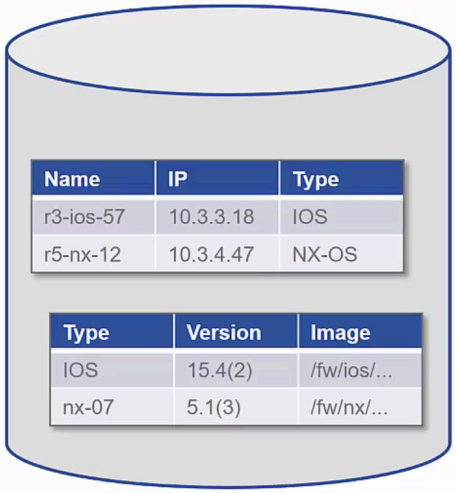
Database Overview

Reasons for using a database

- **Volume:** Large volumes of data that would exceed capacity for normal files
- **Tables:** Multiple tables of information, e.g. devices, users, locations, policies
- **Search:** Searching for a specific item of data
- **Filter:** Retrieving all data that meet a certain criteria



-5:27 1.5x

Database Communication

DB Connection

- First establish connection to the database

DB Cursor

- The database 'cursor' is used to point to a row in a table in the DB

SQL Commands

- SQL commands will be issued to create, read, update, and delete (CRUD) operations on the database.

sqlite3

- Popular and simple Python library for SQL database access

Create Devices Table in DB

Connect to DB and get cursor: Connect to DB and get cursor.

```
db_connection = sqlite3.connect(db_file)
db_cursor = db_connection.cursor()
```

Create table: Execute SQL command to create the table if it doesn't already exist, passing in the fields in the table, including primary key.

```
db_cursor.execute('''CREATE TABLE IF NOT EXISTS devices
                            (name VARCHAR(16) PRIMARY KEY,
                             ip   VARCHAR(16),
                             os   VARCHAR(8),
                             user VARCHAR(16),
                             pw   VARCHAR(16))''')
```

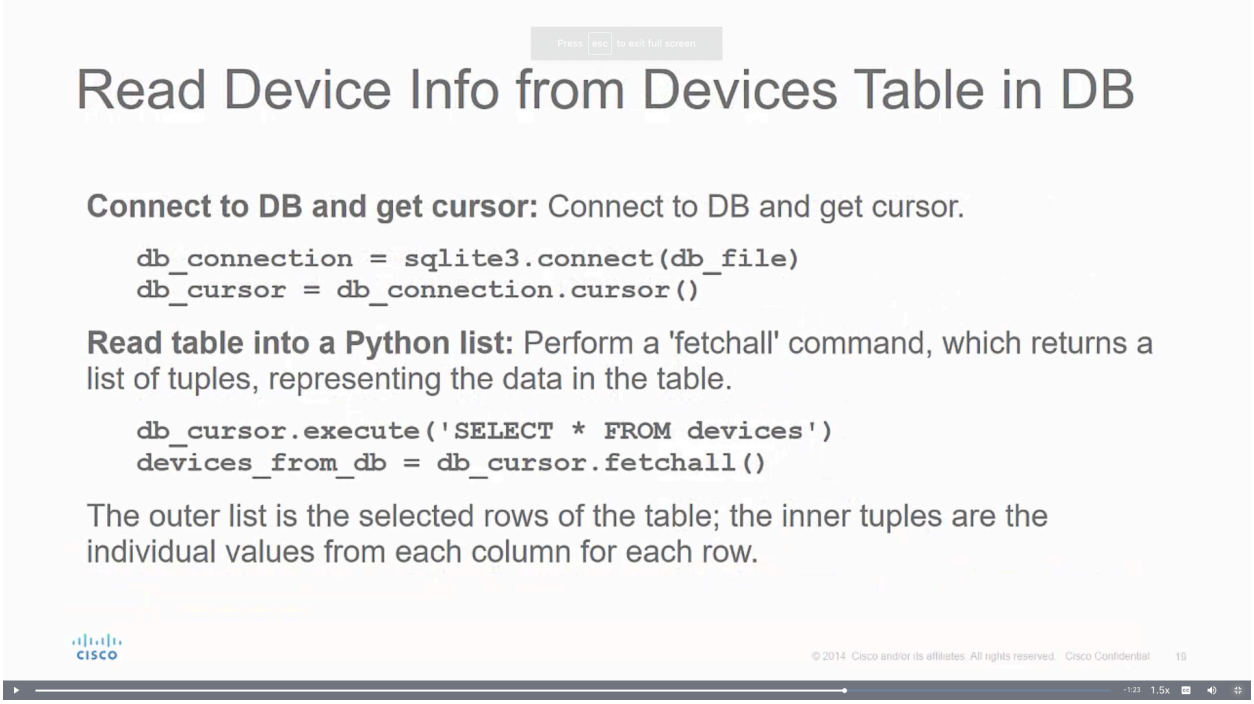

Write Device Info to Devices Table in DB

```
for device in devices_list:  # iterate through all devices

   # Create SQL command for writing a row of data
   sql_command = '''REPLACE INTO devices (name,ip,os,user,pw)
                              VALUES(?,?,?,?,?)'''

   # Execute SQL command, substituting device info for '?'s
   db_cursor.execute(sql_command, (device.name,
                                   device.ip_address,
                                   device.os_type,
                                   device.username,
                                   device.password))

db_connection.commit()  # must commit changes to DB when done
```



There are several aspects of database communication which must be understood in order to make use of the functionality afforded by this type of storage. In particular:

- **Connection**: Before your application can communicate with the database, you must establish a connection to the DB. For sqlite3, the connection is established using the 'connect()' function.
- **Cursor**: Once a connection has been established, your communication with the database will be using the *cursor*, which can be thought of as a pointer to a row in a database table. The cursor tells the SQL library the table and row upon which an operation is to be performed.
- **SQL Commands**: Your application will interact with the database using Structured Query Language (SQL). SQL commands operate on tables and rows in the database, and some common commands allow you to create tables (CREATE), add or replace rows (REPLACE), and search for specific items (SEARCH).
- **sqlite3**: SQLite is a lightweight library for accessing SQL databases, and is well-suited to modest-sized applications and for prototyping solutions. SQLite provides simple functions for connecting to the database, establishing the cursor, and executing SQL commands.

Creating Tables

One of the basic tasks when dealing with databases is the creation of a table. The process begins with establishing the connection and getting the cursor for the database.

```
    # Connect to the database and get the cursor
    db_connection = sqlite3.connect(devices_db_file) # DB connection
    db_cursor = db_connection.cursor()               # DB cursor
```

The next step is to issue the SQL command to create a table. The SQL "CREATE TABLE" command is used for this purpose, providing the name of the table, and the columns representing the data items of the table.

In the following example, the code is executing an SQL command to create a table called "devices," creating columns (fields) for device name, IP address, OS-type, username, and password.

```
    db_cursor.execute('''CREATE TABLE IF NOT EXISTS devices
                              (name VARCHAR(16) PRIMARY KEY,
                               ip   VARCHAR(16),
                               os   VARCHAR(8),
                               user VARCHAR(16),
                               pw   VARCHAR(16))''')
```

In the SQL command above:

- `db_cursor.execute()`: The SQLite function that is called to execute an SQL command.
- **SQL command**: "CREATE TABLE IF NOT EXISTS devices" will create a table in the database by the given name ('devices'), unless it exists.
- **name, ip, os, user, pw:** All the fields for the table, the specific items of information about each device that will be stored.
- **VARCHAR:** The 'type' of data that will be stored – 'VARCHAR' is the SQL name for a character string of variable length. The value in parentheses, for example (16), specifies the maximum length.
- **PRIMARY KEY:** Specifies the key for the table. Primary keys must be unique within the table and may be used to improve efficiency and insure deterministic behavior of the database table.

Writing Device Information to a Table

Writing data to an existing database table can be done using the SQL REPLACE command, which will either add or replace the item based on the prior existence of the given primary key value.

In Python the procedure is to use SQLite and your cursor object to execute an SQL command to write the desired data into the database table. The first step is to connect to the database:

```
    # Connect to the database and get a connection to the database
    db_connection = sqlite3.connect(devices_db_file) # DB connection
    db_cursor = db_connection.cursor()               # DB cursor
```

Next, create the SQL command.

```
        sql_cmd = 'REPLACE INTO devices (name, ip, os, user, pw) VALUES(?,?,?,?,?)
        db_cursor.execute(sql_cmd, (device.name,
                                    device.ip_address,
                                    device.os_type,
                                    device.username,
                                    device.password))
```

In the code above:

- **SQL Command**: "REPLACE INTO devices" tells SQLite that the application is intending to add or replace a row in the devices table.
- **(name,ip,os,user,pw):** The fields into which the data will be placed.
- **VALUES(?,?,?,?,?):** Placeholders for the values that will be provided in the SQLite execute command, making it easy to write applications that pass in the appropriate data to be filled into the SQL command when it is actually executed by the database.
- **`db_cursor.execute()`:** The last step is to pass this command in to SQLite, telling it to execute the command, and passing in the variables that will be substituted for the question marks (?,?,?,?,?) in the actual command.

Reading Device Information from a Table

Gathering information from a database is a matter of performing a 'SELECT' operation and having the result delivered into your Python data structures.

First, connect to the database:

```
    # Connect to the database and get a connection to the database
    db_connection = sqlite3.connect(devices_db_file) # DB connection
    db_cursor = db_connection.cursor()               # DB cursor
```

Next, execute the SELECT command, specifying to get all rows ('*') from the table 'devices':

```
    db_cursor.execute('SELECT * FROM devices')
    devices_from_db = db_cursor.fetchall()
```

The resulting Python object will be a list, of tuples:

- The outer list is a list of the rows of the table – in this case, one item for each device.
- The inner tuple is a sequence of the device information, one value for each field in the table.

The resulting data would look similar to:

```
[('ios-01', '10.30.30.1', 'ios', 'cisco', 'cisco'),
 ('ios-02', '10.30.30.2', 'ios', 'cisco', 'cisco'),
 ('ios-03', '10.30.30.3', 'ios', 'cisco', 'cisco')]
```