Tampereen ammattikorkeakoulu

Tiedon siirto Pulseway-etähallintaohjelmiston APIlta tietokantaan

Joni Pohjanoksa

OPINNÄYTETYÖ
Toukokuu 2021

Tieto- ja viestintätekniikan tutkinto-ohjelma
Tietoliikennetekniikka ja tietoverkot

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikan tutkinto-ohjelma
Tietoliikennetekniikka ja tietoverkot

POHJANOKSA, JONI:
Tiedon siirto Pulseway-etähallintaohjelmiston APIlta tietokantaan
Opinnäytetyö 33 sivua, joista liitteitä 6 sivua
Toukokuu 2021

Opinnäytetyön tarkoituksena oli mahdollistaa muokattavien raporttien tekeminen etähallintaohjelmiston datasta. Työssä tarkasteltiin Pulseway-etähallintaohjelmistoa. Työn tarve syntyi Pulsewayn omien raporttipohjien puutteista, sillä niitä ei pysty muokkaamaan työn tilanneen yrityksen tarpeisiin sopeutuviksi. Työn keskeisenä tavoitteena oli tuoda data Pulsewayltä yrityksen omaan tietokantaan. Omasta tietokannasta datan voi jatkossa viedä eteenpäin ja siitä voi tehdä täysin muokattavia hakuja.

Pulsewayn kaikki data saadaan vietyä Pulsewayn APIn kautta omilla palvelimilla olevaan MariaDB-tietokantaan. Datan siirtäminen tietokantaan tapahtuu Python-skriptillä. Skriptissä apuna käytettiin MariaDB-kirjastoa, joka mahdollistaa SQL-kyselyjen tekemisen koodista. Toinen apukirjasto oli slumber, jolla saadaan yhteys Pulsewayn APIin.

Opinnäytetyön tuloksena oli valmis tietokanta ja perustoiminnallisuuden toteuttava skripti. Tietokannan rakenne suunniteltiin sisältämään kaikki APIn data. Datan viite-eheys varmistetaan pääavaimella, joka on Microsoft Windows -käyttöjärjestelmän laitekohtainen tunnus. Valmis skripti täyttää tietokannan APIlta tulevalla datalla, mutta skripti ei pysty muokkaamaan jo tietokannalla olevaa dataa. Ajankohtaisen raportin luomiseksi tietokanta pitää tyhjentää ennen skriptin ajoa. Tyhjentäminen ei kuitenkaan ole ongelma, sillä tietokantaa käytetään vain raporttien luomiseen eikä sinne säilötä dataa.

Jatkossa skriptiä voisi kehittää joustavammaksi. Skriptin SQL-kyselyitä voisi parantaa lisäämällä niihin päivitysehdon duplikaatti avaimella. Tällöin tietokanta päivittäisi kenttien arvot, eikä sitä tarvitsisi poistaa ennen skriptin jokaista ajoa.

Asiasanat: tietokannat, JSON, Python, API

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Telecommunications and Networks

POHJANOKSA, JONI:
Importing Pulseway API Data to Database

Bachelor's thesis 33 pages, appendices 6 pages
May 2021

The purpose of this study was to create a functioning base for generating customizable reports from Pulseway. The reports that Pulseway creates by default were not suited for use at the company. To generate customizable reports, a database was needed. In addition to the database the data retrieved from Pulseway API needed to be handled by a Python script.

The database was built to a specification that required all data to be in a single relational database. Additionally, in order to ensure relational integrity Microsoft Windows device ID was selected as primary key. The Python script that handled data importing used SQL queries embedded in the script to insert data from Pulseway API to the database.

The result was a database built to a specification that worked as intended. The script used for data importing functioned appropriately, though there is room for improvement. As it stands the Python script only allows populating the database once. For up-to-date information to be present in the reports generated from the data in the database, the database needs to be wiped of all existing data. However, this is not an issue since the database is only used as a tool to generate reports and not for storing data.

To further improve the script, exception handling could use upgrading. As it stands, for the test data to be imported into the database only light exception handling is necessary. The second element that could be examined in the future is the SQL queries, which could be changed so that it is not necessary to wipe the database before every run.

Keywords: databases, JSON, Python, API

SISÄLLYS

1 JOHDANTO

Moderni toimistoympäristö on täynnä tietokoneita. Tietokoneiden hallinta ja ylläpito on IT-osastojen tärkeimpiä tehtäviä. Joillain yrityksillä ei ole omaa IT-osastoa, jolloin tietokoneiden hallinta ja ylläpito voi tuntua raskaalta taakalta työntekijöille. Jos koneet takkuavat, työnteosta ei tule mitään. Tällöin yritys voi harkita ICT-palvelujen tilaamista yrityksen ulkopuolelta. JMJping Oy tarjoaa ICT-palveluita kaiken kokoisille yrityksille. Ilman oikeita työkaluja yritys voi joutua pulaan IT-ongelmien kanssa. Muun muassa tietokoneiden etähallintaan ja ylläpitoon suunniteltu irlantilaisen ohjelmistokehittäjän MMSoftin Pulseway (Pulseway 2021). Pulsewayn tärkeimpiin ominaisuuksiin JMJpingin kannalta kuuluu tietokoneiden etähallinta ja sovellusten etäasennus.

Pulsewayn kattavien ominaisuuksien ansiosta se on valittu JMJpingillä asiakkaiden tietokoneiden etähallintaohjelmaksi. Kattavista ominaisuuksista huolimatta Pulsewayn raportointi kaipaisi JMJpingin käyttöön hieman parannusta. Pulseway ei anna tapaa muokata sieltä tulevia raportteja (Ferril, Rash 2019). Raportit generoidaan JMJpingillä itse Pulsewayn APIn datasta. Sieltä tuleva data ajetaan tietokantaan, josta sitä voidaan käsitellä eteenpäin.

Opinnäytetyön tavoite on viedä data Pulsewaylta MariaDB-tietokantaan. Opinnäytetyön alkuperäinen tavoite oli tutustua REST-rajapintaan osana kyseistä vientiä, mutta sen osuus ketjussa oli pienempi kuin alun perin arvioitiin. Opinnäytetyön painopiste siirrettiin Pulsewayhin, tietokannan suunnitteluun, sekä datan vientiin tietokannalle. Tietokannalta voidaan luoda raportit esimerkiksi ODBC:llä Microsoftin Exceliin juuri sellaisena kuin ne halutaan. Raporttien teko on opinnäytetyön rajauksen ulkopuolella.

2 IT-JÄRJESTELMIEN ETÄHALLINTAOHJELMISTOT

2.1 Yleistä

Etähallintaohjelmistot auttavat IT-järjestelmien ylläpitäjiä hallinnoimaan oman yrityksensä laitteita. Etähallintaohjelmistot toimivat alustana yrityksille, jotka tarjoavat IT-palveluita muille yrityksille. Etuna etähallintaohjelmistoissa on työn tehokkuuden kasvu, kulujen leikkaaminen sekä säästetty aika, joka olisi jouduttu käyttämään asiakkaan luona käymiseen. (Moore, Smith 2020).

Tyypillisesti etähallintaohjelmiston ominaisuuksiin kuuluu tietokoneiden päivittäminen ja ohjelmistojen asennus, palvelinten ylläpito ja seuranta sekä verkkolaitteiden tilan seuranta. Monet tällaiset ohjelmistot tarjoavat myös tehtävien automaatiotyökaluja. Automaatiotyökaluilla voidaan keventää pääkäyttäjien työkuormaa automatisoimalla yksinkertaiset tehtävät, jolloin he voivat toimia tehokkaammin (Moore, Smith 2020).

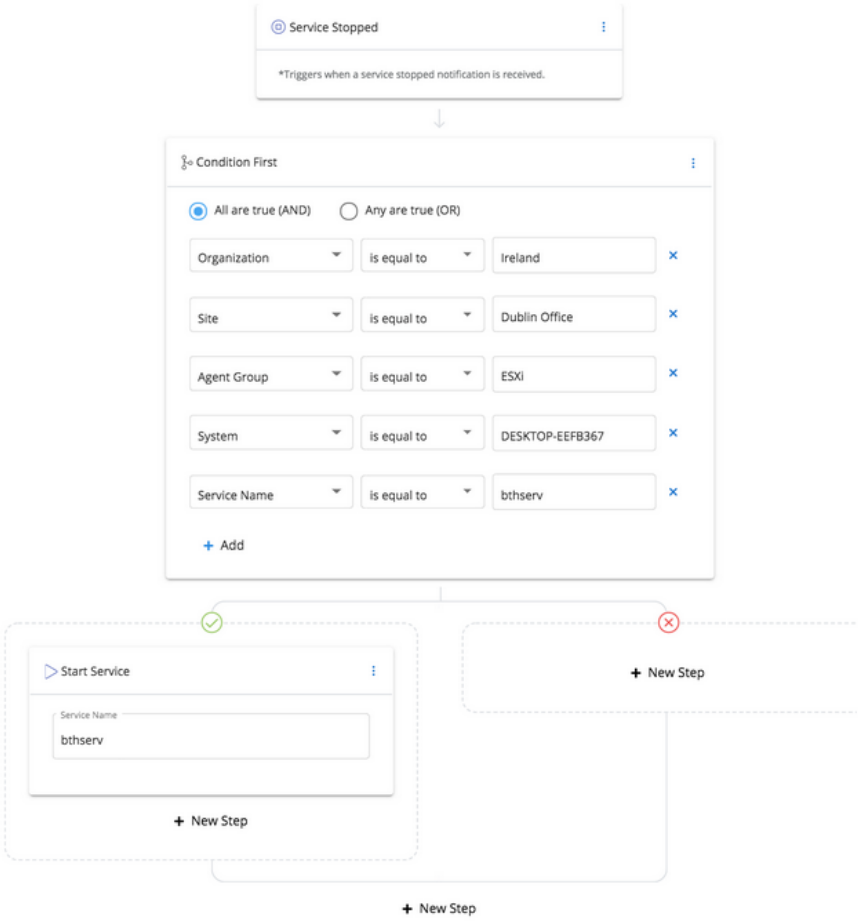
Etähallintaohjelmistot myydään usein tilauspohjaisina palveluina, jolloin yritys maksaa palvelusta kuukausittain. Kaksi yleistä tilausmallia ovat agentti- ja käyttäjäkohtaiset mallit, vaikka muitakin malleja on. Agenttikohtaisessa mallissa laskutetaan hallittujen laitteiden määrän mukaan, kun taas käyttäjäkohtaisessa mallissa laskutetaan perinteisemmin käyttäjien määrän mukaan (NinjaRMM 2020).

Etähallintaohjelmiston hyöty ja tarve riippuvat paljon yrityksen koosta. Suuri yritys, jolla on paljon tietokoneita, hyötyy enemmän kuin yritys, jolla on vain muutama tietokone hallittavana. Jos IT-palvelut ostetaan ulkopuoliselta taholta, yksi kustannus voi myös olla ulkopuolisen tahon etähallintajärjestelmän integraatio yrityksen nykyiseen infrastruktuuriin.

2.2 Pulseway

Pulseway on etähallintaohjelmisto työasemille, palvelimille sekä verkkolaitteille. Yksi Pulsewayn suurimmista vahvuuksista on sen mobiilisovellus. Sovellus on saatavilla Android- ja iOS-laitteille. Sovellus on yhtä tehokas kuin Pulsewayn verkkokäyttöliittymä. Mobiilisovelluksen etuna on tietenkin sen joustavuus, koska sitä voi käyttää missä vain. Sovelluksella voi kirjautua ulos, uudelleen käynnistää ja suorittaa terminaalikomentoja iOS- ja Unix-pohjaisissa järjestelmissä. Windows-koneisiin voi sovelluksella ottaa etäyhteyden täydellä toiminnallisuudella. (Ferril, Rash 2019)

Pulsewayllä pystyy myös automatisoimaan työnkulkua. Pulsewayn käyttäjät voivat valita 17:sta eri laukaisimesta, joihin kuuluu muun muassa korkea prosessorin käyttö, käyttäjän sisään kirjautuminen ja vähäinen levytila (Pulseway 2020). Näiden laukaisimien perusteella voidaan määrittää toiminta, joka suoritetaan laukaisimen lauetessa. Laukaisimia voi myös ketjuttaa yhteen. Ketjutuksen voi toteuttaa JA/TAI-logiikalla. Työnkulun automatisoinnin käyttöliittymä on yksinkertainen ja helppo käyttää. Esimerkki käyttöliittymästä on esillä kuviossa 1.



KUVIO 1. Esimerkki automatisoidusta työnkulusta Pulsewaylla (Csiki 2021)

Monista hyvistä ominaisuuksista huolimatta Pulseway ei kuitenkaan ole täydellinen etähallintaohjelmisto. Pulsewayn raportointi ei ole sen vahvin ominaisuus, saatavilla on 18 valmista raporttipohjaa, joita ei voi muokata tai luoda omia (Ferril, Rash 2019). Valmiit raporttipohjat eivät ole tarpeeksi joustavia JMJpingin käyttöön, tästä syntyi tarve, jonka opinnäytetyön on tarkoitus täyttää.

2.3 Pulsewayn hyödyntäminen yrityksessä

JMJpingillä Pulsewayta käytetään asiakkaiden koneiden monitorointiin, päivitykseen ja hallintaan. Myös Pulsewayn automatisoituja tehtäviä hyödynnetään. Palvelinten ja verkkolaitteiden monitorointiin ja hallintaan JMJpingillä on käytössä toinen ohjelmisto.

Pulseway valittiin etähallintaohjelmistoksi JMJpingillä sen kattavien ominaisuuksien takia. Suurin syy Pulsewayn käyttöön kumminkin on sen

riippumattomuus tenantista. Tenant on Microsoftin Office 365 -palvelun tili, joka sisältää organisaation kaikki käyttäjät ja tilaukset (Vainio 2019). Riippumattomuudella tenantista tarkoitetaan, että ohjelmistoa ei ole sidottu esimerkiksi Microsoftin Office365 tilaukseen, kuten Microsoftin etähallintaohjelmisto Intune. Tämä helpottaa tietokoneiden hallintaa JMJpingin kannalta, koska heillä on hallinnassa useampia eri tenantteja.

Juuri tenantista riippumattomuuden ansiosta Pulsewayn käyttö JMJpingillä jatkuu tulevaisuudessa, vaikka osa asiakkaista siirtyisikin Intuneen. Pulsewayn tarjoama joustavuus ja kätevyys on tervetullut kokemus usein aika kankeisiin etähallintaohjelmisto.

3 TIETOKANTA

3.1 SQL

SQL, eli Structured Query Language, on relaatiotietokantojen ohjelmointikieli. Sen iästä huolimatta, on se eniten käytetty tietokantojen kieli. Kieltä käytetään relaatiotietokantojen hallintaan, joissa data on säilötty tauluissa. SQL:llä voi lisätä, poistaa, päivittää ja hakea dataa tietokannalta. (Picket 2020)

SQL on vanha kieli, vuonna 1969 IBM:n tutkija Edgar F. Codd määritteli relaatiotietokannan perustan, joka toimi SQL:n kehittämisen pohjana. Koska SQL:llä on pitkä historia, sen käyttöä on standardoitu. ISO/IEC 9075-1:2016 on uusin pätevä standardi. Standardin uusin versio lisäsi valinnaista toiminnallisuutta liittyen JSON-tiedostoihin (ISO/IEC 9075-2, 6).

3.2 MariaDB

Työn vaatiman tietokannan alustaksi valittiin MariaDB, joka oli valmiiksi pystytetty JMJpingin palvelimella. Tietokannan rakenne piti suunnitella ja ajaa palvelimelle. Tietokannan taulut suunnitellaan linkitettäviksi toisiinsa tietokoneen tunnuksen perusteella. Tietokannan luettavuuden kannalta joitakin avaimia on muutettava, jotta vältytään toistolta.

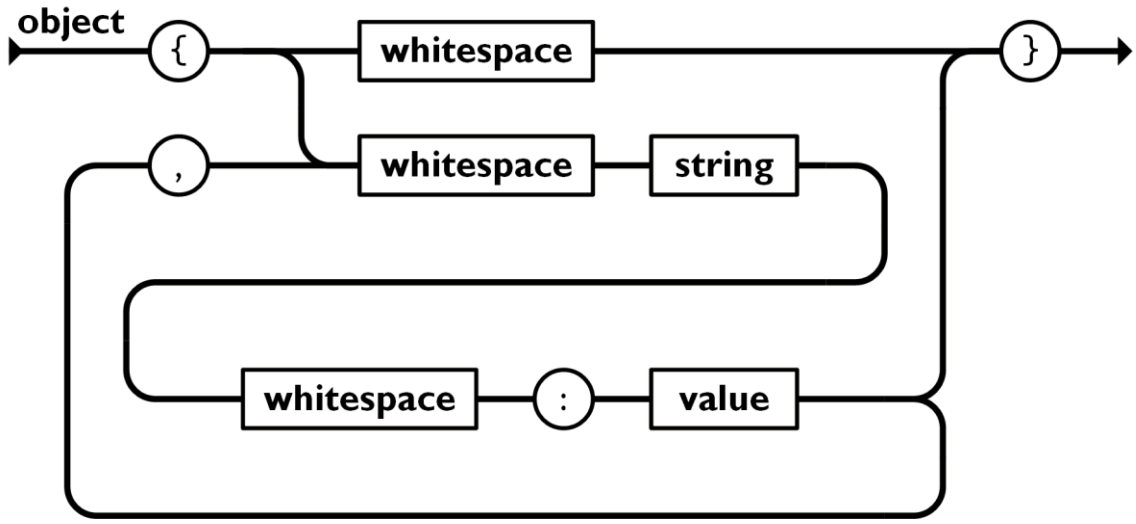
MariaDB on yksi maailman suosituimmista tietokantapalvelimista. Sen ovat kehittäneet MySQL:n alkuperäisen kehittäjät ja sen on taattu pysyvän avoimen lähdekoodin ohjelmistona. MariaDB:n alkuperäinen tarkoitus oli olla korvaaja MySQL:lle. Sitä käytetään, sillä se on nopea, skaalautuva sekä robusti. MariaDB tarjoaa relaatiotietokannan hallintaan SQL-käyttöliittymän. (About MariaDB 2021)

3.3 JSON-formaatti

JSON eli JavaScript Object Notation on tiedon siirtämiseen käytettävä dataformaatti. Se perustuu JavaScript-ohjelmointikielen standardiin ECMA-262:n kolmanteen painokseen joulukuulta 1999. JSON on tekstiformaatti, joka toimii täysin itsenäisesti, ollen riippumaton muista ohjelmointikielistä. (json.org n.d.)

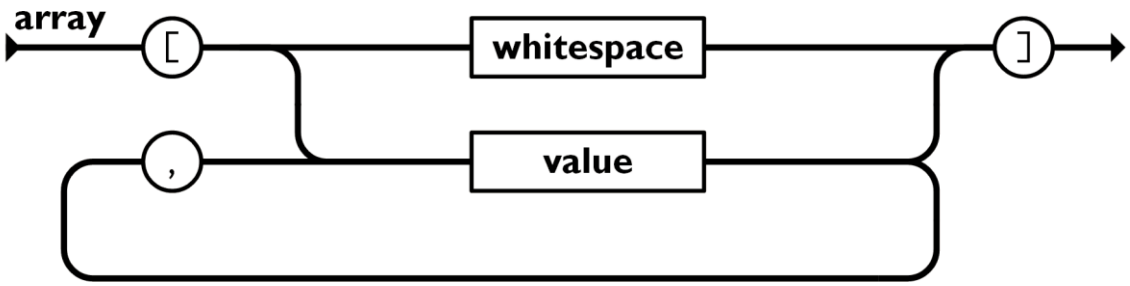
JSON-formaatti perustuu kahdelle rakenteelle. JSON-rakenteen muodostavat avain–arvo-parit, joita useimmissa ohjelmointikielissä kutsutaan objekteiksi, sekä arvot sisältävät järjestetyt listat, joita taas kutsutaan taulukoiksi. (json.org n.d.)

JSON-objektin aloittaa aina vasen aaltosulje ”{” ja sen päättää oikea aaltosulje ”}”. Aaltosulkeiden sisällä ovat avain–arvo-parit erottaa pilkku ”,” ja avaimen arvosta erottaa kaksoispiste ”:”. Kaavio JSON-objektista on esitetty kuviossa 2.



KUVIO 2. JSON-objektin kaavio (json.org n.d.)

JSON-taulukon rakenne vastaavasti alkaa vasemmalla hakasulkeella ”[” ja sen päättää oikea hakasulje ”]”. Taulukossa arvot toisistaan erottaa pilkku ”,”. Kaavio JSON-objektista on esitetty kuviossa 3.



KUVIO 3. JSON-taulukon kaavio (json.org n.d.)

Edellä esitetyissä kaavioissa arvon paikalla voi olla string, numero, objekti tai taulukko. Se voi myös olla tosi, epätosi tai NULL. Opinnäytetyössä käytetystä JSON-rakenteesta on esillä malli kuvassa 1. Kuvassa näkyy koko rakenteen toinen objekti, jonka sisällä on taulukko *data*. Taulukon ensimmäinen arvo on koko tiedoston kolmas objekti, jonka sisällä on lukuisia avain–arvo-pareja ja taulukoita. Kuvassa 1 on vain ote koko rakenteesta.


```
{
    "data": [
        {
            "identifier": "11111111-2222-3333-4444-555555555555",
            "name": "Computer1",
            "group": "Group1",
            "description": "Windows 10 Enterprise",
            "tags": [
                "development"
            ],
            "type": "windows",
            "client_version": "5.1.2",
            "last_updated": "",
            "last_seen_online": "2017-08-11T15:10:06Z",
            "last_reboot": "",
            "external_url": "https://my.pulseway.com/app/main/systems/11111111-2222-3333-4444-555555555555/
details",
            "cpu_usage": 19,
            "memory_usage": 55,
            "memory_total": 8589398016,
            "firewall_enabled": false,
            "antivirus_enabled": "enabled",
            "antivirus_up_to_date": "disabled",
            "uac_enabled": true,
            "event_logs": {
                "error": 2159,
                "warning": 928,
                "information": 55353
            },
            "updates": {
                "critical": 12,
                "important": 1,
                "moderate": 0,
                "low": 0,
                "unspecified": 0
            },
```


KUVA 1. Pulsewayn APIlta tuleva JSON-rakenne (Pulseway n.d.)

3.4 APIn avainten muuttaminen tietokantaan sopiviksi

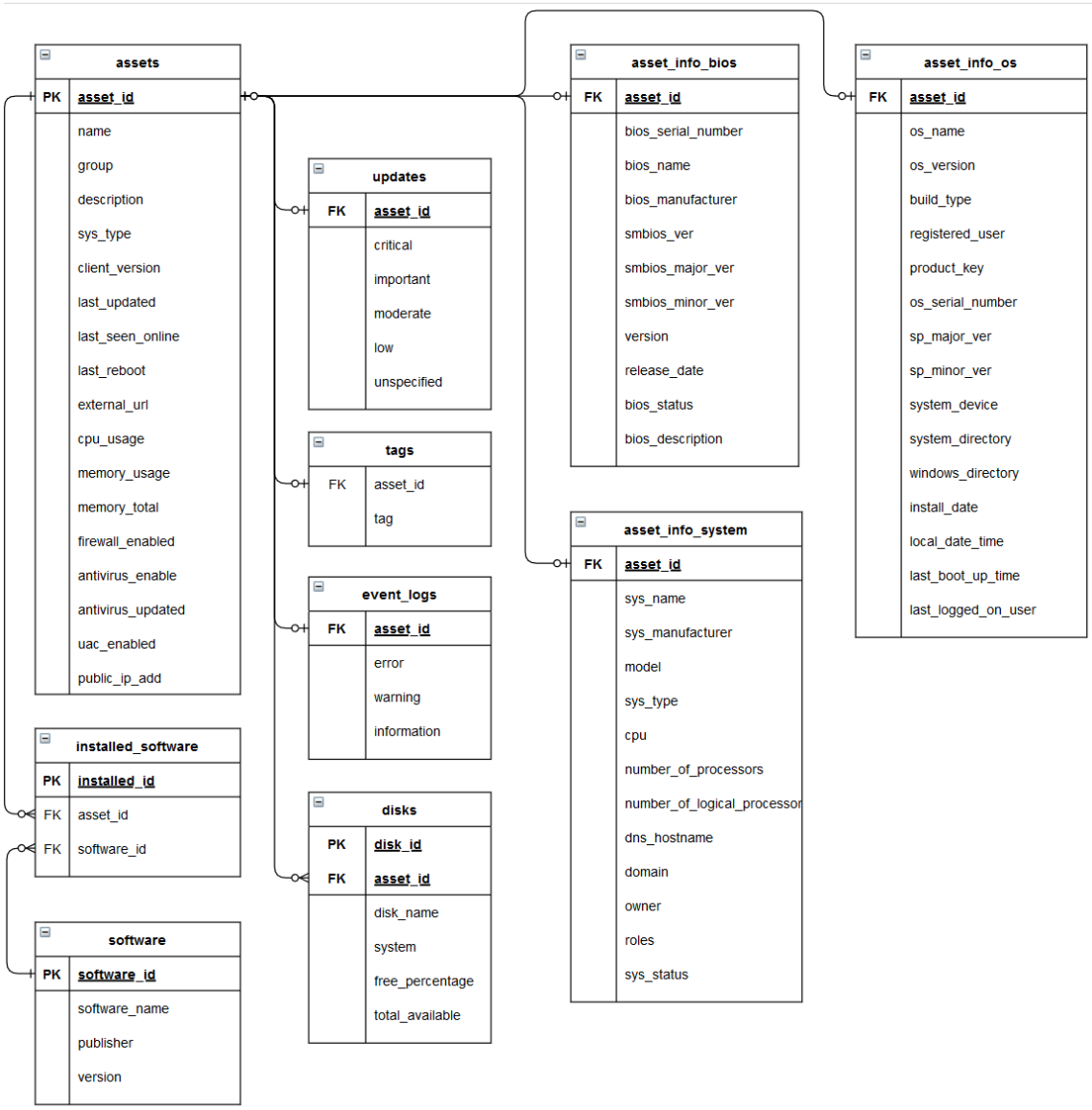
Pulseway APIlta tuleva JSON-tiedosto on valmiiksi segmentoitu kuvaileviin otsikoihin ja avaimiin. Tietokanta suunniteltiin vastaamaan tätä rakennetta, jotta datan syöttäminen tietokantaan olisi mahdollisimman yksinkertaista. Suurin muutos oli joidenkin avainten nimen muuttaminen kuvailevammaksi. Muutetut avaimet ovat esillä taulukossa 1. Tietokantaan tiedon tuomisen kannalta on tärkeää tietää sekä alkuperäinen että muutettu avain. Alkuperäistä avainta käytetään lukemaan tieto JSON-rakenteesta ja muutetulla avaimella se syötetään tietokantaan, tästä kerrotaan tarkemmin luvussa 4. Korostettujen muutosten lisäksi tekstiasu on yhtenäistetty korvaamalla välilyönnit alaviivoilla (_) ja isot kirjaimet pienillä.

TAULUKKO 1. Taulujen avainten muutokset

Taulu	Alkuperäinen	Muutettu
assets	identifier	asset_id
	type	sys_type
	antivirus_up_to_date	antivirus_updated
	public_ip_address	public_ip_add
asset_info_systems	Name	sys_name
	Manufacturer	sys_manufacturer
	Type	sys_type
	DNS Host Name	dns_hostname
	Owner Name	owner
	Status	sys_status
asset_info_bios	Serial Number	bios_serial_number
	Name	bios_name
	Manufacturer	bios_manufacturer
	SMBIOS Version	smbios_ver
	SMBIOS Major Version	smbios_major_ver
	SMBIOS Minor Version	smbios_minor_ver
	Status	bios_status
	Description	bios_description
asset_info_os	Name	os_name
	Version	os_version
	Serial Number	os_serial_number
	Service Pack Major Version	sp_major_ver
	Service Pack Minor Version	sp_minor_ver
	Local Date and Time	local_date_time
tags	tags	tag
disks	name	disk_name
software	name	software_name

3.5 Tietokannan rakenne

JSON-tiedoston rakenteen mukaisen tietokannan rakenne on esitetty kuviossa 4. Lähes kaikelle JSON-tiedostolta tulevalle tiedolle on paikka tietokannassa. Ainoastaan *ip_address*-otsikolle ei ole taulua, koska se on tyhjä jo APIlta tullessa.



KUVIO 4. Tietokannan rakenne

Taulujen väliset yhteydet ovat suurelta osin yksi–yhteen-yhteyksiä. Poikkeus tähän on yksi–moneen-yhteys *assets*- ja *disks*-taulujen välillä. Loogisesti yhdellä tietokoneella voi olla enemmän levyasemia kuin yksi. Moni–moneen-yhteys *assets*- ja *software*-taulun välillä on purettu assosiatiivisella taululla. Vaikka se rakenteessa näyttäisikin selkeämmältä on se syytä purkaa (Hovi, Huotari & Lahdenmäki 2005, 44). Assosiatiivinen taulu keventää tietokannan vaatimaa

suorituskykyä, sillä sen ei tarvitse kirjoittaa jokaista asennettua ohjelmaa jokaisen tietokoneen kohdalle vaan voidaan viitata sen indeksiin.

Yksi–yhteen-yhteydet ovat harvinaisia ja merkki lyhytnäköisestä suunnittelusta (Hovi, Huotari & Lahdenmäki 2005, 37). Yksi–yhteen-yhteydet kuitenkin tähän käyttötarkoitukseen ovat sopivia johtuen JSON-tiedoston rakenteesta, jossa jokaiseen tietokoneeseen liittyvä tieto on yhdessä paikassa, eikä tietokoneeseen liity useampaa tietuetta saman otsikon alla.

Assets-taulun pääavain on *asset_id*, joka on Windowsin määrittämä laitetunnus. Se on uniikki, eikä se muutu, joten on loogista käyttää sitä taulun pääavaimena. Sen avulla kaikki taulut, lukuun ottamatta software-taulua, myös indeksoidaan vierasavaimen avulla. Näin SQL-hakukomennoilla voi hakea tauluista juuri ne tiedot mitä tarvitaan ja viite-eheys säilyy.

Software-tauluun kirjoitetaan jokainen ohjelma, jolla on uniikki nimi ja versio kombinaatio. Sille annetaan juokseva numero tunnisteeksi, jota käytetään taulun pääavaimena: *software_id*. Indeksoimalla *software_id*:tä sekä *asset_id*:tä tauluun *installed_software* lisätään jokaisen koneen asennetut ohjelmat. Menetelmä minimoi tietokantaan toistuvan datan kirjoittamisen.

3.6 Tietokannan luominen

Tässä luvussa annetaan esimerkkejä taulujen luomisesta *pulseway*-tietokantaan. SQL-komennot ovat näkyvillä kuvissa 2-5. Ensimmäisenä esimerkkinä kuvassa 1 on *assets*-taulu, jossa ainut tavallisesta poikkeava on pääavain.

```
CREATE TABLE assets (
        asset_id VARCHAR(36) NOT NULL PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        groups VARCHAR(100),
        description VARCHAR(100),
        sys_type VARCHAR(10),
        client_version VARCHAR(10),
        last_updated VARCHAR(50),
        last_seen_online VARCHAR(50),
        last_reboot VARCHAR(50),
        external_url VARCHAR(100),
        cpu_usage INT,
        memory_usage INT,
        memory_total INT,
        firewall_enabled BOOLEAN,
        antivirus_enabled VARCHAR(10),
        antivirus_updated VARCHAR(10),
        uac_enabled BOOLEAN,
        public_ip_add VARCHAR(15)
);
```

KUVA 2. *Assets*-taulun luominen

Sarakkeille valitut datatyypit on valittu suurin piirtein riittäviksi. Koska tietokantaa käytetään vain apuvälineenä raporttien tekoon ei tarvitse huolehtia sen paisumisesta liian suureksi. Kuvassa 3 on kuvattu *asset_info_bios*-taulun luominen.

```
CREATE TABLE asset_info_bios (
        asset_id VARCHAR(36) NOT NULL UNIQUE,
        bios_serial_number VARCHAR(10),
        bios_name VARCHAR(20),
        manufacturer VARCHAR(20),
        smbios_ver VARCHAR(20),
        smbios_major_ver INT,
        smbios_minor_ver INT,
        bios_version VARCHAR(20),
        release_date VARCHAR(100),
        bios_status VARCHAR(10),
        bios_description VARCHAR(20),
        CONSTRAINT fk_info_bios
                FOREIGN KEY asset_id REFERENCES pulseway.assets (asset_id)
                ON DELETE CASCADE
                ON UPDATE RESTRICT
);
```

KUVA 3. *Asset_info_bios*-taulun luominen

Kuvassa näkyy, kuinka *asset_id*:stä määritetään uniikki. Sen määritys estää useamman saman *asset_id*:n omaavan rivin luomisen. Rajoite (*constraint*) *fk_info_bios* määrittää linkin kyseisen taulun sekä assets-taulun väliin ja varmistaa viite-eheyden dataa poistettaessa. Tämän mallin mukaan luotiin suurin

osa muistakin tauluista. Seuraavana esimerkkinä kuvassa 33 on *installed_software*-taulun luominen.

```
CREATE TABLE installed_software (
        installed_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        asset_id VARCHAR(100),
        software_id INT,
        CONSTRAINT fk_installed_software
                FOREIGN KEY (asset_id) REFERENCES assets (asset_id)
                ON DELETE CASCADE
                ON UPDATE RESTRICT,
                FOREIGN KEY (software_id) REFERENCES software (software_id)
                ON DELETE CASCADE
                ON UPDATE RESTRICT
);
```

KUVA 4. *Installed_software*-taulun luominen

Aikaisemman esimerkin mukaan rajoite *fk_installed_software* määrittää linkin *assets*-tauluun sekä *software*-tauluun. Vielä yhtenä esimerkkinä kuvassa 5 *software*-taulun luominen.

```
CREATE TABLE software (
        software_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100),
        publisher VARCHAR(100),
        version VARCHAR(100),
        CONSTRAINT 'unique_software' UNIQUE (name,version)
);
```

KUVA 5. *Software*-taulun luominen

Tässä rajoitetta käytetään varmistamaan, että ohjelmia ei kirjoiteta useampaan kertaan, ellei sen versio muutu. Tätä sovellettiin myös *disks*-taulussa, jossa jotta saatiin kerättyä koneen kaikki levyt pitää loogisen levyasematunnuksen sekä *asset_id*:n olla uniikit.

4 PYTHON-SKRIPTI

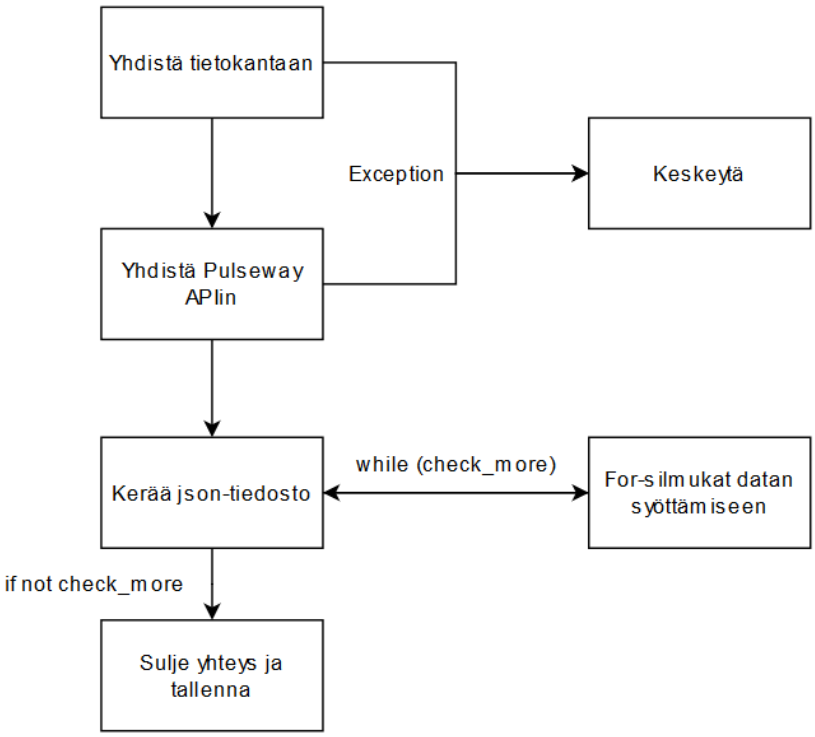
4.1 Yleistä

Tiedon siirtäminen APIlta tietokantaan suoritetaan Python-skriptillä. APIlta tuleva JSON-tiedosto käsitellään otsikko kerrallaan syöttäen tauluun jokaisen tietokoneen avain–arvo-parit. Luvuissa 4.2-4.7 käydään skripti läpi paloittain, ja koko skripti on esitetty liitteessä 1.

Python on hyvä valinta skriptin ohjelmointikieleksi, sillä siihen on saatavilla MariaDB:n oma kirjasto. MariaDB-kirjasto antaa mahdollisuuden suorittaa SQL-kyselyitä Python-skriptistä. Kyselyissä käytetään kysymysmerkkiä ”?” ilmaisemaan muuttujaa. Yhdessä kyselyssä voi myös olla useampia muuttujia. Kysymysmerkkien käyttö on turvallisempaa kuin f-string -syntaksin (Hedgpeth 2020).

Toinen työssä auttava kirjasto on slumber. Slumber-kirjasto käyttäytyy kääreenä requests-kirjaston ympärillä ja auttaa REST APIen käsittelyssä (Giles 2015). Muita käytettäviä kirjastoja on JSON-kirjasto, jota käytetään APIlta tulevan datan kirjoittamiseen JSON-tiedostoon ja sen lukemiseen, lisäksi sys-kirjastoa käytetään koodin ajon keskeyttämiseen virheen sattuessa.

Ohjelman toimintaa kuvaava kaavio on esillä kuviossa 5. Aluksi yhdistetään tietokantaan ja Pulseway APIin. Jos yhteyden muodostaminen ei onnistu, skriptin ajo keskeytetään, sillä molemmat yhteydet ovat kriittisiä ohjelman toiminnalle.
Kun yhteys APIin onnistuu, pyydetään sieltä ensimmäinen JSON-tiedosto. *Check_more*-muuttujan ollessa tosi, ajetaan for-silmukat, jotka lähettävät tiedot tietokantaan. Kun *check_more*-muuttujan arvo on epätosi, suljetaan yhteys ja tallennetaan tiedot tietokantaan.



KUVIO 5. Skriptin vuokaavio

4.2 MariaDB-yhteys

Yhteys MariaDB-kantaan otetaan yhteys-dokumentaation mallin mukaisesti (MariaDB 2021). Muutettu on ainoastaan kirjautumistiedot sekä salasana poistettu tietoturvasyistä. Yhteyden luominen on esitetty kuvassa 6. Myös kursori, jota käytetään SQL-kyselyjen tekemiseen, alustetaan rivillä 19 jo tässä vaiheessa.

```
6   #Connect to database
7   try:
8       conn = mariadb.connect(
9           user="pulseway",
10          password="",
11          host="localhost",
12          database="pulseway"
13      )
14  except mariadb.Error as e:
15      print(f"Error connecting to MariaDB Platform: {e}")
16      sys.exit(1)
17
18  #Initiate cursor
19  cur = conn.cursor()
```

KUVA 6. MariaDB-yhteyden luominen

4.3 Pääsilmukka

Kuvassa 7 on esitetty pääsilmukan alku, silmukan jatkumisen tarkistus, yhteys Pulseway APIin ja tiedoston nimeäminen. Pääsilmukkaa ajetaan *check_more*-muuttujan ollessa tosi. *Check_more*-muuttuja tarkistetaan JSON-tiedostolta *meta*-otsikolta *has_more*-avaimesta riveillä 88–89. Jos avaimen arvo on tosi, silmukan ajo jatkuu normaalisti. *Check_more*-muuttujan arvon ollessa epätosi silmukka ajetaan loppuun, mutta ei enää uudelleen. Ensimmäinen tiedosto nimetään *assets1.json*:ksi ja sitä seuraavat numeroidaan liukuvasti. Näin tiedostot jäävät palvelimelle, jotta myöhemmin voidaan tarkastella skriptin toimivuutta ja datan eheyttä.

```
65  #Pulseway credentials
66  ENDPOINT = "https://jmj.pulseway.com/api/v2/"
67  USERNAME = "joni.pohjanoksa"
68  PASSWORD = ""
69
70  #Variables to count loops and name files
71  filenum = 1
72  check_more = True
73  off = 0
74
75  #1st loop for retrieving json file# from pulseway api
76  while (check_more):
77      try:
78          api = slumber.API(ENDPOINT, auth=(USERNAME, PASSWORD))
79          # use limit & offset parameters
80          str_off=str(off)
81          str_filenum=str(filenum)
82          result = api.assets.get(limit='50', offset=str_off)
83          filename='assets' + str_filenum + '.json'
84          with open(filename, 'w') as json_file:
85              json.dump(result, json_file)
86          filenum+=1
87          off+=50
88          if not result['meta']['has_more']:
89              check_more = False
90
91      except Exception as e:
92          print('GetAssets raised an exception.')
93          print(e)
94          sys.exit(1)
```

KUVA 7. Pääsilmukka ja yhteys Pulseway APIin

4.4 Yhteys Pulseway APIin

Yhteys Pulseway APIin muodostetaan kuvassa 7 näkyvällä rivillä 78, dokumentaation mallin mukaan käyttäen APIn osoitetta ja kirjautumistietoja, joista salasana poistettu tietoturvasyistä (Pulseway n.d.). APIlta tuleva tulos kirjoitetaan listamuuttujaan *result* rivillä 82. *Limit*-muuttuja rajoittaa tuloksen kokoa ja *offset*-muuttujaa käytetään hakemaan seuraavat viisikymmentä tulosta. Ensimmäiseen tiedostoon kirjoitetaan koneiden 0-49 tiedot ja toiseen koneiden 50-99 ja niin edelleen.

4.5 Aliohjelmat

Jokaista tietokannan taulua varten on oma aliohjelma, jota kutsutaan kursorilla sekä avainta vastaavilla arvoilla, jotka luetaan tiedostosta. Arvoja käytetään INSERT IGNORE INTO -SQL-kyselyn tekemiseen. Kysely toteutetaan *cur.execute*-komennolla. Kyselyissä käytetään luvussa 3.4 mainittuja muutettuja avaimia, jotta data kulkeutuu tietokannalle oikeaan paikkaan.

Poikkeuksen edellä mainittuun muodostaa aliohjelma, jolla täytetään assosiatiivinen *installed_software*-taulu. Tämän taulun täyttämiseen tarvittava SQL-lause eroaa muista, sillä sen sarake *software_id* täytetään toisen taulun datalla, eikä suoraan tiedostosta. Jotta *software_id* saataisiin lisättyä oikeaan kohtaan, lähetetään aliohjelmalle arvot, jotka viittaavat omaan *software_id*:hen. *Software_id*:n uniikit arvot ovat nimi ja versio, joten aliohjelmalle lähetetään kursori*, asset_id, software_name* ja *software_version*. Luvussa 4 mainittuihin ?-merkkeihin tieto syötetään *cur.execute*-komennon lopussa. Aliohjelmat *add_disks* ja *add_installed_software* ovat esillä kuvissa 8 ja 9. Huomioitavaa kuvissa on, että raportin kuvia varten Python-skripteihin on lisätty ylimääräinen rivinvaihto.

```
def add_disks(cur, asset_id, disk_name, system_disk, free_percentage, total_value):
    cur.execute("INSERT IGNORE INTO"
    "pulseway.disks(asset_id, disk_name, system, free_percentage, total_available) VALUES(?,?,?,?,?)",
    (asset_id, disk_name, system_disk, free_percentage, total_value))
```

KUVA 8. Aliohjelma *add_disks*

```
#pulseway.installed_software takes the primay keys from pulseway.assets and pulseway.software
#to ensure correct software_id is picked uniques software_name and software_version are used
def add_installed_software(cur,asset_id,software_name,software_version):
    cur.execute("INSERT INTO pulseway.installed_software(asset_id,software_id)"
    "SELECT ?,software.software_id FROM pulseway.software WHERE software_name=? AND software_version=?",
    (asset_id,software_name,software_version))
```

KUVA 9. Aliohjelma *add_installed_software*

4.6 Aliohjelmakutsut

Luvussa 4.4 esiteltyjä aliohjelmia kutsutaan pääsilmukasta. Pääsilmukan rakenteen takia yhtä aliohjelmaa kutsutaan for-silmukasta 50 kertaan, ja sen jälkeen siirrytään seuraavan. Kun kaikkia aliohjelmia on kutsuttu, palataan pääsilmukan alkuun ja luodaan uusi tiedosto APIlta tulevasta datasta. Kuvassa 10 näkyy tiedoston avaaminen ja kaksi eri silmukkaa aliohjelmakutsuille.

```
96      #open file# for reading
97      with open(filename) as json_file:
98          data=json.load(json_file)
99
100         #loops to call functions
101         #descriptive variables for less clutter in function calls
102         #try-except for exception handling
103         #in case of NULL value fill table with "-"
104         #nested loops to iterate json structure
105
106         for i in data['data']:
107             add_asset(cur,i['identifier'],i['name'],i['group'],
108             i['description'],i['type'],i['client_version'],
109             i['last_updated'],i['last_seen_online'],i['last_reboot'],
110             i['external_url'],i['cpu_usage'],i['memory_usage'],
111             i['memory_total'],i['firewall_enabled'],i['antivirus_enabled'],
112             i['antivirus_up_to_date'],i['uac_enabled'],i['public_ip_address'])
113
114         for i in data['data']:
115             for j in i['asset_info']:
116                 cd=j['category_data']
117                 if j['category_name']=="System":
118                     try:
119                         add_asset_info_system(cur,
120                         i['identifier'],cd['Name'],cd['Manufacturer'],
121                         cd['Model'],cd['Type'],cd['CPU'],cd['Number of Processors'],
122                         cd['Number of Logical Processors'],cd['DNS Host Name'],
123                         cd['Domain'],cd['Owner Name'],cd['Roles'],cd['Status'])
124                     except Exception as e:
125                         cd[e.args[0]]="-"
126                         add_asset_info_system(cur,
127                         i['identifier'],cd['Name'],cd['Manufacturer'],
128                         cd['Model'],cd['Type'],cd['CPU'],cd['Number of Processors'],
129                         cd['Number of Logical Processors'],cd['DNS Host Name'],
130                         cd['Domain'],cd['Owner Name'],cd['Roles'],cd['Status'])
```

KUVA 10. Esimerkkejä aliohjelmakutsuista

Ennen silmukoiden ajoa luetaan JSON-tiedosto listamuuttujaan *data*. For-silmukat indeksoivat *data*-muuttujalta järjestyksessä kaikki avaimet ja kutsuvat aliohjelmia niiden arvoilla. Tapauksissa, joissa listan indeksissä on toinen lista, tarvitaan toinen for-silmukka indeksoimaan uutta listaa aiemman sisällä.

Ensimmäinen esimerkkisilmukka kutsuu aliohjelmaa *add_asset*. Se lisää tietokoneet *pulseway.assets*-tauluun, kuvassa 10 rivit 106–112. Kaikissa aliohjelmakutsuissa on tärkeää huomioida luvussa 3.4 eritellyt alkuperäiset ja muutetut avaimet. Aliohjelmaa kutsuessa on käytettävä alkuperäisiä avaimia sillä JSON-tiedostossa olevia avaimia ei ole muokattu.

Toinen esimerkkisilmukka on kuvassa 10 riveillä 114–130. Tässä esimerkissä tarvitaan toinen silmukka ensimmäisen sisälle. Kun indeksoidaan näin syvältä tiedoston rakenteesta, on syytä käyttää lyhennettä *cd*. Tämä eliminoi toiston aliohjelmakutsussa, koska tällöin ei tarvitse kirjoittaa koko indeksiä vaan voi kirjoittaa ”cd[’Name’]”, eikä ”j[’category_data][’Name’]”. Näin aliohjelmakutsu on siistimpi.

Toinen ongelma on samannimiset avaimet *category_data*-otsikon alla. Otsikon alla on kolme eri kategoriaa: ”System”, ”BIOS” ja ”Operating System”. Näillä kaikilla on otsikoilla avain *Name*, eikä sitä voi erottaa kuin *category_name*-avaimen arvolla. Kuvassa 10 rivillä 117 oleva if-lause varmistaa, ettei väärää nimeä kirjoiteta tietokantaan väärään kohtaan.

4.7 Virheenkäsittely

Skriptissä virheenkäsittely suoritetaan matalalla tasolla. Työn tilannut yritys ei asettanut erityisiä vaatimuksia virheenkäsittelylle, joten skriptiin tehty virheenkäsittely on lisätty varmistamaan testauksen onnistuminen. Testidatana käytettiin aitoa APIlta tulevaa dataa. Työntekohetkellä yksi taso virheenkäsittelyä oli riittävä.

Testatessa skriptiä törmättiin uuteen ongelmaan: mitä käy, jos avain puuttuu datasta? Tätä varten tarvittiin virheenkäsittelyrakenne. Puuttuvan avaimen arvoksi kirjoitetaan tylysti viiva ”-”. Puuttuva avain saadaan Pythonin virheistä. Pythonin try–except-rakenteen except-osassa voidaan virhe tallentaa muuttujaan. Tämä muuttuja saa minkä tahansa Pythonin sisäänrakennetuista virheistä (Built-in Exceptions n.d.). Vaikka tiedetään odottaa avaimeen liittyvää virhettä ”KeyError”, suoritetaan sama toiminto joka tapauksessa, virheestä riippumatta. Virhemuuttujalla on lista argumenteista, joihin tallentuu erilaista tietoa. Avainvirheen tapauksessa argumenttilistan ensimmäinen indeksi sisältää puuttuvan avaimen.

Esimerkiksi *Domain*-avaimen puuttuessa palautuu virhe ”KeyError(’Domain’)”. Tämä virhe kirjoitetaan muuttujaan *e*. Kyseisen muuttujan argumentin ensimmäinen indeksi on *Domain*. Kun tiedetään minkä avaimen arvo puuttuu, voidaan manuaalisesti antaa sille arvoksi ”-”. Kuvassa 11 on esitetty tämä virheenkäsittelyrakenne.

```
118                 try:
119                     add_asset_info_system(cur,
120                     i['identifier'],cd['Name'],cd['Manufacturer'],
121                     cd['Model'],cd['Type'],cd['CPU'],cd['Number of Processors'],
122                     cd['Number of Logical Processors'],cd['DNS Host Name'],
123                     cd['Domain'],cd['Owner Name'],cd['Roles'],cd['Status'])
124                 except Exception as e:
125                     cd[e.args[0]]="-"
126                     add_asset_info_system(cur,
127                     i['identifier'],cd['Name'],cd['Manufacturer'],
128                     cd['Model'],cd['Type'],cd['CPU'],cd['Number of Processors'],
129                     cd['Number of Logical Processors'],cd['DNS Host Name'],
130                     cd['Domain'],cd['Owner Name'],cd['Roles'],cd['Status'])
```

KUVA 11: Virheenkäsittelyn rakenne

5 POHDINTA

Opinnäytetyön tuloksena syntyi toimiva kokonaisuus. Vaikka parannettavaa on, täyttää ratkaisu työn tilaajan vaatimukset. Suurimman päänvaivan opinnäytetyössä tuotti Python-skripti, sillä Pythonista työn tekijällä oli vain aivan perusteet hallussa. Tämä oli kuitenkin oiva mahdollisuus oppia kyseistä ohjelmointikieltä ja soveltaa jopa hieman edistyneempiä menetelmiä, kuten argumenttien indeksointi virheenkäsittelyssä. Raporttien luominen vaatii toimiakseen tietokannan tyhjentämisen aina ennen uuden raportin luomista.

Tietokanta on toimiva ja määrittelyn mukainen eli kaikelle datalle on järkevä paikka ja datan eheys ei häviä. Tietokantaan kirjoitetaan aina jotain ja NULL-arvot korvataan. Tietokantaa voisi parantaa pohtimalla tyyppien varaaman tilan uudelleen, jos tietokanta paisuu liian suureksi. Myös joidenkin automaattisesti kasvavien tunnisteiden tarvetta voi olla syytä pohtia.

Skriptin teko onnistui myös kohtalaisen hyvin, joskin skriptissä on paranneltavaa. Ensimmäinen parannuskohde olisi SQL-kyselyt sillä INSERT IGNORE on kankea ratkaisu. Tällä skriptillä lisätty avaimen arvo jää tietokantaan, kunnes se poistetaan. Skriptiä uudelleen ajaessa poistamatta tietokannan dataa etukäteen avainten arvot eivät päivity tietokannassa. Jatkossa olisi suotavaa käyttää ON DUPLICATE KEY UPDATE -kyselyä. Vaikka tämän toteuttaminen olisikin työläämpää, se mahdollistaisi avainten arvojen päivittämisen poistamatta arvoja. Tällöin data olisi reaaliaikaisempaa, eikä tietokantaa tarvitsisi rasittaa jatkuvalla poistamisella ja uudelleenkirjoittamisella.

Toinen parannuskohde on virheenkäsittely. Testauksen aikana ei löytynyt kuin yksi esimerkki, jolloin olisi tarvittu virhetilan ajoa. Tällöin APIlta tulleesta datasta puuttui vain yksi arvo, jonka nykyinen menetelmä ratkaisi. Tilanteessa, jossa datasta puuttuu useampi arvo, nykyinen skripti ei huomaa eroa.

Yllä mainitut kehityskohteet huomioiden opinnäytetyön tulos on hyvä, toimiva paketti, jonka osana tietokanta ja skripti molemmat toimivat hyvin. Sekä työn tilannut yritys että työn tekijä voivat olla tyytyväisiä valmiiseen lopputulokseen.

LÄHTEET

About MariaDB. 2021. MariaDB.org. Luettu 13.5.2021. https://mariadb.org/about/

Built-in Exceptions. n.d. Python.org. Luettu 13.5.2021. https://docs.python.org/3/library/exceptions.html

Csiki, P. 2020. Getting started with Automation Workflows for RMM. Luettu 28.4.2021. https://intercom.help/pulseway/en/articles/3766087-getting-started-with-automation-workflows-for-rmm

Ferril, P., Rash, W. 2019. MMSoft Pulseway Review. Luettu 28.4.2021. https://www.pcmag.com/reviews/mmsoft-pulseway

Giles, S. 2015. Slumber. Luettu 26.4.2021. https://pypi.org/project/slumber/

Hedgpeth, R. 2020. How to connect Python programs to MariaDB. Luettu 26.4.2021. https://mariadb.com/resources/blog/how-to-connect-python-programs-to-mariadb/

Hovi, A., Huotari, J. & Lahdenmäki, T. 2005. Tietokantojen suunnittelu & indeksointi. Jyväskylä: Docendo Finland Oy

ISO/IEC 9075-2:2016, 6. 2016. Information technology — Database languages — SQL — Part 2: Foundation (SQL/ Foundation). Luettu 13.5.2021. https://www.iso.org/standard/63556.html

json.org. n.d. Introducing JSON. Luettu 24.5.2021. https://www.json.org/json-en.html

MariaDB. 2021. MariaDB Connector/Python. Luettu 26.4.2021. https://mariadb.com/docs/clients/connector-python/#connector-python

Moore, J., Smith, S. 2020. RMM software (remote monitoring and management software). Luettu 13.5.2021. https://searchitchannel.techtarget.com/definition/RMM-software-remote-monitoring-and-management-software

NinjaRMM. 2020. How Much Does RMM Software Cost?. Luettu 13.5.2021. https://www.ninjarmm.com/blog/how-much-does-rmm-software-cost/

Picket, P. 2020. What Is SQL?. Luettu 13.5.2021. https://www.thebalancecareers.com/what-is-sql-and-uses-2071909

Pulseway. 2020. Pulseway Introduces Brand New Automation Workflows to Auto-remediate Issues on User's Behalf. Luettu 28.4.2021. https://www.pulseway.com/blog/pulseway-introduces-brand-new-automation-workflows-to-simplify-the-worklife-of-its-customers-and-auto-remediate-issues-on-users-behalf

Pulseway. 2021. Pulseway.com. Luettu 24.5.2021 https://www.pulseway.com/

Pulseway. n.d. Api Documentation. Luettu 26.4.2021. https://api.pulseway.com/?python#assets

Vainio, L. 2019. Yrittäjä: tunnetko nämä tietoturvan termit?. Luettu 13.5.2021. https://www.telia.fi/yrityksille/artikkelit/artikkeli/tietoturva-sanasto-yrittajille

LIITTEET

Liite 1. Python-skripti

```
import mariadb
import slumber
import json
import sys

#Connect to database
try:
    conn = mariadb.connect(
        user="pulseway",
        password="",
        host="localhost",
        database="pulseway"
    )
except mariadb.Error as e:
    print(f"Error connecting to MariaDB Platform: {e}")
    sys.exit(1)

#Initiate cursor
cur = conn.cursor()

#define all necessary functions
#add functions add values into their respective pulseway data-
bases from assets.1json
def add_asset(cur,asset_id, name, groups, description, sys_type, cli-
ent_version, last_updated, last_seen_online, last_reboot, exter-
nal_url, cpu_usage, memory_usage, memory_total, firewall_enabled, antivi-
rus_enabled, antivirus_updated, uac_enabled, public_ip_add):
    cur.execute("INSERT IGNORE INTO pulseway.assets(as-
set_id, name, groups, description, sys_type, client_version, last_up-
dated, last_seen_online, last_reboot, external_url, cpu_usage, memory_us-
age, memory_total, firewall_enabled, antivirus_enabled, antivirus_up-
dated, uac_enabled, public_ip_add) VAL-
UES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
    (asset_id, name, groups, description, sys_type, client_ver-
sion, last_updated, last_seen_online, last_reboot, external_url, cpu_us-
age, memory_usage, memory_total, firewall_enabled, antivirus_enabled, an-
tivirus_updated, uac_enabled, public_ip_add))

def add_asset_info_system(cur, asset_id, sys_name, sys_manufac-
turer, model, sys_type, cpu, number_of_processors, number_of_logical_pro-
cessors, dns_hostname, domain, owner, roles, status):
    cur.execute("INSERT IGNORE INTO pulseway.asset_info_system(as-
set_id, sys_name, sys_manufacturer, model, sys_type, cpu, number_of_pro-
cessors, number_of_logical_processors, dns_hostname, do-
main, owner, roles, status) VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?)",
```

```
    (asset_id, sys_name, sys_manufacturer, model, sys_type, cpu, num-
ber_of_processors, number_of_logical_processors, dns_hostname, do-
main, owner, roles, status))

def add_asset_info_bios(cur, asset_id, bios_serial_num-
ber, bios_name, bios_manufacturer, smbios_ver, smbios_ma-
jor_ver, smbios_minor_ver, bios_version, release_date, bios_sta-
tus, bios_description):
    cur.execute("INSERT IGNORE INTO pulseway.asset_info_bios(as-
set_id, bios_serial_number, bios_name, bios_manufac-
turer, smbios_ver, smbios_major_ver, smbios_minor_ver, bios_version, re-
lease_date, bios_status, bios_description) VAL-
UES(?,?,?,?,?,?,?,?,?,?,?)",
    (asset_id, bios_serial_number, bios_name, bios_manufactu-
rer, smbios_ver, smbios_major_ver, smbios_minor_ver, bios_version, re-
lease_date, bios_status, bios_description))

def add_asset_info_os(cur, asset_id, os_name, os_ver-
sion, build_type, registered_user, product_key, os_serial_number, sp_ma-
jor_ver, sp_minor_ver, system_device, system_directory, windows_direc-
tory, install_date, lo-
cal_date_time, last_boot_up_time, last_logged_on_user):
    cur.execute("INSERT IGNORE INTO pulseway.asset_info_os(as-
set_id, os_name, os_version, build_type, registered_user, prod-
uct_key, os_serial_number, sp_major_ver, sp_minor_ver, system_de-
vice, system_directory, windows_directory, install_date, lo-
cal_date_time, last_boot_up_time, last_logged_on_user) VAL-
UES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)",
    (asset_id, os_name, os_version, build_type, registered_user, prod-
uct_key, os_serial_number, sp_major_ver, sp_minor_ver, system_de-
vice, system_directory, windows_directory, install_date, lo-
cal_date_time, last_boot_up_time, last_logged_on_user))

def add_disks(cur, asset_id, disk_name, system_disk, free_percentage, to-
tal_value):
    cur.execute("INSERT IGNORE INTO pulseway.disks(as-
set_id, disk_name, system, free_percentage, total_available) VAL-
UES(?,?,?,?,?)",
    (asset_id, disk_name, system_disk, free_percentage, total_value))

def add_event_logs(cur, asset_id, error, warning, information):
    cur.execute("INSERT IGNORE INTO pulseway.event_logs(asset_id, er-
ror, warning, information) VALUES (?,?,?,?)",
    (asset_id, error, warning, information))

def add_updates(cur, asset_id, critical, important, moderate, low, un-
specified):
    cur.execute("INSERT IGNORE INTO pulseway.updates(asset_id, criti-
cal, important, moderate, low, unspecified) VALUES (?,?,?,?,?,?)",
    (asset_id, critical, important, moderate, low, unspecified))
```

```
def add_tags(cur, asset_id, tags):
    cur.execute("INSERT INTO pulseway.tags(asset_id, tags) VALUES (?,?)",
    (asset_id,tags))

def add_software(cur,software_name,publisher,software_version):
    cur.execute("INSERT INTO pulseway.software(software_name,pub-
lisher,software_version) VALUES (?,?,?)",
    (software_name,publisher,software_version))

#pulseway.installed_software takes the primay keys from pulseway.as-
sets and pulseway.software
#to ensure correct software_id is picked uniques software_name and soft-
ware_version are used
def add_installed_software(cur,asset_id,software_name,software_version):
    cur.execute("INSERT INTO pulseway.installed_software(asset_id,soft-
ware_id) SELECT ?,software.software_id FROM pulseway.software WHERE soft-
ware_name=? AND software_version=?",
    (asset_id,software_name,software_version))

#Pulseway credentials
ENDPOINT = "https://jmj.pulseway.com/api/v2/"
USERNAME = "joni.pohjanoksa"
PASSWORD = ""

#Variables to count loops and name files
filenum = 1
check_more = True
off = 0

#1st loop for retrieving json file# from pulseway api
while (check_more):
    try:
        api = slumber.API(ENDPOINT, auth=(USERNAME, PASSWORD))
        # use limit & offset parameters
        str_off=str(off)
        str_filenum=str(filenum)
        result = api.assets.get(limit='50', offset=str_off)
        filename='assets' + str_filenum + '.json'
        with open(filename, 'w') as json_file:
            json.dump(result, json_file)
        filenum+=1
        off+=50
        if not result['meta']['has_more']:
            check_more = False

    except Exception as e:
        print('GetAssets raised an exception.')
        print(e)
        sys.exit(1)

    #open file# for reading
```

```
    with open(filename) as json_file:
        data=json.load(json_file)

        #loops to call functions
        #descriptive variables for less clutter in function calls
        #try-except for exception handling
        #in case of NULL value fill table with "-"
        #nested loops to iterate json structure

        for i in data['data']:
            add_asset(cur,i['identifier'],i['name'],i['group'],
            i['description'],i['type'],i['client_version'],
            i['last_updated'],i['last_seen_online'],i['last_reboot'],
            i['external_url'],i['cpu_usage'],i['memory_usage'],
            i['memory_total'],i['firewall_enabled'],i['antivirus_ena-
bled'],
            i['antivirus_up_to_date'],i['uac_enabled'],i['public_ip_ad-
dress'])

        for i in data['data']:
            for j in i['asset_info']:
                cd=j['category_data']
                if j['category_name']=="System":
                    try:
                        add_asset_info_system(cur,
                        i['identifier'],cd['Name'],cd['Manufacturer'],
                        cd['Model'],cd['Type'],cd['CPU'],cd['Num-
ber of Processors'],
                        cd['Number of Logical Proces-
sors'],cd['DNS Host Name'],
                        cd['Do-
main'],cd['Owner Name'],cd['Roles'],cd['Status'])
                    except Exception as e:
                        cd[e.args[0]]="-"
                        add_asset_info_system(cur,
                        i['identifier'],cd['Name'],cd['Manufacturer'],
                        cd['Model'],cd['Type'],cd['CPU'],cd['Num-
ber of Processors'],
                        cd['Number of Logical Proces-
sors'],cd['DNS Host Name'],
                        cd['Do-
main'],cd['Owner Name'],cd['Roles'],cd['Status'])

        for i in data['data']:
            for j in i['asset_info']:
                cd=j['category_data']
                if j['category_name']=="BIOS":
                    try:
                        add_asset_info_bios(cur,
                        i['identifier'],cd['Serial Num-
ber'],cd['Name'],cd['Manufacturer'],
```

```
                        cd['SMBIOS Version'],cd['SMBIOS Major Ver-
sion'],cd['SMBIOS Minor Version'],
                        cd['Version'],cd['Release Date'],cd['Sta-
tus'],cd['Description'])
                    except Exception as e:
                        cd[e.args[0]]="-"
                        add_asset_info_bios(cur,
                        i['identifier'],cd['Serial Num-
ber'],cd['Name'],cd['Manufacturer'],
                        cd['SMBIOS Version'],cd['SMBIOS Major Ver-
sion'],cd['SMBIOS Minor Version'],
                        cd['Version'],cd['Release Date'],cd['Sta-
tus'],cd['Description'])

        for i in data['data']:
            for j in i['asset_info']:
                cd=j['category_data']
                if j['category_name']=="Operating System":
                    try:
                        add_asset_info_os(cur,i['identifi-
er'],cd['Name'],cd['Version'],
                        cd['Build Type'],cd['Registered User'],cd['Prod-
uct Key'],cd['Serial Number'],
                        cd['Service Pack Major Version'],cd['Ser-
vice Pack Minor Version'],
                        cd['System Device'],cd['System Directo-
ry'],cd['Windows Directory'],
                        cd['Install Date'],cd['Lo-
cal Date and Time'],cd['Last Boot Up Time'],
                        cd['Last Logged On User'])
                    except Exception as e:
                        cd[e.args[0]]="-"
                        add_asset_info_os(cur,i['identifi-
er'],cd['Name'],cd['Version'],
                        cd['Build Type'],cd['Registered User'],cd['Prod-
uct Key'],cd['Serial Number'],
                        cd['Service Pack Major Version'],cd['Ser-
vice Pack Minor Version'],
                        cd['System Device'],cd['System Directo-
ry'],cd['Windows Directory'],
                        cd['Install Date'],cd['Lo-
cal Date and Time'],cd['Last Boot Up Time'],
                        cd['Last Logged On User'])

        for i in data['data']:
            for j in i['disks']:
                try:
                    add_disks(cur,i['identifier'],j['name'],j['system'],
                    j['free_percentage'],j['total_value'])
                except Exception as e:
                    j[e.args[0]]="-"
```

```
                    add_disks(cur,i['identifier'],j['name'],j['system'],
                    j['free_percentage'],j['total_value'])

        for i in data['data']:
            logs=i['event_logs']
            try:
                add_event_logs(cur, i['identifier'],logs['error'],
                logs['warning'],logs['information'])
            except:
                add_event_logs(cur, i['identifier'], "-", "-", "-")

        for i in data['data']:
            updates=i['updates']
            try:
                add_updates(cur,i['identifier'],updates['critical'],up-
dates['important'],
                updates['moderate'],updates['low'],updates['unspeci-
fied'])
            except:
                add_updates(cur,i['identifier'],"-","-","-","-","-")

        #only assets with tags as-
signes to them will be added to tags.pulseway
        for i in data['data']:
            try:
                add_tags(cur,i['identifier'],i['tags'][0])
            except:
                continue

        for i in data['data']:
            for j in i['installed_software']:
                try:
                    add_software(cur,j['name'],j['publisher'],j['ver-
sion'])
                except:
                    continue

        for i in data['data']:
            for j in i['installed_software']:
                try:
                    add_installed_software(cur,i['identifi-
er'],j['name'],j['version'])
                except:
                    print('Error in add_installed_software function')
                    continue

conn.commit()
conn.close()
print('Done')
```