In short, osquery turns the metadata and state information of your local system (or remote system(s)) into a SQL-compliant database. It also works on Windows, Linux, BSD and macOS. This means you can query a fleet of systems with a (mostly) normalized set of tables and get aggregated results. Operations and information security staff use this to manage systems and perform incident response tasks, but you can use it to get just about anything and there are even more powerful modes of operation for osquery. But, more on all the features of osquery[r] in another post.

If you are skeptical, here’s some proof (which I need to show regardless of your skepticism state). First, a local “connection”:

library(DBI)

library(osqueryr)

con <- DBI::dbConnect(Osquery())

head(dbListTables(con), 10)

## [1] "account\_policy\_data" "acpi\_tables" "ad\_config"

## [4] "alf" "alf\_exceptions" "alf\_explicit\_auths"

## [7] "alf\_services" "app\_schemes" "apps"

## [10] "apt\_sources"

dbListFields(con, "processes")

## [1] "cmdline" "cwd" "disk\_bytes\_read"

## [4] "disk\_bytes\_written" "egid" "euid"

## [7] "gid" "name" "nice"

## [10] "on\_disk" "parent" "path"

## [13] "pgroup" "pid" "resident\_size"

## [16] "root" "sgid" "start\_time"

## [19] "state" "suid" "system\_time"

## [22] "threads" "total\_size" "uid"

## [25] "user\_time" "wired\_size"

dbGetQuery(con, "SELECT name, system\_time FROM processes WHERE name LIKE '%fire%'")

## # A tibble: 2 x 2

## name system\_time

## 1 Firewall 3

## 2 firefox 517846

then, a remote "connection":

con2 <- osqueryr::dbConnect(Osquery(), host = "hrbrmstr@osq1")

head(dbListTables(con2), 10)

## [1] "account\_policy\_data" "acpi\_tables" "ad\_config"

## [4] "alf" "alf\_exceptions" "alf\_explicit\_auths"

## [7] "alf\_services" "app\_schemes" "apps"

## [10] "apt\_sources"

dbListFields(con2, "processes")

## [1] "cmdline" "cwd" "disk\_bytes\_read"

## [4] "disk\_bytes\_written" "egid" "euid"

## [7] "gid" "name" "nice"

## [10] "on\_disk" "parent" "path"

## [13] "pgroup" "pid" "resident\_size"

## [16] "root" "sgid" "start\_time"

## [19] "state" "suid" "system\_time"

## [22] "threads" "total\_size" "uid"

## [25] "user\_time" "wired\_size"

dbGetQuery(con2, "SELECT name, system\_time FROM processes WHERE name LIKE '%fire%'")

## # A tibble: 1 x 2

## name system\_time

## 1 firefox 1071992

**"You're talking an *awful* lot about the package when you said this was a post on 'standards' and 'consistency'."**

True, but we needed that bit above for context. To explain what this post has to do with "standards" and "consistency" I also need to tell you a bit more about *how* both osquery and the osqueryr package are implemented.

You can read about osquery in-depth starting at the link at the top of this post, but the authors of the tool *really* wanted a consistent idiom for accessing system metadata with usable, normalized output. They chose (to use a word they didn't but one that works for an R audience) a "data frame" as the output format and picked the universal language of "data frames" -- SQL -- as the inquiry interface. So, *right there* are examples of both standards and consistency: using SQL vs coming up with yet-another-query-language and avoiding the chaos of the myriad of outputs from various system commands by making all results conform to a rectangular data structure.

Let's take this one-step further with a specific example. All modern operating systems have the concept of a "process" and said processes have (mostly) similar attributes. However, the commands used to get a detailed listing of those processes differ (sometimes wildly) from OS to OS. The osquery came up with a set of schemas to ensure a common, rectangular output and naming conventions (note that some schemas are unique to a particular OS since some elements of operating systems have no useful counterparts on other operating systems).

Osquery Schemas

**account\_policy\_data NEW**

Additional OS X user account data from the AccountPolicy section of OpenDirectory.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | User ID |
| creation\_time | DOUBLE | When the account was first created |
| failed\_login\_count | BIGINT | The number of times the user failed to login with the correct password. Resets after a correct password is entered |
| failed\_login\_timestamp | DOUBLE | The time of the last failed login attempt. Resets after a correct password is entered |
| password\_last\_set\_time | DOUBLE | The time the password was last changed |

**acpi\_tables**

Firmware ACPI functional table common metadata and content.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | ACPI table name |
| size | INTEGER | Size of compiled table data |
| md5 | TEXT | MD5 hash of table content |

**ad\_config**

OS X Active Directory configuration.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | The OS X-specific configuration name |
| domain | TEXT | Active Directory trust domain |
| option | TEXT | Canonical name of option |
| value | TEXT | Variable typed option value |

**alf**

OS X application layer firewall (ALF) service details.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| allow\_signed\_enabled | INTEGER | 1 If allow signed mode is enabled else 0 |
| firewall\_unload | INTEGER | 1 If firewall unloading enabled else 0 |
| global\_state | INTEGER | 1 If the firewall start by default else 0 |
| logging\_enabled | INTEGER | 1 If logging mode is enabled else 0 |
| logging\_option | INTEGER | Firewall logging option |
| stealth\_enabled | INTEGER | 1 If stealth mode is enabled else 0 |
| version | TEXT | Application Layer Firewall version |

**alf\_exceptions**

OS X application layer firewall (ALF) service exceptions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Path to the executable that is excepted |
| state | INTEGER | Firewall exception state |

**alf\_explicit\_auths**

ALF services explicitly allowed to perform networking.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| process | TEXT | Process name explicitly allowed |

**alf\_services**

OS X application layer firewall (Firewall) services.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| service | TEXT | Firewalled service name |
| process | TEXT | Process name |
| state | INTEGER | Firewall service state |

**app\_schemes**

OS X application schemes and handlers (e.g., http, file, mailto).

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| scheme | TEXT | Name of the scheme/protocol |
| handler | TEXT | Application label for the handler |
| enabled | INTEGER | 1 if this handler is the OS default, else 0 |
| external | INTEGER | 1 if this handler does NOT exist on OS X by default, else 0 |
| protected | INTEGER | 1 if this handler is protected (reserved) by OS X, else 0 |

**appcompat\_shims**

Application Compatibility shims are a way to persist malware. This table presents the AppCompat Shim information from the registry in a nice format. See http://files.brucon.org/2015/Tomczak\_and\_Ballenthin\_Shims\_for\_the\_Win.pdf for more details.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| executable | TEXT | Name of the executable that is being shimmed. This is pulled from the registry. |
| path | TEXT | This is the path to the SDB database. |
| description | TEXT | Description of the SDB. |
| install\_time | INTEGER | Install time of the SDB |
| type | TEXT | Type of the SDB database. |
| sdb\_id | TEXT | Unique GUID of the SDB. |

**apps**

OS X applications installed in known search paths (e.g., /Applications).

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Name of the Name.app folder |
| path | TEXT | Absolute and full Name.app path |
| bundle\_executable | TEXT | Info properties CFBundleExecutable label |
| bundle\_identifier | TEXT | Info properties CFBundleIdentifier label |
| bundle\_name | TEXT | Info properties CFBundleName label |
| bundle\_short\_version | TEXT | Info properties CFBundleShortVersionString label |
| bundle\_version | TEXT | Info properties CFBundleVersion label |
| bundle\_package\_type | TEXT | Info properties CFBundlePackageType label |
| environment | TEXT | Application-set environment variables |
| element | TEXT | Does the app identify as a background agent |
| compiler | TEXT | Info properties DTCompiler label |
| development\_region | TEXT | Info properties CFBundleDevelopmentRegion label |
| display\_name | TEXT | Info properties CFBundleDisplayName label |
| info\_string | TEXT | Info properties CFBundleGetInfoString label |
| minimum\_system\_version | TEXT | Minimum version of OS X required for the app to run |
| category | TEXT | The UTI that categorizes the app for the App Store |
| applescript\_enabled | TEXT | Info properties NSAppleScriptEnabled label |
| copyright | TEXT | Info properties NSHumanReadableCopyright label |
| last\_opened\_time | DOUBLE | The time that the app was last used |

**apt\_sources**

Current list of APT repositories or software channels.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Repository name |
| source | TEXT | Source file |
| base\_uri | TEXT | Repository base URI |
| release | TEXT | Release name |
| version | TEXT | Repository source version |
| maintainer | TEXT | Repository maintainer |
| components | TEXT | Repository components |
| architectures | TEXT | Repository architectures |

**arp\_cache**

Address resolution cache, both static and dynamic (from ARP, NDP).

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| address | TEXT | IPv4 address target |
| mac | TEXT | MAC address of broadcasted address |
| interface | TEXT | Interface of the network for the MAC |
| permanent | TEXT | 1 for true, 0 for false |

**asl**

Queries the Apple System Log data structure for system events.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| time | INTEGER | Unix timestamp. Set automatically |
| time\_nano\_sec | INTEGER | Nanosecond time. |
| host | TEXT | Sender's address (set by the server). |
| sender | TEXT | Sender's identification string. Default is process name. |
| facility | TEXT | Sender's facility. Default is 'user'. |
| pid | INTEGER | Sending process ID encoded as a string. Set automatically. |
| gid | BIGINT | GID that sent the log message (set by the server). |
| uid | BIGINT | UID that sent the log message (set by the server). |
| level | INTEGER | Log level number. See levels in asl.h. |
| message | TEXT | Message text. |
| ref\_pid | INTEGER | Reference PID for messages proxied by launchd |
| ref\_proc | TEXT | Reference process for messages proxied by launchd |
| extra | TEXT | Extra columns, in JSON format. Queries against this column are performed entirely in SQLite, so do not benefit from efficient querying via asl.h. |

**augeas**

Configuration files parsed by augeas.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| node | TEXT | The node path of the configuration item |
| value | TEXT | The value of the configuration item |
| label | TEXT | The label of the configuration item |
| path | TEXT | The path to the configuration file |

**authenticode**

File (executable, bundle, installer, disk) code signing status.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Must provide a path or directory |
| original\_program\_name | TEXT | The original program name that the publisher has signed |
| serial\_number | TEXT | The certificate serial number |
| issuer\_name | TEXT | The certificate issuer name |
| subject\_name | TEXT | The certificate subject name |
| result | TEXT | The signature check result |

**authorization\_mechanisms**

OS X Authorization mechanisms database.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| label | TEXT | Label of the authorization right |
| plugin | TEXT | Authorization plugin name |
| mechanism | TEXT | Name of the mechanism that will be called |
| privileged | TEXT | If privileged it will run as root, else as an anonymous user |
| entry | TEXT | The whole string entry |

**authorizations**

OS X Authorization rights database.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| label | TEXT | Item name, usually in reverse domain format |
| modified | TEXT | Label top-level key |
| allow\_root | TEXT | Label top-level key |
| timeout | TEXT | Label top-level key |
| version | TEXT | Label top-level key |
| tries | TEXT | Label top-level key |
| authenticate\_user | TEXT | Label top-level key |
| shared | TEXT | Label top-level key |
| comment | TEXT | Label top-level key |
| created | TEXT | Label top-level key |
| class | TEXT | Label top-level key |
| session\_owner | TEXT | Label top-level key |

**authorized\_keys**

A line-delimited authorized\_keys table.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | The local owner of authorized\_keys file |
| algorithm | TEXT | algorithim of key |
| key | TEXT | parsed authorized keys line |
| key\_file | TEXT | Path to the authorized\_keys file |

**autoexec**

Aggregate of executables that will automatically execute on the target machine. This is an amalgamation of other tables like services, scheduled tasks, startup\_items and more.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Path to the executable |
| name | TEXT | Name of the program |
| source | TEXT | Source table of the autoexec item |

**bitlocker\_info NEW**

Retrieve bitlocker status of the machine.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| device\_id | TEXT | ID of the encrypted drive. |
| drive\_letter | TEXT | Drive letter of the encrypted drive. |
| persistent\_volume\_id | TEXT | Persistent ID of the drive. |
| conversion\_status | INTEGER | The bitlocker conversion status of the drive. |
| protection\_status | INTEGER | The bitlocker protection status of the drive. |

**block\_devices**

Block (buffered access) device file nodes: disks, ramdisks, and DMG containers.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Block device name |
| parent | TEXT | Block device parent name |
| vendor | TEXT | Block device vendor string |
| model | TEXT | Block device model string identifier |
| size | BIGINT | Block device size in blocks |
| block\_size | INTEGER | Block size in bytes |
| uuid | TEXT | Block device Universally Unique Identifier |
| type | TEXT | Block device type string |
| label | TEXT | Block device label string |

**browser\_plugins**

All C/NPAPI browser plugin details for all users.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | The local user that owns the plugin |
| name | TEXT | Plugin display name |
| identifier | TEXT | Plugin identifier |
| version | TEXT | Plugin short version |
| sdk | TEXT | Build SDK used to compile plugin |
| description | TEXT | Plugin description text |
| development\_region | TEXT | Plugin language-localization |
| native | INTEGER | Plugin requires native execution |
| path | TEXT | Path to plugin bundle |
| disabled | INTEGER | Is the plugin disabled. 1 = Disabled |

**carbon\_black\_info**

Returns info about a Carbon Black sensor install.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| sensor\_id | INTEGER | Sensor ID of the Carbon Black sensor |
| config\_name | TEXT | Sensor group |
| collect\_store\_files | INTEGER | If the sensor is configured to send back binaries to the Carbon Black server |
| collect\_module\_loads | INTEGER | If the sensor is configured to capture module loads |
| collect\_module\_info | INTEGER | If the sensor is configured to collect metadata of binaries |
| collect\_file\_mods | INTEGER | If the sensor is configured to collect file modification events |
| collect\_reg\_mods | INTEGER | If the sensor is configured to collect registry modification events |
| collect\_net\_conns | INTEGER | If the sensor is configured to collect network connections |
| collect\_processes | INTEGER | If the sensor is configured to process events |
| collect\_cross\_processes | INTEGER | If the sensor is configured to cross process events |
| collect\_emet\_events | INTEGER | If the sensor is configured to EMET events |
| collect\_data\_file\_writes | INTEGER | If the sensor is configured to collect non binary file writes |
| collect\_process\_user\_context | INTEGER | If the sensor is configured to collect the user running a process |
| collect\_sensor\_operations | INTEGER | Unknown |
| log\_file\_disk\_quota\_mb | INTEGER | Event file disk quota in MB |
| log\_file\_disk\_quota\_percentage | INTEGER | Event file disk quota in a percentage |
| protection\_disabled | INTEGER | If the sensor is configured to report tamper events |
| sensor\_ip\_addr | TEXT | IP address of the sensor |
| sensor\_backend\_server | TEXT | Carbon Black server |
| event\_queue | INTEGER | Size in bytes of Carbon Black event files on disk |
| binary\_queue | INTEGER | Size in bytes of binaries waiting to be sent to Carbon Black server |

**carves**

Forensic Carves.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| time | BIGINT | Time at which the carve was kicked off |
| sha256 | TEXT | A SHA256 sum of the carved archive |
| size | INTEGER | Size of the carved archive |
| path | TEXT | The path of the requested carve |
| status | TEXT | Status of the carve, can be STARTING, PENDING, SUCCESS, or FAILED |
| carve\_guid | TEXT | Identifying value of the carve session |
| carve | INTEGER | Set this value to '1' to start a file carve |

**certificates**

Certificate Authorities installed in Keychains/ca-bundles.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| common\_name | TEXT | Certificate CommonName |
| subject | TEXT | Certificate distinguished name |
| issuer | TEXT | Certificate issuer distinguished name |
| ca | INTEGER | 1 if CA: true (certificate is an authority) else 0 |
| self\_signed | INTEGER | 1 if self-signed, else 0 |
| not\_valid\_before | TEXT | Lower bound of valid date |
| not\_valid\_after | TEXT | Certificate expiration data |
| signing\_algorithm | TEXT | Signing algorithm used |
| key\_algorithm | TEXT | Key algorithm used |
| key\_strength | TEXT | Key size used for RSA/DSA, or curve name |
| key\_usage | TEXT | Certificate key usage and extended key usage |
| subject\_key\_id | TEXT | SKID an optionally included SHA1 |
| authority\_key\_id | TEXT | AKID an optionally included SHA1 |
| sha1 | TEXT | SHA1 hash of the raw certificate contents |
| path | TEXT | Path to Keychain or PEM bundle |
| serial | TEXT | Certificate serial number |

**chocolatey\_packages**

Chocolatey packages installed in a system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Package display name |
| version | TEXT | Package-supplied version |
| summary | TEXT | Package-supplied summary |
| author | TEXT | Optional package author |
| license | TEXT | License under which package is launched |
| path | TEXT | Path at which this package resides |

**chrome\_extensions**

Chrome browser extensions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | The local user that owns the extension |
| name | TEXT | Extension display name |
| identifier | TEXT | Extension identifier |
| version | TEXT | Extension-supplied version |
| description | TEXT | Extension-optional description |
| locale | TEXT | Default locale supported by extension |
| update\_url | TEXT | Extension-supplied update URI |
| author | TEXT | Optional extension author |
| persistent | INTEGER | 1 If extension is persistent across all tabs else 0 |
| path | TEXT | Path to extension folder |

**cpu\_time**

Displays information from /proc/stat file about the time the cpu cores spent in different parts of the system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| core | INTEGER | Name of the cpu (core) |
| user | BIGINT | Time spent in user mode |
| nice | BIGINT | Time spent in user mode with low priority (nice) |
| system | BIGINT | Time spent in system mode |
| idle | BIGINT | Time spent in the idle task |
| iowait | BIGINT | Time spent waiting for I/O to complete |
| irq | BIGINT | Time spent servicing interrupts |
| softirq | BIGINT | Time spent servicing softirqs |
| steal | BIGINT | Time spent in other operating systems when running in a virtualized environment |
| guest | BIGINT | Time spent running a virtual CPU for a guest OS under the control of the Linux kernel |
| guest\_nice | BIGINT | Time spent running a niced guest |

**cpuid**

Useful CPU features from the cpuid ASM call.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| feature | TEXT | Present feature flags |
| value | TEXT | Bit value or string |
| output\_register | TEXT | Register used to for feature value |
| output\_bit | INTEGER | Bit in register value for feature value |
| input\_eax | TEXT | Value of EAX used |

**crashes**

Application, System, and Mobile App crash logs.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| type | TEXT | Type of crash log |
| pid | BIGINT | Process (or thread) ID of the crashed process |
| path | TEXT | Path to the crashed process |
| crash\_path | TEXT | Location of log file |
| identifier | TEXT | Identifier of the crashed process |
| version | TEXT | Version info of the crashed process |
| parent | BIGINT | Parent PID of the crashed process |
| responsible | TEXT | Process responsible for the crashed process |
| uid | INTEGER | User ID of the crashed process |
| datetime | TEXT | Date/Time at which the crash occurred |
| crashed\_thread | BIGINT | Thread ID which crashed |
| stack\_trace | TEXT | Most recent frame from the stack trace |
| exception\_type | TEXT | Exception type of the crash |
| exception\_codes | TEXT | Exception codes from the crash |
| exception\_notes | TEXT | Exception notes from the crash |
| registers | TEXT | The value of the system registers |

**crontab**

Line parsed values from system and user cron/tab.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| event | TEXT | The job @event name (rare) |
| minute | TEXT | The exact minute for the job |
| hour | TEXT | The hour of the day for the job |
| day\_of\_month | TEXT | The day of the month for the job |
| month | TEXT | The month of the year for the job |
| day\_of\_week | TEXT | The day of the week for the job |
| command | TEXT | Raw command string |
| path | TEXT | File parsed |

**curl**

Perform an http request and return stats about it.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| url | TEXT | The url for the request |
| method | TEXT | The HTTP method for the request |
| user\_agent | TEXT | The user-agent string to use for the request |
| response\_code | INTEGER | The HTTP status code for the response |
| round\_trip\_time | BIGINT | Time taken to complete the request |
| bytes | BIGINT | Number of bytes in the response |
| result | TEXT | The HTTP response body |

**curl\_certificate**

Inspect TLS certificates by connecting to input hostnames.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| hostname | TEXT | Hostname (domain[:port]) to CURL |
| common\_name | TEXT | Common name of company issued to |
| organization | TEXT | Organization issued to |
| organization\_unit | TEXT | Organization unit issued to |
| serial\_number | TEXT | Certificate serial number |
| issuer\_common\_name | TEXT | Issuer common name |
| issuer\_organization | TEXT | Issuer organization |
| issuer\_organization\_unit | TEXT | Issuer organization unit |
| valid\_from | TEXT | Period of validity start date |
| valid\_to | TEXT | Period of validity end date |
| sha256\_fingerprint | TEXT | SHA-256 fingerprint |
| sha1\_fingerprint | TEXT | SHA1 fingerprint |

**deb\_packages**

The installed DEB package database.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Package name |
| version | TEXT | Package version |
| source | TEXT | Package source |
| size | BIGINT | Package size in bytes |
| arch | TEXT | Package architecture |
| revision | TEXT | Package revision |

**device\_file**

Similar to the file table, but use TSK and allow block address access.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| device | TEXT | Absolute file path to device node |
| partition | TEXT | A partition number |
| path | TEXT | A logical path within the device node |
| filename | TEXT | Name portion of file path |
| inode | BIGINT | Filesystem inode number |
| uid | BIGINT | Owning user ID |
| gid | BIGINT | Owning group ID |
| mode | TEXT | Permission bits |
| size | BIGINT | Size of file in bytes |
| block\_size | INTEGER | Block size of filesystem |
| atime | BIGINT | Last access time |
| mtime | BIGINT | Last modification time |
| ctime | BIGINT | Creation time |
| hard\_links | INTEGER | Number of hard links |
| type | TEXT | File status |

**device\_firmware**

A best-effort list of discovered firmware versions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| type | TEXT | Type of device |
| device | TEXT | The device name |
| version | TEXT | Firmware version |

**device\_hash**

Similar to the hash table, but use TSK and allow block address access.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| device | TEXT | Absolute file path to device node |
| partition | TEXT | A partition number |
| inode | BIGINT | Filesystem inode number |
| md5 | TEXT | MD5 hash of provided inode data |
| sha1 | TEXT | SHA1 hash of provided inode data |
| sha256 | TEXT | SHA256 hash of provided inode data |

**device\_partitions**

Use TSK to enumerate details about partitions on a disk device.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| device | TEXT | Absolute file path to device node |
| partition | INTEGER | A partition number or description |
| label | TEXT |  |
| type | TEXT |  |
| offset | BIGINT |  |
| blocks\_size | BIGINT | Byte size of each block |
| blocks | BIGINT | Number of blocks |
| inodes | BIGINT | Number of meta nodes |
| flags | INTEGER |  |

**disk\_encryption**

Disk encryption status and information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Disk name |
| uuid | TEXT | Disk Universally Unique Identifier |
| encrypted | INTEGER | 1 If encrypted: true (disk is encrypted), else 0 |
| type | TEXT | Description of cipher type and mode if available |
| uid | TEXT | Currently authenticated user if available (Apple) |
| user\_uuid | TEXT | UUID of authenticated user if available (Apple) |

**disk\_events(EVENTED TABLE)**

Track DMG disk image events (appearance/disappearance) when opened.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| action | TEXT | Appear or disappear |
| path | TEXT | Path of the DMG file accessed |
| name | TEXT | Disk event name |
| device | TEXT | Disk event BSD name |
| uuid | TEXT | UUID of the volume inside DMG if available |
| size | BIGINT | Size of partition in bytes |
| ejectable | INTEGER | 1 if ejectable, 0 if not |
| mountable | INTEGER | 1 if mountable, 0 if not |
| writable | INTEGER | 1 if writable, 0 if not |
| content | TEXT | Disk event content |
| media\_name | TEXT | Disk event media name string |
| vendor | TEXT | Disk event vendor string |
| filesystem | TEXT | Filesystem if available |
| checksum | TEXT | UDIF Master checksum if available (CRC32) |
| time | BIGINT | Time of appearance/disappearance in UNIX time |
| eid | TEXT | Event ID |

**disk\_info NEW**

Retrieve basic information about the physical disks of a system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| partitions | INTEGER | Number of detected partitions on disk. |
| disk\_index | INTEGER | Physical drive number of the disk. |
| type | TEXT | The interface type of the disk. |
| id | TEXT | The unique identifier of the drive on the system. |
| pnp\_device\_id | TEXT | The unique identifier of the drive on the system. |
| disk\_size | BIGINT | Size of the disk. |
| manufacturer | TEXT | The manufacturer of the disk. |
| hardware\_model | TEXT | Hard drive model. |
| name | TEXT | The label of the disk object. |
| serial | TEXT | The serial number of the disk. |
| description | TEXT | The OS's description of the disk. |

**dns\_resolvers**

Resolvers used by this host.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | INTEGER | Address type index or order |
| type | TEXT | Address type: sortlist, nameserver, search |
| address | TEXT | Resolver IP/IPv6 address |
| netmask | TEXT | Address (sortlist) netmask length |
| options | BIGINT | Resolver options |

**docker\_container\_labels**

Docker container labels.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Container ID |
| key | TEXT | Label key |
| value | TEXT | Optional label value |

**docker\_container\_mounts**

Docker container mounts.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Container ID |
| type | TEXT | Type of mount (bind, volume) |
| name | TEXT | Optional mount name |
| source | TEXT | Source path on host |
| destination | TEXT | Destination path inside container |
| driver | TEXT | Driver providing the mount |
| mode | TEXT | Mount options (rw, ro) |
| rw | INTEGER | 1 if read/write. 0 otherwise |
| propagation | TEXT | Mount propagation |

**docker\_container\_networks**

Docker container networks.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Container ID |
| name | TEXT | Network name |
| network\_id | TEXT | Network ID |
| endpoint\_id | TEXT | Endpoint ID |
| gateway | TEXT | Gateway |
| ip\_address | TEXT | IP address |
| ip\_prefix\_len | INTEGER | IP subnet prefix length |
| ipv6\_gateway | TEXT | IPv6 gateway |
| ipv6\_address | TEXT | IPv6 address |
| ipv6\_prefix\_len | INTEGER | IPv6 subnet prefix length |
| mac\_address | TEXT | MAC address |

**docker\_container\_ports**

Docker container ports.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Container ID |
| type | TEXT | Protocol (tcp, udp) |
| port | INTEGER | Port inside the container |
| host\_ip | TEXT | Host IP address on which public port is listening |
| host\_port | INTEGER | Host port |

**docker\_container\_processes**

Docker container processes.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Container ID |
| pid | BIGINT | Process ID |
| name | TEXT | The process path or shorthand argv[0] |
| cmdline | TEXT | Complete argv |
| state | TEXT | Process state |
| uid | BIGINT | User ID |
| gid | BIGINT | Group ID |
| euid | BIGINT | Effective user ID |
| egid | BIGINT | Effective group ID |
| suid | BIGINT | Saved user ID |
| sgid | BIGINT | Saved group ID |
| wired\_size | BIGINT | Bytes of unpagable memory used by process |
| resident\_size | BIGINT | Bytes of private memory used by process |
| total\_size | BIGINT | Total virtual memory size |
| start\_time | BIGINT | Process start in seconds since boot (non-sleeping) |
| parent | BIGINT | Process parent's PID |
| pgroup | BIGINT | Process group |
| threads | INTEGER | Number of threads used by process |
| nice | INTEGER | Process nice level (-20 to 20, default 0) |
| user | TEXT | User name |
| time | TEXT | Cumulative CPU time. [DD-]HH:MM:SS format |
| cpu | DOUBLE | CPU utilization as percentage |
| mem | DOUBLE | Memory utilization as percentage |

**docker\_container\_stats**

Docker container statistics. Queries on this table take at least one second.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Container ID |
| name | TEXT | Container name |
| pids | INTEGER | Number of processes |
| read | BIGINT | UNIX time when stats were read |
| preread | BIGINT | UNIX time when stats were last read |
| interval | BIGINT | Difference between read and preread in nano-seconds |
| disk\_read | BIGINT | Total disk read bytes |
| disk\_write | BIGINT | Total disk write bytes |
| num\_procs | INTEGER | Number of processors |
| cpu\_total\_usage | BIGINT | Total CPU usage |
| cpu\_kernelmode\_usage | BIGINT | CPU kernel mode usage |
| cpu\_usermode\_usage | BIGINT | CPU user mode usage |
| system\_cpu\_usage | BIGINT | CPU system usage |
| online\_cpus | INTEGER | Online CPUs |
| pre\_cpu\_total\_usage | BIGINT | Last read total CPU usage |
| pre\_cpu\_kernelmode\_usage | BIGINT | Last read CPU kernel mode usage |
| pre\_cpu\_usermode\_usage | BIGINT | Last read CPU user mode usage |
| pre\_system\_cpu\_usage | BIGINT | Last read CPU system usage |
| pre\_online\_cpus | INTEGER | Last read online CPUs |
| memory\_usage | BIGINT | Memory usage |
| memory\_max\_usage | BIGINT | Memory maximum usage |
| memory\_limit | BIGINT | Memory limit |
| network\_rx\_bytes | BIGINT | Total network bytes read |
| network\_tx\_bytes | BIGINT | Total network bytes transmitted |

**docker\_containers**

Docker containers information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Container ID |
| name | TEXT | Container name |
| image | TEXT | Docker image (name) used to launch this container |
| image\_id | TEXT | Docker image ID |
| command | TEXT | Command with arguments |
| created | BIGINT | Time of creation as UNIX time |
| state | TEXT | Container state (created, restarting, running, removing, paused, exited, dead) |
| status | TEXT | Container status information |
| pid | BIGINT | Identifier of the initial process |
| path | TEXT | Container path |
| config\_entrypoint | TEXT | Container entrypoint(s) |
| started\_at | TEXT | Container start time as string |
| finished\_at | TEXT | Container finish time as string |
| privileged | INTEGER | Is the container privileged |
| security\_options | TEXT | List of container security options |
| env\_variables | TEXT | Container environmental variables |
| cgroup\_namespace | TEXT | cgroup namespace |
| ipc\_namespace | TEXT | IPC namespace |
| mnt\_namespace | TEXT | Mount namespace |
| net\_namespace | TEXT | Network namespace |
| pid\_namespace | TEXT | PID namespace |
| user\_namespace | TEXT | User namespace |
| uts\_namespace | TEXT | UTS namespace |

**docker\_image\_labels**

Docker image labels.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Image ID |
| key | TEXT | Label key |
| value | TEXT | Optional label value |

**docker\_images**

Docker images information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Image ID |
| created | BIGINT | Time of creation as UNIX time |
| size\_bytes | BIGINT | Size of image in bytes |
| tags | TEXT | Comma-separated list of repository tags |

**docker\_info**

Docker system information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Docker system ID |
| containers | INTEGER | Total number of containers |
| containers\_running | INTEGER | Number of containers currently running |
| containers\_paused | INTEGER | Number of containers in paused state |
| containers\_stopped | INTEGER | Number of containers in stopped state |
| images | INTEGER | Number of images |
| storage\_driver | TEXT | Storage driver |
| memory\_limit | INTEGER | 1 if memory limit support is enabled. 0 otherwise |
| swap\_limit | INTEGER | 1 if swap limit support is enabled. 0 otherwise |
| kernel\_memory | INTEGER | 1 if kernel memory limit support is enabled. 0 otherwise |
| cpu\_cfs\_period | INTEGER | 1 if CPU Completely Fair Scheduler (CFS) period support is enabled. 0 otherwise |
| cpu\_cfs\_quota | INTEGER | 1 if CPU Completely Fair Scheduler (CFS) quota support is enabled. 0 otherwise |
| cpu\_shares | INTEGER | 1 if CPU share weighting support is enabled. 0 otherwise |
| cpu\_set | INTEGER | 1 if CPU set selection support is enabled. 0 otherwise |
| ipv4\_forwarding | INTEGER | 1 if IPv4 forwarding is enabled. 0 otherwise |
| bridge\_nf\_iptables | INTEGER | 1 if bridge netfilter iptables is enabled. 0 otherwise |
| bridge\_nf\_ip6tables | INTEGER | 1 if bridge netfilter ip6tables is enabled. 0 otherwise |
| oom\_kill\_disable | INTEGER | 1 if Out-of-memory kill is disabled. 0 otherwise |
| logging\_driver | TEXT | Logging driver |
| cgroup\_driver | TEXT | Control groups driver |
| kernel\_version | TEXT | Kernel version |
| os | TEXT | Operating system |
| os\_type | TEXT | Operating system type |
| architecture | TEXT | Hardware architecture |
| cpus | INTEGER | Number of CPUs |
| memory | BIGINT | Total memory |
| http\_proxy | TEXT | HTTP proxy |
| https\_proxy | TEXT | HTTPS proxy |
| no\_proxy | TEXT | Comma-separated list of domain extensions proxy should not be used for |
| name | TEXT | Name of the docker host |
| server\_version | TEXT | Server version |
| root\_dir | TEXT | Docker root directory |

**docker\_network\_labels**

Docker network labels.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Network ID |
| key | TEXT | Label key |
| value | TEXT | Optional label value |

**docker\_networks**

Docker networks information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| id | TEXT | Network ID |
| name | TEXT | Network name |
| driver | TEXT | Network driver |
| created | BIGINT | Time of creation as UNIX time |
| enable\_ipv6 | INTEGER | 1 if IPv6 is enabled on this network. 0 otherwise |
| subnet | TEXT | Network subnet |
| gateway | TEXT | Network gateway |

**docker\_version**

Docker version information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| version | TEXT | Docker version |
| api\_version | TEXT | API version |
| min\_api\_version | TEXT | Minimum API version supported |
| git\_commit | TEXT | Docker build git commit |
| go\_version | TEXT | Go version |
| os | TEXT | Operating system |
| arch | TEXT | Hardware architecture |
| kernel\_version | TEXT | Kernel version |
| build\_time | TEXT | Build time |

**docker\_volume\_labels**

Docker volume labels.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Volume name |
| key | TEXT | Label key |
| value | TEXT | Optional label value |

**docker\_volumes**

Docker volumes information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Volume name |
| driver | TEXT | Volume driver |
| mount\_point | TEXT | Mount point |
| type | TEXT | Volume type |

**drivers**

Details for in-use Windows device drivers. This does not display installed but unused drivers.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| device\_id | TEXT | Device ID |
| device\_name | TEXT | Device name |
| image | TEXT | Path to driver image file |
| description | TEXT | Driver description |
| service | TEXT | Driver service name, if one exists |
| service\_key | TEXT | Driver service registry key |
| version | TEXT | Driver version |
| inf | TEXT | Associated inf file |
| class | TEXT | Device/driver class name |
| provider | TEXT | Driver provider |
| manufacturer | TEXT | Device manufacturer |
| driver\_key | TEXT | Driver key |
| date | BIGINT | Driver date |
| signed | INTEGER | Whether the driver is signed or not |

**ec2\_instance\_metadata**

EC2 instance metadata.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| instance\_id | TEXT | EC2 instance ID |
| instance\_type | TEXT | EC2 instance type |
| architecture | TEXT | Hardware architecture of this EC2 instance |
| region | TEXT | AWS region in which this instance launched |
| availability\_zone | TEXT | Availability zone in which this instance launched |
| local\_hostname | TEXT | Private IPv4 DNS hostname of the first interface of this instance |
| local\_ipv4 | TEXT | Private IPv4 address of the first interface of this instance |
| mac | TEXT | MAC address for the first network interface of this EC2 instance |
| security\_groups | TEXT | Comma separated list of security group names |
| iam\_arn | TEXT | If there is an IAM role associated with the instance, contains instance profile ARN |
| ami\_id | TEXT | AMI ID used to launch this EC2 instance |
| reservation\_id | TEXT | ID of the reservation |
| account\_id | TEXT | AWS account ID which owns this EC2 instance |
| ssh\_public\_key | TEXT | SSH public key. Only available if supplied at instance launch time |

**ec2\_instance\_tags**

EC2 instance tag key value pairs.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| instance\_id | TEXT | EC2 instance ID |
| key | TEXT | Tag key |
| value | TEXT | Tag value |

**etc\_hosts**

Line-parsed /etc/hosts.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| address | TEXT | IP address mapping |
| hostnames | TEXT | Raw hosts mapping |

**etc\_protocols**

Line-parsed /etc/protocols.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Protocol name |
| number | INTEGER | Protocol number |
| alias | TEXT | Protocol alias |
| comment | TEXT | Comment with protocol description |

**etc\_services**

Line-parsed /etc/services.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Service name |
| port | INTEGER | Service port number |
| protocol | TEXT | Transport protocol (TCP/UDP) |
| aliases | TEXT | Optional space separated list of other names for a service |
| comment | TEXT | Optional comment for a service. |

**event\_taps**

Returns information about installed event taps.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| enabled | INTEGER | Is the Event Tap enabled |
| event\_tap\_id | INTEGER | Unique ID for the Tap |
| event\_tapped | TEXT | The mask that identifies the set of events to be observed. |
| process\_being\_tapped | INTEGER | The process ID of the target application |
| tapping\_process | INTEGER | The process ID of the application that created the event tap. |

**example(EVENTED TABLE)**

This is an example table spec.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Description for name column |
| points | INTEGER | This is a signed SQLite int column |
| size | BIGINT | This is a signed SQLite bigint column |
| action | TEXT | Action performed in generation |
| id | INTEGER | An index of some sort |
| path | TEXT | Path of example |

**extended\_attributes**

Returns the extended attributes for files (similar to Windows ADS).

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Absolute file path |
| directory | TEXT | Directory of file(s) |
| key | TEXT | Name of the value generated from the extended attribute |
| value | TEXT | The parsed information from the attribute |
| base64 | INTEGER | 1 if the value is base64 encoded else 0 |

**fan\_speed\_sensors**

Fan speeds.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| fan | TEXT | Fan number |
| name | TEXT | Fan name |
| actual | INTEGER | Actual speed |
| min | INTEGER | Minimum speed |
| max | INTEGER | Maximum speed |
| target | INTEGER | Target speed |

**fbsd\_kmods**

Loaded FreeBSD kernel modules.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Module name |
| size | INTEGER | Size of module content |
| refs | INTEGER | Module reverse dependencies |
| address | TEXT | Kernel module address |

**file**

Interactive filesystem attributes and metadata.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Absolute file path |
| directory | TEXT | Directory of file(s) |
| filename | TEXT | Name portion of file path |
| inode | BIGINT | Filesystem inode number |
| uid | BIGINT | Owning user ID |
| gid | BIGINT | Owning group ID |
| mode | TEXT | Permission bits |
| device | BIGINT | Device ID (optional) |
| size | BIGINT | Size of file in bytes |
| block\_size | INTEGER | Block size of filesystem |
| atime | BIGINT | Last access time |
| mtime | BIGINT | Last modification time |
| ctime | BIGINT | Last status change time |
| btime | BIGINT | (B)irth or (cr)eate time |
| hard\_links | INTEGER | Number of hard links |
| symlink | INTEGER | 1 if the path is a symlink, otherwise 0 |
| type | TEXT | File status |

**file\_events(EVENTED TABLE)**

Track time/action changes to files specified in configuration data.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| target\_path | TEXT | The path associated with the event |
| category | TEXT | The category of the file defined in the config |
| action | TEXT | Change action (UPDATE, REMOVE, etc) |
| transaction\_id | BIGINT | ID used during bulk update |
| inode | BIGINT | Filesystem inode number |
| uid | BIGINT | Owning user ID |
| gid | BIGINT | Owning group ID |
| mode | TEXT | Permission bits |
| size | BIGINT | Size of file in bytes |
| atime | BIGINT | Last access time |
| mtime | BIGINT | Last modification time |
| ctime | BIGINT | Last status change time |
| md5 | TEXT | The MD5 of the file after change |
| sha1 | TEXT | The SHA1 of the file after change |
| sha256 | TEXT | The SHA256 of the file after change |
| hashed | INTEGER | 1 if the file was hashed, 0 if not, -1 if hashing failed |
| time | BIGINT | Time of file event |
| eid | TEXT | Event ID |

**firefox\_addons**

Firefox browser extensions, webapps, and addons.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | The local user that owns the addon |
| name | TEXT | Addon display name |
| identifier | TEXT | Addon identifier |
| creator | TEXT | Addon-supported creator string |
| type | TEXT | Extension, addon, webapp |
| version | TEXT | Addon-supplied version string |
| description | TEXT | Addon-supplied description string |
| source\_url | TEXT | URL that installed the addon |
| visible | INTEGER | 1 If the addon is shown in browser else 0 |
| active | INTEGER | 1 If the addon is active else 0 |
| disabled | INTEGER | 1 If the addon is application-disabled else 0 |
| autoupdate | INTEGER | 1 If the addon applies background updates else 0 |
| native | INTEGER | 1 If the addon includes binary components else 0 |
| location | TEXT | Global, profile location |
| path | TEXT | Path to plugin bundle |

**gatekeeper**

OS X Gatekeeper Details.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| assessments\_enabled | INTEGER | 1 If a Gatekeeper is enabled else 0 |
| dev\_id\_enabled | INTEGER | 1 If a Gatekeeper allows execution from identified developers else 0 |
| version | TEXT | Version of Gatekeeper's gke.bundle |
| opaque\_version | TEXT | Version of Gatekeeper's gkopaque.bundle |

**gatekeeper\_approved\_apps**

Gatekeeper apps a user has allowed to run.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Path of executable allowed to run |
| requirement | TEXT | Code signing requirement language |
| ctime | DOUBLE | Last change time |
| mtime | DOUBLE | Last modification time |

**groups**

Local system groups.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| gid | BIGINT | Unsigned int64 group ID |
| gid\_signed | BIGINT | A signed int64 version of gid |
| groupname | TEXT | Canonical local group name |
| group\_sid | TEXT | Unique group ID |
| comment | TEXT | Remarks or comments associated with the group |

**hardware\_events(EVENTED TABLE)**

Hardware (PCI/USB/HID) events from UDEV or IOKit.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| action | TEXT | Remove, insert, change properties, etc |
| path | TEXT | Local device path assigned (optional) |
| type | TEXT | Type of hardware and hardware event |
| driver | TEXT | Driver claiming the device |
| vendor | TEXT | Hardware device vendor |
| vendor\_id | TEXT | Hex encoded Hardware vendor identifier |
| model | TEXT | Hardware device model |
| model\_id | TEXT | Hex encoded Hardware model identifier |
| serial | TEXT | Device serial (optional) |
| revision | TEXT | Device revision (optional) |
| time | BIGINT | Time of hardware event |
| eid | TEXT | Event ID |

**hash**

Filesystem hash data.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Must provide a path or directory |
| directory | TEXT | Must provide a path or directory |
| md5 | TEXT | MD5 hash of provided filesystem data |
| sha1 | TEXT | SHA1 hash of provided filesystem data |
| sha256 | TEXT | SHA256 hash of provided filesystem data |

**homebrew\_packages**

The installed homebrew package database.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Package name |
| path | TEXT | Package install path |
| version | TEXT | Current 'linked' version |

**ie\_extensions**

Internet Explorer browser extensions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Extension display name |
| registry\_path | TEXT | Extension identifier |
| version | TEXT | Version of the executable |
| path | TEXT | Path to executable |

**intel\_me\_info**

Intel ME/CSE Info.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| version | TEXT | Intel ME version |

**interface\_addresses**

Network interfaces and relevant metadata.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| interface | TEXT | Interface name |
| address | TEXT | Specific address for interface |
| mask | TEXT | Interface netmask |
| broadcast | TEXT | Broadcast address for the interface |
| point\_to\_point | TEXT | PtP address for the interface |
| type | TEXT | Type of address. One of dhcp, manual, auto, other |
| friendly\_name | TEXT | The friendly display name of the interface. |

**interface\_details**

Detailed information and stats of network interfaces.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| interface | TEXT | Interface name |
| mac | TEXT | MAC of interface (optional) |
| type | INTEGER | Interface type (includes virtual) |
| mtu | INTEGER | Network MTU |
| metric | INTEGER | Metric based on the speed of the interface |
| flags | INTEGER | Flags (netdevice) for the device |
| ipackets | BIGINT | Input packets |
| opackets | BIGINT | Output packets |
| ibytes | BIGINT | Input bytes |
| obytes | BIGINT | Output bytes |
| ierrors | BIGINT | Input errors |
| oerrors | BIGINT | Output errors |
| idrops | BIGINT | Input drops |
| odrops | BIGINT | Output drops |
| collisions | BIGINT | Packet Collisions detected |
| last\_change | BIGINT | Time of last device modification (optional) |
| friendly\_name | TEXT | The friendly display name of the interface. |
| description | TEXT | Short description of the object—a one-line string. |
| manufacturer | TEXT | Name of the network adapter's manufacturer. |
| connection\_id | TEXT | Name of the network connection as it appears in the Network Connections Control Panel program. |
| connection\_status | TEXT | State of the network adapter connection to the network. |
| enabled | INTEGER | Indicates whether the adapter is enabled or not. |
| physical\_adapter | INTEGER | Indicates whether the adapter is a physical or a logical adapter. |
| speed | INTEGER | Estimate of the current bandwidth in bits per second. |
| dhcp\_enabled | INTEGER | If TRUE, the dynamic host configuration protocol (DHCP) server automatically assigns an IP address to the computer system when establishing a network connection. |
| dhcp\_lease\_expires | TEXT | Expiration date and time for a leased IP address that was assigned to the computer by the dynamic host configuration protocol (DHCP) server. |
| dhcp\_lease\_obtained | TEXT | Date and time the lease was obtained for the IP address assigned to the computer by the dynamic host configuration protocol (DHCP) server. |
| dhcp\_server | TEXT | IP address of the dynamic host configuration protocol (DHCP) server. |
| dns\_domain | TEXT | Organization name followed by a period and an extension that indicates the type of organization, such as 'microsoft.com'. |
| dns\_domain\_suffix\_search\_order | TEXT | Array of DNS domain suffixes to be appended to the end of host names during name resolution. |
| dns\_host\_name | TEXT | Host name used to identify the local computer for authentication by some utilities. |
| dns\_server\_search\_order | TEXT | Array of server IP addresses to be used in querying for DNS servers. |

**iokit\_devicetree**

The IOKit registry matching the DeviceTree plane.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Device node name |
| class | TEXT | Best matching device class (most-specific category) |
| id | BIGINT | IOKit internal registry ID |
| parent | BIGINT | Parent device registry ID |
| device\_path | TEXT | Device tree path |
| service | INTEGER | 1 if the device conforms to IOService else 0 |
| busy\_state | INTEGER | 1 if the device is in a busy state else 0 |
| retain\_count | INTEGER | The device reference count |
| depth | INTEGER | Device nested depth |

**iokit\_registry**

The full IOKit registry without selecting a plane.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Default name of the node |
| class | TEXT | Best matching device class (most-specific category) |
| id | BIGINT | IOKit internal registry ID |
| parent | BIGINT | Parent registry ID |
| busy\_state | INTEGER | 1 if the node is in a busy state else 0 |
| retain\_count | INTEGER | The node reference count |
| depth | INTEGER | Node nested depth |

**iptables**

Linux IP packet filtering and NAT tool.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| filter\_name | TEXT | Packet matching filter table name. |
| chain | TEXT | Size of module content. |
| policy | TEXT | Policy that applies for this rule. |
| target | TEXT | Target that applies for this rule. |
| protocol | INTEGER | Protocol number identification. |
| src\_port | TEXT | Protocol source port(s). |
| dst\_port | TEXT | Protocol destination port(s). |
| src\_ip | TEXT | Source IP address. |
| src\_mask | TEXT | Source IP address mask. |
| iniface | TEXT | Input interface for the rule. |
| iniface\_mask | TEXT | Input interface mask for the rule. |
| dst\_ip | TEXT | Destination IP address. |
| dst\_mask | TEXT | Destination IP address mask. |
| outiface | TEXT | Output interface for the rule. |
| outiface\_mask | TEXT | Output interface mask for the rule. |
| match | TEXT | Matching rule that applies. |
| packets | INTEGER | Number of matching packets for this rule. |
| bytes | INTEGER | Number of matching bytes for this rule. |

**kernel\_extensions**

OS X's kernel extensions, both loaded and within the load search path.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| idx | INTEGER | Extension load tag or index |
| refs | INTEGER | Reference count |
| size | BIGINT | Bytes of wired memory used by extension |
| name | TEXT | Extension label |
| version | TEXT | Extension version |
| linked\_against | TEXT | Indexes of extensions this extension is linked against |
| path | TEXT | Optional path to extension bundle |

**kernel\_info**

Basic active kernel information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| version | TEXT | Kernel version |
| arguments | TEXT | Kernel arguments |
| path | TEXT | Kernel path |
| device | TEXT | Kernel device identifier |

**kernel\_integrity**

Various Linux kernel integrity checked attributes.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| sycall\_addr\_modified | INTEGER | 0 or 1, for whether a syscall table pointer is modified |
| text\_segment\_hash | TEXT | Hash value for the kernel's .text memory segment |

**kernel\_modules**

Linux kernel modules both loaded and within the load search path.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Module name |
| size | TEXT | Size of module content |
| used\_by | TEXT | Module reverse dependencies |
| status | TEXT | Kernel module status |
| address | TEXT | Kernel module address |

**kernel\_panics**

System kernel panic logs.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Location of log file |
| time | TEXT | Formatted time of the event |
| registers | TEXT | A space delimited line of register:value pairs |
| frame\_backtrace | TEXT | Backtrace of the crashed module |
| module\_backtrace | TEXT | Modules appearing in the crashed module's backtrace |
| dependencies | TEXT | Module dependencies existing in crashed module's backtrace |
| name | TEXT | Process name corresponding to crashed thread |
| os\_version | TEXT | Version of the operating system |
| kernel\_version | TEXT | Version of the system kernel |
| system\_model | TEXT | Physical system model, for example 'MacBookPro12,1 (Mac-E43C1C25D4880AD6)' |
| uptime | BIGINT | System uptime at kernel panic in nanoseconds |
| last\_loaded | TEXT | Last loaded module before panic |
| last\_unloaded | TEXT | Last unloaded module before panic |

**keychain\_acls**

Applications that have ACL entries in the keychain.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| keychain\_path | TEXT | The path of the keychain |
| authorizations | TEXT | A space delimited set of authorization attributes |
| path | TEXT | The path of the authorized application |
| description | TEXT | The description included with the ACL entry |
| label | TEXT | An optional label tag that may be included with the keychain entry |

**keychain\_items**

Generic details about keychain items.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| label | TEXT | Generic item name |
| description | TEXT | Optional item description |
| comment | TEXT | Optional keychain comment |
| created | TEXT | Data item was created |
| modified | TEXT | Date of last modification |
| type | TEXT | Keychain item type (class) |
| path | TEXT | Path to keychain containing item |

**known\_hosts**

A line-delimited known\_hosts table.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | The local user that owns the known\_hosts file |
| key | TEXT | parsed authorized keys line |
| key\_file | TEXT | Path to known\_hosts file |

**kva\_speculative\_info NEW**

Display kernel virtual address and speculative execution information for the system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| kva\_shadow\_enabled | INTEGER | Kernel Virtual Address shadowing is enabled. |
| kva\_shadow\_user\_global | INTEGER | User pages are marked as global. |
| kva\_shadow\_pcid | INTEGER | Kernel VA PCID flushing optimization is enabled. |
| kva\_shadow\_inv\_pcid | INTEGER | Kernel VA INVPCID is enabled. |
| bp\_mitigations | INTEGER | Branch Prediction mitigations are enabled. |
| bp\_system\_pol\_disabled | INTEGER | Branch Predictions are disabled via system policy. |
| bp\_microcode\_disabled | INTEGER | Branch Predictions are disabled due to lack of microcode update. |
| cpu\_spec\_ctrl\_supported | INTEGER | SPEC\_CTRL MSR supported by CPU Microcode. |
| ibrs\_support\_enabled | INTEGER | Windows uses IBRS. |
| stibp\_support\_enabled | INTEGER | Windows uses STIBP. |
| cpu\_pred\_cmd\_supported | INTEGER | PRED\_CMD MSR supported by CPU Microcode. |

**last**

System logins and logouts.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| username | TEXT | Entry username |
| tty | TEXT | Entry terminal |
| pid | INTEGER | Process (or thread) ID |
| type | INTEGER | Entry type, according to ut\_type types (utmp.h) |
| time | INTEGER | Entry timestamp |
| host | TEXT | Entry hostname |

**launchd**

LaunchAgents and LaunchDaemons from default search paths.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Path to daemon or agent plist |
| name | TEXT | File name of plist (used by launchd) |
| label | TEXT | Daemon or agent service name |
| program | TEXT | Path to target program |
| run\_at\_load | TEXT | Should the program run on launch load |
| keep\_alive | TEXT | Should the process be restarted if killed |
| on\_demand | TEXT | Deprecated key, replaced by keep\_alive |
| disabled | TEXT | Skip loading this daemon or agent on boot |
| username | TEXT | Run this daemon or agent as this username |
| groupname | TEXT | Run this daemon or agent as this group |
| stdout\_path | TEXT | Pipe stdout to a target path |
| stderr\_path | TEXT | Pipe stderr to a target path |
| start\_interval | TEXT | Frecuency of running in seconds |
| program\_arguments | TEXT | Command line arguments passed to program |
| watch\_paths | TEXT | Key that launches daemon or agent if path is modified |
| queue\_directories | TEXT | Similar to watch\_paths but only with non-empty directories |
| inetd\_compatibility | TEXT | Run this daemon or agent as it was launched from inetd |
| start\_on\_mount | TEXT | Run daemon or agent every time a filesystem is mounted |
| root\_directory | TEXT | Key used to specify a directory to chroot to before launch |
| working\_directory | TEXT | Key used to specify a directory to chdir to before launch |
| process\_type | TEXT | Key describes the intended purpose of the job |

**launchd\_overrides**

Override keys, per user, for LaunchDaemons and Agents.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| label | TEXT | Daemon or agent service name |
| key | TEXT | Name of the override key |
| value | TEXT | Overriden value |
| uid | BIGINT | User ID applied to the override, 0 applies to all |
| path | TEXT | Path to daemon or agent plist |

**listening\_ports**

Processes with listening (bound) network sockets/ports.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | INTEGER | Process (or thread) ID |
| port | INTEGER | Transport layer port |
| protocol | INTEGER | Transport protocol (TCP/UDP) |
| family | INTEGER | Network protocol (IPv4, IPv6) |
| address | TEXT | Specific address for bind |
| fd | BIGINT | Socket file descriptor number |
| socket | BIGINT | Socket handle or inode number |
| path | TEXT | Path for UNIX domain sockets |
| net\_namespace | TEXT | The inode number of the network namespace |

**lldp\_neighbors**

LLDP neighbors of interfaces.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| interface | TEXT | Interface name |
| rid | INTEGER | Neighbor chassis index |
| chassis\_id\_type | TEXT | Neighbor chassis ID type |
| chassis\_id | TEXT | Neighbor chassis ID value |
| chassis\_sysname | TEXT | CPU brand string, contains vendor and model |
| chassis\_sys\_description | INTEGER | Max number of CPU physical cores |
| chassis\_bridge\_capability\_available | INTEGER | Chassis bridge capability availability |
| chassis\_bridge\_capability\_enabled | INTEGER | Is chassis bridge capability enabled. |
| chassis\_router\_capability\_available | INTEGER | Chassis router capability availability |
| chassis\_router\_capability\_enabled | INTEGER | Chassis router capability enabled |
| chassis\_repeater\_capability\_available | INTEGER | Chassis repeater capability availability |
| chassis\_repeater\_capability\_enabled | INTEGER | Chassis repeater capability enabled |
| chassis\_wlan\_capability\_available | INTEGER | Chassis wlan capability availability |
| chassis\_wlan\_capability\_enabled | INTEGER | Chassis wlan capability enabled |
| chassis\_tel\_capability\_available | INTEGER | Chassis telephone capability availability |
| chassis\_tel\_capability\_enabled | INTEGER | Chassis telephone capability enabled |
| chassis\_docsis\_capability\_available | INTEGER | Chassis DOCSIS capability availability |
| chassis\_docsis\_capability\_enabled | INTEGER | Chassis DOCSIS capability enabled |
| chassis\_station\_capability\_available | INTEGER | Chassis station capability availability |
| chassis\_station\_capability\_enabled | INTEGER | Chassis station capability enabled |
| chassis\_other\_capability\_available | INTEGER | Chassis other capability availability |
| chassis\_other\_capability\_enabled | INTEGER | Chassis other capability enabled |
| chassis\_mgmt\_ips | TEXT | Comma delimited list of chassis management IPS |
| port\_id\_type | TEXT | Port ID type |
| port\_id | TEXT | Port ID value |
| port\_description | TEXT | Port description |
| port\_ttl | BIGINT | Age of neighbor port |
| port\_mfs | BIGINT | Port max frame size |
| port\_aggregation\_id | TEXT | Port aggregation ID |
| port\_autoneg\_supported | INTEGER | Auto negotiation supported |
| port\_autoneg\_enabled | INTEGER | Is auto negotiation enabled |
| port\_mau\_type | TEXT | MAU type |
| port\_autoneg\_10baset\_hd\_enabled | INTEGER | 10Base-T HD auto negotiation enabled |
| port\_autoneg\_10baset\_fd\_enabled | INTEGER | 10Base-T FD auto negotiation enabled |
| port\_autoneg\_100basetx\_hd\_enabled | INTEGER | 100Base-TX HD auto negotiation enabled |
| port\_autoneg\_100basetx\_fd\_enabled | INTEGER | 100Base-TX FD auto negotiation enabled |
| port\_autoneg\_100baset2\_hd\_enabled | INTEGER | 100Base-T2 HD auto negotiation enabled |
| port\_autoneg\_100baset2\_fd\_enabled | INTEGER | 100Base-T2 FD auto negotiation enabled |
| port\_autoneg\_100baset4\_hd\_enabled | INTEGER | 100Base-T4 HD auto negotiation enabled |
| port\_autoneg\_100baset4\_fd\_enabled | INTEGER | 100Base-T4 FD auto negotiation enabled |
| port\_autoneg\_1000basex\_hd\_enabled | INTEGER | 1000Base-X HD auto negotiation enabled |
| port\_autoneg\_1000basex\_fd\_enabled | INTEGER | 1000Base-X FD auto negotiation enabled |
| port\_autoneg\_1000baset\_hd\_enabled | INTEGER | 1000Base-T HD auto negotiation enabled |
| port\_autoneg\_1000baset\_fd\_enabled | INTEGER | 1000Base-T FD auto negotiation enabled |
| power\_device\_type | TEXT | Dot3 power device type |
| power\_mdi\_supported | INTEGER | MDI power supported |
| power\_mdi\_enabled | INTEGER | Is MDI power enabled |
| power\_paircontrol\_enabled | INTEGER | Is power pair control enabled |
| power\_pairs | TEXT | Dot3 power pairs |
| power\_class | TEXT | Power class |
| power\_8023at\_enabled | INTEGER | Is 802.3at enabled |
| power\_8023at\_power\_type | TEXT | 802.3at power type |
| power\_8023at\_power\_source | TEXT | 802.3at power source |
| power\_8023at\_power\_priority | TEXT | 802.3at power priority |
| power\_8023at\_power\_allocated | TEXT | 802.3at power allocated |
| power\_8023at\_power\_requested | TEXT | 802.3at power requested |
| med\_device\_type | TEXT | Chassis MED type |
| med\_capability\_capabilities | INTEGER | Is MED capabilities enabled |
| med\_capability\_policy | INTEGER | Is MED policy capability enabled |
| med\_capability\_location | INTEGER | Is MED location capability enabled |
| med\_capability\_mdi\_pse | INTEGER | Is MED MDI PSE capability enabled |
| med\_capability\_mdi\_pd | INTEGER | Is MED MDI PD capability enabled |
| med\_capability\_inventory | INTEGER | Is MED inventory capability enabled |
| med\_policies | TEXT | Comma delimited list of MED policies |
| vlans | TEXT | Comma delimited list of vlan ids |
| pvid | TEXT | Primary VLAN id |
| ppvids\_supported | TEXT | Comma delimited list of supported PPVIDs |
| ppvids\_enabled | TEXT | Comma delimited list of enabled PPVIDs |
| pids | TEXT | Comma delimited list of PIDs |

**load\_average**

Displays information about the system wide load averages.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| period | TEXT | Period over which the average is calculated. |
| average | TEXT | Load average over the specified period. |

**logged\_in\_users**

Users with an active shell on the system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| type | TEXT | Login type |
| user | TEXT | User login name |
| tty | TEXT | Device name |
| host | TEXT | Remote hostname |
| time | INTEGER | Time entry was made |
| pid | INTEGER | Process (or thread) ID |

**logical\_drives**

Details for logical drives on the system. A logical drive generally represents a single partition.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| device\_id | TEXT | The drive id, usually the drive name, e.g., 'C:'. |
| type | TEXT | The type of disk drive this logical drive represents. |
| free\_space | BIGINT | The amount of free space, in bytes, of the drive. |
| size | BIGINT | The total amount of space, in bytes, of the drive. |
| file\_system | TEXT | The file system of the drive. |
| boot\_partition | INTEGER | True if Windows booted from this drive. |

**magic**

Magic number recognition library table.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Absolute path to target file |
| data | TEXT | Magic number data from libmagic |
| mime\_type | TEXT | MIME type data from libmagic |
| mime\_encoding | TEXT | MIME encoding data from libmagic |

**managed\_policies**

The managed configuration policies from AD, MDM, MCX, etc.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| domain | TEXT | System or manager-chosen domain key |
| uuid | TEXT | Optional UUID assigned to policy set |
| name | TEXT | Policy key name |
| value | TEXT | Policy value |
| username | TEXT | Policy applies only this user |
| manual | INTEGER | 1 if policy was loaded manually, otherwise 0 |

**md\_devices**

Software RAID array settings.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| device\_name | TEXT | md device name |
| status | TEXT | Current state of the array |
| raid\_level | INTEGER | Current raid level of the array |
| size | BIGINT | size of the array in blocks |
| chunk\_size | BIGINT | chunk size in bytes |
| raid\_disks | INTEGER | Number of configured RAID disks in array |
| nr\_raid\_disks | INTEGER | Number of partitions or disk devices to comprise the array |
| working\_disks | INTEGER | Number of working disks in array |
| active\_disks | INTEGER | Number of active disks in array |
| failed\_disks | INTEGER | Number of active disks in array |
| spare\_disks | INTEGER | Number of active disks in array |
| superblock\_state | TEXT | State of the superblock |
| superblock\_version | TEXT | Version of the superblock |
| superblock\_update\_time | BIGINT | Unix timestamp of last update |
| bitmap\_on\_mem | TEXT | Pages allocated in in-memory bitmap, if enabled |
| bitmap\_chunk\_size | TEXT | Bitmap chunk size |
| bitmap\_external\_file | TEXT | External referenced bitmap file |
| recovery\_progress | TEXT | Progress of the recovery activity |
| recovery\_finish | TEXT | Estimated duration of recovery activity |
| recovery\_speed | TEXT | Speed of recovery activity |
| resync\_progress | TEXT | Progress of the resync activity |
| resync\_finish | TEXT | Estimated duration of resync activity |
| resync\_speed | TEXT | Speed of resync activity |
| reshape\_progress | TEXT | Progress of the reshape activity |
| reshape\_finish | TEXT | Estimated duration of reshape activity |
| reshape\_speed | TEXT | Speed of reshape activity |
| check\_array\_progress | TEXT | Progress of the resync activity |
| check\_array\_finish | TEXT | Estimated duration of resync activity |
| check\_array\_speed | TEXT | Speed of resync activity |
| unused\_devices | TEXT | Unused devices |
| other | TEXT | Other information associated with array from /proc/mdstat |

**md\_drives**

Drive devices used for Software RAID.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| md\_device\_name | TEXT | md device name |
| drive\_name | TEXT | Drive device name |
| slot | INTEGER | Slot position of disk |
| state | TEXT | State of the drive |

**md\_personalities**

Software RAID setting supported by the kernel.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Name of personality supported by kernel |

**memory\_info**

Main memory information in bytes.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| memory\_total | BIGINT | Total amount of physical RAM, in bytes |
| memory\_free | BIGINT | The amount of physical RAM, in bytes, left unused by the system |
| buffers | BIGINT | The amount of physical RAM, in bytes, used for file buffers |
| cached | BIGINT | The amount of physical RAM, in bytes, used as cache memory |
| swap\_cached | BIGINT | The amount of swap, in bytes, used as cache memory |
| active | BIGINT | The total amount of buffer or page cache memory, in bytes, that is in active use |
| inactive | BIGINT | The total amount of buffer or page cache memory, in bytes, that are free and available |
| swap\_total | BIGINT | The total amount of swap available, in bytes |
| swap\_free | BIGINT | The total amount of swap free, in bytes |

**memory\_map**

OS memory region map.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Region name |
| start | TEXT | Start address of memory region |
| end | TEXT | End address of memory region |

**mounts**

System mounted devices and filesystems (not process specific).

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| device | TEXT | Mounted device |
| device\_alias | TEXT | Mounted device alias |
| path | TEXT | Mounted device path |
| type | TEXT | Mounted device type |
| blocks\_size | BIGINT | Block size in bytes |
| blocks | BIGINT | Mounted device used blocks |
| blocks\_free | BIGINT | Mounted device free blocks |
| blocks\_available | BIGINT | Mounted device available blocks |
| inodes | BIGINT | Mounted device used inodes |
| inodes\_free | BIGINT | Mounted device free inodes |
| flags | TEXT | Mounted device flags |

**msr**

Various pieces of data stored in the model specific register per processor. NOTE: the msr kernel module must be enabled, and osquery must be run as root.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| processor\_number | BIGINT | The processor number as reported in /proc/cpuinfo |
| turbo\_disabled | BIGINT | Whether the turbo feature is disabled. |
| turbo\_ratio\_limit | BIGINT | The turbo feature ratio limit. |
| platform\_info | BIGINT | Platform information. |
| perf\_ctl | BIGINT | Performance setting for the processor. |
| perf\_status | BIGINT | Performance status for the processor. |
| feature\_control | BIGINT | Bitfield controling enabled features. |
| rapl\_power\_limit | BIGINT | Run Time Average Power Limiting power limit. |
| rapl\_energy\_status | BIGINT | Run Time Average Power Limiting energy status. |
| rapl\_power\_units | BIGINT | Run Time Average Power Limiting power units. |

**nfs\_shares**

NFS shares exported by the host.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| share | TEXT | Filesystem path to the share |
| options | TEXT | Options string set on the export share |
| readonly | INTEGER | 1 if the share is exported readonly else 0 |

**nvram**

Apple NVRAM variable listing.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Variable name |
| type | TEXT | Data type (CFData, CFString, etc) |
| value | TEXT | Raw variable data |

**opera\_extensions**

Opera browser extensions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | The local user that owns the extension |
| name | TEXT | Extension display name |
| identifier | TEXT | Extension identifier |
| version | TEXT | Extension-supplied version |
| description | TEXT | Extension-optional description |
| locale | TEXT | Default locale supported by extension |
| update\_url | TEXT | Extension-supplied update URI |
| author | TEXT | Optional extension author |
| persistent | INTEGER | 1 If extension is persistent across all tabs else 0 |
| path | TEXT | Path to extension folder |

**os\_version**

A single row containing the operating system name and version.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Distribution or product name |
| version | TEXT | Pretty, suitable for presentation, OS version |
| major | INTEGER | Major release version |
| minor | INTEGER | Minor release version |
| patch | INTEGER | Optional patch release |
| build | TEXT | Optional build-specific or variant string |
| platform | TEXT | OS Platform or ID |
| platform\_like | TEXT | Closely related platforms |
| codename | TEXT | OS version codename |

**osquery\_events**

Information about the event publishers and subscribers.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Event publisher or subscriber name |
| publisher | TEXT | Name of the associated publisher |
| type | TEXT | Either publisher or subscriber |
| subscriptions | INTEGER | Number of subscriptions the publisher received or subscriber used |
| events | INTEGER | Number of events emitted or received since osquery started |
| refreshes | INTEGER | Publisher only: number of runloop restarts |
| active | INTEGER | 1 if the publisher or subscriber is active else 0 |

**osquery\_extensions**

List of active osquery extensions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uuid | BIGINT | The transient ID assigned for communication |
| name | TEXT | Extension's name |
| version | TEXT | Extenion's version |
| sdk\_version | TEXT | osquery SDK version used to build the extension |
| path | TEXT | Path of the extenion's domain socket or library path |
| type | TEXT | SDK extension type: extension or module |

**osquery\_flags**

Configurable flags that modify osquery's behavior.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Flag name |
| type | TEXT | Flag type |
| description | TEXT | Flag description |
| default\_value | TEXT | Flag default value |
| value | TEXT | Flag value |
| shell\_only | INTEGER | Is the flag shell only? |

**osquery\_info**

Top level information about the running version of osquery.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | INTEGER | Process (or thread/handle) ID |
| uuid | TEXT | Unique ID provided by the system |
| instance\_id | TEXT | Unique, long-lived ID per instance of osquery |
| version | TEXT | osquery toolkit version |
| config\_hash | TEXT | Hash of the working configuration state |
| config\_valid | INTEGER | 1 if the config was loaded and considered valid, else 0 |
| extensions | TEXT | osquery extensions status |
| build\_platform | TEXT | osquery toolkit build platform |
| build\_distro | TEXT | osquery toolkit platform distribution name (os version) |
| start\_time | INTEGER | UNIX time in seconds when the process started |
| watcher | INTEGER | Process (or thread/handle) ID of optional watcher process |

**osquery\_packs**

Information about the current query packs that are loaded in osquery.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | The given name for this query pack |
| platform | TEXT | Platforms this query is supported on |
| version | TEXT | Minimum osquery version that this query will run on |
| shard | INTEGER | Shard restriction limit, 1-100, 0 meaning no restriction |
| discovery\_cache\_hits | INTEGER | The number of times that the discovery query used cached values since the last time the config was reloaded |
| discovery\_executions | INTEGER | The number of times that the discovery queries have been executed since the last time the config was reloaded |
| active | INTEGER | Whether this pack is active (the version, platform and discovery queries match) yes=1, no=0. |

**osquery\_registry**

List the osquery registry plugins.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| registry | TEXT | Name of the osquery registry |
| name | TEXT | Name of the plugin item |
| owner\_uuid | INTEGER | Extension route UUID (0 for core) |
| internal | INTEGER | 1 If the plugin is internal else 0 |
| active | INTEGER | 1 If this plugin is active else 0 |

**osquery\_schedule**

Information about the current queries that are scheduled in osquery.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | The given name for this query |
| query | TEXT | The exact query to run |
| interval | INTEGER | The interval in seconds to run this query, not an exact interval |
| executions | BIGINT | Number of times the query was executed |
| last\_executed | BIGINT | UNIX time stamp in seconds of the last completed execution |
| blacklisted | INTEGER | 1 if the query is blacklisted else 0 |
| output\_size | BIGINT | Total number of bytes generated by the query |
| wall\_time | BIGINT | Total wall time spent executing |
| user\_time | BIGINT | Total user time spent executing |
| system\_time | BIGINT | Total system time spent executing |
| average\_memory | BIGINT | Average private memory left after executing |

**package\_bom**

OS X package bill of materials (BOM) file list.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| filepath | TEXT | Package file or directory |
| uid | INTEGER | Expected user of file or directory |
| gid | INTEGER | Expected group of file or directory |
| mode | INTEGER | Expected permissions |
| size | BIGINT | Expected file size |
| modified\_time | INTEGER | Timestamp the file was installed |
| path | TEXT | Path of package bom |

**package\_install\_history**

OS X package install history.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| package\_id | TEXT | Label packageIdentifiers |
| time | INTEGER | Label date as UNIX timestamp |
| name | TEXT | Package display name |
| version | TEXT | Package display version |
| source | TEXT | Install source: usually the installer process name |
| content\_type | TEXT | Package content\_type (optional) |

**package\_receipts**

OS X package receipt details.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| package\_id | TEXT | Package domain identifier |
| package\_filename | TEXT | Filename of original .pkg file |
| version | TEXT | Installed package version |
| location | TEXT | Optional relative install path on volume |
| install\_time | DOUBLE | Timestamp of install time |
| installer\_name | TEXT | Name of installer process |
| path | TEXT | Path of receipt plist |

**patches**

Lists all the patches applied. Note: This does not include patches applied via MSI or downloaded from Windows Update (e.g. Service Packs).

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| csname | TEXT | The name of the host the patch is installed on. |
| hotfix\_id | TEXT | The KB ID of the patch. |
| caption | TEXT | Short description of the patch. |
| description | TEXT | Fuller description of the patch. |
| fix\_comments | TEXT | Additional comments about the patch. |
| installed\_by | TEXT | The system context in which the patch as installed. |
| install\_date | TEXT | Indicates when the patch was installed. Lack of a value does not indicate that the patch was not installed. |
| installed\_on | TEXT | The date when the patch was installed. |

**pci\_devices**

PCI devices active on the host system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pci\_slot | TEXT | PCI Device used slot |
| pci\_class | TEXT | PCI Device class |
| driver | TEXT | PCI Device used driver |
| vendor | TEXT | PCI Device vendor |
| vendor\_id | TEXT | Hex encoded PCI Device vendor identifier |
| model | TEXT | PCI Device model |
| model\_id | TEXT | Hex encoded PCI Device model identifier |

**physical\_disk\_performance**

Provides provides raw data from performance counters that monitor hard or fixed disk drives on the system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Name of the physical disk |
| avg\_disk\_bytes\_per\_read | BIGINT | Average number of bytes transferred from the disk during read operations |
| avg\_disk\_bytes\_per\_write | BIGINT | Average number of bytes transferred to the disk during write operations |
| avg\_disk\_read\_queue\_length | BIGINT | Average number of read requests that were queued for the selected disk during the sample interval |
| avg\_disk\_write\_queue\_length | BIGINT | Average number of write requests that were queued for the selected disk during the sample interval |
| avg\_disk\_sec\_per\_read | INTEGER | Average time, in seconds, of a read operation of data from the disk |
| avg\_disk\_sec\_per\_write | INTEGER | Average time, in seconds, of a write operation of data to the disk |
| current\_disk\_queue\_length | INTEGER | Number of requests outstanding on the disk at the time the performance data is collected |
| percent\_disk\_read\_time | BIGINT | Percentage of elapsed time that the selected disk drive is busy servicing read requests |
| percent\_disk\_write\_time | BIGINT | Percentage of elapsed time that the selected disk drive is busy servicing write requests |
| percent\_disk\_time | BIGINT | Percentage of elapsed time that the selected disk drive is busy servicing read or write requests |
| percent\_idle\_time | BIGINT | Percentage of time during the sample interval that the disk was idle |

**pipes**

Named and Anonymous pipes.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | BIGINT | Process ID of the process to which the pipe belongs |
| name | TEXT | Name of the pipe |
| instances | INTEGER | Number of instances of the named pipe |
| max\_instances | INTEGER | The maximum number of instances creatable for this pipe |
| flags | TEXT | The flags indicating whether this pipe connection is a server or client end, and if the pipe for sending messages or bytes |

**pkg\_packages**

pkgng packages that are currently installed on the host system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Package name |
| version | TEXT | Package version |
| flatsize | BIGINT | Package size in bytes |
| arch | TEXT | Architecture(s) supported |

**platform\_info**

Information about EFI/UEFI/ROM and platform/boot.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| vendor | TEXT | Platform code vendor |
| version | TEXT | Platform code version |
| date | TEXT | Self-reported platform code update date |
| revision | TEXT | BIOS major and minor revision |
| address | TEXT | Relative address of firmware mapping |
| size | TEXT | Size in bytes of firmware |
| volume\_size | INTEGER | (Optional) size of firmware volume |
| extra | TEXT | Platform-specific additional information |

**plist**

Read and parse a plist file.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| key | TEXT | Preference top-level key |
| subkey | TEXT | Intemediate key path, includes lists/dicts |
| value | TEXT | String value of most CF types |
| path | TEXT | (optional) read preferences from a plist |

**portage\_keywords**

A summary about portage configurations like keywords, mask and unmask.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| package | TEXT | Package name |
| version | TEXT | The version which are affected by the use flags, empty means all |
| keyword | TEXT | The keyword applied to the package |
| mask | INTEGER | If the package is masked |
| unmask | INTEGER | If the package is unmasked |

**portage\_packages**

List of currently installed packages.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| package | TEXT | Package name |
| version | TEXT | The version which are affected by the use flags, empty means all |
| slot | TEXT | The slot used by package |
| build\_time | BIGINT | Unix time when package was built |
| repository | TEXT | From which repository the ebuild was used |
| eapi | BIGINT | The eapi for the ebuild |
| size | BIGINT | The size of the package |
| world | INTEGER | If package is in the world file |

**portage\_use**

List of enabled portage USE values for specific package.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| package | TEXT | Package name |
| version | TEXT | The version of the installed package |
| use | TEXT | USE flag which has been enabled for package |

**power\_sensors**

Machine power (currents, voltages, wattages, etc) sensors.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| key | TEXT | The SMC key on OS X |
| category | TEXT | The sensor category: currents, voltage, wattage |
| name | TEXT | Name of power source |
| value | TEXT | Power in Watts |

**preferences**

OS X defaults and managed preferences.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| domain | TEXT | Application ID usually in com.name.product format |
| key | TEXT | Preference top-level key |
| subkey | TEXT | Intemediate key path, includes lists/dicts |
| value | TEXT | String value of most CF types |
| forced | INTEGER | 1 if the value is forced/managed, else 0 |
| username | TEXT | (optional) read preferences for a specific user |
| host | TEXT | 'current' or 'any' host, where 'current' takes precedence |

**process\_envs**

A key/value table of environment variables for each process.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | INTEGER | Process (or thread) ID |
| key | TEXT | Environment variable name |
| value | TEXT | Environment variable value |

**process\_events(EVENTED TABLE)**

Track time/action process executions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | BIGINT | Process (or thread) ID |
| path | TEXT | Path of executed file |
| mode | TEXT | File mode permissions |
| cmdline | TEXT | Command line arguments (argv) |
| cmdline\_size | BIGINT | Actual size (bytes) of command line arguments |
| env | TEXT | Environment variables delimited by spaces |
| env\_count | BIGINT | Number of environment variables |
| env\_size | BIGINT | Actual size (bytes) of environment list |
| cwd | TEXT | The process current working directory |
| auid | BIGINT | Audit User ID at process start |
| uid | BIGINT | User ID at process start |
| euid | BIGINT | Effective user ID at process start |
| gid | BIGINT | Group ID at process start |
| egid | BIGINT | Effective group ID at process start |
| owner\_uid | BIGINT | File owner user ID |
| owner\_gid | BIGINT | File owner group ID |
| atime | BIGINT | File last access in UNIX time |
| mtime | BIGINT | File modification in UNIX time |
| ctime | BIGINT | File last metadata change in UNIX time |
| btime | BIGINT | File creation in UNIX time |
| overflows | TEXT | List of structures that overflowed |
| parent | BIGINT | Process parent's PID |
| time | BIGINT | Time of execution in UNIX time |
| uptime | BIGINT | Time of execution in system uptime |
| status | BIGINT | OpenBSM Attribute: Status of the process |
| eid | TEXT | Event ID |

**process\_file\_events(EVENTED TABLE)**

A File Integrity Monitor implementation using the audit service.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| operation | TEXT | Operation type |
| pid | BIGINT | Process ID |
| ppid | BIGINT | Parent process ID |
| time | BIGINT | Time of execution in UNIX time |
| executable | TEXT | The executable path |
| partial | TEXT | True if this is a partial event (i.e.: this process existed before we started osquery) |
| cwd | TEXT | The current working directory of the process |
| path | TEXT | The path associated with the event |
| dest\_path | TEXT | The canonical path associated with the event |
| uid | TEXT | The uid of the process performing the action |
| gid | TEXT | The gid of the process performing the action |
| euid | TEXT | Effective user ID of the process using the file |
| egid | TEXT | Effective group ID of the process using the file |
| uptime | BIGINT | Time of execution in system uptime |
| eid | TEXT | Event ID |

**process\_memory\_map**

Process memory mapped files and pseudo device/regions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | INTEGER | Process (or thread) ID |
| start | TEXT | Virtual start address (hex) |
| end | TEXT | Virtual end address (hex) |
| permissions | TEXT | r=read, w=write, x=execute, p=private (cow) |
| offset | BIGINT | Offset into mapped path |
| device | TEXT | MA:MI Major/minor device ID |
| inode | INTEGER | Mapped path inode, 0 means uninitialized (BSS) |
| path | TEXT | Path to mapped file or mapped type |
| pseudo | INTEGER | 1 If path is a pseudo path, else 0 |

**process\_open\_files**

File descriptors for each process.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | BIGINT | Process (or thread) ID |
| fd | BIGINT | Process-specific file descriptor number |
| path | TEXT | Filesystem path of descriptor |

**process\_open\_sockets**

Processes which have open network sockets on the system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | INTEGER | Process (or thread) ID |
| fd | BIGINT | Socket file descriptor number |
| socket | BIGINT | Socket handle or inode number |
| family | INTEGER | Network protocol (IPv4, IPv6) |
| protocol | INTEGER | Transport protocol (TCP/UDP) |
| local\_address | TEXT | Socket local address |
| remote\_address | TEXT | Socket remote address |
| local\_port | INTEGER | Socket local port |
| remote\_port | INTEGER | Socket remote port |
| path | TEXT | For UNIX sockets (family=AF\_UNIX), the domain path |
| state | TEXT | TCP socket state |
| net\_namespace | TEXT | The inode number of the network namespace |

**processes**

All running processes on the host system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| pid | BIGINT | Process (or thread) ID |
| name | TEXT | The process path or shorthand argv[0] |
| path | TEXT | Path to executed binary |
| cmdline | TEXT | Complete argv |
| state | TEXT | Process state |
| cwd | TEXT | Process current working directory |
| root | TEXT | Process virtual root directory |
| uid | BIGINT | Unsigned user ID |
| gid | BIGINT | Unsigned group ID |
| euid | BIGINT | Unsigned effective user ID |
| egid | BIGINT | Unsigned effective group ID |
| suid | BIGINT | Unsigned saved user ID |
| sgid | BIGINT | Unsigned saved group ID |
| on\_disk | INTEGER | The process path exists yes=1, no=0, unknown=-1 |
| wired\_size | BIGINT | Bytes of unpagable memory used by process |
| resident\_size | BIGINT | Bytes of private memory used by process |
| total\_size | BIGINT | Total virtual memory size |
| user\_time | BIGINT | CPU time spent in user space |
| system\_time | BIGINT | CPU time spent in kernel space |
| disk\_bytes\_read | BIGINT | Bytes read from disk |
| disk\_bytes\_written | BIGINT | Bytes written to disk |
| start\_time | BIGINT | Process start in seconds since boot (non-sleeping) |
| parent | BIGINT | Process parent's PID |
| pgroup | BIGINT | Process group |
| threads | INTEGER | Number of threads used by process |
| nice | INTEGER | Process nice level (-20 to 20, default 0) |

**programs**

Represents products as they are installed by Windows Installer. A product generally correlates to one installation package on Windows. Some fields may be blank as Windows installation details are left to the discretion of the product author.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Commonly used product name. |
| version | TEXT | Product version information. |
| install\_location | TEXT | The installation location directory of the product. |
| install\_source | TEXT | The installation source of the product. |
| language | TEXT | The language of the product. |
| publisher | TEXT | Name of the product supplier. |
| uninstall\_string | TEXT | Path and filename of the uninstaller. |
| install\_date | TEXT | Date that this product was installed on the system. |
| identifying\_number | TEXT | Product identification such as a serial number on software, or a die number on a hardware chip. |

**prometheus\_metrics**

Network interfaces and relevant metadata.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| target\_name | TEXT | Address of prometheus target |
| metric\_name | TEXT | Name of collected Prometheus metric |
| metric\_value | DOUBLE | Value of collected Prometheus metric |
| timestamp\_ms | BIGINT | Unix timestamp of collected data in MS |

**python\_packages**

Python packages installed in a system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Package display name |
| version | TEXT | Package-supplied version |
| summary | TEXT | Package-supplied summary |
| author | TEXT | Optional package author |
| license | TEXT | License under which package is launched |
| path | TEXT | Path at which this module resides |

**quicklook\_cache**

Files and thumbnails within OS X's Quicklook Cache.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Path of file |
| rowid | INTEGER | Quicklook file rowid key |
| fs\_id | TEXT | Quicklook file fs\_id key |
| volume\_id | INTEGER | Parsed volume ID from fs\_id |
| inode | INTEGER | Parsed file ID (inode) from fs\_id |
| mtime | INTEGER | Parsed version date field |
| size | BIGINT | Parsed version size field |
| label | TEXT | Parsed version 'gen' field |
| last\_hit\_date | INTEGER | Apple date format for last thumbnail cache hit |
| hit\_count | TEXT | Number of cache hits on thumbnail |
| icon\_mode | BIGINT | Thumbnail icon mode |
| cache\_path | TEXT | Path to cache data |

**registry**

All of the Windows registry hives.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| key | TEXT | Name of the key to search for |
| path | TEXT | Full path to the value |
| name | TEXT | Name of the registry value entry |
| type | TEXT | Type of the registry value, or 'subkey' if item is a subkey |
| data | TEXT | Data content of registry value |
| mtime | BIGINT | timestamp of the most recent registry write |

**routes**

The active route table for the host system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| destination | TEXT | Destination IP address |
| netmask | TEXT | Netmask length |
| gateway | TEXT | Route gateway |
| source | TEXT | Route source |
| flags | INTEGER | Flags to describe route |
| interface | TEXT | Route local interface |
| mtu | INTEGER | Maximum Transmission Unit for the route |
| metric | INTEGER | Cost of route. Lowest is preferred |
| type | TEXT | Type of route |

**rpm\_package\_files**

RPM packages that are currently installed on the host system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| package | TEXT | RPM package name |
| path | TEXT | Path name |
| username | TEXT | File default username from info DB |
| groupname | TEXT | File default groupname from info DB |
| mode | TEXT | File permissions mode from info DB |
| size | BIGINT | Expected file size in bytes from RPM info DB |
| sha256 | TEXT | SHA256 file digest from RPM info DB |

**rpm\_packages**

RPM packages that are currently installed on the host system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | RPM package name |
| version | TEXT | Package version |
| release | TEXT | Package release |
| source | TEXT | Source RPM package name (optional) |
| size | BIGINT | Package size in bytes |
| sha1 | TEXT | SHA1 hash of the package contents |
| arch | TEXT | Architecture(s) supported |

**safari\_extensions**

Safari browser extension details for all users.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | The local user that owns the extension |
| name | TEXT | Extension display name |
| identifier | TEXT | Extension identifier |
| version | TEXT | Extension long version |
| sdk | TEXT | Bundle SDK used to compile extension |
| update\_url | TEXT | Extension-supplied update URI |
| author | TEXT | Optional extension author |
| developer\_id | TEXT | Optional developer identifier |
| description | TEXT | Optional extension description text |
| path | TEXT | Path to extension XAR bundle |

**sandboxes**

OS X application sandboxes container details.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| label | TEXT | UTI-format bundle or label ID |
| user | TEXT | Sandbox owner |
| enabled | INTEGER | Application sandboxings enabled on container |
| build\_id | TEXT | Sandbox-specific identifier |
| bundle\_path | TEXT | Application bundle used by the sandbox |
| path | TEXT | Path to sandbox container directory |

**scheduled\_tasks**

Lists all of the tasks in the Windows task scheduler.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Name of the scheduled task |
| action | TEXT | Actions executed by the scheduled task |
| path | TEXT | Path to the executable to be run |
| enabled | INTEGER | Whether or not the scheduled task is enabled |
| state | TEXT | State of the scheduled task |
| hidden | INTEGER | Whether or not the task is visible in the UI |
| last\_run\_time | INTEGER | Timestamp the task last ran |
| next\_run\_time | INTEGER | Timestamp the task is scheduled to run next |
| last\_run\_message | TEXT | Exit status message of the last task run |
| last\_run\_code | TEXT | Exit status code of the last task run |

**services**

Lists all installed Windows services and their relevant data.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Service name |
| service\_type | TEXT | Service Type: OWN\_PROCESS, SHARE\_PROCESS and maybe Interactive (can interact with the desktop) |
| display\_name | TEXT | Service Display name |
| status | TEXT | Service Current status: STOPPED, START\_PENDING, STOP\_PENDING, RUNNING, CONTINUE\_PENDING, PAUSE\_PENDING, PAUSED |
| pid | INTEGER | the Process ID of the service |
| start\_type | TEXT | Service start type: BOOT\_START, SYSTEM\_START, AUTO\_START, DEMAND\_START, DISABLED |
| win32\_exit\_code | INTEGER | The error code that the service uses to report an error that occurs when it is starting or stopping |
| service\_exit\_code | INTEGER | The service-specific error code that the service returns when an error occurs while the service is starting or stopping |
| path | TEXT | Path to Service Executable |
| module\_path | TEXT | Path to ServiceDll |
| description | TEXT | Service Description |
| user\_account | TEXT | The name of the account that the service process will be logged on as when it runs. This name can be of the form Domain\UserName. If the account belongs to the built-in domain, the name can be of the form .\UserName. |

**shadow**

Local system users encrypted passwords and related information. Please note, that you usually need superuser rights to access `/etc/shadow`.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| password\_status | TEXT | Password status |
| hash\_alg | TEXT | Password hashing algorithm |
| last\_change | BIGINT | Date of last password change (starting from UNIX epoch date) |
| min | BIGINT | Minimal number of days between password changes |
| max | BIGINT | Maximum number of days between password changes |
| warning | BIGINT | Number of days before password expires to warn user about it |
| inactive | BIGINT | Number of days after password expires until account is blocked |
| expire | BIGINT | Number of days since UNIX epoch date until account is disabled |
| flag | BIGINT | Reserved |
| username | TEXT | Username |

**shared\_folders**

Folders available to others via SMB or AFP.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | The shared name of the folder as it appears to other users |
| path | TEXT | Absolute path of shared folder on the local system |

**shared\_memory**

OS shared memory regions.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| shmid | INTEGER | Shared memory segment ID |
| owner\_uid | BIGINT | User ID of owning process |
| creator\_uid | BIGINT | User ID of creator process |
| pid | BIGINT | Process ID to last use the segment |
| creator\_pid | BIGINT | Process ID that created the segment |
| atime | BIGINT | Attached time |
| dtime | BIGINT | Detached time |
| ctime | BIGINT | Changed time |
| permissions | TEXT | Memory segment permissions |
| size | BIGINT | Size in bytes |
| attached | INTEGER | Number of attached processes |
| status | TEXT | Destination/attach status |
| locked | INTEGER | 1 if segment is locked else 0 |

**shared\_resources**

Displays shared resources on a computer system running Windows. This may be a disk drive, printer, interprocess communication, or other sharable device.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| description | TEXT | A textual description of the object |
| install\_date | TEXT | Indicates when the object was installed. Lack of a value does not indicate that the object is not installed. |
| status | TEXT | String that indicates the current status of the object. |
| allow\_maximum | INTEGER | Number of concurrent users for this resource has been limited. If True, the value in the MaximumAllowed property is ignored. |
| maximum\_allowed | INTEGER | Limit on the maximum number of users allowed to use this resource concurrently. The value is only valid if the AllowMaximum property is set to FALSE. |
| name | TEXT | Alias given to a path set up as a share on a computer system running Windows. |
| path | TEXT | Local path of the Windows share. |
| type | INTEGER | Type of resource being shared. Types include: disk drives, print queues, interprocess communications (IPC), and general devices. |

**sharing\_preferences**

OS X Sharing preferences.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| screen\_sharing | INTEGER | 1 If screen sharing is enabled else 0 |
| file\_sharing | INTEGER | 1 If file sharing is enabled else 0 |
| printer\_sharing | INTEGER | 1 If printer sharing is enabled else 0 |
| remote\_login | INTEGER | 1 If remote login is enabled else 0 |
| remote\_management | INTEGER | 1 If remote management is enabled else 0 |
| remote\_apple\_events | INTEGER | 1 If remote apple events are enabled else 0 |
| internet\_sharing | INTEGER | 1 If internet sharing is enabled else 0 |
| bluetooth\_sharing | INTEGER | 1 If bluetooth sharing is enabled for any user else 0 |
| disc\_sharing | INTEGER | 1 If CD or DVD sharing is enabled else 0 |

**shell\_history**

A line-delimited (command) table of per-user .\*\_history data.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | Shell history owner |
| time | INTEGER | Entry timestamp |
| command | TEXT | Unparsed date/line/command history line |
| history\_file | TEXT | Path to the .\*\_history for this user |

**signature**

File (executable, bundle, installer, disk) code signing status.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Must provide a path or directory |
| signed | INTEGER | 1 If the file is signed else 0 |
| identifier | TEXT | The signing identifier sealed into the signature |
| cdhash | TEXT | SHA1 hash of the application Code Directory |
| team\_identifier | TEXT | The team signing identifier sealed into the signature |
| authority | TEXT | Certificate Common Name |

**sip\_config**

Apple's System Integrity Protection (rootless) status.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| config\_flag | TEXT | The System Integrity Protection config flag |
| enabled | INTEGER | 1 if this configuration is enabled, otherwise 0 |
| enabled\_nvram | INTEGER | 1 if this configuration is enabled, otherwise 0 |

**smbios\_tables**

BIOS (DMI) structure common details and content.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| number | INTEGER | Table entry number |
| type | INTEGER | Table entry type |
| description | TEXT | Table entry description |
| handle | INTEGER | Table entry handle |
| header\_size | INTEGER | Header size in bytes |
| size | INTEGER | Table entry size in bytes |
| md5 | TEXT | MD5 hash of table entry |

**smc\_keys**

Apple's system management controller keys.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| key | TEXT | 4-character key |
| type | TEXT | SMC-reported type literal type |
| size | INTEGER | Reported size of data in bytes |
| value | TEXT | A type-encoded representation of the key value |
| hidden | INTEGER | 1 if this key is normally hidden, otherwise 0 |

**socket\_events(EVENTED TABLE)**

Track network socket opens and closes.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| action | TEXT | The socket action (bind, listen, close) |
| pid | BIGINT | Process (or thread) ID |
| path | TEXT | Path of executed file |
| fd | TEXT | The file description for the process socket |
| auid | BIGINT | Audit User ID |
| success | INTEGER | The socket open attempt status |
| family | INTEGER | The Internet protocol family ID |
| protocol | INTEGER | The network protocol ID |
| local\_address | TEXT | Local address associated with socket |
| remote\_address | TEXT | Remote address associated with socket |
| local\_port | INTEGER | Local network protocol port number |
| remote\_port | INTEGER | Remote network protocol port number |
| socket | TEXT | The local path (UNIX domain socket only) |
| time | BIGINT | Time of execution in UNIX time |
| uptime | BIGINT | Time of execution in system uptime |
| eid | TEXT | Event ID |

**startup\_items**

Applications and binaries set as user/login startup items.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Name of startup item |
| path | TEXT | Path of startup item |
| args | TEXT | Arguments provided to startup executable |
| type | TEXT | Startup Item or Login Item |
| source | TEXT | Directory or plist containing startup item |
| status | TEXT | Startup status; either enabled or disabled |
| username | TEXT | The user associated with the startup item |

**sudoers**

Rules for running commands as other users via sudo.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| header | TEXT | Symbol for given rule |
| rule\_details | TEXT | Rule definition |

**suid\_bin**

suid binaries in common locations.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | Binary path |
| username | TEXT | Binary owner username |
| groupname | TEXT | Binary owner group |
| permissions | TEXT | Binary permissions |

**syslog\_events(EVENTED TABLE)**

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| time | BIGINT | Current unix epoch time |
| datetime | TEXT | Time known to syslog |
| host | TEXT | Hostname configured for syslog |
| severity | INTEGER | Syslog severity |
| facility | TEXT | Syslog facility |
| tag | TEXT | The syslog tag |
| message | TEXT | The syslog message |
| eid | TEXT | Event ID |

**system\_controls**

sysctl names, values, and settings information.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Full sysctl MIB name |
| oid | TEXT | Control MIB |
| subsystem | TEXT | Subsystem ID, control type |
| current\_value | TEXT | Value of setting |
| config\_value | TEXT | The MIB value set in /etc/sysctl.conf |
| type | TEXT | Data type |

**system\_info**

System information for identification.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| hostname | TEXT | Network hostname including domain |
| uuid | TEXT | Unique ID provided by the system |
| cpu\_type | TEXT | CPU type |
| cpu\_subtype | TEXT | CPU subtype |
| cpu\_brand | TEXT | CPU brand string, contains vendor and model |
| cpu\_physical\_cores | INTEGER | Max number of CPU physical cores |
| cpu\_logical\_cores | INTEGER | Max number of CPU logical cores |
| cpu\_microcode | TEXT | Microcode version |
| physical\_memory | BIGINT | Total physical memory in bytes |
| hardware\_vendor | TEXT | Hardware or board vendor |
| hardware\_model | TEXT | Hardware or board model |
| hardware\_version | TEXT | Hardware or board version |
| hardware\_serial | TEXT | Device or board serial number |
| computer\_name | TEXT | Friendly computer name (optional) |
| local\_hostname | TEXT | Local hostname (optional) |

**temperature\_sensors**

Machine's temperature sensors.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| key | TEXT | The SMC key on OS X |
| name | TEXT | Name of temperature source |
| celsius | DOUBLE | Temperature in Celsius |
| fahrenheit | DOUBLE | Temperature in Fahrenheit |

**time**

Track current date and time in the system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| weekday | TEXT | Current weekday in the system |
| year | INTEGER | Current year in the system |
| month | INTEGER | Current month in the system |
| day | INTEGER | Current day in the system |
| hour | INTEGER | Current hour in the system |
| minutes | INTEGER | Current minutes in the system |
| seconds | INTEGER | Current seconds in the system |
| timezone | TEXT | Current timezone in the system |
| local\_time | INTEGER | Current local UNIX time in the system |
| local\_timezone | TEXT | Current local timezone in the system |
| unix\_time | INTEGER | Current UNIX time in the system, converted to UTC if --utc enabled |
| timestamp | TEXT | Current timestamp (log format) in the system |
| datetime | TEXT | Current date and time (ISO format) in the system |
| iso\_8601 | TEXT | Current time (ISO format) in the system |

**time\_machine\_backups**

Backups to drives using TimeMachine.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| destination\_id | TEXT | Time Machine destination ID |
| backup\_date | INTEGER | Backup Date |

**time\_machine\_destinations**

Locations backed up to using Time Machine.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| alias | TEXT | Human readable name of drive |
| destination\_id | TEXT | Time Machine destination ID |
| consistency\_scan\_date | INTEGER | Consistency scan date |
| root\_volume\_uuid | TEXT | Root UUID of backup volume |
| bytes\_available | INTEGER | Bytes available on volume |
| bytes\_used | INTEGER | Bytes used on volume |
| encryption | TEXT | Last known encrypted state |

**uptime**

Track time passed since last boot.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| days | INTEGER | Days of uptime |
| hours | INTEGER | Hours of uptime |
| minutes | INTEGER | Minutes of uptime |
| seconds | INTEGER | Seconds of uptime |
| total\_seconds | BIGINT | Total uptime seconds |

**usb\_devices**

USB devices that are actively plugged into the host system.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| usb\_address | INTEGER | USB Device used address |
| usb\_port | INTEGER | USB Device used port |
| vendor | TEXT | USB Device vendor string |
| vendor\_id | TEXT | Hex encoded USB Device vendor identifier |
| version | TEXT | USB Device version number |
| model | TEXT | USB Device model string |
| model\_id | TEXT | Hex encoded USB Device model identifier |
| serial | TEXT | USB Device serial connection |
| class | TEXT | USB Device class |
| subclass | TEXT | USB Device subclass |
| protocol | TEXT | USB Device protocol |
| removable | INTEGER | 1 If USB device is removable else 0 |

**user\_events(EVENTED TABLE)**

Track user events from the audit framework.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | User ID |
| auid | BIGINT | Audit User ID |
| pid | BIGINT | Process (or thread) ID |
| message | TEXT | Message from the event |
| type | INTEGER | The file description for the process socket |
| path | TEXT | Supplied path from event |
| address | TEXT | The Internet protocol address or family ID |
| terminal | TEXT | The network protocol ID |
| time | BIGINT | Time of execution in UNIX time |
| uptime | BIGINT | Time of execution in system uptime |
| eid | TEXT | Event ID |

**user\_groups**

Local system user group relationships.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | User ID |
| gid | BIGINT | Group ID |

**user\_interaction\_events(EVENTED TABLE)**

Track user interaction events from macOS' event tapping framework.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| time | BIGINT | Time |

**user\_ssh\_keys**

Returns the private keys in the users ~/.ssh directory and whether or not they are encrypted.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | The local user that owns the key file |
| path | TEXT | Path to key file |
| encrypted | INTEGER | 1 if key is encrypted, 0 otherwise |

**users**

Local system users.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| uid | BIGINT | User ID |
| gid | BIGINT | Group ID (unsigned) |
| uid\_signed | BIGINT | User ID as int64 signed (Apple) |
| gid\_signed | BIGINT | Default group ID as int64 signed (Apple) |
| username | TEXT | Username |
| description | TEXT | Optional user description |
| directory | TEXT | User's home directory |
| shell | TEXT | User's configured default shell |
| uuid | TEXT | User's UUID (Apple) |
| type | TEXT | Whether the account is roaming (domain), local, or a system profile |

**video\_info NEW**

Retrieve video card information of the machine.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| color\_depth | INTEGER | The amount of bits per pixel to represent color. |
| driver | TEXT | The driver of the device. |
| driver\_date | TEXT | The date listed on the installed driver. |
| driver\_version | TEXT | The version of the installed driver. |
| manufacturer | TEXT | The manufaturer of the gpu. |
| model | TEXT | The model of the gpu. |
| series | TEXT | The series of the gpu. |
| video\_mode | TEXT | The current resolution of the display. |

**virtual\_memory\_info**

Darwin Virtual Memory statistics.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| free | BIGINT | Total number of free pages. |
| active | BIGINT | Total number of active pages. |
| inactive | BIGINT | Total number of inactive pages. |
| speculative | BIGINT | Total number of speculative pages. |
| throttled | BIGINT | Total number of throttled pages. |
| wired | BIGINT | Total number of wired down pages. |
| purgeable | BIGINT | Total number of purgeable pages. |
| faults | BIGINT | Total number of calls to vm\_faults. |
| copy | BIGINT | Total number of copy-on-write pages. |
| zero\_fill | BIGINT | Total number of zero filled pages. |
| reactivated | BIGINT | Total number of reactivated pages. |
| purged | BIGINT | Total number of purged pages. |
| file\_backed | BIGINT | Total number of file backed pages. |
| anonymous | BIGINT | Total number of anonymous pages. |
| uncompressed | BIGINT | Total number of uncompressed pages. |
| compressor | BIGINT | The number of pages used to store compressed VM pages. |
| decompressed | BIGINT | The total number of pages that have been decompressed by the VM compressor. |
| compressed | BIGINT | The total number of pages that have been compressed by the VM compressor. |
| page\_ins | BIGINT | The total number of requests for pages from a pager. |
| page\_outs | BIGINT | Total number of pages paged out. |
| swap\_ins | BIGINT | The total number of compressed pages that have been swapped out to disk. |
| swap\_outs | BIGINT | The total number of compressed pages that have been swapped back in from disk. |

**wifi\_networks**

OS X known/remembered Wi-Fi networks list.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| ssid | TEXT | SSID octets of the network |
| network\_name | TEXT | Name of the network |
| security\_type | TEXT | Type of security on this network |
| last\_connected | INTEGER | Last time this netword was connected to as a unix\_time |
| passpoint | INTEGER | 1 if Passpoint is supported, 0 otherwise |
| possibly\_hidden | INTEGER | 1 if network is possibly a hidden network, 0 otherwise |
| roaming | INTEGER | 1 if roaming is supported, 0 otherwise |
| roaming\_profile | TEXT | Describe the roaming profile, usually one of Single, Dual or Multi |
| captive\_portal | INTEGER | 1 if this network has a captive portal, 0 otherwise |
| auto\_login | INTEGER | 1 if auto login is enabled, 0 otherwise |
| temporarily\_disabled | INTEGER | 1 if this network is temporarily disabled, 0 otherwise |
| disabled | INTEGER | 1 if this network is disabled, 0 otherwise |

**wifi\_status**

OS X current WiFi status.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| interface | TEXT | Name of the interface |
| ssid | TEXT | SSID octets of the network |
| bssid | TEXT | The current basic service set identifier |
| network\_name | TEXT | Name of the network |
| country\_code | TEXT | The country code (ISO/IEC 3166-1:1997) for the network |
| security\_type | TEXT | Type of security on this network |
| rssi | INTEGER | The current received signal strength indication (dbm) |
| noise | INTEGER | The current noise measurement (dBm) |
| channel | INTEGER | Channel number |
| channel\_width | INTEGER | Channel width |
| channel\_band | INTEGER | Channel band |
| transmit\_rate | TEXT | The current transmit rate |
| mode | TEXT | The current operating mode for the Wi-Fi interface |

**wifi\_survey**

Scan for nearby WiFi networks.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| interface | TEXT | Name of the interface |
| ssid | TEXT | SSID octets of the network |
| bssid | TEXT | The current basic service set identifier |
| network\_name | TEXT | Name of the network |
| country\_code | TEXT | The country code (ISO/IEC 3166-1:1997) for the network |
| rssi | INTEGER | The current received signal strength indication (dbm) |
| noise | INTEGER | The current noise measurement (dBm) |
| channel | INTEGER | Channel number |
| channel\_width | INTEGER | Channel width |
| channel\_band | INTEGER | Channel band |

**windows\_crashes**

Extracted information from Windows crash logs (Minidumps).

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| datetime | TEXT | Timestamp (log format) of the crash |
| module | TEXT | Path of the crashed module within the process |
| path | TEXT | Path of the executable file for the crashed process |
| pid | BIGINT | Process ID of the crashed process |
| tid | BIGINT | Thread ID of the crashed thread |
| version | TEXT | File version info of the crashed process |
| process\_uptime | BIGINT | Uptime of the process in seconds |
| stack\_trace | TEXT | Multiple stack frames from the stack trace |
| exception\_code | TEXT | The Windows exception code |
| exception\_message | TEXT | The NTSTATUS error message associated with the exception code |
| exception\_address | TEXT | Address (in hex) where the exception occurred |
| registers | TEXT | The values of the system registers |
| command\_line | TEXT | Command-line string passed to the crashed process |
| current\_directory | TEXT | Current working directory of the crashed process |
| username | TEXT | Username of the user who ran the crashed process |
| machine\_name | TEXT | Name of the machine where the crash happened |
| major\_version | INTEGER | Windows major version of the machine |
| minor\_version | INTEGER | Windows minor version of the machine |
| build\_number | INTEGER | Windows build number of the crashing machine |
| type | TEXT | Type of crash log |
| crash\_path | TEXT | Path of the log file |

**windows\_events(EVENTED TABLE)**

Windows Event logs.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| time | BIGINT | Timestamp the event was received |
| datetime | TEXT | System time at which the event occurred |
| source | TEXT | Source or channel of the event |
| provider\_name | TEXT | Provider name of the event |
| provider\_guid | TEXT | Provider guid of the event |
| eventid | INTEGER | Event ID of the event |
| task | INTEGER | Task value associated with the event |
| level | INTEGER | The severity level associated with the event |
| keywords | BIGINT | A bitmask of the keywords defined in the event |
| data | TEXT | Data associated with the event |
| eid | TEXT | Event ID |

**wmi\_cli\_event\_consumers**

WMI CommandLineEventConsumer, which can be used for persistance on Windows. See https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf for more details.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Unique name of a consumer. |
| command\_line\_template | TEXT | Standard string template that specifies the process to be started. This property can be NULL, and the ExecutablePath property is used as the command line. |
| executable\_path | TEXT | Module to execute. The string can specify the full path and file name of the module to execute, or it can specify a partial name. If a partial name is specified, the current drive and current directory are assumed. |
| class | TEXT | The name of the class. |
| relative\_path | TEXT | Relative path to the class or instance. |

**wmi\_event\_filters**

Lists WMI event filters.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Unique identifier of an event filter. |
| query | TEXT | Windows Management Instrumentation Query Language (WQL) event query that specifies the set of events for consumer notification, and the specific conditions for notification. |
| query\_language | TEXT | Query language that the query is written in. |
| class | TEXT | The name of the class. |
| relative\_path | TEXT | Relative path to the class or instance. |

**wmi\_filter\_consumer\_binding**

Lists the relationship between event consumers and filters.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| consumer | TEXT | Reference to an instance of \_\_EventConsumer that represents the object path to a logical consumer, the recipient of an event. |
| filter | TEXT | Reference to an instance of \_\_EventFilter that represents the object path to an event filter which is a query that specifies the type of event to be received. |
| class | TEXT | The name of the class. |
| relative\_path | TEXT | Relative path to the class or instance. |

**wmi\_script\_event\_consumers**

WMI ActiveScriptEventConsumer, which can be used for persistance on Windows. See https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf for more details.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Unique identifier for the event consumer. |
| scripting\_engine | TEXT | Name of the scripting engine to use, for example, 'VBScript'. This property cannot be NULL. |
| script\_file\_name | TEXT | Name of the file from which the script text is read, intended as an alternative to specifying the text of the script in the ScriptText property. |
| script\_text | TEXT | Text of the script that is expressed in a language known to the scripting engine. This property must be NULL if the ScriptFileName property is not NULL. |
| class | TEXT | The name of the class. |
| relative\_path | TEXT | Relative path to the class or instance. |

**xprotect\_entries**

Database of the machine's XProtect signatures.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Description of XProtected malware |
| launch\_type | TEXT | Launch services content type |
| identity | TEXT | XProtect identity (SHA1) of content |
| filename | TEXT | Use this file name to match |
| filetype | TEXT | Use this file type to match |
| optional | INTEGER | Match any of the identities/patterns for this XProtect name |
| uses\_pattern | INTEGER | Uses a match pattern instead of identity |

**xprotect\_meta**

Database of the machine's XProtect browser-related signatures.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| identifier | TEXT | Browser plugin or extension identifier |
| type | TEXT | Either plugin or extension |
| developer\_id | TEXT | Developer identity (SHA1) of extension |
| min\_version | TEXT | The minimum allowed plugin version. |

**xprotect\_reports**

Database of XProtect matches (if user generated/sent an XProtect report).

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Description of XProtected malware |
| user\_action | TEXT | Action taken by user after prompted |
| time | TEXT | Quarantine alert time |

**yara**

Track YARA matches for files or PIDs.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| path | TEXT | The path scanned |
| matches | TEXT | List of YARA matches |
| count | INTEGER | Number of YARA matches |
| sig\_group | TEXT | Signature group used |
| sigfile | TEXT | Signature file used |
| strings | TEXT | Matching strings |
| tags | TEXT | Matching tags |

**yara\_events(EVENTED TABLE)**

Track YARA matches for files specified in configuration data.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| target\_path | TEXT | The path scanned |
| category | TEXT | The category of the file |
| action | TEXT | Change action (UPDATE, REMOVE, etc) |
| transaction\_id | BIGINT | ID used during bulk update |
| matches | TEXT | List of YARA matches |
| count | INTEGER | Number of YARA matches |
| strings | TEXT | Matching strings |
| tags | TEXT | Matching tags |
| time | BIGINT | Time of the scan |
| eid | TEXT | Event ID |

**yum\_sources NEW**

Current list of Yum repositories or software channels.

| **COLUMN** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
| name | TEXT | Repository name |
| baseurl | TEXT | Repository base URL |
| enabled | TEXT | Whether the repository is used |
| gpgcheck | TEXT | Whether packages are GPG checked |
| gpgkey | TEXT | URL to GPG key |

The osquery also took consistency and standards to yet-another-level by taking advantage of a feature of SQLite called virtual tables. That enables them to have C/C++/Objective-C "glue" that gets called when a query is made so they can dispatch the intent to the proper functions or shell commands and then send all the results back -- **or** -- use the SQLite engine capabilities to do joining, filtering, UDF-calling, etc to produce rich, targeted rectangular output back.

SQLite Virtual Tables

# **1. Introduction**

A virtual table is an object that is registered with an open SQLite database connection. From the perspective of an SQL statement, the virtual table object looks like any other table or view. But behind the scenes, queries and updates on a virtual table invoke callback methods of the virtual table object instead of reading and writing on the database file.

The virtual table mechanism allows an application to publish interfaces that are accessible from SQL statements as if they were tables. SQL statements can do almost anything to a virtual table that they can do to a real table, with the following exceptions:

* One cannot create a trigger on a virtual table.
* One cannot create additional indices on a virtual table. (Virtual tables can have indices but that must be built into the virtual table implementation. Indices cannot be added separately using CREATE INDEX statements.)
* One cannot run ALTER TABLE ... ADD COLUMN commands against a virtual table.

Individual virtual table implementations might impose additional constraints. For example, some virtual implementations might provide read-only tables. Or some virtual table implementations might allow INSERT or DELETE but not UPDATE. Or some virtual table implementations might limit the kinds of UPDATEs that can be made.

A virtual table might represent an in-memory data structures. Or it might represent a view of data on disk that is not in the SQLite format. Or the application might compute the content of the virtual table on demand.

Here are some existing and postulated uses for virtual tables:

* A full-text search interface
* Spatial indices using R-Trees
* Introspect the disk content of an SQLite database file (the dbstat virtual table)
* Read and/or write the content of a comma-separated value (CSV) file
* Access the filesystem of the host computer as if it were a database table
* Enabling SQL manipulation of data in statistics packages like R

See the list of virtual tables page for a longer list of actual virtual table implementations.

## **1.1. Usage**

A virtual table is created using a CREATE VIRTUAL TABLE statement.

**create-virtual-table-stmt:** hide

CREATEVIRTUALTABLEIFNOTEXISTSschema-name.table-nameUSINGmodule-name(module-argument),

The CREATE VIRTUAL TABLE statement creates a new table called table-name derived from the class class module-name. The module-name is the name that is registered for the virtual table by the sqlite3\_create\_module() interface.

CREATE VIRTUAL TABLE tablename USING modulename;

One can also provide comma-separated arguments to the module following the module name:

CREATE VIRTUAL TABLE tablename USING modulename(arg1, arg2, ...);

The format of the arguments to the module is very general. Each module-argument may contain keywords, string literals, identifiers, numbers, and punctuation. Each module-argument is passed as written (as text) into the constructor method of the virtual table implementation when the virtual table is created and that constructor is responsible for parsing and interpreting the arguments. The argument syntax is sufficiently general that a virtual table implementation can, if it wants to, interpret its arguments as column definitions in an ordinary CREATE TABLE statement. The implementation could also impose some other interpretation on the arguments.

Once a virtual table has been created, it can be used like any other table with the exceptions noted above and imposed by specific virtual table implementations. A virtual table is destroyed using the ordinary DROP TABLE syntax.

### **1.1.1. Temporary virtual tables**

There is no "CREATE TEMP VIRTUAL TABLE" statement. To create a temporary virtual table, add the "temp" schema before the virtual table name.

CREATE VIRTUAL TABLE **temp.**tablename USING module(arg1, ...);

### **1.1.2. Eponymous virtual tables**

Some virtual tables exist automatically in the "main" schema of every database connection in which their module is registered, even without a CREATE VIRTUAL TABLE statement. Such virtual tables are called "eponymous virtual tables". To use an eponymous virtual table, simply use the module name as if it were a table. Eponymous virtual tables exist in the "main" schema only, so they will not work if prefixed with a different schema name.

An example of an eponymous virtual table is the dbstat virtual table. To use the dbstat virtual table as an eponymous virtual table, simply query against the "dbstat" module name, as if it were an ordinary table. (Note that SQLite must be compiled with the SQLITE\_ENABLE\_DBSTAT\_VTAB option to include the dbstat virtual table in the build.)

SELECT \* FROM dbstat;

A virtual table is eponymous if its xCreate method is the exact same function as the xConnect method, or if the xCreate method is NULL. The xCreate method is called when a virtual table is first created using the CREATE VIRTUAL TABLE statement. The xConnect method is invoked whenever a database connection attaches to or reparses a schema. When these two methods are the same, that indicates that the virtual table has no persistent state that needs to be created and destroyed.

### **1.1.3. Eponymous-only virtual tables**

If the xCreate method is NULL, then CREATE VIRTUAL TABLE statements are prohibited for that virtual table, and the virtual table is an "eponymous-only virtual table". Eponymous-only virtual tables are useful as table-valued functions.

Note that prior to version 3.9.0 (2015-10-14), SQLite did not check the xCreate method for NULL before invoking it. So if an eponymous-only virtual table is registered with SQLite version 3.8.11.1 (2015-07-29) or earlier and a CREATE VIRTUAL TABLE command is attempted against that virtual table module, a jump to a NULL pointer will occur, resulting in a crash.

## **1.2. Implementation**

Several new C-level objects are used by the virtual table implementation:

typedef struct sqlite3\_vtab sqlite3\_vtab;

typedef struct sqlite3\_index\_info sqlite3\_index\_info;

typedef struct sqlite3\_vtab\_cursor sqlite3\_vtab\_cursor;

typedef struct sqlite3\_module sqlite3\_module;

The sqlite3\_module structure defines a module object used to implement a virtual table. Think of a module as a class from which one can construct multiple virtual tables having similar properties. For example, one might have a module that provides read-only access to comma-separated-value (CSV) files on disk. That one module can then be used to create several virtual tables where each virtual table refers to a different CSV file.

The module structure contains methods that are invoked by SQLite to perform various actions on the virtual table such as creating new instances of a virtual table or destroying old ones, reading and writing data, searching for and deleting, updating, or inserting rows. The module structure is explained in more detail below.

Each virtual table instance is represented by an sqlite3\_vtab structure. The sqlite3\_vtab structure looks like this:

struct sqlite3\_vtab {

const sqlite3\_module \*pModule;

int nRef;

char \*zErrMsg;

};

Virtual table implementations will normally subclass this structure to add additional private and implementation-specific fields. The nRef field is used internally by the SQLite core and should not be altered by the virtual table implementation. The virtual table implementation may pass error message text to the core by putting an error message string in zErrMsg. Space to hold this error message string must be obtained from an SQLite memory allocation function such as sqlite3\_mprintf() or sqlite3\_malloc(). Prior to assigning a new value to zErrMsg, the virtual table implementation must free any preexisting content of zErrMsg using sqlite3\_free(). Failure to do this will result in a memory leak. The SQLite core will free and zero the content of zErrMsg when it delivers the error message text to the client application or when it destroys the virtual table. The virtual table implementation only needs to worry about freeing the zErrMsg content when it overwrites the content with a new, different error message.

The sqlite3\_vtab\_cursor structure represents a pointer to a specific row of a virtual table. This is what an sqlite3\_vtab\_cursor looks like:

struct sqlite3\_vtab\_cursor {

sqlite3\_vtab \*pVtab;

};

Once again, practical implementations will likely subclass this structure to add additional private fields.

The sqlite3\_index\_info structure is used to pass information into and out of the xBestIndex method of the module that implements a virtual table.

Before a CREATE VIRTUAL TABLE statement can be run, the module specified in that statement must be registered with the database connection. This is accomplished using either of the sqlite3\_create\_module() or sqlite3\_create\_module\_v2() interfaces:

int sqlite3\_create\_module(

sqlite3 \*db, /\* SQLite connection to register module with \*/

const char \*zName, /\* Name of the module \*/

const sqlite3\_module \*, /\* Methods for the module \*/

void \* /\* Client data for xCreate/xConnect \*/

);

int sqlite3\_create\_module\_v2(

sqlite3 \*db, /\* SQLite connection to register module with \*/

const char \*zName, /\* Name of the module \*/

const sqlite3\_module \*, /\* Methods for the module \*/

void \*, /\* Client data for xCreate/xConnect \*/

void(\*xDestroy)(void\*) /\* Client data destructor function \*/

);

The sqlite3\_create\_module() and sqlite3\_create\_module\_v2() routines associates a module name with an sqlite3\_module structure and a separate client data that is specific to each module. The only difference between the two create\_module methods is that the \_v2 method includes an extra parameter that specifies a destructor for client data pointer. The module structure is what defines the behavior of a virtual table. The module structure looks like this:

struct sqlite3\_module {

int iVersion;

int (\*xCreate)(sqlite3\*, void \*pAux,

int argc, char \*const\*argv,

sqlite3\_vtab \*\*ppVTab,

char \*\*pzErr);

int (\*xConnect)(sqlite3\*, void \*pAux,

int argc, char \*const\*argv,

sqlite3\_vtab \*\*ppVTab,

char \*\*pzErr);

int (\*xBestIndex)(sqlite3\_vtab \*pVTab, sqlite3\_index\_info\*);

int (\*xDisconnect)(sqlite3\_vtab \*pVTab);

int (\*xDestroy)(sqlite3\_vtab \*pVTab);

int (\*xOpen)(sqlite3\_vtab \*pVTab, sqlite3\_vtab\_cursor \*\*ppCursor);

int (\*xClose)(sqlite3\_vtab\_cursor\*);

int (\*xFilter)(sqlite3\_vtab\_cursor\*, int idxNum, const char \*idxStr,

int argc, sqlite3\_value \*\*argv);

int (\*xNext)(sqlite3\_vtab\_cursor\*);

int (\*xEof)(sqlite3\_vtab\_cursor\*);

int (\*xColumn)(sqlite3\_vtab\_cursor\*, sqlite3\_context\*, int);

int (\*xRowid)(sqlite3\_vtab\_cursor\*, sqlite\_int64 \*pRowid);

int (\*xUpdate)(sqlite3\_vtab \*, int, sqlite3\_value \*\*, sqlite\_int64 \*);

int (\*xBegin)(sqlite3\_vtab \*pVTab);

int (\*xSync)(sqlite3\_vtab \*pVTab);

int (\*xCommit)(sqlite3\_vtab \*pVTab);

int (\*xRollback)(sqlite3\_vtab \*pVTab);

int (\*xFindFunction)(sqlite3\_vtab \*pVtab, int nArg, const char \*zName,

void (\*\*pxFunc)(sqlite3\_context\*,int,sqlite3\_value\*\*),

void \*\*ppArg);

int (\*Rename)(sqlite3\_vtab \*pVtab, const char \*zNew);

/\* The methods above are in version 1 of the sqlite\_module object. Those

\*\* below are for version 2 and greater. \*/

int (\*xSavepoint)(sqlite3\_vtab \*pVTab, int);

int (\*xRelease)(sqlite3\_vtab \*pVTab, int);

int (\*xRollbackTo)(sqlite3\_vtab \*pVTab, int);

/\* The methods above are in versions 1 and 2 of the sqlite\_module object.

\*\* Those below are for version 3 and greater. \*/

int (\*xShadowName)(const char\*);

};

The module structure defines all of the methods for each virtual table object. The module structure also contains the iVersion field which defines the particular edition of the module table structure. Currently, iVersion is always 3 or less, but in future releases of SQLite the module structure definition might be extended with additional methods and in that case the maximum iVersion value will be increased.

The rest of the module structure consists of methods used to implement various features of the virtual table. Details on what each of these methods do are provided in the sequel.

## **1.3. Virtual Tables And Shared Cache**

Prior to SQLite version 3.6.17 (2009-08-10), the virtual table mechanism assumes that each database connection kept its own copy of the database schema. Hence, the virtual table mechanism could not be used in a database that has shared cache mode enabled. The sqlite3\_create\_module() interface would return an error if shared cache mode is enabled. That restriction was relaxed beginning with SQLite version 3.6.17.

## **1.4. Creating New Virtual Table Implementations**

Follow these steps to create your own virtual table:

1. Write all necessary methods.
2. Create an instance of the sqlite3\_module structure containing pointers to all the methods from step 1.
3. Register your sqlite3\_module structure using one of the sqlite3\_create\_module() or sqlite3\_create\_module\_v2() interfaces.
4. Run a CREATE VIRTUAL TABLE command that specifies the new module in the USING clause.

The only really hard part is step 1. You might want to start with an existing virtual table implementation and modify it to suit your needs. The SQLite source tree contains many virtual table implementations that are suitable for copying, including:

* **templatevtab.c** → A virtual table created specifically to serve as a template for other custom virtual tables.
* **series.c** → Implementation of the generate\_series() table-valued function.
* **json1.c** → Contains the sources for the json\_each() and json\_tree() table-valued functions.
* **csv.c** → A virtual table that reads CSV files.

There are many other virtual table implementations in the SQLite source tree that can be used as examples. Locate these other virtual table implementations by searching for "sqlite3\_create\_module".

You might also want to implement your new virtual table as a loadable extension.

# **2. Virtual Table Methods**

## **2.1. The xCreate Method**

int (\*xCreate)(sqlite3 \*db, void \*pAux,

int argc, char \*const\*argv,

sqlite3\_vtab \*\*ppVTab,

char \*\*pzErr);

The xCreate method is called to create a new instance of a virtual table in response to a CREATE VIRTUAL TABLE statement. If the xCreate method is the same pointer as the xConnect method, then the virtual table is an eponymous virtual table. If the xCreate method is omitted (if it is a NULL pointer) then the virtual table is an eponymous-only virtual table.

The db parameter is a pointer to the SQLite database connection that is executing the CREATE VIRTUAL TABLE statement. The pAux argument is the copy of the client data pointer that was the fourth argument to the sqlite3\_create\_module() or sqlite3\_create\_module\_v2() call that registered the virtual table module. The argv parameter is an array of argc pointers to null terminated strings. The first string, argv[0], is the name of the module being invoked. The module name is the name provided as the second argument to sqlite3\_create\_module() and as the argument to the USING clause of the CREATE VIRTUAL TABLE statement that is running. The second, argv[1], is the name of the database in which the new virtual table is being created. The database name is "main" for the primary database, or "temp" for TEMP database, or the name given at the end of the ATTACH statement for attached databases. The third element of the array, argv[2], is the name of the new virtual table, as specified following the TABLE keyword in the CREATE VIRTUAL TABLE statement. If present, the fourth and subsequent strings in the argv[] array report the arguments to the module name in the CREATE VIRTUAL TABLE statement.

The job of this method is to construct the new virtual table object (an sqlite3\_vtab object) and return a pointer to it in \*ppVTab.

As part of the task of creating a new sqlite3\_vtab structure, this method must invoke sqlite3\_declare\_vtab() to tell the SQLite core about the columns and datatypes in the virtual table. The sqlite3\_declare\_vtab() API has the following prototype:

int sqlite3\_declare\_vtab(sqlite3 \*db, const char \*zCreateTable)

The first argument to sqlite3\_declare\_vtab() must be the same database connection pointer as the first parameter to this method. The second argument to sqlite3\_declare\_vtab() must a zero-terminated UTF-8 string that contains a well-formed CREATE TABLE statement that defines the columns in the virtual table and their data types. The name of the table in this CREATE TABLE statement is ignored, as are all constraints. Only the column names and datatypes matter. The CREATE TABLE statement string need not to be held in persistent memory. The string can be deallocated and/or reused as soon as the sqlite3\_declare\_vtab() routine returns.

The xConnect method can also optionally request special features for the virtual table by making one or more calls to the sqlite3\_vtab\_config() interface:

int sqlite3\_vtab\_config(sqlite3 \*db, int op, ...);

Call calls to sqlite3\_vtab\_config() are optional. But for maximum security, it is recommended that virtual table implementations invoke "sqlite3\_vtab\_config(db, SQLITE\_VTAB\_DIRECTONLY)" if the virtual table will not be used from inside of triggers or views.

The xCreate method need not initialize the pModule, nRef, and zErrMsg fields of the sqlite3\_vtab object. The SQLite core will take care of that chore.

The xCreate should return SQLITE\_OK if it is successful in creating the new virtual table, or SQLITE\_ERROR if it is not successful. If not successful, the sqlite3\_vtab structure must not be allocated. An error message may optionally be returned in \*pzErr if unsuccessful. Space to hold the error message string must be allocated using an SQLite memory allocation function like sqlite3\_malloc() or sqlite3\_mprintf() as the SQLite core will attempt to free the space using sqlite3\_free() after the error has been reported up to the application.

If the xCreate method is omitted (left as a NULL pointer) then the virtual table is an eponymous-only virtual table. New instances of the virtual table cannot be created using CREATE VIRTUAL TABLE and the virtual table can only be used via its module name. Note that SQLite versions prior to 3.9.0 (2015-10-14) do not understand eponymous-only virtual tables and will segfault if an attempt is made to CREATE VIRTUAL TABLE on an eponymous-only virtual table because the xCreate method was not checked for null.

If the xCreate method is the exact same pointer as the xConnect method, that indicates that the virtual table does not need to initialize backing store. Such a virtual table can be used as an eponymous virtual table or as a named virtual table using CREATE VIRTUAL TABLE or both.

### **2.1.1. Hidden columns in virtual tables**

If a column datatype contains the special keyword "HIDDEN" (in any combination of upper and lower case letters) then that keyword it is omitted from the column datatype name and the column is marked as a hidden column internally. A hidden column differs from a normal column in three respects:

* Hidden columns are not listed in the dataset returned by "PRAGMA table\_info",
* Hidden columns are not included in the expansion of a "\*" expression in the result set of a SELECT, and
* Hidden columns are not included in the implicit column-list used by an INSERT statement that lacks an explicit column-list.

For example, if the following SQL is passed to sqlite3\_declare\_vtab():

CREATE TABLE x(a HIDDEN VARCHAR(12), b INTEGER, c INTEGER Hidden);

Then the virtual table would be created with two hidden columns, and with datatypes of "VARCHAR(12)" and "INTEGER".

An example use of hidden columns can be seen in the FTS3 virtual table implementation, where every FTS virtual table contains an FTS hidden column that is used to pass information from the virtual table into FTS auxiliary functions and to the FTS MATCH operator.

### **2.1.2. Table-valued functions**

A virtual table that contains hidden columns can be used like a table-valued function in the FROM clause of a SELECT statement. The arguments to the table-valued function become constraints on the HIDDEN columns of the virtual table.

For example, the "generate\_series" extension (located in the ext/misc/series.c file in the source tree) implements an eponymous virtual table with the following schema:

CREATE TABLE generate\_series(

value,

start HIDDEN,

stop HIDDEN,

step HIDDEN

);

The sqlite3\_module.xBestIndex method in the implementation of this table checks for equality constraints against the HIDDEN columns, and uses those as input parameters to determine the range of integer "value" outputs to generate. Reasonable defaults are used for any unconstrained columns. For example, to list all integers between 5 and 50:

SELECT value FROM generate\_series(5,50);

The previous query is equivalent to the following:

SELECT value FROM generate\_series WHERE start=5 AND stop=50;

Arguments on the virtual table name are matched to hidden columns in order. The number of arguments can be less than the number of hidden columns, in which case the latter hidden columns are unconstrained. However, an error results if there are more arguments than there are hidden columns in the virtual table.

### **2.1.3. WITHOUT ROWID Virtual Tables**

Beginning with SQLite version 3.14.0 (2016-08-08), the CREATE TABLE statement that is passed into sqlite3\_declare\_vtab() may contain a WITHOUT ROWID clause. This is useful for cases where the virtual table rows cannot easily be mapped into unique integers. A CREATE TABLE statement that includes WITHOUT ROWID must define one or more columns as the PRIMARY KEY. Every column of the PRIMARY KEY must individually be NOT NULL and all columns for each row must be collectively unique.

Note that SQLite does not enforce the PRIMARY KEY for a WITHOUT ROWID virtual table. Enforcement is the responsibility of the underlying virtual table implementation. But SQLite does assume that the PRIMARY KEY constraint is valid - that the identified columns really are UNIQUE and NOT NULL - and it uses that assumption to optimize queries against the virtual table.

The rowid column is not accessible on a WITHOUT ROWID virtual table (of course).

The xUpdate method was originally designed around having a ROWID as a single value. The xUpdate method has been expanded to accommodate an arbitrary PRIMARY KEY in place of the ROWID, but the PRIMARY KEY must still be only one column. For this reason, SQLite will reject any WITHOUT ROWID virtual table that has more than one PRIMARY KEY column and a non-NULL xUpdate method.

## **2.2. The xConnect Method**

int (\*xConnect)(sqlite3\*, void \*pAux,

int argc, char \*const\*argv,

sqlite3\_vtab \*\*ppVTab,

char \*\*pzErr);

The xConnect method is very similar to xCreate. It has the same parameters and constructs a new sqlite3\_vtab structure just like xCreate. And it must also call sqlite3\_declare\_vtab() like xCreate. It should also make all of the same sqlite3\_vtab\_config() calls as xCreate.

The difference is that xConnect is called to establish a new connection to an existing virtual table whereas xCreate is called to create a new virtual table from scratch.

The xCreate and xConnect methods are only different when the virtual table has some kind of backing store that must be initialized the first time the virtual table is created. The xCreate method creates and initializes the backing store. The xConnect method just connects to an existing backing store. When xCreate and xConnect are the same, the table is an eponymous virtual table.

As an example, consider a virtual table implementation that provides read-only access to existing comma-separated-value (CSV) files on disk. There is no backing store that needs to be created or initialized for such a virtual table (since the CSV files already exist on disk) so the xCreate and xConnect methods will be identical for that module.

Another example is a virtual table that implements a full-text index. The xCreate method must create and initialize data structures to hold the dictionary and posting lists for that index. The xConnect method, on the other hand, only has to locate and use an existing dictionary and posting lists that were created by a prior xCreate call.

The xConnect method must return SQLITE\_OK if it is successful in creating the new virtual table, or SQLITE\_ERROR if it is not successful. If not successful, the sqlite3\_vtab structure must not be allocated. An error message may optionally be returned in \*pzErr if unsuccessful. Space to hold the error message string must be allocated using an SQLite memory allocation function like sqlite3\_malloc() or sqlite3\_mprintf() as the SQLite core will attempt to free the space using sqlite3\_free() after the error has been reported up to the application.

The xConnect method is required for every virtual table implementation, though the xCreate and xConnect pointers of the sqlite3\_module object may point to the same function if the virtual table does not need to initialize backing store.

## **2.3. The xBestIndex Method**

SQLite uses the xBestIndex method of a virtual table module to determine the best way to access the virtual table. The xBestIndex method has a prototype like this:

int (\*xBestIndex)(sqlite3\_vtab \*pVTab, sqlite3\_index\_info\*);

The SQLite core communicates with the xBestIndex method by filling in certain fields of the sqlite3\_index\_info structure and passing a pointer to that structure into xBestIndex as the second parameter. The xBestIndex method fills out other fields of this structure which forms the reply. The sqlite3\_index\_info structure looks like this:

struct sqlite3\_index\_info {

/\* Inputs \*/

const int nConstraint; /\* Number of entries in aConstraint \*/

const struct sqlite3\_index\_constraint {

int iColumn; /\* Column constrained. -1 for ROWID \*/

unsigned char op; /\* Constraint operator \*/

unsigned char usable; /\* True if this constraint is usable \*/

int iTermOffset; /\* Used internally - xBestIndex should ignore \*/

} \*const aConstraint; /\* Table of WHERE clause constraints \*/

const int nOrderBy; /\* Number of terms in the ORDER BY clause \*/

const struct sqlite3\_index\_orderby {

int iColumn; /\* Column number \*/

unsigned char desc; /\* True for DESC. False for ASC. \*/

} \*const aOrderBy; /\* The ORDER BY clause \*/

/\* Outputs \*/

struct sqlite3\_index\_constraint\_usage {

int argvIndex; /\* if >0, constraint is part of argv to xFilter \*/

unsigned char omit; /\* Do not code a test for this constraint \*/

} \*const aConstraintUsage;

int idxNum; /\* Number used to identify the index \*/

char \*idxStr; /\* String, possibly obtained from sqlite3\_malloc \*/

int needToFreeIdxStr; /\* Free idxStr using sqlite3\_free() if true \*/

int orderByConsumed; /\* True if output is already ordered \*/

double estimatedCost; /\* Estimated cost of using this index \*/

**/\* Fields below are only available in SQLite 3.8.2 and later \*/**

sqlite3\_int64 estimatedRows; /\* Estimated number of rows returned \*/

**/\* Fields below are only available in SQLite 3.9.0 and later \*/**

int idxFlags; /\* Mask of SQLITE\_INDEX\_SCAN\_\* flags \*/

**/\* Fields below are only available in SQLite 3.10.0 and later \*/**

sqlite3\_uint64 colUsed; /\* Input: Mask of columns used by statement \*/

};

Note the warnings on the "estimatedRows", "idxFlags", and colUsed fields. These fields were added with SQLite versions 3.8.2, 3.9.0, and 3.10.0, respectively. Any extension that reads or writes these fields must first check that the version of the SQLite library in use is greater than or equal to appropriate version - perhaps comparing the value returned from sqlite3\_libversion\_number() against constants 3008002, 3009000, and/or 3010000. The result of attempting to access these fields in an sqlite3\_index\_info structure created by an older version of SQLite are undefined.

In addition, there are some defined constants:

#define SQLITE\_INDEX\_CONSTRAINT\_EQ 2

#define SQLITE\_INDEX\_CONSTRAINT\_GT 4

#define SQLITE\_INDEX\_CONSTRAINT\_LE 8

#define SQLITE\_INDEX\_CONSTRAINT\_LT 16

#define SQLITE\_INDEX\_CONSTRAINT\_GE 32

#define SQLITE\_INDEX\_CONSTRAINT\_MATCH 64

#define SQLITE\_INDEX\_CONSTRAINT\_LIKE 65 /\* 3.10.0 and later \*/

#define SQLITE\_INDEX\_CONSTRAINT\_GLOB 66 /\* 3.10.0 and later \*/

#define SQLITE\_INDEX\_CONSTRAINT\_REGEXP 67 /\* 3.10.0 and later \*/

#define SQLITE\_INDEX\_CONSTRAINT\_NE 68 /\* 3.21.0 and later \*/

#define SQLITE\_INDEX\_CONSTRAINT\_ISNOT 69 /\* 3.21.0 and later \*/

#define SQLITE\_INDEX\_CONSTRAINT\_ISNOTNULL 70 /\* 3.21.0 and later \*/

#define SQLITE\_INDEX\_CONSTRAINT\_ISNULL 71 /\* 3.21.0 and later \*/

#define SQLITE\_INDEX\_CONSTRAINT\_IS 72 /\* 3.21.0 and later \*/

#define SQLITE\_INDEX\_CONSTRAINT\_FUNCTION 150 /\* 3.25.0 and later \*/

#define SQLITE\_INDEX\_SCAN\_UNIQUE 1 /\* Scan visits at most 1 row \*/

Use the sqlite3\_vtab\_collation() interface to find the name of the collating sequence that should be used when evaluating the i-th constraint:

const char \*sqlite3\_vtab\_collation(sqlite3\_index\_info\*, int i);

The SQLite core calls the xBestIndex method when it is compiling a query that involves a virtual table. In other words, SQLite calls this method when it is running sqlite3\_prepare() or the equivalent. By calling this method, the SQLite core is saying to the virtual table that it needs to access some subset of the rows in the virtual table and it wants to know the most efficient way to do that access. The xBestIndex method replies with information that the SQLite core can then use to conduct an efficient search of the virtual table.

While compiling a single SQL query, the SQLite core might call xBestIndex multiple times with different settings in sqlite3\_index\_info. The SQLite core will then select the combination that appears to give the best performance.

Before calling this method, the SQLite core initializes an instance of the sqlite3\_index\_info structure with information about the query that it is currently trying to process. This information derives mainly from the WHERE clause and ORDER BY or GROUP BY clauses of the query, but also from any ON or USING clauses if the query is a join. The information that the SQLite core provides to the xBestIndex method is held in the part of the structure that is marked as "Inputs". The "Outputs" section is initialized to zero.

The information in the sqlite3\_index\_info structure is ephemeral and may be overwritten or deallocated as soon as the xBestIndex method returns. If the xBestIndex method needs to remember any part of the sqlite3\_index\_info structure, it should make a copy. Care must be take to store the copy in a place where it will be deallocated, such as in the idxStr field with needToFreeIdxStr set to 1.

Note that xBestIndex will always be called before xFilter, since the idxNum and idxStr outputs from xBestIndex are required inputs to xFilter. However, there is no guarantee that xFilter will be called following a successful xBestIndex.

The xBestIndex method is required for every virtual table implementation.

### **2.3.1. Inputs**

The main thing that the SQLite core is trying to communicate to the virtual table is the constraints that are available to limit the number of rows that need to be searched. The aConstraint[] array contains one entry for each constraint. There will be exactly nConstraint entries in that array.

Each constraint will usually correspond to a term in the WHERE clause or in a USING or ON clause that is of the form

column OP EXPR

Where "column" is a column in the virtual table, OP is an operator like "=" or "<", and EXPR is an arbitrary expression. So, for example, if the WHERE clause contained a term like this:

a = 5

Then one of the constraints would be on the "a" column with operator "=" and an expression of "5". Constraints need not have a literal representation of the WHERE clause. The query optimizer might make transformations to the WHERE clause in order to extract as many constraints as it can. So, for example, if the WHERE clause contained something like this:

x BETWEEN 10 AND 100 AND 999>y

The query optimizer might translate this into three separate constraints:

x >= 10

x <= 100

y < 999

For each such constraint, the aConstraint[].iColumn field indicates which column appears on the left-hand side of the constraint. The first column of the virtual table is column 0. The rowid of the virtual table is column -1. The aConstraint[].op field indicates which operator is used. The SQLITE\_INDEX\_CONSTRAINT\_\* constants map integer constants into operator values. Columns occur in the order they were defined by the call to sqlite3\_declare\_vtab() in the xCreate or xConnect method. Hidden columns are counted when determining the column index.

If the xFindFunction() method for the virtual table is defined, and if xFindFunction() sometimes returns SQLITE\_INDEX\_CONSTRAINT\_FUNCTION or larger, then the constraints might also be of the form:

FUNCTION( column, EXPR)

In this case the aConstraint[].op value is the same as the value returned by xFindFunction() for FUNCTION.

The aConstraint[] array contains information about all constraints that apply to the virtual table. But some of the constraints might not be usable because of the way tables are ordered in a join. The xBestIndex method must therefore only consider constraints that have an aConstraint[].usable flag which is true.

In addition to WHERE clause constraints, the SQLite core also tells the xBestIndex method about the ORDER BY clause. (In an aggregate query, the SQLite core might put in GROUP BY clause information in place of the ORDER BY clause information, but this fact should not make any difference to the xBestIndex method.) If all terms of the ORDER BY clause are columns in the virtual table, then nOrderBy will be the number of terms in the ORDER BY clause and the aOrderBy[] array will identify the column for each term in the order by clause and whether or not that column is ASC or DESC.

In SQLite version 3.10.0 (2016-01-06) and later, the colUsed field is available to indicate which fields of the virtual table are actually used by the statement being prepared. If the lowest bit of colUsed is set, that means that the first column is used. The second lowest bit corresponds to the second column. And so forth. If the most significant bit of colUsed is set, that means that one or more columns other than the first 63 columns are used. If column usage information is needed by the xFilter method, then the required bits must be encoded into either the output idxNum field or idxStr content.

#### **2.3.1.1. LIKE, GLOB, REGEXP, and MATCH functions**

For the LIKE, GLOB, REGEXP, and MATCH operators, the aConstraint[].iColumn value is the virtual table column that is the left operand of the operator. However, if these operators are expressed as function calls instead of operators, then the aConstraint[].iColumn value references the virtual table column that is the second argument to that function:

LIKE(*EXPR*, *column*)  
GLOB(*EXPR*, *column*)  
REGEXP(*EXPR*, *column*)  
MATCH(*EXPR*, *column*)

Hence, as far as the xBestIndex() method is concerned, the following two forms are equivalent:

*column* LIKE *EXPR*  
LIKE(*EXPR*,*column*)

This special behavior of looking at the second argument of a function only occurs for the LIKE, GLOB, REGEXP, and MATCH functions. For all other functions, the aConstraint[].iColumn value references the first argument of the function.

This special feature of LIKE, GLOB, REGEXP, and MATCH does not apply to the xFindFunction() method, however. The xFindFunction() method always keys off of the left operand of an LIKE, GLOB, REGEXP, or MATCH operator but off of the first argument to function-call equivalents of those operators.

### **2.3.2. Outputs**

Given all of the information above, the job of the xBestIndex method it to figure out the best way to search the virtual table.

The xBestIndex method conveys an indexing strategy to the xFilter method through the idxNum and idxStr fields. The idxNum value and idxStr string content are arbitrary as far as the SQLite core is concerned and can have any meaning as long as xBestIndex and xFilter agree on what that meaning is. The SQLite core just copies the information from xBestIndex through to the xFilter method, assuming only that the char sequence referenced via idxStr is NUL terminated.

The idxStr value may be a string obtained from an SQLite memory allocation function such as sqlite3\_mprintf(). If this is the case, then the needToFreeIdxStr flag must be set to true so that the SQLite core will know to call sqlite3\_free() on that string when it has finished with it, and thus avoid a memory leak. The idxStr value may also be a static constant string, in which case the needToFreeIdxStr boolean should remain false.

If the virtual table will output rows in the order specified by the ORDER BY clause, then the orderByConsumed flag may be set to true. If the output is not automatically in the correct order then orderByConsumed must be left in its default false setting. This will indicate to the SQLite core that it will need to do a separate sorting pass over the data after it comes out of the virtual table.

The estimatedCost field should be set to the estimated number of disk access operations required to execute this query against the virtual table. The SQLite core will often call xBestIndex multiple times with different constraints, obtain multiple cost estimates, then choose the query plan that gives the lowest estimate. The SQLite core initializes estimatedCost to a very large value prior to invoking xBestIndex, so if xBestIndex determines that the current combination of parameters is undesirable, it can leave the estimatedCost field unchanged to discourage its use.

If the current version of SQLite is 3.8.2 or greater, the estimatedRows field may be set to an estimate of the number of rows returned by the proposed query plan. If this value is not explicitly set, the default estimate of 25 rows is used.

If the current version of SQLite is 3.9.0 or greater, the idxFlags field may be set to SQLITE\_INDEX\_SCAN\_UNIQUE to indicate that the virtual table will return only zero or one rows given the input constraints. Additional bits of the idxFlags field might be understood in later versions of SQLite.

The aConstraintUsage[] array contains one element for each of the nConstraint constraints in the inputs section of the sqlite3\_index\_info structure. The aConstraintUsage[] array is used by xBestIndex to tell the core how it is using the constraints.

The xBestIndex method may set aConstraintUsage[].argvIndex entries to values greater than zero. Exactly one entry should be set to 1, another to 2, another to 3, and so forth up to as many or as few as the xBestIndex method wants. The EXPR of the corresponding constraints will then be passed in as the argv[] parameters to xFilter.

For example, if the aConstraint[3].argvIndex is set to 1, then when xFilter is called, the argv[0] passed to xFilter will have the EXPR value of the aConstraint[3] constraint.

By default, the SQLite core double checks all constraints on each row of the virtual table that it receives. If such a check is redundant, the xBestFilter method can suppress that double-check by setting aConstraintUsage[].omit.

### **2.3.3. Return Value**

The xBestIndex method should return SQLITE\_OK on success. If any kind of fatal error occurs, an appropriate error code (ex: SQLITE\_NOMEM) should be returned instead.

If xBestIndex returns SQLITE\_CONSTRAINT, that does not indicate an error. Rather, SQLITE\_CONSTRAINT indicates that the particular combination of input parameters specified is insufficient for the virtual table to do its job. This is logically the same as setting the estimatedCost to infinity. If every call to xBestIndex for a particular query plan returns SQLITE\_CONSTRAINT, that means there is no way for the virtual table to be safely used, and the sqlite3\_prepare() call will fail with a "no query solution" error.

### **2.3.4. Enforcing Required Parameters On Table-Valued Functions**

The SQLITE\_CONSTRAINT return from xBestIndex is useful for table-valued functions that have required parameters. If the aConstraint[].usable field is false for one of the required parameter, then the xBestIndex method should return SQLITE\_CONSTRAINT. If a required field does not appear in the aConstraint[] array at all, that means that the corresponding parameter is omitted from the input SQL. In that case, xBestIndex should set an error message in pVTab->zErrMsg and return SQLITE\_ERROR. To summarize:

1. The aConstraint[].usable value for a required parameter is false → return SQLITE\_CONSTRAINT.
2. A required parameter does not appears anywhere in the aConstraint[] array → Set an error message in pVTab->zErrMsg and return SQLITE\_ERROR

The following example will better illustrate the use of SQLITE\_CONSTRAINT as a return value from xBestIndex:

SELECT \* FROM realtab, tablevaluedfunc(realtab.x);

Assuming that the first hidden column of "tablevaluedfunc" is "param1", the query above is semantically equivalent to this:

SELECT \* FROM realtab, tablevaluedfunc

WHERE tablevaluedfunc.param1 = realtab.x;

The query planner must decide between many possible implementations of this query, but two plans in particular are of note:

1. Scan all rows of realtab and for each row, find rows in tablevaluedfunc where param1 is equal to realtab.x
2. Scan all rows of tablevalued func and for each row find rows in realtab where x is equal to tablevaluedfunc.param1.

The xBestIndex method will be invoked once for each of the potential plans above. For plan 1, the aConstraint[].usable flag for for the SQLITE\_CONSTRAINT\_EQ constraint on the param1 column will be true because the right-hand side value for the "param1 = ?" constraint will be known, since it is determined by the outer realtab loop. But for plan 2, the aConstraint[].usable flag for "param1 = ?" will be false because the right-hand side value is determined by an inner loop and is thus an unknown quantity. Because param1 is a required input to the table-valued functions, the xBestIndex method should return SQLITE\_CONSTRAINT when presented with plan 2, indicating that a required input is missing. This forces the query planner to select plan 1.

## **2.4. The xDisconnect Method**

int (\*xDisconnect)(sqlite3\_vtab \*pVTab);

This method releases a connection to a virtual table. Only the sqlite3\_vtab object is destroyed. The virtual table is not destroyed and any backing store associated with the virtual table persists. This method undoes the work of xConnect.

This method is a destructor for a connection to the virtual table. Contrast this method with xDestroy. The xDestroy is a destructor for the entire virtual table.

The xDisconnect method is required for every virtual table implementation, though it is acceptable for the xDisconnect and xDestroy methods to be the same function if that makes sense for the particular virtual table.

## **2.5. The xDestroy Method**

int (\*xDestroy)(sqlite3\_vtab \*pVTab);

This method releases a connection to a virtual table, just like the xDisconnect method, and it also destroys the underlying table implementation. This method undoes the work of xCreate.

The xDisconnect method is called whenever a database connection that uses a virtual table is closed. The xDestroy method is only called when a DROP TABLE statement is executed against the virtual table.

The xDestroy method is required for every virtual table implementation, though it is acceptable for the xDisconnect and xDestroy methods to be the same function if that makes sense for the particular virtual table.

## **2.6. The xOpen Method**

int (\*xOpen)(sqlite3\_vtab \*pVTab, sqlite3\_vtab\_cursor \*\*ppCursor);

The xOpen method creates a new cursor used for accessing (read and/or writing) a virtual table. A successful invocation of this method will allocate the memory for the sqlite3\_vtab\_cursor (or a subclass), initialize the new object, and make \*ppCursor point to the new object. The successful call then returns SQLITE\_OK.

For every successful call to this method, the SQLite core will later invoke the xClose method to destroy the allocated cursor.

The xOpen method need not initialize the pVtab field of the sqlite3\_vtab\_cursor structure. The SQLite core will take care of that chore automatically.

A virtual table implementation must be able to support an arbitrary number of simultaneously open cursors.

When initially opened, the cursor is in an undefined state. The SQLite core will invoke the xFilter method on the cursor prior to any attempt to position or read from the cursor.

The xOpen method is required for every virtual table implementation.

## **2.7. The xClose Method**

int (\*xClose)(sqlite3\_vtab\_cursor\*);

The xClose method closes a cursor previously opened by xOpen. The SQLite core will always call xClose once for each cursor opened using xOpen.

This method must release all resources allocated by the corresponding xOpen call. The routine will not be called again even if it returns an error. The SQLite core will not use the sqlite3\_vtab\_cursor again after it has been closed.

The xClose method is required for every virtual table implementation.

## **2.8. The xEof Method**

int (\*xEof)(sqlite3\_vtab\_cursor\*);

The xEof method must return false (zero) if the specified cursor currently points to a valid row of data, or true (non-zero) otherwise. This method is called by the SQL engine immediately after each xFilter and xNext invocation.

The xEof method is required for every virtual table implementation.

## **2.9. The xFilter Method**

int (\*xFilter)(sqlite3\_vtab\_cursor\*, int idxNum, const char \*idxStr,

int argc, sqlite3\_value \*\*argv);

This method begins a search of a virtual table. The first argument is a cursor opened by xOpen. The next two arguments define a particular search index previously chosen by xBestIndex. The specific meanings of idxNum and idxStr are unimportant as long as xFilter and xBestIndex agree on what that meaning is.

The xBestIndex function may have requested the values of certain expressions using the aConstraintUsage[].argvIndex values of the sqlite3\_index\_info structure. Those values are passed to xFilter using the argc and argv parameters.

If the virtual table contains one or more rows that match the search criteria, then the cursor must be left point at the first row. Subsequent calls to xEof must return false (zero). If there are no rows match, then the cursor must be left in a state that will cause the xEof to return true (non-zero). The SQLite engine will use the xColumn and xRowid methods to access that row content. The xNext method will be used to advance to the next row.

This method must return SQLITE\_OK if successful, or an sqlite error code if an error occurs.

The xFilter method is required for every virtual table implementation.

## **2.10. The xNext Method**

int (\*xNext)(sqlite3\_vtab\_cursor\*);

The xNext method advances a virtual table cursor to the next row of a result set initiated by xFilter. If the cursor is already pointing at the last row when this routine is called, then the cursor no longer points to valid data and a subsequent call to the xEof method must return true (non-zero). If the cursor is successfully advanced to another row of content, then subsequent calls to xEof must return false (zero).

This method must return SQLITE\_OK if successful, or an sqlite error code if an error occurs.

The xNext method is required for every virtual table implementation.

## **2.11. The xColumn Method**

int (\*xColumn)(sqlite3\_vtab\_cursor\*, sqlite3\_context\*, int N);

The SQLite core invokes this method in order to find the value for the N-th column of the current row. N is zero-based so the first column is numbered 0. The xColumn method may return its result back to SQLite using one of the following interface:

* sqlite3\_result\_blob()
* sqlite3\_result\_double()
* sqlite3\_result\_int()
* sqlite3\_result\_int64()
* sqlite3\_result\_null()
* sqlite3\_result\_text()
* sqlite3\_result\_text16()
* sqlite3\_result\_text16le()
* sqlite3\_result\_text16be()
* sqlite3\_result\_zeroblob()

If the xColumn method implementation calls none of the functions above, then the value of the column defaults to an SQL NULL.

To raise an error, the xColumn method should use one of the result\_text() methods to set the error message text, then return an appropriate error code. The xColumn method must return SQLITE\_OK on success.

The xColumn method is required for every virtual table implementation.

## **2.12. The xRowid Method**

int (\*xRowid)(sqlite3\_vtab\_cursor \*pCur, sqlite\_int64 \*pRowid);

A successful invocation of this method will cause \*pRowid to be filled with the rowid of row that the virtual table cursor pCur is currently pointing at. This method returns SQLITE\_OK on success. It returns an appropriate error code on failure.

The xRowid method is required for every virtual table implementation.

## **2.13. The xUpdate Method**

int (\*xUpdate)(

sqlite3\_vtab \*pVTab,

int argc,

sqlite3\_value \*\*argv,

sqlite\_int64 \*pRowid

);

All changes to a virtual table are made using the xUpdate method. This one method can be used to insert, delete, or update.

The argc parameter specifies the number of entries in the argv array. The value of argc will be 1 for a pure delete operation or N+2 for an insert or replace or update where N is the number of columns in the table. In the previous sentence, N includes any hidden columns.

Every argv entry will have a non-NULL value in C but may contain the SQL value NULL. In other words, it is always true that argv[i]!=0 for **i** between 0 and argc-1. However, it might be the case that sqlite3\_value\_type(argv[i])==SQLITE\_NULL.

The argv[0] parameter is the rowid of a row in the virtual table to be deleted. If argv[0] is an SQL NULL, then no deletion occurs.

The argv[1] parameter is the rowid of a new row to be inserted into the virtual table. If argv[1] is an SQL NULL, then the implementation must choose a rowid for the newly inserted row. Subsequent argv[] entries contain values of the columns of the virtual table, in the order that the columns were declared. The number of columns will match the table declaration that the xConnect or xCreate method made using the sqlite3\_declare\_vtab() call. All hidden columns are included.

When doing an insert without a rowid (argc>1, argv[1] is an SQL NULL), on a virtual table that uses ROWID (but not on a WITHOUT ROWID virtual table), the implementation must set \*pRowid to the rowid of the newly inserted row; this will become the value returned by the sqlite3\_last\_insert\_rowid() function. Setting this value in all the other cases is a harmless no-op; the SQLite engine ignores the \*pRowid return value if argc==1 or argv[1] is not an SQL NULL.

Each call to xUpdate will fall into one of cases shown below. Not that references to **argv[i]** mean the SQL value held within the argv[i] object, not the argv[i] object itself.

**argc = 1  
argv[0] ≠ NULL**

DELETE: The single row with rowid or PRIMARY KEY equal to argv[0] is deleted. No insert occurs.

**argc > 1  
argv[0] = NULL**

INSERT: A new row is inserted with column values taken from argv[2] and following. In a rowid virtual table, if argv[1] is an SQL NULL, then a new unique rowid is generated automatically. The argv[1] will be NULL for a WITHOUT ROWID virtual table, in which case the implementation should take the PRIMARY KEY value from the appropriate column in argv[2] and following.

**argc > 1  
argv[0] ≠ NULL  
argv[0] = argv[1]**

UPDATE: The row with rowid or PRIMARY KEY argv[0] is updated with new values in argv[2] and following parameters.

**argc > 1  
argv[0] ≠ NULL  
argv[0] ≠ argv[1]**

UPDATE with rowid or PRIMARY KEY change: The row with rowid or PRIMARY KEY argv[0] is updated with the rowid or PRIMARY KEY in argv[1] and new values in argv[2] and following parameters. This will occur when an SQL statement updates a rowid, as in the statement:

UPDATE table SET rowid=rowid+1 WHERE ...;

The xUpdate method must return SQLITE\_OK if and only if it is successful. If a failure occurs, the xUpdate must return an appropriate error code. On a failure, the pVTab->zErrMsg element may optionally be replaced with error message text stored in memory allocated from SQLite using functions such as sqlite3\_mprintf() or sqlite3\_malloc().

If the xUpdate method violates some constraint of the virtual table (including, but not limited to, attempting to store a value of the wrong datatype, attempting to store a value that is too large or too small, or attempting to change a read-only value) then the xUpdate must fail with an appropriate error code.

If the xUpdate method is performing an UPDATE, then sqlite3\_value\_nochange(X) can be used to discover which columns of the virtual table were actually modified by the UPDATE statement. The sqlite3\_value\_nochange(X) interface returns true for columns that do not change. On every UPDATE, SQLite will first invoke xColumn separately for each unchanging column in the table to obtain the value for that column. The xColumn method can check to see if the column is unchanged at the SQL level by invoking sqlite3\_vtab\_nochange(). If xColumn sees that the column is not being modified, it should return without setting a result using one of the sqlite3\_result\_xxxxx() interfaces. Only in that case sqlite3\_value\_nochange() will be true within the xUpdate method. If xColumn does invoke one or more sqlite3\_result\_xxxxx() interfaces, then SQLite understands that as a change in the value of the column and the sqlite3\_value\_nochange() call for that column within xUpdate will return false.

There might be one or more sqlite3\_vtab\_cursor objects open and in use on the virtual table instance and perhaps even on the row of the virtual table when the xUpdate method is invoked. The implementation of xUpdate must be prepared for attempts to delete or modify rows of the table out from other existing cursors. If the virtual table cannot accommodate such changes, the xUpdate method must return an error code.

The xUpdate method is optional. If the xUpdate pointer in the sqlite3\_module for a virtual table is a NULL pointer, then the virtual table is read-only.

## **2.14. The xFindFunction Method**

int (\*xFindFunction)(

sqlite3\_vtab \*pVtab,

int nArg,

const char \*zName,

void (\*\*pxFunc)(sqlite3\_context\*,int,sqlite3\_value\*\*),

void \*\*ppArg

);

This method is called during sqlite3\_prepare() to give the virtual table implementation an opportunity to overload functions. This method may be set to NULL in which case no overloading occurs.

When a function uses a column from a virtual table as its first argument, this method is called to see if the virtual table would like to overload the function. The first three parameters are inputs: the virtual table, the number of arguments to the function, and the name of the function. If no overloading is desired, this method returns 0. To overload the function, this method writes the new function implementation into \*pxFunc and writes user data into \*ppArg and returns either 1 or a number between SQLITE\_INDEX\_CONSTRAINT\_FUNCTION and 255.

Historically, the return value from xFindFunction() was either zero or one. Zero means that the function is not overloaded and one means that it is overload. The ability to return values of SQLITE\_INDEX\_CONSTRAINT\_FUNCTION or greater was added in version 3.25.0 (2018-09-15). If xFindFunction returns SQLITE\_INDEX\_CONSTRAINT\_FUNCTION or greater, than means that the function takes two arguments and the function can be used as a boolean in the WHERE clause of a query and that the virtual table is able to exploit that function to speed up the query result. When xFindFunction returns SQLITE\_INDEX\_CONSTRAINT\_FUNCTION or larger, the value returned becomes the sqlite3\_index\_info.aConstraint.op value for one of the constraints passed into xBestIndex() and the second argument becomes the value corresponding to that constraint that is passed to xFilter(). This enables the xBestIndex()/xFilter implementations to use the function to speed its search.

The technique of having xFindFunction() return values of SQLITE\_INDEX\_CONSTRAINT\_FUNCTION was initially used in the implementation of the Geopoly module. The xFindFunction() method of that module returns SQLITE\_INDEX\_CONSTRAINT\_FUNCTION for the geopoly\_overlap() SQL function and it returns SQLITE\_INDEX\_CONSTRAINT\_FUNCTION+1 for the geopoly\_within() SQL function. This permits search optimizations for queries such as:

SELECT \* FROM geopolytab WHERE geopoly\_overlap(\_shape, $query\_polygon);

Note that infix functions (LIKE, GLOB, REGEXP, and MATCH) reverse the order of their arguments. So "like(A,B)" would normally work the same as "B like A". However, xFindFunction() always looks a the left-most argument, not the first logical argument. Hence, for the form "B like A", SQLite looks at the left operand "B" and if that operand is a virtual table column it invokes the xFindFunction() method on that virtual table. But if the form "like(A,B)" is used instead, then SQLite checks the A term to see if it is column of a virtual table and if so it invokes the xFindFunction() method for the virtual table of column A.

The function pointer returned by this routine must be valid for the lifetime of the sqlite3\_vtab object given in the first parameter.

## **2.15. The xBegin Method**

int (\*xBegin)(sqlite3\_vtab \*pVTab);

This method begins a transaction on a virtual table. This is method is optional. The xBegin pointer of sqlite3\_module may be NULL.

This method is always followed by one call to either the xCommit or xRollback method. Virtual table transactions do not nest, so the xBegin method will not be invoked more than once on a single virtual table without an intervening call to either xCommit or xRollback. Multiple calls to other methods can and likely will occur in between the xBegin and the corresponding xCommit or xRollback.

## **2.16. The xSync Method**

int (\*xSync)(sqlite3\_vtab \*pVTab);

This method signals the start of a two-phase commit on a virtual table. This is method is optional. The xSync pointer of sqlite3\_module may be NULL.

This method is only invoked after call to the xBegin method and prior to an xCommit or xRollback. In order to implement two-phase commit, the xSync method on all virtual tables is invoked prior to invoking the xCommit method on any virtual table. If any of the xSync methods fail, the entire transaction is rolled back.

## **2.17. The xCommit Method**

int (\*xCommit)(sqlite3\_vtab \*pVTab);

This method causes a virtual table transaction to commit. This is method is optional. The xCommit pointer of sqlite3\_module may be NULL.

A call to this method always follows a prior call to xBegin and xSync.

## **2.18. The xRollback Method**

int (\*xRollback)(sqlite3\_vtab \*pVTab);

This method causes a virtual table transaction to rollback. This is method is optional. The xRollback pointer of sqlite3\_module may be NULL.

A call to this method always follows a prior call to xBegin.

## **2.19. The xRename Method**

int (\*xRename)(sqlite3\_vtab \*pVtab, const char \*zNew);

This method provides notification that the virtual table implementation that the virtual table will be given a new name. If this method returns SQLITE\_OK then SQLite renames the table. If this method returns an error code then the renaming is prevented.

The xRename method is optional. If omitted, then the virtual table may not be renamed using the ALTER TABLE RENAME command.

The PRAGMA legacy\_alter\_table setting is enabled prior to invoking this method, and the value for legacy\_alter\_table is restored after this method finishes. This is necessary for the correct operation of virtual tables that make use of shadow tables where the shadow tables must be renamed to match the new virtual table name. If the legacy\_alter\_format is off, then the xConnect method will be invoked for the virtual table every time the xRename method tries to change the name of the shadow table.

## **2.20. The xSavepoint, xRelease, and xRollbackTo Methods**

int (\*xSavepoint)(sqlite3\_vtab \*pVtab, int);

int (\*xRelease)(sqlite3\_vtab \*pVtab, int);

int (\*xRollbackTo)(sqlite3\_vtab \*pVtab, int);

These methods provide the virtual table implementation an opportunity to implement nested transactions. They are always optional and will only be called in SQLite version 3.7.7 (2011-06-23) and later.

When xSavepoint(X,N) is invoked, that is a signal to the virtual table X that it should save its current state as savepoint N. A subsequent call to xRollbackTo(X,R) means that the state of the virtual table should return to what it was when xSavepoint(X,R) was last called. The call to xRollbackTo(X,R) will invalidate all savepoints with N>R; none of the invalided savepoints will be rolled back or released without first being reinitialized by a call to xSavepoint(). A call to xRelease(X,M) invalidates all savepoints where N>=M.

None of the xSavepoint(), xRelease(), or xRollbackTo() methods will ever be called except in between calls to xBegin() and either xCommit() or xRollback().

## **2.21. The xShadowName Method**

Some virtual table implementations (ex: FTS3, FTS5, and RTREE) make use of real (non-virtual) database tables to store content. For example, when content is inserted into the FTS3 virtual table, the data is ultimately stored in real tables named "%\_content", "%\_segdir", "%\_segments", "%\_stat", and "%\_docsize" where "%" is the name of the original virtual table. This auxiliary real tables that store content for a virtual table are called "shadow tables". See (1), (2), and (3) for additional information.

The xShadowName method exists to allow SQLite to determine whether a certain real table is in fact a shadow table for a virtual table.

SQLite understands a real table to be a shadow table if all of the following are true:

* The name of the table contains one or more "\_" characters.
* The part of the name prior to the last "\_" exactly matches the name of a virtual table that was created using CREATE VIRTUAL TABLE. (Shadow tables are not recognized for eponymous virtual tables and table-valued functions.)
* The virtual table contains an xShadowName method.
* The xShadowName method returns true when its input is the part of the table name past the last "\_" character.

If SQLite recognizes a table as a shadow table, and if the SQLITE\_DBCONFIG\_DEFENSIVE flag is set, then the shadow table is read-only for ordinary SQL statements. The shadow table can still be written, but only by SQL that is invoked from within one of the methods of some virtual table implementation.

The whole point of the xShadowName method is to protect the content of shadow tables from being corrupted by hostile SQL. Every virtual table implementation that uses shadow tables should be able to detect and cope with corrupted shadow table content. However, bugs in particular virtual table implementation might allow a deliberately corrupted shadow table to cause a crash or other malfunction. The xShadowName mechanism seeks to avoid zero-day exploits by preventing ordinary SQL statements from deliberately corrupting shadow tables.

Shadow tables are read/write by default. Shadow tables only become read-only when the SQLITE\_DBCONFIG\_DEFENSIVE flag is set using sqlite3\_db\_config(). Shadow tables need to be read/write by default in order to maintain backwards compatibility. For example, the SQL text generated by the .dump command of the CLI writes directly into shadow tables.

By not reinventing the wheel and relying on well-accepted features like data frames, SQL and SQLite the authors could direct all their focus on solving the problem they posited.

**"Um, you're talking alot about everything but R now."**

We're getting to the good (i.e. "R") part now.

Because the authors didn't try to become SQL parser writer experts and relied on the standard SQL offerings of SQLite, the queries made are "real" SQL (if you've worked with more than one database engine, you know how they all implement different flavours of SQL).

Because these queries are "real" SQL, we can write an R DBI driver for it. The DBI package aims *"[to define] a common interface between R and database management systems (DBMS). The interface defines a small set of classes and methods similar in spirit to Perl's DBI, Java's JDBC, Python's DB-API, and Microsoft's ODBC. It defines a set of classes and methods defines what operations are possible and how they are performed."*

If you look at the osqueryr package source, you'll see a bunch of DBI boilerplate code and only a handful of "touch points" for the actual calls to osqueryi (the command that processes SQL). No handling of anything but passing on SQL to the osqueryi engine and getting rectangular results back. By abstracting the system call details, R users can work with a familiar, consistent, standard interface and have full access to the power of osquery without firing up a terminal.

*But it gets even better.*

As noted above, one design aspect of osquery was to enable remote usage. Rather than come up with yet-another-daemon-and-custom-protocol, the osquery suggest [ssh[](https://www.openssh.com/)](https://www.openssh.com/) as one way of invoking the command on remote systems and getting the rectangular results back.

Because the osqueryr package used the sys[](https://github.com/jeroen/sys) package for making local system calls, there was only a tiny bit of extra effort required to switch from sys::exec\_internal() to a sibling call in the [ssh[](https://github.com/ropensci/ssh)](https://github.com/ropensci/ssh) package -- ssh::ssh\_exec\_internal() when remote connections were specified. (Said effort could have been zero if I chose a slightly different function in sys, too.)

Relying on well-accepted standards made both osqueryi and the R DBI-driver work seamlessly without much code at all and definitely without a rats nest of nested if/else statements and custom httr functions.

**But it gets *even* more better-er**

Some folks like & grok SQL, others don't. (Humans have preferences, *go figure*.)

A few years ago, Hadley (*do I even need to use his last name at this point in time?*) came up with the idea to have a more expressive and consistent way to work with data frames. We now know this as the tidyverse but one core element of the tidyverse is dplyr, which can really level-up your data frame game (no comments about data.table, or the beauty of base R, please). Not too long after the birth of dplyr came the ability to work with remote, rectangular, SQL-based data sources with (mostly) the same idioms.

And, not too long after that, the remote dplyr interface (now, dbplyr) got up close and personal with DBI. Which ultimately means that if you make a near-fully-compliant DBI interface to a SQL back-end you can now do something like this:

library(DBI)

library(dplyr)

library(osqueryr)

con <- DBI::dbConnect(Osquery())

osqdb <- src\_dbi(con)

procs <- tbl(osqdb, "processes")

listen <- tbl(osqdb, "listening\_ports")

left\_join(procs, listen, by="pid") %>%

filter(port != "", protocol == "17") %>% # 17 == TCP

distinct(name, port, address, pid)

## # Source: lazy query [?? x 4]

## # Database: OsqueryConnection

## address name pid port

## 1 0.0.0.0 BetterTouchTool 46317 57183

## 2 0.0.0.0 Dropbox 1214 17500

## 3 0.0.0.0 SystemUIServer 429 0

## 4 0.0.0.0 SystemUIServer 429 62240

## 5 0.0.0.0 UserEventAgent 336 0

## 6 0.0.0.0 WiFiAgent 493 0

## 7 0.0.0.0 WiFiProxy 725 0

## 8 0.0.0.0 com.docker.vpnkit 732 0

## 9 0.0.0.0 identityservicesd 354 0

## 10 0.0.0.0 loginwindow 111 0

## # ... with more rows

The src\_dbi() call wires up everything for us because d[b]plyr can rely on DBI doing it's standard & consistent job and DBI can rely on the SQLite processing crunchy goodness of osqueryi to ultimately get us a list of really dangerous (if not firewalled off) processes that are listening on all network interfaces. (*Note to self: find out why the BetterTouchTool and Dropbox authors feel the need to bind to 0.0.0.0…*)

FIN

What did standards and consistency get us?

* The osquery authors spent time solving a hard problem vs creating new data formats and protocols
* Rectangular data (i.e. "data frame") provides consistency and structure which ends up causing more freedom
* "Standard" SQL enables a consistent means to work with rectangular data
* ssh normalizes (secure) access across systems with a consistent protocol
* A robust, well-defined standard mechanism for working with SQL databases enabled nigh instantaneous wiring up of a whole new back-end to R
* ssh and sys common idioms made working with the new back-end on remote systems as easy as is on a local system
* Another robust, well-defined modern mechanism for working with rectangular data got wired up to this new back-end with (pretty much) one line of code because of the defined standard and expectation of consistency (and works for local and remote)

Introduction to SSH Package

## Installation

This package is available on CRAN and can be installed via:

install.packages('ssh')

Alternatively it can be installed from source using devtools:

remotes::install\_github('ropensci/ssh')

Installation from source on MacOS or Linux requires libssh (the original libssh, **not** the unrelated libssh2 library). On **Debian** or **Ubuntu** use libssh-dev:

sudo apt-get install -y libssh-dev

On **Fedora** we need libssh-devel:

sudo yum install libssh-devel

On **CentOS / RHEL** we install libssh-devel via EPEL:

sudo yum install epel-release

sudo yum install libssh-devel

On **OS-X** use libssh from Homebrew:

brew install libssh

Using **conda** (need a conda R environment conda create -n Renv r-base r-essentials)

conda install --channel conda-forge r-ssh

## Getting Started

First create an ssh session by connecting to an SSH server. You can either use private key or passphrase authentication:

session <- ssh\_connect("jeroen@dev.opencpu.org")

You can use the session in subsequent ssh functions below.

### Run a command

Run a command or script on the host while streaming stdout and stderr directly to the client.

ssh\_exec\_wait(session, command = c(

'curl -fOL https://cloud.r-project.org/src/contrib/Archive/jsonlite/jsonlite\_1.5.tar.gz',

'R CMD check jsonlite\_1.5.tar.gz',

'rm -f jsonlite\_1.5.tar.gz'

))

If you want to capture the stdout/stderr:

out <- ssh\_exec\_internal(session, "R -e 'rnorm(100)'")

cat(rawToChar(out$stdout))

#### Using 'sudo'

Note that the exec functions are non interactive so they cannot prompt for a sudo password. A trick is to use -S which reads the password from stdin:

out <- ssh\_exec\_wait(session, 'echo "mypassword!" | sudo -s -S apt-get update -y')

Be very careful with hardcoding passwords!

### Uploading and Downloading via SCP

Upload and download files via SCP. Directories are automatically traversed as in scp -r.

# Upload a file to the server

file\_path <- R.home("COPYING")

scp\_upload(session, file\_path)

# Download the file back and verify it is the same

scp\_download(session, "COPYING", to = tempdir())

tools::md5sum(file\_path)

tools::md5sum(file.path(tempdir(), "COPYING"))

### Create a Tunnel

Opens a port on your machine and tunnel all traffic to a custom target host via the SSH server.

ssh\_tunnel(session, port = 5555,target = "ds043942.mongolab.com:43942")

This function blocks while the tunnel is active. Use the tunnel by connecting to localhost:5555 from a separate process. The tunnel can only be used once and will automatically be closed when the client disconnects.

### Disconnect

When you are done with the session you should disconnect:

ssh\_disconnect(session)