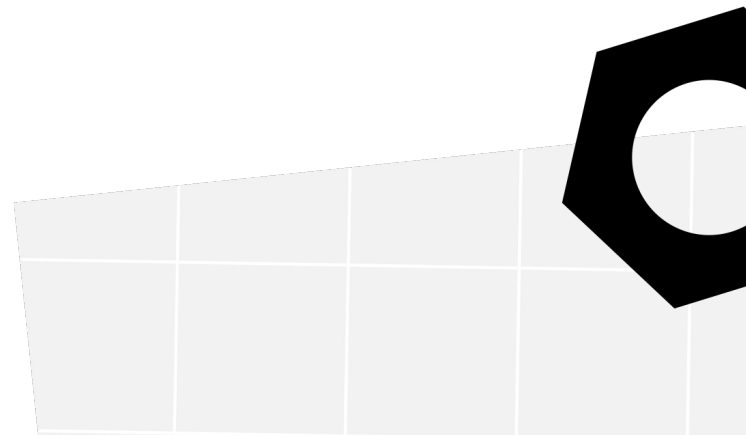




Przetwarzanie zbiorów danych

Kurs Data Science

Część II



JĘZYK MODYFIKACJI DANYCH





INSERT



Za pomocą instrukcji INSERT istnieje możliwość wprowadzania nowych rekordów do stworzonych tabel.

Polecenie to umożliwia wprowadzenie nowego wiersza do tabeli, gdzie każda wartość odpowiada kolumnie o analogicznej pozycji:

wartość1 -> kolumna1

wartość2 -> kolumna2

wartośćN -> kolumnaN

Wprowadzenie danych może się również odbywać bez podawania nazw kolumn, np.:

```
INSERT INTO tabela [(kolumna1, kolumna2, ..., kolumnaN)]  
VALUES (wartość1, wartość2, ..., wartośćN);
```

```
INSERT INTO Produkt (ProduktId, Nazwa, Opis)  
VALUES (1, 'Macbook Pro 16', 'Late 2019');
```

```
INSERT INTO Produkt  
VALUES (1, 'Macbook Pro 16', 'Late 2019');
```





UPDATE



Instrukcja UPDATE umożliwia aktualizowanie wprowadzonych już rekordów.

```
UPDATE tabela SET kolumna1=wartość1, kolumna2=wartość2, ...,  
kolumnaN=wartośćN [WHERE warunek];
```

```
UPDATE Produkt SET Nazwa='Macbook Pro' where Nazwa='Macbook';
```





DELETE

Instrukcja DELETE umożliwia usuwanie rekordów z tabel bazodanowych.

```
DELETE FROM Produkt [WHERE warunek];
```

```
DELETE FROM Produkt;
```

```
DELETE FROM Produkt WHERE Nazwa='Macbook' AND ProduktID  
In(2,3,10);
```





Zadanie 9

1. Dodaj dane do bazy danych:
 - a. clients:
 - i. 'Jan', 'Kowalski', 'ul. Florianska 12', 'Krakow'
 - ii. 'Andrzej', 'Nowak', 'ul. Saska 43', 'Wroclaw'
 - b. cars:
 - i. 'Seat', 'Leon', 2016, 80, 200
 - ii. 'Toyota', 'Avensis', 2014, 72, 100
 - c. bookings:
 - i. 1, 2, '2020-07-05', '2020-07-06', 100
 - ii. 2, 2, '2020-07-10', '2020-07-12', 200
2. *Zastąp dane wybranego klienta swoimi danymi.
3. *W związku z RODO – usuń swoje dane.
4. *Dodaj 2 nowych klientów.



Odpowiedź



```
INSERT INTO clients (name, surname, address, city)
```

```
VALUES
```

```
('Jan', 'Kowalski', 'ul. Florianska 12', 'Krakow'),  
( 'Andrzej', 'Nowak', 'ul. Saska 43', 'Wroclaw' );
```

```
INSERT INTO cars (producer, model, year, horse_power,  
price_per_day)
```

```
VALUES
```

```
('Seat', 'Leon', 2016, 80, 200),  
( 'Toyota', 'Avensis', 2014, 72, 100);
```

```
INSERT INTO bookings (client_id, car_id, start_date,  
end_date, total_amount)
```

```
VALUES
```

```
(1, 2, '2020-07-05', '2020-07-06', 100),  
(2, 2, '2020-07-10', '2020-07-12', 200);
```



SELECT

Dane z tabel mogą być pobierane za pomocą instrukcji SELECT. Sama instrukcja może być bardzo rozbudowana ze względu na sporą liczbę dostępnych klauzul.

Dane wyjściowe można sortować (rosnąco - ASC, malejąco - DESC).

Instrukcja SELECT może dodatkowo ograniczać zestaw zwracanych danych, do tych spełniających zadany warunek, co jest realizowane za pomocą klauzuli WHERE:

Dodatkowo za pomocą BETWEEN można:

```
SELECT kolumna1, kolumna2, ..., kolumnaN  
FROM tabela  
[WHERE warunek]  
[ORDER BY kolumna1, kolumna2, ..., kolumnaN [ASC | DESC]]
```

```
SELECT *  
FROM Produkt
```

```
SELECT *  
FROM Produkt  
ORDER BY Nazwa ASC
```

```
SELECT *  
FROM Produkt  
WHERE Nazwa='Macbook' AND Opis LIKE 'Late %';
```

```
SELECT *  
FROM Produkt  
WHERE ProduktId BETWEEN 3 and 10;
```




Wyświetlenie tabeli na podstawie wybranych kolumn

W celu wyświetlenia danych z określonych kolumn należy wskazać ich nazwy, oddzielając je przecinkiem, np.:

```
SELECT Nazwa, Opis FROM Produkt;
```

Powyższe zapytanie wyświetli dane tylko i wyłącznie dla wskazanych kolumn, nieujęte kolumny będą ignorowane.



Aliasy

W ramach zapytań SELECT można zmodyfikować nazwy wyświetlanych kolumn, co jest realizowane za pomocą tzw. aliasów. Alias definiujemy dodając słowo kluczowe AS oraz wybraną przez nas nazwę po nazwie kolumny, np.:

SELECT

Nazwa AS nazwa_produktu,

Opis AS opis_produktu

FROM Produkt;

Powyższe zapytanie w zwróconym wyniku wyświetli nazwy nazwa_produktu oraz opis_produktu, zamiast rzeczywistych nazw tych kolumn.

Kryteria pobierania danych

Określanie konkretnych kryteriów pobierania danych realizowane jest za pomocą klauzuli WHERE. W ramach niej możemy wykorzystywać wiele operatorów zarówno relacyjnych, jak i logicznych.

Operatory relacyjne

Operator	Opis
=	Równość
>	Większe niż
<	Mniejsze niż
>=	Większe bądź równe
<=	Mniejsze bądź równe
!=	Różne
BETWEEN	znajdujące się w zakresie
LIKE	Wyszukiwanie w ramach określonego wzorca
IN	Znajdujące się w podanym zbiorze

Operatory logiczne

Operator	Opis
AND	Iloczyn logiczny, zwraca wartość true jeśli oba argumenty mają wartość true
&&	Iloczyn logiczny, znaczenie takie samo jak AND
OR	Suma logiczna, zwraca wartość true jeśli jeden z argumentów ma wartość true
	Suma logiczna, znaczenie takie samo jak OR
NOT	Negacja logiczna. Zmienia wartość argumentu na przeciwną
!	Negacja logiczna, znaczenie takie samo jak NOT



Przykłady



```
SELECT * FROM Produkt WHERE Nazwa='Macbook';
```

Powyższe zapytanie odpowiedzialne jest za wyszukanie wszystkich rekordów, których nazwa to Macbook.

```
SELECT * FROM Produkt WHERE Nazwa='Macbook' AND Opis LIKE 'Late%';
```

Powyższe zapytanie odpowiedzialne jest za pobranie wszystkich rekordów, które mają nazwę Macbook oraz opis wskazuje na to, że wartość zaczyna się od znaków Late. Operator LIKE, oprócz znaku %, który oznacza wiele (również zero) dowolnych znaków, pozwala również wykorzystać znak _, który reprezentuje dowolny pojedynczy znak.

```
SELECT * FROM Produkt WHERE ProduktId BETWEEN 3 and 10;
```

Powyższe zapytanie pozwoli znaleźć wszystkie produkty, które mają ProduktId z przedziału 3 i 10. Warto zauważyć, że zarówno dolna, jak i górna wartość graniczna jest uwzględniana podczas wyszukiwania.

```
SELECT * FROM Produkt WHERE Nazwa IN('Macbook', 'Dell');
```

W ramach zapytania wyżej można pobrać wszystkie rekordy których Nazwa jest jedną ze zdefiniowanych w ramach operatora IN. Pamiętajmy o tym, że wykorzystując wiele wartości wewnątrz zbioru IN, czas wykonania takiego zapytania może znacznie wzrosnąć.



Ograniczenie liczby rekordów



Ograniczenie zwracanych rekordów może być realizowane za pomocą klauzuli LIMIT, np.:

```
SELECT * FROM Produkt LIMIT 10;
```

Powyższe zapytanie umożliwi zwrócenie pierwszych 10 rekordów z tabeli Produkt.

W ramach tej samej klauzuli możemy ograniczyć pobieranie danych począwszy od konkretnej pozycji:

```
SELECT * FROM Produkt LIMIT 4,3; -- pierwsza liczba oznacza pozycję, druga ilość rekordów
```

Powyższe zapytanie umożliwi zwrócenie rekordów z tabeli Produkt zaczynając od pozycji nr 4.

Dane do zadania:

```
INSERT INTO clients (name, surname, address, city)
VALUES
    ('Michał', 'Taki', 'os. Środkowe 12', 'Poznań'),
    ('Paweł', 'Który', 'ul. Stara 11', 'Gdynia'),
    ('Anna', 'Inna', 'os. Średnie 1', 'Gniezno'),
    ('Alicja', 'Panna', 'os. Duże 33', 'Toruń'),
    ('Damian', 'Papa', 'ul. Skosna 66', 'Warszawa'),
    ('Marek', 'Troska', 'os. Małe 90', 'Radom'),
    ('Jakub', 'Kłos', 'os. Polskie 19', 'Wadowice'),
    ('Lukasz', 'Lis', 'os. Podlaskie 90',
'Białystok');
```

```
INSERT INTO cars (producer, model, year, horse_power,
price_per_day) VALUES
    ('Mercedes', 'CLK', 2018, 190, 400),
    ('Hyundai', 'Coupe', 2014, 165, 300),
    ('Dacia', 'Logan', 2015, 103, 150),
    ('Saab', '95', 2012, 140, 140),
    ('BMW', 'E36', 2007, 110, 80),
    ('Fiat', 'Panda', 2016, 77, 190),
    ('Honda', 'Civic', 2019, 130, 360),
    ('Volvo', 'XC70', 2013, 180, 280);
```

```
INSERT INTO bookings (client_id, car_id, start_date,
end_date, total_amount) VALUES
    (3, 3, '2020-07-06', '2020-07-08', 400),
    (6, 10, '2020-07-10', '2020-07-16', 1680),
    (4, 5, '2020-07-11', '2020-07-14', 450),
    (5, 4, '2020-07-11', '2020-07-13', 600),
    (7, 3, '2020-07-12', '2020-07-14', 800),
    (5, 7, '2020-07-14', '2020-07-17', 240),
    (3, 8, '2020-07-14', '2020-07-16', 380),
    (5, 9, '2020-07-15', '2020-07-18', 1080),
    (6, 10, '2020-07-16', '2020-07-20', 1120),
    (8, 1, '2020-07-16', '2020-07-19', 600),
    (9, 2, '2020-07-16', '2020-07-21', 500),
    (10, 6, '2020-07-17', '2020-07-19', 280),
    (1, 9, '2020-07-17', '2020-07-19', 720),
    (3, 7, '2020-07-18', '2020-07-21', 240),
    (5, 4, '2020-07-18', '2020-07-22', 1200);
```



Zadanie 10



1. Wypisz wszystkie samochody, których rok produkcji jest większy niż 2015.
2. *Wypisz wszystkie rezerwacje, których koszt całkowity znajduje się w przedziale 1000–2555.
3. *Wypisz id wszystkich klientów, których nazwisko zaczyna się na literę 'N' oraz imię kończy się na litery 'ej'.



Odpowiedź



1)

```
SELECT *  
FROM cars  
WHERE year > 2015;
```

2)

```
SELECT *  
FROM bookings  
WHERE total_amount  
BETWEEN 1000 AND 2555;
```

3)

```
SELECT client_id  
FROM clients  
WHERE surname  
LIKE "N%" AND name LIKE "%ej";
```





JĘZYK KONTROLI DANYCH



GRANT



Za pomocą instrukcji GRANT istnieje możliwość modyfikacji uprawnień użytkowników, będących częścią systemu bazodanowego.

```
GRANT prawa [kolumny] ON poziom TO użytkownik  
[IDENTIFIED By hasło] [WITH [GRANT OPTION |  
MAX_QUERIES_PER_HOUR ile | MAX_UPDATES_PER_HOUR ile |  
MAX_USER_CONNECTIONS ile |  
MAX_CONNECTIONS_PER_HOUR]  
];
```

```
GRANT CREATE, SELECT, INSERT, UPDATE, DELETE ON * TO  
sda_user;
```



REVOKE

Za pomocą instrukcji Revoke istnieje możliwość odbierania uprawnień użytkownikom.

```
REVOKE prawa [kolumny] ON obiekt FROM użytkownik;
```

```
REVOKE UPDATE, DELETE ON sda.* FROM sda_user,  
sda_employee;
```

Łączenia w bazach danych





POBIERANIE DANYCH Z KILKU TABEL



Instrukcja Select umożliwia pobieranie danych z kilku tabel:

```
SELECT kolumna1, kolumna2, kolumna3, ...,kolumnaN  
FROM tabela1, tabela2, ..., tabelaN  
[WHERE warunek]  
...;
```

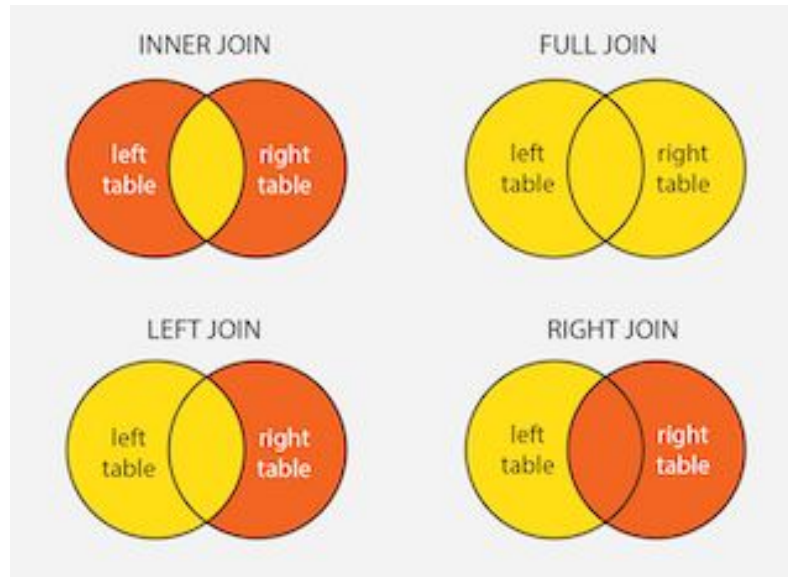
Zapytania takie często wykonujemy nadając aliasy poszczególnym tabelom, np.:=

```
SELECT k.Wydawca, p.Nazwa  
FROM Książka AS k, Produkt AS p;
```

ZŁĄCZENIA

W języku SQL istnieją różne rodzaje złączeń w tym:

- INNER JOIN
- LEFT JOIN/RIGHT JOIN
- FULL JOIN



Przykład:

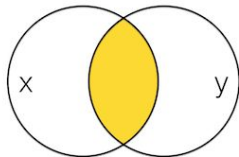
Dla kolejnych przykładów zastosujemy następujące tabele wraz z danymi:

ProducentId	Nazwa	Adres
1	Apple	Cupertino, CA 95014, United States
3	Dell	Berkshire, RG12 1LF, UK
4	Microsoft	1045 La Avenida St, Mountain View, CA 94043, United States



Id	Nazwa	ProducetnId
1	Macbook Pro 16	1
2	Iphone 11 Pro	1
3	Dell XPS 15	3
4	Lenovo ThinkPad 13	

inner_join(x, y)



INNER JOIN

Złączenie tego typu umożliwia połączenie danych tabel na wzór zwykłego polecenia SELECT z uwzględnieniem tabel (klauzula ON definiuje warunek złączenia).

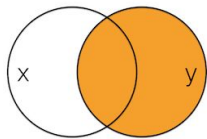
```
SELECT kolumna1, kolumna2, ..., kolumnaN  
FROM tabela1  
[INNER] JOIN tabela2 [ON warunek];
```

Dla prezentowanego przykładu polecenie będzie przyjmować poniższą formę:

```
SELECT ProduktId, Produkt.Nazwa, Produkt.ProductentId,  
Producent.ProducentId, Producent.Nazwa, Producent.Adres  
FROM Produkt  
INNER JOIN Producent ON Produkt.ProducentId=Producent.Producent.Id;
```

Id	Nazwa	ProducentId	Nazwa	Adres
1	Macbook Pro 16	1	Apple	Cupertino, CA 95014, United States
2	Iphone 11 Pro	1	Apple	Cupertino, CA 95014, United States
3	Dell XPS 15	3	Dell	Berkshire, RG12 1LF, UK

right_join(x, y)



LEFT JOIN/RIGHT JOIN

Złączenie tego typu pozwala uwzględnić wynikowe dane, które nie są powiązane relacją ze złączoną tabelą. W skrócie, jeśli w tabela1 znajdują się rekordy, które nie są skorelowane z rekordami tabela2 to i tak zostaną uwzględnione w złączeniu, a brakujące wartości zostaną uzupełnione wartościami NULL.

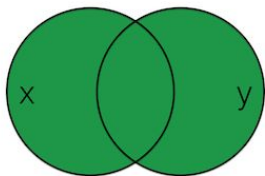
```
SELECT kolumna1, kolumna2, ..., kolumnaN  
FROM tabela1  
[LEFT] JOIN tabela2 [ON warunek];
```

Dla prezentowanego przykładu polecenie będzie przyjmować poniższą formę:

```
SELECT ProduktId, Produkt.Nazwa, Produkt.ProductentId,  
Producent.ProducentId, Producent.Nazwa, Producent.Adres  
FROM Produkt  
LEFT JOIN Producent ON  
Produkt.ProducentId=Producent.Producent.Id;
```

Id	Nazwa	ProducentId	Nazwa	Adres
1	Macbook Pro 16	1	Apple	Cupertino, CA 95014, United States
2	Iphone 11 Pro	1	Apple	Cupertino, CA 95014, United States
3	Dell XPS 15	3	Dell	Berkshire, RG12 1LF, UK
4	Lenovo ThinkPad 13	NULL	NULL	NULL

full_join(x, y)



FULL JOIN

Złączenie to umożliwia pobranie wszystkich rekordów z dwóch tabel. Biorąc pod uwagę zarówno rekordy z tabela1, które nie mają swoich odpowiedników w tabela2, jak i odwrotną sytuację.

```
SELECT kolumna1, kolumna2, ..., kolumnaN FROM tabela1 [FULL OUTER]  
JOIN tabela2 [ON warunek];
```

Dla prezentowanego przykładu polecenie będzie przyjmować poniższą formę:

```
SELECT ProduktId, Produkt.Nazwa, Produkt.ProductentId,  
Producent.ProducentId, Producent.Nazwa, Producent.Adres FROM  
Produkt FULL OUTER JOIN Producent ON  
Produkt.ProducentId=Producent.ProducentId;
```

Id	Nazwa	ProducentId	Nazwa	Adres
1	Macbook Pro 16	1	Apple	Cupertino, CA 95014, United States
2	Iphone 11 Pro	1	Apple	Cupertino, CA 95014, United States
3	Dell XPS 15	3	Dell	Berkshire, RG12 1LF, UK
NULL	NULL	4	Microsoft	1045 La Avenida St, Mountain View, CA 94043, United States
4	Lenovo ThinkPad 13	NULL	NULL	NULL



Zadanie 11



1. Wypisz imiona wszystkich klientów oraz poniesione przez nich koszty rezerwacji, które są większe niż 1000.
2. Wypisz miasto zamieszkania wszystkich klientów, którzy wypożyczyli samochód w okresie 12-20.07.2020, a moc silnika wypożyczonego samochodu jest nie większa niż 120, sortując po największych kosztach wynajmu.
3. *Wypisz liczbę wypożyczonych samochodów, dla których dzienny koszt wypożyczenia jest większy lub równy 300, grupując samochody po mocy silnika, sortując od najmniejszej.
4. *Wypisz sumę kosztów wszystkich rezerwacji, które zostały zrealizowane w okresie 14-18.07.2020.
5. *Wypisz:
 - a. średnią liczbę pieniędzy wydanych przez każdego klienta - nazewnictwo kolumny: Srednia_wartosc_rezerwacji
 - b. liczbę wypożyczonych samochodów dla każdego klienta, uwzględniając tylko tych klientów, którzy wypożyczyli co najmniej dwa samochody - nazewnictwo kolumny: Liczba_wypożyczonych_samochodow
 - c. imię i nazwisko klienta - nazewnictwo kolumn: Imię, Nazwisko
 - d. sortując po największej liczbie wypożyczonych samochodów. Wszystko jednym zapytaniem.

Odpowiedź

1)

```
SELECT c.name, b.total_amount
FROM bookings b
JOIN clients c ON c.client_id = b.client_id
WHERE b.total_amount > 1000;
```

2)

```
SELECT c.city, b.total_amount
FROM clients c
JOIN bookings b ON c.client_id = b.client_id
JOIN cars r ON b.car_id = r.car_id
WHERE b.start_date >= '2020-07-12'
AND b.end_date <= '2020-07-20'
AND r.horse_power <= 120
ORDER BY b.total_amount DESC;
```

3)

```
SELECT COUNT(b.car_id)
FROM bookings b
JOIN cars r ON b.car_id = r.car_id
WHERE r.price_per_day >= 300
GROUP BY r.horse_power
ORDER BY r.horse_power;
```

4)

```
SELECT SUM(total_amount)
FROM bookings
WHERE start_date >= '2020-07-14'
AND end_date <= '2020-07-18';
```

5)

```
SELECT AVG(b.total_amount) AS
Srednia_wartosc_rezerwacji,
COUNT(b.car_id) AS
Liczba_wypożyczonych_samochodow,
c.name AS Imie, c.surname AS Nazwisko
FROM bookings b
JOIN clients c ON c.client_id = b.client_id
GROUP BY b.client_id
HAVING Liczba_wypożyczonych_samochodow >= 2
ORDER BY Liczba_wypożyczonych_samochodow DESC;
```

Grupowanie danych



FUNKCJE STATYSTYCZNE I AGREGUJĄCE

Język SQL udostępnia zestaw wielu funkcji statystycznych i agregujących, m. in:

Funkcja	Opis
AVG	Obliczanie wartości średniej z wartości zwróconych w zapytaniu
COUNT	Obliczanie liczby wartości zwróconych w zapytaniu
MIN	Zwracanie wartości minimalnej z zwróconego zapytania
MAX	Zwracanie wartości maksymalnej z zwróconego zapytania
SUM	Obliczanie sumy wartości zwróconych w ramach zapytania



COUNT



Funkcja ta umożliwia zwrócenie liczby wszystkich uzyskanych z zapytania rekordów. Operator * jest aliasem dla wszystkich wierszy. Wynikiem zapytania będzie tabela z kolumną o nazwie COUNT(*) i jednym rekordem określającym liczbę wszystkich rekordów tabeli macierzystej.


W celu zmiany nazwy kolumny wynikowej można skorzystać z aliasu:

W ramach zapytania można określić też warunek, który docelowo powinien zmniejszyć liczbę przeszukiwanych rekordów:

```
SELECT COUNT(*)  
FROM Produkt;
```

```
SELECT COUNT(*) AS 'Liczba produktów'  
FROM Produkt;
```

```
SELECT COUNT(*) AS 'Liczba produktów'  
FROM Produkt  
WHERE Nazwa='Macbook';
```





OBLICZANIE ŚREDNIEJ



Funkcja ta umożliwia obliczanie średniej z wartości zwróconej w zapytaniu:

```
SELECT AVG(Cena) AS 'Średnia wartość  
komputera'  
FROM Produkt
```

Dodatkowo, standardowo jak dla każdego zapytania SELECT możemy wprowadzić klauzulę WHERE, co spowoduje zwrócenie rezultatu na podstawie rekordów spełniających podany w zapytaniu warunek:

```
SELECT AVG(Cena) AS 'Średnia wartość  
produktu'  
FROM Produkt  
WHERE Cena > 10;
```




MIN I MAX

Funkcje te umożliwiają wyszukiwanie wartości minimalnej i maksymalnej z danych zwróconych w zapytaniu:

```
SELECT MIN(Cena) AS 'Najtańszy komputer'  
FROM Komputer  
WHERE ProducentId=10;
```

```
SELECT MAX(Cena) AS 'Najdroższy komputer'  
FROM Komputer  
WHERE ProducentId=4;
```



SUM

Funkcja ta umożliwia zwrócenie sumy konkretnych wartości na podstawie uzyskanych rezultatów:

```
SELECT SUM(Cena) AS `Łączna wartość komputerów  
APPLE` FROM Komputer  
WHERE ProducentId=4;
```




GRUPOWANIE WYNIKÓW ZAPYTAŃ

Klauzula GROUP BY umożliwia nam grupowanie wyników zapytań w ramach wybranej kolumny, bądź też wielu kolumn:

Dzięki tej klauzuli istnieje możliwość zwrócenia, np. liczby produktów dla każdego z producentów w bazie danych. Wynikiem zapytania będzie nowa tabela zawierająca liczbę rekordów pokrywającą się z liczbą producentów, mającą skorelowane produkty w bazie.


Kolejny reprezentuje wykorzystanie klauzuli GROUP BY z dodatkowym warunkiem:



```
SELECT kolumna1, kolumna2, ..., kolumnaN  
FROM tabela1, tabela2, ..., tabelaN  
WHERE warunki  
GROUP BY kolumna1, kolumna2, ..., kolumnaN;
```

```
SELECT COUNT(*)  
FROM Produkt  
GROUP BY ProducentId;
```

```
SELECT MIN(Cena), Max(Cena)  
FROM Komputer  
WHERE Procesor='Intel I9'  
GROUP BY ProducentId;
```





WARUNKI GRUPOWANIA

Klauzula HAVING umożliwia
ograniczanie wyników
zapytań grupujących:

```
SELECT kolumna1, kolumna2, ..., kolumnaN  
FROM tabela1, tabela2, ..., tabelaN  
WHERE warunki_where  
GROUP BY kolumna1, kolumna2, ..., kolumnaN  
HAVING warunki_having;
```

```
SELECT SUM(Cena)  
FROM Komputer  
WHERE Procesor='Intel I9'  
GROUP BY ProducentId  
HAVING COUNT(*) > 40;
```



SQLAlchemy





SQLAlchemy

Otwarto-źródłowa biblioteka programistyczna napisana w języku programowania Python i służąca do pracy z bazami danych.



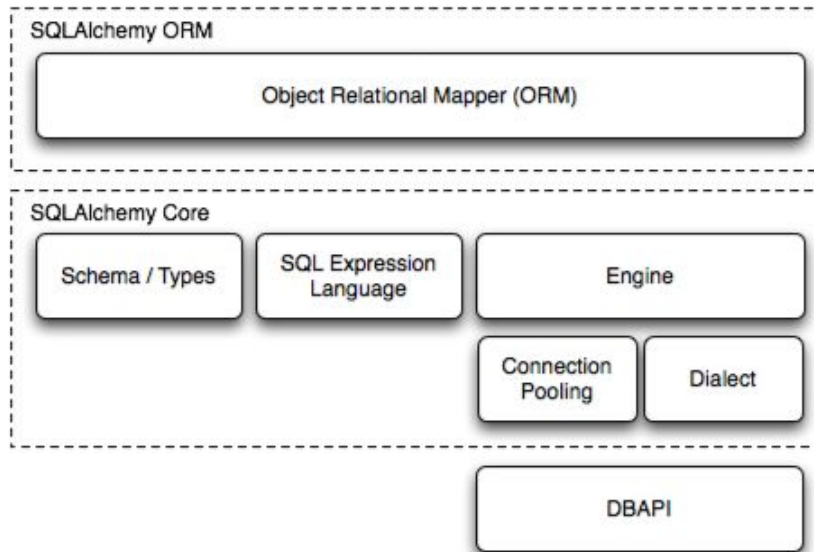
SQLAlchemy

SQLAlchemy

Składa się z dwóch części:

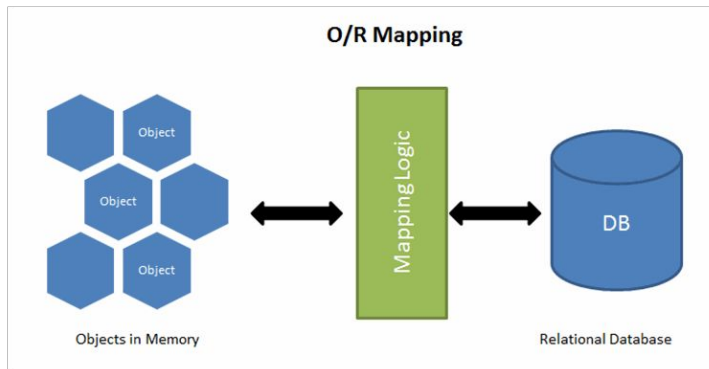
- Core
 - Daje ujednolicone Pythonowe API dostępne do różnych typów baz danych.
 - Pozwala także na wygodne stosowanie Pythonowych wyrażeń w zapytaniach SQL.
- ORM (Object-Relational Mapping)
 - Opcjonalny komponent pozwalający opisać za pomocą klas tabele w bazie danych.
 - Pozwala także na manipulowanie danymi poprzez operacje na obiektach (bez pisania zapytań SQL przez samego programistę).

Na chwilę obecną wspierane są bazy takie jak: SQLite, PostgreSQL, MySQL, Oracle, MS-SQL, Firebird czy Sybase.



ORM

ORM to skrótowe oznaczenie dla "mapowanie obiektowo-relacyjne" (od angielskiego Object-Relational Mapping). Chodzi więc o zamianę danych w postaci tabelarycznej (relacji w bazie danych) na obiekty, albo w drugą stronę. Jest nowoczesnym podejściem do zagadnienia współpracy z bazą danych, wykorzystującym filozofię programowania obiektowego.





Zapytania SQL, czy ORM?



ORM

- Idealne do prostszych zapytań i pracy z wieloma dialektami jednocześnie
- Dostęp do wielu open-sourcowych bibliotek umożliwiających np. serializowanie danych

Zapytania SQL

- Idealne do bardziej złożonych operacji na bazach danych
- Wykorzystuje pełen potencjał silnika bazodanowego

Połączenie z bazą danych

By połączyć się z wybraną bazą danych musimy użyć funkcji **create_engine**:

Funkcja ta tworzy obiekt klasy 'Engine' bazujący na podanym URLu.

Funkcja ta tworzy obiekt klasy 'Engine' bazujący na podanym URLu:
`create_engine('dialect+driver://username:password@host:port/database')`

Przykład:

```
create_engine('postgresql+pg8000://user:password@localhost:3306/car_rental', echo = True)
```

Flaga echo włącza wyświetlanie/logowanie czynności, zapytań jakie SQLAlchemy będzie wykonywać na danej bazie danych.

- dialect – rodzaj sqlowego dialektu (qlite, mysql, postgresql, oracle, or mssql)
- driver – nazwa DBAPI (Python Database API) używanego w celu połączenia z bazą danych. Jeżeli nie zostanie podana to interpreter wykorzysta domyślną dla danego dialectu

Bazy plikowe SQLite wymagają podania ścieżki, można także tworzyć ulotne bazy w pamięci RAM (znikają po zakończeniu działania skryptu).

Dla baz takich jak MySQL czy PostgreSQL podajemy host i nazwę bazy danych oraz opcjonalnie inne parametry.



Standardowe kwerendy



Aby wykonać dowolną kwerendę wystarczy użyć metody execute:

```
mysql_db = create_engine('mysql://login:hasło@localhost/nazwa_bazy', echo=True)
s = mysql_db.execute("SHOW TABLES;")
print list(s)
```

Podstawy ORMa

W przypadku użytkowania ORMa każdą tabelę, z którą chcemy pracować musimy opisać klasą. Oto przykład:

Na początku podajemy nazwę tabeli w bazie danych za pomocą `__tablename__`.

Następnie listujemy kolumny tabeli po ich nazwach. Klasa `Column` przyjmuje wiele argumentów, w tym typy pól, np. tekstowe to `String`, a liczbowe to `Integer`. Dla np. MySQL przełoży się to na pola `VARCHAR` i `INTEGER`.

Opcjonalnie możemy definiować metody takie jak np. `__init__` czy `__repr__`.

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String
```

```
# baza dla klas tabel
Base = declarative_base()
```

```
# przykładowa klasa mapująca tabelę z bazy danych
```

```
class User(Base):
    __tablename__ = 'users'
```

```
# pola i ich typy
id = Column(Integer, primary_key=True)
name = Column(String)
fullname = Column(String)
password = Column(String)
```

```
def __init__(self, name, fullname, password):
    self.name = name
    self.fullname = fullname
    self.password = password
```

```
def __repr__(self):
    return "<User('%s', '%s', '%s')>" % (self.name,
self.fullname, self.password)
```



Tworzenie tabel

Mamy klasę, która nie jest jeszcze połączona z żadną realną tabelą w istniejącej bazie danych. Nasza klasa "User" to metadane tabeli (metadata) w terminologii SQLAlchemy. Na tych metadanych możemy operować przez Base.metadata. Np. jeżeli tabelę nie istnieją to trzeba je stworzyć:

```
engine = create_engine('sqlite:///memory:', echo=True)
```

```
Base.metadata.create_all(engine)
```



Inicjalizacja środowiska Google Colab



```
!apt-get install mysql-server > /dev/null
```

```
!service mysql start
```

```
!mysql -e "ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'root'"
```

```
!pip -q install PyMySQL
```

```
from sqlalchemy import create_engine
```

```
engine = create_engine("mysql+pymysql://root:root@/")
```

```
engine.execute("CREATE DATABASE car_rental") #create db
```

```
engine.execute("USE car_rental") # select new db
```





Zadanie 12

1. Stwórz tabele cars, clients, bookings według wytycznych (bez relacji):
 - a. cars: car_id(int, pk), producer(str), model(str), year(int), horse_power(int), price_per_day(int)
 - b. clients: client_id(int, pk), name(str), surname(str), address(str), city(str)
 - c. bookings: booking_id(int, pk), client_id(int), car_id(int), start_date(date), end_date(date), total_amount(int)
2. * Dokonaj tego samego za pomocą zapytania SQLowego.

Odpowiedź

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, String, Integer, Date

engine = create_engine("mysql+pymysql://root:root@/", echo =
True)

engine.execute("CREATE DATABASE car_rental") #create db
engine.execute("USE car_rental") # select new db

eng = create_engine("mysql+pymysql://root:root@/car_rental")
base = declarative_base()

class Cars(base):
    __tablename__ = 'cars'

    car_id = Column(Integer, primary_key=True,
autoincrement=True)
    producer = Column(String(30), nullable=False)
    model = Column(String(30), nullable=False)
    year = Column(Integer, nullable=False)
    horse_power = Column(Integer, nullable=False)
    price_per_day = Column(Integer, nullable=False)
```

```
class Clients(base):
    __tablename__ = 'clients'

    client_id = Column(Integer, primary_key=True,
autoincrement=True)
    name = Column(String(30), nullable=False)
    surname = Column(String(30), nullable=False)
    address = Column(String(30), nullable=False)
    city = Column(String(30), nullable=False)

class Bookings(base):
    __tablename__ = 'bookings'

    booking_id = Column(Integer, primary_key=True,
autoincrement=True)
    client_id = Column(Integer, nullable=False)
    car_id = Column(Integer, nullable=False)
    start_date = Column(Date, nullable=False)
    end_date = Column(Date, nullable=False)
    total_amount = Column(Integer, nullable=False)

base.metadata.create_all(eng)
```




Dodawanie rekordów

By zapisać rekord
potrzebujemy obsługi sesji i
przekazanie w jej ramach
obiektu.

```
from sqlalchemy.orm import sessionmaker

engine = create_engine('sqlite:///memory:', echo=True)

Base.metadata.create_all(engine)

# tworzenie sesji dla danej bazy:
Session = sessionmaker(bind=engine)
session = Session()

ed_user = User('James', 'James Hajto', 'pas(s)y')
session.add(ed_user)
# wykonywanie operacji
session.commit()
print (ed_user.id)
```





Zadanie 13

1. Uzupełnij następującymi danymi tabele clients oraz cars:
 - a. clients: 'Andrzej', 'Nowak', 'ul. Saska 43', 'Wroclaw'
 - b. cars: 'Seat', 'Leon', 2016, 80, 200
2. * Dodaj swoje dane.
3. * Dodaj za pomocą zapytania SQL:
 - a. 'Andrzej', 'Poziomka', 'ul. Saska 43', 'Kraków'
 - b. 'Opel', 'Vectra', 2010, 240, 70000000




Odpowiedź



```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=eng)
session = Session()
client_1 = Clients(name='Jan', surname='Kowalski',
address='ul. Florianska 12', city='Krakow')
car_1 = Cars(producer='Seat', model='Leon', year=2016,
horse_power=80, price_per_day=200)

session.add(client_1)
session.add(car_1)
session.commit()
```





Zapytania

Zapytania wykonujemy na instancji sesji, np:

Lub

Za pomocą instrukcji select

```
for instance in session.query(User).order_by(User.id):  
    print (instance)
```

```
s = select([users]).order_by(users)  
for row in eng.execute(s):  
    print(row)
```



Zadanie 14

1. Sprawdź, czy dane dodane w poprzednim zadaniu zostały zapisane w bazie – dopisz kawałek kodu, który wypisze wszystkie dane z tabel cars oraz clients.
2. *Sprawdź to samo za pomocą zapytania SQL.
3. * Zaktualizuj klasy Cars, Clients, Bookings tak, aby wyświetlanie danych funkcją print() było bardziej przejrzyste na przykład tak, jak poniżej (nie mamy jeszcze dodanej rezerwacji):

```
<Client: id=1, name=Jan, surname=Kowalski, address=ul. Florianska 12,city=Krakow>  
<Car: id=1, producer=Seat, model=Leon, year=2016, horse_power=80,  
price_per_day=200>
```

Odpowiedź

```
...
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=eng)
session = Session()

for client in session.query(Clients).all():
    print(client)

for car in session.query(Cars).all():
    print(car)
```

3) (Example)

```
class Cars(base):
    __tablename__ = 'cars'

    car_id = Column(Integer, primary_key=True,
autoincrement=True)
    producer = Column(String(30), nullable=False)
    model = Column(String(30), nullable=False)
    year = Column(Integer, nullable=False)
    horse_power = Column(Integer, nullable=False)
    price_per_day = Column(Integer, nullable=False)

    def __repr__(self):
        return f'<Car: id={self.car_id},
producer={self.producer},          model={self.model},
year={self.year}, ' \
            f'horse_power={self.horse_power},
price_per_day={self.price_per_day}>'
```

Zadanie 15

1. Potrzebujemy danych, aby móc przeprowadzić serię zapytań. Napisz funkcję `insert_data`, która będzie przyjmować argumenty:
 - a. `base` – klasę bazową modelu, którego instancję chcemy stworzyć
 - b. `params` – słownik z konkretnymi parametrami tworzonego obiektu
2. Przetestuj funkcję na podanych danych:

```
clients = [
    {'name': 'Jan', 'surname': 'Kowalski', 'address': 'ul. Florianska 12', 'city': 'Krakow'},
    {'name': 'Andrzej', 'surname': 'Nowak', 'address': 'ul. Saska 43', 'city': 'Wroclaw'},
    {'name': 'Michal', 'surname': 'Taki', 'address': 'os. Srodkowe 12', 'city': 'Poznan'},
    {'name': 'Pawel', 'surname': 'Ktory', 'address': 'ul. Stara 11', 'city': 'Gdynia'},
    {'name': 'Anna', 'surname': 'Inna', 'address': 'os. Srednie 1', 'city': 'Gniezno'},
    {'name': 'Alicja', 'surname': 'Panna', 'address': 'os. Duze 33', 'city': 'Torun'},
    {'name': 'Damian', 'surname': 'Papa', 'address': 'ul. Skosna 66', 'city': 'Warszawa'},
    {'name': 'Marek', 'surname': 'Troska', 'address': 'os. Male 90', 'city': 'Radom'},
    {'name': 'Jakub', 'surname': 'Klos', 'address': 'os. Polskie 19', 'city': 'Wadowice'},
    {'name': 'Lukasz', 'surname': 'Lis', 'address': 'os. Podlaskie 90', 'city': 'Bialystok'}]

cars = [
    {'producer': 'Seat', 'model': 'Leon', 'year': 2016, 'horse_power': 80, 'price_per_day': 200},
    {'producer': 'Toyota', 'model': 'Avenis', 'year': 2014, 'horse_power': 72, 'price_per_day': 100},
    {'producer': 'Mercedes', 'model': 'CLK', 'year': 2018, 'horse_power': 190, 'price_per_day': 400},
    {'producer': 'Hyundai', 'model': 'Coupe', 'year': 2014, 'horse_power': 165, 'price_per_day': 300},
    {'producer': 'Dacia', 'model': 'Logan', 'year': 2015, 'horse_power': 103, 'price_per_day': 150},
    {'producer': 'Saab', 'model': '95', 'year': 2012, 'horse_power': 140, 'price_per_day': 140},
    {'producer': 'BMW', 'model': 'E36', 'year': 2007, 'horse_power': 110, 'price_per_day': 80},
    {'producer': 'Fiat', 'model': 'Panda', 'year': 2016, 'horse_power': 77, 'price_per_day': 190},
    {'producer': 'Honda', 'model': 'Civic', 'year': 2019, 'horse_power': 130, 'price_per_day': 360},
    {'producer': 'Volvo', 'model': 'XC70', 'year': 2013, 'horse_power': 180, 'price_per_day': 280}]

bookings = [
    {'client_id': 3, 'car_id': 3, 'start_date': '2020-07-06', 'end_date': '2020-07-08', 'total_amount': 400},
    {'client_id': 6, 'car_id': 10, 'start_date': '2020-07-10', 'end_date': '2020-07-16', 'total_amount': 1680},
    {'client_id': 4, 'car_id': 5, 'start_date': '2020-07-11', 'end_date': '2020-07-14', 'total_amount': 450},
    {'client_id': 5, 'car_id': 4, 'start_date': '2020-07-11', 'end_date': '2020-07-13', 'total_amount': 600},
    {'client_id': 7, 'car_id': 3, 'start_date': '2020-07-12', 'end_date': '2020-07-14', 'total_amount': 800},
    {'client_id': 5, 'car_id': 7, 'start_date': '2020-07-14', 'end_date': '2020-07-17', 'total_amount': 240},
    {'client_id': 3, 'car_id': 8, 'start_date': '2020-07-14', 'end_date': '2020-07-16', 'total_amount': 380},
    {'client_id': 5, 'car_id': 9, 'start_date': '2020-07-15', 'end_date': '2020-07-18', 'total_amount': 1080},
    {'client_id': 6, 'car_id': 10, 'start_date': '2020-07-16', 'end_date': '2020-07-20', 'total_amount': 1120},
    {'client_id': 8, 'car_id': 1, 'start_date': '2020-07-16', 'end_date': '2020-07-19', 'total_amount': 600},
    {'client_id': 9, 'car_id': 2, 'start_date': '2020-07-16', 'end_date': '2020-07-21', 'total_amount': 500},
    {'client_id': 10, 'car_id': 6, 'start_date': '2020-07-17', 'end_date': '2020-07-19', 'total_amount': 280},
    {'client_id': 1, 'car_id': 9, 'start_date': '2020-07-17', 'end_date': '2020-07-19', 'total_amount': 720},
    {'client_id': 3, 'car_id': 7, 'start_date': '2020-07-18', 'end_date': '2020-07-21', 'total_amount': 240},
    {'client_id': 5, 'car_id': 4, 'start_date': '2020-07-18', 'end_date': '2020-07-22', 'total_amount': 1200}]
```



Odpowiedź



```
...
def insert_data(session, base, params):
    session.add(base(**params))
    session.commit()

for client in clients:
    insert_data(session, Clients, client)
for car in cars:
    insert_data(session, Cars, car)
for booking in bookings:
    insert_data(session, Bookings, booking)
```





Zadanie 16

1. Wypisz wszystkie rezerwacje dla klienta o id = 3. Spróbuj zarówno za pomocą `query()` jak i funkcji `select()`.
2. *Wypisz wszystkie wypożyczone samochody przez klienta o id = 5. Samochody mogą się powtarzać, chodzi nam o historię wypożyczeń. Spróbuj zarówno za pomocą funkcji `select()`.

Odpowiedź

1)

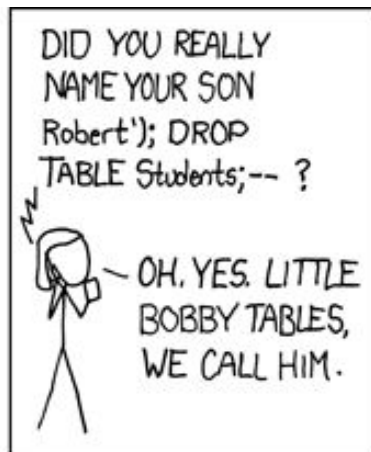
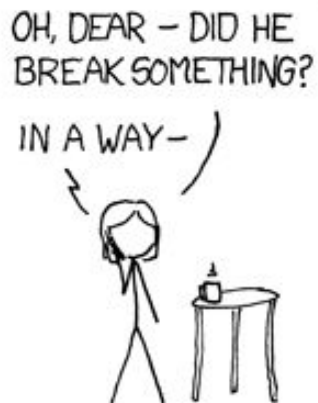
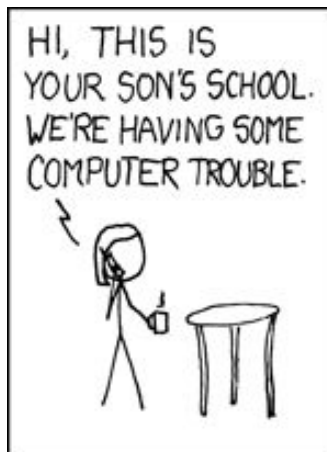
```
...
#1
result = session.query(Bookings).filter(Bookings.client_id == 3)
for booking in result:
    print(booking)
#2
from sqlalchemy.sql import select

conn = eng.connect()
s = select([Bookings]).where(Bookings.client_id == 3)
result = conn.execute(s).fetchall()
print(result)
```

2)

```
from sqlalchemy.sql import select
from sqlalchemy import join

j = join(Bookings, Cars, Bookings.car_id == Cars.car_id)
s = select([Cars]).select_from(j).where(Bookings.client_id == 5)
result = conn.execute(s)
for car in result:
    print(car)
```



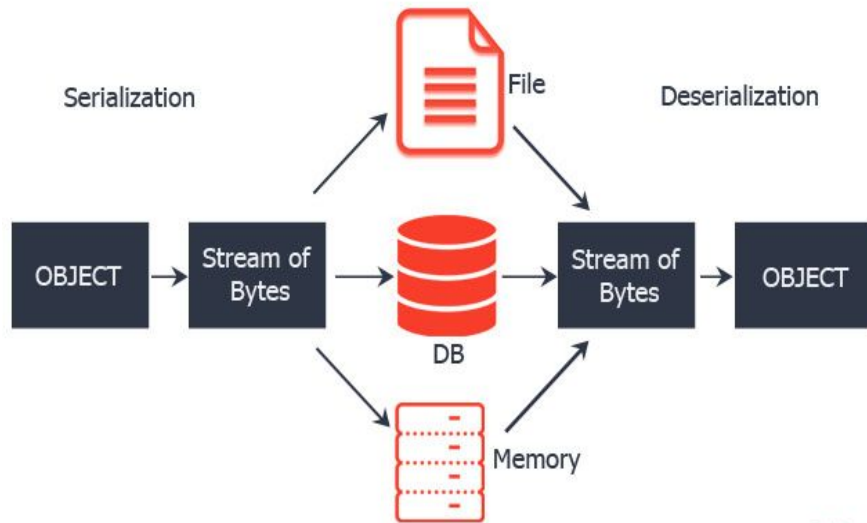
Serializacja plików



Serializacja

Proces przekształcania obiektów, tj. instancji określonych klas, do postaci szeregowej, czyli w strumień bajtów, z zachowaniem aktualnego stanu obiektu.

Serializowany obiekt może zostać utrwalony w pliku dyskowym, przesłany do innego procesu lub innego komputera poprzez sieć. Procesem odwrotnym do serializacji jest deserializacja.





Dane



Flat

```
{  
  "Type" : "A",  
  "field1": "value1",  
  "field2": "value2",  
  "field3": "value3"  
}
```

Nested

```
{  
  "A"  
  {  
    "field1": "value1",  
    "field2": "value2",  
    "field3": "value3"  
  }  
}
```





Pickle(nested data)



Pickle umożliwia serializację i deserializację struktury i właściwości obiektów.

- “Pickling” – Jest to proces w których Pythonowy obiekt konwertowany jest na ciąg bajtów
- “Unpickling” – Jest to odwrotna operacja

Przykład:

```
>>> grades = { 'Alice': 89, 'Bob': 72, 'Charles': 87 }
```

```
#Use dumps to convert the object to a serialized string
```

```
>>> serial_grades = pickle.dumps(grades)
```

```
b'\x80\x03}q\x00(X\x05\x00\x00\x00Aliceq\x01KYY\x03\x00\x00\x00Bobq\x02KHX\x07\x00\x00\x00Charlesq\x03KWu.'
```

```
#Use loads to de-serialize an object
```

```
>>> received_grades = pickle.loads(serial_grades)
```

```
{'Alice': 89, 'Bob': 72, 'Charles': 87}
```

Pandas Profiling

Generuje automatyczny raport na podstawie dostarczonej ramki danych umożliwiający przeprowadzenie EDA (Exploratory Data Analysis)

Dla każdej kolumny obliczane są podstawowe statystyki, a wszystko przedstawione jest w formie interaktywnego HTMLowego raportu:

- typ kolumny
- unikalne i brakujące wartości
- kwantyle, minima, maksima, zakres
- średnia, odchylenie standardowe
- najczęściej występujące wartości
- histogram

i wiele wiele innych!



Pandas Profiling





Tworzenie raportu



Tworzenie raportu jest banalnie proste, co widać z prawej strony.

W przypadku dużych zbiorów danych zaleca się wyłączenie opcji tworzenia wykresów interakcji. W tym celu należy ustawić odpowiednią flagę.


Produkt ciągle się rozwija, co można śledzić pod poniższym linkiem:

<https://github.com/pandas-profiling/pandas-profiling>

```
from pandas_profiling import ProfileReport

# Generate the Profiling Report
profile = ProfileReport(df,
                        title="Titanic Dataset",
                        html={ 'style': { 'full_width':
True}},
                        sort="None")

# Save to file
profile.to_file('report.html')
```



W pierwszym podstawowym widoku mamy możliwość zapoznać się z ogólnymi statystykami oraz określeniem ilości poszczególnych typów w ramce danych

Overview Warnings **11** Reproduction

Dataset statistics

Number of variables	12
Number of observations	891
Missing cells	866
Missing cells (%)	8.1%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	83.7 KiB
Average record size in memory	96.1 B

Variable types

CAT	6
NUM	5
BOOL	1

Warunki

Pandas profiling oferuje również wyświetlanie informacji o potencjalnych problemach ze zbiorem danych, takich jak wiele brakujących wartości, wartości jedynie unikalne, czy w większości zerowe.

Warnings

Ticket has a high cardinality: 681 distinct values

High cardinality

Cabin has a high cardinality: 147 distinct values

High cardinality

Age has 177 (19.9%) missing values

Missing

Cabin has 687 (77.1%) missing values

Missing

Ticket is uniformly distributed

Uniform

Cabin is uniformly distributed

Uniform

PassengerId has unique values

Unique

Name has unique values

Unique

SibSp has 608 (68.2%) zeros

Zeros



Kolumny

Każda kolumna analizowana jest tutaj osobno, po prawej wyświetlany jest histogram określający rozkład wartości w danej kolumnie, na środku podstawowe statystyki, a po lewej ewentualne ostrzeżenia. W celu wyświetlenia większej ilości informacji należy kliknąć w 'toggle details'

Variables

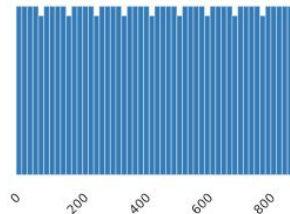
PassengerId

Real number ($\mathbb{R}_{\geq 0}$)

UNIQUE

Distinct	891
Distinct (%)	100.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	446
Minimum	1
Maximum	891
Zeros	0
Zeros (%)	0.0%
Memory size	7.0 KiB



Toggle details

Szczegóły

Szczegóły zawierają praktycznie wszystkie interesujące informacje o danych, a oprócz statystyk możliwe jest również podejrzenie najczęściej występujących wartości czy wartości ekstremalnych.

Statistics	Histogram	Common values	Extreme values
Quantile statistics		Descriptive statistics	
Minimum	1	Standard deviation	257.353842
5-th percentile	45.5	Coefficient of variation (CV)	0.5770265516
Q1	223.5	Kurtosis	-1.2
median	446	Mean	446
Q3	668.5	Median Absolute Deviation (MAD)	223
95-th percentile	846.5	Skewness	0
Maximum	891	Sum	397386
Range	890	Variance	66231
Interquartile range (IQR)	445	Monotocity	Strictly increasing

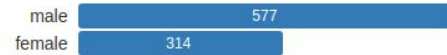
Wartości kategoryczne

Pandas profiling radzi sobie również z różnymi rodzajami zmiennych, zarówno ze zmiennymi liczbowymi, jak i kategorycznymi co widać poniżej:

Sex

Categorical

Distinct	2
Distinct (%)	0.2%
Missing	0
Missing (%)	0.0%
Memory size	7.0 KiB

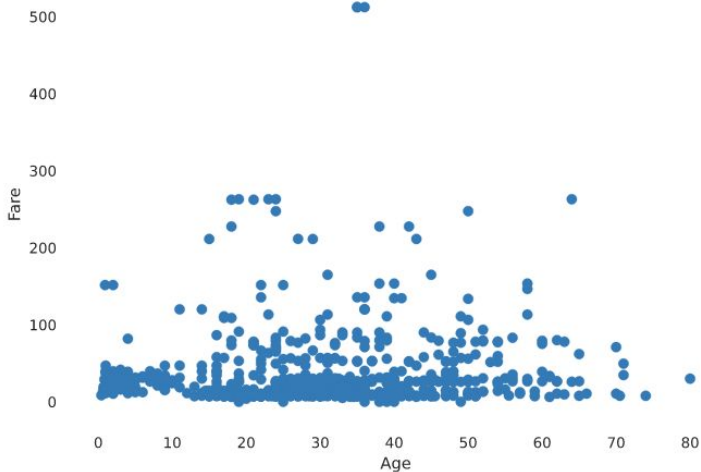


Toggle details



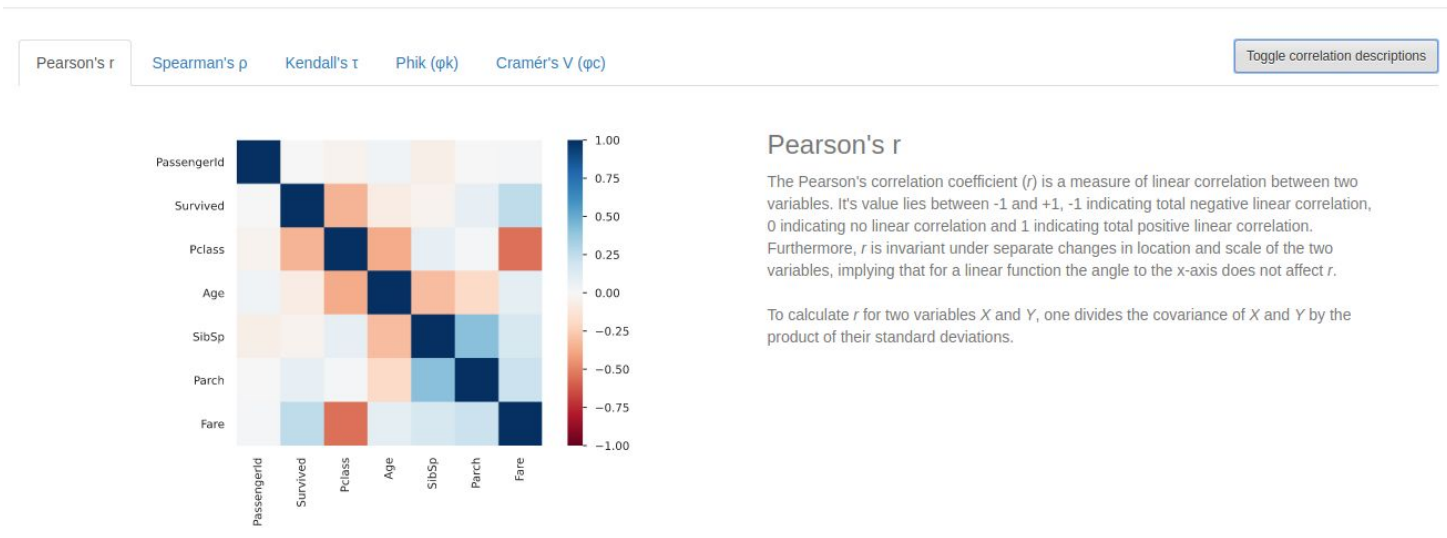
Interakcje

W dalszej części raportu można odnaleźć wykresy określające zależności pomiędzy poszczególnymi wartościami liczbowymi. Dzięki nim można w łatwy sposób zauważyć pewne zjawiska. Jak np. na przykładzie znajdowały się dwie osoby, których bilet był prawie dwa razy droższy niż pozostałej części pasażerów



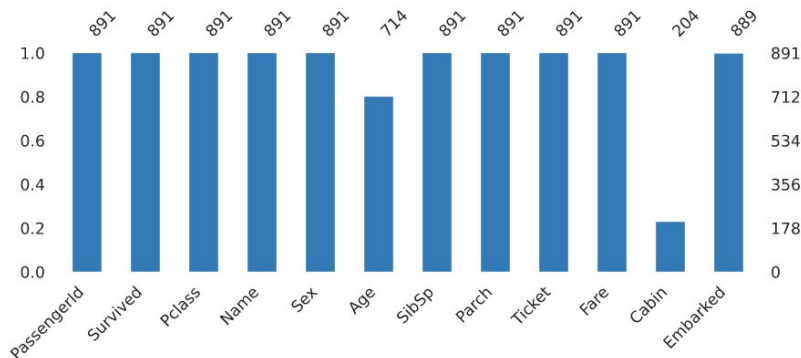
Korelacje

W dalszej części znajdują się wszelkie możliwe korelacje, wraz z dostępnym opisem (po kliknięciu w 'toggle correlation descriptions'. Od razu możemy zauważyć, że opłata za bilet koreluje z informacją o tym, że ktoś przeżył.



Brakujące wartości i “próbka”

Na samym końcu znajdują się informacje o brakujących wartościach, oraz poglądowa próbka danych w postaci tabelarycznej.



First rows

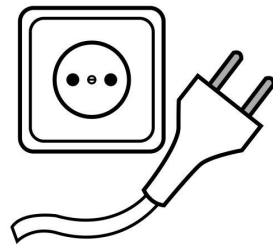
	PassengerId	Survived	Pclass	Name	Sex	Age
0	1	0	3	Braund, Mr. Owen Harris	male	22.0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38.0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
4	5	0	3	Allen, Mr. William Henry	male	35.0
5	6	0	3	Moran, Mr. James	male	NaN
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0

API – łączność między aplikacjami



Czym jest API?

Skrót **API (Application Programming Interface)** oznacza interfejs przeznaczony dla programistów tworzących aplikacje. Większość dużych firm tworzy takie interfejsy dla swoich klientów, a także do użytku wewnętrznego. Zapewne korzystałeś z API, nawet o tym nie wiedząc, na przykład komentując post na Facebooku na stronie internetowej innej firmy, płacąc za modne różowe buty w ekskluzywnym sklepie internetowym przy użyciu konta PayPal, albo nawet używając wyszukiwarki na stronie sklepu lub korzystając z Google Maps, aby sprawdzić jego lokalizację.





Co można pobierać z API?

API można używać przede wszystkim do pobierania aktualnych i historycznych informacji, na przykład o:

- pogodzie
- natężeniu ruchu
- lotach
- najnowszych wiadomościach
- twittach
- mapach
- filmach
- wiele, wiele innych

Trzeba pamiętać o tym, że większość dobrych API jest płatna, ewentualnie posiadają darmową wersję testową/demo. Zawsze musimy posiadać wygenerowany dla nas klucz API.



Prosty przykład – dane pogodowe



Użyjemy serwisu OpenWeatherAPI: <https://openweathermap.org>

Aby uzyskać klucz trzeba się zarejestrować – klucz powinien przyjść na maila.

W ramach darmowej wersji możemy wykonywać 60 zapytań na minutę, sprawdzać aktualną pogodę oraz jej prognozy do 7 dni dla całego świata.

Dane są zwracane jako JSON lub XML.

Do API jest załączona bardzo dobra dokumentacja:

<https://openweathermap.org/current>



JSON?

JSON (ang. JavaScript Object Notation) to otwarty format zapisu struktur danych. Jego przeznaczeniem jest najczęściej wymiana danych pomiędzy aplikacjami. JSON składa się z par atrybut – wartość oraz typów danych tablicowych. Notacja JSONa jest zbieżna z obiektami w języku JavaScript.

Jego zaletą jest popularność, prostota działania, zwięzłość syntaktyki, a jako że dane są zapisywane do tekstu – po sformatowaniu czytelne dla ludzi. JSON może być alternatywą dla XML lub CSV. Pliki JSON zapisujemy z rozszerzeniem .json. Logo JSON to torus Mobiusa.

```
{  
  "title" : "This Is What You Came For",  
  "artist" : "Calvin Harris",  
  "length" : "3:41",  
  "released" : "2016.04.29"  
}
```

XML?

XML (ang. Extensible Markup Language, w wolnym tłumaczeniu Rozszerzalny Język Znaczników) – uniwersalny język znaczników przeznaczony do reprezentowania różnych danych w strukturalizowany sposób. Jest niezależny od platformy, co umożliwia łatwą wymianę dokumentów pomiędzy heterogenicznymi systemami i znacząco przyczyniło się do popularności tego języka w dobie Internetu.

```
<?xml version="1.0" encoding="iso-8859-8" standalone="yes" ?>
<CURRENCIES>
  <LAST_UPDATE>2004-07-29</LAST_UPDATE>
  <CURRENCY>
    <NAME>dollar</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>USD</CURRENCYCODE>
    <COUNTRY>USA</COUNTRY>
    <RATE>4.527</RATE>
    <CHANGE>0.044</CHANGE>
  </CURRENCY>
  <CURRENCY>
    <NAME>euro</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>EUR</CURRENCYCODE>
    <COUNTRY>European Monetary Union</COUNTRY>
    <RATE>5.4417</RATE>
    <CHANGE>-0.013</CHANGE>
  </CURRENCY>
</CURRENCIES>
```




Prosty przykład – dane pogodowe



```
print(response) #powoduje wyświetlenie statusu
```

```
# wyświetli status i ewentualnie kod błędu
```

```
print(f"Request returned {response.status_code} : '{response.reason}')
```

```
# odczytanie danych z odpowiedzi (parsowanie jako json)
```

```
payload = response.json()
```

```
<Response [200]>
```

```
Żądanie zwróciło 200 : 'OK'
```



Prosty przykład – dane pogodowe

```
print(response) #wyświetli status

# wyświetlanie statusu i ewentualnego kodu błędu
print(f"Request returned {response.status_code} : '{response.reason}'")

# odczytywanie danych z odpowiedzi (parsowanie jako json)
payload = response.json()

<Response [200]>
Request returned 200 : 'OK'
```

Prosty przykład – dane pogodowe

Wyświetlanie danych

```
print(payload)
```

```
{'coord': {'lon': 17.03, 'lat': 51.1}, 'weather': [{'id': 803, 'main': 'Clouds', 'description':
```

Ładniejszy format – biblioteka pprint

Wyświetlanie kluczy: payload.keys()


```
#ładniejsze formatowanie
```

```
import pprint
```


```
pp = pprint.PrettyPrinter(indent=1)
```

```
pp.pprint(payload)
```

```
{'base': 'stations',  
 'clouds': {'all': 75},  
 'cod': 200,  
 'coord': {'lat': 51.1, 'lon': 17.03},  
 'dt': 1602757964,  
 'id': 3081368,  
 'main': {'feels_like': 7.48,
```



Prosty przykład – dane pogodowe



Wyświetlanie danych historycznych w ramach darmowego API:
<https://openweathermap.org/api/one-call-api#history>

Prognoza pogody:
<https://openweathermap.org/api/one-call-api>



Prosty przykład – dane pogodowe

Zapisywanie danych z API do dataframe'u:

```
import pandas as pd
df = pd.DataFrame(columns=['datetime', 'temp', 'pressure'])
for i in range(len(payload['hourly'])):
    df.loc[i]=[pd.to_datetime(payload['hourly'][i]['dt'],unit='s'), payload['hourly'][i]['temp'], payload['hourly'][i]['pressure']]
print(df)
```

	datetime	temp	pressure
0	2020-10-15 13:00:00	19.72	1011
1	2020-10-15 14:00:00	19.68	1012
2	2020-10-15 15:00:00	19.85	1013
3	2020-10-15 16:00:00	18.55	1014
4	2020-10-15 17:00:00	16.19	1015
5	2020-10-15 18:00:00	15.86	1015
6	2020-10-15 19:00:00	15.96	1016
7	2020-10-15 20:00:00	15.72	1016



Prosty przykład – dane pogodowe

Zapis do pliku i pobranie z Google Colabulatory:

```
from google.colab import files  
  
df.to_csv('pogoda.csv')  
files.download('pogoda.csv')
```





Może ciekawiej – dane giełdowe



Można korzystać z wielu dostępnych API. Ja polecam yahoo-fin:
http://theautomatic.net/yahoo_fin-documentation/

```
!pip install yahoo-fin  
!pip install requests-html
```

```
import yahoo_fin.stock_info as yh
```

Może ciekawiej – dane giełdowe

Największe wzrosty w danym dniu:

```
response = yh.get_day_gainers()
pp.pprint(response)
```

	Symbol	Name	...	Market Cap	PE Ratio (TTM)
0	CIT	CIT Group Inc.	...	2.386000e+09	NaN
1	NAV	Navistar International Corporation	...	4.268000e+09	NaN
2	FLEX	Flex Ltd.	...	7.220000e+09	80.98
3	RAZFF	Razer Inc.	...	2.778000e+09	NaN
4	BMI	Badger Meter, Inc.	...	2.341000e+09	50.89
..
64	CGNX	Cognex Corporation	...	1.200300e+10	86.71
65	MCRB	Seres Therapeutics, Inc.	...	3.000000e+09	NaN
66	VLVLY	AB Volvo (publ)	...	4.190200e+10	13.54
67	ACAD	ACADIA Pharmaceuticals Inc.	...	6.991000e+09	NaN
68	CHWY	Chewy, Inc.	...	2.753400e+10	NaN



Może ciekawiej – dane giełdowe



Zadanie: zdobądź za pośrednictwem API dane giełdowe spółki netflix za ostatnie 30 dni. Policz średnią cenę otwarcia, średnią z najwyższych i najniższych cen dnia oraz oblicz procentowy wzrost (lub spadek) akcji spółki w ciągu miesiąca.

Może ciekawiej – dane giełdowe

Odpowiedź:

```
response = yh.get_data('nflx', start_date = '01/09/2020', end_date = '30/09/2020', index_as_date = True, interval = '1d')
print("Średnia wartość otwarcia: ", response.open.mean())
print("Średnia wartość najwyższa dobową: ", response.high.mean())
print("Średnia wartość najniższa dobową: ", response.low.mean())
print("Procentowy wzrost akcji spółki: ", (response.open[-1]-response.open[0])/response.open[0] * 100, "%")
```

```
Średnia wartość otwarcia: 427.887157836247
Średnia wartość najwyższa dobową: 436.24497277098277
Średnia wartość najniższa dobową: 419.83683168171535
Procentowy wzrost akcji spółki: 43.12865497076023 %
```



Dziękujemy!

