Laboratorium Analiza i bazy danych

Łączenie tabel, podzapytania i funkcje agregujące

Przykładowe tabele obrazujące łączenie

Do zobrazowania operacji łączenia zostaną użyte tabele:

```
CREATE TABLE shape_a (
    id INT PRIMARY KEY,
    shape VARCHAR (100) NOT NULL
);

CREATE TABLE shape_b (
    id INT PRIMARY KEY,
    shape VARCHAR (100) NOT NULL
);
```

Polecenie CREATE TABLE tworzy tabelę o zadanej nazwie i strukturze. Ogólna postać to:

```
CREATE TABLE tab_name (
    col_name1 data_type constrain,
    col_name1 data_type constrain,
    ...
);
```

Należy uzupełnić ją danymi:

```
INSERT INTO shape_a (id, shape)
VALUES
    (1, 'Trójkąt'),
    (2, 'Kwadrat'),
    (3, 'Deltoid'),
    (4, 'Traper');

INSERT INTO shape_b (id, shape)
VALUES
    (1, 'Kwadrat'),
    (2, 'Trójkąt'),
    (3, 'Romb'),
    (4, 'Równoległobok');
```

Komenda INSERT INTO pozwala na dodanie do tabeli rekordów. Ogólna postać to:

```
INSERT INTO tab_name (col1_name, col2_name2, ...)
VALUES
    (val1_col1, val2_col2),
    (val2_col1, val2_col2),
    ...
```

Inner join

Jest to podstawowy rodzaj złączenie. Ten sposób złączenia wybiera te wiersze, dla których warunek złączenia jest spełniony. W żadnej z łączonych tabel kolumna użyta do łączenia nie może mieć wartości NULL.

Przykład:

```
SELECT
    a.id id_a,
    a.shape shape_a,
    b.id id_b,
    b.shape shape_b
FROM
    shape_a a
INNER JOIN shape_b b ON a.shape = b.shape;
```

W zapytaniu powyżej użyto *aliasów* nazw tabel i column wynikowych, jest to szczególnie przydatne przy długich nazwach tabel i wprowadza czytelność w zapytaniu.

Wynik:

id_a	shape_a	id_b	shape_b
1	Trójkąt	2	Trójkąt
2	Kwadrat	1	Kwadrat

OUTER JOIN

Istnieją trzy rodzaje złączeń OUTER:

- LEFT OUTER JOIN,
- RIGHT OUTER JOIN,
- FULL OUTER JOIN.

LEFT OUTER JOIN

Ten rodzaj złączenie zwróci wszystkie rekordy z lewej tablicy i dopasuje do nich rekordy z prawej tablicy które spełniją zadany warunek złączenia. Jeżeli w prawej tablicy nie występują rekordy spełnijące warunek złączenia z lewą w ich miejscu pojawią się wartości NULL.

Przykład 1:

```
SELECT
    a.id id_a,
    a.shape shape_a,
    b.id id_b,
    b.shape shape_b
FROM
    shape_a a
LEFT JOIN shape_b b ON a.shape = b.shape;
```

Wynik:

id_a	shape_a	id_b	shape_b
1	Trójkąt	2	Trójkąt
2	Kwadrat	1	Kwadrat
3	Deltoid	NULL	NULL
4	Traper	NULL	NULL

Przykład 2:

```
SELECT
    b.id id_b,
    b.shape shape_b,
    a.id id_a,
    a.shape shape_a
FROM
    shape_b b
LEFT JOIN shape_a a ON a.shape = b.shape;
```

Wynik:

id_a	shape_a	id_b	shape_b
1	Kwadrat	2	Kwadrat
2	Trójkąt	1	Trójkąt
3	Romb	NULL	NULL

id_a	shape_a	id_b	shape_b
4	Równoległobok	NULL	NULL

RIGHT OUTER JOIN

Działa jak left outer join z tym, że prawa tablica w zapytaniu jest brana w całości.

Przykład:

```
SELECT
    a.id id_a,
    a.shape shape_a,
    b.id id_b,
    b.shape shape_b
FROM
    shape_a a
RIGHT JOIN shape_b b ON a.shape = b.shape;
```

Wynik:

id_a	shape_a	id_b	shape_b
2	Kwadrat	1	Kwadrat
1	Trójkąt	2	Trójkąt
NULL	NULL	3	Romb
NULL	NULL	4	Równoległobok

FULL OUTER JOIN

Jest złączeniem które zwraca:

- wiersze dla których warunek złączenia jest spełniony,
- wiersze z lewej tabeli dla których nie ma odpowiedników w prawej,
- wiersze z prawej tabeli dla których nie ma odpowiedników w lewej.

Przykład:

```
SELECT
    a.id id_a,
    a.shape shape_a,
    b.id id_b,
    b.shape shape_b
FROM
```

```
    shape_a a
FULL JOIN shape_b b ON a.shape = b.shape;
```

id_a	shape_a	id_b	shape_b
1	Trójkąt	2	Trójkąt
2	Kwadrat	1	Kwadrat
3	Deltoid"	NULL	NULL
4	Traper	NULL	NULL
NULL	NULL	3	Romb
NULL	NULL	4	Równoległobok

Podzapytania

Podzapytanie zagnieżdżone SELECT znajduje się wewnątrz zewnętrznego zapytania SELECT, np. po klauzuli WHERE, HAVING lub FROM. W przypadku tego rodzaju zapytań w pierwszej kolejności wykonywane są wewnętrzne zapytania SELECT, a ich wynik jest wykorzystywany do zewnętrznego zapytania SELECT. Stąd łatwo zuważyć, że mogą one służyć do poprawy wydajności obsługi zapytania. Należy dobierać podzapytania tak by najbardziej zagnieżdżone podzapytanie zawierało najmniejszy zbiór poszukiwań.

Przykład:

Jeżeli chcemy znaleźć w bazie informację o tytułach filmów zwróconych w zadanym okresie możemy wykonać następujące zapytanie:

```
SELECT
   film_id,
   title
FROM
   film
WHERE
   film_id IN (
      SELECT
         inventory.film_id
      FROM
         rental
      INNER JOIN inventory ON inventory.inventory_id = rental.inventory_id
      WHERE
         return_date BETWEEN '2005-05-29'
      AND '2005-05-30'
   );
```

Wynik

film_id	title
307	Fellowship Autumn
255	Driving Polish
388	Gunfight Moon
130	Celebrity Horn
563	Massacre Usual
397	Hanky October
...	...

Używanie podzapytań

Pod zapytania mogą być używane w :

- SELECT,
- UPDATE,
- DELETE,
- Funkcjach agregujących,
- Do definiowania tabel tymczasowych.

Używając podzapytań zapytania SQL szybko mogą stać się mało czytelne. Przez co będą trudne w zrozumieniu i późniejszym utrzymaniu. W celu analizy zapytań można użyć klauzuli **EXPLAIN**, która przeanalizuje zapytanie. Klauzula ta może służyć również do porównywania wydajności zapytań

Przykład:

```
EXPLAIN SELECT
   *
FROM
   film
```

Funkcje agregujące

Funkcje agregujące wykonują obliczenia na zestawie wierszy i zwracają pojedynczy wiersz. PostgreSQL udostępnia wszystkie standardowe funkcje agregujące SQL w następujący sposób:

- AVG () - zwraca średnią wartość.

- COUNT () - zwraca liczbę wartości.
- MAX () - zwraca maksymalną wartość.
- MIN () - zwraca minimalną wartość.
- SUM () - zwraca sumę wszystkich lub różnych wartości.

Pełna lista funkcji agregującej: https://www.postgresql.org/docs/9.5/functions-aggregate.html

Często używamy funkcji agregujących z klauzulą GROUP BY w instrukcji SELECT. W tych przypadkach klauzula GROUP BY dzieli zestaw wyników na grupy wierszy i funkcja agregująca wykonuje obliczenia dla każdej grupy, np. maksimum, minimum, średnia itp. Funkcji agregujących można używać funkcji agregujących jako wyrażeń tylko w następujących klauzulach: SELECT i HAVING.

GROUP BY

Klauzula GROUP BY dzieli wiersze zwrócone z instrukcji SELECT na grupy. Dla każdej grupy można zastosować funkcję agregującą, np. SUM aby obliczyć sumę pozycji lub COUNT aby uzyskać liczbę elementów w grupach.

Poniższa instrukcja ilustruje składnię klauzuli GROUP BY:

```
SELECT
    column_1,
    aggregate_function(column_2)
FROM
    tbl_name
GROUP BY
    column_1;
```

Klauzula GROUP BY musi pojawić się zaraz po klauzuli FROM lub WHERE, n0astępnie GROUP BY zawiera listę kolumna oddzielonych przecinkami.

HAVING

Często używamy klauzuli HAVING w połączeniu z klauzulą GROUP BY do filtrowania wierszy grup które nie spełniają określonego warunku.

Poniższa instrukcja ilustruje typową składnię klauzuli HAVING:

```
SELECT
    column_1,
    aggregate_function (column_2)
FROM
    tbl_name
GROUP BY
    column_1
```

```
HAVING
    condition;
```

Klauzula HAVING ustawia warunek dla wierszy grup utworzonych przez klauzulę GROUP BY.

Klauzula GROUP BY ma zastosowanie, podczas gdy klauzula WHERE określa wcześniej warunki dla poszczególnych wierszy.

Zadania wprowadzające

Wykonaj zapytania przy użyciu DBMS:

1. Znajdź listę wszystkich filmów o tej samej długości.
2. Znajdź wszystkich klientów mieszkających w tym samym mieście.
3. Oblicz średni koszt wypożyczenia wszystkich filmów.
4. Oblicz i wyświetl liczbę filmów we wszystkich kategoriach.
5. Wyświetl liczbę wszystkich klientów pogrupowanych według kraju.
6. Wyświetl informacje o sklepie, który ma więcej niż 100 klientów i mniej niż 300 klientów.
7. Wybierz wszystkich klientów, którzy oglądali filmy ponad 200 godzin.
8. Oblicz średnią wartość wypożyczenia filmu.
9. Oblicz średnią wartość długości filmu we wszystkich kategoriach.
10. Znajdź najdłuższe tytuły filmowe we wszystkich kategoriach.
11. Znajdź najdłuższy film we wszystkich kategoriach. Porównaj wynik z pkt 10.

Zadanie implementacyjne

Zaimplementuj wszystkie funkcje w pliku main.py zgodnie z opisem a następnie przetestuj je w notatniku.

```
28 import main
   import pandas as pd
   import psycopg2 as pg
   connection = pg.connect(host='pgsql-196447.vipserv.org', port=5432, dbname='wbauer_adb', user='wbauer
                           password='adb2020');
```

1. Znajdź listę wszystkich filmów o tej samej długości.

```
17 film_list = pd.read_sql('''select
                              f.title , f.length,
                              f2.title, f2.length
                              from film f
                              INNER JOIN film f2 using(length)
                              where f.title != f2.title
```

```
                            ''' ,con = connection)
film_list
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/2115749182.py:1: UserWarning: pandas only suppor
  film_list = pd.read_sql('''select
```

17

	title	length	title	length
0	Chamber Italian	117	Resurrection Silverado	117
1	Chamber Italian	117	Magic Mallrats	117
2	Chamber Italian	117	Graffiti Love	117
3	Chamber Italian	117	Affair Prejudice	117
4	Grosse Wonderful	49	Hurricane Affair	49
...	...	...	...	...
6967	Zorro Ark	50	Muppet Mile	50
6968	Zorro Ark	50	Lion Uncut	50
6969	Zorro Ark	50	Crossing Divorce	50
6970	Zorro Ark	50	Blues Instinct	50
6971	Zorro Ark	50	Adaptation Holes	50

6972 rows × 4 columns

2.Znajdź wszystkich klientów mieszkających w tym samym mieście.

18

```
cust_list = pd.read_sql('''select
                           c.first_name, c.last_name,
                           c2.first_name, c2.last_name,
                           ci.city
                           from customer c
                           INNER JOIN address a using(address_id)
                           INNER JOIN city ci using(city_id)
                           INNER JOIN customer c2 on
                           (select address.city_id from address where address.address_id = c.address_
                           (select address.city_id from address where address.address_id = c2.addres
                           where c.address_id != c2.address_id
                           ''' ,con = connection)
cust_list
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/623510546.py:1: UserWarning: pandas only supports
  cust_list = pd.read_sql('''select
```

18

	first_name	last_name	first_name	last_name	city
0	Scott	Shelley	Clinton	Buford	Aurora
1	Clinton	Buford	Scott	Shelley	Aurora
2	Cecil	Vines	Mattie	Hoffman	London
3	Mattie	Hoffman	Cecil	Vines	London

3.Oblicz średni koszt wypożyczenia wszystkich filmów

19
```
avg_all = pd.read_sql(''' select
                      AVG(p.amount)
                      from film f
                      INNER JOIN inventory i using(film_id)
                      INNER JOIN rental r using(inventory_id)
                      INNER JOIN payment p using(rental_id)
                      ''',con = connection)
avg_all
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/2949189287.py:1: UserWarning: pandas only support
  avg_all = pd.read_sql(''' select
```

19

	avg
0	4.200606

4.Oblicz i wyświetl liczbę filmów we wszystkich kategoriach.

20
```
pd.read_sql(''' select
            cat.name , COUNT(f.film_id)
            from film f
            INNER JOIN film_category fcat using(film_id)
            INNER JOIN category cat using(category_id)
            GROUP BY cat.name
            ''',con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/4077863986.py:1: UserWarning: pandas only support
  pd.read_sql(''' select
```

20

	name	count
0	Sports	74
1	Classics	57
2	New	63
3	Family	69
4	Comedy	58
5	Animation	66
6	Travel	57
7	Music	51
8	Drama	62
9	Horror	56
10	Sci-Fi	61
11	Games	61

	name	count
12	Documentary	68
13	Foreign	73
14	Action	64
15	Children	60

5.Wyświetl liczbę wszystkich klientów pogrupowanych według kraju.

21
```
pd.read_sql(''' select
            cou.country , cas.first_name, cas.last_name
            from customer cas
            INNER JOIN address a using(address_id)
            INNER JOIN city ci using(city_id)
            INNER JOIN country cou using(country_id)
            ORDER BY cou.country ASC
            ''',con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/3372613487.py:1: UserWarning: pandas only suppor
  pd.read_sql(''' select
```

21

	country	first_name	last_name
0	Afghanistan	Vera	Mccoy
1	Algeria	Judy	Gray
2	Algeria	Mario	Cheatham
3	Algeria	June	Carroll
4	American Samoa	Anthony	Schwab
...	...	...	...
594	Yemen	Ella	Oliver
595	Yemen	Gina	Williamson
596	Yugoslavia	Maria	Miller
597	Yugoslavia	Max	Pitt
598	Zambia	Barry	Lovelace

599 rows × 3 columns

6.Wyświetl informacje o sklepie, który ma więcej niż 100 klientów i mniej niż 300 klientów

22
```
pd.read_sql(''' select
            st.store_id , a.address , ci.city  ,COUNT(c.customer_id)
            from store st
            INNER JOIN staff sf using(store_id)
```

```
            INNER JOIN rental r using(staff_id)
            INNER JOIN customer c using(customer_id)
            INNER JOIN address a on a.address_id = st.address_id
            INNER JOIN city ci using(city_id)
            GROUP BY st.store_id, a.address, ci.city
            HAVING COUNT(c.customer_id) BETWEEN 100 AND 300
            ''',con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/179793543.py:1: UserWarning: pandas only supports
  pd.read_sql(''' select
```

22

	store_id	address	city	count

7.Wybierz wszystkich klientów, którzy oglądali filmy ponad 200 godzin.

23

```
pd.read_sql(''' select
            c.customer_id , c.first_name, c.last_name , SUM(f.length)
            from customer c
            INNER JOIN rental r using(customer_id)
            INNER JOIN inventory i using(inventory_id)
            INNER JOIN film f using(film_id)
            GROUP BY c.customer_id , c.first_name, c.last_name
            HAVING SUM(f.length) > 200*60
            ''',con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/1899564983.py:1: UserWarning: pandas only support
  pd.read_sql(''' select
```

23

	customer_id	first_name	last_name	sum

8.Oblicz średnią wartość wypożyczenia filmu.

26

```
pd.read_sql(''' select
            f.title ,  AVG(p.amount)
            from film f
            INNER JOIN inventory i using(film_id)
            INNER JOIN rental r using(inventory_id)
            INNER JOIN payment p using(rental_id)
            GROUP BY f.title
            ''',con = connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/1002897191.py:1: UserWarning: pandas only support
  pd.read_sql(''' select
```

26

	title	avg
0	Graceland Dynamite	6.323333
1	Opus Ice	6.090000
2	Braveheart Human	3.390000
3	Wonderful Drop	5.865000
4	Rush Goodfellas	2.918571
...	...	...

	title	avg
953	Mockingbird Hollywood	2.444545
954	Gathering Calendar	1.990000
955	Drums Dynamite	1.913077
956	Samurai Lion	4.240000
957	Pond Seattle	3.434444

958 rows × 2 columns

9.Oblicz średnią wartość długości filmu we wszystkich kategoriach.

32
```
pd.read_sql(''' select
            cat.name , AVG(f.length)
            from film f
            INNER JOIN film_category fcat using(film_id)
            INNER JOIN category cat using(category_id)
            GROUP BY cat.name

            ''',con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/2894006868.py:1: UserWarning: pandas only support
  pd.read_sql(''' select
```

32

	name	avg
0	Sports	128.202703
1	Classics	111.666667
2	New	111.126984
3	Family	114.782609
4	Comedy	115.827586
5	Animation	111.015152
6	Travel	113.315789
7	Music	113.647059
8	Drama	120.838710
9	Horror	112.482143
10	Sci-Fi	108.196721
11	Games	127.836066
12	Documentary	108.750000
13	Foreign	121.698630
14	Action	111.609375

	name	avg
15	Children	109.800000

10.Znajdź najdłuższe tytuły filmowe we wszystkich kategoriach.

83
```
pd.read_sql(''' select
            cat.name , MAX(LENGTH(f.title))
            from category cat
            INNER JOIN film_category fcat using(category_id)
            INNER JOIN film f using(film_id)
            GROUP BY cat.name

            ''',con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/3757433134.py:1: UserWarning: pandas only support
  pd.read_sql(''' select
```

83

	name	max
0	Sports	25
1	Classics	23
2	New	21
3	Family	22
4	Comedy	23
5	Animation	22
6	Travel	23
7	Music	22
8	Drama	22
9	Horror	27
10	Sci-Fi	20
11	Games	21
12	Documentary	22
13	Foreign	20
14	Action	23
15	Children	20

11.Znajdź najdłuższy film we wszystkich kategoriach. Porównaj wynik z pkt 10.

79
```
pd.read_sql(''' select cat.name, MAX(f.length)
            from film f
```

```
            INNER JOIN film_category fcat using(film_id)
            INNER JOIN category cat using(category_id)
            GROUP BY cat.name

            ''',con=connection)
```

```
C:\Users\SEBAST~1\AppData\Local\Temp/ipykernel_15260/440265915.py:1: UserWarning: pandas only supports
  pd.read_sql(''' select cat.name, MAX(f.length)
```

79

	name	max
0	Sports	184
1	Classics	184
2	New	183
3	Family	184
4	Comedy	185
5	Animation	185
6	Travel	185
7	Music	185
8	Drama	181
9	Horror	181
10	Sci-Fi	185
11	Games	185
12	Documentary	183
13	Foreign	184
14	Action	185
15	Children	178