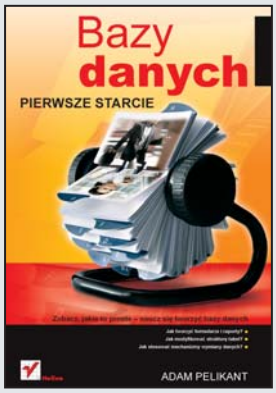
# Bazy danych. Pierwsze starcie

Autor: Adam Pelikant
ISBN: 83-246-2041-9
Format: 158x23 , stron: 208




**Zobacz, jakie to proste – naucz się tworzyć bazy danych!**

- Jak tworzyć formularze i raporty?
- Jak modyfikować strukturę tabel?
- Jak stosować mechanizmy wymiany danych?

Współczesny świat wymusza na przedsiębiorstwach gromadzenie oraz przetwarzanie ogromnej ilości informacji. To sprawia, że muszą one dysponować wydajnymi i sprawnymi bazami danych. Aby zbudować taki system zarządzania danymi, niezbędne są odpowiednie narzędzia – jednym z nich jest program MS Access. Ta aplikacja przede wszystkim pozwala na łatwą kontrolę poprawności tworzonych projektów oraz zapewnia integrację narzędzi służących do tworzenia struktury relacyjnej. Dba także o zgodność tych narzędzi ze standardem języka zapytań SQL, wykorzystywanym do tworzenia i modyfikowania baz danych oraz operowania na zgromadzonych w nich informacjach.

Książka „Bazy danych. Pierwsze starcie" stanowi doskonałe wprowadzenie w tematykę tworzenia baz danych. Zawiera wszystkie potrzebne informacje, podane w prosty i przejrzysty sposób. Ten podręcznik przyda się zarówno studentom kierunków informatycznych, jak i wszystkim tym, którzy chcą zdobyć wiedzę o nowoczesnych metodach budowania takich baz. Stąd dowiesz się m.in., jak wykorzystywać język zapytań SQL, w jaki sposób tworzyć tabele, formularze i raporty oraz stosować mechanizmy wymiany danych, a także na czym polega filtrowanie i sortowanie w zapytaniach. Zdobędziesz wiedzę i umiejętności wystarczające do samodzielnego zbudowania wydajnej bazy danych i sprawnego nią zarządzania.

- Projektowanie bazy danych – narzędzia wizualne
- Tworzenie formularzy i raportów
- Strukturalny język zapytań SQL w wersji MS JetSQL
- Składnia podstawowa
- Unia – koniunkcja zbiorów
- Grupowanie i funkcje agregujące
- Zastosowanie języka SQL z poziomu formularzy
- Mechanizmy wymiany danych
- Obiekty: DAO, RDO, ADO
- Zastosowanie mechanizmów wymiany danych przy tworzeniu aplikacji

**Stwórz własną, niezawodną bazę danych!**



Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

# Spis treści

Rozdział 3.

# Strukturalny język zapytań SQL w wersji MS Jet SQL

## Informacje podstawowe

Dla baz danych o strukturze relacyjnej podstawowym narzędziem dostępu do danych i manipulowania nimi jest strukturalny język zapytań SQL. Na rynku nie istnieje liczące się środowisko zarządzania bazą danych, w którym nie stosowano by tego narzędzia. Tworzenie kodu SQL stanowi najbardziej podstawowy, ale jednocześnie najwszechstronniejszy sposób programowania w środowiskach bazodanowych. Pomimo znacznie posuniętej standaryzacji nie istnieje w praktyce pełna zgodność między „językami SQL" stosowanymi przez różnych twórców oprogramowania. Nawet produkty tej samej firmy mogą stosować różne warianty SQL. Na szczęście różnice między poszczególnymi „klonami" języka są niewielkie, tak że dogłębne poznanie jednego z nich pozwala na sprawne posługiwanie się tym językiem w innych środowiskach. W książce skupiono się na MS Jet SQL będącym językiem stosowanym w systemie zarządzania bazą danych Access, jako najbardziej podstawowym zakresie tego języka. Inne klony zostaną przedstawione w pracy dotyczącej rozszerzeń proceduralnych SQL.

W wielu publikacjach stosowany jest podział na trzy części: instrukcje wybierające (SELECT), język manipulowania danymi (*DML — Data Manipulation Language*) oraz język definicji baz danych (*DDL — Database Definition Language*). Podział ten jest sztuczny, dodatkowo sugeruje, że mamy do czynienia z różnymi językami programowania, co jest ewidentnym mijaniem się z prawdą. W związku z tym taki podział nie będzie stosowany w niniejszej książce.

W języku SQL nie jest rozróżniana wielkość liter (w niektórych środowiskach nie dotyczy to łańcuchów będących argumentem funkcji), stąd zapis wielkimi lub małymi literami czy też dowolną ich kombinacją jest równoważny, np.:

```
select nazwisko from osoby
SELECT NAZWISKO FROM OSOBY
Select Nazwisko From Osoby
```
} *są równoważne*

W celu poprawienia czytelności poleceń w całej książce wprowadzono specjalną notację. Polecenia SQL, nazwy operatorów i funkcji będą pisane wielkimi literami, natomiast nazwy tabel i pól małymi (pierwsza wielka) literami.

Polecenia SQL mogą być pisane w wielu liniach. Jedynym ograniczeniem jest to, aby podział na linie nie następował w obrębie słów kluczowych, poleceń, operatorów, nazw pól itp.:

```
SELECT Nazwisko FROM Osoby  ≡  SELECT
                               Nazwisko
                               FROM
                               Osoby
```

Standardowo znacznikiem końca polecenia w SQL jest znak średnika (*;*) jednak w większości środowisk, w tym również w MS Jet SQL, może on być pominięty. Dlatego nie będzie on używany w tej książce. Czytelnik powinien jednak pamiętać o jego roli, gdyż w przypadku rozwinięć proceduralnych często pominięcie średnika nie jest dopuszczalne (trudno zinterpretować sens poleceń).

Możliwe jest pokazanie składni języka na przykładzie ogólnych postaci poleceń, czy też ich grafów. Jednakże autor uważa, że o wiele bardziej dydaktyczne jest operowanie przykładami, przy wprowadzaniu najpierw najprostszych postaci poleceń i stopniowym dochodzeniu do postaci bardziej rozbudowanych.

Rozpocznijmy od stanu, gdy mamy już zdefiniowane tabele oraz wpisane do nich przykładowe dane, a naszym zadaniem jest wybranie tylko pewnych interesujących nas danych. Innymi słowy zaczynamy od poznania najpotężniejszego polecenia SQL, jakim jest `SELECT`.

# Zapytania wybierające — SELECT

## Składnia podstawowa

W celu wybrania wszystkich pól i wszystkich rekordów z tabeli o nazwie *Osoby* możemy wykonać następujące zapytanie:

```
SELECT * FROM Osoby
```

gdzie symbol gwiazdki jest znakiem specjalnym i zastępuje nazwy wszystkich pól. Formalnie znakiem oznaczającym koniec zapytania jest symbol *;* (średnik), ale większość parserów pozwala na jego pominięcie, stąd w książce nie będzie on stosowany. Jednak w niektórych systemach zarządzania bazami danych może on być niezbędny, o czym czytelnicy powinni pamiętać.

Jeżeli chcemy ograniczyć się do wybrania tylko niektórych pól z tabeli, musimy w sposób jawny podać ich nazwy, np.:

```
SELECT Nazwisko, Imie, RokUrodz FROM Osoby
```

W tym przypadku wyświetlona będzie zawartość trzech pól ze wszystkich rekordów tabeli *Osoby*.

W części głównej zapytania wybierającego dozwolone jest stosowanie wyrażeń. Możemy na przykład wyświetlić zawartość dwóch pól: *Imie* i *Nazwisko*, w postaci jednego połączonego pola:

```
SELECT Nazwisko & ' ' & Imie FROM Osoby
```

Należy zwrócić uwagę, że operatorem konkatenacji (łączenia) łańcuchów (zmiennych znakowych) jest znak ampersand (&). Operator ten jest charakterystyczny dla MS Jet SQL. W innych systemach jest on zastępowany przez plus (+) lub podwójny znak more (||). Łańcuch zawierający spację jest dołączony, aby rozdzielić zawartość pól. Przy tak skonstruowanym zapytaniu nazwa pola obliczeniowego jest nadawana przez system zarządzania bazą danych i charakterystyczna dla tego systemu. Jeśli w sposób jawny chcemy nazwać pole obliczeniowe lub zmienić nazwę pola wyświetlaną przy wykonaniu zapytania, stosujemy tzw. alias nazwy, na przykład poprzednie zapytanie możemy przepisać do postaci:

```
SELECT Nazwisko & ' ' & Imie AS Osoba FROM Osoby
```

Tak samo wyrażenia te mogą dotyczyć pól numerycznych oraz w części głównej zapytania może występować ich wiele.

```
SELECT Brutto, 0.8*Brutto AS Dochod, 0.2*Brutto AS Podatek FROM Zarobki
```

Dostępne są podstawowe operatory algebraiczne i logiczne (tabela 3.1)

**Tabela 3.1.** *Podstawowe operatory dostępne w MS Jet SQL*

| Arytmetyczne | Logiczne i bitowe | Znakowe |
|---|---|---|
| + dodawanie | `AND` iloczyn bitowy | `&` konkatenacja łańcuchów |
| - odejmowanie | `OR` suma bitowa | |
| * mnożenie | `NOT` przeczenie bitowe | |
| / dzielenie | | |
| \ dzielenie całkowitoliczbowe | | |
| `mod` modulo | | |

Oprócz przedstawionych powyżej operatorów dostępnych jest szereg funkcji operujących na każdym z dostępnych typów danych, charakterystycznych dla danego środowiska. Ponieważ ich liczba jest znaczna, nie jest sensowne omawianie ich w tym miejscu.

Nazwy aliasów pól muszą być w obrębie pojedynczego zapytania różne od siebie i nie mogą być nazwami pól tabeli, a także nie mogą być słowami zastrzeżonymi języka SQL (nazwy poleceń, klauzul etc.). Aliasy mogą być również stosowane do określenia zmienionej nazwy nie tylko pola, ale również tabeli.

```
SELECT Nazwisko AS Nazwa FROM Osoby AS O
```

Z reguły do określenia pola wystarcza podanie jego nazwy w tabeli, jednakże czasami konieczne jest stosowanie tzw. nazwy kwalifikowanej, składającej się z nazwy tabeli lub jej aliasu oraz nazwy pola, tak jak to pokazują dwa równoważne przykłady:

```
SELECT Osoby.Nazwisko AS Nazwa FROM Osoby
SELECT O.Nazwisko AS Nazwa FROM Osoby AS O
```

Przy czym jeśli przy nazwie tabeli występuje alias, to w nazwie kwalifikowanej musi się pojawić ten alias, a nie właściwa nazwa tabeli. Przy tworzeniu aliasów nazw tabel dopuszczalne jest pominięcie słowa kluczowego *AS*.

```
SELECT O.Nazwisko AS Nazwa FROM Osoby O
```

# Sortowanie w zapytaniach

Do tej pory rekordy wyprowadzane były w kolejności występującej w tabeli, w kolejności wpisywania (dokładniej rzecz ujmując, w kolejności narzuconej przez indeks grupujący — o czym później). Można jednak narzucić kolejność wyprowadzania rekordów w definicji zapytania. Za sortowanie rekordów odpowiada klauzula `ORDER BY`.

```
SELECT Nazwisko, Imie FROM Osoby ORDER BY Nazwisko
```

W tym przypadku rekordy zostaną posortowane według nazwiska, a domyślnym sposobem sortowania jest sortowanie rosnące. W jawny sposób rosnący kierunek sortowania możemy wskazać przez użycie operatora `ASC` (*ascending* — rosnąco) w klauzuli `ORDER BY`.

```
SELECT Nazwisko, Imie FROM Osoby ORDER BY Nazwisko ASC
```

Jeśli chcemy wymusić sortowanie w kierunku malejącym, możemy użyć operatora `DESC` (*descending* — malejąco).

```
SELECT Nazwisko, Imie FROM Osoby ORDER BY Nazwisko DESC
```

Sortowanie może być hierarchiczne, czyli np. w obrębie grupy takich samych nazwisk możemy zastosować sortowanie względem imienia. Przy czym kierunek sortowania (rosnący lub malejący) jest niezależny dla każdego z poziomów sortowania.

```
SELECT Nazwisko, Imie FROM Osoby ORDER BY Nazwisko DESC, Imie ASC
```

Sortowanie zawsze jest rozpoczynane od pierwszego pola występującego po klauzuli `ORDER BY`. Nie istnieje ograniczenie liczby poziomów sortowania. Wskaźnikiem sortowania może być nie tylko pojedyncze pole, ale także dowolne wyrażenie oparte o pola tej tabeli.

```
SELECT RokUrodz*Wzrost AS Iloczyn FROM Osoby ORDER BY RokUrodz*Wzrost
```

W składni standardu SQL możliwe jest sortowanie w oparciu o alias pola, metoda ta nie jest jeszcze zaimplementowana w MS Jet SQL.

Dodatkowo należy pamiętać, iż pola użyte do sortowania nie muszą być wyświetlane (nie muszą występować w części głównej zapytania wybierającego) oraz że klauzula sortująca (ORDER BY) jest zawsze ostatnią z klauzul występujących w zapytaniu, co zostanie pokazane w dalszej części książki.

# Filtrowanie w zapytaniach

W poprzednich przykładach zawsze wyświetlane były informacje obejmujące wszystkie rekordy tabeli. W celu wybrania (odfiltrowania) interesującej nas informacji możemy użyć klauzuli WHERE.

```
SELECT Nazwisko, Imie FROM Osoby WHERE RokUrodz >1970
```

Przy czym po klauzuli WHERE wpisywane jest wyrażenie logiczne. Zapytanie wyświetli te rekordy, dla których wartość wyrażenia jest prawdziwa (TRUE). Powyższy przykład pokazuje wyświetlenie tych osób, które urodziły się po 1970 roku. W wyrażeniu został użyty operator algebraiczny większości. Poza tym w wyrażeniach mogą być używane operatory logiczne oraz specjalne, tak jak to pokazuje tabela 3.2.

**Tabela 3.2.** *Operatory dostępne w MS Jet SQL*

| Operatory algebraiczne | Operatory logiczne | Operatory specjalne |
|---|---|---|
| = równe | NOT przeczenie | BETWEEN przedział |
| < mniejsze niż | OR suma logiczna | IN lista |
| <= mniejsze lub równe | AND iloczyn logiczny | LIKE podobieństwo |
| > większe niż | | SOME lista z operatorem |
| >= większe lub równe | | ANY lista z operatorem |
| <> różne | | ALL lista z operatorem |
| != nierówne | | EXISTS istnieje |

W najprostszym przypadku możemy mieć połączenie operatorem logicznym dwóch wyrażeń algebraicznych, np.: wyświetlenie danych osób urodzonych po roku 1960, a przed rokiem 1970.

```
SELECT Nazwisko, Imie FROM Osoby WHERE RokUrodz>1960 AND RokUrodz<1970
```

lub osób urodzonych przed rokiem 1960 oraz po roku 1970:

```
SELECT Nazwisko, Imie FROM Osoby WHERE RokUrodz<1960 OR RokUrodz>1970
```

Przy tworzeniu bardziej skomplikowanych wyrażeń filtrujących należy pamiętać, że wyrażenia przetwarzane są zawsze od strony lewej do strony prawej, chyba że zadecydujemy inaczej, grupując odpowiednie elementy nawiasami.

```
…WHERE RokUrodz>1950 AND
  RokUrodz<1960 OR Wzrost>180
…WHERE RokUrodz>1950 AND
  (RokUrodz<1960 OR Wzrost>180)
```

W podanych wyżej przykładach wyrażenia filtrujące nie są tożsame, gdyż w pierwszym zapytaniu zostanie wyprowadzona osoba urodzona przed 1950 rokiem, o ile tylko jest wyższa niż 180 cm, natomiast w drugim zapytaniu taka osoba nie zostanie wyprowadzona.

W poprzednich przykładach zakładano, że zawsze wszystkie pola tabeli są wypełnione. Jednak może się zdarzyć, że posiadamy informacje niekompletną albo że informacja zawarta w którymś polu nie dotyczy tego rekordu czy też nie może być dla niego uzyskana. W takim przypadku zawartość pola pozostaje „pusta" — ściśle mówiąc, ma wartość NULL (nieznaną, niezidentyfikowaną). W ten sposób wchodzimy w zakres logiki trójwartościowej, gdzie dwa dotychczasowe stany są uzupełnione o wartość NULL. Stąd musimy wprowadzić nowe definicje operatorów logicznych obowiązujących w tej logice, przedstawione w postaci tabelarycznej (tabele 3.3 – 3.5).

**Tabela 3.3.** *Wartości operatora AND w logice trójwartościowej*

| AND | TRUE | FALSE | NULL |
|---|---|---|---|
| TRUE | TRUE | FALSE | NULL |
| FALSE | FALSE | FALSE | FALSE |
| NULL | NULL | FALSE | NULL |

**Tabela 3.4.** *Wartości operatora OR w logice trójwartościowej*

| OR | TRUE | FALSE | NULL |
|---|---|---|---|
| TRUE | TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE | NULL |
| NULL | TRUE | NULL | NULL |

**Tabela 3.5.** *Wartości operatora NOT w logice trójwartościowej*

| A | NOT A |
|---|---|
| TRUE | FALSE |
| FALSE | TRUE |
| NULL | NULL |

Jednym z najważniejszych wniosków płynących z tych tabel jest to, że ponieważ:

```
NULL AND NULL = NULL
```

to

```
(NULL=NULL) ⇒ NULL
```

stąd aby wykryć wartości niezdefiniowane (nieznane) występujące w jakimś polu, należy użyć specjalnego operatora IS NULL:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE RokUrodz IS NULL
```

Bardziej szczegółowego omówienia wymagają operatory specjalne. Pierwszym z nich jest operator przedziału BETWEEN. Ilustracją niech będzie zapytanie:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE RokUrodz BETWEEN 1960 AND 1980
```

które zwraca rekordy z przedziału obustronnie domkniętego, ograniczonego dwoma podanymi parametrami (datami). Tak więc zapytanie to jest równoważne zapytaniu:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE RokUrodz >=1960 AND RokUrodz >=1980
```

w którym obie nierówności ograniczające przedział są nieostre.

Kolejnym przykładem operatora specjalnego jest operator `IN` pozwalający na dokonanie wyboru tych elementów, które zawiera lista. Na przykład możemy wyprowadzić dane dotyczące osób urodzonych w konkretnych (wymienionych latach):

```
SELECT Nazwisko, Imie FROM Osoby
WHERE RokUrodz  IN (1960, 1970, 1980)
```

Jest ono równoważne wielokrotnemu porównaniu z wartościami, jak pokazano niżej:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE (RokUrodz = 1960 OR RokUrodz = 1970 OR
 RokUrodz = 1980)
```

Przewaga operatora `IN` jest widoczna w przypadku długich (o wielu elementach) list lub, co zostanie podane dalej, list dynamicznych — o bliżej nieokreślonej liczbie elementów. Operator listy `IN` może być wykorzystywany z dowolnym typem danych, np. dla zmiennych znakowych możemy zapisać zapytanie:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko  IN ('Kowalski', 'Nowak');
```

które jest równoważne zapytaniu:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE (Nazwisko = 'Kowalski'  OR Nazwisko= 'Nowak')
```

W wersji statycznej jednak operator listy najczęściej jest wykorzystywany z danymi numerycznymi, natomiast do danych znakowych używany jest najczęściej operator podobieństwa `LIKE`. W podstawowej postaci np.:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE 'KOW'
```

jest on równoważny operatorowi równości:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko = 'KOW'
```

Czyli wyprowadzone będą dane osób o nazwisku KOW. Zwrócić uwagę należy na to, iż w zależności od systemu rozróżniana jest lub nie wielkość liter. W przypadku MS Jet SQL wielkość liter nie odgrywa roli, w związku z tym łańcuchy znaków (KOW, kow, Kow) są równoważne. Siłę operatora podobieństwa można ocenić dopiero, jeżeli zastosujemy znaki specjalne, dla MS Jet SQL są to znaki opisane w tabeli 3.6.

**Tabela 3.6.** *Znaki specjalne dla operatora podobieństwa LIKE*

| |
|---|
| * dowolny ciąg znaków (w tym ciąg pusty) |
| ? dokładnie jeden znak |

Zapytanie wyświetlające dane dla osób, których nazwiska rozpoczynają się od frazy *KOW*:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE 'KOW*'
```

Wybrane zostaną dane np. dla nazwisk: Kowalski, Kowalczyk, Kowal etc., ale również dla nazwiska Kow.

Zapytanie wyświetlające dane dla osób, których nazwiska kończą się frazą *KOW*:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '*KOW'
```

Wybrane zostaną dane np. dla nazwisk: Jakow, Bułhakow etc., ale również dla nazwiska Kow.

Zapytanie wyświetlające dane dla osób, których nazwiska posiadają w środku frazę *KOW*:

```
SELECT *Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '*KOW*'
```

Wybrane zostaną dane np. dla nazwisk: Jankowski, Ziółkowski, Kwiatkowski etc., ale również dla nazwisk rozpoczynających się frazą *KOW*: Kowalski, Kowalczyk, Kowal etc., oraz nazwisk kończących się frazą KOW: Jakow, Bułhakow etc., a także dla nazwiska Kow.

Zapytanie wyświetlające dane dla osób, których nazwiska rozpoczynają się od litery *K*, a kończą literą *I*:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE 'K*I'
```

Wybrane zostaną dane np. dla nazwisk: Kowalski, Kwiatkowski, Kamieński etc., ale również dla nazwiska *Ki* — o ile oczywiście takie istnieje.

Zapytanie wyświetlające dane dla osób, w których nazwisku na trzeciej pozycji występuje litera *W*:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '??W*'
```

Wyświetlone zostaną dane np. dla nazwisk: Kowal, Nowak, Pawlak etc., ale również dla nazwisk: Kow, Now etc. Gdyby we wzorcu nie została użyta kończąca frazę gwiazdka, wybrane zostałyby dane tylko dla nazwisk drugiego typu — składających się dokładnie z trzech znaków.

Zapytanie wyświetlające dane dla osób, w których nazwisku pierwsza litera zawiera się w przedziale od *k* do *n*:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '[k-n]*'
```

Wyświetlone zostaną dane np. dla nazwisk: Kowalski, Lwow, Makowski, Nowak etc.; należy zauważyć, że przedział jest przedziałem obustronnie domkniętym.

Zapytanie wyświetlające dane dla osób, w których nazwisku pierwsza litera nie zawiera się w przedziale od *k* do *n*:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '[!k-n]*'
```

Wypisane zostaną dane dla nazwisk, których pierwsza litera poprzedza *k*, np.: Adamczyk, Ciesielski, Janik, oraz nazwisk, których pierwsza litera występuje za *n*, np.: Olszewski, Pawlak, Zięba etc.

Zapytanie wyświetlające dane dla osób, w których nazwisku pierwsza litera jest jedną z listy znaków: *k*, *z*, *a*, *n*:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '[kzan]*'
```

Wyświetlone zostaną dane np. dla nazwisk: Kowal, Zięba, Adamczyk, Nowak etc. Należy zauważyć, że w liście ujętej w nawias kwadratowy nie występuje żaden znak separatora. W przypadku zastosowania jakiegokolwiek separatora, np. (, spacja ;) zostanie on potraktowany jak kolejny znak listy — dosłownie.

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '[k,z a;n]*'
```

Zostałyby wyświetlone nazwiska rozpoczynające się od tych znaków. Oczywiście w praktyce, poza błędami przy wpisywaniu, trudno się spodziewać nazwisk rozpoczynających się od znaków niebędących literami, ale dla pól zawierających inne dane, np.: kody, symbole, jest to już bardziej prawdopodobne.

Poniżej pokazano zapytanie wyświetlające dane dla osób, których nazwiska zaczynają się od cyfry.

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '[0-9]*'
```

W przypadku MS Jet SQL przedział obejmujący cyfry można zastąpić znakiem hash (#), czyli następne zapytanie jest równoważne poprzedniemu.

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '#*'
```

Ponieważ symbole ujęte w nawias kwadratowy są traktowane dosłownie, to poniższe zapytanie wyświetli dane dla osób, w których nazwisku na dowolnej pozycji występuje znak *:

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '*[*]*'
```

Można postawić pytanie, jak wyświetlić dane osób, których nazwisko zaczyna się od jednego ze znaków *k* – *n*. Zastosowanie wzorca `'[k-n]*'` zwraca przedział od *k* do *n*, ale wystarczy zamienić kolejność znaków, aby otrzymać poszukiwane rozwiązanie.

```
SELECT Nazwisko, Imie FROM Osoby
WHERE Nazwisko LIKE '[-kn]*'
```

Przedstawione tutaj przykłady wzorców mogą być oczywiście w dowolny sposób łączone, dając bardzo złożone sposoby filtrowania; świadczy to dobitnie o sile operatora `LIKE`.

Pozostałe operatory specjalne (`ALL`, `SOME`, `ANY`, `EXISTS`) zostaną omówione w części poświęconej podzapytaniom.

# Grupowanie i funkcje agregujące

Obliczenia wykonywane do tej pory dotyczyły pojedynczego wyprowadzanego rekordu. Możemy dokonywać operacji algebraicznych dotyczących wszystkich rekordów lub też ustalonych ich grup. Na przykład wyznaczmy sumę wszystkich zarobków (dla całej tabeli — wszystkich osób):

```
SELECT SUM(Brutto) AS Razem FROM Zarobki
```

Funkcję występującą w części głównej zapytania (w naszym przypadku `SUM`) będziemy nazywać funkcją agregującą lub agregatem. Wyznaczenie funkcji agregującej jest również możliwe dla grupy (w naszym przypadku — dla każdej osoby):

```
SELECT SUM(Brutto) AS Razem FROM Zarobki
GROUP BY IdOsoby
```

W celu wyznaczenia grup, względem których wyznaczane są funkcje agregujące, stosujemy klauzulę `GROUP BY`. W zaprezentowanym przykładzie pokazane są jedynie sumy i nie wiadomo, komu je wypłacono, w związku z tym możemy w części głównej wymienić pole grupujące — *IdOsoby*:

```
SELECT IdOsoby, SUM(Brutto) AS Razem FROM Zarobki
GROUP BY IdOsoby
```

Jak widać, w części głównej zapytania agregującego nie musi występować pole (pola) grupowania, ale jeżeli w części głównej pojawi się jakiekolwiek pole bez funkcji agregującej, to musi się ono pojawić w klauzuli `GROUP BY`. Tak jak przy każdym z pól tak i przy polu z funkcją agregującą może pojawić się alias nazwy. W części głównej zapytania może pojawić się wiele funkcji agregujących, dotyczyć mogą one wielu pól oraz możliwe jest wykonywanie na nich działań algebraicznych.

```
SELECT IdOsoby, SUM(Brutto) AS Razem, AVG(Brutto) AS Srednio, COUNT(IdZarobku) AS Ile,
MAX(Brutto)/AVG(Brutto) AS Wskaźnik
FROM Zarobki GROUP BY IdOsoby
```

W MS Jet SQL dostępne są funkcje agregujące wymienione w tabeli 3.7.

**Tabela 3.7.** *Funkcje agregujące dostępne w MS Jet SQL*

| | |
|---|---|
| `AVG` wartość średnia | $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ |
| `SUM` suma | $\sum_{i=1}^{n} x_i$ |
| `MAX` maksimum | |
| `MIN` minimum | |
| `FIRST` pierwszy w grupie | |
| `LAST` ostatni w grupie | |
| `STDEV` odchylenie standardowe | $\sigma = \sqrt{\frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{(n-1)}}$ |
| `VAR` wariancja | $\sigma^2 = \frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{(n-1)}$ |
| `COUNT` zlicz | |

Zastanówmy się nad skutkiem działania następującego zapytania:

```
SELECT COUNT(*), COUNT(IdOsoby), COUNT(RokUrodz)
FROM Osoby
```

Czy zawsze zwrócone zostaną trzy takie same liczby? Jeżeli przypomnimy sobie informacje dotyczące wartości `NULL`, oczywistym się stanie, że nie. Przy czym różnica może dotyczyć trzeciego z wyprowadzanych pól. Pierwsza liczba zawsze pokaże liczbę wszystkich rekordów (gwiazdka oznacza wszystkie pola — jeśli nie ma żadnego pola, nie ma rekordu), druga również zlicza wszystkie rekordy (*IdOsoby* jako klucz podstawowy określa rekord, ponadto z definicji nie może mieć wartości `NULL`). Inaczej rzecz się ma w przypadku zliczania pola niebędącego kluczem podstawowym. Jeżeli nie postanowiono inaczej, wprowadzając dodatkowe ograniczenia, może się w nim pojawić wartość `NULL`. Zliczane są tylko te rekordy, które są określone (*RokUrodz* jest określony, nie ma wartości `NULL`). Rozważmy działanie:

```
NULL ϑ A ≡ NULL
```

gdzie A to dowolna wartość, ϑ jest dowolnym operatorem.

Ta właściwość powoduje, że jeśli w którymkolwiek z rekordów pola, na którym wyznaczamy funkcję agregującą, byłaby `NULL`, funkcja ta byłaby także `NULL`. Aby tego uniknąć, do wyznaczania funkcji agregujących brane są tylko wartości określone (niebędące `NULL`). Reguła ta dotyczy wszystkich funkcji agregujących — również zliczania (`COUNT`)

W podanych przykładach występował jedynie pojedynczy poziom grupowania, istnieje możliwość grupowania na wielu poziomach, ale ponieważ wymaga to operacji na więcej niż jednej tabeli, zostanie pokazane po wprowadzeniu złączeń.

W przypadku zapytań agregujących możliwe jest również filtrowanie. W takim typie zapytania możliwe jest filtrowanie na dwóch poziomach. Pierwszy na poziomie rekordów jest oparty o znaną nam już klauzulę `WHERE`.

```
SELECT IdOsoby, SUM(Brutto) AS Razem FROM Zarobki
WHERE Brutto >100
GROUP BY IdOsoby
```

Wykonanie takiego zapytania powoduje wyprowadzenie sum brutto składających się ze składników większych niż 100, przykładowe działanie ilustruje tabela 3.8.

**Tabela 3.8.** *Skutek wykonania zapytania agregującego z filtrowaniem na poziomie rekordu*

| | Kowalski | Nowak | Janik |
|---|---|---|---|
| | 100 | 100 | 300 |
| | 100 | 200 | 200 |
| | 100 | 100 | |
| | 100 | | |
| RAZEM bez filtrowania | 400 | 400 | 500 |
| RAZEM z filtrowaniem | Niewyświetlane | 200 | 500 |

Jeżeli chcemy zastosować filtrowanie względem funkcji agregujących, używamy klauzuli `HAVING`. Ponieważ dotyczy ona funkcji wyznaczanych przy zmianie grupy, musi zawsze występować po klauzuli `GROUP BY`, np.:

```
SELECT IdOsoby, SUM(Brutto) AS Razem FROM Zarobki
GROUP BY IdOsoby
HAVING SUM(Brutto)>400
```

Wyświetlone będą te rekordy, których suma przekroczy 400, czyli te, które wskazano w tabeli 3.9.

**Tabela 3.9.** *Skutek wykonania zapytania agregującego z filtrowaniem na poziomie grupy rekordów*

| | Kowalski | Nowak | Janik |
|---|---|---|---|
| | 100 | 100 | 300 |
| | 100 | 200 | 200 |
| | 100 | 100 | |
| | 100 | | |
| RAZEM | 400 | 400 | 500 |
| | Niewyświetlane | Niewyświetlane | Wyświetlane |

Należy zauważyć, że funkcja, względem której odbywa się filtrowanie w klauzuli `HAVING`, nie musi być tą samą funkcją, której rezultat jest wyświetlany (jest podana w części głównej zapytania). Oczywiście możliwe jest stosowanie obu rodzajów filtrowania (na poziomie rekordu oraz na poziomie grupy — funkcji agregującej) jednocześnie, w jednym zapytaniu:

```
SELECT IdOsoby, SUM(Brutto) AS Razem FROM Zarobki
WHERE Brutto >100
GROUP BY IdOsoby
```

```
HAVING SUM(Brutto)>400
ORDER BY SUM(Brutto)
```

W takim przypadku przykładowe wyniki będą wyglądały tak, jak pokazano w tabeli 3.10.

**Tabela 3.10.** *Skutek wykonania zapytania agregującego z filtrowaniem na poziomie rekordu i grupy rekordów*

| | Kowalski | Nowak | Janik |
|---|---|---|---|
| | 100 | 100 | 300 |
| | 100 | 200 | 200 |
| | 100 | 100 | |
| | 100 | | |
| RAZEM bez filtrowania | 400 | 400 | 500 |
| RAZEM z filtrowaniem | NULL | 200 | 500 |
| | Niewyświetlane | Niewyświetlane | Wyświetlane |

Należy zwrócić uwagę, że klauzula `ORDER BY` znajduje się na końcu, a sortowanie może się odbywać zarówno w oparciu o pole grupujące, jak i funkcję agregującą. Przy czym w przypadku funkcji agregującej nie musi być to ta sama funkcja, która występuje w części głównej, ani ta, która jest używana z klauzulą filtrującą `HAVING`.

# Zapytania do wielu tabel — złączenia

Dotychczas prezentowane zapytania wyświetlały dane z pojedynczej tabeli. Jednakże często interesują nas dane zgromadzone w kilku tabelach, taka organizacja danych jest przecież charakterystyczna dla modelu relacyjnego. W przypadku danych zgromadzonych w dwóch tabelach (*Osoby* i *Zarobki*) możemy wykonać zapytanie:

```
SELECT Nazwisko, Brutto FROM Osoby, Zarobki
```

gdzie *Nazwisko* jest polem tabeli *Osoby*, natomiast *Brutto* polem tabeli *Zarobki*. Zapytanie takie jest poprawne składniowo, a skutkiem jego wykonania jest iloczyn kartezjański (krotka) na wartościach wybranych pól ze wszystkich rekordów obu tabel. Czyli wyświetlone zostaną dla każdego nazwiska wszystkie wartości brutto. W większości przypadków interesuje nas, aby dla danego *Nazwiska* zostały wyświetlone tylko wartości *Brutto* jego zarobków (wynagrodzeń). To zadanie realizowane jest poprzez złączenie tabel — najczęściej na polach zawierających takie same dane (pomiędzy kluczem głównym jednej a kluczem obcym drugiej tabeli). Pierwszą metodą realizacji złączenia jest zastosowanie znanej już klauzuli filtrującej `WHERE`.

```
SELECT Nazwisko, Brutto FROM Osoby, Zarobki
WHERE Osoby.IdOsoby=Zarobki.IdOsoby
```

Warunkiem złączenia jest najczęściej równość wartości pól. Ponieważ z reguły mają one taką samą nazwę w obu tabelach, w takim przypadku konieczne jest użycie nazwy kwalifikowanej pól. Natomiast warunkiem koniecznym realizacji złączenia jest tylko zgodność typów pól. Ponieważ w produktach Microsoft wszystkie zmienne są przekazywane przez typ *variant* (dopasowujący się do typu danych), zgodność typu w tym