i>&lt;char&gt;</i> type and the performance is better for deduplication, scanning, insert, and delete operations.	2
<b>int</b>	An integer value. The default value is 0.	0
<b>time</b>	The number of seconds since midnight January 1, 1970.	1
<b>timestamp</b>	Same as the <b>time</b> data type, but automatically populated by the Cisco Info Server when the record is created.	4
<b>itime</b>	A time field that has modified semantics to allow Cisco Info Mediators to update it.	8
<b>incr</b>	An integer field used for unique serial number generation. This field is automatically populated with a Cisco Info Server internal variable when the record is inserted.	5
<b>optime</b>	A time field automatically updated with the current date and time whenever an insert or update operation is performed.	7
<b>opcount</b>	An integer field incremented whenever a record is inserted or an attempt is made to insert a record that already exists.	6

## Date Literals

Date literals return information about the current time and date, as shown in [Table 2-3](#). They are used for time-sensitive automations or filters.

Table 2-3 Date Literals

Date Literal	Description
<b>getdate</b>	Returns an integer value containing the current date and time in seconds.
<b>dayname</b>	Returns the name of the current day (for example, <b>mon, tue, fri</b> ).
<b>monthname</b>	Returns the name of the current month (for example, <b>jan, feb, jul</b> ).

Table 2-3 Date Literals (continued)

Date Literal	Description
<b>dayofmonth</b>	Returns the day of the current month (for example, if it is January 21st, <b>dayofmonth</b> returns <b>21</b> ).
<b>minuteofhour</b>	Returns the minute of the current hour (for example, if it is 11:31 then <b>minuteofhour</b> returns <b>31</b> ).
<b>hourofday</b>	Returns the hour of the current day (for example, if it is 11:31 then <b>hourofday</b> returns <b>11</b> ).
<b>dayasnum</b>	Returns the day of the current week as a number (for example, if it is Sunday, <b>dayasnum</b> returns <b>0</b> , for Monday it returns <b>1</b> , and so on).
<b>monthasnum</b>	Returns the month of the current year as a number (for example, if it is January, <b>monthasnum</b> returns <b>1</b> , for February it returns <b>2</b> , and so on).

## Date Function

You can use the **to\_char** function to convert a date into a string.

Table 2-4 to\_char Function

Date Literal	Description
<b>to_char</b> (<date_type>, <conversion_spec>)	Converts a <b>time</b> , <b>timestamp</b> , or <b>ltime</b> value into a string. The second argument is a conversion specification to format the output. This format is determined by the POSIX <b>strptime</b> function.  For example:  <b>select to_char&gt;LastOccurrence, '%d') from alerts.status;</b>

## SQL Building Blocks: Operators, Expressions, and Conditions

You can use the following SQL building blocks to manipulate data in a Cisco Info Server database in the commands described in [Data Manipulation Commands, page 2-15](#):

- Operators
- Expressions
- Conditions.

## Operators

You can use operators to compute values from data items, known as operands. An operator processes (adds, subtracts, and so on) a data item or items. The operand is a data item on which the operation is performed. Together operators and operands form expressions, described in more detail on [Expressions, page 2-12](#). In the expression `7 + 3`, 7 and 3 are operands and the plus symbol (+) is the operator.

Operators can be unary or binary. Unary operators act on only one operand. For example, the minus (-) operator can be used to indicate negation. Binary operators act on two operands. For example, the same minus (-) operator can be used to subtract one operand from another.

Some operators, such as the plus (+) operator, are polymorphic; they can be assigned a different meaning in different contexts. For example, you can use the plus (+) operator to add two numbers (7+3) or concatenate two strings ('The Cisco Info Server' + ' started').

Operators used in Cisco Info Server SQL are grouped into the following categories:

- [Math and String Operators](#)
- [Binary Comparison Operators](#)
- [List Comparison Operators](#).

Each of these categories and their corresponding operators are described in the following sections.

Operator precedence is described in [Operator Precedence, page 2-11](#).

### Math and String Operators

You can use math operators to add, subtract, divide, and multiply numeric operands in expressions. [Table 2-5](#) describes the math operators supported by the Cisco Info Server.

**Table 2-5 Math Operators**

Operator	Description	Example
+ -	Unary operators indicating a positive or negative operand.	<code>select * from alerts.status where Severity = -1;</code>
* /	Binary operators used to multiply (*) or divide (/) two operands.	<code>select * from alerts.status where Tally * Severity &gt; 10;</code>
+ -	Binary operators used to add (+) or subtract (-) two operands.	<code>select * from alerts.status where Severity - 1 &gt; 3;</code>

You can use string operators to manipulate character strings (**VARCHAR** and **CHAR** data types). [Table 2-6](#) describes the string operator supported by the Cisco Info Server.

**Table 2-6 String Operator**

Operator	Description	Example
+	Binary operator used to concatenate two strings.	<code>update alerts.status set Location = Node + NodeAlias;</code>

## Binary Comparison Operators

You can use binary comparison operators to compare numeric and string values for equality and inequality. [Table 2-7](#) describes the comparison operators supported by the Cisco Info Server.

**Table 2-7 Comparison Operators**

Operator	Description	Example
=	Tests for equality.	<code>select * from alerts.status where Severity = 3;</code>
<> !=	Tests for inequality.	<code>select * from alerts.status where Severity &lt;&gt; 1;</code>

The **LIKE** and **NOT LIKE** comparison operators allow special regular expression pattern-matching metacharacters in the string being compared to the column expression. Regular expressions are made up of normal characters and metacharacters. Normal characters include upper and lower case letters and numbers. Regular expression pattern-matching can be performed with either a single character or a pattern of one or more characters in parentheses, called a character pattern. Metacharacters have special meanings, described in [Table 2-8](#).

**Table 2-8 Pattern Matching Metacharacters**

Pattern-Matching Metacharacter	Description	Example
*	Matches zero or more instances of the preceding character or character pattern.	The pattern 'goo*' matches 'my godness', 'my goodness', and 'my goodness', but not 'my gdness'.
+	Matches one or more instances of the preceding character or character pattern.	The pattern 'goo+' matches 'my goodness' and 'my goodness', but not 'my godness'.
?	Matches zero or one instance of the preceding character or character pattern.	The pattern 'goo?' matches 'my godness' and 'my goodness', but not 'my goodness' or 'my gdness'.
\$	Matches the end of the string.	The pattern 'end\$' matches 'the end', but not 'the ending'.
^	Matches the beginning of the string.	The pattern '^severity' matches 'severity level 5', but not 'The severity is 5'.
.	Matches any single character.	The pattern 'b.at' matches 'baat', 'bBat', and 'b4at', but not 'bat' or 'bB4at'.
[abcd]	Matches any characters in the square brackets or in the range of characters separated by a hyphen (-), such as [0-9].	^[A-Za-z]+\$ matches any string that contains only upper or lower case letter characters.
[^abcd]	Matches any character except those in the square brackets or in the range of characters separated by a hyphen (-), such as [0-9].	[^0-9] matches any string not containing any numeric characters.

Table 2-8 Pattern Matching Metacharacters (continued)

Pattern-Matching Metacharacter	Description	Example
()	Indicates the characters in the parenthesis should be treated as a character pattern.	A(boo)+Z matches 'AbooZ', 'AboobooZ', and 'AboobooZ', but not 'AboZ' or 'AboooZ'.
	Matches one of the characters or character patterns on either side of the vertical bar.	A(B C)D matches 'ABD' and 'ACD', but not 'AD', 'ABCD', 'ABBD', or 'ACCD'.
\	The backslash escape character indicates the metacharacter following should be treated as a regular character. The metacharacters in this table require a backslash before them if they appear in a regular expression.	To match an opening square bracket, followed by any digits or spaces, followed by a closed bracket, use the regular expression \[[0-9]*\].

## List Comparison Operators

You can use list comparison operators to compare a value to a list of values. The syntax of a list comparison expression is:

```
<expression> [NOT] IN (<expression>,...)
```

For example, in the query:

```
select * from alerts.status where Severity in (1, 3, 5)
```

The query returns the rows in which **Severity** is equal to the number 1, 3, or 5.

## Logical Operators

You can use logical operators to combine comparisons with the **AND** and **OR** operators. An **AND** expression is true only if all of its inputs are true. An **OR** expression is **TRUE** if any of its inputs are true. For example, you could create the following query:

```
SELECT * from alerts.status where Node = 'node1' and Severity > 4 and Summary like 'alert
on *.*'
```

## Operator Precedence

If an expression contains multiple operators, the Cisco Info Server uses operator precedence to determine the order in which to evaluate the expression. Operators are evaluated from those with the highest precedence to those with the lowest precedence. For example, the binary plus (+) operator has a lower precedence than the multiplication operator (\*). In the expression  $3 + 5 * 2$ , the result is 13 because 5 is multiplied by 2 before the result (10) is added to 3.

Use parentheses in an expression to change the order in which the items are evaluated. The contents of parentheses are always evaluated before anything outside of the parentheses. In the expression  $(3 + 5) * 2$ , the result is 16 because 3 is added to 5 before the result (8) is multiplied by 2.

If operators have equal precedence, they are evaluated in order from left to right. The order of precedence of all Cisco Info Server operators is shown below.

<b>Highest Precedence</b>
Unary + -
Math * /
Binary + -
Comparison operators (including list comparisons)
NOT
AND
OR
<b>Lowest Precedence</b>

## Expressions

An expression is a syntactic combination of values and operations combined to compute new values. Expressions can be simple or complex.

### Simple Expressions

A simple expression is a single constant or variable value or a column name. This can be any of the following:

- a quoted string ('Node XB1')
- a number (9)
- a date literal (**getdate**)
- a column name (**Severity**)
- a Cisco Info Server property (**%UpdateSummary**)
- the **%HostName** internal variable, which indicates the machine on which the Cisco Info Server is running
- the **%LogFile** internal variable, which indicates the location of the current Cisco Info Server log file
- the **%ServerOS** internal variable, which indicates the type of operating system on which the Cisco Info Server is running
- the **%ServerName** internal variable, which indicates the name of the Cisco Info Server
- the **%PAName** internal variable, which indicates the name of the process agent that started the Cisco Info Server
- the **%OMNIHOME** internal variable, which indicates the installation directory.

## Complex Expressions

A complex expression is created from simple expressions combined using operators (**Severity - 1**). You can combine simple or complex expressions with other simple or complex expressions to create increasingly complex expressions, such as **-(Severity+Tally)**.

**Note**

Complex expressions are subject to type constraints. For example, the expression **5 + 'Node XB1'** is not valid because you cannot add an integer to a string. For more information about data types, see [Cisco Info Server Data Types, page 2-6](#).

## Conditions

A condition is a combination of expressions and operators that evaluate to true or false. You can use conditions to search, filter, and test rows in the **WHERE** clause of the **SELECT**, **UPDATE**, and **DELETE** SQL commands.

The following are valid conditions:

TRUE | FALSE

*<condition>*

NOT *<condition>*

*<condition>* AND *<condition>*

*<condition>* OR *<condition>*

*<expression>* *<operator>* *<expression>*

*<expression>*[NOT] IN (*<subquery>*)

*<expression>* [NOT] IN (*<expression>*,...)

*<expression>* [NOT] LIKE *<regexp\_pattern>*

You can combine conditions into increasingly complex conditions. For example:

**(Severity > 4) AND (Node = 'node%')**

The following example shows the use of a condition in a subquery:

```
select * from alerts.status where Serial in
(select Serial from alerts.journal);
```

# SQL Commands

This section details Cisco Info Server SQL commands, including syntax and examples.



**Note**

---

SQL keywords are not case sensitive.

---

## Data Definition Language Commands

Data definition language commands enable you to create data structures, such as databases and tables, using Cisco Info Server SQL.

The names of Cisco Info Server structures must begin with an uppercase or lowercase letter, followed by uppercase or lowercase letters, numbers, or underscore (\_) characters, up to forty characters in length.



**Note**

---

Names of Cisco Info Server structures are case sensitive.

---

## Creating a Database

Use the **CREATE DATABASE** command to create a new database. A database is a structured collection of data organized for quick access to desired information. A relational database uses tables as logical containers to store this information in rows and columns.

### Syntax

```
CREATE DATABASE <database_name>;
```

The <database\_name> must be unique within the Cisco Info Server.

### Example

```
create database newthings;
```

## Creating a Table

Use the **CREATE TABLE** command to create a table in the database currently in use.

### Syntax

```
CREATE TABLE <table_name>
(
  <column_name> <data_type> [UPDATE ON DEDUPLICATION] ,...
  [, PRIMARY KEY(<column_name> ) ]
  [, PERMANENT]
);
```

The table name must be unique in the database.

For each column in the table, the column name and data type must be specified with a comma separator between column definitions. If you specify **UPDATE ON DEDUPLICATION** for a column, the column is updated whenever an alert is deduplicated, as described in the [“Alert Processing” section on page 2-1](#).

Following the column list, you can specify a primary key column that uniquely identifies each row. The primary key is used for row retrieval and in deduplication.

**Note**

The Cisco Info Server does not support multiple primary keys. If you specify more than one primary key entry for a table, only the last entry is used.

If you specify the table is permanent, it is maintained when the Cisco Info Server is shut down. When the Cisco Info Server is restarted, the table is restored automatically. Otherwise, it is deleted.

**Example**

```
create table userlist
(
  userid char(20),
  username char(128),
  primary key(userid),
  permanent
);
```

## Data Manipulation Commands

Data manipulation language commands enable you to query and modify data in existing tables using Cisco Info Server SQL.

### Viewing Table Data: The SELECT Command

Use the **SELECT** command to retrieve one or more rows, or partial rows, of data from an existing table, and to perform grouping functions on the data.

#### Basic (Scalar) Select Syntax

The scalar **SELECT** command retrieves columns and rows from a table based on specified criteria. The syntax is:

```
SELECT [TOP num_rows] { * | <scalar_column_expr> [AS <alias_name>] ,... }
FROM [<database_name>.<table_name>]
[WHERE <condition>]
[ORDER BY <column_name_or_alias> [ASC | DESC] ,... ];
```

Use the optional **TOP** keyword to display only the first **num\_rows** number of rows of the query results that match the selection criteria. If you include the **TOP** keyword, you should also include an **ORDER BY** clause to order the selected rows.

Use an "\*" to retrieve all columns in the table. Otherwise, specify a comma-separated list of columns to retrieve. You can also create virtual columns using:

- simple expressions (**Severity**)
- complex expressions that contain math or string operators (**Severity + Tally**)
- date literals (**getdate - 60**).

Expressions are described in [Expressions, page 2-12](#). Math and string operators are listed in [Table 2-5 on page 2-9](#) and [Table 2-6 on page 2-9](#). Date literals are listed in [Date Literals, page 2-7](#).

Following a column or virtual column, you can include the **AS** keyword followed by an alias. This alias is a replacement heading for the column or virtual column name, displayed in the query results. If you specify a column alias, use that alias in any references in the **ORDER BY** clause. The maximum length of a column name or alias is 32 characters. You can specify up to 30 columns or virtual columns.

If you include a **WHERE** clause, only rows meeting the criteria specified in the condition are returned. Conditions are described in [Conditions, page 2-13](#).

Use the optional **ORDER BY** clause to display the results in sequential order depending on the values of one or more column names, in either descending (**DESC**) or ascending (**ASC**), order. If the **ORDER BY** clause is not specified, no ordering is used. If you have specified a column alias using the **AS** keyword, use that alias in any references in the **ORDER BY** column list rather than the corresponding column name.

### Basic (Scalar) Select Examples

To select all rows of the **alerts.status** table where the **Severity** is equal to **4**, enter:

```
select * from alerts.status where Severity = 4;
```

To select all rows of the **alerts.status** table where the **Node** contains the string terminal followed by any other characters, enter:

```
select * from alerts.status where Node like 'terminal.*';
```

For information on pattern-matching metacharacters used in the **WHERE** clause, see [Table 2-8 on page 2-10](#).

In the following example, the virtual column **Severity+Tally** is populated by adding the values of the two columns together:

```
select Severity, Severity+Tally from alerts.status;
```

The following example is the same as the previous example, except the virtual column **Severity+Tally** is renamed **Real\_Severity**.

```
select Severity, Severity+Tally as Real_Severity from alerts.status;
```

### Aggregate Select Syntax

An aggregate **SELECT** command differs from a scalar **SELECT** command because it performs a calculation on a number of rows and returns a single value. The syntax is:

```
SELECT <aggr_expression> [AS <alias_name>],...
FROM [<database_name>.<table_name>]
[WHERE <condition>];
```

Following an aggregate expression, you can include the **AS** keyword followed by an alias. This alias is a replacement heading for the aggregate expression, displayed in the query results.

The following aggregate expressions are allowed:

```
MAX(<scalar_column_expr>) |
MIN(<scalar_column_expr>) |
COUNT(<scalar_column_expr> | *) |
AVG(<scalar_column_expr>) |
SUM(<scalar_column_expr>) |
DIST(<scalar_column_expr>, <value>)
```

The calculation performed by each of these aggregate functions are listed in [Table 2-9](#).

**Table 2-9 Aggregate Functions**

Function	Result Returned
<b>MAX</b> (<scalar_column_expr>)	Returns the maximum numeric value for the column expression from the rows that meet the <b>SELECT</b> condition.
<b>MIN</b> (<scalar_column_expr>)	Returns the minimum numeric value for the column expression from the rows that meet the <b>SELECT</b> condition.
<b>AVG</b> (<scalar_column_expr>)	Returns the average numeric value for the column expression from the rows that meet the <b>SELECT</b> condition.
<b>SUM</b> (<scalar_column_expr>)	Returns the sum (total) of the numeric values for the column expression from the rows that meet the <b>SELECT</b> condition.
<b>COUNT</b> (<scalar_column_expr>) <b>COUNT</b> (*)	Returns the total number of rows that meet the <b>SELECT</b> condition.
<b>DIST</b> (<scalar_column_expr>, <value>)	Returns the total number of rows for which the column equals the specified value. The result of <b>DIST</b> (<scalar_column_expr>, <value>) is equivalent to:  <b>SELECT COUNT</b> (<scalar_column_expr>) <b>from</b> <table_name> <b>WHERE</b> <scalar_column_expr> = <value>

The maximum length of a column name or alias is 32 characters. You can specify up to 30 aggregate expressions.

If you include a **WHERE** clause, only rows meeting the criteria specified in the condition are returned. Conditions are described in [Conditions, page 2-13](#).

### Aggregate Select Examples

The following example returns the highest **Severity** value, the average **Severity** value, and the number of rows for which the **Severity** is equal to **4**:

```
select MAX(Severity), AVG(Severity), DIST(Severity, 4) from alerts.status;
```

The following example returns the number of rows for which the value of **Node** is **myhost**:

```
select DIST(Node, 'myhost') from alerts.status;
```

The following examples do comparisons using the **getdate** date literal, which returns the current time:

```
select MAX(getdate-LastOccurrence) from alerts.status;
```

```
select AVG((getdate-LastOccurrence)/60) as ResponseTime from alerts.status where OwnerUID=34;
```

Date literals are described in [Date Literals, page 2-7](#).

## Group By Select Syntax

A **SELECT** command containing a **GROUP BY** clause enables you to group into a single row all rows that have identical values in a specified column or combination of columns. Used with aggregate functions, described in the preceding section, you can find the aggregate value for each group of column values. The syntax is:

```
SELECT <column_name> [AS <alias_name>] ,... ,
aggr_expression [AS <alias_name>] ,...
FROM [<database_name>.<table_name>]
[WHERE <condition>]
GROUP BY <column_or_alias_name> ,...
[ORDER BY <column_or_alias_name> [ASC | DESC] ,... ];
```

For this syntax, you must specify the actual column names, rather than using scalar expressions to create virtual columns as described in [Basic \(Scalar\) Select Syntax, page 2-15](#). Additionally, you cannot use the **TOP** keyword. An “\*” is allowed only in the **COUNT(\*)** aggregate function. Aggregate functions are listed in [Table 2-9 on page 2-17](#).



### Note

The column list (<column\_name> [**AS** <alias\_name>],... ,) must precede all aggregate expressions (**aggr\_expression** [**AS** <alias\_name>] ,...).

Following a column name or aggregate expression, you can include the **AS** keyword followed by an alias. This alias is a replacement heading for the column name or aggregate expression, displayed in the query results. If you specify an alias, use that alias in any references in any **GROUP BY** and **ORDER BY** clauses. The maximum length of a column name or alias is 32 characters. You can specify up to 30 columns and 30 aggregate expressions.

The **GROUP BY** clause gathers all of the rows together that contain data in the specified columns and allows aggregate functions to be performed on these columns based on column values. If you specified a column alias using the **AS** keyword, use that alias in any references in the **GROUP BY** column list rather than the corresponding column name.



### Note

The column list in the **GROUP BY** clause must match the column list being selected and must not contain any of the aggregate expressions.

Use the optional **ORDER BY** clause to display the results in sequential order depending on the values of one or more column names, in either descending (**DESC**) or ascending (**ASC**), order. If the **ORDER BY** clause is not specified, no ordering is used. If you specified a column alias using the **AS** keyword, use that alias in any references in the **ORDER BY** column list rather than the corresponding column name.

## Group By Select Examples

The following example returns the highest **Severity** value found for each node.

```
select Node, max(Severity) from alerts.status group by Node;
```

The following example returns the highest severity value found for each node except the node named **Sun1**, ordered from lowest to highest maximum severity.

```
select Node, max(Severity) as MAX_Sev from alerts.status where Node <> 'Sun1' group by
Node order by MAX_Sev;
```

The column alias for **max(Severity)**, which is **MAX\_Sev**, is displayed as the heading in the query results.

## Removing Table Data: The DELETE Command

Use the **DELETE** command to delete one or more rows of data from an existing table.

### Syntax

```
DELETE FROM [<database_name>.<table_name>  
[VIA ('<value_of_primary_key_column>')]  
[WHERE <condition>];
```

If you are deleting a single row and you know the value of the primary key for the row you wish to delete, specify the value using the optional **VIA** clause to improve the performance of the delete command. The value of the primary key column must be enclosed in quotes.

If you include a **WHERE** clause, only rows meeting the criteria specified in the condition are returned. Conditions are described in [Conditions, page 2-13](#). If no **WHERE** clause is specified, all rows are deleted.

### Example

To remove all the rows of the **alerts.status** table where the value of the **Node** field is equal to **Fred**, enter:

```
delete from alerts.status where Node = 'Fred' ;
```

## Adding Table Data: The INSERT Command

Use the **INSERT** command to insert a new row of data into an existing table.

### Syntax

```
INSERT INTO [<database_name>.<table_name>  
[(<column_name>,...)] VALUES (<expression>,...)  
[UPDATING (<column_name>,...)];
```

You must specify a value for the primary key column in the **INSERT** command.

If you are not inserting values for every column in the row, you must specify a comma-separated list of columns being inserted in parentheses, followed by the **VALUES** keyword, followed by a comma-separated list of values in parentheses.

The **UPDATING** keyword forces the specified column(s) to be updated if the insert is deduplicated.

### Examples

To insert an alert into the **alerts.status** table specifying the values in the specified four columns, enter:

```
insert into alerts.status (Identifier, Severity, Tally, Serial) values  
( 'MasterMachineStats15', 5, 12, 21);
```

Specify the data to be inserted following the **VALUES** keyword in a parenthesized, comma-separated list of literal values. Enter the values in the same sequence as the specified fields. All other fields are populated with default values.

The **UPDATING** keyword forces the specified columns to be updated if the insert is deduplicated. To force the **Severity** field of the previous example to be updated, enter:

```
insert into status (Identifier, Severity, Tally, Serial) values ('MasterMachineStats15',  
5, 12, 21) updating (Severity);
```

## Modifying Table Data: The UPDATE Command

Use the **UPDATE** command to update one or more columns in an existing row of data in a table.

### Syntax

```
UPDATE [<database_name>.<table_name>
[VIA ('<value_of_primary_key_column>')]
SET <column_name> = <expression>,...
[WHERE< condition>];
```

If you are updating a single row and you know the value of the primary key for the row you wish to update, specify the value using the optional **VIA** clause to improve the performance of the update command. The value of the primary key column must be enclosed in quotes.

If you include a **WHERE** clause, only rows meeting the criteria specified in the condition are returned. Conditions are described in [Conditions, page 2-13](#). If no condition is specified, all rows are updated.

### Examples

To set the **Severity** to **0** for any row of the **alerts.status** table where the **Node** is equal to **Fred**, enter:

```
update alerts.status set Severity = 0 where Node = 'Fred';
```

You can use the values of fields in calculations. In the following example, **Severity** is set to **0** when an alert has been acknowledged:

```
update status set Severity=(1-Acknowledged)*Severity;
```

To search for a row where **Severity** is equal to 1, **Node** is equal to Fred, and you want to set **Severity** to 0 and the **Summary** field to the string Discarded, enter:

```
update alerts.status set Severity = 0 , Summary = 'Discarded'
where Severity = 1 and Node = 'Fred';
```

## Adding and Modifying Service Status Data: The SVC Command

Use the **SVC** command to add or update the state of a service status event in the **service.status** table for Cisco Info Center Service Monitors.

### Syntax

```
SVC UPDATE '<name>' <integer>;
```

In this command, <name> is the name of the profile element generating the alert and <integer> is its current status. Valid values for the service status are shown in [Table 2-10](#).

**Table 2-10 Service Update Command Status Levels**

Integer	Service Status Level
0	Good
1	Marginal

If any other value is entered, the service level is set to **3** (unknown).

### Example

```
svc update 'newservice' 2;
```

## Changing the Default Database

Use the **USE** command to indicate which database is the default for locating tables. You can then specify a table name in an **nco\_sql** session without preceding it with the database name.

### Syntax

```
USE DATABASE <database_name>;
```

### Example

```
use database newthings;
```

## Viewing Table Column Information: The DESCRIBE Command

Use the **DESCRIBE** command to view information about the columns of the specified table. The output includes the column name, the length of the column, whether the column is a primary key (**1** if TRUE, **0** if FALSE), and the data type, returned as the Cisco Info Server ID as listed in [Table 2-2 on page 2-7](#).

### Syntax

```
DESCRIBE [<database_name>.<table_name>;
```

### Example

```
describe newtable;
```

## Listing All Databases: The LIST DATABASES Command

Use the **LIST DATABASES** command to list the all of the databases in the Cisco Info Server.

### Syntax

```
LIST DATABASES;
```

### Example

```
list databases;
```

## Listing All Tables: The LIST TABLES Command

Use the **LIST TABLES** command to list all of the tables in a database.

### Syntax

```
LIST TABLES [IN <database_name>;
```

If the *<database\_name>* is specified, the tables in the specified database are listed. Otherwise, tables in the current database are listed.

## Examples

```
list tables;
list tables in mydb;
```

## Listing Connected Users

Use the **USERS** command to list the names of all the users connected to the Cisco Info Server.

### Syntax

```
USERS;
```

### Example

```
users;
```

## Changing a User Password

Use the **PASSWORD** command to change the password for the specified user. Only a super user can execute this command.

### Syntax

```
PASSWORD <user>, <new_pass>;
```

where <user> is the name of the user whose password is to be changed. <new\_pass> is the new password.

### Example

```
password 'john', 'realpassword';
```

## System Administration Commands

Cisco Info Server SQL includes commands that enable you to view and change the current settings and perform other administrative tasks.

## Shutting Down the Cisco Info Server: The SHUTDOWN Command

Use the **SHUTDOWN** command to shut down the Cisco Info Server after synchronizing the persistent store. Only a super user can execute this command.

### Syntax

```
SHUTDOWN [DUMP];
```

The **DUMP** option is used to generate a logical checkpoint **.dat** file instead of a physical checkpoint **.tab** file when physical checkpointing is enabled. You can use the **DUMP** command, described in [Generating a Logical Checkpoint File: The DUMP Command, page 2-24](#), to generate the checkpoint file without shutting down the Cisco Info Server.

## Examples

```
shutdown;  
shutdown dump;
```

## Setting a Property: The SET PROP Command

Use the **SET PROP** command to set a property value.

### Syntax

```
SET PROP <property_name> TO <value>;
```

You can set the specified property to an integer value (for example, 10), a boolean value (for example, **TRUE** or **FALSE**), or a single quoted string (for example, '<string value>'), depending on the data type of the property. For more information on Cisco Info Server properties, see [Specifying Cisco Info Server Properties, page 1-7](#).

### Example

```
set prop LogInterval to 200;
```

## Listing All Properties and Their Values: The SHOW PROPS Command

Use the **SHOW PROPS** command to list all the properties, returning the names and values as rows of data.

### Syntax

```
SHOW PROPS;
```

For more information on Cisco Info Server properties, see [Specifying Cisco Info Server Properties, page 1-7](#).

### Example

```
show props;
```

## Listing A Single Property and Its Value: The GET PROP Command

Use the **GET PROP** command to list the value of the specified property.

### Syntax

```
GET PROP <property_name>;
```

This returns the value of the specified property as an SQL row. For more information on Cisco Info Server properties, see [Specifying Cisco Info Server Properties, page 1-7](#).

### Example

```
get prop DebugLevel;
```

## Saving Properties to a File: The SAVE PROPS Command

Use the **SAVE PROPS** command to save the current property settings to a file.

### Syntax

```
SAVE PROPS [TO '<filename>'];
```

You can specify a file name other than the default; otherwise, the properties are saved to the default property file location. For more information on Cisco Info Server properties, see [Specifying Cisco Info Server Properties, page 1-7](#).

### Examples

```
save props;
save props to '/etc/mysavedprops.props';
```

## Synchronizing Database or Table Data: The SYNC Command

Use the **SYNC** command to force the consolidation of either the database or table specified. This writes out the current data and applies the **.trans** file changes to the **.dat** file.

### Syntax

```
SYNC [DATABASE <database_name> | TABLE <table_name>];
```

### Examples

```
sync database alerts;
sync table alerts.status;
```

## Generating a Logical Checkpoint File: The DUMP Command

Use the **DUMP** command to generate a logical checkpoint **.dat** file instead of a physical checkpoint **.tab** file when physical checkpointing is enabled. Only a super user can execute this command.

### Syntax

```
DUMP;
```

For more information about checkpointing, see [Checkpointing - Logical and Region Storage, page 2-28](#).

### Example

```
dump;
```

## Reclaiming Unused Disk Space: The VACUUM Command

Occasionally the use of region storage can result in table fragmentation. Use the **VACUUM** command to reclaim unused space in Cisco Info Server tables when physical checkpointing is enabled.



#### Note

---

This command should only be used on the advice of Cisco Systems Support.

---

## Syntax

```
VACUUM [<database_name>.]<table_name>;
```

In this command, <database\_name> is the name of the database and <table\_name> is the name of the table. If the command is issued when logical storage is enabled, it has no effect.

## Example

```
vacuum alerts.status;
```

# Adding and Modifying Cisco Info Server Data Structures

This section describes how to add databases and tables to the Cisco Info Server and how to modify existing tables.

## Adding a New Database or Table

You can add databases and tables to the Cisco Info Server to complement the standard tables. For example, you might require a table that contains all of the alerts from a single host machine. You can use automation to populate this table with data from the existing **alerts.status** table.

You can create new tables in the custom database, which exists for this purpose. If you prefer, you can create a new database.

To add a new data structure, modify the SQL file for your Cisco Info Server as follows:

---

**Step 1** Stop the Cisco Info Server, as described in [Stopping a Cisco Info Server Manually, page 1-2](#).



**Note** If you create any tables or databases while the Cisco Info Server is running using the SQL interactive interface, these changes are lost when the Cisco Info Server is restarted.

---

**Step 2** Make a backup copy of the `$OMNIHOME/etc/<servername>.sql` SQL file.

where <servername> is the name of the Cisco Info Server. This allows you to roll back any changes if necessary.

**Step 3** Edit the SQL file. Add the **CREATE DATABASE** and **CREATE TABLE** commands at the end of the SQL file.

Do not modify the **CREATE DATABASE** and **CREATE TABLE** commands for existing data structures. The **CREATE DATABASE** and **CREATE TABLE** commands are described in [Data Definition Language Commands, page 2-14](#).

**Step 4** Save the SQL file.

**Step 5** Restart the system.

**Step 6** Check for any error messages when the Cisco Info Server restarts.

These messages can indicate errors in the modified SQL file.

---

## Modifying Database Tables

You can add or remove a column, or change the data type of a column, in an existing table. To change a table definition, modify the SQL file for your Cisco Info Server as follows:

**Step 1** Stop the Cisco Info Server, as described in [Stopping a Cisco Info Server Manually, page 1-2](#).



**Note** If you create any tables or databases while the Cisco Info Server is running using the SQL interactive interface, these changes are lost when the Cisco Info Server is restarted.

**Step 2** Make a backup of all `$OMNIHOME/db` database files.

The data in these files is converted to the new table definition as part of this procedure. The existing files are backed up with a `.orig` extension and the new files are created in their place. The `.bak` files in the database directory are not changed.

**Step 3** Copy the `$OMNIHOME/etc/<servername>.sql` file to the `$OMNIHOME/etc/<servername>.orig` file.

The `nco_migrate` command uses the original SQL file (`.orig`) to convert the database data to the format of the new SQL file (`.sql`), created in the next steps.

**Step 4** Edit the SQL file. To modify an existing table, locate the `CREATE TABLE` definition for the table and change the definition in parentheses ( ).



**Note** See [Changing the alerts.status Table, page 2-28](#) for restrictions on the changes you can make to the `alerts.status` table.

For details on available field types, see [Cisco Info Server Data Types, page 2-6](#).

**Step 5** Save the SQL file.

**Step 6** Migrate the existing data to the new database, as described in [Converting Data Using nco\\_migrate](#).

**Step 7** Restart the Cisco Info Server.

When region storage is enabled, use the `-fuzzy` option or set the `RegionStorage` property to `True` in the Cisco Info Server properties file.

For more information about logical and region storage, see [Checkpointing - Logical and Region Storage, page 2-28](#). The Cisco Info Server properties file is described in [Specifying Cisco Info Server Properties, page 1-7](#).

## Converting Data Using nco\_migrate

When columns have been added to or removed from an existing Cisco Info Server table, the existing data may not conform to the new data definition. The `nco_migrate` command takes the existing data, based on the definition in the original `$OMNIHOME/bin/nco_migrate -server <servername>.orig` SQL file and converts the data into the format based on the definition in the new `$OMNIHOME/bin/nco_migrate -server <servername>.sql` SQL file.

where `<servername>` is the Cisco Info Server name. If you do not specify the `-server` command line option, `NCOMS` is the default.

[Table 2-11](#) lists the command line options.

Table 2-11 *nco\_migrate* Command Line Options

Command Line Option	Description
<b>-defaults</b> <string>	Specifies the file containing default values, described in <a href="#">Default Files, page 2-27</a> .
<b>-fuzzy</b>	Specify this option to convert databases using region storage using <b>.tab</b> files. If this option is used on a Cisco Info Server that is running logical storage, the command is ignored.
<b>-help</b>	Displays help information about the command line options and exits.
<b>-orig</b> <string>	Specifies the name of the file that contains the old SQL definition with which the database was created. The default is <b>servername.orig</b> . The option <b>-old</b> can be used as an alias.
<b>-server</b> <string>	Specifies the name of the server to be converted. The default is <b>NCOMS</b> .
<b>-sql</b> <string>	Specifies the name of the file that contains the new SQL definition to which to convert. The default is <b>servername.sql</b> . The option <b>-new</b> can be used as an alias.
<b>-user</b> <string>	Specifies the name of a Cisco Info Center user. If you do not specify a user name, the default is the user running the command.
<b>-verbose</b>	Displays detailed information during the conversion.
<b>-version</b>	Displays version information and exits.

## Default Files

To automatically assign values to new columns, specify the **-defaults** option followed by a file name. Create file entries in the format:

```
<databasename>.<tablename>.<fieldname>=<value>
```

where <databasename> is the name of the database, <tablename> is the name of the table, <fieldname> is the name of the field to set, and <value> is the value to set it to. The format of <value> can be a quoted string or an integer value depending on the type of the field being set. For example, if a field called **HtmlPage** is added to the **alerts.status** table, existing entries should have this field set to **http://www.random.org/**, and the defaults file should contain:

```
alerts.status.HtmlPage="http://www.random.org/"
```



### Note

If you specify an existing field in the defaults file, any data stored in that field is overwritten by the default setting.

## Changing the alerts.status Table

This section describes the changes that can be made to the **alerts.status** table. The overall procedure for modifying Cisco Info Server tables is described in [Adding a New Database or Table, page 2-25](#).



### Note

You cannot remove mandatory columns from the **alerts.status** table. Mandatory columns are identified in [alerts.status Table, page A-1](#).

## Adding Columns to the alerts.status Table

You can add new columns to the **alerts.status** table. For example, you might require a column named **Customer**, which contains the name of the customer who uses the device that produced the alert. Another new column might contain a URL that indicates where the online documentation for a device exists on the company intranet.

You must add the new column after the standard columns in the table definition. Additional columns can be populated using instructions in the Cisco Info Mediator rules files or during the execution of automations.

## Changing String Character Data Types

You can change columns that use the **varchar** data type to the **char** data type if required. See [Cisco Info Server Data Types, page 2-6](#) for the differences between the **varchar** and **char** data types.



### Note

Do not change the data types of primary keys. Primary keys that consist of character strings should always use the **varchar** data type rather than the **char** type. Because primary keys are never updated, performance is not an issue.

Do not modify mandatory columns that use other data types.

# Checkpointing - Logical and Region Storage

Cisco Info Server data is stored in memory for high speed access. Checkpoints and logging between checkpoints copy the data in memory to disk. This enables you to restore the data after a planned or unexpected shutdown occurs.

During a checkpoint, all Cisco Info Server data is copied to files on disk. Between checkpoints, new and changed data is written to replay log files. The following types of files are maintained on disk:

- checkpoint files, containing the data for entire tables
- replay logs, containing the changes made to tables since the last checkpoint.

Checkpointing takes place at specified time intervals, set using the **LogInterval** property or the **-interval** command line option. The default time between checkpoints is **300** seconds. Checkpoints also occur each time there is a planned Cisco Info Server shutdown.

When a checkpoint occurs, each table is copied to a checkpoint file. If there is an associated replay log, it is deleted.

When a Cisco Info Server is started, the database is restored by loading the most recent checkpoint files from disk. When recovering from an unexpected shutdown, the Cisco Info Server loads both the checkpoint files and the replay logs.

The following types of storage are available for checkpoint and replay log files:

- logical storage, using logical checkpoint and replay log files
- region storage, using physical checkpoint and replay log files.

Each storage type uses a different procedure, as described in the following sections.

## Logical (Default) Storage

When you use logical storage, the rows of each database table are written to the checkpoint file as SQL commands. A checkpoint file is created on disk for each database table. The checkpoint file is named:

`<servername>.<databasename>.<tablename>.dat`

Before writing to the `.dat` file, the existing file is copied to a backup file named:

`<servername>.<databasename>.<tablename>.bak`

Replay logs are only created for the **alerts.status** and **alerts.journal** tables. The changes made to these tables between checkpoints (because the `.dat` file was created) are added, as SQL commands, to files named:

`<servername>.<databasename>.<tablename>.trans`

For example, in the **NCOMS** Cisco Info Server, the **alerts.status** table replay log file is named:

**NCOMS.alerts.status.trans**

When the Cisco Info Server restarts, the database is recreated by executing the SQL commands in the `.dat` file. If a `.trans` file exists for a table, the SQL commands in this file are also executed. The `.bak` file is only used if the `.dat` file has been corrupted during an unexpected shutdown.

## Region Storage

When you use region storage, two checkpoint files are generated for each database table in memory. The checkpoint files are named:

`<servername>.<databasename>.<tablename>.chk0`

`<servername>.<databasename>.<tablename>.chk1`

Two replay logs are also generated for each table. The files are named:

`<servername>.<databasename>.<tablename>.log0`

`<servername>.<databasename>.<tablename>.log1`

The checkpoint process writes alternately to the `.chk0` and `.chk1` files. If one file has become corrupted during an unexpected shutdown, the data in the other checkpoint file and the replay logs is used to rebuild the database tables in memory. As each checkpoint starts, the logging process switches to the alternate log file. The older log file is deleted before the start of the next checkpoint.

When a planned Cisco Info Server shutdown occurs (because the **shutdown** command is executed), a table file is created for each table, named:

`<servername>.<databasename>.<tablename>.tab`

For example, in the **NCOMS** Cisco Info Server, the **alerts.status** table file is named:

**NCOMS.alerts.status.tab**

The `.tab` file format is specific to the hardware and operating system platform on which it was created.

When the Cisco Info Server is restarted after a planned shutdown, the database is rebuilt using the table files.

When the Cisco Info Server is restarted after an unexpected shutdown, the database is rebuilt using the checkpoint and log files.

## Checkpointing in a Production Environment

Region storage is the preferred method of checkpointing for production environments.

When logical storage is used, the tables in memory are locked during checkpoints. This can cause Desktops or Cisco Info Mediators to temporarily stall if they are attempting to update a table that is locked. The update occurs once the checkpoint is complete and the lock is released.

When region storage is used, tables are not locked during checkpointing.

Region storage also has a performance advantage over logical storage when the Cisco Info Server contains large database tables because the format of the tables means they can be copied to disk more quickly.

## Region Storage Configuration

To enable region storage, set the **RegionStorage** property to **True** in the Cisco Info Server properties file. You must shut down the Cisco Info Server before changing this property.

**Note**

You must convert the checkpoint and log files before the Cisco Info Server can start correctly, as described next.

## Converting from Logical Storage to Region Storage

To convert the **.dat** files to **.tab** files, enter the command **nco\_objserv -dattofuzzy** while the Cisco Info Server is shut down.

The **-dattofuzzy** command line option is described in [Specifying Cisco Info Server Command Line Options, page 1-3](#).

## Converting from Region Storage to Logical Storage

There are two ways to change from region storage to logical storage:

### Conversion When the Cisco Info Server Is Already Shut Down

To convert the **.tab** files to **.dat** files without starting the Cisco Info Server, use the command:

**nco\_objserv -fuzzytodat.**

The **-fuzzytodat** command line option is described in [Specifying Cisco Info Server Command Line Options, page 1-3](#).

### Conversion During Cisco Info Server Shutdown

Log in to the Cisco Info Server using the SQL interactive interface. For details see [Direct Access Using the SQL Interactive Interface \(nco\\_sql\)](#), page 2-4.

Shut down the Cisco Info Server using the following commands:

```
1> shutdown dump;
```

```
2> go
```

The Cisco Info Server is shut down and **.dat** checkpoint files are generated instead of **.tab** files.

