SQL in Python

⏪ Recap

- **Subquery**

- **Aggregation**

- **Grouping**

- **Joining**

⏪ Recap

> I know she's around 5'5" (65") or 5'7" (67"). She has red hair and she drives a Tesla Model S.

```sql
select id
from drivers_license
where hair_color = "red"
and car_make = "Tesla"
and car_model = "Model S"
and height between 65 and 67
```

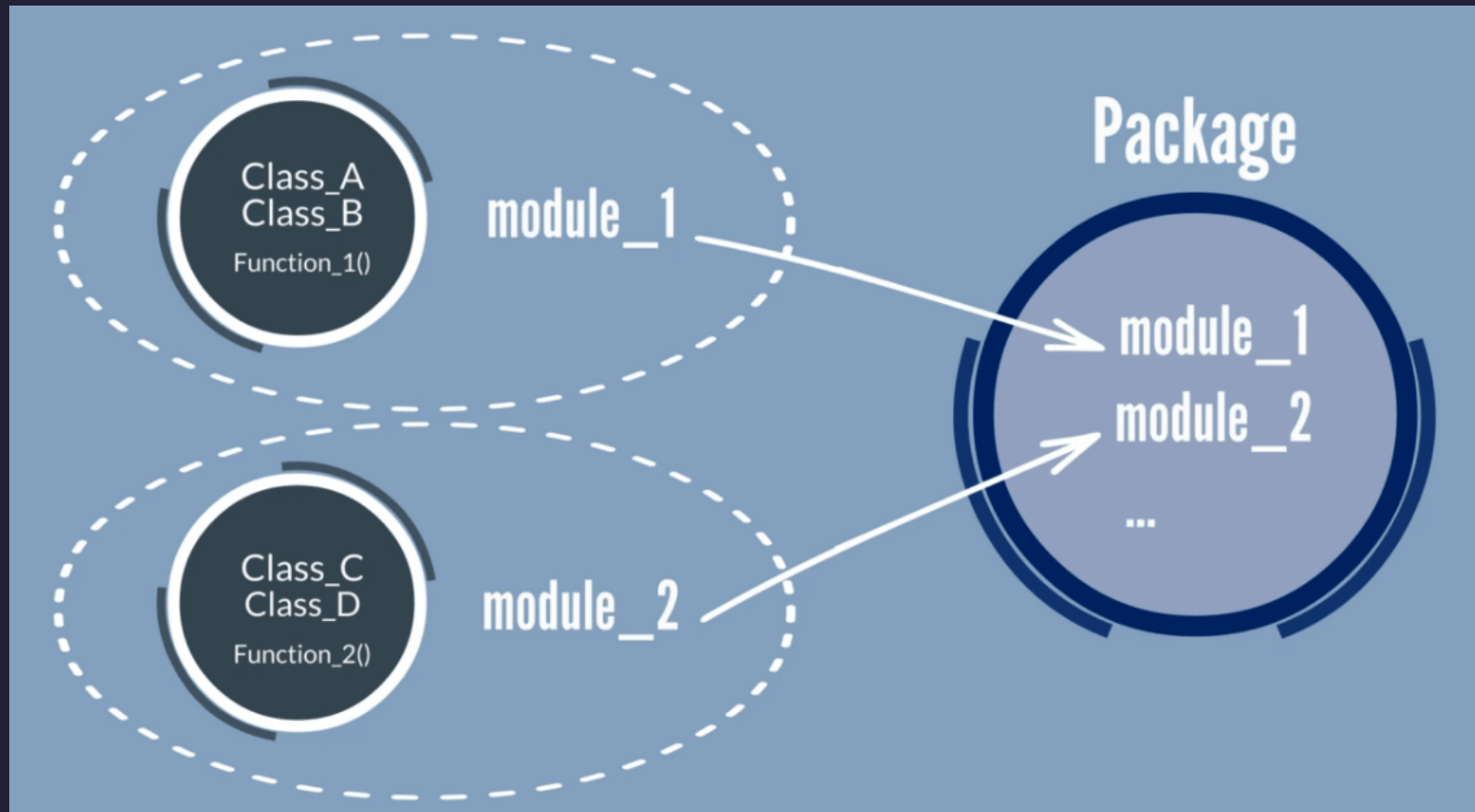> I know that she attended the SQL Symphony Concert 3 times in December 2017.

```sql
select person_id
from facebook_event_checkin
where event_name like "%SQL Symphony Concert%"
and date between 20170101 and 20171231
group by person_id
having count(*)=3
```

⏪ Recap

```sql
-- Subquery
select *
from person
where license_id in (...)
and id in (...)
```

Packages

functions < modules < packages = libraries

Built-in functions

- `print()`
- `type()`
- `len()`
- `range()`
- `input()`

  ...

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

Built-in modules / packages

- `math`
- `random`
- `datetime`
- `os`
- `sqlite3`
  …

https://docs.python.org/3/py-modindex.html

`math`

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

`import` module

```python
import math

print(math.pi)
print(math.sqrt(4))
print(math.pow(2, 3))
print(math.floor(3.14))
print(math.ceil(3.14))
print(math.factorial(5))
```

from module **import** function

```python
from math import pi, sqrt

print(pi)
print(sqrt(4))
```

as to give alias

```python
import math as m

print(m.pi)

print(m.sqrt(4))
```

Install third party packages

- Python Package Index (PyPI; https://pypi.org/)

- `pip install <package_name>`

- `!pip install <package_name>` on Jupyter Notebook

`sqlite3`

a Python library that provides a SQL interface to the SQLite database engine

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

SQLite

Lightweight disk-based database that doesn't require a separate server process

Pros:

- no need to install a database server
- no need to configure a database
- no need to worry about access control

Cons:

- not suitable for large-scale applications
- not suitable for client-server applications
- not suitable for multi-user applications

Working with databases in Python (DB-API 2.0)

1. Connect to a database (`connect()`)

2. Execute a query (`execute()`)

3. Get query results (`fetchone()` , `fetchmany(n)` , `fetchall()`)

4. Close connection (`close()`)

Connect to a database

- `connect()` : create a connection object that enables access to a database
  - `connect('name.db')` : create or load a database file in the current directory

```python
import sqlite3

conn = sqlite3.connect('harrypoter.db')
```

Execute a query (DQL)

- `execute()` : execute a query

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

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

```python
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()
```

tuple

list : mutable

tuple : immutable

```python
# list
cities = ["Montreal", "Toronto", "Vancouver", "Detroit"]
print(type(cities))
cities[0] = "New York"

# tuple
cities = ("Montreal", "Toronto", "Vancouver", "Detroit")
print(type(cities))
cities[0] = "New York" # TypeError: 'tuple' object does not support item assignment
```

Execute a query (DDL)

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

Execute a query (DML)

```python
query = """
    INSERT INTO students (id, name, house, age)
    VALUES (1, 'Harry Potter', 'Gryffindor', 11)
"""

conn.execute(query)
```

Commit and close

- `commit()` : commit the current transactions

- `close()` : close the database connection

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

🖥️ Query `harrypotter.db` with SQL in Pandas

- **connect to** `harrypotter.db` **using** `sqlite3`

- **execute queries to answer the following questions:**

  - What year was Harry Potter born?

  - List the name of students who are born after 1980

  - What is the name of the oldest student?

- **print query results**

🖥️ Query `harrypotter.db` with SQL in Pandas (solution)

```python
import sqlite3
import pandas as pd

conn = sqlite3.connect('harrypotter.db')

query1 = "SELECT * FROM students WHERE first_name = 'Harry'"
query2 = "SELECT * FROM students WHERE birthdate > 1980"
query3 = "SELECT * FROM students ORDER BY birthdate DESC LIMIT 1"

df1 = pd.read_sql_query(query1, conn)
df2 = pd.read_sql_query(query2, conn)
df3 = pd.read_sql_query(query3, conn)
```