JĘZYK SQL

Język SQL został opracowany przez korporację IBM w laboratorium badawczym w San Jose we wczesnych latach 70. Prototyp SQL-a, zaprezentowany na konferencji ACM w roku 1974 nosił początkowo miano SEQUEL (Strukturalny Angielski Język Zapytań; ang. Structured English Query Language). Nazwa ta została później skrócona do SQL [5] [7].

Mimo, że iż ojcem SQL-a była firma IBM, pierwszej implementacji sprzętowej dostarczyła korporacja Oracle (wówczas nosząca nazwę Relational Software Inc.). Wczesne implementacje komercyjne koncentrowały się na średniej wielkości systemach, działających pod systemem operacyjnym UNIX.

W roku 1986 instytut ANSI dokonał pierwszej standaryzacji SQL-a (stąd nazwa SQL-86). Międzynarodowa wersja tego standardu, opracowana przez ISO, pojawiła się w roku 1987.

Znacznie ulepszona wersja SQL-a pojawiła się w 1989 roku pod nazwą SQL-89. Większość stosowanych dziś relacyjnych systemów baz danych obsługuje przynajmniej tę wersję języka. W 1992 roku istniejący standard poddano kolejnemu unowocześnieniu (SQL-92), dzięki czemu SQL zyskał szereg nowych możliwości. Ponieważ SQL-92 jest rozszerzeniem SQL-89, starsze systemy zarządzania bazami danych mogły po niewielkich modyfikacjach pracować w nowym standardzie.

Aż do października 1996, producenci oprogramowania systemów baz danych mogli przedstawiać swoje produkty instytutowi NIST (Narodowy Instytut Standardów i Technologii) w celu określenia ich zgodności ze standardami SQL. Proces testowania i certyfikacji stanowił silny bodziec do trzymania się obowiązujących norm. Pomimo, iż zaniechanie tego procesu oszczędzi producentom systemów baz danych wielu wydatków, nie wiadomo jaki będzie jego wpływ na zgodność tworzonego obecnie oprogramowania ze wzorcowym SQL-em.

Strukturalny język zapytań obecna nazwa ***SQL (Structured Query Language)*** jest językiem wysokiego poziomu do manipulowania danymi. Język ten zawiera małą liczbę instrukcji o bardzo dużych możliwościach. Język SQL różni się w szczegółach, w zależności od tego z jakiego pakietu narzędzi korzystamy. Ogólnie jest jednak przestrzegany standard ISO języka SQL zatwierdzony w 1992 roku. Obecna specyfikacja tego języka jest czymś więcej niż tylko językiem zapytań. Określenie język baz danych jest o wiele trafniejsze.

Użytkownik, który ma mało doświadczenia w przetwarzaniu danych lub nie ma go w cale, może szybko nauczyć się podstawowych konstrukcji języka SQL, a ekspert od przetwarzania danych może znaleźć w tym języku pełny zestaw potrzebnych mu narzędzi. Jest to więc język, którego używają zarówno zwykli użytkownicy, jak i fachowcy. Język SQL jest przeznaczony dla różnych grup odbiorców:

- zwykłych użytkowników,
- ekspertów od przetwarzania danych,
- twórców systemów baz danych,
- administratorów baz danych,
- kadry kierowniczej,
- personelu centrum informacji.

Można użyć języka SQL jako standardowego narzędzia umożliwiającego dostęp do danych w różnorodnych środowiskach z różnym sprzętem komputerowym i różnymi systemami operacyjnymi. Środowiska te obejmują wszystkie rodzaje komputerów - od komputerów osobistych po minikomputery i duże komputery. SQL jest pojęciowo łatwym narzędziem dla użytkownika przy wybieraniu, operowaniu i kontrolowaniu danych.

SQL pozwala użytkownikom definiować strukturę bazy danych, wprowadzać dane do bazy, modyfikować dane i manipulować tymi danymi.

SQL jest zaimplementowany w ponad 140 produktach. Obecnie wszystkie liczące się narzędzia umożliwiają tworzenie aplikacji baz danych w architekturze Klient-Serwer z wykorzystaniem języka SQL. Wtedy na Serwerze plików umieszczona jest baza danych w postaci jednego wielkiego pliku razem z generatorami, wyzwalaczami, procedurami napisanymi w języku SQL i system zarządzający tą bazą danych. Klient-Serwer jest architekturą oprogramowania, w której dwa procesy współdziałają ze sobą jako nadrzędny i podrzędny. Proces klienta zawsze rozpoczyna współdziałanie od zgłoszenia zapotrzebowania. Proces serwera odpowiada na zgłoszone zapotrzebowanie. Odbywa się to w ten sposób, że stacja klienta przesyła zapytanie do bazy danych na ogół w języku SQL, a serwer przekazuje z powrotem gotową odpowiedź.

System zarządzający bazą danych zapewnia również współdzielenie danych, zachowanie integralności danych , bezpieczeństwo danych, oraz chroni bazę danych przed naruszeniem referencyjnych reguł integralności. Na stacjach roboczych zainstalowane są aplikacje klientów odpowiedzialne za generowanie zapytań do bazy danych i obsługę interfejsu użytkownika.

SKŁADNIA INSTRUKCJI

Do opisu składni instrukcji języka SQL przyjęto następującą konwencję. Instrukcje i słowa kluczowe będą pisane dużymi literami. Nawiasy „< >" oznaczają informację wprowadzaną przez programistę. Nawiasy „[ ]" wskazują (zaznaczają) opcjonalną część instrukcji, tzn. część instrukcji umieszczona w tych nawiasach może wystąpić lub nie. Trzy kropki „..." oznaczają, że w tym miejscu można umieścić kilka podobnych składowych danej instrukcji. Znak „/" dzielenia oznacza, że wpisuje się albo, to co jest podane przed znakiem lub to co jest podane po znaku.

INSTRUKCJA CREATE

Do tworzenia bazy danych i jej obiektów używa się instrukcji CREATE. Poniżej podano podstawowe definicje i przykłady tworzenia bazy danych tabel, i indeksów.

Tworzenie tabel

Nową tabelę tworzy się poleceniem:

```
CREATE   TABLE   <nazwa tabeli>
( <nazwa atrybutu>   <typ danych>
[, <nazwa atrybutu>   <typ danych> ...]);
```

Przykład 3 Tworzenie tabeli Wyroby

```
CREATE TABLE Wyroby
(
   Id_Wyrobu          INTEGER   PRIMARY   KEY,
   Nazwa_Wyrobu       VARCHAR(32)   NOT   NULL,
   Cena_Wyrobu        DECIMAL (12,2),
   Jedn_miary         CHAR (5),
   Data_Zapisu        DATĘ);
```

Po wykonaniu tego polecenia zostanie utworzona tabela Wyroby z atrybutami Id_Wyrobu (klucz główny) , Nazwa_Wyrobu, Cena_Wyrobu, Jedn_miary i data zapisu.

Przykład 4 Tworzenie struktury tabeli „Firmy"

```
CREATE TABLE Firmy
(
      Id_Firmy          INTEGER   NOT NULL,
      Firma             CHAR(20)  NOT NULL,
      Nazwa_firmy       CHAR(80),
      Kod_poczt         Char(6),
      Miasto            Char(40),
      Ulica             DS20,
      Nip               Char(13),
      Id_Bank           DS20,
      Nr_Konta          Char(40),
      Telefon           DS20,
      Fax               DS20,
      Data_Zmiany       Date,
      Uwagi             CHAR(60),
      Użytkownik        USER,
      CONSTRAINT  PK_Firmy  PRIMARY KEY (Id_Firmy)
);
```

Po wykonaniu tego polecenia zostanie utworzona tabela Firmy z kluczem głównym Id_Firmy i indeksem o nazwie PK_Firmy.

Kolumny obliczane

Instrukcja CREATE TABLE umożliwia także definiowanie kolumn obliczanych. Poniżej przedstawiamy tabelę Towary rozszerzoną o kolumnę obliczaną Podatek

Przykład 5 Tworzenie tabeli Towary z kolumną obliczaną Podatek

```
CREATE TABLE Towary
(
   Id_Towaru          INTEGER PRIMARY KEY,
   Nazwa_Towaru       VARCHAR (32) NOT NULL,
   Cena_Tow           DECIMAL (12,2),
   Vat                DECIMAL (6,4),
   jedn_miary         CHAR (5),
   ilosc_Tow          DECIMAL (10,3),
   Podatek            COMPUTED BY (Vat *Cena_Tow),
   Data_Zapisu        DATE
);
```

Powyższe polecenie utworzy tabelę o nazwie towary z kolumną obliczaną podatek. Kolumna ta będzie aktualizowana po każdej zmianie wartości kolumn Vat i Cena_Tow. Ponieważ w kolumnie obliczanej nie został podany typ danych, Interbase ustala go na podstawie typu z kolumn wykorzystywanych do obliczeń. Definicje kolumn wykorzystywanych przez kolumnę obliczaną muszą oczywiście być na liście kolumn polecenia CREATE TABLE i wystąpić przed nią jak w powyższym przypadku.

Podobne możliwości daje instrukcja SELECT, czy perspektywa. Jednak w tym wypadku wpisujemy obliczenia bezpośrednio do definicji tabeli. Jest to dobry sposób na ustalenie pewnych zależności obliczeniowych. Wtedy bowiem aplikacje i inne obiekty SQL mogą po prostu pobierać obliczone wartości z kolumny obliczanej.

IV.2.6 Tworzenie indeksów

Podstawowym sposobem uporządkowania danych, a co zatem idzie szybkiego ich wyszukiwania jest poindeksowanie tablicy. Do indeksowania służy instrukcja:

```
CREATE [UNIQUE] [ASC/DESC] INDEX < nazwa_indeksu >
ON < nazwa tablicy >
(                                                    ( 11 )
    < nazwa kolumny > [DESC]
  [, < nazwa kolumny > [DESC] ... ]
);
```

Klauzula UNIQUE określa czy indeks ma być unikalny, DESC czy uporządkowanie ma być malejące. Domyślnie uporządkowanie jest rosnące.

Przykład 7 Tworzenie indeksu do tabeli Klienci

```
CREATE UNIQUE INDEX FK_Klienci ON
Klienci   (Id_Klienta, Id_Rejonu);
```

Po wykonaniu tego polecenia zostanie utworzony unikalny indeks o nazwie „FK_Klienci" związany z tabelą Klienci z uporządkowaniem wg klucza Id_Klienta+Id_Rejonu.

DODAWANIE MODYFIKACJA I USUWANIE DANYCH

Tabele można usunąć z bazy danych poleceniem:

```
DROP TABLE < Nazwa tablicy >
```

Przykład 8 Usuwanie tabeli

```
DROP TABLE Towary ;
```

Powyższe polecenie usuwa z bazy danych ostatecznie tabelę Towary. Indeks usuwa się z bazy danych poleceniem:

```
DROP INDEX < Nazwa indeksu >
```

Przykład 9 Usuwanie indeksu

```
DROP INDEX FK_Towary
```

Powyższe polecenie usunie z bazy danych indeks FK_Towary.

1V.3.1 Wprowadzanie nowych wierszy do tabel

Dane do tabel wprowadza się instrukcją:

```
INSERT   INTO   <nazwa tabeli>
[ (lista atrybutów ) ]
VALUES (<Lista wartości>);
```

Listę atrybutów można pominąć, gdy ich wartości podajemy w ustalonym porządku zgodnym ze strukturą tabeli.

Przykład 10 Wprowadzamy do tablicy towary nowy wiersz

```
INSERT   INTO   Towary
(
    id_Towaru, Nazwa_Towaru, Cena_Tow,
   jedn_miary, ilosc_Tow, Data_Zapisu
)
VALUES   ( 12341, "KAWA", "pacz", 8.73, 1000, "1998.01.15" );
```

Po wykonaniu tej instrukcji do tabeli Towary zostanie wstawiony nowy wiersz o wartościach podanych w ostatnim nawiasie instrukcji.

Modyfikowanie danych

Do zmiany wartości jednego lub kilku wierszy tabeli służy instrukcja:

```
UPDATE <nazwa tabeli>
SET <nazwa atrybutu>   =   <wyrażenie>
        [,<nazwa atrybutu>   =   <wyrażenie> …]
[WHERE   <predykat>];
```

Przykład 11 Zmiana ceny towarów

```
UPDATE   Towary
SET   Cena_Tow=1.05*Cena_Tow
WHERE Cena_Tow > 2500 ;
```

Wykonanie tej instrukcji spowoduje zmianę cen o *5%,* wszystkich tych towarów w tabeli Towary, których poprzednia cena była większa od 2500 zł.

Usuwanie danych

Usunięcie danych z tabeli wykonuje się instrukcją:

```
DELETE FROM   <nawa tabeli>
WHERE   <warunek>
```

Przykład 12 Usuwanie wierszy z tabeli towary

```
DELETE   FROM   Towary WHERE
Cena_Tow> 10000;
```

Polecenie to usunie bezpowrotnie wszystkie wiersze tabeli towary których cena jest większa od 10000.

POLECENIA JĘZYKA SQL

Język SQL składa się z kilkunastu różnych komend. Najważniejsze z nich to polecenie SELECT zwane zapytaniem. Polecenie to umożliwia pobieranie danych z bazy danych i ich prezentację. Składnia tego polecenia jest następująca:

```
SELECT   [ DISTINCT ] * / < atrybut > [,< atrybut > ...]
                                [ ,< funkcja agregująca > ... ] [ ,<
                                wyrażenie > ... ]
FROM < nazwa tabeli > [,< nazwa tabeli > ...] / < złączone tabele >
[ WHERE   < predykat wyboru wierszy> ]
[ GROUP BY <atrybut> [,<atrybut> ...]
[ HAVING   < predykat wyboru grup > ]
[ UNION / INTERSECT / EXCEPT ]
[ORDER BY <atrybut> [ASC / DESC] [,<atrybut> ...] [ASC / DESC]]
```

Po słowie *SELECT* należy podać atrybuty tabel, które mają być pokazane lub wyrażenia określające wartości atrybutów wyliczanych np. jeżeli w tabeli są atrybuty ilość i cena to można podać Ilosc * Cena co oznacza wartość. Jeżeli zamiast listy atrybutów podamy gwiazdkę „ * " to wyrażenie SELECT zwróci nam wszystkie atrybuty jakie występują w tabeli.

Klauzula ***FROM*** określa z jakich tabel będą pobierane dane. Można w tym przypadku podawać nazwy tabel oddzielone przecinkami lub złączenia tabel. Tabele łączy się operatorem ***JOIN*** podając po słowie *ON,* wg których atrybutów łączymy tabele. Mogą to być następujące złączenia:

- *NATURAL* - stanowi złączenie tych wierszy, dla których wartość klucza obcego jednej tabeli jest tożsama z kluczem głównym innej tabeli.
- *Cross* - wszystkie możliwe kombinacje wierszy łączonych tabel. Oznacza to, że gdy np. mamy dwie tabele po 100 wierszy każda, to otrzymamy tabelę wynikową zawierająca 10000 wierszy.
- **Wewnętrzne** (INNER) - tabela wynikowa zawiera tylko pasujące do siebie wiersze tzn. takie dla których wartość przynajmniej jednego atrybutu w obydwu tabelach jest taka sama.
- **Lewostronne zewnętrzne** (LEFT OUTER) - tabela wynikowa składa się ze wszystkich wierszy jednej tablicy i pasujących (tzn. mających taką samą wartość przynajmniej jednego atrybutu) do nich wierszy drugiej tablicy.
- **Prawostronne zewnętrzne** (RIGHT OUTER) - jest odwrotnością złączenia lewostronnego zewnętrznego.
- **Pełne zewnętrzne** (FULL OUTER) - stanowi kombinację złączenia lewo i prawostronnego.

Klauzula ***WHERE*** precyzuje warunki selekcji wierszy. Predykat wyboru wierszy może być wyrażeniem warunkowym określonym przez atrybuty i operatory języka SQL.

Klauzula ***GROUP BY*** umożliwia podział wierszy na mniejsze grupy. Do każdej z grup można zastosować jedną z tzw. funkcji grupowych.

Klauzula ***HAVING*** może wystąpić tylko razem z poleceniem GROUB BY. Precyzuje ona warunki grupowej selekcji wierszy.

UNION to suma wierszy, ***EXCEPT*** to różnica, a ***INTERSECT*** to część wspólna wierszy z dwóch tabel spełniających warunek zgodności,.

Klauzula ***ORDER BY*** określa sposób sortowania wyników zapytania przy czym:

- ***ASC*** - oznacza porządek rosnący,
- ***DESC*** - *oznacza* porządek malejący.

W relacyjnych bazach danych, dane są przechowywane w tabelach. Przykładowa tabela „ ***Kraje*** " będzie wiązała (przechowywała relacje) między nazwą kraju, jego obszarem, liczbą ludności i kontynentem, na którym leży.

W celu przyswojenia sobie struktury polecenia SELECT utwórzmy kilka zapytań do tej tabeli. Wyniki zapytań dla wszystkich dalszych przykładów będą podawane w tabelach poniżej zapytania SELECT.

KRAJE

Nazwa kraju	Obszar (w tys.km^2)	Ludność (mln)	Kontynent
Argentyna	2766	26	Ameryka
Polska	313	39	Europa
USA	9363	230	Ameryka
Kanada	9976	27	Ameryka
Albania	28	3	Europa
Francja	547	55	Europa
Angola	1246	9	Afryka
Kenia	582	21	Afryka

Tabela 27 Wybrane „Kraje'

Przykład 13 Wybór krajów

W przykładzie tym należy sformułować zapytanie jakie są nazwy i obszar tych krajów których ludność jest większa od 30 min. Zapytanie to w języku SQL ma postać:

```
SELECT Nazwa, Obszar FROM Kraje
WHERE  Ludność > 30;
```

Nazwa	***Obszar***
Polska	313
USA	9363
Francja	547

Jak można zauważyć w przykładzie ma się do czynienia z połączeniem selekcji i projekcji, w sensie działań algebry relacyjnej. Z tabeli kraje zostały wyselekcjonowane tylko te wiersze, które spełniły predykat „ ludność > 30 " oraz nastąpiła projekcja tylko tych atrybutów, które były wymienione po słowie SELECT. Zauważ, że nazwy kolumn i nazwy tabel nie mają spacji. Muszą być wprowadzane jako jedno słowo. Polecenie SELECT kończy się średnikiem ';'.

Przykład 14 Zapytanie o nazwę i obszar

W przykładzie tym należy sformułować zapytanie jakie są nazwy i obszar tych krajów których obszar jest mniejszy od 500 tys. km i tych co leżą na kontynencie Afryka. Zapytanie to w języku SQL ma postać:

```
SELECT Nazwa, Obszar FROM Kraje
 WHERE Obszar < 500
 UNION
SELECT Nazwa, Obszar FROM Kraje
 WHERE Kontynent = 'Afryka';
```

Nazwa	***Obszar***
Albania	28
Angola	1246
Kenia	582
Polska	313

Jak można zauważyć w przykładzie tym ma się do czynienia z działaniem algebry relacyjnej jaką jest UNION, czyli sumą wierszy z dwóch zapytań SELECT. Dalsze przykłady związane z grupowaniem wierszy tabel zostaną podane po wprowadzeniu funkcji języka SQL.

Funkcje języka SQL

W Poleceniu SELECT języka SQL można używać funkcji określonych na atrybutach i wyrażeniach tabel, które umieszczono w poniższej tabeli.

Nazwa funkcji	Opis funkcji
AVG	Obliczanie średniej wartości atrybutów
CAST	Zmiana typu danych
COUNT	Oblicza liczbę wierszy podlegających selekcji
MAX	Obliczanie wartości maksymalnej dla danej kolumny
MIN	Obliczanie wartości minimalnej w danej kolumny
TRIM	Obcięcie spacji
SUM	Obliczanie sumy wartości atrybutu liczbowego w kolumnie
UPPER	Zamiana liter z małych na duże
LOWER	Zamiana liter z dużych na małe

Tabela 28 Funkcje języka SQL

Funkcje *AVG, COUT, MAX, MIN, SUM* należą do tzw. funkcji grupowych, czyli można ich używać w warunkach grupowej selekcji. Jako przykład takiego zastosowania rozważmy polecenie grupujące wiersze według kontynentów i obliczające sumę obszaru krajów na danym kontynencie.

Przykład 15 Zapytanie z klauzulą GROUP BY

```
SELECT Kontynent, SUM(Obszar) Suma
FROM Kraj
GROUP  BY  Kontynent;
```

W pierwszym wierszu zapytania wprowadzono tzw. ALIAS tzn. nazwę zastępczą „Suma" dla wartości pola SUM(Obszar). W wyniku otrzymujemy:

Kontynent	***Suma***
Afryka	1828
Ameryka	22105
Europa	888

Przykład 16 Zastosowanie funkcji COUNT

W tym przykładzie wykorzystano funkcję COUNT do obliczenia liczby krajów na danym kontynencie. Wyniki zapytania posortowano wg nazw kontynentów.

```
SELECT Kontynent, COUNT(Nazwa) Ilosc
FROM Kraje
GROUP  BY Kontynent
ORDER BY Kontynent;
```

Kontynent	***Ilosc***
Afryka	2
Ameryka	3
Europa	3

Razem z klauzulą GROUP BY można stosować grupowy warunek selekcji HAVING. Klauzula ta ogranicza prezentację grup do tych, które spełniają określony w niej warunek.

Przykład 17 Zastosowanie funkcji COUNT

Jako przykład obliczmy średnią liczbę ludności, ale tylko dla kontynentów, z liczbą krajów większą od dwu. Jak można zauważyć predykat podany w klauzuli HAVING akceptuje grupy krajów o liczbie większej od dwa. Stosowne zapytanie i wynik zapytania podano poniżej.

```
SELECT Kontynent, A VG(Ludnosc) Średnia FROM Kraje
GROUP BY Kontynent HAVING COUNT (Nazwa) > 2
ORDER BY Kontynent;
```

Kontynent	***Średnia***
Ameryka	**94**
Europa	**32**

Operatory relacyjne

Istnieje sześć operatorów relacyjnych w języku SQL. Operatory te podano w tabeli poniżej. W dalszej części skryptu będzie opisane jak z nich korzystać.

Operator	Znaczenia
=	Równy
!= , <>	Nierówny
<	Mniejszy
<=	Mniejszy lub równy
>	Większy
>=	Większy lub równy

Tabela 29 Operatory relacyjne

W języku SQL można ze sobą porównywać liczby, dane tekstowe, dane typu daty i czasu. Porównania te odbywają się wg zasady:

- porównanie danych liczbowych oparte jest na zwykłej kolejności liczb,
- porównanie danych tekstowych oparte jest na kolejności alfabetycznej,
- porównanie dat i godzin oparte sana porządku chronologicznym.

Przykład 18 Zastosowanie operatorów relacyjnych

Poniższy przykład przedstawia zastosowanie operatorów relacyjnych i funkcji grupowych.

```
SELECT Kontynent, Max(Ludnosc)
FROM Kraje
WHERE Obszar >= 230 and Ludność <= 55
GROUP BY Kontynent
HAYING MIN(Obszar)> 100
```

Kontynent	*Max*
Afryka	21
Ameryka	27
Europa	55

Operatory logiczne

W języku SQL istnieją trzy operatory logiczne:

Operator	**Znaczenia**
NOT	Negacja
AND	Koniunkcja
OR	Alternatywa

Tabela 30 Operatory logiczne

Działanie tych operatorów jest takie same jak w działaniach logicznych powszechnie stosowanych. Operator **AND** łączy dwa lub więcej warunków i wyświetla wiersz tylko wtedy, gdy dane w tym wierszu spełniają wszystkie przedstawione warunki. Operator OR łączy dwa łub więcej warunków, ale zwraca wiersz, jeżeli dowolny z nich (warunków) jest spełniony. Aby zobaczyć wszystkie kraje, których obszar jest większy od 350, a ludność mniejsza od 55 wykorzystaj następujące warunek: *WHERE Obszar >= 350 OR Ludność <= 55* Operatory **AND** i OR można łączyć ze sobą, np.:

WHERE Obszar >= 350 OR Ludność <= 55 AND *Ludność > 15*

Najpierw SQL wyszukuje kraje, których ludność zawiera się w przedziale (15,55] , a potem biorąc tą nową listę wierszy, dodaje do niej kraje o obszarze większym od 350.

Przykład 19 Zastosowanie operatorów logicznych

Poniżej podano przykład zapytania z operatorami relacyjnymi OR i NOT. W zapytaniu tym występują dwa warunki selekcji: zwykły WHERE najpierw realizowany i grupowy HAVING. W klauzuli HAVING występuje operator NOT, który zmienia wartość logiczną wyrażenia na przeciwną. Wynik zapytania podano w tabeli poniżej.

```
SELECT Kontynent, Max(Ludnosc)
FROM Kraje
 WHERE Obszar >= 230 OR Ludność <= 55
GROUP BY Kontynent
HAYING NOT MIN (Obszar) > 100
```

Kontynent	***Max***
Europa	55

Operatory arytmetyczne

W języku SQL można używać następujących operatorów arytmetycznych.

Operator	**Znaczenia**
+	Dodawanie
-	Odejmowanie
*	Mnożenie
/	Dzielenie

Tabela 31 Operatory arytmetyczne

Wynik działania tych operatorów na danych liczbowych jest oczywisty. Natomiast do danych typu ***DATA TIME*** (data, czas) i ***INTERVAL*** (przedział czasu w dniach) typ wyniku tych działań określa tablica podana poniżej:

Postać wyrażenia	**Typ wyniku**
DATETIME - DATETIME	INTERVAL
DATETIME + DATETIME	DATETIME
DATETIME - INTERVAL	DATETIME
DATETIME + INTERVAL	DATETIME
INTERVAL + INTERVAL	INTERVAL
INTERVAL* Liczba	INTERYAL
INTERYAL / Liczba	INTERYAL

Tabela 32 Typ wyniku działań dla daty i czasu

Operatory specjalne języka SQL

Oprócz typowych operatorów występujących w innych językach programowania, w języku SQL wprowadzono operatory specjalne podane poniżej.

Operator	**Znaczenia**
BETWEEN	W określonym przedziale
LIKE	Podobny do
IN	Należy do listy wartości
IS NULL	Jest równy wartości pustej NULL
ANY	Dla jakiejś wartości
ALL	Dla każdej wartości

Tabela 33 ***Operatory specjalne języka SQL***

Przykład 20 Zastosowanie operatora BETWEEN

Z tabeli kraje wybrać państwa, których obszar zawiera się w przedziale od 300 do 550 mln. Zapytanie ma postać:

```
SELECT Nazwa, Obszar
FROM Kraje
WHERE Obszar BETWEEN 300 and 550
ORDER BY OBSZAR;
```

Nazwa	*Obszar*
Polska	313
Francja	547

Używanie operatora LIKE

Przykład 21 Zastosowanie operatora LIKE w klauzuli WHERE

Powiedzmy, że z tabeli *Kraje* chcielibyśmy wybrać wszystkie te kraje, których nazwy zaczynają się na „ A ".

SELECT Nazwa, Ludność
FROM Kraje
WHERE Nazwa LIKE 'A % *

Wynikiem zapytania jest tabela.

Nazwa	*Ludność*
Argentyna	26
Albania	3
Angola	9

Znak procenta *{%)* został użyty do zastąpienia ciągu dowolnych znaków (cyfry, litery) które mogą pojawić się po "A".

Aby znaleźć te kraje, których nazwy kończą się na „ a ", użyj '%a' Aby znaleźć kraje, które mają "i" w środku nazwy, użyj '%i%'. Znak '%' może być używany zamiast dowolnych innych znaków na określonej pozycji, w stosunku do podanych znaków. Polecenie **NOT LIKE** wyświetli wiersze, które nie odpowiadają temu opisowi.

Przykład 22 Zastosowanie operatora LIKE

Z tabeli Kraje wybrać te państwa, których nazwa zawiera literę „ i " i uporządkować wg liczby ludności. Odpowiednie zapytanie ma postać:

SELECT Nazwa, Ludność
FROM Kraje
WHERE Nazwa like ' % i % '
ORDER BY Ludność DESC;

Jego wynikiem jest poniższa tabela.

Nazwa	***Ludność***
Kenia	21
Albania	3

Przykład 23 Zastosowanie złożonej postaci operatora LIKE

Z tabeli kraje wybrać państwa, których nazwa rozpoczyna się od dużej litery „ A ", oraz wewnątrz nazwy mają literę „ g ". Odpowiednie zapytanie ma postać:

```
SELECT Nazwa, Ludność
FROM Kraje
WHERE Nazwa like 'A%g% '
ORDER BY Nazwa;
```

Wynikiem zapytania jest poniższa tabela

Nazwa	Ludność
Angola	9
Argentyna	26

ZAPYTANIA ZAGNIEŻDŻONE

Zapytania języka SQL mogą być w sobie zagnieżdżone. W ten sposób wynik jednego zapytania może być użyty w warunkach selekcji innego zapytania. Zapytanie zagnieżdżone wykonywane jest przed zapytaniem zewnętrznym. Ogólny format zagnieżdżenia zapytań jest następujący:

```
SELECT   [ DISTINCT ] * / < atrybut > [,< atrybut > ...]
FROM < nazwa tabeli > [,< nazwa tabeli > ...]
WHERE   atrybut =   (SELECT   <atrybut>
                          FROM   <RELACJA>
                         WHERE   <warunek>;
[ GROUP BY <atrybut> [,<atrybut> ...]
[ HAVING   < predykat wyboru grup > ]
[ORDER BY < atrybut > [,< atrybut >…];
```

Zapytania mogą być zagnieżdżane w sobie wielokrotnie. Reguły zagnieżdżania zapytań są następujące:

- podzapytanie w nawiasach piszemy po prawej stronie zapytania zewnętrznego,
- w podzapytaniu nie używa się klauzuli ORDER BY,
- klauzula ORDER BY może wystąpić tylko jako ostatnia klauzula zapytania zewnętrznego,
- w podzapytaniu można używać operatorów UNION i MINUS,
- liczba i typ atrybutów występujących w klauzuli SELECT podzapytania muszą być zgodne z liczbą i typem atrybutów użytych w warunku zapytania zewnętrznego,
- najpierw są wykonywane zapytania najbardziej zagnieżdżone kolejno do najbardziej zewnętrznego.

Przykład 24 Zapytanie zagnieżdżone

W przykładzie zostaną wybrane kraje leżące na tym samym kontynencie co Kanada.
Zapytanie ma postać:

```
SELECT Nazwa, Obszar
FROM Kraje
WHERE Kontynent = ( SELECT Kontynent
                    FROM Kraje
                    WHERE Nazwa = 'Kanada')
ORDER BY Nazwa;
```

Nazwa	***Obszar***
Argentyna	2766
Kanada	9976
USA	9363

Podzapytania dotyczące wielu wierszy

Jeżeli podzapytanie wyznacza listę wartości to w zapytaniu zewnętrznym musi być użyty jeden z następujących operatorów SQL: IN, ANY lub ALL, Rozważmy tabelę o nazwie ***„Programy",*** która będzie używana w przykładach polecenia SELECT. W tabeli tej podano programy różnych rodzajów i ich przykładowe ceny.

Nazwa	Rodzaj	Cena
Windows	S.O.	500
Word	Edytor	450
Linux	S.O.	200
Star	Edytor	220

Paradox	Baza	900
DeltaCad	CAD	1800
Delphi	Baza	3100
Designer	CAD	800
Optima C++	Baza	1600
AutoCad	CAD	6000

Tabela 34 Relacja „Programy"

Przykład 25 Podzapytanie z operatorem IN

Napisać zapytanie pozwalające wybrać najdroższe programy danego rodzaju. W poniższym przykładzie wyszukujemy maksymalne ceny w poszczególnych rodzajach programów (zapytanie wewnętrzne). Otrzymana lista cen zostaje następnie użyta w warunku selekcji zapytania zewnętrznego. Zapytanie to z uporządkowaniem według cen ma postać:

```
SELECT Nazwa, Rodzaj, Cena
FROM Programy
WHERE Cena IN (SELECT max(Cena)
                    FROM Programy GROUP  BY
                    Rodzaj)
ORDER BY Cena;
```

Nazwa	*Rodzaj*	*Cena*
Word	Edytor	450
Windows	S.O.	500
Delphi	Baza	3100
AutoCad	CAD	6000

Typ wartości wyznaczonych przez podzapytanie musi być zgodny z typem atrybutów użytych po lewej stronie operatora IN.

Przykład 26 Podzapytanie z operatorem ANY

Operator ANY umożliwia sprawdzenie, czy warunek jest spełniony dla choćby jednego elementu listy wyboru uzyskanej z podzapytania.

Poniższe polecenie wybiera z tabeli te programy których cena jest większa przynajmniej od jednego programu rodzaju Baza, Czyli od ceny programu Paradox. Wynik zapytania podano poniżej.

```
SELECT Nazwa, Rodzaj, Cena
FROM Programy
WHERE Cena > ANY ( SELECT Cena
                   FROM Programy
                   WHERE Rodzaj = 'Baza')
ORDER BY Cena DESC;
```

Nazwa	*Rodzaj*	*Cena*
AutoCad	CAD	6000
Delphi	Baza	3100
DeltaCad	CAD	1800
OptimaC++	Baza	1600

Przykład 27 Podzapytanie operatorem ALL

Poniższe polecenie wybiera tabeli 34 programy, których cena jest mniejsza od każdego programu rodzaju Baza. Wynik zapytania uporządkowano według cen został zamieszczony poniżej w tabeli.

```
SELECT Nazwa, Cena, Rodzaj FROM  Programy WHERE
Cena < ALL (SELECT distinct Cena
            FROM Programy
            WHERE Rodzaj = 'Baza')
ORDER BY Cena;
```

Operator ALL umożliwia sprawdzenie, czy warunek jest spełniony dla wszystkich elementów listy wyboru uzyskanej z podzapytania.

Nazwa	*Cena*	*Rodzaj*
Linux	200	SO
Star	220	Edytor
Word	450	Edytor
Windows	500	SO
Desinger	800	CAD

Podzapytania z klauzulą HAVING

Podzapytania mogą być dołączane również do klauzuli HAVING. Dotyczą wtedy warunku wyboru grup określonych w klauzuli GROUP BY.

Przykład 28 Podzapytanie z klauzulą HAVING

W przytoczonym poniżej przykładzie wybierane są te rodzaje programów, dla których średnie ceny są większe niż średnia cena w rodzaju 'SO'.

```
SELECT Rodzaj, AVG (Cena)
FROM Programy
GROUP BY Rodzaj
HAVING AVG (Cena) > ( SELECT AVG (Cena)
                      FROM   Programy
                      WHERE   Rodzaj ='SO')
ORDER BY Rodzaj;
```

Wynik zapytania podano w tabeli poniżej.

RODZAJ	AVG
Baza	1867
CAD	2867

Podzapytania wielokrotnie zagnieżdżone

Polecenia SELECT mogą być wielokrotnie w sobie zagnieżdżane.

Przykład 29 Podzapytania wielokrotnie zagnieżdżone

W poniższym przykładzie wybierane są te programy, których ceny są wyższe niż każdego programu edytorskiego.

```
SELECT  Nazwa, Cena
FROM   Programy
WHERE   Cena >( SELECT max (Cena)
                FROM Programy
                WHERE   Rodzaj = ( SELECT Rodzaj
                                   FROM   Programy
                                   WHERE Rodzaj = 'Edytor'
                                   GROUP BY Rodzaj))
ORDER BY Nazwa;
```

Wynik zapytania uporządkowany według Nazw zamieszczono poniżej w tabeli.

NAZWA	*CENA*
AutoCad	6000
Delphi	3100
DeltaCad	1800
Desinger	800
OptimaC++	600
Paradox	900
Windows	500

ZŁĄCZENIA TABEL

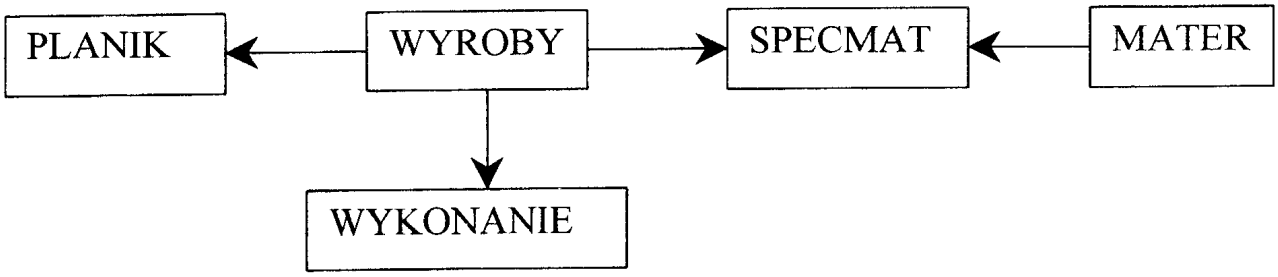
Dotychczasowe ćwiczenia zapytań w języku SQL należy potraktować jako „lekką rozgrzewkę". Prawdziwe zastosowania rozpoczynają się, gdy stawiamy zapytania względem tabel złączonych. Zapytania do tabel złączonych najlepiej prześledzić na przykładach wziętych z praktyki, np. przy produkcji wyrobów, gospodarce magazynowej, czy też przy systemie Zamówień. Zapytania do tabel połączonych mają duże zastosowania w systemach baz danych do tworzenie wszelkiego rodzaju raportów.

Przykłady zastosowania zapytań połączonych

W każdym zakładzie wytwarzającym wyroby są przygotowane takie dane jak wykaz wyrobów, wykaz materiałów, specyfikacja materiałów do wykonania wyrobów, dane technologiczne określające sposób wykonania wyrobów, plan okresowy wykonania wyrobów z ilościami i terminami realizacji.
Weźmy pod uwagę następujące tabele:

- WYROBY - najważniejsze dane o wyrobach,
- MATER - dane **o** materiałach,
- SPECMAT - wykaz materiałów dla wykonania wyrobów,
- WYKONANIE - dane technologiczne wykonania wyrobów,
- PLANIK - plany wykonania wyrobów.

Powiązania między tabelami są realizowane za pomocą kluczy głównych Id_Wyr z tabeli WYROBY i Id_Mat z tabeli MATER. Poniżej podano diagram związków (relacji) między tymi tabelami i same tabele z przykładowymi wartościami



WYROBY

Id_Wyr	Nazwa_Wyr	Jedn	Waga	D_Zatw
W1	KLAMKA	szt	0.2	1999-09-12
W2	KOREK	Szt	0.4	1999-09-15
W3	PRZYCISK	szt	0.1	1999-09-20
W4	LISTWA	szt	0.3	1999-10-10
W5	SPINKA	szt	0.1	1999-10-11
W6	USZCZELKA	Mb	0.4	1999-10-12

SPECMAT

Id_Wyr	Id_Mat	Ilosc
W1	M1	1
W1	M4	0.2
W3	M2	2
W3	M3	1
W3	M4	0.1
W4	M1	1
W4	M2	2
W4	M4	0.2
W4	M5	0.1
W5	M5	0.1

MATER

Id_Mat	Nazwa_Mat	Jedn	Cena	Data
M1	Śruba	Szt	0.1	1999-08-12
M2	Nit	Szt	0.1	1999-09-15
M3	Rolka	Szt	0.2	1999-08-11
M4	Nykan	Kg	10.0	1999-10-10
M5	Epsilon	Kg	5.0	1999-11-11

WYKONANIE

Id_Wyr	Nr_Oper	Operacja	Centrum	Czas_Tech	Czas_Cyklu
W1	1	Formowanie	UT-120	30	0.1
W1	2	Formowanie	UT-160	10	0.2
W1	3	Montaz	MN-1	15	0.1
W3	1	Formowanie	UT-160	40	0.2
W3	2	Montaz	MN-2	10	0.1
W4	1	Formowanie	UT-160	10	0.2
W4	2	Montaz	MN-1	30	0.1
W5	1	Formowanie	UT-120	30	0.1

PLANIK

Nr_Planu	Id_Wyr	Ilosc_Plan	Termin
1	W1	5000	10-NOV-1999
1	W3	6000	20-NOV-1999
1	W5	8000	30-NOV-1999
2	W1	6000	20-DEC-1999
2	W3	4000	30-DEC-1999
2	W4	5000	10-DEC-1999

Tabela 35 Relacje wykorzystane w złączeniach

Przykład 30 Zapytanie - tabele złączone

Wykonać poniższe zapytanie, które określi jakich używa się materiałów i w jakiej ilości oraz wartości na jeden wyrób.

```
SELECT Nazwa_Wyr, Nazwa_mat, S.Ilosc, Jedn, S.Ilosc*M.Cena Wartość
FROM WYROBY W JOIN SPECMAT S JOIN MATER M
ON W.Id_Wyr = S.Id_Wyr ON S.Id_Mat = M.Id_Mat
ORDER BY Nazwawyr;
```

W zapytaniu powyżej użyto aliasów tabel (Wyroby W, SpecMat S, Mater M). Aliasy W, S, M wykorzystano do skrócenia zapisu polecenia jeszcze przed ich definicją co dopuszcza standard języka SQL. Poniżej zamieszczono otrzymany wynik.

Nazwa_ Wyr	*Nazwa_Mat*	*Ilosc*	*Jedn*	*Wartość*
KLAMKA	Nykan	0.2	Kg	2
KLAMKA	Śruba	1	Szt	0.1
LISTWA	Epsilon	0.1	kg	0.5
LISTWA	Nykan	0.2	kg	2
LISTWA	Nit	2	szt	0.2
LISTWA	Śruba	1	szt	0.1
PRZYCISK	Nykan	0.1	kg	1
PRZYCISK	Nit	2	szt	0.2
PRZYCISK	Rolka	1	szt	0.2
SPINKA	Epsilon	0.1	kg	0.5

Operacje na łańcuchach

Operator konkatenacji || dokleja jeden łańcuch tekstowy do drugiego. Co można wykorzystać do formatowania danych wynikowych oraz do odtwarzania wartości złożonych kluczy z ich składowych atrybutów. W poniżej podano dwa przykłady zastosowania operatora || z wykorzystaniem relacji „CENNIK" (tabela 36).

Nazwa	Rodzaj	Cena
Windows	S.O.	500
Word	Edytor	450
Linux	S.O.	200
Star	Edytor	220
Paradox	Baza	900
Delphi	Baza	3100
Optima C++	Baza	1600

Tabela 36 Relacja „Cennik”

Przykład 38 Lista połączonych wartości pól

```
SELECT Rodzaj ||' - '|| Nazwa NowaNazwa
FROM CENNIK
ORDER BY NowaNazwa;
```

W wyniku zapytania otrzymujemy jednokolumnową tabelę której wartościami są złączone łańcuchy z dwóch kolumn tabeli Cennik:

NowaNazwa
Baza-Delphi
Baza-Optima C++
Baza-Paradox
Edytor-Star
Edytor-Word
SO-Linux
SO-Windows

Przykład 39 Zastosowanie funkcji SUBSTIRNG

Funkcja SUBSTRING wycina część podanego łańcucha tekstowego. Można to wykorzystać do formatowania danych wynikowych. Oto jej składnia:

SUBSTRING (Łańcuch FROM Pozycja_startowa FOR Liczba_znaków)

Przykładowo z tabeli „CENNIK" tworzymy listę, w której podajemy tylko pierwszą literę rodzaju programu, kreskę i nazwę programu za pomocą zapytania:

```
SELECT SUBSTRING ( Rodzaj FROM 1 FOR 1) || ' - '|| Nazwa  InnaNazwa
FROM  CENNIK
ORDER BY InnaNazwa
```

Wynik zapytania podano w tabel poniżej.

InnaNazwa
B-Delphi
B-Optima C++
B-Paradox
E-Star
E-Word
S-Linux
S-Windows

PERSPEKTYWY

Perspektywa (View) jest tabelą wirtualną wygenerowaną przez odpowiednie zapytanie języka SQL. Zapytanie to ma własną nazwę i jest przechowywane w słowniku danych. Perspektywy umożliwiają zapisanie często wykonywanych złożonych zapytań do bazy danych, którymi można się formalnie posługiwać jak tabelami np. SELECT * FROM <Nazwa perspektywy>. Perspektywa charakteryzuje się następującymi cechami:

- perspektywa jest definiowana na podstawie jednej lub kilku tabel,
- perspektywa pamiętana jest w postaci polecenia SELECT,
- perspektywa nie ma własnych danych.

Rozróżnia się perspektywy proste i złożone. Perspektywa prosta udostępnia dane z jednej tabeli a złożone z wielu tabel. Wprowadzać dane można tylko do perspektyw prostych.

Do tworzenia perspektyw służy polecenie CREATE VIEW języka SQL. Jego składnia jest następująca:

```
CREATE VIEW < Nazwa perspektywy >
[ (<atrybut> [, < Atrybut >…])]
AS SELECT < Ciało polecenia SELECT >
```

Przykład 41 Przykład perspektywy prostej

Poniżej podano przykład perspektywy prostej „Bazy" utworzonej na podstawie jednej tabeli „Programy" z wykorzystaniem predykatu WHERE.

```
CREATE VIEW Bazy
(Nazwa_bazy, Cena)
AS    SELECT  Nazwa, Cena
      FROM   Programy
      WHERE Rodzaj = 'Baza';
```

W perspektywie tej przechowywane są następujące dane:

Nazwa_bazy	Cena
Paradox	900
Delphi	3100
Optima C++	1600

Tabela 37 Perspektywa prosta

Przykład 42 Przykład perspektywy złożonej

Perspektywy złożone otrzymuje się najczęściej jako złączenie kilku tabel w poleceniu SELECT. Poniżej podano przykład takiej perspektywy.

```
CREATE VIEW WIDOKI
 ( Id_Wyr,
   Nazwa_Wyrobu,
   Wartość)
AS
        SELECT Id_Wyr, Name_Wyr, SUM( S.Ilosc*M.Cena) WartoscMater
        FROM WYROBY W JOIN SPECMAT S JOIN MATER M
        ON W.Id_Wyr - S.Id_Wyr
        ON S.Id_Mat =M.Id_Mat
        GROUP BY Id_Wyr, Name_Wyr;
```

W perspektywie tej przechowuje się następujące dane będące wynikiem zapytania select.

Id_Wyrobu	Nazwa_Wyrobu	Wartosc_Materialow
W1	KLAMKA	2.1
W3	PRZYCISK	1.4
W4	LISTWA	2.8
W5	SPINKA	0.5

Tabela 38 Perspektywa złożona