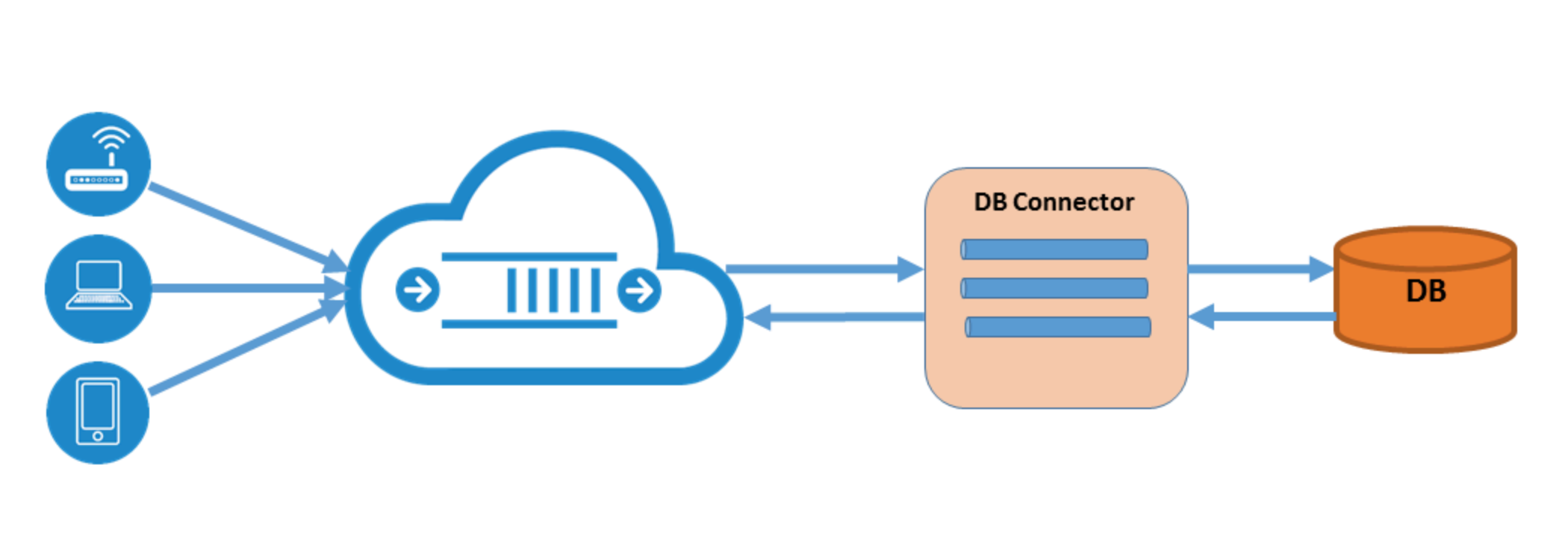
SQL in Python

```
[!] # Run SELECT query in Python?
    SELECT * FROM students

[!]   Cell In[1], line 1
        SELECT * FROM students
                             ^
    SyntaxError: invalid syntax
```

DB connector to interface between Python and SQL DB



`sqlite3`: Connector for SQLite DB

A Python package that provides a SQL interface to the **SQLite** database engine

```python
import sqlite3

# Connect to a database
conn = sqlite3.connect('harry-potter.db')

# Use the connection to execute a query by passing a SQL string
# and get the query result
result = conn.execute("SELECT * FROM students").fetchall()

print(result)
```

---

https://docs.python.org/3/library/sqlite3.html

SQLite database

Lightweight disk-based database without a separate server process

Pros:

No installation, no configuration, no maintenance (Great for prototyping and learning SQL)

Cons:

Limited capabilities compared to full-scale database systems (e.g., MySQL, PostgreSQL)

- access control
- data storage and retrieval
- data backup and recovery
- data dictionary management
- transaction processing (concurrency and recovery)

Working with databases in Python

`sqlite3` or any database connector that follows DB-API 2.0 provides functions to:

1. Connect to a database (`connect()`)
2. Execute a query (`execute()`)
3. Get query results (`fetchone()`, `fetchmany(n)`, `fetchall()`)
4. Save (`commit()`) and close connection (`close()`)

Connect to a database

`connect()`: Connect to a database

- Create a connection **object** that enables access to a database
- `connect('name.db')`: load database file `name.db` or create a new one if it doesn't exist

```
# Import the sqlite3 package
import sqlite3

# Connect to a database called harry-potter.db in the current directory
# If it doesn't exist, it will create a new one.
conn = sqlite3.connect('harry-potter.db')
```

Execute a query (DQL)

`execute()` : Execute a query

- Takes a SELECT query as an argument
- Returns a cursor **object** that represents the query result

```
conn.execute("SELECT * FROM students")
```

Get query results from cursor object

Cursor object methods to get query results:

- `fetchone()`: fetch the next row of a query result set, returning a single tuple, or `None` when no more data is available
- `fetchmany(n)`: fetch the next n rows of a query result, returning a list of tuples, or an empty list when no more data is available
- `fetchall()`: fetch all (remaining) rows of a query result, returning a list of tuples

Get query results from cursor object

```
one_record = conn.execute("SELECT * FROM students").fetchone()

five_records = conn.execute("SELECT * FROM students").fetchmany(5)

all_records = conn.execute("SELECT * FROM students").fetchall()
```

Query result type: `tuple`

`list` : mutable

`tuple` : like list, but immutable

```
results = conn.execute("select * from students").fetchall()
first_row = results[0]

print(type(first_row))              # <class 'tuple'>
first_row[0] = 200                  # TypeError: 'tuple' doesn't support assignment

first_row_list = list(first_row)
print(type(first_row_list))         # <class 'list'>
first_row_list[0] = 200             # works fine
```

Quote inside string values

```
# Error
conn.execute("SELECT * FROM students WHERE first_name = "Harry"")
```

```
# Solutions:
# Double quotes for outer string
conn.execute("SELECT * FROM students WHERE first_name = 'Harry'")

# Single quotes for outer string
conn.execute('SELECT * FROM students WHERE first_name = "Harry"')

# Escape quotes with escape character (\)
conn.execute("SELECT * FROM students WHERE first_name = \"Harry\"")
```

Long queries: use triple quotes (`"""`)

Multi-line string in Python: use triple quotes (`"""` or `'''`)

```
query = """
    SELECT *
    FROM students
    WHERE birthyear > 1980
    ORDER BY birthyear DESC
"""

print(conn.execute(query).fetchall())
```

💻 Query `harry-potter.db` with SQL in Pandas

- **Connect to `harry-potter.db` using `sqlite3`**
- **Execute queries to answer the following questions:**
  - List the name of students who are born after 1980
  - What is the name of the oldest student?
- **Print query results**

Execute a query (DDL & DML)

`execute()` : Execute a query

```
query = """
    CREATE TABLE students2 (
        id INTEGER PRIMARY KEY,
        name TEXT,
        house TEXT,
        age INTEGER
    )
"""
conn.execute(query)
```

```
query = """
    INSERT INTO students2 (id, name, house, age)
    VALUES (1, 'Harry Potter', 'Gryffindor', 11)
"""
conn.execute(query)
```

Insert data dynamically (single record)

Option 1. Insert tuple using f-string

```
record = (1, 'Harry Potter', 'Gryffindor', 11)
conn.execute(f"INSERT INTO students2 VALUES {record}")
```

Option 2. Insert tuple using argument

```
record = (1, 'Harry Potter', 'Gryffindor', 11)
conn.execute("INSERT INTO students2 VALUES (?, ?, ?, ?)", record)
```

Option 3. Insert dict using argument

```
record = {'id': 1, 'name': 'Harry Potter', 'house': 'Gryffindor', 'age': 11}
conn.execute("INSERT INTO students2 VALUES (:id, :name, :house, :age)", record)
```

Insert data dynamically (multiple records)

Option 1. Insert tuple using f-string

```
records = [
    (2, 'Ron Weasley', 'Gryffindor', 11),
    (3, 'Hermione Granger', 'Gryffindor', 11),
    (4, 'Draco Malfoy', 'Slytherin', 11),
    (5, 'Cedric Diggory', 'Hufflepuff', 14),
    (6, 'Cho Chang', 'Ravenclaw', 13),
]

for record in records:
    query = f"INSERT INTO students2 VALUES {record}"
    conn.execute(query)
```

Insert data dynamically (multiple records)

Option 2. Insert tuple using argument

```
records = [
    (2, 'Ron Weasley', 'Gryffindor', 11),
    (3, 'Hermione Granger', 'Gryffindor', 11),
    (4, 'Draco Malfoy', 'Slytherin', 11),
    (5, 'Cedric Diggory', 'Hufflepuff', 14),
    (6, 'Cho Chang', 'Ravenclaw', 13),
]

for record in records:
    query = "INSERT INTO students2 VALUES (?, ?, ?, ?)"
    conn.execute(query, record)
```

Insert data dynamically (multiple records)

Option 3. Insert dict using argument

```
records = [
    {'id': 2, 'name': 'Ron Weasley', 'house': 'Gryffindor', 'age': 11},
    {'id': 3, 'name': 'Hermione Granger', 'house': 'Gryffindor', 'age': 11},
    {'id': 4, 'name': 'Draco Malfoy', 'house': 'Slytherin', 'age': 11},
    {'id': 5, 'name': 'Cedric Diggory', 'house': 'Hufflepuff', 'age': 14},
    {'id': 6, 'name': 'Cho Chang', 'house': 'Ravenclaw', 'age': 13},
]

for record in records:
    query = "INSERT INTO students2 VALUES (:id, :name, :house, :age)"
    conn.execute(query, record)
```

`execute()` vs. `executemany()`

`execute()`: execute one query per record

- Pros: simple and straightforward; error handling per record
- Cons: inefficient for large datasets

```
for record in records:
    conn.execute("INSERT INTO students2 VALUES (?, ?, ?, ?)", record)
```

`executemany()`: execute one query for multiple records

- Pros: more efficient for large datasets
- Cons: less control over error handling

```
conn.executemany("INSERT INTO students2 VALUES (?, ?, ?, ?)", records)
```

Commit and close

- `commit()`: commit the current transactions (save the changes to the database)
- `close()`: close the database connection

```
conn.commit()
conn.close()
```

💻 SQL murder mystery

Create a new table and populate it with query results

Q1. Execute SELECT query to answer:

> I don't know her name but I know she's around 5'5" (65") or 5'7" (67"). She has red hair and she drives a Tesla Model S.

SELECT `id`, `name`, `license_id` columns only

💻 SQL murder mystery

Create a new table and populate it with query results

Q2. Execute CREATE TABLE query to create a new table `suspect`

Table: `suspect`

Column	Type
id	INT
name	TEXT
license_id	INT

Q3. Execute INSERT INTO query to populate `suspect` with the output from Q1

SQLite FAQ

Where is my database?

- Current working directory of your Python environment (e.g., `mydatabase.db`)
- If you delete the file, the database is gone!

"Database is locked" error?

- because SQLite allows only one write operation at a time
- try closing other connections to the same database (`conn.close()`)
- if you're using Jupyter Notebook, restart the kernel
- if all else fails, delete the original db file (e.g., `mydatabase.db`) and create a new one