

a

Materiały do przedmiotu:

Bazy danych

Laboratorium nr 5:

**Pisanie kwerend wykorzystujących podzapytania
nieskorelowane oraz skorelowane w celu uzyskania
specyficznych informacji z bazy danych**

Autor:

mgr Alicja Żmudzińska



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	3
Cel ćwiczenia/laboratorium	3
Instrukcja laboratoryjna/ćwiczeniowa	3
Ćwiczenia samodzielne	8
Pytania pomocnicze	10
Podsumowanie	10
Literatura	11



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

W języku SQL często zachodzi potrzeba uzyskania informacji, które nie wynikają bezpośrednio z jednej tabeli czy prostego zapytania. Zdarza się, że zanim pobierzemy ostateczne dane, musimy najpierw coś policzyć, przefiltrować lub porównać z wynikiem innego zapytania. Właśnie do tego służą podzapytania, nazywane także subqueries. Podzapytanie to po prostu zapytanie umieszczone wewnątrz innego zapytania SQL. Można je potraktować jako „zapytanie w zapytaniu” najpierw wykonywane jest zapytanie wewnętrzne, a jego wynik jest następnie używany przez zapytanie główne. Dzięki temu SQL pozwala uzyskiwać bardzo precyzyjne i złożone informacje, bez konieczności tworzenia dodatkowych tabel czy widoków.

Cel ćwiczenia/laboratorium

Celem ćwiczeń jest poznanie zasad tworzenia oraz stosowania podzapytań w języku SQL, a także zrozumienie różnicy pomiędzy podzapytaniami nieskorelowanymi a skorelowanymi. Podczas pracy z bazą danych Sakila uczestnik zajęć nauczy się w praktyce, jak wykorzystać podzapytania do uzyskania bardziej szczegółowych informacji z relacyjnych zbiorów danych. W trakcie ćwiczeń zostaną utrwalone podstawy składni SQL w zakresie zapytań zagnieżdżonych, a także omówione zostaną sytuacje, w których warto zastosować podzapytanie zamiast tradycyjnego złączenia tabel. W ramach realizacji ćwiczeń uczestnik pozna sposób działania zapytań wewnętrznych oraz mechanizm przekazywania wyników pomiędzy zapytaniem podrzędnym a zapytaniem głównym. Uświadomi sobie, że podzapytania mogą być wykonywane niezależnie lub w ściślejszej korelacji z danymi zapytania nadrzędnego, co pozwala uzyskać dynamiczne i precyzyjne wyniki. Ćwiczenia obejmują również praktyczne zastosowanie operatorów logicznych i porównawczych takich jak IN, EXISTS, NOT EXISTS, ANY czy ALL, które stanowią nieodłączny element pracy z podzapytaniami.

Instrukcja laboratoryjna/ćwiczeniowa

Podzapytanie (ang. *subquery*) to zapytanie SQL umieszczone wewnątrz innego zapytania, które można traktować jako „zapytanie w zapytaniu”. Umożliwia ono uzyskanie dodatkowych informacji na podstawie wyników pośrednich i wykorzystanie tych danych w zapytaniu głównym, co sprawia, że SQL staje się niezwykle elastycznym narzędziem do analizy i przetwarzania danych. Podzapytania mogą pojawiać się w różnych częściach instrukcji SQL. Najczęściej w klauzuli WHERE, gdy chcemy porównać dane z innymi wartościami; w klauzuli HAVING, gdy filtrujemy grupy po wykonaniu agregacji; w klauzuli FROM, gdy traktujemy wynik innego zapytania jako tymczasową tabelę; oraz w klauzuli SELECT, gdy chcemy, aby wynik podzapytania został wykorzystany jako dodatkowa kolumna.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Podzapytania mogą przyjmować różne formy w zależności od sposobu, w jaki są powiązane z zapytaniem głównym. W praktyce oznacza to, że niektóre z nich działają niezależnie, a inne muszą odwoływać się do danych z zapytania nadrzędnego. Na tej podstawie można wyróżnić dwa główne typy podzapytań: nieskorelowane i skorelowane.

Podzapytania nieskorelowane to takie, które może być wykonane samodzielnie, niezależnie od zapytania głównego. Najpierw wykonywane jest podzapytanie, jego wynik jest zapamiętywany, a następnie używany przez zapytanie nadrzędne. Tego typu zapytania są zazwyczaj bardziej wydajne, ponieważ SQL musi wykonać je tylko raz.

Przykład 5.1: Wyświetlenie filmów dłuższy niż średnia długość wszystkich filmów

```
SELECT title, length
FROM sakila.film
WHERE length > (SELECT AVG(length) FROM sakila.film);
```

Opis działania:

1. Podzapytanie (SELECT AVG(length) FROM sakila.film) oblicza średnią długość wszystkich filmów w bazie.
2. Zapytanie główne pobiera tytuły i długości tylko tych filmów, które są dłuższe niż ta średnia.

To bardzo często spotykany wzorzec – porównanie każdej wartości z wartością globalną (np. średnią, maksymalną lub minimalną).

Przykład 5.2: Wyświetlenie klientów, którzy wypożyczyli jakikolwiek film

```
SELECT first_name, last_name
FROM sakila.customer
WHERE customer_id IN (SELECT DISTINCT customer_id FROM sakila.rental);
```

Opis działania:

Podzapytanie wewnętrzne zwraca identyfikatory klientów, którzy dokonali co najmniej jednego wypożyczenia. Zapytanie główne wybiera imiona i nazwiska tych klientów z tabeli customer. To przykład użycia podzapytania z operatorem IN, który sprawdza, czy dana wartość znajduje się w zbiorze wyników podzapytania.

Podzapytania skorelowane nie może zostać wykonane samodzielnie, ponieważ jego wynik zależy od bieżącego wiersza zapytania głównego. W praktyce oznacza to, że dla każdego wiersza z zapytania nadrzędnego SQL ponownie uruchamia podzapytanie, dlatego takie konstrukcje mogą być wolniejsze, ale dają ogromne możliwości analizy danych.

Przykład 5.3: Wyświetlenie filmów droższych niż średnia cena w ich kategorii

```
SELECT f.title, f.rental_rate, c.name AS category
FROM sakila.film f
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



```
JOIN sakila.film_category fc ON f.film_id = fc.film_id
JOIN sakila.category c ON fc.category_id = c.category_id
WHERE f.rental_rate > (
    SELECT AVG(f2.rental_rate)
    FROM sakila.film f2
    JOIN sakila.film_category fc2 ON f2.film_id = fc2.film_id
    WHERE fc2.category_id = fc.category_id
);
```

Opis działania:

- Dla każdego filmu z zapytania głównego SQL oblicza średnią stawkę wypożyczenia w danej kategorii.
- Następnie zwraca tylko te filmy, których cena wypożyczenia jest wyższa od tej średniej.

To klasyczny przykład podzapytania skorelowanego, ponieważ w podzapytaniu odwołujemy się do `fc.category_id` – kolumny z zapytania nadrzędnego.

Przykład 5.4: Wyświetlenie klientów, którzy wypożyczyli więcej filmów niż wynosi średnia

```
SELECT c.first_name, c.last_name
FROM sakila.customer c
WHERE (
    SELECT COUNT(*)
    FROM sakila.rental r
    WHERE r.customer_id = c.customer_id
) > (
    SELECT AVG(rental_count)
    FROM (
        SELECT COUNT(*) AS rental_count
        FROM sakila.rental
        GROUP BY customer_id
    ) AS sub
);
```

Opis działania:

1. Wewnętrzne podzapytanie oblicza średnią liczbę wypożyczeń przypadającą na jednego klienta.
2. Drugie podzapytanie (skorelowane) liczy, ile razy dany klient wypożyczył film.
3. Do wyniku trafiają tylko ci klienci, którzy wypożyczyli więcej filmów niż średnia.

Podzapytania mogą być wykorzystywane w różnych częściach zapytania SQL:



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Miejsce użycia	Przykład	Zastosowanie
WHERE	WHERE salary > (SELECT AVG(salary) FROM employees)	porównanie wartości z wynikiem podzapytania
HAVING	HAVING COUNT(*) > (SELECT AVG(order_count) FROM ...)	filtrowanie grup po agregacji
FROM	SELECT * FROM (SELECT ...) AS sub	użycie podzapytania jako tymczasowej tabeli
SELECT	SELECT name, (SELECT COUNT(*) FROM orders WHERE ...) AS liczba_zamowien	wstawienie wyniku podzapytania jako kolumny

W podzapytaniach można wykorzystywać różne operatory, które pozwalają na elastyczne porównywanie i filtrowanie danych. Operator IN (oraz jego przeciwieństwo NOT IN) sprawdza, czy dana wartość znajduje się w zbiorze wyników zwróconym przez podzapytanie. Z kolei EXISTS i NOT EXISTS zwracają wartość logiczną TRUE lub FALSE w zależności od tego, czy podzapytanie zwróciło jakiegokolwiek wiersze. Operatory ANY, SOME oraz ALL umożliwiają porównywanie wartości z wieloma elementami jednocześnie, co daje większą kontrolę nad warunkami logicznymi. Natomiast standardowe operatory porównania, takie jak =, < czy >, są stosowane wtedy, gdy podzapytanie zwraca pojedynczą wartość, czyli ma charakter skalarny.

Przykład 5.5. Wyświetlenie filmów dłuższych niż wszystkie filmy z kategorii „Action”.

```
SELECT title, length
FROM sakila.film
WHERE length > ALL (
    SELECT length
    FROM sakila.film f2
    JOIN sakila.film_category fc2 ON f2.film_id = fc2.film_id
    JOIN sakila.category c2 ON fc2.category_id = c2.category_id
    WHERE c2.name = 'Action'
);
```

Opis działania:

1. Podzapytanie wewnętrzne wybiera długości wszystkich filmów należących do kategorii „Action”.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



2. Główne zapytanie pobiera tytuły i długości filmów z tabeli film.
3. Warunek WHERE length > ALL (...) sprawia, że do wyniku trafiają tylko te filmy, które są dłuższe niż **wszystkie** filmy z kategorii „Action”, czyli mają większą długość niż najdłuższy film akcji.

Różnice między podzapytaniami skorelowanymi a nieskorelowanymi:

Cecha	Podzapytanie nieskorelowane	Podzapytanie skorelowane
Zależność od zapytania głównego	Nie	Tak
Liczba wykonań	Jednorazowo	Dla każdego wiersza
Wydajność	Zazwyczaj wyższa	Może być niższa
Można uruchomić samodzielnie	Tak	Nie
Typowe zastosowanie	Porównanie do wartości globalnej	Analiza w kontekście danego wiersza

Podczas tworzenia podzapytań warto stosować kilka dobrych praktyk, które poprawiają wydajność i czytelność kodu SQL. Przede wszystkim zaleca się używanie operatora EXISTS zamiast IN przy porównywaniu dużych zbiorów danych, ponieważ działa on szybciej i jest odporny na błędy wynikające z wartości NULL. Należy również unikać nadmiernego zagnieżdżania wielu poziomów podzapytań – w takich sytuacjach lepszym rozwiązaniem jest skorzystanie z tymczasowych tabel lub instrukcji WITH, które ułatwiają zarządzanie kodem. Dobrą praktyką jest także nadawanie aliasów podzapytaniom umieszczonym w klauzuli FROM, co znacznie poprawia czytelność zapytań. Warto zwracać uwagę na indeksy, szczególnie w kolumnach wykorzystywanych w warunkach korelacji, ponieważ mają one duży wpływ na czas wykonania zapytania. Dodatkowo, korzystanie z polecenia EXPLAIN pozwala przeanalizować plan wykonania zapytania i lepiej zrozumieć sposób, w jaki optymalizator bazy danych przetwarza poszczególne operacje.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Ćwiczenia samodzielne

1. Filmy dłuższe niż średnia długość wszystkich filmów.
Wskazówka: użyj AVG(length) w podzapytaniu w klauzuli WHERE.
2. Filmy droższe niż średnia cena wypożyczenia (rental_rate) wszystkich filmów.
3. Filmy, które kosztują tyle samo co najdroższy film.
Wskazówka: użyj MAX(rental_rate).
4. Filmy, które mają długość równą minimalnej długości filmu w bazie.
Wskazówka: MIN(length).
5. Wyświetl tytuły filmów, które zostały wypożyczone przez klientów z Polski.
Wskazówka: użyj IN i połącz tabele country, city, address, customer.
6. Wyświetl nazwiska aktorów, którzy grali w filmach o długości większej niż średnia długość wszystkich filmów.
7. Znajdź klientów, którzy mają taką samą nazwę miasta jak któryś z pracowników (staff).
8. Wyświetl nazwy kategorii, które mają więcej filmów niż średnia liczba filmów przypadająca na kategorię.
Wskazówka: użyj GROUP BY + HAVING + podzapytania.
9. Wyświetl filmy, których długość jest większa niż średnia długość filmów o tej samej ocenie (rating).
10. Znajdź wszystkich klientów, którzy zapłacili kwotę większą niż średnia wszystkich płatności w tabeli payment.
11. Znajdź klientów, którzy wypożyczyli jakikolwiek film.
Wskazówka: WHERE customer_id IN (SELECT customer_id FROM rental).
11. Znajdź klientów, którzy nigdy nic nie wypożyczyli.
Wskazówka: użyj NOT EXISTS.
12. Wyświetl filmy, które nigdy nie zostały wypożyczone.
Tabele: film, inventory, rental.
13. Znajdź aktorów, którzy grali w przynajmniej jednym filmie z kategorii „Comedy”.
14. Znajdź aktorów, którzy nie wystąpili w żadnym filmie z kategorii „Horror”.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



15. Wyświetl kraje, w których istnieje przynajmniej jeden klient (EXISTS).
16. Znajdź klientów, którzy dokonali przynajmniej jednej płatności większej niż 8.00 USD.
Użyj EXISTS i warunku w podzapytaniu.
17. Znajdź klientów, którzy wypożyczyli wszystkie filmy, które są w kategorii „Action”.
Wskazówka: użyj ALL (lub zagnieżdżonych NOT EXISTS).
18. Wyświetl filmy, które są dłuższe niż jakikolwiek film z kategorii „Family”.
Operator: > ANY.
19. Wyświetl filmy, które są dłuższe niż wszystkie filmy z kategorii „Music”.
Operator: > ALL.
21. Dla każdego klienta wyświetl jego imię, nazwisko i liczbę wypożyczeń.
*Wskazówka: użyj podzapytania w klauzuli SELECT z korelacją r.customer_id = c.customer_id.**
22. Wyświetl klientów, którzy mają więcej wypożyczeń niż średnia liczba wypożyczeń na klienta.
Zagnieżdżenie dwóch poziomów: jedno podzapytanie do liczenia średniej, drugie skorelowane do liczenia wypożyczeń.
23. Znajdź klientów, którzy zapłacili łącznie więcej niż średnia suma płatności wszystkich klientów.
24. Wyświetl tytuły filmów, które zostały wypożyczone więcej razy niż średnia liczba wypożyczeń na film.
25. Znajdź klientów, którzy dokonali swojej ostatniej płatności po dacie ostatniej płatności pracownika o ID = 1.
26. Wyświetl filmy, których stawka wypożyczenia (rental_rate) jest wyższa niż średnia w ich kategorii.
Korelacja przez category_id.
27. Wyświetl klientów, których średnia wartość płatności jest większa niż średnia globalna.
28. Dla każdego aktora wyświetl jego imię, nazwisko i liczbę filmów, w których zagrał.
Podzapytanie w kolumnie SELECT.
29. Wyświetl klientów, którzy mają aktywne wypożyczenia (czyli nie oddali jeszcze filmu).
Użyj EXISTS z warunkiem return_date IS NULL.
30. Znajdź klientów, którzy wypożyczali tylko filmy dłuższe niż 100 minut.
Skorelowane podzapytanie z warunkiem „nie istnieje film krótszy niż 100 minut wypożyczony przez tego klienta”.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Pytania pomocnicze

1. Czym jest podzapytanie i w jakim celu stosuje się je w języku SQL?
2. Na czym polega różnica pomiędzy podzapytaniem nieskorelowanym a skorelowanym?
3. Jak przebiega wykonanie zapytania skorelowanego – czy podzapytanie uruchamiane jest raz, czy dla każdego wiersza osobno?
4. W jakich sytuacjach bardziej odpowiednie będzie użycie podzapytania, a w jakich złączenia tabel (JOIN)?
5. Jakie znaczenie mają operatory IN, EXISTS, ANY i ALL w kontekście podzapytań?
6. Co się stanie, jeśli podzapytanie zwróci więcej niż jeden wiersz, a zapytanie główne oczekuje tylko jednej wartości?
7. Dlaczego operator NOT EXISTS jest w niektórych przypadkach bezpieczniejszy niż NOT IN?
8. W jakich częściach zapytania SQL można stosować podzapytania (np. WHERE, FROM, SELECT, HAVING)?
9. W jaki sposób podzapytania mogą współpracować z funkcjami agregującymi, takimi jak AVG, MAX czy COUNT?
10. Jakie korzyści daje stosowanie podzapytań w analizie danych w porównaniu do prostych zapytań jednowarstwowych?

Podsumowanie

Podzapytania są jednym z najważniejszych elementów języka SQL. Pozwalają wykonywać bardziej zaawansowane analizy, łączyć dane z wielu tabel w sposób pośredni, porównywać wartości w obrębie całych grup lub kategorii, a także budować warunki zależne od danych. Podzapytania nieskorelowane działają jak niezależne mini-zapytania, natomiast skorelowane tworzą dynamiczne zależności między danymi – dzięki czemu możemy uzyskać bardzo precyzyjne wyniki, np. dla każdego klienta, filmu czy pracownika osobno. W praktyce, dobry analityk SQL potrafi elastycznie przełączać się między różnymi technikami: używać podzapytań tam, gdzie są naturalne, a w innych przypadkach zastępować je złączeniami lub funkcjami okienkowymi, jeśli poprawi to wydajność i czytelność kodu.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

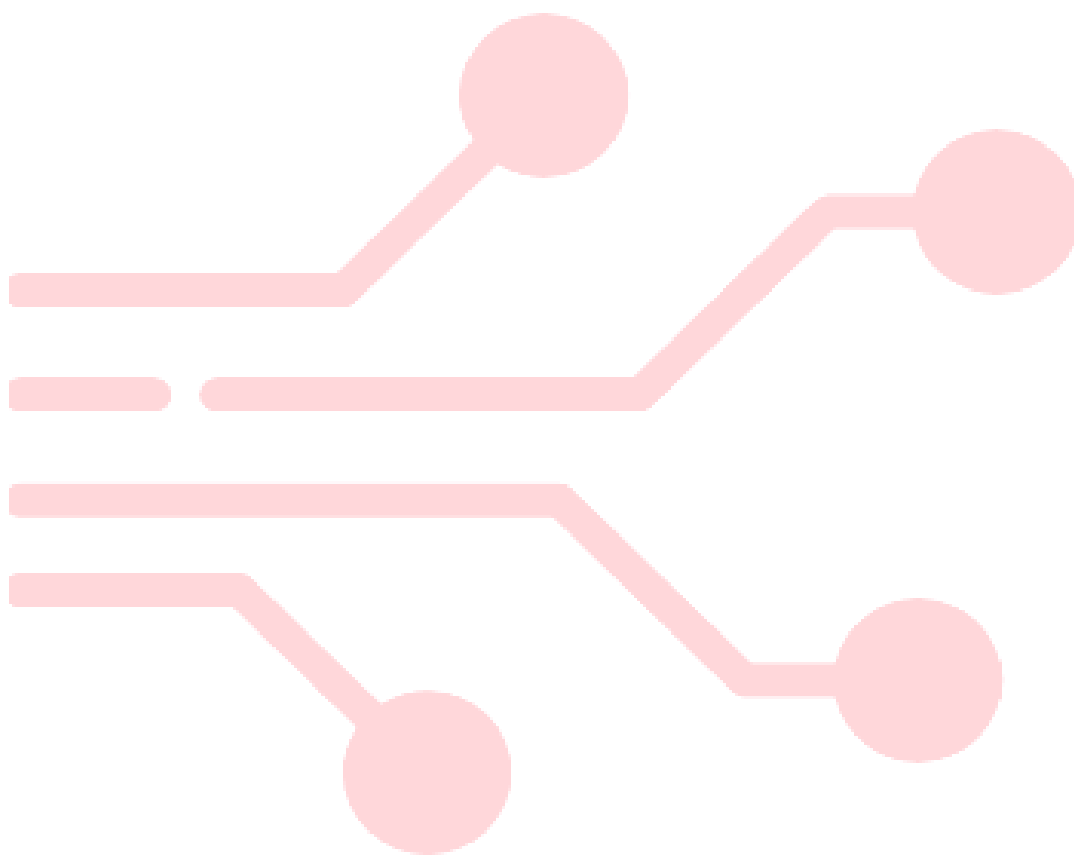


Literatura

Karwin, Bill, i in. Antywzorce języka SQL : jak uniknąć pułapek podczas programowania baz danych. Wyd. Helion, 2012.

<https://dev.mysql.com/doc/sakila/en/>

<https://www.w3schools.com/sql/>



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

