Pobieranie danych, czyli instrukcja SELECT

Pobieranie danych

Informacje przechowywane w bazach danych mogą być pobrane za pomocą instrukcji języka SQL SELECT. Instrukcja SELECT (zapytanie) określa, jakie dane mają zostać zwrócone w wyniku jej wykonania, natomiast to, w jaki sposób instrukcja będzie wykonana, zależy od serwera baz danych.

Język SQL jest językiem strukturalnym, a nie proceduralnym. W języku tym nie określamy sposobu wykonania zadania (tak jak np. w C), ale jego wynik (a właściwie wymagane do jego otrzymania operacje na pewnych strukturach — zbiorach). Z założenia instrukcje języka SQL przypominają potoczny język angielski. Żeby na przykład odczytać nazwę i cenę produktu, nie napiszemy programu za pomocą jakiegoś algorytmu wyszukiwania znajdującego odpowiednie informacje, ale po prostu powiemy (napiszemy): Odczytaj nazwę i cenę produktu o podanym identyfikatorze.

Instrukcja SELECT służy do pobierania danych z bazy. Instrukcja musi zawierać (z wyjątkiem polecenia SELECT odwołującego się wyłącznie do stałych, zmiennych lub wyrażeń arytmetycznych) co najmniej jedną klauzulę: za pomocą polecenia SELECT określamy interesujące nas kolumny (dokonujemy operacji selekcji pionowej, projekcji), za pomocą klauzuli FROM wskazujemy tabelę, z której pobieramy dane. Z reguły ogranicza się również, za pomocą klauzuli WHERE, liczbę zwracanych wierszy do rekordów spełniających określone kryteria (operacja selekcji poziomej, selekcji).

Odczytujemy wszystkie dane z tabeli

Najprostszy przykład użycia instrukcji SELECT to odczytanie całej zawartości wskazanej tabeli. W SQL-u możemy posługiwać się kilkoma znakami specjalnymi; jednym z nich jest *, oznaczająca „wszystko”. Czyli żeby odczytać wszystko z tabeli customer, należy wykonać instrukcję:

```
SELECT *

FROM customer;
```

albo:

```
SELECT *

FROM test.customer;
```

Klauzula FORM. Wybieranie kolumn

Wybieramy kolumny z tabel

Wiemy już, że instrukcja SELECT służy do pobierania danych z bazy i że z reguły zawiera ona co najmniej dwie klauzule: w klauzuli SELECT określamy interesujące nas kolumny, w klauzuli FROM wskazujemy tabelę, z której pobieramy dane (listing 3.1).

Listing 3.1. Zapytanie zwracające imiona i nazwiska klientów

```
SELECT fname, lname
FROM customer;
```

Nazwy kolumn wymienionych w klauzuli SELECT należy oddzielić przecinkami.

Kolejność podawanych nazw kolumn nie jest obojętna. Serwer baz danych zwróci dane, szeregując poszczególne kolumny według kolejności ich występowania w klauzuli SELECT.

Oprócz nazw kolumn język SQL dopuszcza używanie w klauzuli SELECT wyrażeń, aliasów oraz literałów. Dodatkowo możliwe jest łączenie (konkatenacja) kolumn. Kolejne punkty przedstawiają sposoby wykorzystania tych możliwości.

Wyrażenia

W skład wyrażenia mogą wchodzić nazwy kolumn, stałe, wartości liczbowe i ogólnie znane operatory, takie jak: + (dodawania), – (odejmowania), * (mnożenia), / (dzielenia) (listing 3.3).

Listing 3.3. Prosty przykład zastosowania funkcji arytmetycznych — zwróć uwagę, że ta sama kolumna tabeli może być wielokrotnie zwrócona przez zapytanie

```
SELECT cost_price,cost_price*0.22
FROM item;
```

Serwer baz danych wykonuje poszczególne operacje arytmetyczne w takiej samej kolejności, jaka obowiązuje w klasycznych działaniach matematycznych, tj. jako pierwsze wykonywane są wyrażenia umieszczone w nawiasach, następnie mnożenie, dzielenie, dodawanie i odejmowanie.

Eliminowanie duplikatów

Eliminacja duplikatów

Domyślnie serwery bazodanowe wyświetlają wszystkie wiersze wchodzące w skład wyniku zapytania, nawet jeżeli w kilku wierszach przechowywana jest taka sama wartość. Dlatego, odczytując identyfikatory klientów z tabeli orderinfo, zobaczymy identyfikator tego samego klienta tyle razy, ile razy składał on u nas zamówienia

Do wyeliminowania z powyższego zestawienia powtarzających się wartości służy słowo kluczowe DISTINCT. Słowo DISTINCT musi pojawić się bezpośrednio po słowie kluczowym SELECT i odnosi się do wszystkich kolumn występujących w klauzuli SELECT. Aby w naszym przypadku wyeliminować powtarzające się identyfikatory, wystarczy napisać :

SELECT DISTINCT customer_id

FROM orderinfo;

Słowo kluczowe DISTINCT eliminuje powtarzające się wartości wierszy, a nie pojedynczych wyrażeń.

Funkcje arytmetyczne.

Wbudowane, skalarne funkcje matematyczne, służą do operacji na liczbach. Poniżej znajdziesz opis kilku wybranych, najczęściej używanych.

ROUND (wartość_liczbowa, precyzja) – zaokrągla wartość liczbową, zmiennoprzecinkową do zadenj precyzji.

```
USE AdventureWorks2008
GO

SELECT SalesOrderId, TotalDue,
       ROUND(TotalDue,2) as TotalDueRounded
FROM Sales.SalesOrderHeader
WHERE TotalDue BETWEEN 123 AND 124
```

RAND () – funkcja pseudolosowa, zwraca liczbę typu float z zakresu 0-1. Łatwo w oparciu o nią stworzyć generator liczb pseudolosowych zwracający liczby losowe z określonego przedziału.

```
-- przedział 10 - 20
SELECT 10 + CONVERT(INT, (20-10+1) * RAND())
-- Przedział 0-10
SELECT 0 + CONVERT(INT, (10-0+1) * RAND())
```

POWER (liczba, potęga) oraz **SQRT**(liczba) to funkcje potęgowania oraz pierwiastkowania (kwadratowego)

Funkcje znakowe, funkcje daty i czasu.

Przekształcenia ciągów znakowych (stringów) w SQL

Funkcje operujące na ciągach znaków, są jednymi z najczęściej stosowanych. Ich nazewnictwo jest intuicyjne i często bardzo podobne lub nawet identyczne, ze stosowanym w innych językach programowania.

W tym artykule przedstawiam najczęściej używane funkcje związane z przetwarzaniem wartości ciągów znakowych typu (,n)char/varchar.

LEFT(exp, n) oraz **RIGHT**(exp, n) – to jedne z najprostszych funkcji tekstowych. Zwracają *n* znaków od lewej lub prawej z wyrażenia *exp* na którym działa.

```
USE Northwind
GO

-- LEFT/RIGHT - n znaków z lewej / prawej strony

SELECT LEFT(FirstName,1) + '. ' + LastName as FullName,
       RIGHT(HomePhone,8) as ShortPhone,
       FirstName, HomePhone
FROM dbo.Employees
```

Results | Messages

	FullName	ShortPhone	FirstName	HomePhone
1	N. Davolio	555-9857	Nancy	(206) 555-9857
2	A. Fuller	555-9482	Andrew	(206) 555-9482
3	J. Leverling	555-3412	Janet	(206) 555-3412
4	M. Peacock	555-8122	Margaret	(206) 555-8122
5	S. Buchanan	555-4848	Steven	(71) 555-4848
6	M. Suyama	555-7773	Michael	(71) 555-7773
7	R. King	555-5598	Robert	(71) 555-5598
8	L. Callahan	555-1189	Laura	(206) 555-1189
9	A. Dodsworth	555-4444	Anne	(71) 555-4444

W tym przykładzie funkcja LEFT(FirstName,1) zwraca zawsze sensowną wartość – inicjał od imienia. Z kolei RIGHT (HomePhone,8) – zwróci zawsze tylko ostatnie 8 znaków. Jeśli byśmy chcieli wyciągnąć zmienną liczbę znaków np. od lewej (numer kierunkowy) – potrzebujemy wsparcia kolejnej funkcji, która nam namierzy ten zmienny punkt.

UPPER (string) oraz **LOWER**(string) zamienia wszystkie litery na duże lub małe.

LEN(exp) – funkcja zwraca wartość typu integer, równej liczbie znaków (długości) wyrażenia będącego jej argumentem. Często stosowana z innymi funkcjami – np. SUBSTRING.

```
-- UPPER / LOWER  - WIELKIE/ male litery, LEN - długość stringu

SELECT ProductName, UPPER( ProductName ) as UpperName,
       LOWER( ProductName ) as LowerName,
       LEN( ProductName ) NameLength
FROM dbo.Products
WHERE LEN( ProductName ) < 10
```

Results | Messages

	ProductName	UpperName	LowerName	NameLength
1	Chai	CHAI	chai	4
2	Chang	CHANG	chang	5
3	Chocolade	CHOCOLADE	chocolade	9
4	Filo Mix	FILO MIX	filo mix	8
5	Geitost	GEITOST	geitost	7
6	Ikura	IKURA	ikura	5
7	Konbu	KONBU	konbu	5
8	Maxilaku	MAXILAKU	maxilaku	8
9	Pavlova	PAVLOVA	pavlova	7
10	Spegesild	SPEGESILD	spegesild	9
11	Tofu	TOFU	tofu	4
12	Tourtiere	TOURTIERE	tourtiere	9
13	Tunnbröd	TUNNBRÖD	tunnbröd	8

SUBSTRING (exp , start_location , n) – zwraca fragment tekstu, liczbę n znaków z wyrażenia podanego w parametrze exp, startując od zadanego miejsca start_location. Napiszmy zapytanie wyciągające tylko nazwę domeny z pełnego adresu URL. W tym celu musimy wyciągnąć fragment tekstu z pominięciem http:// (od ósmego znaku) do pierwszego wystąpienia znaku / (licząc również od ósmej pozycji)

```
-- SUBSTRING - wyciąga konrketny framgent stringu

SELECT  WebPage,
        SUBSTRING(WebPage,8, CHARINDEX('/',webPage,8)-8 ) as OnlyDomain
FROM
(
        -- potraktuj to jak tabelę z dwoma wierszami ;)
        SELECT 'http://www.sqlpedia.pl/pisanie-zapytan-w-jezyku-sql-kurs/'
as WebPage
        UNION
        SELECT 'http://sqlpedia.pl/kurs-sql/'

) as Pages
```

Results | Messages

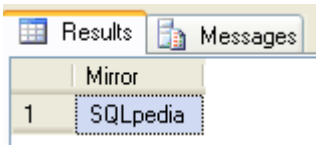
	WebPage	OnlyDomain
1	http://www.sqlpedia.pl/pisanie-zapytan-w-jezyku-sql-kurs/	www.sqlpedia.pl
2	http://sqlpedia.pl/kurs-sql/	sqlpedia.pl

Łącząc ze sobą podstawowe funkcje, można osiągnąć całkiem skomplikowane przekształcenia.

RTRIM (exp) oraz **LTRIM** (exp) – obcięcie z prawej/lewej znaków spacji danego wyrażenia exp. Szczególnie często spotykane przy łączeniu stringów typu CHAR o stałej długości.

REVERSE (exp) – odbicie lustrzane stringu czyli ostatni będą pierwszymi.

```
-- REVERSE - lustro

SELECT REVERSE('aidepLQS') as Mirror
```



i

Kolejną istotną grupą wbudowanych funkcji skalarnych są obiekty związane z przetwarzaniem typów danych daty i czasu.

YEAR (data), **MONTH** (data), **DAY** (data) – dokonują ekstraktu z daty, odpowiednio roku, miesiąca oraz dnia.

Najczęściej stosowane funkcje zwracające date i czas systemowy to **GETDATE**() oraz **SYSDATETIME**(). Są to funkcje nie przyjmujące żadnych argumentów i zwracają po prostu bieżącą datę i czas systemowy.

```
-- funkcje zwracające aktualny czas i datę systemową

SELECT SYSDATETIME(),
       SYSDATETIMEOFFSET(),
       GETDATE(),
       GETUTCDATE()
```

DATEADD (datepart, liczba, data) – dodaje (lub odejmuje) liczbę jednostek daty/czasu określonych za pomocą datepart np dni (day, dd, d), lat (years,yy,yyyy), miesięcy (month,mm,m), minut (minute,mi,n) etc. do zadanej daty. Jednostki określone mogą być za pomocą pełnej nazwy, lub skrótu. Pełny ich opis znajdziesz tutaj. Stosowana często w warunkach filtracji, np. wszystkie zlecenia z ostatnich 14 dni. Funkcja DATEADD, jest też bardzo użyteczna w określaniu zakresów

```
-- DATEADD - dodawanie/odejmowanie jednostek określonego typu z zadanej
daty

SELECT  DATEADD ( dd,-DAY( GETDATE()-1 ), GETDATE() ) as FirstDayCurrMonth,
        DATEADD ( dd,-DAY( GETDATE() ), GETDATE() ) as LastDayPrevMonth
```

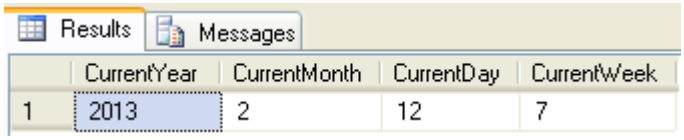
DATEDIFF (datepart, startdate, enddate) – różnica pomiędzy dwiema datami (end – start) wyrażona w jednostkach określonych przez datepart. Wiek pracowników :

```
USE Northwind
GO

-- DATEDIFF - określanie różnicy wyrażonej w konkretnych jednostkach
-- pomiędzy dwiema datami

SELECT  FirstName, LastName, BirthDate,
```

```
        DATEDIFF ( yy , BirthDate , GETDATE() )  as Age
FROM dbo.Employees
```

DATEPART(datepart, data) – wyciąga określoną parametrem datepart, jednostkę podanej daty.

```
-- DATEPART - ekstrakt określonej części daty /czasu

SELECT  DATEPART( yy, GETDATE() ) as CurrentYear,
        DATEPART( mm, GETDATE() ) as CurrentMonth,
        DATEPART( dd, GETDATE() ) as CurrentDay,
        DATEPART( ww, GETDATE() ) as CurrentWeek
```

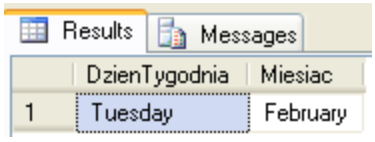


Results | Messages

	CurrentYear	CurrentMonth	CurrentDay	CurrentWeek
1	2013	2	12	7

DATENAME (datepart, data) – pododbna do DATEPART, zwraca wartość znakowa, określonej parametrem datepart, części daty w tym nazwę dnia tygodnia, miesiąca zgodnie z ustawieniami @@LANGID (bieżący język dla sesji)

```
SELECT  DATENAME(dw, GETDATE() ) as DzienTygodnia,
        DATENAME(mm, GETDATE() ) as Miesiac
```



Results | Messages

	DzienTygodnia	Miesiac
1	Tuesday	February

Formatowanie wyników. Aliasy. Sortowanie wyników.

Aliasy

Język SQL umożliwia zastąpienie nazwami opisowymi domyślnych, utworzonych podczas tworzenia tabel, nagłówków kolumn. Jest to szczególnie przydatne w przypadku używania wyrażeń. Ponieważ wynik wyrażenia jest obliczany w chwili wykonania instrukcji SELECT, w tabeli nie znajduje się kolumna, w której byłyby przechowywane dane będące wynikiem operacji arytmetycznej. W związku z tym serwer baz danych „nie wie”, jaka miałaby być domyślna nazwa nowej kolumny. Jak widzieliśmy, MySQL za nagłówek nowej kolumny przyjmie wyrażenie arytmetyczne. Aby utworzyć alias dla kolumny, należy bezpośrednio po nazwie, którą chcemy zastąpić, użyć słowa kluczowego AS, a następnie podać nową nazwę kolumny .

Aliasy kolumn definiuje się w klauzuli SELECT albo podając je bezpośrednio po oryginalnej nazwie kolumny, albo poprzedzając je słowem kluczowym AS. Radzę zawsze używać słowa kluczowego AS — poprawia ono czytelność zapytań, dzięki czemu łatwiej można zauważyć brak przecinka pomiędzy nazwami dwóch odczytywanych kolumn.

Przykład użycia aliasów

```
SELECT cost_price AS 'Cena kupna', sell_price AS 'Cena sprzedaży' ,sell_price-cost_price
AS Zysk

FROM item;
```

Zwróćmy uwagę, że dwa pierwsze aliasy zostały zapisane w apostrofach — to dlatego, że zawierają spację. Jeżeli nazwa jakiegoś obiektu jest niezgodna z konwencją języka SQL (na przykład zawiera spację), trzeba zapisywać ją w apostrofach.

Literały

Oprócz nazw kolumn, ich aliasów i wyrażeń w klauzuli SELECT można umieszczać literały. Literałem jest dowolny ciąg znaków lub liczb. W wyniku umieszczenia w niej literału serwer baz danych do każdego zwróconego wiersza wpisze do odpowiedniej kolumny treść literału. Na przykład do utworzenia raportu informującego o tym, gdzie mieszka dana osoba, można wykorzystać instrukcję pokazaną poniżej.

SQL jest językiem operowania na zbiorach. Dzięki temu dodanie do każdego wiersza wyniku stałej wartości sprowadza się do wymienienia jej w klauzuli SELECT

```
SELECT fname, lname,' mieszka w ', town

FROM customer;
```

Porządkowanie danych

Instrukcja SELECT zwraca wiersze w tej kolejności, w jakiej dane są przechowywane w tabeli. Z reguły jest to kolejność, w jakiej były dopisywane następne wiersze z danymi. Do zmiany kolejności, w jakiej zwracane będą wyniki zapytania, służy klauzula ORDER BY. ORDER BY („uporządkuj według”) jest opcjonalnym składnikiem instrukcji SELECT. Jeżeli jednak ta klauzula wystąpi, musi być ostatnią.

Kolejność klauzul instrukcji SELECT nie jest dowolna.

Obowiązkowym parametrem klauzuli ORDER BY jest wyrażenie lub nazwa kolumny — według ich wartości należy posortować dane wynikowe. Wykonanie poniższej instrukcji spowoduje wyświetlenie opisów towarów i cen ich zakupu uszeregowanych według cen zakupu

Posortowana lista towarów

```
SELECT description, cost_price

FROM item

ORDER BY cost_price;
```

Domyślnie dane szeregowane są w porządku rosnącym, czyli od wartości najmniejszych do największych w przypadku danych liczbowych, od najwcześniejszych do najpóźniejszych w przypadku dat oraz w porządku alfabetycznym w przypadku ciągów znakowych. Aby odwrócić kolejność sortowania, należy bezpośrednio po nazwie kolumny użyć słowa kluczowego DESC (ang. Descending) (listing 3.16).

Listing 3.16. Lista towarów posortowana od najdroższego do najtańszego

```
SELECT description, cost_price

FROM item

ORDER BY cost_price DESC;
```

Oczywiście możemy sortować dane według więcej niż jednego kryterium. Instrukcja z listingu 3.18 zawiera alfabetycznie uporządkowaną listę adresów klientów — najpierw dane sortowane są według nazw miast, a następnie według nazw ulic.

Listing 3.18. Jeżeli kilku klientów mieszka w tym samym mieście, o ich kolejności na liście zadecyduje nazwa i numer ulicy

```
SELECT town, addressline

FROM customer

ORDER BY town, addressline
```

Klauzula WHERE.

Wybieranie wierszy

Instrukcje SELECT w tej postaci, której używaliśmy do tej pory, zwracały wszystkie wiersze z danej tabeli. Do ograniczenia (wybrania) wierszy w wyniku należy wykorzystać klauzulę WHERE. Odpowiada ona teoriomnogościowemu operatorowi selekcji, czyli wybierania wierszy. Operacja ta najczęściej polega na wyborze grupy wierszy z tabeli na podstawie pewnych kryteriów. Serwer baz danych dla każdego wiersza sprawdzi, czy spełnia on kryteria wyboru, i jeżeli tak — zostanie on dodany do wyniku zapytania.

Klauzula WHERE, o ile była użyta, musi wystąpić bezpośrednio po klauzuli FROM. Kryterium wyboru może być sformułowane za pomocą typowych operatorów porównania lub operatorów charakterystycznych dla języka SQL.

Operatory logiczne

Rolę operatorów logicznych przedstawimy na kilku przykładach:

1. Operator AND (logiczne I, czyli koniunkcja) może być użyty do połączenia następujących warunków logicznych: cena < 500 AND kolor = niebieski. Otrzymany w ten sposób złożony warunek logiczny jest prawdziwy tylko wtedy, gdy cena jest niższa niż 500, a kolor — niebieski.

2. Operator OR (logiczne LUB, czyli alternatywa) może być użyty do połączenia następujących warunków logicznych: kolor = niebieski LUB kolor = czerwony LUB kolor = żółty. Otrzymany w ten sposób złożony warunek logiczny jest prawdziwy, jeżeli kolor jest niebieski, czerwony lub żółty.

3. Operatory AND, OR i NOT (logiczne NIE, czyli negacja) mogą być użyte do połączenia kilku prostych warunków logicznych: cena < 500 AND (kolor NOT czarny OR marka NOT Ford). Otrzymany w ten sposób złożony warunek logiczny jest prawdziwy tylko wtedy, gdy cena jest niższa niż 500 i albo kolor nie jest czarny, albo marką nie jest Ford.

Operatory charakterystyczne dla języka SQL

Operatorów języka SQL, tak jak pozostałych operatorów, można używać do porównywania różnych typów danych. Do operatorów SQL zalicza się:

1. Operator BETWEEN ... AND (należy do przedziału).
2. Operator IN (lista) (należy do zbioru).
3. Operator LIKE (jest zgodny z wzorcem).
4. Operator IS (jest).

BETWEEN ... AND

Operator BETWEEN ... AND służy do sprawdzenia, czy dana wartość znajduje się w podanym przedziale. MySQL traktuje ten przedział jako obustronnie zamknięty, czyli zalicza do niego wartości graniczne. Dodatkowo, jeżeli chcemy, żeby zapytanie zwróciło jakiekolwiek dane, górna granica przedziału nie może być mniejsza niż dolna (listing 3.26).

Listing 3.26. Lista towarów, których cena sprzedaży jest większa od 10, ale mniejsza niż 15

```
SELECT *

FROM item

WHERE sell_price BETWEEN 10 AND 15;
```

IN

Operator IN służy do sprawdzenia, czy dana wartość należy do podanego zbioru. Przy czym może to być zbiór dowolnych wartości, niekoniecznie liczb (listing 3.27).

Listing 3.27. Dane osób mieszkających w Nicetown lub Welltown

```
SELECT fname, town
FROM customer
WHERE town IN ('Nicetown','Welltown');
```

LIKE

Za pomocą operatora LIKE możemy wyszukać dane tekstowe zgodne z podanym wzorcem. Przy tworzeniu wzorca można wykorzystać symbole o specjalnym znaczeniu:

1. Znak % (procent) — zastępuje dowolny ciąg znaków.
2. Znak _ (podkreślenie) — zastępuje dokładnie jeden dowolny znak.

Aby na przykład odszukać wszystkie panie lub panny, których nazwisko zaczyna się od liter ST, napiszemy (listing 3.28):

Listing 3.28. Operator LIKE powinien być używany tylko z danymi tekstowymi

```
SELECT title, fname, lname
FROM customer
WHERE title LIKE '%s' AND lname LIKE 'St%';
```

IS

Operator IS służy przede wszystkim do wyszukiwania tych rekordów, w których przechowywana jest wartość nieokreślona (wartość Null), oraz do sprawdzania, czy dana wartość nie jest nieokreślona (listing 3.29).

Listing 3.29. Zapytanie zwracające informacje o towarach, które mają nazwę, ale nie mają określonej ceny sprzedaży

```
SELECT *
FROM item
WHERE sell_price IS NULL AND description IS NOT NULL;
```

Hierarchia operatorów

Ostateczny wynik złożonych warunków logicznych zależy od kolejności, w jakiej serwery bazodanowe sprawdzą prawdziwość składających się na nie prostych warunków. Kolejność ta wynika z hierarchii operatorów przyjętej w standardzie języka SQL:

1. Jako pierwsze wykonywane są operacje mnożenia i dzielenia.

2. Następnie dodawania i odejmowania.

3. W trzeciej kolejności wykonywane są standardowe operacje porównania, takie jak „równy” lub „mniejszy niż”.

4. Pierwszym wykonywanym operatorem logicznym jest operator NOT.

5. Następnie wykonywany jest operator AND.

6. Jako ostatnie — operatory SQL (IN, BETWEEN ... AND, LIKE) oraz operator OR.

Jeżeli wyrażenie zawiera kilka operatorów o tym samym priorytecie, wykonywane są one od lewej do prawej.

[ii]

Funkcje grupujące. COUNT(), MIN(), MAX(), SUM(), AVG().

Funkcje grupujące

Grupowanie danych polega na łączeniu wielu wierszy w jeden. W najprostszym przypadku łączy się wszystkie wiersze tabeli w jedną grupę, ale możliwe jest też ich podzielenie pomiędzy wiele grup. Wtedy podstawą zaklasyfikowania wiersza do danej grupy jest wartość jednej z kolumn lub wynik wyrażenia.

Funkcje, które zwracają jedną wartość obliczoną na podstawie przekazanego zbioru parametrów, nazywamy funkcjami grupującymi. W każdym serwerze baz danych, w tym w MySQL-u, zaimplementowano najważniejsze i najczęściej używane funkcje tego typu — minimum, maksimum, średnią, sumę itd.

Wiesz już, że parametrem wywołania funkcji grupujących nie są pojedyncze wartości, ale grupy (zbiory) wartości i że dzięki tym funkcjom uzyskujemy pojedynczy wynik obliczony na podstawie wielu argumentów. Na przykład możemy w tabeli policzyć wiersze spełniające określone kryteria lub możemy wyliczyć wartość średnią dla wszystkich wartości z wybranej kolumny. Użycie tych funkcji zwykle związane jest z operacją na wskazanych kolumnach (na których wykonywane są obliczenia), a jako wynik zwracany jest tylko jeden wiersz.

Charakterystyczną cechą funkcji grupujących jest operowanie na zbiorach, a nie pojedynczych wartościach. Dzięki temu otrzymane w wyniku grupowania dane mogą być użyte jako argumenty ich wywołania. Jeżeli wszystkie wiersze tabeli są połączone w jedną grupę, funkcja grupująca będzie wywołana tylko raz, w innym przypadku zostanie wywołana dla każdej grupy. Funkcje grupujące

zwracają pojedyncze (skalarne) wartości, więc wywołuje się je w klauzuli SELECT, tak jak wcześniej poznane funkcje systemowe.

Funkcja COUNT()

Pierwszą funkcją agregującą, którą chcę dokładnie omówić, jest funkcja COUNT(). Funkcja ta zlicza w przekazanym zbiorze wartości wystąpienia różne od NULL, chyba że jako argumentu użyto znaku * (gwiazdka) — takie wywołanie funkcji spowoduje zliczenie wszystkich wierszy, łącznie z duplikatami i wartościami NULL. Argumentem funkcji mogą być liczby, daty, znaki i ciągi znaków (listing 6.1).

Jeśli chcemy znać liczbę wierszy zwróconych przez zapytanie, najprościej jest użyć funkcji COUNT(*). Są dwa powody, dla których warto tak wywołać funkcję COUNT() do tego celu. Po pierwsze, pozwalamy optymalizatorowi bazy danych wybrać kolumnę do wykonywania obliczeń, co czasem nieznacznie podnosi wydajność zapytania, po drugie, nie musimy się martwić o wartości Null zawarte w kolumnie oraz o to, czy kolumna o podanej nazwie w ogóle istnieje.

Listing 6.1. Zapytanie zwracające liczbę klientów

```
SELECT COUNT(*) as 'Liczba klientów'

FROM customer;
```

Jak widać, zapytanie zwróciło jedną wartość wyliczoną przez funkcję grupującą COUNT() na zbiorze równym zawartości tabeli item. Gdybyśmy chcieli policzyć imiona i nazwiska klientów, otrzymalibyśmy nieco inny wynik. Wywołanie funkcji w postaci COUNT(nazwa kolumny) nie uwzględnia pól z wartościami Null . Fakt, że wiersze z wartością Null nie są zliczane, może być przydatny, gdy wartość Null oznacza coś szczególnego lub gdy chcemy sprawdzić, czy w bazie nie brakuje istotnych informacji (listing 6.2).

Listing 6.2. Funkcja COUNT() wywołana dla dwóch różnych zbiorów — raz dla nazwisk, raz dla imion klientów. Jak widać, jedna osoba nie podała nam imienia.

```
SELECT COUNT(fname), COUNT(lname)

FROM customer;
```

```
+--------------+--------------+
| COUNT(fname) | COUNT(lname) |
+--------------+--------------+
| 15           | 16           |
+--------------+--------------+
```

Jeżeli chcemy policzyć unikatowe wystąpienia wartości, wystarczy wykorzystać wiedzę z wcześniejszych odcinków kursu i właściwie użyć słowa kluczowego DISTINCT (listing 6.3).

Listing 6.3. Zapytanie zwracające liczbę miast, w których mieszkają nasi klienci — w pierwszej kolumnie to samo miasto liczone jest tyle razy, ilu mieszka w nim klientów, w drugiej kolumnie każde miasto policzone jest tylko raz

```
SELECT COUNT(town), COUNT(DISTINCT(town))

FROM customer;
```

```
+-------------+----------------------+
| COUNT(town) | COUNT(DISTINCT(town)) |
+-------------+----------------------+
| 15          | 12                   |
+-------------+----------------------+
```

Funkcja SUM()

Za pomocą funkcji SUM() dodawane są wszystkie wartości rekordów wybranych w zapytaniu i zwracany jest pojedynczy wynik. W przeciwieństwie do funkcji COUNT(), która działa dla wszystkich typów danych, argumentami funkcji SUM() mogą być wyłącznie liczby. Funkcja SUM(), tak jak inne funkcje grupujące, nie uwzględnia wartości Null (listing 6.4).

Listing 6.4. Zapytanie zwracające liczbę wszystkich towarów w magazynie

```
SELECT SUM(quantity)

FROM stock;
```

Funkcja AVG()

W wyniku działania funkcji AVG() zwracana jest wartość średnia dla podanych wyrażeń. Wiersze przechowujące wartość Null nie są uwzględniane. Argumentami funkcji AVG() muszą być dane liczbowe. Aby na przykład obliczyć średnie ceny sprzedaży i zakupu towarów, napiszemy (listing 6.6):

Listing 6.6. Średnia cena wszystkich kupionych i sprzedawanych towarów

```
SELECT AVG (cost_price), AVG (sell_price)

FROM item;
```

Funkcje MIN() i MAX()

Funkcja MIN() służy do znajdowania wartości najmniejszej w zbiorze wartości, a funkcja MAX() — największej. Obie funkcje, podobnie jak funkcja COUNT(), mogą być użyte dla różnych typów danych.

Za pomocą funkcji MAX() można znaleźć największy łańcuch danych, najnowszą datę (lub najodleglejszą w przyszłości) oraz największą liczbę w zestawieniu. W wyniku działania funkcji MIN() można znaleźć odpowiednio wartości najmniejsze. Aby znaleźć datę pierwszej operacji naszej firmy, możemy skorzystać z instrukcji pokazanej na listingu 6.8.

Listing 6.8. Data pierwszego zamówienia

```
SELECT MIN(date_placed)

FROM orderinfo;
```

Wyrażenia. Klazula GROUP BY.

Grupowanie danych

Do tej pory wywoływaliśmy funkcje grupujące raz dla całych tabel lub ich fragmentów. Klauzula GROUP BY umożliwia grupowanie wyników względem zawartości wybranych kolumn. W wyniku jej działania uzyskujemy podział wierszy tablicy na dowolne grupy. W pewnym sensie jej działanie jest podobne do działania operatora DISTINCT, ponieważ po jej zastosowaniu zwracany jest pojedynczy wynik dla każdej grupy (listing 6.14).

Listing 6.14. Klauzula GROUP BY użyta do wyeliminowania duplikatów — skoro dane są grupowane według identyfikatorów osób, czyli zamówienia złożone przez tę samą osobę dodawane są do jednej grupy, to w wyniku nie może pojawić się kilka razy ten sam identyfikator klienta

```
SELECT customer_id

FROM orderinfo

GROUP BY customer_id;
```

```
+-------------+
| customer_id |
+-------------+
| 3           |
| 8           |
| 13          |
| 15          |
+-------------+
```

Jeżeli jednak w zapytaniu użyjemy jednocześnie funkcji grupującej, to ta funkcja zostanie wywołana niezależnie dla każdej grupy zdefiniowanej w klauzuli GROUP BY. W bazie test informacje o

zamówieniach przechowywane są w tabeli orderinfo, a poszczególne pozycje zamówienia — w tabeli orderline (dzięki temu w ramach każdego zamówienia klient może kupić dowolną liczbę najróżniejszych towarów). Pierwsze zapytanie (listing 6.15) zwraca liczbę wszystkich sprzedanych towarów, drugie (listing 6.16) rozbija tę liczbę na poszczególne zamówienia.

Zapamiętaj — jeżeli w klauzuli SELECT występują dowolne funkcje grupujące, to wszystkie nazwy kolumn i wyrażenia, które NIE SĄ argumentami tych funkcji, muszą być wymienione w klauzuli GROUP BY. Innymi słowy, w takich zapytaniach w klauzuli SELECT mogą występować tylko i wyłącznie wyrażenia, które są wymienione w klauzuli GROUP BY, chyba że są one argumentami dowolnej funkcji agregującej.

Jak wiemy, język SQL umożliwia, poprzez zastosowanie klauzuli ORDER BY, porządkowanie wyników zapytania. Możliwe jest też sortowanie wyników na podstawie wyniku funkcji grupujących (listing 6.18).

Listing 6.18. Uporządkowany wynik poprzedniego zapytania

```
SELECT orderinfo_id, SUM(quantity)

FROM orderline

GROUP BY orderinfo_id

ORDER BY SUM(quantity) DESC;
```

Do wyjaśnienia w domu: Operator ROLLUP

Klauzula Having.

Klauzula HAVING

Język SQL dostarcza jeszcze jedną metodę filtrowania wyników zapytań — jeżeli grupujemy wyniki (a więc używamy klauzuli GROUP BY), możemy sprawdzić, czy te grupy wierszy spełniają jakiś warunek. Wiesz już, że po zastosowaniu klauzuli WHERE wyniki zapytania najpierw są filtrowane, a potem grupowane. Klauzula HAVING, tak jak WHERE, umożliwia określenie testu logicznego, ale w jej przypadku będzie on zastosowany do grup, a nie pojedynczych wierszy. Testy logiczne zawarte w klauzuli HAVING wykonywane są na całych grupach, a nie na pojedynczych rekordach. Tak więc klauzula ta służy do wybierania interesujących nas grup, a klauzula WHERE — interesujących nas wierszy. Warunek umieszczony w klauzuli HAVING wyrażony jest za pomocą dowolnej funkcji grupowej (listing 6.23).

Listing 6.23. Informacje o zamówieniach, których wartość przekroczyła 30. Iloczyn quantity*sell_price jest wyliczany dla każdego sprzedanego towaru, a następnie wyliczana jest suma wartości dla każdej grupy, czyli w tym przypadku dla każdego zamówienia. Na końcu klauzula HAVING usuwa z wyniku te grupy (zamówienia), których wartość nie przekroczyła 30

```
SELECT orderinfo_id, SUM(quantity*sell_price)

FROM orderinfo JOIN orderline USING (orderinfo_id)

JOIN item USING (item_id)

GROUP BY orderinfo_id

HAVING  SUM(quantity*sell_price)> 30;
```

Klauzule HAVING i WHERE mogą wystąpić w tym samym zapytaniu — w takim przypadku najpierw będzie zastosowany test z klauzuli WHERE, a następnie — z klauzuli HAVING (listing 6.24).

Listing 6.24. Wyniki poprzedniego zamówienia ograniczone do zamówień złożonych w pierwszej połowie 2000 roku

```
SELECT orderinfo_id, SUM(quantity*sell_price)

FROM orderinfo JOIN orderline USING (orderinfo_id)

JOIN item USING (item_id)

   WHERE date_placed BETWEEN '2000-01-01' AND '2000-06-31'

   GROUP BY orderinfo_id

   HAVING SUM(quantity*sell_price)> 30;
```

Kolejność wykonywania klauzuli zapytań

Skoro poznałeś wszystkie klauzule instrukcji SELECT, powinieneś wiedzieć, kiedy są one wykonywane przez serwery bazodanowe. Logiczna kolejność wykonywania zapytania zawierającego omówione dotychczas klauzule jest następująca:

1. Jako pierwsza wykonywana jest klauzula FROM. Jeżeli zapytanie odwołuje się do wielu tabel, są one kolejno ze sobą złączane.

2. Otrzymany w ten sposób zbiór pośredni jest filtrowany na podstawie warunku logicznego umieszczonego w klauzuli WHERE. Tylko te wiersze, dla których jest on prawdziwy, trafiają do kolejnego zbioru pośredniego.

3. Następnie wykonywana jest klauzula GROUP BY, czyli grupowane są tylko przefiltrowane wiersze.

4. Utworzone grupy są filtrowane poprzez porównanie ich z warunkiem umieszczonym w klauzuli HAVING.

5. Wybrane w poprzednim punkcie wiersze są zwracane, czyli wykonywana jest klauzula SELECT.

6. Następnie wiersze są sortowane, czyli wykonywana jest klauzula ORDER BY.

W konstrukcji zapytań możemy wyszczególnić 6 głównych bloków logicznych.

```
(Step 5)   SELECT   -- określanie kształtu wyniku, selekcja pionowa
(kolumn)
(Step 1)   FROM     -- określenie źródła (źródeł) i relacji między nimi
(Step 2)   WHERE    -- filtracja rekordów
(Step 3)   GROUP BY -- grupowanie rekordów
(Step 4)   HAVING   -- filtrowanie grup
(Step 6)   ORDER BY -- sortowanie wyniku
```

Łączenie tabel.

Złączenia zewnętrzne.

Złączenia lewostronne, prawostronne, obustronne.

Więzy integralności.

Łączenie wyników zapytań.

Podzapytania.

Podzapytania.

Instrukcja INSERT.

Instrukcja UPDATE.

Instrukcja DELETE.

[i] http://www.sqlpedia.pl/kurs-sql/

[ii] http://webmaster.helion.pl/index.php/kursmysql-pobieranie-danych-z-pojedynczych-tabel