W klauzuli WHERE możemy stosować wszystkie standardowe operatory porównania: = (równy), <> (różny), > (większy), >= (większy lub równy), < (mniejszy), <= (mniejszy lub równy).

Do łączenia warunków stosujemy operatory logiczne AND (i) oraz OR (lub).

Przykład 3. Stosowanie operatora logicznego AND

Pobieranie listy owoców w cenie od 2 do 4 zł za kilogram:

```
SELECT NazwaOwocu, CenaKg FROM Owoce WHERE CenaKg>=2 AND CenaKg<=4;
```

Ćwiczenie 3.

Przygotuj kwerendę zgodnie z opisem podanym w przykładzie 2. Zmodyfikuj ją tak, aby wyświetlane były tylko zamówienia dokonane przez Janusza Nowaka.

Ćwiczenie 4.

Napisz kwerendę, która wyświetli listę zamówień na jabłka. W tabeli wynikowej powinny być widoczne kolumny z imieniem i nazwiskiem klienta, datą złożenia i datą realizacji zamówienia oraz liczbą kilogramów.

Zwykle w zapytaniach podajemy nazwy pól, bez określania tabel. Nawet jeśli pobieramy dane z kilku tabel, to baza potrafi rozpoznać, z której z nich pochodzi dane pole. Tylko w sytuacji, kiedy w dwóch tabelach umieszczonych w klauzuli FROM znajdą się pola o takich samych nazwach, odwołując się do nich, musimy podawać nazwę tabeli oddzieloną kropką od nazwy pola: *Tabela.Pole*.

3. Wybrane klauzule instrukcji SELECT

Omówiliśmy instrukcję SELECT z jej podstawowymi klauzulami – FROM i WHERE. Pokażemy teraz, jak ograniczać zakres pobieranych danych do unikalnych rekordów, sortować je i grupować. Opiszemy także sposoby łączenia danych z kilku tabel.

```
SELECT DISTINCT Pole1 FROM Tabela1;
```

Dodanie słowa kluczowego DISTINCT po słowie SELECT powoduje, że w wynikach zapytania nie będą zwracane rekordy zawierające takie same wartości we wszystkich wymienionych polach. Otrzymamy więc po jednym rekordzie dla każdej kombinacji wartości pól wymienionych w klauzuli SELECT.

KPrzyklad4

Miejscowosc
Leszczyna
Mietków
Oborniki Śląskie
Wrocław

Rys. 6. *Wynik działania kwerendy (przykład 4.)*

Przykład 4. Stosowanie słowa kluczowego DISTINCT

Zapytanie:

```
SELECT DISTINCT Miejscowosc FROM Dostawcy;
```

zwraca listę miejscowości, w których mamy dostawców. Nawet jeśli kilku dostawców pochodzi z jednej miejscowości, zostanie ona wymieniona na liście tylko raz (rys. 6.).

Ćwiczenie 5.

Utwórz kwerendę, korzystając z przykładu 4.

3.1. Klauzula ORDER BY

```
SELECT * FROM Tabela1 ORDER BY Pole1;
```

Klauzula ORDER BY pozwala sortować zwracany przez instrukcję SELECT zbiór rekordów.

Możemy podać listę kolumn, według których ma się odbyć sortowanie, oraz dla każdej z nich określić porządek sortowania (malejący lub rosnący).

Przykład 5. Stosowanie klauzuli ORDER BY

Zapytanie:

```
SELECT Nazwisko, Imie FROM Klienci ORDER BY
Nazwisko, Imie;
```

zwróci listę klientów posortowanych w porządku alfabetycznym (rys. 7.). Możemy także określić odwrotny porządek sortowania:

```
SELECT Nazwisko, Imie FROM Klienci ORDER BY
Nazwisko DESC, Imie DESC;
```

Słowo DESC obok nazwy kolumny oznacza sortowanie w porządku malejącym (DESC to skrót angielskiego słowa *descending* – „schodzenie", „schodzący").

ASC (skrót angielskiego słowa *ascending*) oznacza sortowanie w porządku rosnącym.

Zauważmy, że odwrotny porządek sortowania określamy dla każdej kolumny z osobna.

KPrzyklad5

Nazwisko	Imie
Barucha	Wacław
Celińska	Maria
Fąfara	Henryk
Kowal	Adam
Krajewski	Kacper
Lindert	Tomasz
Nowak	Janusz
Nowak	Marian
Pietrzak	Maria
Romanowski	Adam
Szynalski	Michał
Wójcik	Anna
Zając	Alicja
Zając	Tomasz

Rys. 7. *Wynik działania kwerendy z klauzulą ORDER BY (przykład 5.)*

Ćwiczenie 6.

Zmodyfikuj kwerendę z przykładu 2. tak, aby lista zamówień była wyświetlana według dat ich złożenia.

3.2. Klauzula INNER JOIN

```
SELECT * FROM Tabela1 INNER JOIN Tabela2 ON Tabela1.Id=Tabela2.Id;
```

Klauzula INNER JOIN dokonuje wewnętrznego złączenia tabel. Wyświetlane są te rekordy, dla których w polu wspólnym dla dwóch tabel znajdują się takie same wartości.

Zapytanie:

```
SELECT Klienci.Nazwisko, Klienci.Imie, Zamowienia.DataZlozenia,
Zamowienia.DataRealizacji FROM Klienci INNER JOIN Zamowienia ON Klienci.
IdKlienta = Zamowienia.IdKlienta;
```

[1]

zwraca dokładnie takie same rezultaty jak:

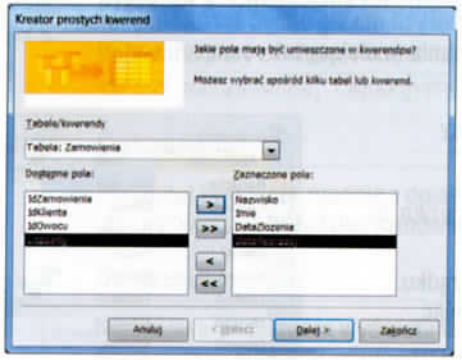
```
SELECT Klienci.Nazwisko, Klienci.Imie, Zamowienia.DataZlozenia,
Zamowienia.DataRealizacji FROM Klienci, Zamowienia WHERE Klienci.
IdKlienta=Zamowienia.IdKlienta;
```

[2]

Program Microsoft Access stosuje klauzulę INNER JOIN w przypadku, gdy w oknie projektu kwerendy tworzymy kwerendę pobierającą dane z więcej niż jednej tabeli.

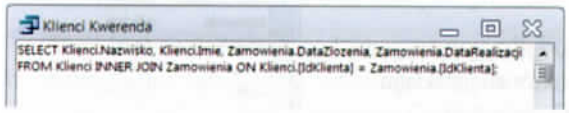
Przykład 6. Stosowanie klauzuli INNER JOIN

Za pomocą **Kreatora prostych kwerend** (rys. 8.) utworzymy kwerendę wyświetlającą listę zamówień złożonych przez określonych klientów. Wybieramy do wyświetlania pola *Nazwisko* i *Imie* z tabeli *Klienci* oraz *DataZlozenia* i *DataRealizacji* z tabeli *Zamowienia*.



Rys. 8.** Okno **Kreatora prostych kwerend

Po utworzeniu kwerendy przechodzimy do **Widoku SQL**. Możemy zobaczyć, w jaki sposób program Microsoft Access zrealizował nasze zapytanie (rys. 9.). Takie same rezultaty można uzyskać, pisząc kwerendę [2].



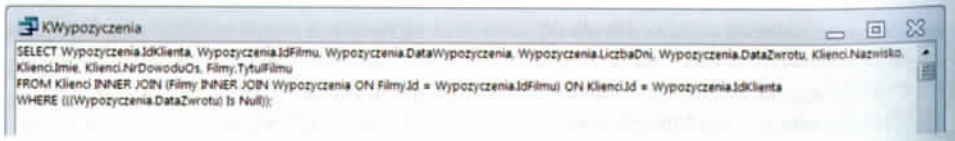
***Rys. 9.** Okno z widokiem kodu SQL kwerendy (przykład 6.)*

Ćwiczenie 7.

Zmodyfikuj kwerendę z przykładu 6. tak, aby dodatkowo wyświetlana była liczba kilogramów i nazwa zamówionego owocu. Przeanalizuj jej kod SQL.

Ćwiczenie 8.

Otwórz bazę SOWy (którą tworzyłeś w tematach 19-22), a w niej kwerendę *KWypozyczenia*. Przeanalizuj jej kod SQL (rys. 10.). Jakie występują w niej klauzule instrukcji SQL? Omów ich przeznaczenie.



***Rys. 10.** Okno z widokiem kodu SQL kwerendy* KWypozyczenia *z bazy SOWy*

3.3. Klauzula LEFT JOIN

```
SELECT * FROM Tabela1 LEFT JOIN Tabela2 ON Tabela1.Id=Tabela2.Id;
```

Klauzula `LEFT JOIN` dokonuje lewostronnego złączenia tabel.

Oznacza to, że w wynikach będą uwzględnione wszystkie rekordy z tabeli znajdującej się po lewej stronie napisu `LEFT JOIN` (*Tabela1*), nawet jeśli w tabeli po prawej stronie (*Tabela2*) nie ma odpowiadających im wpisów. W takim przypadku do pól z tabeli *Tabela1* dołączone będą pola o wartościach **Null**, odpowiadające polom z tabeli *Tabela2*.

Przykład 7. Stosowanie klauzuli LEFT JOIN

Chcemy uzyskać listę dostawców wraz z ich adresami korespondencyjnymi. Jednak nie każdy dostawca posiada taki adres.

Zapytanie:

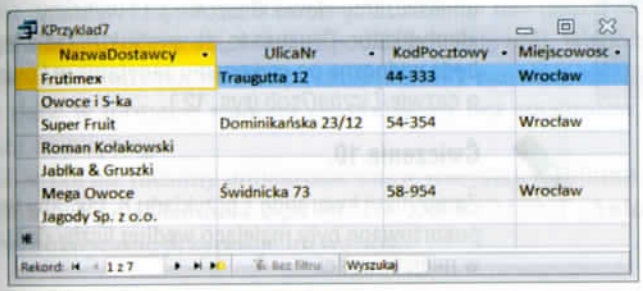
```
SELECT * FROM Dostawcy, AdresyKorespondencyjne WHERE Dostawcy.
IdDostawcy = AdresyKorespondencyjne.IdDostawcy;
```

zwróci tylko tych dostawców, którzy mają adres korespondencyjny.

Aby uzyskać listę wszystkich dostawców, należy zastosować złączenie lewostronne:

```
SELECT * FROM Dostawcy LEFT JOIN AdresyKorespondencyjne ON Dostawcy.
IdDostawcy = AdresyKorespondencyjne.IdDostawcy;
```

Wyniki działania kwerendy ze złączeniem lewostronnym widoczne są na rysunku 11. (liczba wyświetlanych pól została tu ograniczona). Rekordy z tabeli *Dostawcy* bez przypisanych adresów korespondencyjnych mają wartości **Null** w polach odpowiadających adresowi korespondencyjnemu.



***Rys. 11.** Wyniki działania kwerendy ze złączeniem lewostronnym (przykład 7.)*

Ćwiczenie 9.

Wykonaj kwerendę z przykładu 7, dokonującą lewostronnego złączenia danych z tabel *Dostawcy* i *AdresyKorespondencyjne*. Dopisz w wynikach kwerendy dane adresowe jednego z dostawców. Zaobserwuj efekty.

3.4. Klauzula GROUP BY

```
SELECT Pole1, <funkcja agregująca> FROM Tabela1 GROUP BY Pole1;
```

Klauzula `GROUP BY` jest stosowana w celu grupowania rekordów posiadających identyczne wartości w wymienionych polach.

Wywołanie: `SELECT Pole1 FROM Tabela1 GROUP BY Pole1;`
daje takie same rezultaty jak:
`SELECT DISTINCT Pole1 FROM Tabela1;`
Klauzula `GROUP BY` pozwala dodatkowo stosować funkcje agregujące na zgrupowanych rekordach.
Funkcje agregujące pozwalają obliczać wartości na podstawie grup rekordów.

Najczęściej stosuje się funkcje:

- `COUNT(*)` – zwraca liczbę rekordów w każdej grupie,
- `SUM(Liczba1)` – zwraca sumę wartości w kolumnie *Liczba1* w każdej grupie,
- `AVG(Liczba2)` – zwraca średnią wartość w kolumnie *Liczba2* w każdej grupie,
- `MAX(Liczba3)` – zwraca maksymalną wartość zapisaną w kolumnie *Liczba3* w każdej grupie,
- `MIN(Liczba4)` – zwraca minimalną wartość zapisaną w kolumnie *Liczba4* w każdej grupie.

Zastosowanie funkcji `COUNT` i `AVG` pokażemy na przykładach.

Przykład 8. Stosowanie funkcji `COUNT`

Policzymy, ilu dostawców mamy w każdej miejscowości. Należy wybrać rekordy z tabeli *Dostawcy*, grupując je według pola *Miejscowosc*, i skorzystać z funkcji agregującej `COUNT(*)`:

```
SELECT COUNT(*) AS LiczbaOsob, Miejscowosc
FROM Dostawcy GROUP BY Miejscowosc;
```

KPrzykład8

LiczbaOsob	Miejscowosc
1	Leszczyna
1	Mietków
1	Oborniki Śląskie
4	Wrocław

Rekord: 1 z 4

***Rys. 12.** Wynik działania kwerendy (przykład 8.)*

Warto zwrócić uwagę, że po funkcji `COUNT(*)` umieszczamy słowo kluczowe `AS` i wybrany identyfikator. Oznacza to, że wyniki działania tej funkcji będą widoczne po wykonaniu zapytania w kolumnie o nazwie *LiczbaOsob* (rys. 12.).

Ćwiczenie 10.

Zmodyfikuj kwerendę z przykładu 8. tak, aby wyniki posortowane były malejąco według liczby dostawców w miejscowości.

Wskazówka: Umieść w klauzuli `ORDER BY LiczbaOsob`.

Przykład 9. Stosowanie funkcji `AVG`

Napiszemy kwerendę, która policzy średnią wartość zamówień dokonywanych przez poszczególnych klientów i posortuje rekordy malejąco według tej wartości (rys. 13.).

Skorzystamy z funkcji agregującej AVG:

```
SELECT Klienci.Nazwisko, Klienci.
Imie, AVG(Zamowienia.LiczbaKg*Owoce.
CenaKg) AS SrednieZamowienie
FROM Klienci LEFT JOIN (Zamowienia
LEFT JOIN Owoce ON Zamowienia.
IdOwocu=Owoce.IdOwocu) ON Klienci.
IdKlienta=Zamowienia.IdKlienta
GROUP BY Klienci.Nazwisko, Klienci.
Imie, Zamowienia.IdKlienta ORDER
BY SrednieZamowienie DESC;
```

KPrzykład9

Nazwisko	Imie	SrednieZamowienie
Nowak	Marian	2 120,00 zł
Nowak	Janusz	1 450,00 zł
Zając	Alicja	650,00 zł
Romanowski	Adam	580,00 zł
Barucha	Wacław	450,00 zł
Fąfara	Henryk	348,00 zł
Szynalski	Michał	250,00 zł
Pietrzak	Maria	240,00 zł
Lindert	Tomasz	234,00 zł
Wójcik	Anna	230,00 zł
Zając	Tomasz	135,00 zł
Krajewski	Kacper	
Kowal	Adam	

Rekord: 1 z 14 Bez filtru Wyszuka

***Rys. 13.** Wynik działania kwerendy (przykład 9.)*

Można wyświetlać tylko wartości funkcji zagregowanych i pola, które są wymienione w klauzuli `GROUP BY`. Dlatego warto zauważyć, że aby wyświetlić wartości pól *Imie* i *Nazwisko* klienta, musimy ich nazwy umieścić po `GROUP BY`.

Ćwiczenie 11.

Zmodyfikuj kwerendę z przykładu 9. tak, aby wyświetlana była również suma wartości zamówień dokonanych przez klienta (rekordy powinny być posortowane malejąco według tej sumy). Na liście nie wyświetlaj klientów, którzy nie dokonali żadnych zamówień.

4. Dopisywanie rekordów

Nowe rekordy dodajemy do tabeli bazy, korzystając z instrukcji `INSERT`:

```
INSERT INTO Tabela1 (Pole1, Pole2) VALUES ('Wartosc1', 'Wartosc2');
```

Po nazwie tabeli wpisujemy w nawiasach okrągłych listę pól (oddzielonych przecinkami). Następnie umieszczamy słowo kluczowe `VALUES` i w nawiasach okrągłych podajemy listę wartości (w takiej kolejności, w jakiej podaliśmy listę pól). Wartości takie, jak napisy, numery telefonów, kody pocztowe itp., umieszczamy zawsze pomiędzy znakami apostrofu. Liczby i dane w formacie walutowym podajemy bez apostrofów, używając kropki jako separatora dziesiętnego.

Na liście pól nie musimy umieszczać nazw wszystkich kolumn występujących w tabeli. Pomijamy na przykład pola typu AUTO_INCREMENT (**Autonumerowanie**) – baza danych automatycznie przypisuje im wartość.

Przykład 10. Dodawanie rekordów

Dodamy do bazy dane nowego owocu. Wywołamy w tym celu polecenie:

```
INSERT INTO Owoce (NazwaOwocu, CenaKg) VALUES ('Banany', 4.70);
```

Ćwiczenie 12.

Napisz instrukcję `INSERT`, która doda dane nowego klienta do tabeli *Klienci*.

5. Aktualizacja danych

Instrukcja `UPDATE` pozwala modyfikować dane w bazie.

```
UPDATE Tabela1 SET Pole1='Wartosc1', Pole2='Wartosc2' WHERE Id=1;
```

W instrukcji `UPDATE` stosujemy klauzulę `SET`, aby określić wartości, jakie chcemy zapisać w bazie (podajemy nazwę pola, a po znaku równości wartość, która zostanie mu przypisana). Opcjonalna klauzula `WHERE` określa, które rekordy zostaną zaktualizowane.

Przykład 11. Stosowanie klauzuli SET

Napiszemy instrukcję, która podwyższy ceny wszystkich owoców o 10%:

```
UPDATE Owoce SET CenaKg = CenaKg * 1.10;
```

Instrukcja ta poleca bazie danych pomnożyć wartość pola *CenaKg* dla każdego owocu przez liczbę 1.10 (pamiętaj, że do zapisu liczb używamy kropki dziesiętnej zamiast przecinka) i zapisać jako nową cenę.

Można również dodać klauzulę WHERE, aby na przykład zmienić cenę owocu o konkretnym *Id*:

```
UPDATE Owoce SET CenaKg = 2.00 WHERE IdOwocu = 4;
```

Można też zmienić cenę owocu o konkretnej nazwie:

```
UPDATE Owoce SET CenaKg = CenaKg + 0.30 WHERE NazwaOwocu = 'Wiśnie';
```

Ćwiczenie 13.

Napisz instrukcję, która obniży o 15% ceny owoców kosztujących więcej niż 2 złote za kilogram.

6. Usuwanie rekordów

Instrukcja DELETE pozwala usuwać niepotrzebne rekordy z tabel.

```
DELETE FROM Tabela1 WHERE Id=1;
```

Klauzula WHERE określa, które rekordy mają zostać usunięte.

W przypadku instrukcji DELETE, podobnie jak dla instrukcji UPDATE, należy dokładnie sprawdzać klauzulę WHERE. Po wykonaniu zapytania cofnięcie wprowadzonych zmian nie jest możliwe.

Przykład 12. Usuwanie jednego rekordu

Załóżmy, że omyłkowo wprowadziliśmy do bazy dane nowego owocu. Nowemu rekordowi został nadany numer *Id* 12. Możemy usunąć ten rekord, wywołując kwerendę:

```
DELETE FROM Owoce WHERE IdOwocu = 12;
```

Zastosowanie instrukcji DELETE jest ograniczone przez więzy integralności. Nie uda się nam usunąć z bazy rekordu, do którego odwołania znajdują się w innych tabelach. Oznacza to na przykład, że nie będziemy mogli usunąć danych klienta, który wcześniej złożył zamówienia. W takim przypadku musimy najpierw usunąć wszystkie rekordy z tabeli *Zamowienia* powiązane z danym klientem – wraz z nimi znikną odwołania do danych tego klienta. Dopiero potem można usunąć dane klienta. Przypomnijmy, że z podobną sytuacją spotkaliśmy się w systemie SOWy, gdy chcieliśmy usunąć dane klienta wypożyczalni filmów, który miał wypożyczone filmy.

Przykład 13. Usuwanie wielu rekordów

Wykorzystamy mechanizm chroniący rekordy, do których istnieją odwołania, aby usunąć z bazy dane klientów, którzy nigdy nie składali zamówień. Wystarczy wywołać zapytanie:

```
DELETE FROM Klienci;
```

Celowo pominęliśmy klauzulę WHERE – tak sformułowane zapytanie poleca usunąć wszystkie rekordy z tabeli.

Jeśli jednak mamy prawidłowo zdefiniowane relacje, baza pozwoli usunąć tylko te rekordy, które nie są powiązane więzami integralności (czyli klientów, którzy nie składali zamówień).

Ćwiczenie 14.

Dodaj nowe zamówienie, korzystając z formularza *FZamowienia*. Następnie usuń je, korzystając z instrukcji DELETE (klient rozmyślił się).

Warto zapamiętać

- Głównym zastosowaniem języka SQL jest wykonywanie zapytań (kwerend) do bazy danych. Zapytania służą do pobierania, dopisywania, aktualizacji i usuwania rekordów oraz tworzenia tabel.
- Wszystkie kwerendy w programie Microsoft Access są przechowywane jako instrukcje SQL, nawet te projektowane przy użyciu narzędzi tego programu, np. kreatora. Znajomość języka SQL jest niezbędna przy tworzeniu rozbudowanych kwerend.
- Za pomocą instrukcji SELECT można pobrać pola z wybranej tabeli. Korzystając z klauzul tej instrukcji, możemy określać dodatkowe warunki dotyczące wybieranych danych.

Pytania, problemy

1. Jakie są zastosowania języka SQL?
2. Do czego służy instrukcja SELECT?
3. W jaki sposób napisać własną kwerendę, stosując język SQL?
4. Podaj przykładowe operacje, które można wykonać na rekordach bazy, korzystając z klauzul instrukcji SELECT.
5. Omów zastosowanie instrukcji INSERT, UPDATE i DELETE.

Zadania

1. Otwórz bazę *Hurtownia.mdb* (CD). Napisz w języku SQL kwerendę, która:
   a. wyświetli wszystkich klientów pochodzących z Wrocławia,
   b. wyświetli listę dostawców z Wrocławia, którzy posiadają adresy korespondencyjne,
   c. wyświetli nazwy owoców, które kosztują od 2 do 4 zł za kilogram,
   d. zmieni numer telefonu Mariana Nowaka na 888-01-01,
   e. doda nowy owoc – arbuzy w cenie 1,5 zł za kilogram,
   f. usunie z bazy dane klienta o *Id* 14, jeśli nie złożył żadnego zamówienia.
2. Zmodyfikuj kwerendę *KListaZamowien* w bazie *Hurtownia* tak, aby wyświetlana była lista niezrealizowanych zamówień.
   Wskazówka: Niezrealizowane zamówienia mają wartość **Null** w polu *DataRealizacji*. Wyszukasz je, umieszczając w klauzuli WHERE warunek *DataRealizacji* IS NULL.
3. Napisz kwerendę, która wyświetli liczbę zamówień dokonanych przez poszczególnych klientów w bazie *Hurtownia* (bez uwzględniania klientów, którzy nie dokonali żadnych zamówień).
4. Otwórz bazę SOWy, a w niej formularz *FZwroty*. Zobacz kod SQL kwerendy, która sortuje dane klienta w tym formularzu (temat 22, przykład 5.). Jakie występują w niej klauzule instrukcji SQL? Omów ich przeznaczenie.