BAZY DANYCH

Lab. 4 Podstawy języka zapytań SQL

Laboratorium nr 4

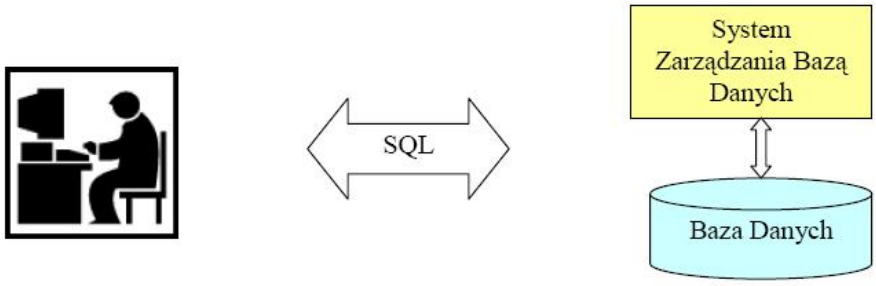
Temat: Podstawy języka zapytań SQL

Systemy Zarządzania Bazą Danych (SZBD) pozwalają na rozwiązanie wielu problemów związanych ze składowaniem, przeszukiwaniem i przekształcaniem danych w bazach danych. Poszczególne czynności, które mają zostać wykonane przez SZBD, programista opisuje w specjalnym języku. Celem tego ćwiczenia, jest zapoznanie Studentów z podstawami języka SQL, który jest jednym z najbardziej popularnych języków baz danych, stosowanym w prawie każdym komercyjnym SZBD na rynku.

PLAN LABORATORIUM:

1. Wprowadzenie do laboratorium
2. Podstawowe informacje o języku SQL
3. Najprostsze zapytania
4. Projekcja
5. Literały
6. Aliasy
7. Funkcje
8. Sortowanie
9. Warunek WHERE
10. Eliminacja powtórzeń - DISTINCT
11. Wyrażenia: BETWEEN...AND, IN, LIKE, IS NULL
12. Porcjowanie wyników zapytania – LIMIT

1. Wprowadzenie do laboratorium



Ćwiczenia z przedmiotu „Bazy Danych" są poświęcone przede wszystkim językowi SQL (ang. *Structured Query Language)* i jego zastosowaniom. Język SQL jest strukturalnym językiem zapewniającym możliwość wydawania poleceń do systemu zarządzania bazą danych (SZBD). Dzięki temu językowi możliwe jest tworzenie poleceń, które nakazują SZBD odnaleźć potrzebne przez nas dane w bazie danych, jak również poleceń, dzięki którym można składowane dane zmodyfikować, wstawić nowe dane, bądź je usunąć. Dzięki temu językowi możliwe jest również definiowanie, modyfikacja i usuwanie struktur danych, w których dane są składowane. Język SQL pozwala również na zarządzanie transakcjami i mechanizmami autoryzacji dostępu do danych.

2. Podstawowe informacje o języku SQL

- **SQL jest językiem deklaratywnym** (Charakterystyczną cechą języków deklaratywnych jest to, że opisują „co" ma być zrobione, ale nie „jak". W praktyce oznacza to, że użytkownik opisuje w języku SQL efekt jaki chce uzyskać (odczytanie telefonów wszystkich pracowników o nazwisku zaczynającym się na literę 'A', podniesienie pensji wszystkim kierownikom itp.), ale nie sposób w jaki ma to być zrobione (sekwencja operacji dyskowych, które prowadzą do wykonania polecenia). Sposób wykonania polecenia jest automatycznie dobierany przez SZBD i zależy od fizycznego sposobu składowania danych).
- **Język SQL jest zorientowany na przetwarzanie zbiorów** (Ponieważ dane są przechowywane w relacjach, które są zbiorami krotek, konstrukcje języka SQL dotyczą przetwarzania zbiorów i nie zawierają poleceń uwzględniających dowolny porządek na krotkach)

Język SQL można podzielić na:

- **DML (ang. *Data Manipulation Language)*** - język manipulacji danymi pozwalający na odczytywanie danych z relacji (polecenie SELECT) oraz na wstawianie, modyfikację i usuwanie danych z relacji (polecenia: INSERT, UPDATE, MERGE i DELETE).
- **DDL (ang. *Data Definition Language)*** - język definicji danych pozwalający na tworzenie, modyfikację i usuwanie relacji (polecenia CREATE, ALTER i DROP).
- **DCL (ang. *Data Control Language)*** - język kontroli danych pozwalający na zapewnienie autoryzacji dostępu do danych oraz zarządzanie transakcjami. Najważniejsze polecenia to GRANT i REVOKE (czasem zaliczane do DDLa) oraz COMMIT, ROLLBACK i SAVEPOINT.

W języku SQL nie rozróżnia się dużych i małych liter - Wszystkie słowa kluczowe i nazwy (np. relacji i atrybutów) w języku SQL można pisać zarówno dużymi jak i małymi literami.

Przykładowo, wyrażenia: nazwisko, Nazwisko i NAZWISKO są identyczne.

W poleceniach SQL ignorowane są znaki końca linii - Wszystkie słowa kluczowe i wyrażenia w języku SQL można rozdzielać zarówno spacjami jak i znakami końca linii. W rezultacie dowolne polecenie SQL można sformatować w dowolny sposób (może ono zajmować jeden długi wiersz, lub kilka krótszych wierszy).

Każde polecenie SQL powinno być zakończone średnikiem - Próba wykonania polecenia nie zakończonego w ten sposób może zakończyć się błędem, chociaż istnieją sytuacje, gdy pominięcie średnika jest dozwolone.

Ćwiczenia z języka SQL zostaną przeprowadzone na bazie danych złożonej z 6 relacji przechowujących dane dotyczące wypożyczeń w bibliotece.

SQL Query Area

```
1 desc czytelnicy
```

Field	Type	Null	Key	Default	Extra
Nr_czytelnika	int(11)	NO	PRI	NULL	auto_increment
Nazwisko	varchar(35)	YES		NULL	
Imie	varchar(15)	YES		NULL	
Data_ur	date	YES		NULL	
Ulica	varchar(40)	YES		NULL	
Kod	int(11)	YES		NULL	
Miasto	varchar(40)	YES		NULL	
Data_zapisania	date	YES		NULL	
Data_skreslenia	date	YES		NULL	
Nr_legitymacji	int(11)	YES		NULL	
Funkcja	enum('PD','S')	YES		NULL	
Plec	enum('K','M')	YES		NULL	

SQL Query Area

```
1 desc pracownicy
```

Field	Type	Null	Key	Default	Extra
Id_pracownika	int(11)	NO	PRI	NULL	auto_increment
Nazwisko	varchar(50)	YES		NULL	
Imie	varchar(40)	YES		NULL	
Id_stanowisko	int(11)	YES	MUL	NULL	
Data_zatrudnienia	date	YES		NULL	
wynagrodzenie	decimal(10,2)	YES		NULL	

SQL Query Area

```
1 desc dzialy
```

Field	Type	Null	Key	Default	Extra
Id_dzial	int(11)	NO	PRI	NULL	auto_increment
Nazwa	varchar(40)	YES		NULL	

```
1 desc stanowiska
```

Field	Type	Null	Key	Default	Extra
Id_stanowisko	int(11)	NO	PRI	NULL	auto_increment
Nazwa	varchar(40)	YES		NULL	

```
1 desc ksiazki
```

Field	Type	Null	Key	Default	Extra
Sygnatura	int(11)	NO	PRI	NULL	auto_increment
Tytul	varchar(100)	YES		NULL	
Nazwisko	varchar(50)	YES		NULL	
Imie	varchar(40)	YES		NULL	
Wydawnictwo	varchar(40)	YES		NULL	
Miejsce_wyd	varchar(40)	YES		NULL	
Rok_wyd	int(11)	YES		NULL	
Objetosc_ks	int(11)	YES		NULL	
Cena	decimal(10,2)	YES		NULL	
Id_dzial	int(11)	YES	MUL	NULL	

```
1 desc wypozyczenia
```

Field	Type	Null	Key	Default	Extra
Nr_transakcji	int(11)	NO	PRI	NULL	auto_increment
Sygnatura_ksiazki	int(11)	NO	MUL	NULL	
Id_pracownika	int(11)	NO	MUL	NULL	
Nr_czytelnika	int(11)	NO	MUL	NULL	
Data_wypozyczenia	date	NO		NULL	
Data_zwrotu	date	YES		NULL	

3. Najprostsze zapytania

Naukę języka SQL zaczniemy od zapoznania ze składnią polecenia **SELECT**. Jest to polecenie pozwalające na odczytywanie danych z bazy danych oraz wykonywanie na tych danych prostych obliczeń i przekształceń. Polecenie **SELECT** pobiera krotki z relacji w bazie danych, przetwarza je (opcjonalnie) i zwraca wynik w postaci zbioru odczytanych krotek. W wyniku wykonania polecenia **SELECT** otrzymujemy zatem relację (zbiór krotek), tzw. „relację wynikową". Aplikacje klienckie, pozwalające na bezpośrednie wykonywanie poleceń SQL przedstawiają relację wynikową postaci tabelarycznej. Ponieważ polecenia **SELECT** służą do odczytywania danych w bazie danych, nazywa się te polecenia „zapytaniami". Najprostszą wersją polecenia SELECT, jest polecenie postaci:

„SELECT * FROM {nazwa relacji};"

Polecenie odczytujące dane z relacji rozpoczyna się zawsze słowem kluczowym SELECT po którym podaje się listę atrybutów, które mają zostać odczytane. Sposób definiowania listy atrybutów zostanie opisany później. Wstawiona po słowie kluczowym SELECT gwiazdka oznacza „odczytaj wszystkie atrybuty". Następnie, umieszcza się w poleceniu słowo kluczowe FROM, po którym podaje się nazwę relacji z której mają zostać odczytane krotki. Polecenie SELECT o postaci SELECT * FROM {nazwa relacji}; powoduje odczytanie wszystkich krotek i wszystkich atrybutów z relacji o podanej nazwie. Przykładowo, polecenie odczytujące całą zawartość relacji *dzialy* wygląda następująco:

```
1 select * from dzialy
```

Id_dzial	Nazwa
1	Informatyka
2	Ekonomia
3	Fantastyka
4	Historia
5	Prawo
6	Literatura dla dzieci i modziey
7	Literatura
8	Medyczne
9	Przyrodnicze

Powyższe polecenie można przetłumaczyć na język naturalny następująco: „Odczytaj wszystkie krotki z relacji o nazwie *dzialy,* zachowując wszystkie atrybuty krotek (*)".

W wyniku takiego zapytania SZBD odczyta z bazy danych, z relacji *dzialy,* wszystkie krotki i w postaci niezmienionej zwróci je aplikacji klienckiej, która w naszym przypadku wyświetli je na ekranie. Należy zwrócić uwagę na następującą rzecz. Relacje są zbiorami, a zatem kolejność zwracania krotek przez SZBD i ich wyświetlania przez aplikację kliencką, jest dowolna.

4. Projekcja

W większości przypadków nie ma konieczności odczytywania wszystkich atrybutów krotek z relacji. Najczęściej istotny jest tylko niewielki ich podzbiór. W takich sytuacjach można skorzystać z nieco bardziej skomplikowanej wersji polecenia SELECT, w której zamiast znaku „*" wymienia się listę nazw atrybutów, które mają się znaleźć w wyniku. Poszczególne atrybuty na tej liście są rozdzielane za pomocą przecinków. Proces wybierania atrybutów, które mają się znaleźć w wyniku nazywa się „projekcją".

Rozszerzona wersja polecenia SELECT wygląda następująco:

SELECT{atrybutl, atrybut2, FROM {nazwa relacji};

Gdzie „atrybut" to nazwa atrybutu zdefiniowanego w relacji podanej za słowem kluczowym **FROM**.

SQL Query Area

```
1 select nazwisko, imie, wynagrodzenie
2 from pracownicy
```

nazwisko	imie	wynagrodzenie
Kowalczuk	Jan	1700.00
Czuj	Krystyna	2850.00
Brzeski	Mateusz	2900.00
Darecki	Antoni	2700.00
Molek	Anna	1200.00
Potepa	Krzysztof	9000.00
Potepa	Wojciech	1900.00
Tomaszewki	Radosaw	2100.00
Ignatowicz	Emilia	2000.00
Potepa	Mariusz	1700.00
Borowik	Lukasz	1900.00
Malinowski	Dariusz	3000.00
Zielinska	Danuta	2000.00
Makarski	Tomasz	2000.00
Zielonka	Mateusz	2000.00

Polecenie odczyta z relacji *pracownicy* wszystkie krotki, ale zwróci jedynie wartości dotyczące atrybutów nazwisko, imie i wynagrodzenie.

5. *Literały*

W klauzuli **SELECT** można umieszczać literały. Literałem jest dowolny ciąg znaków lub liczb. W wyniku umieszczenia literałów system bazy danych do każdego zwróconego wiersza wpisze do odpowiedniej kolumny podany ciąg znaków.

```
1 select nazwisko, imie,' zarabia ', wynagrodzenie
2 from pracownicy
```

nazwisko	imie	zarabia	wynagrodzenie
Kowalczuk	Jan	zarabia	1700.00
Czuj	Krystyna	zarabia	2850.00
Brzeski	Mateusz	zarabia	2900.00
Darecki	Antoni	zarabia	2700.00
Molek	Anna	zarabia	1200.00
Potepa	Krzysztof	zarabia	9000.00
Potepa	Wojciech	zarabia	1900.00
Tomaszewki	Radosaw	zarabia	2100.00
Ignatowicz	Emilia	zarabia	2000.00
Potepa	Mariusz	zarabia	1700.00
Borowik	Lukasz	zarabia	1900.00
Malinowski	Dariusz	zarabia	3000.00
Zielinska	Danuta	zarabia	2000.00
Makarski	Tomasz	zarabia	2000.00
Zielonka	Mateusz	zarabia	2000.00

6. Aliasy

Aliasy nie są obowiązkowe i najlepiej stosować je jedynie do wyrażeń bardziej skomplikowanych niż sama nazwa atrybutu. Aby nadać wyrażeniu alias, należy za tym wyrażeniem użyć słowa kluczowego AS, a następnie podać alias. Jeżeli alias składa się ze słów ze spacja wówczas umieszczamy go w apostrofach np.: **…AS ‘Dane osób’**

```
SQL Query Area
1 select nazwisko, imie, wynagrodzenie AS 'Kwota Wynagrodzenia'
2 from pracownicy
```

nazwisko	imie	Kwota Wynagrodzenia
Kowalczuk	Jan	1700.00
Czuj	Krystyna	2850.00
Brzeski	Mateusz	2900.00
Darecki	Antoni	2700.00
Molek	Anna	1200.00
Potepa	Krzysztof	9000.00
Potepa	Wojciech	1900.00
Tomaszewki	Radosaw	2100.00
Ignatowicz	Emilia	2000.00
Potepa	Mariusz	1700.00
Borowik	Lukasz	1900.00
Malinowski	Dariusz	3000.00
Zielinska	Danuta	2000.00
Makarski	Tomasz	2000.00
Zielonka	Mateusz	2000.00

W kolejnym przykładzie zostanie zastosowany alias dla określenia nazwy dla funkcji łączenia – **CONCAT()**.

```
SQL Query Area
1 select CONCAT(nazwisko,' ', imie, ' zarabia ', wynagrodzenie,' zł')
2 AS 'Dane osob i wynagrodzenie'
3 from pracownicy
```

Dane osob i wynagrodzenie
Kowalczuk Jan zarabia 1700.00 zł
Czuj Krystyna zarabia 2850.00 zł
Brzeski Mateusz zarabia 2900.00 zł
Darecki Antoni zarabia 2700.00 zł
Molek Anna zarabia 1200.00 zł
Potepa Krzysztof zarabia 9000.00 zł
Potepa Wojciech zarabia 1900.00 zł
Tomaszewki Radosaw zarabia 2100.00 zł
Ignatowicz Emilia zarabia 2000.00 zł
Potepa Mariusz zarabia 1700.00 zł
Borowik Lukasz zarabia 1900.00 zł
Malinowski Dariusz zarabia 3000.00 zł
Zielinska Danuta zarabia 2000.00 zł
Makarski Tomasz zarabia 2000.00 zł
Zielonka Mateusz zarabia 2000.00 zł

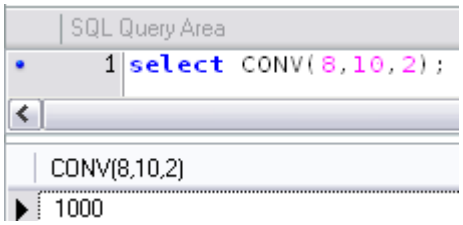
7. Funkcje

MySQL obsługuje duży zestaw funkcji, których można używać w zapytaniach. Poniżej przedstawiono przykładowe, często wykorzystywane funkcje.

- **CONCAT()-** funkcja pozwala na łączenie danych przechowywanych odrębnych kolumnach w jedną kolumnę.
- **UPPER(), LOWER()-** funkcja pozwala na zamianę łańcucha tekstowego zapisanego małymi literami na duże i na odwrót.
- **REVERSE()**- funkcja pozwala na zapisanie łańcucha tekstowego w odwrotnej kolejności liter.
- **LENGTH()**- funkcja zwraca długość łańcucha tekstowego podanego jako argument.
- **LEFT(), RIGHT()-** funkcje zwracają fragment łańcucha tekstowego podanego jako pierwszy argument o długości podanej jako drugi argument.
- **Ceil, Floor** - funkcje zaokrąglające- w górę i w dół, np. **CEIL(5.21)=6, FLOOR(4.34)=4**

  Uwaga: W SQL w celu oddzielenia części całkowitej od dziesiętnej używany ‘.’ (kropki)

- **Ln(n)** - zwraca logarytm naturalny z *n*
- **Log(n, m)** - zwraca logarytm z m przy podstawie *n*
- **Mod(m, n)** - zwraca reszte z dzielenia *m* przez *n*
- **Power(m, n)** - zwraca *m* podniesione do potęgi *n*
- **Round(m [, n])** - zwraca *m* zaokrąglone do *n* miejsc po przecinku. Dla *n* < 0, określa liczbę miejsc przed przecinkiem. Domyślnie *n* = 0.
- **Sign(n)** - zwraca −1 dla *n* < 0, 0 dla *n* = 0 oraz 1 dla *n* > 0
- **Sin(n)** - zwraca sinus kata *n* (wyrażonego w radianach)
- **Sqrt(n)** - zwraca pierwiastek kwadratowy z *n*
- **Tan(n)** - zwraca tangens kata *n* (wyrażonego w radianach)
- **Trunc(m [,n])** - zwraca *m* po obcięciu do *n* miejsc po przecinku. Dla *n* < 0, określa liczbę miejsc przed przecinkiem. Domyślnie *n* = 0
- **Conv** - zamienia systemy np, z dziesiętnego na binarny



Zamiana liczby 8 z systemu dziesiętnego na binarny

- **DAYNAME(), DAYOFMONTH(), DAYOFWEEK(), DAYOFYEAR(), MONTH(),**

MONTHNAME(), NOW(), WEEK(), YEAR() funkcje daty i czasu wymagają podania argumentu w postaci daty.

- **USER(), VERSION(), DATABASE()** - inne funkcje pomocnicze.

8. *Sortowanie*

Wyniki polecenia **SELECT** są zwracane w przypadkowej kolejności. Czasem jednak możemy być zainteresowani otrzymaniem ich w jakiejś określonej przez nas kolejności. W tym celu należy zażądać od SZBD, aby przed zwróceniem wyników zapytania, posortował je według wartości dowolnego, zdefiniowanego przez nas wyrażenia. Robi się to za pomocą klauzuli **ORDER BY** dodawanej na końcu polecenia. Za klauzulą **ORDER BY** podaje się listę wyrażeń, lub aliasów wyrażeń (zdefiniowanych przy klauzuli **SELECT**) oddzielonych przecinkami. Wynik zapytania zostanie posortowany według wartości tych wyrażeń.

Rozszerzona o klauzulę **ORDER BY** składnia polecenia **SELECT** wygląda następująco:

SELECT[DISTINCT]{wyrażeniel[AS aliasl],wyrażenie2[AS alias2]…} FROM {nazwa relacji}

ORDER BY{wyrażenie3[ASC|DESC],wyrażeniie[ASC|DESC], aliasl [ASC|DESC], alias2 [ASC|DESC]……};

W celu wyjaśnienia działania klauzuli **ORDER BY** zostanie omówionych kilka przykładowych zapytań:

1. **SELECT nazwisko FROM PRACOWNICY ORDER BY nazwisko;**

Początek zapytania można już wytłumaczyć na podstawie tego co dotychczas omówiono. **SELECT nazwisko FROM pracownicy…** znaczy: odczytaj z relacji **PRACOWNICY** wszystkie krotki, odczytaj z nich atrybut **NAZWISKO** i zwróć odczytane wartości w relacji wynikowej. Końcówka **ORDER BY** mówi SZBD, aby posortować otrzymane krotki rosnąco według nazwisk (porządek leksykograficzny).

Analizując podaną powyżej ogólną składnię polecenia **SELECT** można zauważyć, że za wyrażeniem w klauzuli **ORDER BY** można opcjonalnie podać słowo kluczowe **ASC** bądź **DESC.** Słowo to określa porządek sortowania, **ASC** oznacza sortowanie rosnąco i jest domyślne, **DESC**

oznacza sortowanie malejąco. Przykładowo, polecenie **SELECT** nazwisko FROM pracownicy **ORDER BY** nazwisko **ASC**; zwróci te same wyniki i w tej samej kolejności, co zapytanie 1.

Z kolei zapytanie:

SELECT nazwisko FROM pracownicy ORDER BY nazwisko DESC;

zwróci wyniki w odwrotnym porządku w stosunku do poprzednich zapytań.

2. **SELECT nazwisko FROM pracownicy ORDER BY wynagrodzenie;**

Zapytanie 2 podobnie jak poprzednie zapytania zwraca wszystkie nazwiska pracowników zapisane w relacji **PRACOWNICY**. Tym razem jednak wyniki są sortowane według atrybutu **WYNAGRODZENIE**. Jak łatwo zatem można zauważyć, sortować można również według atrybutów, które nie są wymienione w klauzuli **SELECT**. Jest to możliwe, gdyż projekcja odbywa się dopiero po posortowaniu krotek otrzymanych w wyniku zapytania.

3. **SELECT nazwisko FROM pracownicy ORDER BY WYNAGROZDENIE/20;**

Niniejsze zapytanie odczytuje wszystkie krotki z relacji **PRACOWNICY**, oblicza dzienną płacę każdego pracownika, sortuje otrzymane krotki według obliczonych wartości dziennej płacy i ostatecznie odczytuje z posortowanych krotek atrybut nazwisko i zwraca go w relacji wynikowej. W klauzuli **ORDER BY** można zatem podawać nie tylko nazwy kolumn, ale również wyrażenia.

4. **SELECT nazwisko,wynagrodzenie/20 AS dniowka FROM pracownicy ORDER BY dniowka;**

W niniejszym zapytaniu wyrażenie obliczające dzienną płacę pracownika umieszczono w klauzuli **SELECT** żądając tym samym, aby, prócz nazwiska, w relacji wynikowej znalazła się jego wartość. Warto również zauważyć, że wyrażeniu temu nadano alias **DNIÓWKA**, oraz że po klauzuli **ORDER BY** użyto właśnie tego aliasu. W zapytaniu tym wyniki są również sortowane według wartości dziennej płacy pracowników. Można zatem, zamiast wyrażenia, umieścić w klauzuli **ORDER BY** jego alias.

5. **SELECT nazwisko, imie, wynagrodzenie FROM egzaminatorzy ORDER BY data_zatrudnienia, wynagrodzenie;**

Niniejsze zapytanie odczytuje z pobranych z relacji EGZAMINATORZY krotek nazwisko, stanowisko i wynagrodzenie egzaminatora. W klauzuli ORDER BY umieszczono dwa proste wyrażenia, oddzielone od siebie przecinkami. Sortowanie dla tak sformułowanego polecenia odbywa się w następujący sposób. Najpierw krotki są sortowane według atrybutu **DATA_ZATRUDNIENIA**. W ramach zbioru krotek, o takiej samej wartości atrybutu **DATA_ZATRUDNIENIA**, krotki są sortowane według atrybutu **WYNAGRODZENIE**. W klauzuli **ORDER BY** można podać dowolną liczbę wyrażeń, przy czym każde kolejne jest wykorzystywane do posortowania krotek, dla których wszystkie poprzednie wyrażenia mają taką samą wartość.

Porządek sortowania zależy od typu sortowanych danych i wygląda następująco (porządek domyślny, nie zmodyfikowany za pomocą słowa kluczowego DESC).

liczby - od mniejszych do większych

daty - od wcześniejszych do późniejszych

łańcuchy znaków - alfabetycznie

wartości puste - w zależności od SZBD (najczęściej są wymieniana jako pierwsze albo ostatnie).

9. Warunek WHERE

W celu wybrania, które krotki mają się znaleźć w relacji wynikowej, stosuje się klauzulę **WHERE**. Za klauzulą **WHERE** podaje się warunek, zdefiniowany na wartościach atrybutów w relacji, który musi być spełniony, aby krotka znalazła się w relacji wynikowej. Operację wyboru krotek, które mają się znaleźć w rozwiązaniu nazywa się **„selekcją".** Składnia polecenia **SELECT**, rozszerzona o klauzulę **WHERE**, wygląda następująco:

SELECT[DISTINCT{wyrażenie1 [AS aliasl],wyrażenie2 [AS alias2]}

FROM {nazwa relacji}

WHERE warunek_elementarny

ORDERBY{wyrażenie5[ASC|DESC],wyrażenie6 ASC|DESC], alias [ASC|DESC],alias2[ASC|DESC]};

gdzie „warunek_elementarny", to porównanie jednego, dwóch lub większej liczby wyrażeń za pomocą odpowiednich operatorów logicznych.

SQL Query Area

```
1 SELECT imie, nazwisko, wynagrodzenie
2 FROM pracownicy
3 WHERE year(data_zatrudnienia)>'2005'
4 ORDER BY nazwisko DESC, imie ASC;
```

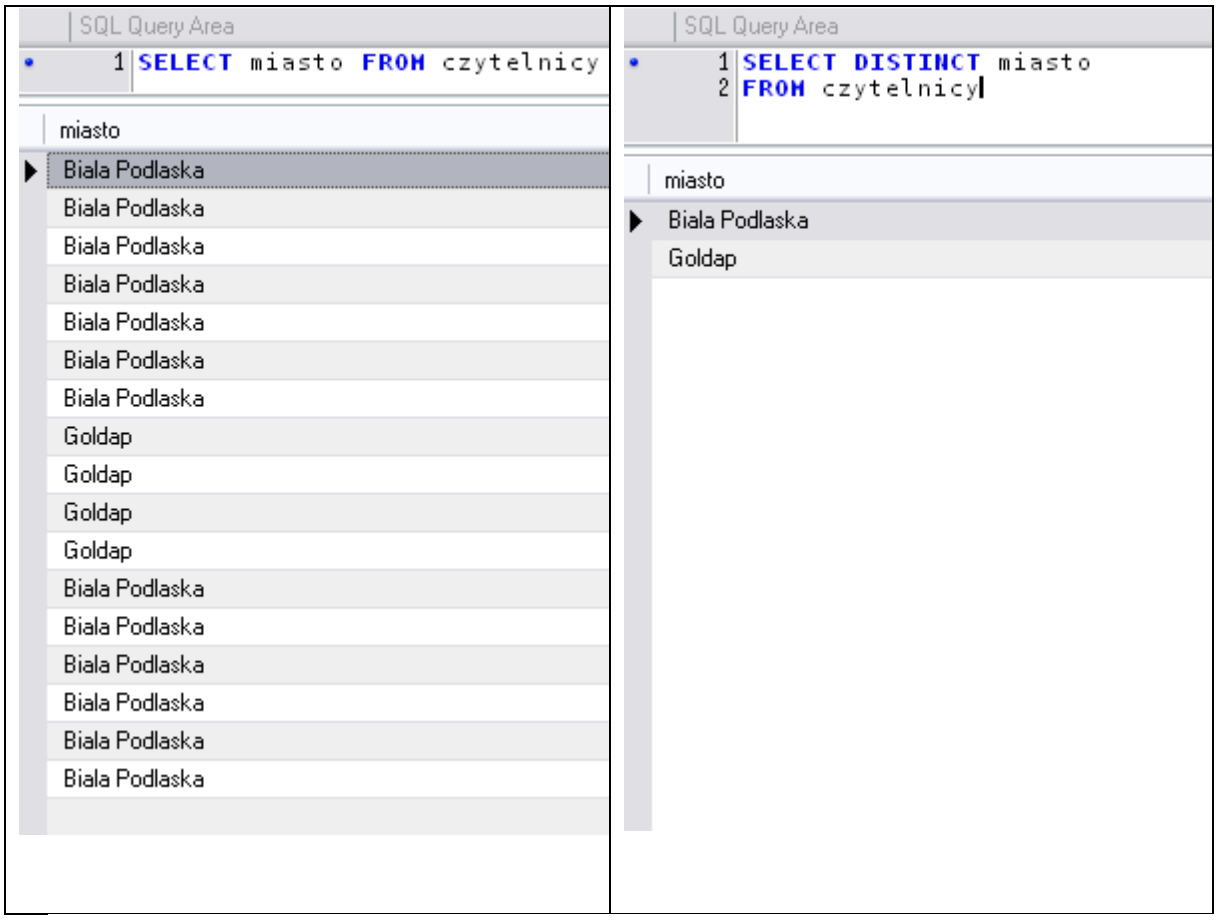
imie	nazwisko	wynagrodzenie
Mateusz	Zielonka	2000.00
Mariusz	Potepa	1700.00
Wojciech	Potepa	1900.00
Anna	Molek	1200.00
Jan	Kowalczuk	1700.00
Antoni	Darecki	2700.00
Krystyna	Czuj	2850.00
Mateusz	Brzeski	2900.00

Przy tworzeniu warunków w klauzuli WHERE można stosować znane z różnych języków programowania operatory binarne porównujące dwie wartości. Są to operatory testujące czy dwie liczby są równe ('='), różne ('!=','<>'), czy jedna z liczb jest większa albo większa lub równa drugiej ('>','>=') oraz czy jedna z liczb jest mniejsza, albo mniejsza lub równa drugiej ('<','<='). Operatory te można stosować na liczbach, łańcuchach (porządek alfabetyczny) oraz datach (data wcześniejsza jest mniejsza).

10. Eliminacja powtórzeń

Gdy zachodzi potrzeba, aby wśród podanych w jednej kolumnie wyników zapytania nie powtarzały się wartości, można użyć słowa kluczowe **DISTINCT**.

W tabeli przedstawiono to samo zapytanie w dwóch wersjach: bez eliminacji powtórzeń i z eliminacja powtórzeń. Zapytanie do otrzymanego wyniku brzmi: z jakich miast pochodzą czytelnicy? i z jakich różnych miast pochodzą czytelnicy?



11. Wyrażenia: BETWEEN...AND, IN, LIKE, IS NULL.

Operator BETWEEN AND sprawdza, czy jedna wartość zawiera się pomiędzy dwoma innymi (włącznie). Składnia operatora wygląda następująco:

x BETWEEN y AND z

gdzie 'x' to sprawdzana wartość, y to wartość określająca początek przedziału a 'z' koniec. Pod 'x', y i 'z' można podstawić dowolne wyrażenia, przy czym wymagane jest aby ‘y’ było mniejsze od ‘z’. Wartości wyrażeń podstawianych pod 'x', y albo 'z' mogą być zarówno liczbami jak i łańcuchami oraz datami. Ważne jest jednak aby wszystkie były tego samego typu. Przeanalizujmy klauzulę **WHERE** pierwszego przykładu:

SQL Query Area

```
1 SELECT CONCAT(imie,' ', nazwisko) OSOBA, wynagrodzenie
2 FROM pracownicy
3 WHERE wynagrodzenie BETWEEN 1500 AND 2700;
```

OSOBA	wynagrodzenie
Jan Kowalczuk	1700.00
Antoni Darecki	2700.00
Wojciech Potepa	1900.00
Radosaw Tomaszewki	2100.00
Emilia Ignatowicz	2000.00
Mariusz Potepa	1700.00
Lukasz Borowik	1900.00
Danuta Zielinska	2000.00
Tomasz Makarski	2000.00
Mateusz Zielonka	2000.00

Oznacza ona, że poszukiwane są wszystkie krotki, w których wartość atrybutu wynagrodzenie mieści się pomiędzy 1500 a 2700.

Operator IN sprawdza czy jedna wartość jest równa przynajmniej jednej z wartości wymienionych na liście. Składnia operatora wygląda następująco:

$$x \text{ IN } (a_1, a_2, a_3, \ldots, a_n),$$

gdzie „x" to sprawdzana wartość, a „a_n to wartości z którymi porównywana jest wartość „x". Pod „x" oraz „a_n można podstawić dowolne wyrażenia. „x" oraz „a_i" mogą być liczbami, łańcuchami oraz datami, ważne jest jednak aby były tego samego typu. Przeanalizujmy klauzulę WHERE drugiego przykładu:

SQL Query Area

```
1 SELECT CONCAT(imie,' ', nazwisko) OSOBA, wynagrodzenie
2 FROM pracownicy
3 WHERE nazwisko IN ('Potepa', 'Borowik');
```

OSOBA	wynagrodzenie
Krzysztof Potepa	9000.00
Wojciech Potepa	1900.00
Mariusz Potepa	1700.00
Lukasz Borowik	1900.00

Oznacza ona, że poszukiwane są wszystkie krotki, w których wartość atrybutu **NAZWISKO** jest równa Potepa albo Borowik .

Operator **LIKE** jest specjalnym operatorem stosowanym do złożonego porównywania łańcuchów. Składnia operatora **LIKE** to:

x LIKE maska,

gdzie x jest dowolnym wyrażeniem typu łańcuchowego, a maska jest specjalnym łańcuchem zawierającym normalne znaki oraz znaki o specjalnym znaczeniu. Znaki specjalne to: '%' i'_', gdzie '%' oznacza dowolny ciąg znaków (zero lub więcej) a '_' dokładnie jeden, dowolny, znak. Operator **LIKE** sprawdza, czy zadany łańcuch spełnia warunki zdefiniowane przez maskę.

SQL Query Area

```
1 SELECT CONCAT(imie,' ', nazwisko) OSOBA, wynagrodzenie
2 FROM pracownicy
3 WHERE nazwisko LIKE 'Pote%'
```

OSOBA	wynagrodzenie
Krzysztof Potepa	9000.00
Wojciech Potepa	1900.00
Mariusz Potepa	1700.00

Klauzula WHERE w tym zapytaniu zawiera warunek: „nazwisko LIKE 'Pote%'". W warunku tym maska ma postać 'Pote%'. Reprezentuje ona wszystkie łańcuchy, które jako pierwsze znaki mają litery Pote, po których może się znajdować dowolny ciąg znaków.

Warunek ten będzie zatem spełniony, jeśli łańcuch zapisany w atrybucie NAZWISKO będzie się zaczynał od liter 'Pote'. Poniżej przedstawiono kilka innych przykładów maski: '%SKI' - wszystkie łańcuchy kończące się na SKI (np. 'KOTARSKI'). 'MAlinowsk_ - wszystkie łańcuchy zaczynające się od Malinowsk i mające dowolny jeden znak na końcu (np. 'MALINOWSKI', 'MALINOWSKA').

Operator **IS NULL** sprawdza, czy wartość danego wyrażenia jest równa **NULL**. W przykładowym zapytaniu sprawdza on, czy wartość atrybutu wynagrodzenie wynosi **NULL**. Należy w tym miejscu zwrócić szczególną uwagę na to, iż sprawdzanie, czy dany atrybut ma wartość **NULL** za pomocą operatora '=' nie ma sensu. Wynika to z faktu, że **NULL** stanowi wartość nieznaną. Wynik porównania wartości nieznanej z jakąś inną wartością jest również nieznany. Różnica w semantyce pomiędzy dwoma przykładowymi warunkami: **wynagrodzenie = NULL** i **wynagrodzenie IS NULL** jest zatem znaczna.

SQL Query Area

```
1 SELECT Nr_transakcji, data_wypozyczenia
2 FROM wypozyczenia
3 WHERE data_zwrotu IS NULL
```

Nr_transakcji	data_wypozycz...
1	2009-12-05
2	2009-10-09
3	2009-05-01
4	2006-12-12
5	2009-01-09
8	2009-10-14
16	2009-12-05
17	2009-10-09
18	2009-05-01
20	2009-01-09
23	2009-11-14
27	2008-11-05
28	2009-12-22
31	2009-11-05
32	2009-11-09
33	2009-05-01
35	2009-02-09
38	2009-10-14
46	2009-12-05
47	0009-10-09
48	2009-05-01
50	2009-01-09

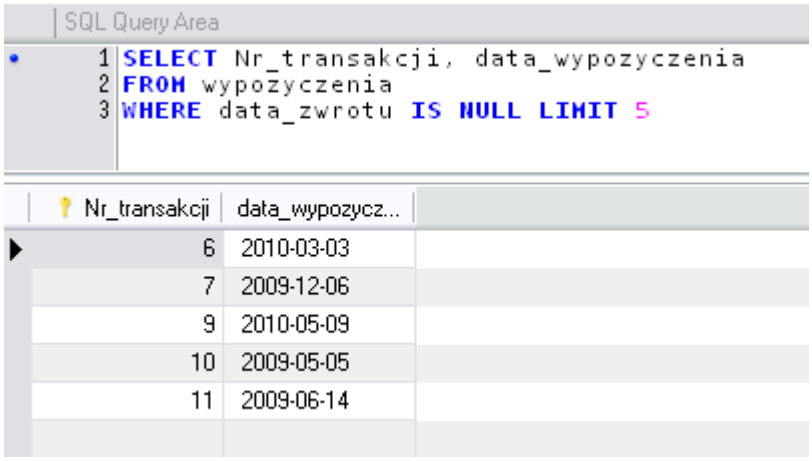
SQL Query Area

```
1 SELECT Nr_transakcji, data_wypozyczenia
2 FROM wypozyczenia
3 WHERE data_zwrotu = NULL
```

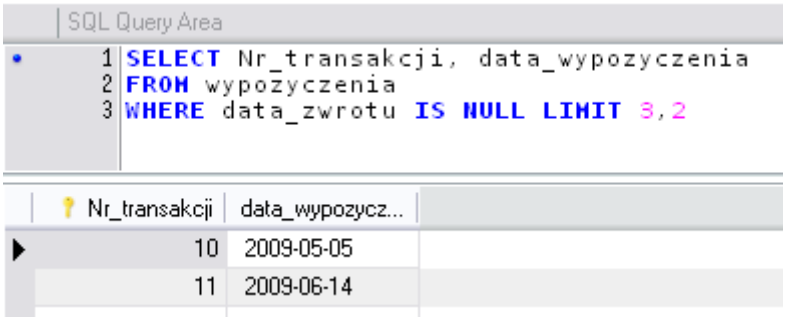
12. Porcjowanie wyników zapytania

Jeżeli nie ma potrzeby uzyskać wszystkich rekordów pasujących do zapytania, a na przykład jedynie pierwsze 10, możesz posłużyć się klauzulą LIMIT .

Aby otrzymać określoną liczbę rekordów, należy posłużyć się klauzulą LIMIT, po której podaje się odpowiednią wartość.



Jeśli w LIMIT poda się dwa argumenty (oddzielone przecinkiem), pierwszy z nich zostanie potraktowany jako przesunięcie od początku wyniku (tyle pierwszych wierszy ma zostać pominiętych), a drugi -jako liczba wierszy do wyświetlenia.



„Trzy pierwsze wiersze mają być pominięte, Wyświetlone mają zostać dwa kolejne wiersze"

Ćwiczenie do samodzielnej realizacji:

1. Wyświetl wszystkie dane z tabeli książki
2. Wyświetl nazwy książek i ich autorów wg wzoru:

Dane ksiazek i autorow
"Fotografowanie aparatem cyfrowym - samouczek" zostala napisana przez autora: Bogdan Krzymowski
"Strategia bkitnego oceanu" zostala napisana przez autora: Rene Mauborgne
"Metro 2033" zostala napisana przez autora: Dmitry Glukhovsky
"Pieko Pacyfiku" zostala napisana przez autora: Eugene B. Sledge
"Prawo pracy" zostala napisana przez autora: Ludwik Florek
"Uwarunkowania i plany rozwoju turystyki" zostala napisana przez autora: Zygmunt Mlynarczyk
"Ekspresja receptorow sterydowych" zostala napisana przez autora: Maciej Skrzypczak
"Adobe Flash i PHP" zostala napisana przez autora: Matthew Keefe
"Pancerni korsarze Kriegsmarine" zostala napisana przez autora: Rafal Kaczmarek
"ksiazka BJORK inrock" zostala napisana przez autora: Wieslaw Halinski
"Pan_Tadeusz" zostala napisana przez autora: Adam Mickiewicz
"Programowanie" zostala napisana przez autora: Roman Matejek
"Grafika" zostala napisana przez autora: Grzegorz Malas
"Programowanie" zostala napisana przez autora: Roman Matejek
"100 najwiekszych osiagniec medycyny" zostala napisana przez autora: Straus W. Eugene
"Atlas Ptakow" zostala napisana przez autora: Frank Hecker
"Geografia Fizyczna Polski" zostala napisana przez autora: Andrzej Richling
"Podstawy ekonomii" zostala napisana przez autora: Eugeniusz Kwiatkowski

3. Wyświetl duzymi literami imie i nazwisko pracowników biblioteki
4. Wyświetl inicjał imienia z kropką i nazwisko autorów książek oraz tytuły książek w jednej kolumnie z nagłówkiem ‘Ksiazki’
5. Wyświetl dane pracowników: nazwisko i imię (nadaj alias: osoba) oraz datę zatrudnienia podaną wg wzoru: dzień, nazwa miesiąca, rok (nadaj alias ‘data zatrudnienia’

osoba	data zatrudnienia
Kowalczuk Jan	5 May 2007
Czuj Krystyna	2 April 2006
Brzeski Mateusz	7 May 2006
Darecki Antoni	28 May 2007
Molek Anna	11 September 2008
Potepa Krzysztof	24 July 2000
Potepa Wojciech	2 August 2008
Tomaszewki Radosaw	2 December 2005
Ignatowicz Emilia	24 July 2000
Potepa Mariusz	13 February 2006
Borowik Lukasz	14 March 1999
Malinowski Dariusz	24 May 2004
Zielinska Danuta	6 May 1999
Makarski Tomasz	7 July 2000
Zielonka Mateusz	14 March 2009

6. Wyświetl imie i nazwisko pracowników pisane małymi literami od końca np. **atram akydohc**
7. Wyświetl w jednej kolumnie nazwy działów oraz liczbę liter ile posiadają te nazwy wg wzoru:
   kierownik – 9 liter
8. Wyświetl wyniki równań:
   a. Zaokrąglony do dołu wynik równania: 33-2,45
   b. Zaokrąglony do góry wynik równania: 123,23 + 34,4*3
9. Używając funkcji matematycznych wyświetlić wynik równania:
   $2+\sqrt{|3{,}75-33|*2^3}\,/(44/11-12{,}5)$ (WYNIK= 0,200346054)
10. * Wyświetl nr_transakcji oraz liczbę dni ile były wypożyczone poszczególne książki.
11. * Wyświetl nazwy stanowisk wg opisu: Pierwsza , środkowa i ostatnia litera duże, reszta małe.
12. Dla każdego pracownika wyświetl jego imię i nazwisko oraz jego roczną płacę. Wyrażeniu obliczającemu roczną płace nadaj alias „DOCHÓD". Dane posortuj rosnąco wg dochodu.
13. Wyświetl w kolejności alfabetycznej (wg nazwiska i imienia) raport zawierający dane o pracownikach wg wzoru:
    „Krzysztof Potepa jest zatrudniony od daty 2000-07-24 i jego dochód miesięczny wynosi 9000 zł".
14. Wyświetl te nazwiska czytelników z bazy, których długość przekracza 6 znaków. Uporządkuj wynik od najdłuższego nazwiska do najkrótszego i w kolejności alfabetycznej.
15. Wyświetl czytelników urodzonych między 1970 a 1990 rokiem. Wynik wyświetl uporządkowany malejąco wg daty urodzenia.
16. Wyświetl nazwisko, imię i miasto z jakiego pochodzą czytelnicy, którzy są zapisani w bazie jako aktualni (nie maja podanej daty skreślenia)
17. Wyświetl 5 pierwszych wypożyczeń dokonanych w miesiącu maju 2009 roku
18. Wyświetl numery transakcji wypożyczeń, które odbyły się jako 20, 21 i 22
19. Wyświetl nazwiska czytelników, które alfabetycznie mieszczą się w przedziale od Da … Ko
20. Wyświetl w jednej kolumnie imiona i nazwiska wszystkich czytelniczek z tabeli czytelnicy urodzone po roku 1970.
21. Wyświetl w jednej kolumnie nazwiska czytelników, którzy w nazwisku maja literę „K". Wyniki wyświetl alfabetycznie wg nazwiska. Usuń powtórzenia.
22. Jaka data będzie za 100 dni ? A jak była 200 dni temu?