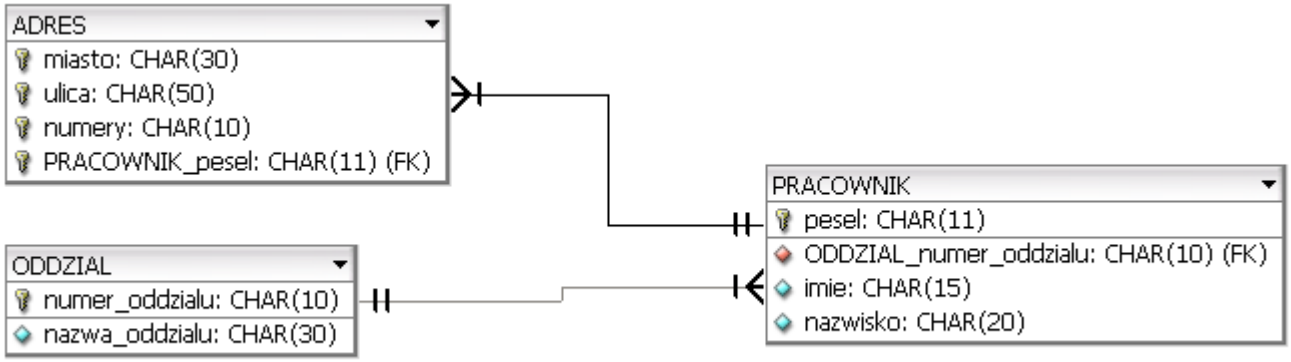
SQL – Podstawy języka III: powtórzenie

Krzysztof Regulski
WIMiIP, KISiM,
regulski@agh.edu.pl
B5, pok. 408

Modyfikacja schematu relacji

– Utwórz tabelę wg schematu:




```
CREATE TABLE ODDZIAL (
numer_oddzialu CHAR(10) NOT NULL,
nazwa_oddzialu CHAR(30) NULL,
PRIMARY KEY(numer_oddzialu)
);

CREATE TABLE ADRES (
miasto CHAR(30) NOT NULL,
ulica CHAR(50) NOT NULL,
numery CHAR(10) NOT NULL,
PRACOWNIK_pesel CHAR(11) NOT NULL,
PRIMARY KEY(miasto, ulica, numery, PRACOWNIK_pesel),
);
```

```
CREATE TABLE PRACOWNIK (
pesel CHAR(11) NOT NULL,
ODDZIAL_numer_oddzialu CHAR(10) NOT NULL,
imie CHAR(15) NULL,
nazwisko CHAR(20) NULL,
PRIMARY KEY(pesel),
);
```

– Jakie tabele znajdują się w bazie?

```
SHOW TABLES;
```

– Jakie kolumny znajdują się w tabeli PRACOWNIK?

```
SHOW COLUMNS FROM PRACOWNIK;
```

lub

```
DESCRIBE PRACOWNIK;
```

– Usuń tabelę PRACOWNIK

```
DROP TABLE PRACOWNIK;
```

– Usuń i dodaj ponownie klucz podstawowy:

```
ALTER TABLE PRACOWNIK DROP PRIMARY KEY;
ALTER TABLE PRACOWNIK ADD PRIMARY KEY (pesel);
```

– Dodaj kolumnę *placa* do tabeli pracownicy:

```
ALTER TABLE PRACOWNIK
ADD placa DOUBLE NULL DEFAULT 0
AFTER nazwisko;
```

Modyfikacja danych

- Modyfikacja bazy danych przez ich dodanie może być zrealizowana w SQL za pomocą poleceń postaci:

```
INSERT INTO <tabela> VALUES <wiersz>
INSERT INTO <tabela(schemat)> VALUES <wiersz>
```

- Można zatem dodawać jedynie całe krotki oraz jednokrotnie tylko do jednej relacji (tabeli). Zachowany być musi schemat relacji i domeny atrybutów.
- O ile to jest dopuszczone <krotka> może zawierać wartości *NULL*.

Dodaj wiersz do tabeli *PRACOWNICY* z wynagrodzeniem ustawionym na *NULL*

```
INSERT INTO PRACOWNIK
VALUES('99999900000','WIMiIP-KISiM','Regulski','Krzysztof',NULL)
```

lub

```
INSERT INTO pracownicy (pesel, ODDZIAL_numer_oddzialu, nazwisko, placa)
VALUES ('99999900000','WIMiIP-KISiM','Regulski', NULL)
```

– Uzupełnij tabele danymi:

id_oddzialu	nazwa_oddzialu	adres
L140	Betatrex	Słoneczna 1/2, Kraków
A4	Alfatron	Miodowa 12, Ozimek
B340	Tetrix	Puchatka 123A/2, Kalisz

pesel	imie	nazwisko	id_oddzialu	adres
75102406713	Okowita	Ambrozjowa	L140	Brożka 24/56, Kraków
54032204567	Fableusz	Kosonosy	L140	Czarnowiejska 36/12, Kraków
56123099087	Atanazy	Angonilewicz	A4	Mocarna 3, Ozimek
67051757834	Kosmateusz	Buler	A4	Plac Mały 56, Mrozki

```
INSERT
INTO PRACOWNIK (pesel, imie, nazwisko, id_oddzialu, adres)
VALUES (…)
lub
INSERT INTO PRACOWNIK VALUES (…)
```

– Popraw miejscowość w adresie Kosmateusza Bulera na ‘Ozimek’:

```
UPDATE PRACOWNIK
SET adres = ”Plac Maly 56, Ozimek”
WHERE imie = ”Kosmateusz”
AND nazwisko = ”Buler”;
```

– Podnieś wynagrodzenie Regulskiemu o 30%

```
UPDATE PRACOWNIK
SET placa=placa*1.3
WHERE nazwisko LIKE ‘Regulski’;
```

– Zmień w bazie danych id_oddzialu dla oddziału Betatrex na B1 (pamiętaj o numerach u pracowników!)

```
UPDATE ODDZIAL
SET id_oddzialu='B1'
WHERE id_oddzialu='Betatrex';

UPDATE PRACOWNIK
SET id_oddzialu='B1'
WHERE id_oddzialu='Betatrex';
```

– żeby nie popełnić błędu w przyszłości, można założyć więzy integralności w postaci klucza obcego:

```
ALTER TABLE PRACOWNIK ADD FOREIGN KEY (id_oddzialu)
REFERENCES ODDZIAL(id_oddzialu)
ON DELETE CASCADE ON UPDATE CASCADE;
```

- Usuń pracownika o peselu 56123099087

```
DELETE FROM PRACOWNIK
WHERE pesel='56123099087';
```

- Usuń wszystkie dane z tabeli PRACOWNIK
(uważaj, żeby nie usunąć danych z innych tabel!)

```
DELETE FROM PRACOWNIK;
```

- Usuń tabelę PRACOWNIK

```
DROP PRACOWNIK;
```

SELECT

– Pokaż zawartość tabeli PRACOWNIK

```
SELECT * FROM PRACOWNIK;
```

– SQL dopuszcza duplikaty zarówno w relacjach jak i rezultatach zapytań.

– Dla wymuszenia eliminacji duplikatów wstawia się słowo kluczowe **DISTINCT** po **SELECT**.

Przykład: znajdź imiona wszystkich pracowników i usuń duplikaty

```
SELECT DISTINCT imie
FROM pracownik;
```

SELECT

- Klauzula **SELECT** może zawierać wyrażenia arytmetyczne z operatorami **+, -, *, /** operujące na stałych i atrybutach krotek

- Zapytanie:

```
SELECT nazwisko, imie, placa + 100
FROM pracownik;
```

zwróci relację, w której atrybut `placa` będzie zwiększony o 100.

SELECT

– Polecenia **SELECT** można używać również nie odwołując się do żadnej tabeli w celu obliczania wyrażeń lub pracy na ciągach znaków lub zmiennych:

```
SELECT 1+1;

SELECT ‘ten napis pojawi sie na ekranie’;

SELECT ‘slowa’, ‘w’, ‘osobnych’, ‘kolumnach’;

SELECT @liczba1:=8 AS A, @liczba2:=2 AS B,
@wynik:=@liczba1+@liczba2 AS 'WYNIK A+B';
```

```
+---+---+-----------+
| A | B | WYNIK A+B |
+---+---+-----------+
| 8 | 2 |        10 |
+---+---+-----------+
```

Klauzula WHERE

- Klauzula **WHERE** składa się z warunków dotyczących atrybutów relacji z klauzuli **FROM.** Umożliwia wyświetlanie wierszy, których kolumny spełniają określony warunek. Pola objęte klauzurą **WHERE** nie muszą być na liście wyboru.

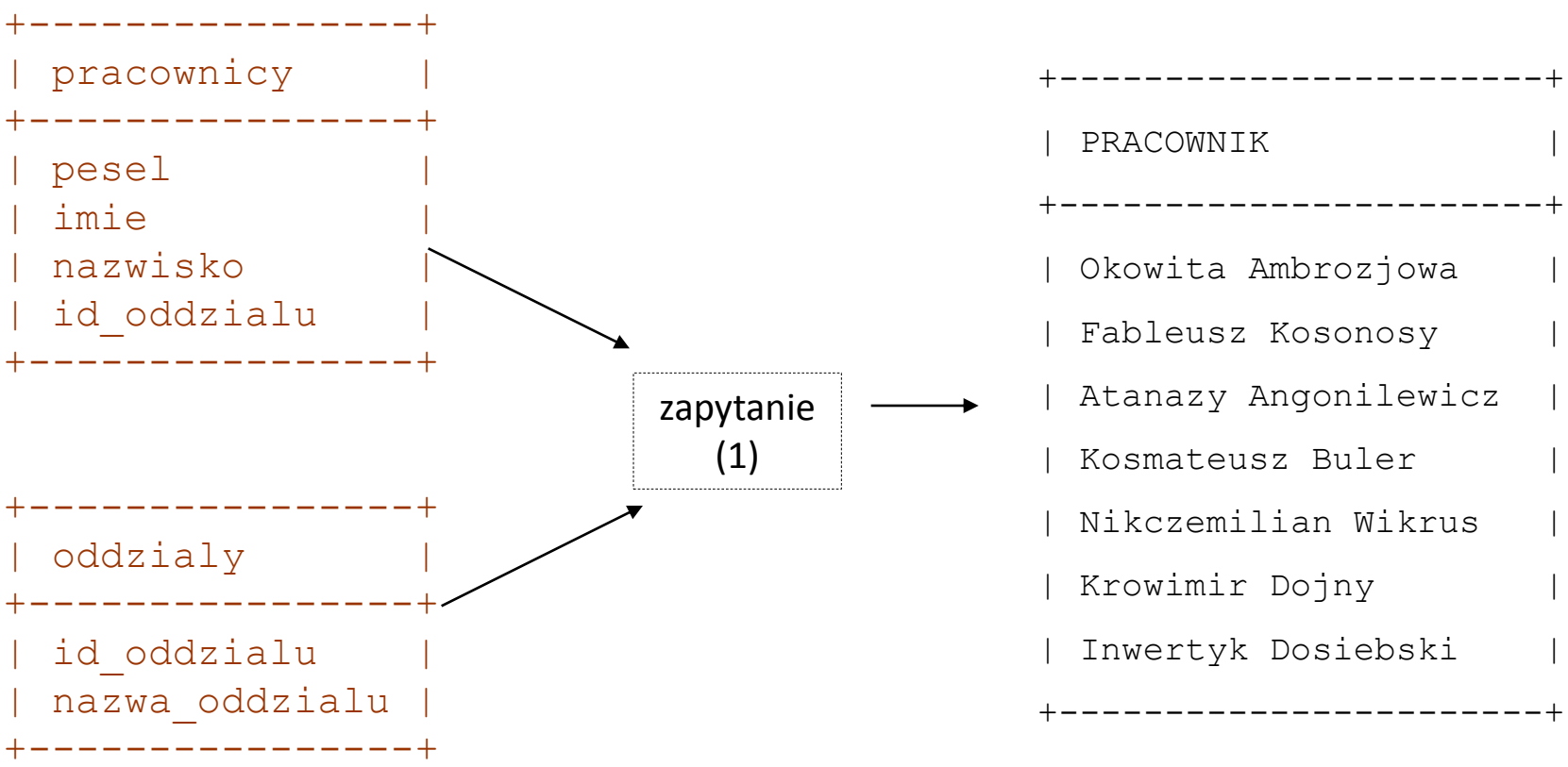
```
SELECT nazwisko, imie
FROM pracownicy
WHERE placa > 100;
```

- umożliwia łączenie tabel według rożnych kryteriów.

zapytanie (1)

```
SELECT CONCAT (pracownicy.imie, ' ',
pracownicy.nazwisko)
AS PRACOWNIK
FROM pracownicy, oddzialy
WHERE
pracownicy.id_oddzialu=oddzialy.id_oddzialu
AND oddzialy.nazwa_oddzialu LIKE 'Betatrex';
```

Klauzula WHERE



Operatory porównań

- `WHERE placa BETWEEN 'dolna_granica' AND 'górna_granica'`
- `WHERE imie NOT IN ('Stefan', 'Bożena')`
- `(NOT) LIKE` kiedy porównujemy ciągi znaków do wzorca

  Wyrażenia regularne:

%	- dowolny ciąg znaków
_	- dowolny znak
[ck]	- znak 'c' lub 'k'
[c-k]	- znak z zakresu od 'c' do 'k'
[^c]	- nie 'c'

- `IS (NOT) NULL` służy do sprawdzania, czy wartość w polu to NULL

– Znajdź pracowników, którzy obchodzą dziś imieniny (dziś są imieniny Stefana, Bogumiła, Krystyny i Bożeny):

```
SELECT imie, nazwisko
FROM pracownik
WHERE imie IN ('Stefan','Bogumił','Krystyna','Bożena');
```

– Znajdź pracowników, którzy zarabiają pomiędzy 1500 a 2000 zł (być może są skłonni wziąć nadgodziny?)

```
SELECT imie, nazwisko
FROM pracownik
WHERE placa BETWEEN 1500 AND 2000;
```

– Znajdź pracowników, których nazwisko zaczyna się na literę R:

```
SELECT imie, nazwisko
FROM pracownik
WHERE nazwisko LIKE 'R%';
```

– Zaginął portfel szefa, sekretarka pamięta, że ostatnio w pokoju był pracownik, który załatwiał jakiś podpis, ale nie pamięta dokładnie jak się przedstawił… Na nazwisko miał jakoś… ‘Mot…’ albo ‘Lot…’ ale na pewno nie ‘Kot…’. Wyszukaj pracowników pasujących do opisu:

```
SELECT imie, nazwisko
FROM pracownik
WHERE nazwisko LIKE '[ML]ot%';
```

lub

```
SELECT imie, nazwisko
FROM pracownik
WHERE nazwisko LIKE '[^K]ot%'
```

Rodzaje złączeń

- **złączenie wewnętrzne (`INNER JOIN`)** – w relacji wynikowej występują wyłącznie te krotki, które spełniają warunek złączenia
- **złączenie lewostronne zewnętrzne (`LEFT OUTER JOIN`)** – zawiera wszystkie krotki ***R*** uzupełnione krotkami ***S*** spełniającymi warunek
- **złączenie prawostronne zewnętrzne (`RIGHT OUTER JOIN`)** - zawiera wszystkie krotki ***S*** uzupełnione krotkami ***R*** spełniającymi warunek
- **złączenie zewnętrzne pełne (`FULL OUTER JOIN`)** – zawiera wszystkie krotki ***R*** oraz ***S*** uzupełnione wartościami typu **`NULL`** gdy do danej krotki nie pasuje żadna krotka z drugiej relacji

Przykład – dane:

– Relacja **oddzial**:

id_oddzialu	nazwa_oddzialu
L140	Betatrex
A4	Alfatron
B340	Tetrix

– Relacja **pracownik**:

pesel	imie	nazwisko	id_oddzialu
75102406713	Okowita	Ambrozjowa	L140
54032204567	Fableusz	Kosonosy	L140
56123099087	Atanazy	Angonilewicz	A4

Przykład:

```
SELECT oddzialy.nazwa_oddzialu, pracownicy.nazwisko
FROM (oddzialy INNER JOIN pracownicy
ON oddzialy.id_oddzialu=pracownicy.id_oddzialu);
```

```
+-----------------+-----------------+
| nazwa_oddzialu  | nazwisko        |
+-----------------+-----------------+
| Betatrex        | Ambrozjowa      |
| Betatrex        | Kosonosy        |
| Alfatron        | Angonilewicz    |
+-----------------+-----------------+
```

```
SELECT oddzialy.nazwa_oddzialu, pracownicy.nazwisko
FROM (oddzialy LEFT OUTER JOIN pracownicy
ON oddzialy.id_oddzialu=pracownicy.id_oddzialu);
```

```
+-----------------+-----------------+
| nazwa_oddzialu  | nazwisko        |
+-----------------+-----------------+
| Betatrex        | Ambrozjowa      |
| Betatrex        | Kosonosy        |
| Alfatron        | Angonilewicz    |
| Tetrix          | NULL            |
+-----------------+-----------------+
```

Aliasy

– Możliwe jest używanie aliasów nazw kolumn i nazw tabel. Umożliwiają one:
  - » zmianę nazwy kolumny wyświetlanej
  - » nadanie nazwy kolumnie będącej wynikiem wyrażenia lub stałą

```
SELECT
@liczba1:=8 AS A,
@liczba2:=2 AS B,
@wynik:=@liczba1+@liczba2 AS 'WYNIK A+B';
```

```
+---+---+-----------+
| A | B | WYNIK A+B |
+---+---+-----------+
| 8 | 2 |        10 |
+---+---+-----------+
```

Sortowanie wyników

- Sortowanie wyników osiąga się dzięki klauzuli **ORDER BY**. Sortowanie odbywa się kolejno według wartości atrybutów wymienionych w klauzuli.
- Dla każdego z atrybutów można podać specyfikator **DESC** dla porządku malejącego.
- Pokaż imiona i nazwiska wszystkich pracowników, posortuj wyniki malejąco.

```
SELECT nazwisko, imie
FROM pracownicy
ORDER BY nazwisko DESC, imie DESC;
```

- Zobacz jak wygląda tabela pracownik:

```
DESCRIBE pracownik;
```

- Wykonaj zapytanie o imiona i nazwiska wszystkich pracowników:

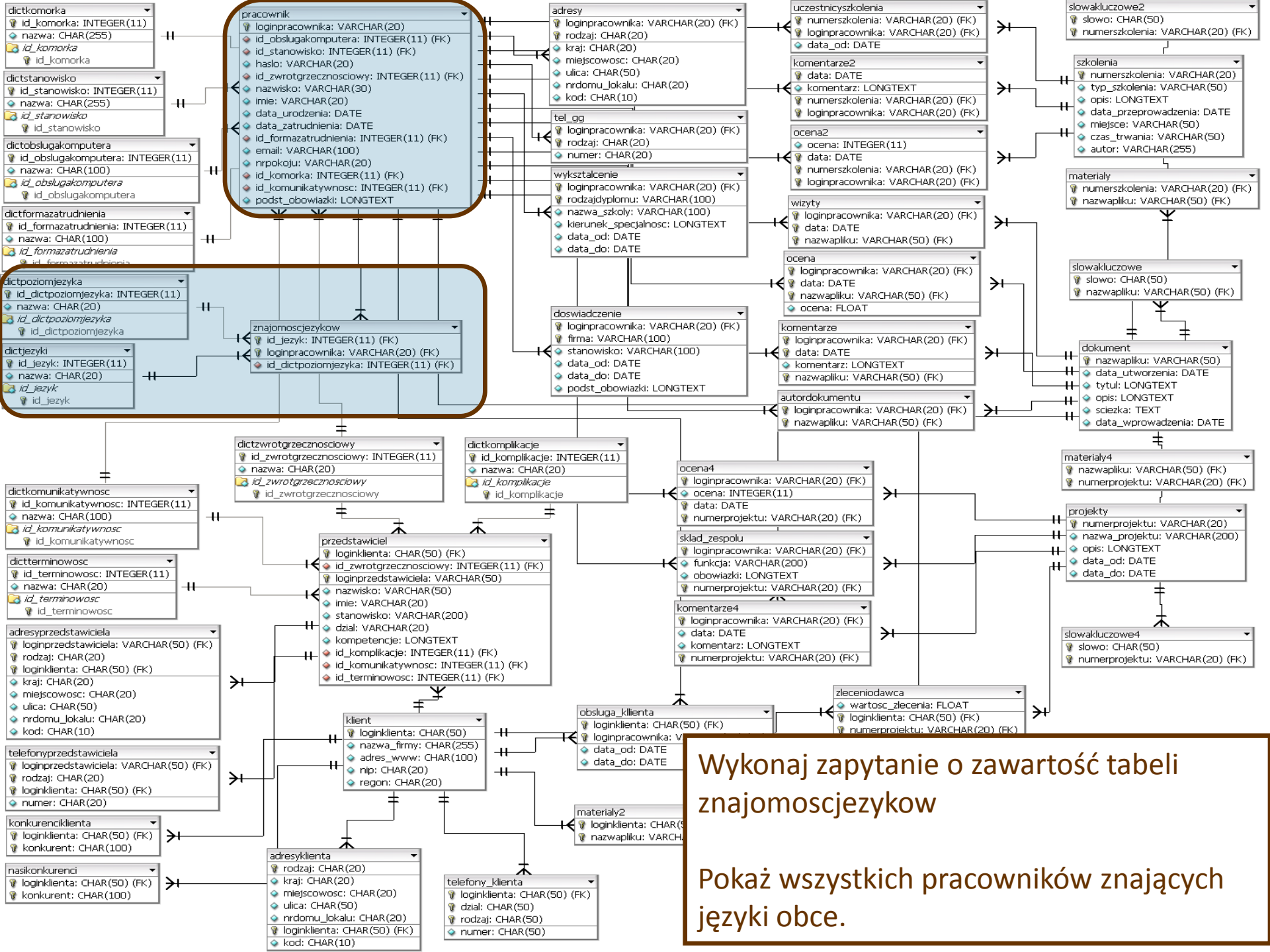
```
SELECT imie, nazwisko FROM pracownik;
```

- Pokaż listę imion i nazwisk pracowników, oraz numery porządkowe w pierwszej kolumnie:

```
SELECT @lp:=0;

SELECT @lp:=@lp+1

AS 'Lp.', imie, nazwisko FROM pracownik;
```



Wykonaj zapytanie o zawartość tabeli znajomoscjezykow

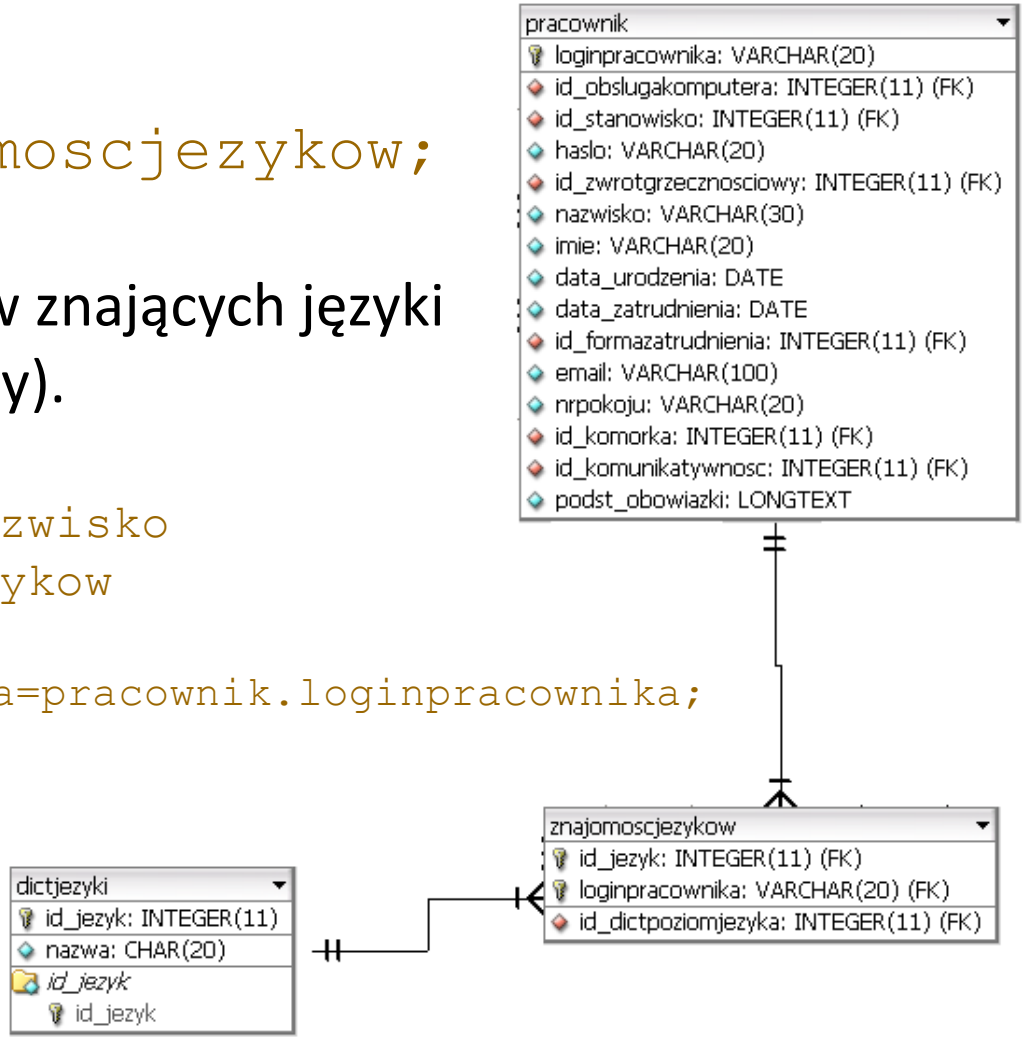
Pokaż wszystkich pracowników znających języki obce.

– Wykonaj zapytanie o zawartość tabeli *znajomoscjezykow*

```
SELECT * FROM znajomoscjezykow;
```

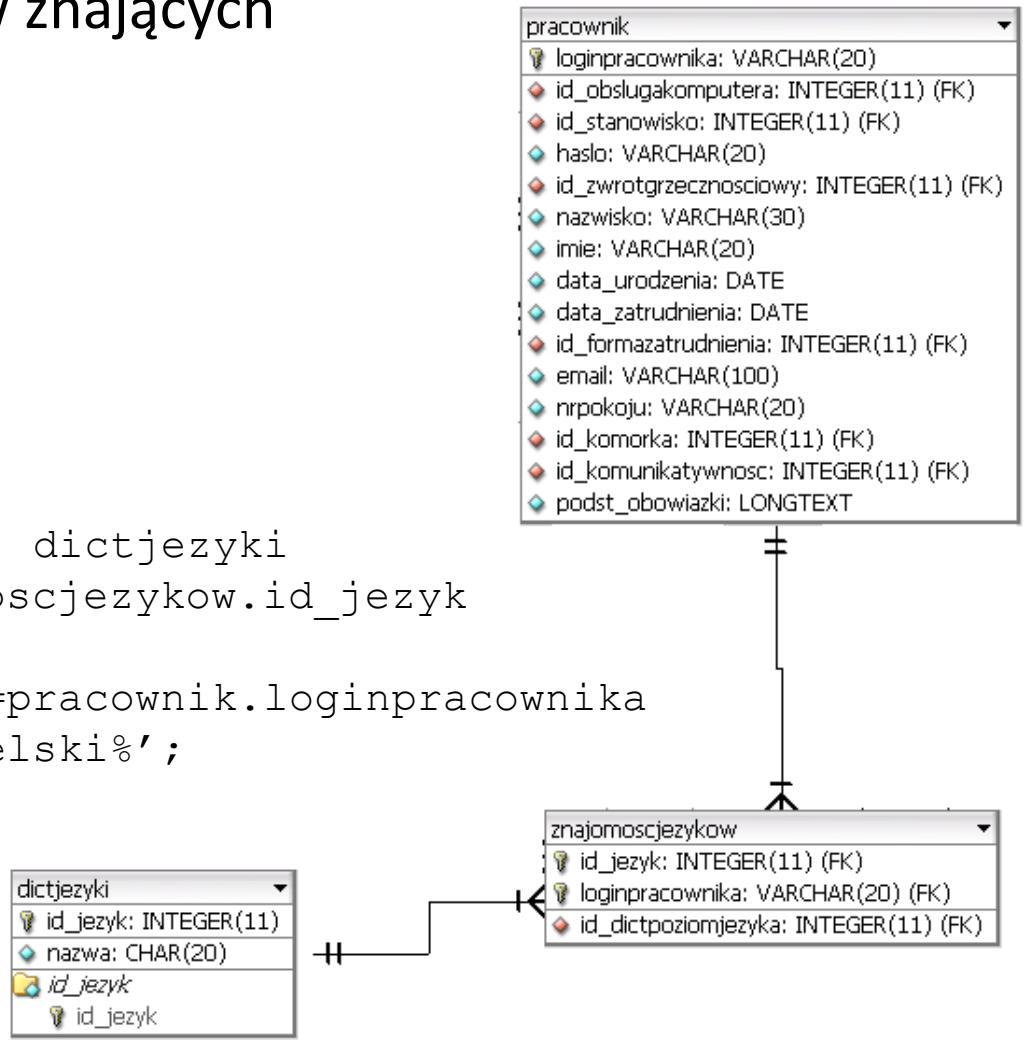
– Pokaż wszystkich pracowników znających języki obce (eliminując duplikaty).

```
SELECT DISTINCT pracownik.nazwisko
FROM pracownik, znajomoscjezykow
WHERE
znajomoscjezykow.loginpracownika=pracownik.loginpracownika;
```



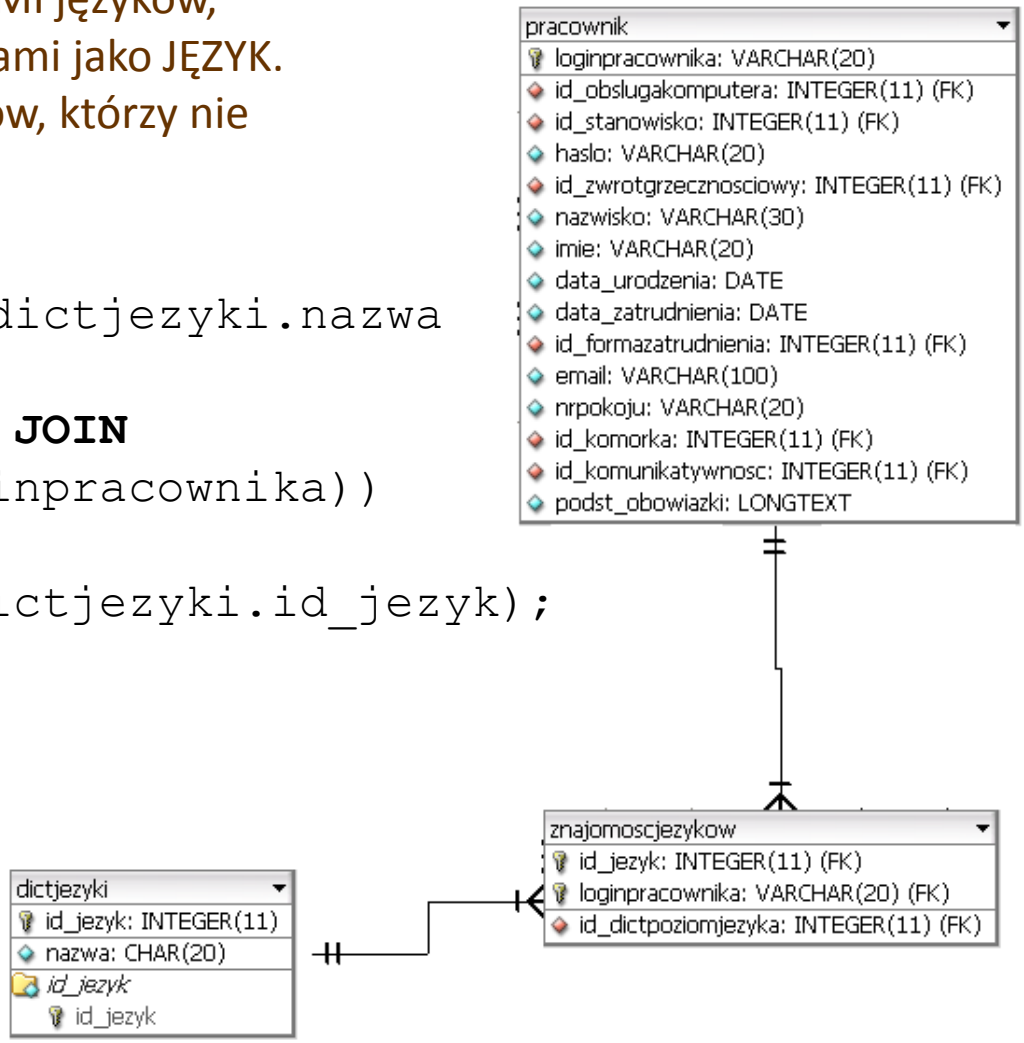
– Znajdź wszystkich pracowników znających język angielski:

```
select pracownik.nazwisko
from pracownik, znajomoscjezykow, dictjezyki
where dictjezyki.id_jezyk=znajomoscjezykow.id_jezyk
and
znajomoscjezykow.loginpracownika=pracownik.loginpracownika
and dictjezyki.nazwa LIKE '%angielski%';
```



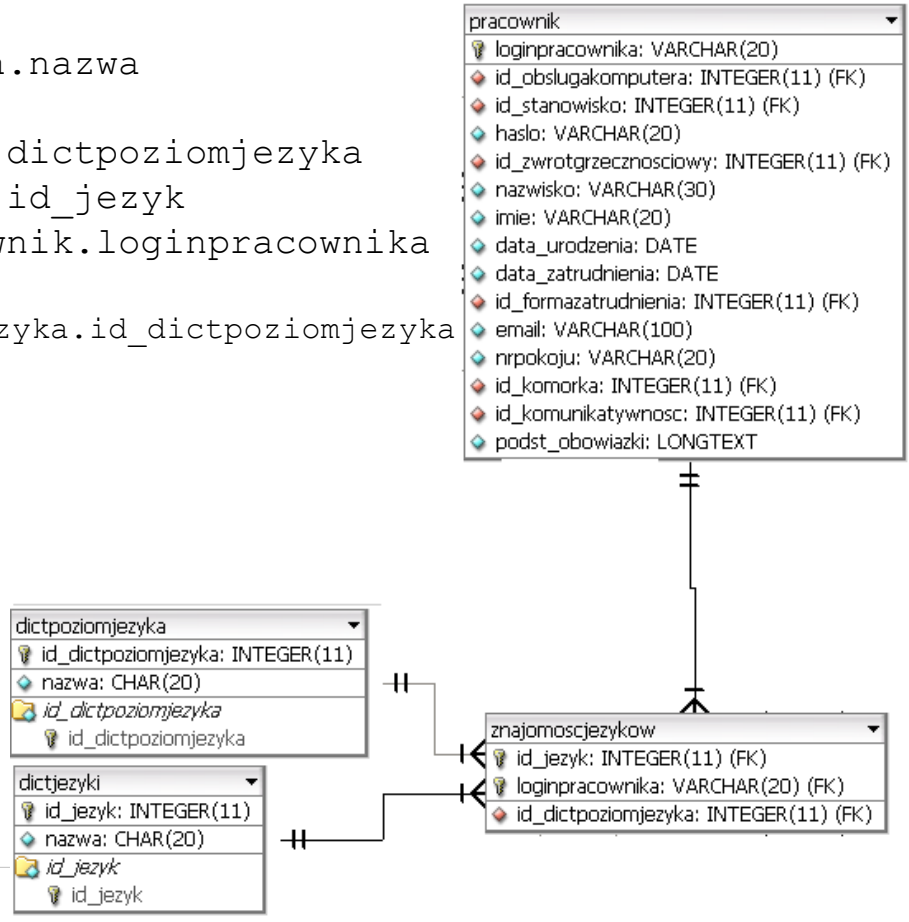
Pokaż listę pracowników wraz z NAZWAMI języków, które znają, oznaczając kolumnę z językami jako JĘZYK. Na liście pokaż również tych pracowników, którzy nie znają obcych języków.

```
SELECT pracownik.nazwisko, dictjezyki.nazwa
AS JĘZYK
FROM ((pracownik LEFT OUTER JOIN
znajomoscjezykow USING (loginpracownika))
INNER JOIN dictjezyki ON
znajomoscjezykow.id_jezyk=dictjezyki.id_jezyk);
```



Znajdź wszystkich pracowników znających język angielski i pokaż stopień jego znajomości w kolumnie POZIOM ANGIELSKIEGO. Wyniki poukładaj alfabetycznie (wg nazwisk).

```
SELECT pracownik.nazwisko, dictpoziomjezyka.nazwa
AS 'POZIOM ANGIELSKIEGO'
FROM pracownik,znajomoscjezykow,dictjezyki,dictpoziomjezyka
WHERE dictjezyki.id_jezyk=znajomoscjezykow.id_jezyk
AND znajomoscjezykow.loginpracownika=pracownik.loginpracownika
AND
znajomoscjezykow.id_dictpoziomjezyka=dictpoziomjezyka.id_dictpoziomjezyka
AND znajomoscjezykow.id_jezyk=1
ORDER BY pracownik.nazwisko;
```



Funkcje agregujące

– **Funkcje agregujące** SQL operują na zbiorach wartości (np. kolumna relacji) i obliczają pojedynczą wartość. Są to:

AVG - wartość średnia,
MIN - wartość minimalna,
MAX - wartość maksymalna,
SUM – suma,
COUNT – liczność zbioru.

» Znajdź ilość krotek w relacji *pracownicy*.

```
SELECT COUNT (*)
FROM pracownicy;
```

» Znajdź średnią płacę .

```
SELECT AVG (placa)
FROM  pracownicy;
```

Funkcje agregujące

- Funkcje agregujące mogą być zastosowane do grup krotek. Uzyskuje się to przez zastosowanie klauzuli **GROUP BY** i odpowiednie uformowanie klauzuli **SELECT**.
- Znajdź ilość pracowników w każdym oddziale firmy.

```
SELECT nazwa_oddzialu, COUNT (DISTINCT nazwisko)
FROM pracownicy, oddzialy
WHERE pracownicy.id_oddzialu=oddzialy.id_oddzialu
GROUP BY nazwa_oddzialu;
```

Uwaga: atrybuty z klauzuli **SELECT** poza funkcją agregującą muszą wystąpić w liście **GROUP BY**.

Funkcje agregujące

– Funkcje agregujące mogą być użyte do nakładania **warunków na grupy krotek**. Wówczas stosuje się rozwinięcie klauzuli **GROUP BY** o postaci: **HAVING** funkcja agregująca.

» Znajdź nazwy wszystkich oddziałów, gdzie średnia płaca jest większa niż 1,200zł

```
SELECT nazwa_oddzialu, AVG (placa)
FROM pracownicy, oddzialy
WHERE pracownicy.id_oddzialu=oddzialy.id_oddzialu
GROUP BY nazwa_oddzialu
HAVING AVG (placa) > 1200;
```

Uwaga: warunki z klauzuli **HAVING** są stosowane po uformowaniu grup.

Znajdź średnią ocenę plików zawierających w nazwie słowo „opis” i umieść ją w kolumnie „Średnia ocena”.

```
select nazwapliku,avg(ocena) as 'Średnia ocena'
from ocena
group by nazwapliku
having nazwapliku like '%opis%';
```