SQL

Perspektywy

Perspektywa lub **widok** (ang. *view*) to rodzaj tabeli logicznej czy też wirtualnej, tworzonej na bazie zapytania typu `SELECT`. Pozwala po prostu w inny sposób spojrzeć na strukturę bazy, z innej perspektywy — stąd też nazwa tej konstrukcji.

Widoki tworzy się za pomocą instrukcji `CREATE VIEW`, której ogólna postać wygląda następująco:

```
CREATE [OR REPLACE] VIEW nazwa_widoku [(lista_kolumn)]
AS zapytanie;
```

Użycie opcjonalnej klauzuli `OR REPLACE` spowoduje, że w przypadku gdyby perspektywa o podanej nazwie istniała w bazie, zostanie ona zmieniona zgodnie z podaną definicją (konstrukcja niedostępna w MS SQL i SQLite). Lista kolumn jest również opcjonalna (konstrukcja niedostępna w SQLite). Jeżeli zostanie pominięta, będą użyte nazwy wynikające z zapytania znajdującego się za słowem `AS`. Perspektywa zachowuje się podobnie jak tabela, a zatem można na niej operować, stosując typowe konstrukcje SQL.

17.1 Utworzenie perspektywy z lokalnej tabeli

Utwórz perspektywę, która będzie zawierała dane pracowników o płacy wyższej niż 1600 zł (pobrane z tabeli pracownicy. Następnie wyświetl wszystkie dane z tej perspektywy.

Perspektywa może być utworzona za pomocą następującej instrukcji:

```
CREATE VIEW pracownicy_up_1600
AS SELECT * FROM pracownicy WHERE placa > 1600;
```

17.1 Utworzenie perspektywy z lokalnej tabeli

Perspektywa będzie identyfikowana przez nazwę `pracownicy_up_1600`. W klauzuli `AS` została umieszczona zwykła instrukcja `SELECT` pobierająca dane z tabeli pracownicy spełniające warunek `placa > 1600`. Do nazwy `pracownicy_up_1600` można się odwoływać tak samo jak do zwykłej tabeli. A zatem aby wyświetlić wszystkie dane z perspektywy, należy użyć instrukcji:

```
SELECT * FROM pracownicy_up_1600;
```

17.1 Utworzenie perspektywy z lokalnej tabeli

```
mysql> CREATE VIEW pracownicy_up_1600
    -> AS SELECT * FROM pracownicy WHERE placa > 1600;
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT * FROM pracownicy_up_1600;
+----+---------+----------+---------+------------+------------+
| id | imie    | nazwisko | placa   | stanowisko | pesel      |
+----+---------+----------+---------+------------+------------+
|  1 | Adam    | Kowalski | 1624.50 | magazynier | 12345678901 |
|  2 | Adam    | Nowak    | 3760.00 | kierownik  | 92345678901 |
|  3 | Andrzej | Kowalski | 4200.00 | kierownik  | 72345678901 |
|  7 | Kacper  | Adamczyk | 1610.50 | serwisant  | 92341678903 |
+----+---------+----------+---------+------------+------------+
4 rows in set (0.01 sec)
```

17.2 Utworzenie perspektywy z wielu tabel

Utwórz perspektywę prezentującą dane dotyczące pracowników i przypisanych im stanowisk (z tabel przedstawionych w poprzednich ćwiczeniach. Dostępne mają być wyłącznie następujące informacje: identyfikator, imię, nazwisko pracownika oraz nazwa stanowiska. Imię i nazwisko mają być prezentowane w jednej kolumnie.

17.2 Utworzenie perspektywy z wielu tabel

Do utworzenia perspektywy niezbędne będzie użycie zapytania złączającego tabele `pracownicy` i `stanowiska`. Ponieważ jednak imię i nazwisko mają być prezentowane jako jedna kolumna, trzeba będzie połączyć dane z odpowiednich kolumn. W MySQL, MS SQL od wersji 2012 i PostgreSQL można do tego celu użyć funkcji `CONCAT`, przekazując jej w postaci argumentów nazwy łączonych kolumn:

```
CONCAT(ciąg1,…, ciągN);
```

17.2 Utworzenie perspektywy z wielu tabel

W przypadku Oracle powyższa funkcja może przyjmować tylko dwa argumenty:

```
CONCAT(ciąg1, ciąg2);
```

W PostgreSQL i Oracle można użyć operatora konkatenacji ||:

```
ciąg1 || ciąg2 || … ciągN
```

W MS SQL można użyć operatora konkatenacji +:

```
ciąg1 + ciąg2 + … ciągN
```

17.2 Utworzenie perspektywy z wielu tabel

Dodatkowo warto określić nazwy kolumn w perspektywie. Do wykonania ćwiczenia można więc zastosować instrukcję (MySQL, MS SQL od wersji 2012 i PostgreSQL):

```
CREATE VIEW pracownicy_stanowiska (id,
pracownik, stanowisko)
AS
SELECT p.id, CONCAT(p.imie, ' ', p.nazwisko),
s.nazwa
FROM pracownicy p INNER JOIN stanowiska s
ON p.stanowisko_id = s.id;
```

17.2 Utworzenie perspektywy z wielu tabel

lub (Oracle, PostgreSQL):

```
CREATE VIEW pracownicy_stanowiska (id,
pracownik, stanowisko)
AS
SELECT p.id, p.imie || ' ' || p.nazwisko,
s.nazwa
FROM pracownicy p INNER JOIN stanowiska s
ON p.stanowisko_id = s.id;
```

17.2 Utworzenie perspektywy z wielu tabel

lub (MS SQL):

```
CREATE VIEW pracownicy_stanowiska (id,
pracownik, stanowisko)
AS
SELECT p.id, p.imie + ' ' + p.nazwisko, s.nazwa
FROM pracownicy p INNER JOIN stanowiska s
ON p.stanowisko_id = s.id;
```

17.2 Utworzenie perspektywy z wielu tabel

lub (SQLite):

```
CREATE VIEW pracownicy_stanowiska
AS
SELECT p.id, p.imie || ' ' || p.nazwisko,
s.nazwa
FROM pracownicy p INNER JOIN stanowiska s
ON p.stanowisko_id = s.id;
```

17.2 Utworzenie perspektywy z wielu tabel

W celu skrócenia zapisu zostały zastosowane aliasy nazw tabel (`p` dla `pracownicy` i `s` dla `stanowiska`). Po wykonaniu powyższej instrukcji dane z perspektywy `pracownicy_stanowiska` można pobrać za pomocą zwykłego zapytania typu `SELECT`, stosując komendę:

```
SELECT * FROM pracownicy_stanowiska ORDER BY id;
```

17.2 Utworzenie perspektywy z wielu tabel

```
SELECT * FROM pracownicy_stanowiska ORDER BY id;
+----+----------------------+-----------+
| id | pracownik            | stanowisko |
+----+----------------------+-----------+
|  1 | Adam Kowalski        | magazynier |
|  2 | Adam Nowak           | kierownik  |
|  3 | Andrzej Kowalski     | kierownik  |
|  4 | Arkadiusz Malinowski | kierowca   |
|  5 | Andrzej Malinowski   | sprzedawca |
|  6 | Krzysztof Nowicki    | sprzedawca |
|  7 | Kacper Adamczyk      | serwisant  |
|  8 | Kamil Andrzejczak    | asystent   |
|  9 | Krzysztof Arkuszewski | magazynier |
| 10 | Kamil Borowski       | sprzedawca |
+----+----------------------+-----------+
10 rows in set (0.00 sec)
```

17.3 Perspektywa z sumą narastającą

Utwórz widok prezentujący sumę narastającą wartości zamówień z tabeli `zamowienia`. W wynikach uwzględnione powinny być: identyfikator, data oraz wartość zamówienia

17.3 Perspektywa z sumą narastającą

Uzyskanie identyfikatora, daty oraz wartości poszczególnych zamówień nie stanowi problemu, to zwykłe zapytanie typu `SELECT`. Pozostaje więc kwestia obliczeń kolejnych wartości dla sumy narastającej. Trzeba będzie skorzystać z podzapytania sumującego wartości z aktualnie przetwarzanego wiersza (przyda się do tego funkcja sumująca `SUM`) oraz wszystkich wcześniejszych wierszy. Należy zatem odpowiednio posortować wyniki zapytania zewnętrznego i skorzystać z warunków w klauzuli `WHERE` zapytania wewnętrznego (z tego powodu widok nie będzie mógł być utworzony w MS SQL, gdzie nie dopuszcza się do użycia w takiej konstrukcji klauzuli `ORDER BY`).

17.3 Perspektywa z sumą narastającą

Pełne zapytanie przyjmie postać:

```
CREATE VIEW suma_zamowien AS
      SELECT id, data, wartosc, (
         SELECT sum(wartosc)
         FROM zamowienia z2
      WHERE z2.id <= z1.id
   ) AS suma
  FROM zamowienia z1
  ORDER BY id;
```

17.3 Perspektywa z sumą narastającą

W MS SQL, ze względu na nieobsługiwanie przez ten serwer w widokach klauzuli `ORDER BY` bez wskazania liczby pobieranych wierszy, konieczne będzie dodanie do podzapytania klauzuli `TOP 100 PERCENT`:

```
CREATE VIEW suma_zamowien AS
   SELECT TOP 100 PERCENT id, data, wartosc, (
      SELECT sum(wartosc)
      FROM zamowienia z2
      WHERE z2.id <= z1.id
   ) AS suma
FROM zamowienia z1
ORDER BY id;
```

17.3 Perspektywa z sumą narastającą

Do usuwania widoków służy polecenie `DROP VIEW` o schematycznej postaci:

```
DROP VIEW [IF EXISTS] nazwa1[, nazwa2,…,nazwaN];
```

Użycie klauzuli `IF EXISTS` pozwala uniknąć wygenerowania przez instrukcję błędu w sytuacji, gdy widok o podanej nazwie nie istnieje (tylko w przypadku MySQL, PostgreSQL i SQLite). Jednoczesne usuwanie kilku widoków dostępne jest w MySQL, PostgreSQL i MS SQL.

17.4 Usuwanie perspektywy

Usuń perspektywy utworzone w ćwiczeniach

Aby wykonać ćwiczenie, można użyć serii instrukcji:

```
DROP VIEW pracownicy_up_1600;
DROP VIEW pracownicy_stanowiska;
DROP VIEW suma_zamowien;
```

lub jednej instrukcji (MySQL, PostgreSQL i MS SQL):

```
DROP VIEW pracownicy_up_1600,
pracownicy_stanowiska, suma_zamowien;
```