Fundusze Europejskie
Wiedza Edukacja Rozwój

Unia Europejska
Europejski Fundusz Społeczny

Programowanie zespołowe 2.0

Pobieranie informacji z bazy danych

Scenariusz zajęć

Czas realizacji: 180 minut
Cele ogólne: Podstawy języka SQL w dialekcie PostgreSQL.
Cel szczegółowy: Najważniejsze metody pobierania i prostego przetwarzania danych z bazy.
Konieczne wiadomości wstępne: Podstawy logiki matematycznej.
Metoda prowadzenia zajęć: Dyskusja z elementami wykładu.

Klauzula `SELECT`

Najprostsze zapytanie pobierające dane z bazy ma następującą strukturę:

```
SELECT Atrybuty
FROM Tabela
WHERE WarunekLogiczny
ORDER BY AtrybutySortujące;
```

Z tabeli `Tabela` pobiera ono obserwacje spełniające `WarunekLogiczny` i dla każdej z nich wyświetla jej `Atrybuty`. Wyświetlana tabela posortowana jest rosnąco względem atrybutów `AtrybutySortujące` (aby posortować tabelę w kolejności malejącej, należy użyć słowa kluczowego `DESC`).

Uwaga. Słowa kluczowe `WHERE` oraz `ORDER BY` nie są obligatoryjne. W przypadku braku warunku logicznego wszystkie rekordy z tabeli zostaną wyświetlone. Z kolei jeżeli nie wymusimy sortowania, obserwacje wyświetlą się w kolejności wprowadzania.

We wszystkich przykładach będziemy korzystać z bazy wypożyczalni filmów stworzonej na pierwszych zajęciach i uzupełnionej o przykładowe dane (najlepiej usuń stworzone wcześniej tabele i wykonaj wszystkie zapytania z pliku z przykładową bazą).

Przykład 1. Poniżej kilka prostych przykładów zapytań pobierających dane z tabel:

```
SELECT Imie, Nazwisko
FROM Uzytkownicy;
```

```
SELECT *
FROM Filmy
ORDER BY Rok;
```

```
SELECT UPPER(Tytul), Rok
FROM Filmy
WHERE Rok <= 1995 AND Cena >= 5
ORDER BY Cena DESC;
```

```
SELECT Tytul, ROUND(Cena/1.23, 2) AS CenaNetto
FROM Filmy
WHERE Tytul LIKE 'A%a%';
```

Ćwiczenie 1. Wykonaj powyższe zapytania. Co oznacza * ? Jakie filmy wyświetliły się w ostatnim zapytaniu?

Uwaga. Funkcje `UPPER` i `ROUND` są przykładami tzw. funkcji wierszowych. Służą one do przetwarzania atrybutów wyświetlanych obserwacji. Można je stosować również do konstrukcji warunków logicznych. Więcej o funkcjach wierszowych i warunkach logicznych dostępnych w bazie PostgreSQL można przeczytać w dokumentacji.

Pobieranie informacji z wielu tabel

Istnieją dwie metody syntezy informacji z wielu tabel: złączenia tabel i podzapytania. Większość problemów można rozwiązać korzystając z dowolnej z metod, ale w praktyce używa się tej, która jest w danym przypadku wygodniejsza.

Złączenia tabel

Złączenia dwóch tabel możemy dokonać na dwa sposoby:

```
Tabela1 JOIN Tabela2 ON WarunekZłączenia
```

lub

```
Tabela1 JOIN Tabela2 USING(Atrybuty)
```

`WarunekZłączenia` w pierwszej konstrukcji określa które krotki powinny być ze sobą „sklejone". Sposobu drugiego możemy użyć jeżeli w obu tabelach występuje atrybut o tej samej nazwie i chcemy złączyć obserwacje porównując ten atrybut. Efektem złączenia tabel jest tabela, więc złączania dokonujemy po słowie kluczowym `FROM`, tj.

```
SELECT Atrybuty
FROM Tabela1 JOIN Tabela2 ON WarunekZłączenia
WHERE WarunekLogiczny
ORDER BY AtrybutySortujące;
```

Uwaga. Operator złączenia `JOIN` jest łączny, więc możemy łączyć ze sobą więcej niż dwie tabele w następujący sposób:

```
Tabela1 JOIN Tabela2 USING(Atrybut1) JOIN Tabela3 ON WarunekZłączenia1
JOIN Tabela4 ON WarunekZłączenia2 ...
```

Przykład 2. Dwa poniższe pobierają informacje o zarejestrowanych wypożyczeniach:

```
SELECT Imie, Nazwisko, IdFilmu
FROM Uzytkownicy JOIN Wypozyczenia USING(IdUzytkownika);
```

```
SELECT Imie, Nazwisko, Tytul, Rok
FROM Uzytkownicy JOIN Wypozyczenia USING(IdUzytkownika)
JOIN Filmy USING(IdFilmu);
```

Ważnym typem złączenia jest złączenie tabeli z samą sobą. Wówczas musimy nadać kopiom tej samej tabeli lokalne nazwy (aliasy) i wskazywać z której tabeli chcemy pobrać atrybut. Poniżej wypisujemy pary użytkowników, którzy wypożyczyli ten sam film:

```
SELECT w1.IdUzytkownika, w2.IdUzytkownika
FROM Wypozyczenia w1 JOIN Wypozyczenia w2 USING(IdFilmu);
```

```
SELECT u1.Imie, u1.Nazwisko, Tytul, Rok, u2.Imie, u2.Nazwisko
FROM Uzytkownicy u1 JOIN Wypozyczenia w1 USING(IdUzytkownika)
JOIN Filmy USING(IdFilmu) JOIN Wypozyczenia w2 USING(IdFilmu)
JOIN Uzytkownicy u2 ON u2.IdUzytkownika = w2.IdUzytkownika
WHERE u1.IdUzytkownika < u2.IdUżytkownika;
```

Warunek logiczny w ostatnim zapytaniu pozwala uniknąć powtórzeń.

Ćwiczenie 2. Napisz zapytanie które:

1. wyświetli imiona i nazwiska użytkowników, którzy wypożyczyli jakiś film (rekordy nie powinny się powtarzać);
2. wyświetli imiona i nazwiska użytkowników, którzy wypożyczyli jakiś film wydany przed 2000 rokiem i tańszy niż 10 zł (rekordy nie powinny się powtarzać);
3. wyświetli imiona i nazwiska użytkowników, którzy wypożyczyli jakiś film w sobotę;
4. wypisze pary filmów wypożyczonych przez tego samego użytkownika;

Uwaga. Operator `JOIN` pomija krotki, których nie można do niczego przyłączyć. Do wypisywania wszystkich obserwacji z tabel służą tzw. złączenia zewnętrzne, tj. operatory `LEFT JOIN`, `RIGHT JOIN` oraz `FULL JOIN`. Jeżeli któraś obserwacja nie ma pasującej krotki w drugiej tabeli, to w wyniku dołączana jest obserwacja złożona z samych `NULL`-i.

Ćwiczenie 3. Wyświetl loginy użytkowników, którzy nie wypożyczyli jeszcze żadnego filmu.

Podzapytania

Jak sama nazwa wskazuje, podzapytanie jest zapytaniem wewnątrz zapytania. Podzapytania możemy umieszczać w dowolnych miejscach zapytania głównego, zapisujemy je w nawiasach. Podzapytanie może korzystać z informacji z tabel zewnętrznych oraz z tabel użytych w zapytaniach **wyższego** rzędu (nazywamy je wówczas podzapytaniami skorelowanymi).

Przykład 3. Podzapytania nieskorelowane najczęściej wykorzystujemy w konstrukcji warunków logicznych

```
SELECT *
FROM Uzytkownicy
WHERE IdUzytkownika NOT IN (SELECT IdUzytkownika
                            FROM Wypozyczenia);
```

```
SELECT *
FROM Filmy
WHERE Cena >= ALL(SELECT Cena
                  FROM Filmy);
```

Podzapytania skorelowane również bywają przydatne w konstrukcji warunków logicznych. W poniższym podzapytaniu obliczamy liczbę filmów starszych niż ustalony film z zapytania głównego. W efekcie otrzymujemy listę 30 najstarszych filmów z bazy.

```
SELECT *
FROM Filmy f1
WHERE (SELECT COUNT(*)
       FROM Filmy f2
       WHERE f2.Rok < f1.Rok) < 30;
```

Podzapytań możemy użyć również w celu wyświetlania informacji. Należy jednak wówczas pamiętać, by wynikiem podzapytania była pojedyncza informacja (jeden atrybut z pojedynczej obserwacji):

```
SELECT Imie, Nazwisko, (SELECT COUNT(*)
                        FROM Wypozyczenia w
                        WHERE w.IdUzytkownika = u.IdUzytkownika) AS Ile
FROM Uzytkownicy u
ORDER BY Ile DESC;
```

Ćwiczenie 4. Wykonaj zapytania z przykładu 3. Zastąp operator `NOT IN` operatorem `<> ALL`. Sprawdź w dokumentacji jak działa przyrównywanie liczby do listy z użyciem operatorów `ALL` i `ANY`.

Uwaga. W przykładzie 3 użyliśmy funkcji `COUNT`. Zlicza ona niepuste krotki na liście atrybutów podanej jako argument. `*` oznacza wszystkie atrybuty.

Grupowanie obserwacji

Czasami interesują nas pewne informacji nt. danych zagregowanych, np. pracowników w podziale na działy, studentów w podziale na grupy, widzów oglądających różne gatunki filmów, itp.

Aby móc wyświetlać takie informacje rozbudujemy nasz podstawowy schemat zapytania o dwa dodatkowe słowa kluczowe: `GROUP BY` i `HAVING`. Zapytanie grupujące będzie wyglądało następująco (znowu kolejność słów kluczowych jest ważna):

```
SELECT InformacjeOGrupach
FROM Tabela
WHERE WarunekLogicznyDlaObserwacji
GROUP BY AtrybutyGrupujące
HAVING WarunekLogicznyDlaGrup
ORDER BY AtrybutySortującеGrupy;
```

W odpowiedzi na takie zapytanie serwer wypisuje informacje o grupach utworzonych z obserwacji spełniających `WarunekLogicznyDlaObserwacji` przez identyfikację na atrybutach grupujących i spełniających `WarunekLogicznyDlaGrup`. Słowo kluczowe `HAVING` podobnie jak `WHERE` nie jest obligatoryjne. `InformacjeOGrupach` mogą być atrybutami użytymi do grupowania lub wynikami operacji na grupach. Najważniejszymi operacjami na grupach są: `COUNT`, `MIN`, `MAX`, `AVG`, `SUM`. Więcej o funkcjach grupujących można przeczytać w dokumentacji.

Przykład 4.

```
SELECT Rok, AVG(Cena)
FROM Filmy
GROUP BY Rok;
```

```
SELECT Cena, COUNT(*)
FROM Filmy
WHERE Rok < 2000
GROUP BY Cena;
```

```
SELECT Imie, Nazwisko, COUNT(DISTINCT IdFilmu)
FROM Uzytkownicy JOIN Wypozyczenia USING(IdUzytkownika)
GROUP BY Imie, Nazwisko
HAVING COUNT(DISTINCT IdFilmu) > 1;
```

```
SELECT Imie, Nazwisko, COUNT(DISTINCT IdFilmu) AS Liczba
FROM Uzytkownicy LEFT JOIN Wypozyczenia USING(IdUzytkownika)
GROUP BY Imie, Nazwisko
ORDER BY COUNT(*) DESC;
```

Ostatnie zapytanie zwróci **wszystkich** użytkowników, również tych, którzy jeszcze nie wypożyczyli żadnego filmu.

Ćwiczenie 5. Wykonaj zapytania z przykładu 4. Czy ostatnia tabela została posortowana według kolumny `Liczba`? Jeżeli nie, to co było kluczem sortowania?

Ćwiczenie 6. Jakie dane będzie pobierała przygotowywana aplikacja z bazy danych? Jakimi metodami będzie najwygodniej pobrać te informacje? Napisz odpowiednie zapytania SQL.

Odnośniki i załączniki

[1] S. Brown, S. L. Emerson, M. Darnovsky, *Podręcznik języka SQL*, WNT 2001.

[2] Język SQL w dialekcie PostgreSQL: https://www.postgresql.org/docs/11/sql.html

[3] Plik `baza.sql` z zapytaniami tworzącymi bazę danych wypożyczalni filmów uzupełnioną o przykładowe wpisy.