

# Structured Query Language 2

30.05.2021

# AGENDA

1. SELECT
2. SELECT – Składnia
3. Klauzule DISTINCT oraz ORDER BY
4. Operatory algebry relacji
5. Funkcje pomocnicze

# Czym jest klauzula SELECT?

**SELECT – Czyli definiowanie wyniku**

# Czym jest klauzula SELECT?

**SELECT – Czyli definiowanie wyniku**

**Klauzula SELECT, musi znaleźć się w każdym zapytaniu**

# Czym jest klauzula SELECT?

SELECT – Czyli definiowanie wyniku

Klauzula SELECT, musi znaleźć się w każdym zapytaniu

Mimo, iż występuje jako pierwsza, jej treść przetwarzana jest jako jedna z ostatnich

# Czym jest klauzula SELECT?

**SELECT – Czyli definiowanie wyniku**

**Klauzula SELECT, musi znaleźć się w każdym zapytaniu**

**Mimo, iż występuje jako pierwsza, jej treść przetwarzana jest jako jedna z ostatnich**

|     |          |
|-----|----------|
| (5) | SELECT   |
| (1) | FROM     |
| (2) | WHERE    |
| (3) | GROUP BY |
| (4) | HAVING   |
| (6) | ORDER BY |

# Czym jest klauzula SELECT?

**SELECT – Czyli definiowanie wyniku**

**Klauzula SELECT, musi znaleźć się w każdym zapytaniu**

**Mimo, iż występuje jako pierwsza, jej treść przetwarzana jest jako jedna z ostatnich**

(5) SELECT  
(1) FROM  
(2) WHERE  
(3) GROUP BY  
(4) HAVING  
(6) ORDER BY

SELECT służy do wybierania wierszy i kolumn z jednej lub kilku tabel. Może być używany jako osobna instrukcja lub jako zapytanie lub podzapytanie w innych poleceniach.

# SELECT - Składnia

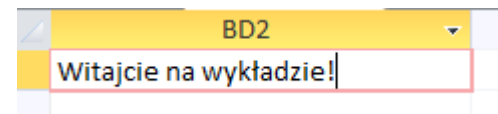
- SELECT
- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY



# SELECT (1)

**Brak danych w tabeli? To nie problem 😊**

SELECT 'Witajcie na wykładzie!' as BD2



| BD2                    |
|------------------------|
| Witajcie na wykładzie! |

# SELECT (2)

Brak danych w tabeli? To nie problem 😊

```
SELECT 'Witajcie na wykładzie!' as BD2
```

| BD2                    |
|------------------------|
| Witajcie na wykładzie! |

```
SELECT 'Witajcie na wykładzie!' AS Kolumna1, 'Z BD2' AS Kolumna2;
```

| Kolumna1               | Kolumna2 |
|------------------------|----------|
| Witajcie na wykładzie! | Z BD2    |

# Tabela pomocnicza

| Kontakty |         |            |              |                        |        |          |
|----------|---------|------------|--------------|------------------------|--------|----------|
| ID       | Imie    | Nazwisko   | Firma        | Stanowisko             | Pensja | Miasto   |
| 1        | Mikołaj | Nowaczyk   | Google       | Stażysta               |        | Gdańsk   |
| 2        | Juliusz | Miler      | Google       | Junior Data Specialist | 4000   | Gdańsk   |
| 3        | Paweł   | Sikora     | Testowa      | Tester                 | 4500   | Wrocław  |
| 4        | Jan     | Kowalski   | Testowa      | CEO                    | 10000  | Wrocław  |
| 5        | Adam    | Nowak      | Urząd Miasta | Urzędnik               | 3000   | Szczecin |
| 6        | Adam    | Nowakowski | Urząd Miasta | Urzędnik               | 3100   | Szczecin |
| 7        | Amir    | Miller     | Google       | Informatyk             | 6000   | Gdynia   |
| 8        | Zenon   | Malinowski | Google       | Księgowy               | 1200   | Gdańsk   |
| 9        | Ewelina | Kujawa     | Google       | Księgowy               | 1400   | Gdańsk   |

# FROM (1)

SELECT – definiowanie wyniku

FROM – określanie źródła

**SELECT** lista\_kolumn **FROM** wyrażenie\_tabelowe

| SELECT *<br>FROM Kontakty; |         |            |              |                        |        |          |
|----------------------------|---------|------------|--------------|------------------------|--------|----------|
| ID                         | Imie    | Nazwisko   | Firma        | Stanowisko             | Pensja | Miasto   |
| 1                          | Mikołaj | Nowaczyk   | Google       | Stażysta               |        | Gdańsk   |
| 2                          | Juliusz | Miler      | Google       | Junior Data Specialist | 4000   | Gdańsk   |
| 3                          | Paweł   | Sikora     | Testowa      | Tester                 | 4500   | Wrocław  |
| 4                          | Jan     | Kowalski   | Testowa      | CEO                    | 10000  | Wrocław  |
| 5                          | Adam    | Nowak      | Urząd Miasta | Urzędnik               | 3000   | Szczecin |
| 6                          | Adam    | Nowakowski | Urząd Miasta | Urzędnik               | 3100   | Szczecin |
| 7                          | Amir    | Miller     | Google       | Informatyk             | 6000   | Gdynia   |
| 8                          | Zenon   | Malinowski | Google       | Księgowy               | 1200   | Gdańsk   |
| 9                          | Ewelina | Kujawa     | Google       | Księgowa               | 1400   | Gdańsk   |

# FROM (2)

SELECT – definiowanie wyniku

FROM – określanie źródła

| SELECT Imie, Nazwisko, Stanowisko<br>FROM Kontakty; |            |                        |
|---|------------|------------------------|
| Imie  | Nazwisko   | Stanowisko             |
| Mikołaj   | Nowaczyk   | Stażysta               |
| Juliusz   | Miler      | Junior Data Specialist |
| Paweł   | Sikora     | Tester                 |
| Jan   | Kowalski   | CEO                    |
| Adam  | Nowak      | Urzędnik               |
| Adam  | Nowakowski | Urzędnik               |
| Amir  | Miller     | Informatyk             |
| Zenon   | Malinowski | Księgowy               |
| Ewelina   | Kujawa     | Księgowa               |

# FROM (3)

- FROM określa źródłowe *tabele* wykorzystywane w *zapytaniu*.
- Jeśli wymienimy więcej niż jedną tabelę to nastąpi operacja ich złączenia.
- Klauzula FROM jest obowiązkowa.
- Istnieje kilka rodzajów złączeń.

# TOP (1)

SELECT – definiowanie wyniku

FROM – określanie źródła

TOP – ogranicza

```
SELECT TOP n [PERCENT] FROM wyrażenie_tabelowe [ORDER BY ...]
```

```
SELECT TOP 5  
ID, Imie, Nazwisko  
FROM Kontakty;
```

| ID | Imie    | Nazwisko |
|----|---------|----------|
| 1  | Mikołaj | Nowaczyk |
| 2  | Juliusz | Miler    |
| 3  | Paweł   | Sikora   |
| 4  | Jan     | Kowalski |
| 5  | Adam    | Nowak    |

# TOP (2)

- W Ms Access TOP jest używany wraz z instrukcją SELECT.
- Powoduje, że zwracane są *rekordy* zawierające się w podanym zakresie liczbowym lub procentowym.



# WHERE (1)

SELECT – definiowanie wyniku

FROM – określanie źródła

WHERE – filtrowanie rekordów

**SELECT** lista\_kolumn **FROM** wyrażenie\_tabelowe **WHERE** predykat

```
SELECT *  
FROM Kontakty  
WHERE Stanowisko = 'Księgowy'
```

| ID | Imie    | Nazwisko   | Firma  | Stanowisko | Pensja | Miasto |
|----|---------|------------|--------|------------|--------|--------|
| 8  | Zenon   | Malinowski | Google | Księgowy   | 1200   | Gdańsk |
| 9  | Ewelina | Kujawa     | Google | Księgowy   | 1400   | Gdańsk |

# WHERE (2)

SELECT – definiowanie wyniku

FROM – określanie źródła

WHERE – filtrowanie rekordów

```
SELECT *  
FROM Kontakty  
WHERE (Firma = 'Google' OR Firma = 'Testowa') AND Pensja >=1350 AND Miasto <> 'Gdynia';
```

| ID | Imie    | Nazwisko | Firma   | Stanowisko             | Pensja | Miasto  |
|----|---------|----------|---------|------------------------|--------|---------|
| 2  | Juliusz | Miler    | Google  | Junior Data Specialist | 4000   | Gdańsk  |
| 3  | Paweł   | Sikora   | Testowa | Tester                 | 4500   | Wrocław |
| 4  | Jan     | Kowalski | Testowa | CEO                    | 10000  | Wrocław |
| 9  | Ewelina | Kujawa   | Google  | Księgowy               | 1400   | Gdańsk  |

# WHERE (3)

- WHERE określa te rekordy z tabel wymienionych w klauzuli FROM, które spełniają podane kryteria.
- W przypadku braku klauzuli WHERE zwracane są wszystkie wiersze tabeli.
- Klauzula ta nie jest obowiązkowa, jednak jeśli występuje, musi być umieszczona po klauzuli FROM.

# GROUP BY (1)

SELECT – definiowanie wyniku

FROM – określanie źródła

GROUP BY – grupowanie zapytania

**SELECT** lista\_kolumn **FROM** wyrażenie\_tabelowe **GROUP BY** pola\_grupowania

**SELECT Stanowisko, COUNT(\*) as Zliczenie**  
**FROM Kontakty**  
**GROUP BY Stanowisko;**

| Stanowisko             | Zliczenie |
|------------------------|-----------|
| CEO                    | 1         |
| Informatyk             | 1         |
| Junior Data Specialist | 1         |
| Księgowy               | 2         |
| Stażysta               | 1         |
| Tester                 | 2         |
| Urzędnik               | 2         |

# GROUP BY (2)

- GROUP BY scala rekordy o tych samych wartościach wskazanych na liście kolumn przy instrukcji SELECT.
- Klauzula ta nie jest obowiązkowa.
- Domyślna kolejność sortowania jest rosnąca (od A do Z, od 0 do 9).
- Bardzo często łączy się ją z funkcjami agregacji oraz z klauzulą HAVING, która pozwala przefiltrować rekordy po ich zgrupowaniu.

# ORDER BY (1)

SELECT – definiowanie wyniku

FROM – określanie źródła

WHERE – filtrowanie rekordów

ORDER BY – sortowanie wyników

**SELECT** lista\_pól **FROM** wyrażenie\_tabelowe **ORDER BY** kolumna [ASC | DESC] [,kolumna2] [ASC | DESC] [,...]

```
SELECT *  
FROM Kontakty  
WHERE (Firma = 'Google' OR Firma = 'Testowa') AND Miasto <> 'Gdynia'  
ORDER BY Pensja DESC;
```

| ID | Imie    | Nazwisko   | Firma   | Stanowisko             | Pensja | Miasto  |
|----|---------|------------|---------|------------------------|--------|---------|
| 4  | Jan     | Kowalski   | Testowa | CEO                    | 10000  | Wrocław |
| 3  | Paweł   | Sikora     | Testowa | Tester                 | 4500   | Wrocław |
| 2  | Juliusz | Miler      | Google  | Junior Data Specialist | 4000   | Gdańsk  |
| 9  | Ewelina | Kujawa     | Google  | Księgowy               | 1400   | Gdańsk  |
| 8  | Zenon   | Malinowski | Google  | Księgowy               | 1200   | Gdańsk  |
| 1  | Mikołaj | Nowaczyk   | Google  | Stażysta               |        | Gdańsk  |

# ORDER BY (2)

SELECT – definiowanie wyniku

FROM – określanie źródła

WHERE – filtrowanie rekordów

ORDER BY – sortowanie wyników

```
SELECT *  
FROM Kontakty  
WHERE (Firma = 'Google' OR Firma = 'Testowa') AND Miasto <> 'Gdynia' AND Pensja IS NOT NULL  
ORDER BY Pensja ASC;
```

| ID | Imie    | Nazwisko   | Firma   | Stanowisko             | Pensja | Miasto  |
|----|---------|------------|---------|------------------------|--------|---------|
| 8  | Zenon   | Malinowski | Google  | Księgowy               | 1200   | Gdańsk  |
| 9  | Ewelina | Kujawa     | Google  | Księgowy               | 1400   | Gdańsk  |
| 2  | Juliusz | Miler      | Google  | Junior Data Specialist | 4000   | Gdańsk  |
| 3  | Paweł   | Sikora     | Testowa | Tester                 | 4500   | Wrocław |
| 4  | Jan     | Kowalski   | Testowa | CEO                    | 10000  | Wrocław |

# ORDER BY (3)

- Klauzula ORDER BY w programie rekordy stanowiące wynik zapytania według określonego pola lub pól w kolejności rosnącej lub malejącej.
- Klauzula ORDER BY jest opcjonalna. Jednak aby wyświetlać dane w posortowanej kolejności, musisz użyć klauzuli ORDER BY.
- Domyślna kolejność sortowania jest rosnąca (od A do Z, od 0 do 9).
- by sortować w kolejności malejącej (od Z do A, od 9 do 0), dodaj słowo zastrzeżone DESC po nazwie każdego pola, które chcesz posortować w kolejności malejącej.



# HAVING (1)

SELECT – definiowanie wyniku

FROM – określanie źródła

WHERE – filtrowanie rekordów

GROUP BY – grupowanie zapytania

HAVING – ogranicza wyniki

**SELECT** lista\_pól **FROM** wyrażenie\_tabelowe **WHERE** kryteria **GROUP BY** pola\_grupowania **HAVING** kryteria\_grupowania

```
Select Miasto, COUNT(ID) as CustQty
From Kontakty
Where Miasto <> 'Gdynia'
GROUP BY Miasto
HAVING COUNT(ID)>1;
```

| Miasto   | CustQty |
|----------|---------|
| Gdańsk   | 4       |
| Szczecin | 2       |
| Wrocław  | 2       |

# HAVING (2)

- HAVING pozwala określić, które ze zgrupowanych rekordów mają być wyświetlone.
- Po zgrupowaniu rekordów klauzulą GROUP BY pokazywane są te rekordy, które spełniają kryteria klauzuli HAVING.

# DISTINCT (1)

SELECT – definiowanie wyniku

DISTINCT – usuwa duplikaty

FROM – określanie źródła

```
SELECT DISTINCT lista_pól FROM wyrażenie_tabelowe
```

```
SELECT DISTINCT Miasto  
FROM Kontakty;
```

| Miasto   |
|----------|
| Gdańsk   |
| Gdynia   |
| Szczecin |
| Wrocław  |

# DISTINCT (2)

SELECT – definiowanie wyniku

DISTINCT – usuwa duplikaty

FROM – określanie źródła

```
SELECT DISTINCT lista_pól FROM wyrażenie_tabelowe ORDER BY ...
```

```
SELECT DISTINCT Miasto  
FROM Kontakty  
ORDER BY Miasto DESC;
```

| Miasto   |
|----------|
| Wrocław  |
| Szczecin |
| Gdynia   |
| Gdańsk   |

# DISTINCT (3)

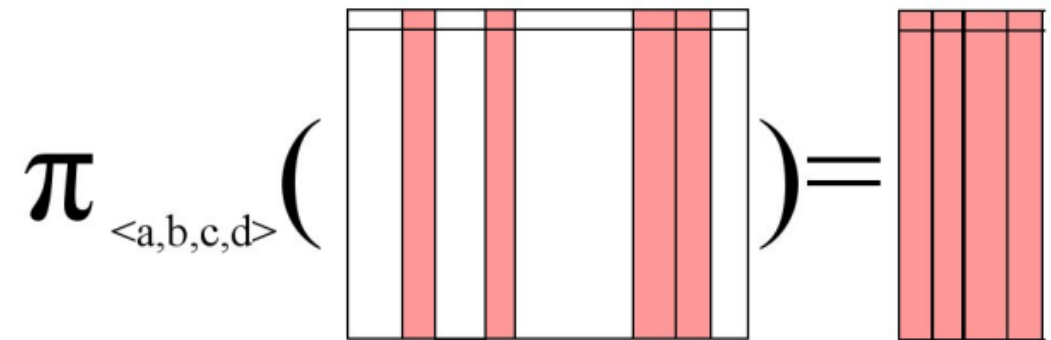
- DISTINCT jest używany wraz z instrukcją SELECT.
- Określa, że *wiersze* powtarzające się powinny zostać usunięte przed zwróceniem ich na zewnątrz.
- Dwa wiersze traktuje się jako równe jeśli wszystkie wartości dla każdej z *kolumn* zwracanych rozkazem SELECT są sobie równe.

# Operatory Algebry Relacji

- Rzut (projekcja),

# PROJEKCJA (1)

- Umożliwia pobieranie wartości wybranych atrybutów, wymienionych po słowie kluczowym SELECT z wszystkich krotek relacji.
- Operacja ta jest nazwana także podzbiorem pionowym i/lub Rzutem;
- Zwraca wiersze dla których warunek złączenia jest spełniony;



# PROJEKCJA (1)

| SELECT Imie, Nazwisko, Pensja FROM kontakty; |            |        |
|--|------------|--------|
| Imie   | Nazwisko   | Pensja |
| Mikołaj                                      | Nowaczyk   |        |
| Juliusz                                      | Miler      | 4000   |
| Paweł  | Sikora     | 4500   |
| Jan  | Kowalski   | 10000  |
| Adam   | Nowak      | 3000   |
| Adam   | Nowakowski | 3100   |
| Amir   | Miller     | 6000   |
| Zenon  | Malinowski | 1200   |
| Ewelina                                      | Kujawa     | 1400   |

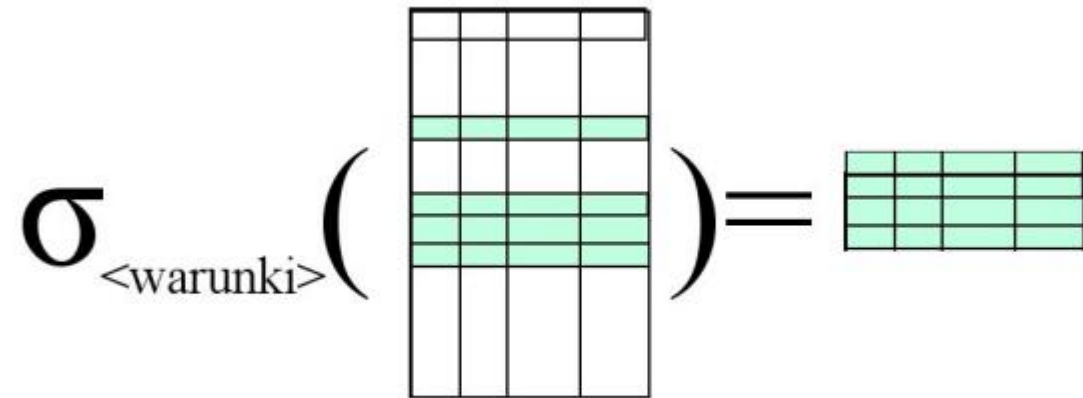


# Operatory Algebry Relacji

- Rzut (projekcja),
- Restrykacja (selekcja),

# Restrykcja (1)

- Daje w wyniku relację składającą się ze wszystkich krotek, które spełniają określone warunki.
- Operacja ta jest nazwana Selekcją;
- Operacja ta nazwana jest również podzbiorem poziomym;



# Restrykcja (2)

```
SELECT *  
FROM Kontakty  
Where Stanowisko<>'CEO';
```

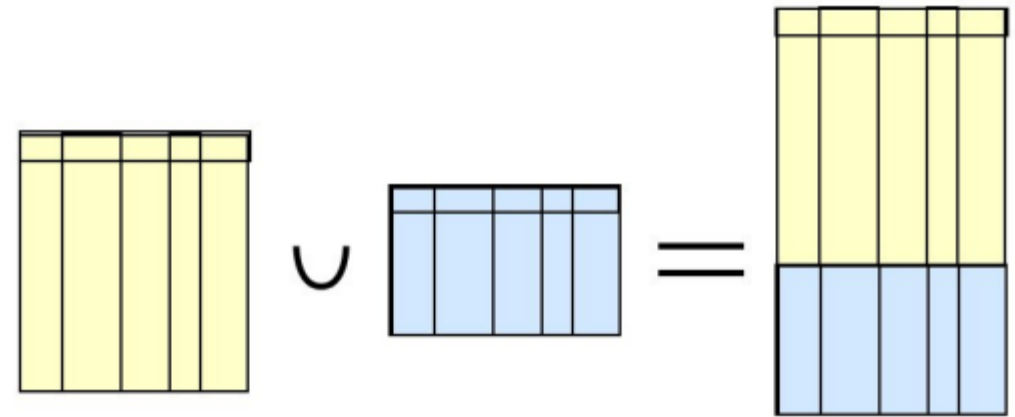
| ID | Imie    | Nazwisko   | Firma        | Stanowisko             | Pensja | Miasto   |
|----|---------|------------|--------------|------------------------|--------|----------|
| 1  | Mikołaj | Nowaczyk   | Google       | Stażysta               |        | Gdańsk   |
| 2  | Juliusz | Miler      | Google       | Junior Data Specialist | 4000   | Gdańsk   |
| 3  | Paweł   | Sikora     | Testowa      | Tester                 | 4500   | Wrocław  |
| 5  | Adam    | Nowak      | Urząd Miasta | Urzędnik               | 3000   | Szczecin |
| 6  | Adam    | Nowakowski | Urząd Miasta | Urzędnik               | 3100   | Szczecin |
| 7  | Amir    | Miller     | Google       | Informatyk             | 6000   | Gdynia   |
| 8  | Zenon   | Malinowski | Google       | Księgowy               | 1200   | Gdańsk   |
| 9  | Ewelina | Kujawa     | Google       | Księgowy               | 1400   | Gdańsk   |

# Operatory Algebry Relacji

- Rzut (projekcja),
- Restrykacja (selekcja),
- Suma,

# SUMA (1)

- Suma dwóch relacji zgodnych typów R i S jest relacją zawierającą wszystkie krotki relacji R i S;
- Suma jest operacją komutatywną  $R \cup S = S \cup R$ ;



# SUMA (2)

```
SELECT Stanowisko FROM Kontakty
WHERE Miasto='Gdańsk'
UNION
SELECT Stanowisko FROM Kontakty
WHERE Miasto='Szczecin'
UNION
SELECT Stanowisko FROM Kontakty
WHERE Miasto='Wrocław';
```

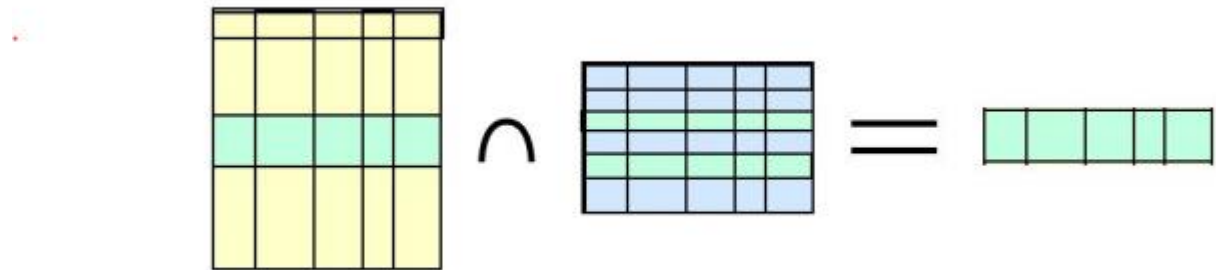
| Stanowisko             |
|------------------------|
| CEO                    |
| Junior Data Specialist |
| Księgowy               |
| Stażysta               |
| Tester                 |
| Urzędnik               |

# Operatory Algebry Relacji

- Rzut (projekcja),
- Restrykcja (selekcja),
- Suma,
- Przecięcie,

# Przecięcie (1)

- Przecięcie pozwala znaleźć iloczyn dwóch lub więcej zbiorów krotek tzn. takich, które występują zarówno w jednej jak i w drugiej relacji.
- Przecięcie jest operacją komutatywną  $R \cap S = S \cap R$ ;
- Warunkiem poprawności tej operacji jest zgodność liczby i typów atrybutów relacji bazowych;





# Operatory Algebry Relacji

| Contacts |         |              |         |             |        |           |
|----------|---------|--------------|---------|-------------|--------|-----------|
| ID       | Imie    | Nazwisko     | Firma   | Stanowisko  | Pensja | Miasto    |
| 1        | Adam    | Niemczyk     | Krzak   | Stażysta    |        | Warszawa  |
| 2        | Jan     | Walesa       | Krzak   | Programista | 4000   | Warszawa  |
| 3        | Michał  | Walesiak     | Krzak   | Tester      | 4500   | Warszawa  |
| 4        | Paweł   | Rychter      | Intel   | Programista | 7500   | Kołobrzeg |
| 5        | Juliusz | Nowak        | Intel   | Informatyk  | 3650   | Kołobrzeg |
| 6        | Jan     | Nowak        | Intel   | Stażysta    | 1440   | Koszalin  |
| 7        | Adam    | Mada         | Fujitsu | Informatyk  | 2555   | Koszalin  |
| 8        | Zenon   | Truskawa     | Toshiba | Księgowy    | 2550   | Koszalin  |
| 9        | Lech    | Wielkopolski | Fujitsu | Tester      | 2500   | Gdynia    |
| 10       | Jerzy   | Dudek        | Fujitsu | Stażysta    | 1480   | Gdynia    |

# Przecięcie (2)

```
SELECT Stanowisko  
FROM Kontakty  
INTERSECT  
SELECT Stanowisko  
FROM Contacts;
```

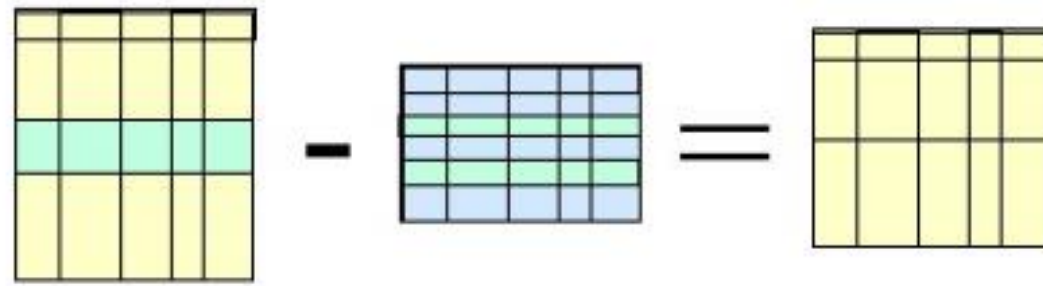
| Stanowisko |
|------------|
| Informatyk |
| Księgowy   |
| Stażysta   |
| Tester     |

# Operatory Algebry Relacji

- Rzut (projekcja),
- Restrykcja (selekcja),
- Suma,
- Przecięcie,
- Różnica,

# Różnica (1)

- Operacja obliczania różnicy dwóch relacji polega na znalezieniu wszystkich krotek, które występują w pierwszej relacji, ale nie występują w drugiej.
- Różnica nie jest operacją komutatywną  $R - S \neq S - R$ .



# Różnica (2)

```
SELECT Stanowisko FROM Contacts  
WHERE Firma='Intel'  
MINUS  
SELECT Stanowisko FROM Contacts  
WHERE Firma='Krzak';
```

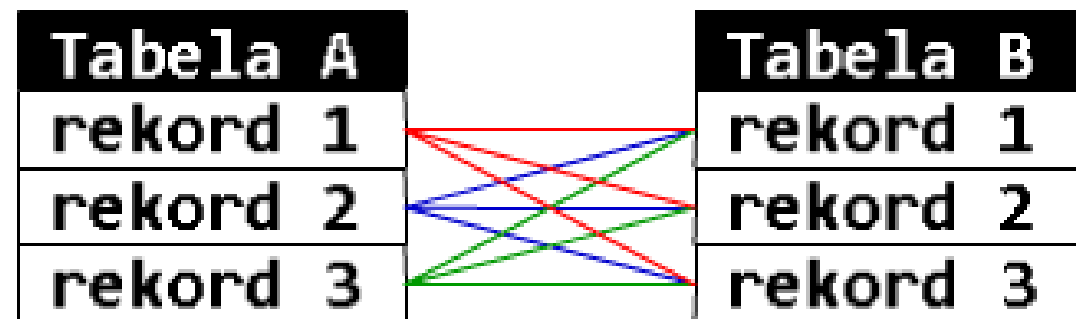
| Stanowisko |
|------------|
| Informatyk |
| Stażysta   |

# Operatory Algebry Relacji

- Rzut (projekcja),
- Restrykacja (selekcja),
- Suma,
- Przecięcie,
- Różnica,
- Iloczyn kartezjański,

# Iloczyn Kartezjański (1)

- CROSS JOIN służy do łączenia tabel przez utworzenie par wszystkich rekordów z jednej tabeli i z drugiej;
- Każdy rekord z jednej tabeli stworzy parę ze wszystkimi rekordami z drugiej tabeli.
- W przypadku większej ilości relacji, operacja ta jest wykonywana na pierwszych dwóch, a następnie na otrzymanym wyniku i relacji następnej, aż do wyczerpania wszystkich argumentów.



# Iloczyn Kartezjański (2)

| IDRozmiaru | Rozmiar |
|------------|---------|
| 1          | Mały    |
| 2          | Średni  |
| 3          | Duży    |

| IDKoloru | Kolor     |
|----------|-----------|
| 1        | Czerwony  |
| 2        | Niebieski |
| 3        | Zielony   |
| 4        | Żółty     |



# Iloczyn Kartezjański (3)

| Rozmiar | Kolor     |
|---------|-----------|
| Mały    | Czerwony  |
| Mały    | Niebieski |
| Mały    | Zielony   |
| Mały    | Żółty     |
| Średni  | Czerwony  |
| Średni  | Niebieski |
| Średni  | Zielony   |
| Średni  | Żółty     |
| Duży    | Czerwony  |
| Duży    | Niebieski |
| Duży    | Zielony   |
| Duży    | Żółty     |

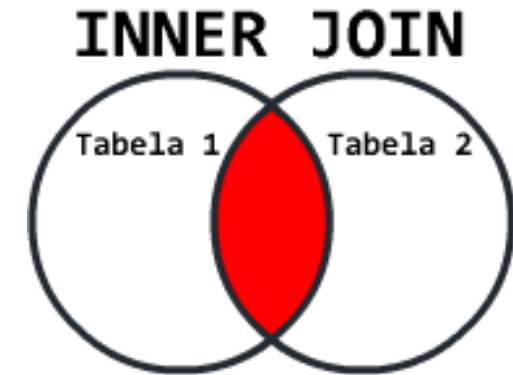
```
SELECT Rozmiar,Kolor  
FROM Rozmiary  
CROSS JOIN Kolory;
```

# Operatory Algebry Relacji

- Rzut (projekcja),
- Restrykcja (selekcja),
- Suma,
- Przecięcie,
- Różnica,
- Iloczyn kartezjański,
- Złączenie

# Złączenie INNER JOIN (1)

- INNER JOIN służy do łączenia tabel;
- INNER JOIN to inaczej JOIN czyli złączenie wewnętrzne (zawężające);
- Parametry do złączenia podajemy w nawiasie po słówku ON;
- Łączenie następuje wg kolumn które wskazujemy po jednej stronie i po drugiej stronie;
- Typu danych po obu stronach muszą być takie same;
- Przy złączeniu INNER JOIN (JOIN) wybierzemy tylko część wspólną obu zbiorów (tabel);



# Złączenie INNER JOIN (2)

```
SELECT *  
FROM dbo.EMP as e INNER JOIN dbo.CAR as c ON e.IdPrac=c.IdPrac
```

dbo.CAR

|   | NrRej   | Marka     | Rocznik | IdPrac |
|---|---------|-----------|---------|--------|
| 1 | P0841XY | FIAT 500  | 2013    | NULL   |
| 2 | P0745AX | Mercedes  | 2013    | 1      |
| 3 | P064231 | VOLVO S40 | 2012    | 2      |
| 4 | P0123AA | VOLVO V60 | 2012    | 4      |
| 5 | P0422VX | FIAT 500  | 2012    | 6      |

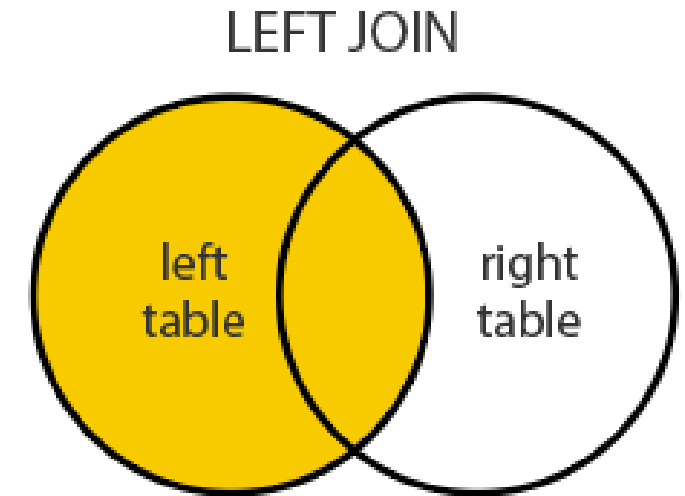
dbo.EMP

|   | IdPrac | Imie    | Nazwisko | DtZatr     | Stanowisko           | IdManager |
|---|--------|---------|----------|------------|----------------------|-----------|
| 1 | 1      | Adam    | Nowak    | 2010-01-01 | BOSS                 | NULL      |
| 2 | 2      | Piotr   | Kowalski | 2010-03-15 | Kierownik Sprzedaży  | 1         |
| 3 | 3      | Michał  | Pogodny  | 2010-04-04 | Sprzedawca           | 2         |
| 4 | 4      | Anna    | Dymna    | 2010-05-15 | Kierownik Marketingu | 1         |
| 5 | 5      | Jan     | Mały     | 2011-08-01 | Marketer             | 4         |
| 6 | 6      | Leopold | Stuff    | 2011-11-15 | Sprzedawca           | 2         |
| 7 | 7      | Monika  | Miła     | 2012-03-31 | Asystent             | 1         |
| 8 | 8      | Joanna  | Wesoła   | 2012-08-01 | Sprzedawca           | 2         |

|   | IdPrac | Imie    | Nazwisko | DtZatr     | Stanowisko           | IdManager | NrRej   | Marka     | Rocznik | IdPrac |
|---|--------|---------|----------|------------|----------------------|-----------|---------|-----------|---------|--------|
| 1 | 1      | Adam    | Nowak    | 2010-01-01 | BOSS                 | NULL      | P0745AX | Mercedes  | 2013    | 1      |
| 2 | 2      | Piotr   | Kowalski | 2010-03-15 | Kierownik Sprzedaży  | 1         | P064231 | VOLVO S40 | 2012    | 2      |
| 3 | 4      | Anna    | Dymna    | 2010-05-15 | Kierownik Marketingu | 1         | P0123AA | VOLVO V60 | 2012    | 4      |
| 4 | 6      | Leopold | Stuff    | 2011-11-15 | Sprzedawca           | 2         | P0422VX | FIAT 500  | 2012    | 6      |

# Złączenie LEFT OUTER JOIN (1)

- Zwraca wiersze dla których warunek złączenia jest spełniony;
- Zwraca wiersze z “lewej tabeli” dla których nie ma odpowiedników w prawej;
- Złączenie typu LEFT OUTER JOIN pozwala nam na uwzględnienie w wyniku danych, które nie posiadają swoich odpowiedników w złączanych tabelach;



# Złączenie LEFT OUTER JOIN (2)

| bajka |   |                    |
|-------|---|--------------------|
| id    |   | tytuł              |
|       | 1 | 101 Dalmatyńczyków |
|       | 2 | Flinstonowie       |
|       | 3 | Jetsonowie         |
|       | 4 | Epoka lodowcowa    |
|       | 5 | Rozbójnik Rumcajs  |
|       | 6 | Muminki            |
|       | 7 | Smerfy             |

| postac |          |         |
|--------|----------|---------|
| id     | bajka_id | imie    |
| 1      | 1        | Czika   |
| 2      | 1        | Pongo   |
| 3      | 2        | Wilma   |
| 4      | 2        | Fred    |
| 5      | 4        | Elka    |
| 6      | 4        | Maniek  |
| 7      | 6        | Migotka |
| 8      | 6        | Muminek |
| 9      |          | Maja    |
| 10     |          | Gucio   |
| 11     |          | Fiona   |
| 12     |          | Shrek   |

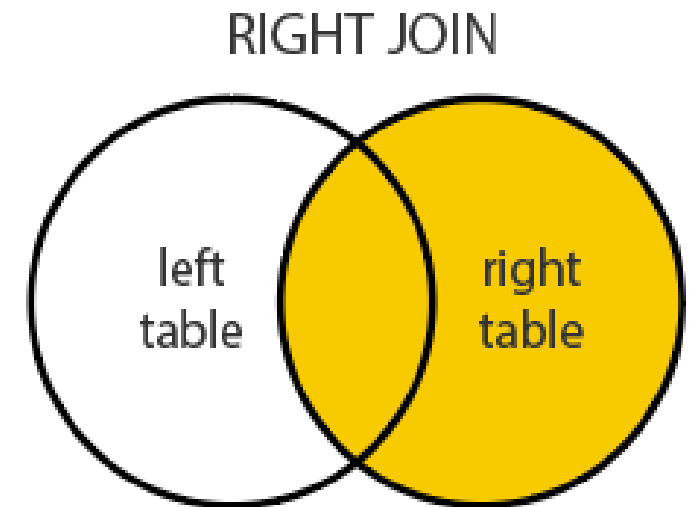
# Złączenie LEFT OUTER JOIN (3)

```
SELECT *  
FROM bajka LEFT OUTER JOIN postac  
ON bajka.id = postac.bajka_id;
```

| id | tytuł              | id | bajka_id | imie    |
|----|--------------------|----|----------|---------|
| 1  | 101 Dalmatyńczyków | 1  | 1        | Czika   |
| 1  | 101 Dalmatyńczyków | 2  | 1        | Pongo   |
| 2  | Flinstonowie       | 3  | 2        | Wilma   |
| 2  | Flinstonowie       | 4  | 2        | Fred    |
| 3  | Jetsonowie         |    |          |         |
| 4  | Epoka lodowcowa    | 5  | 4        | Elka    |
| 4  | Epoka lodowcowa    | 6  | 4        | Maniek  |
| 5  | Rozbójnik Rumcajs  |    |          |         |
| 6  | Muminki            | 7  | 6        | Migotka |
| 6  | Muminki            | 8  | 6        | Muminek |
| 7  | Smerfy             |    |          |         |

# Złączenie RIGHT OUTER JOIN (1)

- Jest złączeniem podobnym do LEFT OUTER JOIN;
- Zwraca wiersze z “prawej tabeli” dla których nie ma odpowiedników w lewej;
- Zwraca wiersze dla których warunek złączenia jest spełniony;





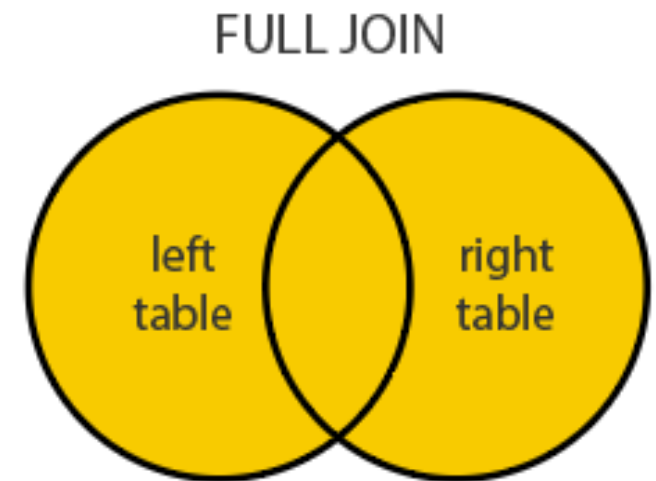
# Złączenie RIGHT OUTER JOIN (2)

```
SELECT *  
FROM bajka RIGHT OUTER JOIN postac  
ON bajka.id = postac.bajka_id;
```

| id | tytul              | id | bajka_id | imie    |
|----|--------------------|----|----------|---------|
| 1  | 101 Dalmatyńczyków | 1  | 1        | Czika   |
| 1  | 101 Dalmatyńczyków | 2  | 1        | Pongo   |
| 2  | Flinstonowie       | 3  | 2        | Wilma   |
| 2  | Flinstonowie       | 4  | 2        | Fred    |
| 4  | Epoka lodowcowa    | 5  | 4        | Elka    |
| 4  | Epoka lodowcowa    | 6  | 4        | Maniek  |
| 6  | Muminki            | 7  | 6        | Migotka |
| 6  | Muminki            | 8  | 6        | Muminek |
|    |                    | 9  |          | Maja    |
|    |                    | 10 |          | Gucio   |
|    |                    | 11 |          | Fiona   |
|    |                    | 12 |          | Shrek   |

# Złączenie OUTER JOIN (1)

- FULL OUTER JOIN jest złączeniem, która zwraca wiersze z połączenia LEFT OUTER JOIN i RIGHT OUTER JOIN;



# Złączenie OUTER JOIN (2)

```
SELECT *  
FROM bajka  
FULL OUTER JOIN postac ON bajka.id = postac.bajka_id;
```

| id | tytuł              | id | bajka_id | imie    |
|----|--------------------|----|----------|---------|
| 1  | 101 Dalmatyńczyków | 1  | 1        | Czika   |
| 1  | 101 Dalmatyńczyków | 2  | 1        | Pongo   |
| 2  | Flinstonowie       | 3  | 2        | Wilma   |
| 2  | Flinstonowie       | 4  | 2        | Fred    |
| 3  | Jetsonowie         |    |          |         |
| 4  | Epoka lodowcowa    | 5  | 4        | Elka    |
| 4  | Epoka lodowcowa    | 6  | 4        | Maniek  |
| 5  | Rozbójnik Rumcajs  |    |          |         |
| 6  | Muminki            | 7  | 6        | Migotka |
| 6  | Muminki            | 8  | 6        | Muminek |
| 7  | Smerfy             |    |          |         |
|    |                    | 9  |          | Maja    |
|    |                    | 10 |          | Gucio   |
|    |                    | 11 |          | Fiona   |
|    |                    | 12 |          | Shrek   |

# Funkcja Decode

Select decode (nazwa\_kolumny, wartość, zamiennik, wartość2, zamiennik2, wartość domyślna) [alias] from nazwa\_tabeli.

*Wypisać nazwę, cenę i podatek wszystkich towarów. Zamiast podawać wielkość podatku, wypisać słowa „pełna stawka” dla podatku 22%, lub „ulgowa stawka” dla podatku 7%. Dla pozostałych stawek wypisać „brak podatku”. Kolumnę tą nazwać „podatki”.*

Funkcja testuje wartość w podanej kolumnie i w zależności od wartości mieszczącej się w pierwszym argumencie zwraca wartość podaną w drugim argumencie. Jeśli nie znajdzie odpowiedniej wartości w podanych warunkach, wyświetli wartość domyślną podaną jako ostatni warunek.

```
SELECT tow_nazwa,  
       tow_cena,  
       decode(tow_podatek,  
              22, 'pełna stawka',  
              7, 'ulgowa stawka',  
              'brak podatku') podatki  
FROM towary;
```

```
SQL> select decode ( min salary,  
2 1000, 'płaca minimalna',  
3 20000, 'płaca za duża',  
4 'normalna wypłata') płace from jobs;
```

| PLACE            |
|------------------|
| placa minimalna  |
| placa minimalna  |
| placa za duza    |
| normalna wypłata |

# Null

- Specjalny znacznik w języku SQL, wskazujący, że dana nie istnieje w bazie danych;
- Ponieważ Null nie jest członkiem jakiejkolwiek domeny danych, nie jest on rozważany jako „wartość”;
- Porównywanie z Null nigdy nie może zwrócić Prawdy lub Fałszu, lecz zawsze trzeci logiczny wynik ”Nieznany”
- NULL nie jest równe 0.
- Większość funkcji agregacji ignoruje wartość NULL.
- NULL występuje w SQL jako predykat oraz wartość.

# Null

```
SELECT Imie, Nazwisko, Pensja  
FROM Kontakty  
WHERE Pensja IS NULL;
```

| Imie    | Nazwisko | Pensja |
|---------|----------|--------|
| Mikołaj | Nowaczyk |        |

# Null

Na początku napiszmy pytanie bez ISNULL'a

```
SELECT TOP 10
    [FirstName] AS Imię
    , [MiddleName] AS [Drugie imię]
    , [LastName] AS Nazwisko
FROM
    [AdventureWorks2008R2].[Person].[Person]
```

W efekcie uruchomienia zapytanie otrzymamy wynik jak poniżej

|           |      |             |
|-----------|------|-------------|
| Syed      | E    | Abbas       |
| Catherine | R.   | Abel        |
| Kim       | NULL | Abercrombie |
| Kim       | NULL | Abercrombie |
| Kim       | B    | Abercrombie |
| Hazem     | E    | Abolrous    |
| Sam       | NULL | Abolrous    |
| Humberto  | NULL | Acevedo     |
| Gustavo   | NULL | Achong      |
| Pilar     | NULL | Ackerman    |

```
SELECT TOP 10
    [FirstName] AS Imię
    , ISNULL([MiddleName], '') AS [Drugie imię]
    , [LastName] AS Nazwisko
FROM
    [AdventureWorks2008R2].[Person].[Person]
```

|           |    |             |
|-----------|----|-------------|
| Syed      | E  | Abbas       |
| Catherine | R. | Abel        |
| Kim       |    | Abercrombie |
| Kim       |    | Abercrombie |
| Kim       | B  | Abercrombie |
| Hazem     | E  | Abolrous    |
| Sam       |    | Abolrous    |
| Humberto  |    | Acevedo     |
| Gustavo   |    | Achong      |
| Pilar     |    | Ackerman    |

# Null

```
Select 100000 + NULL as wynik1,  
       (5000+300) * 2 - null as wynik2,  
       'Ala ma ' + 'kota' + null + '!' as wynik3
```

| Results  |        |        |        |
|----------|--------|--------|--------|
| Messages |        |        |        |
|          | wynik1 | wynik2 | wynik3 |
| 1        | NULL   | NULL   | NULL   |

**Wszystkie operacje z udziałem wartości nieznanej – dają zawsze w wyniku NULL.**

Niezależnie czy są wartości przechowywane jako zmienne, kolumny czy stałe – cokolwiek połączymy z NULL – da nam NULL.



# Null

|    | FullName          | FirstName | MiddleName | LastName    |
|----|-------------------|-----------|------------|-------------|
| 1  | Syed E Abbas      | Syed      | E          | Abbas       |
| 2  | Catherine R. Abel | Catherine | R.         | Abel        |
| 3  | Kim Abercrombie   | Kim       | NULL       | Abercrombie |
| 4  | Kim Abercrombie   | Kim       | NULL       | Abercrombie |
| 5  | Kim B Abercrombie | Kim       | B          | Abercrombie |
| 6  | Hazem E Abolrous  | Hazem     | E          | Abolrous    |
| 7  | Sam Abolrous      | Sam       | NULL       | Abolrous    |
| 8  | Humberto Acevedo  | Humberto  | NULL       | Acevedo     |
| 9  | Gustavo Achong    | Gustavo   | NULL       | Achong      |
| 10 | Pilar Ackeman     | Pilar     | NULL       | Ackeman     |
| 11 | Pilar G Ackeman   | Pilar     | G          | Ackeman     |
| 12 | Aaron B Adams     | Aaron     | B          | Adams       |
| 13 | Adam Adams        | Adam      | NULL       | Adams       |

```
SELECT FirstName + ' ' + ISNULL(MiddleName,'') + ' ' + LastName AS FullName  
FROM Person.Person
```

|    | FullName          |
|----|-------------------|
| 1  | Syed E Abbas      |
| 2  | Catherine R. Abel |
| 3  | Kim Abercrombie   |
| 4  | Kim Abercrombie   |
| 5  | Kim B Abercrombie |
| 6  | Hazem E Abolrous  |
| 7  | Sam Abolrous      |
| 8  | Humberto Acevedo  |
| 9  | Gustavo Achong    |
| 10 | Pilar Ackeman     |
| 11 | Pilar G Ackeman   |
| 12 | Aaron B Adams     |
| 13 | Adam Adams        |

# Not Null

```
SELECT nazwisko FROM studenci WHERE predykat IS NOT NULL;
```

```
SELECT Imie, Nazwisko, Pensja  
FROM Kontakty  
WHERE Pensja IS NOT NULL;
```

| Imie    | Nazwisko   | Pensja |
|---------|------------|--------|
| Juliusz | Miler      | 4000   |
| Paweł   | Sikora     | 4500   |
| Jan     | Kowalski   | 10000  |
| Adam    | Nowak      | 3000   |
| Adam    | Nowakowski | 3100   |
| Amir    | Miller     | 6000   |
| Zenon   | Malinowski | 1200   |
| Ewelina | Kujawa     | 1400   |
| Emil    | Podoliński | 4000   |

# Not Null

- Not służy do wyznaczania logicznej negacji wyrażenia;
- Dzięki użyciu NOT NULL wartość w danej kolumnie tabeli nie może być NULLEM.

# Q&A

# Źródła

[https://www.mechanikryki.pl/renata/pliki\\_pdf/SQL.pdf](https://www.mechanikryki.pl/renata/pliki_pdf/SQL.pdf)

<https://www.sqlpedia.pl/>

<https://icis.pcz.pl/~olga/dydaktyka/BAZYw02.p.pdf>

<https://tomaszkenig.pl/kurs-sql-server/>

<https://strefainzyniera.pl/artukul/11532/zlaczenia-krzyzowe-cross-join>

<https://www.samouczekprogramisty.pl/>

<https://avendi.edu.pl/>

<http://anonco.pl/>

Dziękuję za uwagę! 😊