

# Creating a Query

- [NOTE](#)
- [Introduction](#)
- [How Does It Work?](#)
  - [The SELECT](#)
  - [The FROM](#)
  - [The WHERE](#)
- [Database Schema](#)
- [Examples](#)
  - [The Elevated User query](#)
  - [Devices Older Than X](#)
  - [Devices with Expired Warranties](#)
  - [Devices Missing Information](#)
  - [List All NMAP Ports, Protocols and Programs for Each Device](#)
  - [Return a list of all Modules installed on Routers](#)

## NOTE

As at Open-Audit 5.x, we no longer have a 'system' table. We now use 'devices'.

Our foreign keys are no longer named \$table.system\_id, rather \$table.device\_id.

## Introduction

Queries are an important part of Open-Audit. They are what enables you to get meaningful information from all the device data you have discovered. Along with summaries and groups, they provide a powerful mechanism to extract crucial information.

## How Does It Work?

A query is essentially a SQL statement. This statement is run against the database with the automatic addition of the limit, filtered to apply to only those requested items and only those items the user has permission to view. A Query can be created using menu -> Manage -> Queries -> Create Queries. Queries contain an org\_id and are hence restricted to the appropriate users. A user must have the org\_admin or reporter role(s) to create, update or delete a query. All users can execute a query. A query has a name attribute used for the menu item as well as a menu category attribute. This tells the Open-Audit GUI which submenu to place the query in. There is also menu display which should be set to 'y' to enable the query in the GUI ('n' to prevent the query from appearing at all). The query would still run if called using it's \$id, regardless of menu display's value.

To view the details of a query, the standard URL structure of /open-audit/index.php/queries/{ \$id } should be used.

To actually execute the query, append a /execute, thus /open-audit/index.php/queries/{ \$id }/execute.

## Breaking it Down

The SQL query is essentially broken into three parts.

### The SELECT

The SELECT section of the query should use full dot notation and also request the field with it's full dot name. IE - SELECT system.id AS `system.id`. Each field should be selected in this fashion to enable GUI side attribute filtering.

```
SELECT devices.id AS `devices.id`, devices.icon AS `devices.icon`, devices.type AS `devices.type`,  
sysdevicestem.name AS `devices.name`, devices.domain AS `devices.domain`, devices.ip AS `devices.ip`,  
user_group.name as `user_group.name`, user_group.members AS `user_group.members`
```

### The FROM

You should use only those tables that contain attributes you need. I usually use a LEFT JOIN. IE - SELECT devices.id AS `devices.id`, ip.ip AS `ip.ip` FROM devices LEFT JOIN ip ON (devices.id = ip.device\_id).

All device sub-tables contain a couple of important columns. \$table.device\_id and \$table.current. \$table.device\_id is the link to the devices system.id column. The \$table.current column will contain either 'y' or 'n'. This indicates if this row is currently present on the device. For example software may be installed (which would result in software.current = 'y'), but on a subsequent audit it may not be detected. Open-Audit will then change this rows current attribute to 'n'.

```
FROM user_group LEFT JOIN devices ON (user_group.device_id = devices.id AND user_group.current = 'y')
```

## The WHERE

In order for Open-Audit to be able to apply user permissions on items, we mandate the user of WHERE @filter. If you do not use this format, the query::create form will throw a warning. Only users with the Admin role are permitted to create queries that lack this attribute.

Other than that restriction, you are free to select attributes as required. It's a good idea to use the menu -> Admin -> Database -> List Tables item to view the specific tables and their columns. This will enable you to find exactly what you need, rather than trawling through the MySQL console of the schema creation script.

```
WHERE @filter AND (user_group.name = 'Administrators' OR user_group.name = 'Power Users' OR user_group.name = 'Remote Desktop Users' OR user_group.name = 'wheel' OR user_group.name = 'sudo') AND user_group.members > '' GROUP BY devices.id, user_group.name ORDER BY devices.name
```

## Database Schema

The database schema can be found in the application if the user has database::read permission by going to menu: Admin -> Database -> List Tables, then clicking on the details button for the table. Device details are stored in the **devices** table.

## Examples

NOTE - The SQL queries used in Open-Audit require the use of the backtick - ` character and NOT the standard single quote for fields. On most US Windows keyboards the backtick key is located in the top-left of the keyboard along with the tilde ~. On a US Mac keyboard the backtick key is located next to the SHIFT key. The standard single quote is still used to enclose values as the examples below illustrate.

### The Elevated User query

```
SELECT devices.id AS `devices.id`, devices.icon AS `devices.icon`, devices.type AS `devices.type`, devices.name AS `devices.name`, devices.domain AS `devices.domain`, devices.ip AS `devices.ip`, user_group.name as `user_group.name`, user_group.members AS `user_group.members` FROM user_group LEFT JOIN devices ON (user_group.device_id = devices.id AND user_group.current = 'y') WHERE @filter AND (user_group.name = 'Administrators' OR user_group.name = 'Power Users' OR user_group.name = 'Remote Desktop Users' OR user_group.name = 'wheel' OR user_group.name = 'sudo') AND user_group.members > '' GROUP BY devices.id, user_group.name ORDER BY devices.name
```

### Devices Older Than X

This example query retrieves a list of devices OVER 3 years old. The query uses today (NOW) and system.purchase\_date as the reference point and filters out all virtual machines via a check of the system.serial field for %VM%.

```
SELECT devices.id AS `devices.id`, devices.purchase_date AS `devices.purchase_date`, devices.type AS `devices.type`, devices.name AS `devices.name`, devices.last_seen AS `devices.last_seen`, devices.manufacturer AS `devices.manufacturer`, devices.model AS `devices.model`, devices.description AS `devices.description`, devices.function AS `devices.function`, locations.name AS `locations.name` FROM devices LEFT JOIN locations ON (devices.location_id = locations.id) LEFT JOIN windows ON (devices.id = windows.device_id AND windows.current = 'y') LEFT JOIN orgs ON (devices.org_id = orgs.id) WHERE @filter AND devices.purchase_date < DATE_SUB(NOW(),INTERVAL 3 YEAR) AND devices.serial NOT LIKE '%VM%'
```

### Devices with Expired Warranties

This example uses system.warranty\_expires and looks for a warranty expiration date prior to today.

```
SELECT devices.id AS `devices.id`, devices.warranty_expires AS `devices.warranty_expires`, devices.type AS
`devices.type`, devices.name AS `devices.name`, devices.last_seen AS `devices.last_seen`, devices.manufacturer
AS `devices.manufacturer`, devices.model AS `devices.model`,
devices.description AS `devices.description`, devices.function AS `devices.function`, locations.name AS
`locations.name` FROM devices LEFT JOIN locations ON (devices.location_id = locations.id) LEFT JOIN windows ON
(devices.id = windows.device_id AND windows.current = 'y') LEFT JOIN orgs ON (devices.org_id = orgs.id) WHERE
@filter AND devices.warranty_expires <= CURDATE() AND devices.serial NOT LIKE '%VM%'
```

## Devices Missing Information

This example creates a list of devices where the Function or Description fields are blank OR the Purchase Date is the default.

```
SELECT devices.id AS `devices.id`, devices.ip AS `devices.ip`, devices.name AS `devices.name`, devices.
description AS `devices.description`, devices.function AS `devices.function`, devices.purchase_date AS `devices.
purchase_date`, devices.type AS `devices.type`, locations.name AS `locations.name` FROM system LEFT JOIN
locations ON (devices.location_id = locations.id) WHERE @filter AND devices.purchase_date = '2000-01-01' OR
devices.function = '' OR devices.description = ''
```

## List All NMAP Ports, Protocols and Programs for Each Device

This example creates a list of devices and the open Ports, Protocols, and Programs found by the NMAP scan.

```
SELECT devices.id AS `devices.id`, devices.type AS `devices.type`, devices.name AS `devices.name`, devices.
domain AS `devices.domain`, devices.ip AS `devices.ip`, nmap.first_seen AS `nmap.first_seen`, nmap.last_seen AS
`nmap.last_seen`, nmap.port AS `nmap.port`, nmap.protocol AS `nmap.protocol`, nmap.program AS `nmap.program`
FROM nmap LEFT JOIN devices ON (nmap.device_id = devices.id) WHERE @filter
```

## Return a list of all Modules installed on Routers

This example creates a list of all Modules marked as current='y' on devices of type 'router'

```
SELECT devices.id AS `devices.id`, devices.type AS `devices.type`, devices.name AS `devices.name`, devices.
manufacturer AS `devices.manufacturer`, devices.model AS `devices.model`, devices.serial AS `devices.serial`,
module.description AS `module.description` FROM devices LEFT JOIN module ON (module.device_id = devices.id AND
module.current = 'y') WHERE @filter AND devices.type = 'router' ORDER BY devices.name
```