



# Tworzenie zapytań do bazy danych na platformie MS SQL Server - SQL dla analityków

# Poznajmy się!



Dzień dobry,  
Mam na imię ...  
W pracy zajmuję się ...  
Mam doświadczenie w ...  
Od szkolenia oczekuję ...



# Cele kursu

- Zapoznanie się z zaawansowanymi opcjami zapytania select
- Praktyczne użycie metod wbudowanych
- Łączenie zapytań w celu efektywnego pobierania wyników
- Umiejętność integracji bazy z innymi źródłami danych



# Agenda

1. **Wykorzystanie funkcji szeregujących**
2. Nietypowe grupowanie danych
3. **Słowniki, sumy, różnice i iloczyny zbiorów**
4. **Wybrane funkcje** tekstowe, daty i czasu, **matematyczne**, sprawdzające
5. **Podzapytania**
6. **Zapytania CTE (Common Table Expression)**
7. **Tworzenie sparametryzowanych funkcji i procedur składowanych**
8. **Integracja z Excel i Power Query**

## Przerwy

10:30 - 10:45 **15min**

12:15 - 13:00 **45min**

14:25 - 14:40 **15min**



# Relacyjna baza danych

## przypomnienie podstawowych informacji i składni SELECT

# Relacyjne bazy danych, RDBMS

**RDB** (skrót od Relational Database) przechowują powiązane ze sobą dane w formie dwuwymiarowych tabel, które łączą się ze sobą za pomocą relacji.

**RDBMS** (skrót od Relational Database Management System) to zintegrowane środowisko programistyczne (IDE) do zarządzania relacyjnymi bazami danych. Pozwala ono w szczególności na zarządzanie przechowywaniem informacji, dostępem do nich oraz ich przetwarzaniem.

Systemy bazodanowe zapewniają wykonywanie na znajdujących się w nich obiektach operacji CRUD, w RDB zazwyczaj za pomocą języka SQL



# Narzędzia używane podczas szkolenia

1. MS SQL Server 2019, lokalnie na naszym komputerze

2. MS SQL Management Studio

3. MS Office 2021 (Excel, Access)

do migracji bazy z Access to SQL Server:

3. Microsoft Access Database Engine

<https://www.microsoft.com/en-us/download/details.aspx?id=54920>

4. Microsoft SQL Server Migration Assistant for Access

<https://www.microsoft.com/en-us/download/details.aspx?id=54255>

# Utworzenie baz danych:

## 1. Baza danych BS (plik **Baza\_Sprzedaz.sql**)

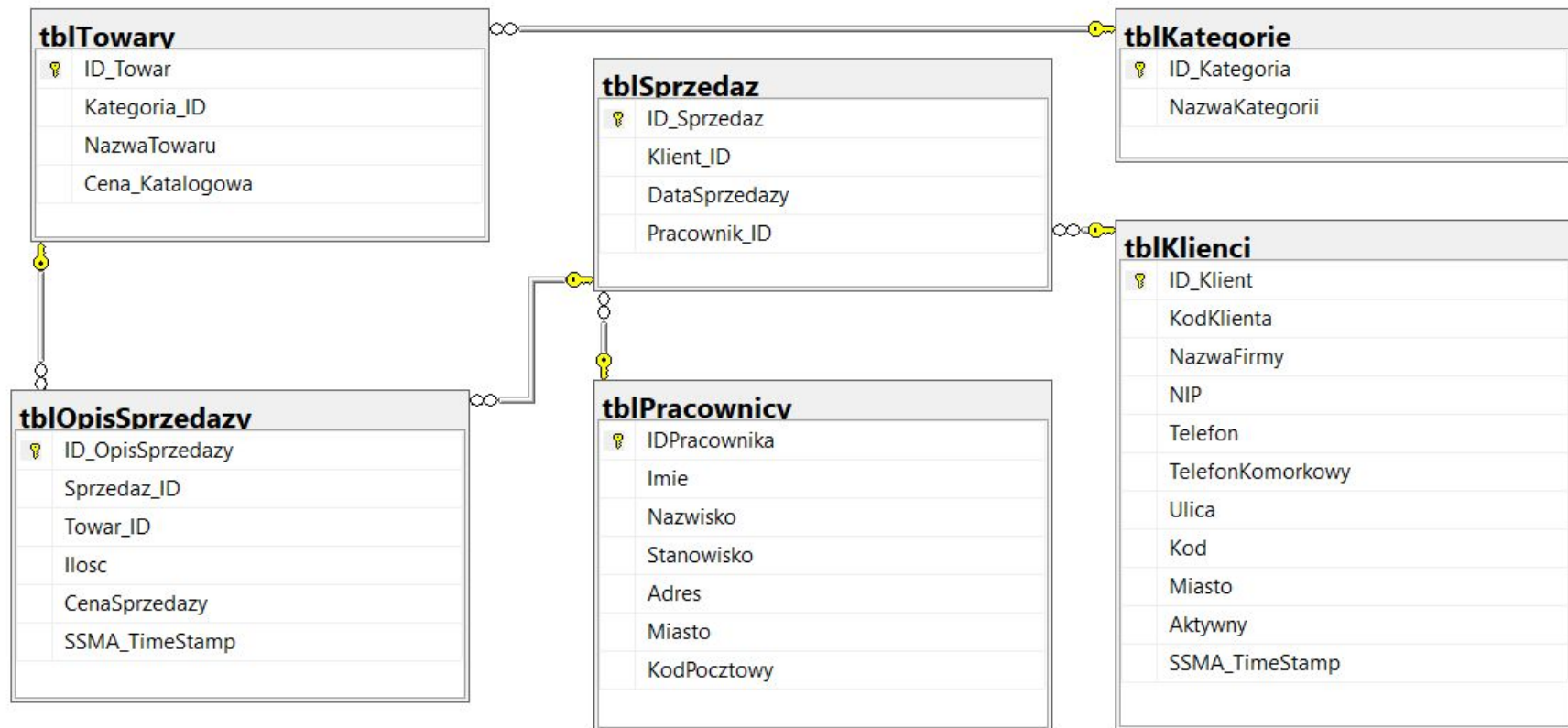
- Uruchomienie MS SQL Management Studio
- Utworzenie nowej bazy danych HM
- Uruchomienie skryptu zawierającego obiekty bazy i dane

## 2. Baza danych HM (Faktury, plik **HMdb.sql**)

- Uruchomienie MS SQL Management Studio
- Utworzenie nowej bazy danych HM
- Uruchomienie skryptu zawierającego obiekty bazy i dane



# Diagram relacji



# Więzy integralności

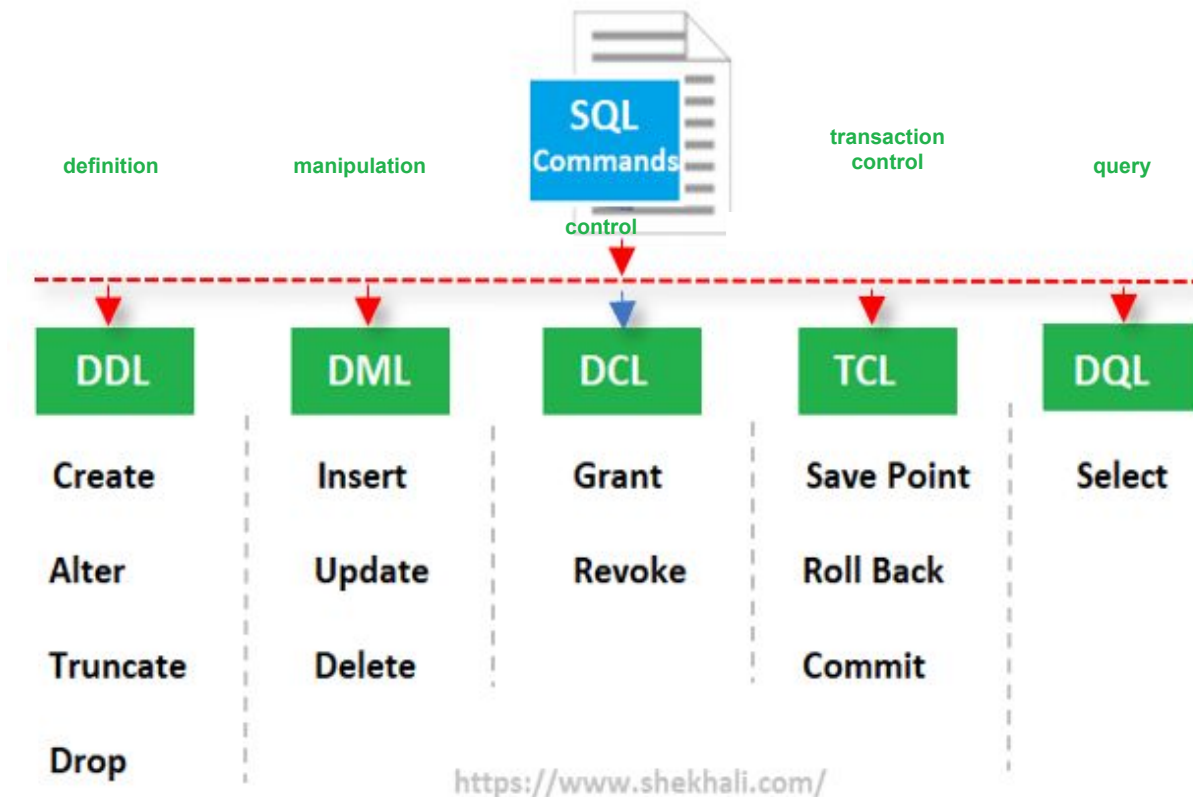
Właściwości tabel, które dbają o spójność i poprawność danych

- Klucze główne (PK) - unikalne identyfikatory wierszy
- Klucze obce (FK) - korespondują do odpowiedniego PK, nie mogą zawierać odniesień do nieistniejących PK
- Default, NOT NULL (ustawienie wartości domyślnej, wymuszenie wstawienia w polu danych)
- Check

```
CREATE TABLE Dzialy (  
    ID_Dzial INT PRIMARY KEY,  
    NazwaDzialu VARCHAR(100) NOT NULL UNIQUE  
);  
  
CREATE TABLE Pracownicy (  
    ID_Pracownik INT PRIMARY KEY, -- Klucz główny  
    ID_Dzial INT,  
    DataZatrudnienia DATE NOT NULL CHECK (DataZatrudnienia > '2000-01-01'),  
    Stanowisko NVARCHAR(50) NOT NULL DEFAULT 'Pracownik' -- Ograniczenie DEFAULT  
    FOREIGN KEY (ID_Dzial) REFERENCES Dzialy (ID_Dzial) -- Klucz obcy  
);
```

# SQL (Structured Query Language)

Język zapytań używany do komunikacji z bazami danych



# SELECT


Order	Element
5	SELECT
1	FROM
2	WHERE
3	GROUP BY
4	HAVING
6	ORDER BY

```
SELECT [ ALL | DISTINCT ] [lista_select]  
FROM tabela + [JOIN | APPLY | PIVOT | UNPIVOT]  
WHERE warunki_selekcji_dla_wierszy  
GROUP BY lista_group_by  
HAVING warunki_selekcji_dla_grup  
ORDER BY wyrażenie ASC | DESC;
```

```
SELECT pf.FakturaID, SUM(CenaSprz) AS wartosc  
FROM tbPozycjeFaktur pf  
JOIN tbFaktury f  
    ON f.IDFaktury = pf.FakturaID  
WHERE pf.FakturaID NOT IN (1,4,8)  
GROUP BY pf.FakturaID  
HAVING SUM(CenaSprz) > 33  
ORDER BY SUM(CenaSprz) DESC
```

# Warunek WHERE

Łańcuchy znakowe (char, varchar2) oraz daty  
zawsze wpisuje w apostrofach



## operatory porównania:

=, !=, >, <, >=, <=    **WHERE** kolumna >= 5 **AND** kolumna != 'Abc'                      'ABC' < 'ACC'

**WHERE** kolumna **IS NOT NULL** -- jeśli wartość jest porównywana z NULLEM

## zakresy i zbiory:

**WHERE** kolumna NOT **BETWEEN** 5 **AND** 10 (przedział domknięty)

**WHERE** kolumna NOT **IN** (1,2,4,6)

## operator LIKE (wzorce):

**WHERE** kolumna **LIKE** 'A%' OR kolumna **LIKE** '\_\_\_A'

OR kolumna **LIKE** '%A\_B'    OR kolumna **LIKE** 'A-D'

\_ - jeden znak,

% - dowolny ciąg znaków,

**a-d, 0-9** - znaki z zakresu od do

## łączenie kilku warunków:

**AND** | **OR** -- zwróć uwagę, że kolejny warunek po OR / AND musi znów składać się z pełnego porównania:

**WHERE** kolumna = 5 OR kolumna = 10

# zapytanie select, użycie funkcji wbudowanych

# Funkcję wbudowane

Możemy używać funkcji jako kolumn, w warunku oraz w sortowaniu:

```
SELECT  
  
LTRIM(NazwaTowaru), ID_Towar + Kategoria_ID  
  
FROM tblTowary  
  
WHERE LTRIM(NazwaTowaru) like 'Czekolada%'  
  
ORDER BY ID_Towar + Kategoria_ID;
```

Wyniki funkcji mogą być pobierane zapytaniem select bez podania tabeli:

```
SELECT GETDATE(), ROUND(SIN(14),2)
```

# Funkcję skalarne (wbudowane)

```
SELECT
ROUND(15.193, 1) AS zaokrąglenie,
POWER(3, 2) as potęga,
FLOOR(15.7) AS podłoga,    -- to samo: CAST(15.7 AS INT)
CEILING(15.7) AS sufit;
```

```
SELECT
LEN('Hello World') AS długość_napisu,
LEFT('Hello World', 5) AS lewa_część_napisu,
RIGHT('Hello World', 5) AS prawa_część_napisu,
SUBSTRING('Hello World',1, 3) AS wycinek,
REPLACE('Hello World', 'World', 'DB2') AS napis_z_zastąpieniem,
UPPER('Hello World') AS wielkie_litery,
LOWER('Hello World') AS małe_litery;
```

```
SELECT (Imie + ' ' + Nazwisko) AS pełne_imię FROM tblPracownicy;
```

```
SELECT NazwaFirmy,
       Telefon,
       COALESCE(TelefonKomorkowy, 'brak numeru komórkowego') AS TelefonKomorkowy
FROM tblKlienci;
```



# Funkcję skalarne

```
SELECT
CAST(GETDATE() AS DATE) AS aktualna_data,
CAST(GETDATE() AS TIME) AS aktualny_czas,
'Data: ' + CAST(GETDATE() AS VARCHAR) AS data_napis,
GETDATE() AS aktualna_data_i_czas,
MONTH(GETDATE()) AS miesiąc, -- to samo z YEAR, DAY
DATENAME(year, GETDATE()) AS rok,
DATENAME(day, GETDATE()) AS dzien,
DATENAME(weekday, GETDATE()) AS nazwa_dnia,
DATEPART(weekday, GETDATE()) AS numer_dnia_tygodnia,
CAST('2023-03-15' AS DATE) AS zmiana_napisu_na_date;
```

```
SELECT
ID_Sprzedaz,
DATEDIFF(day, DataSprzedazy, GETDATE()) AS roznica_dni,
DATEDIFF(month, DataSprzedazy, GETDATE()) AS różnica_miesiący
FROM tblSprzedaz;
```

# Polecenie OVER

Używana z funkcjami okienkowymi, które umożliwiają wykonanie obliczeń na zestawie wierszy (oknie) powiązanych z bieżącym wierszem. Użycie OVER pozwala na zastosowanie funkcji **agregujących**, **rankingu** i **analitycznych** bez konieczności grupowania danych za pomocą klauzuli GROUP BY.

```
SELECT
    FakturaID,
    SUM(CenaSprz * Ilosc) AS TotalAmount
FROM dbo.tbPozycjeFaktur
GROUP BY FakturaID;
```

```
SELECT DISTINCT
    FakturaID, * -- dane się nie agregują,
możemy wyświetlić wszystkie kolumny
    SUM(CenaSprz * Ilosc) OVER(PARTITION BY
FakturaID) AS TotalAmount
FROM dbo.tbPozycjeFaktur;
```

CustID	OrderID	TotalDue
1	101	\$100
2	102	\$150
1	103	\$90
3	104	\$80
2	105	\$200
1	106	\$150



Partition by CustID  
Order by TotalDue

CustID	OrderID	TotalDue
1	103	\$90
1	101	\$100
1	106	\$150

CustID	OrderID	TotalDue
2	102	\$150
2	105	\$200

CustID	OrderID	TotalDue
3	104	\$80

# Funkcje rankingowe

```
SELECT f.IDFaktury, p.CenaSprz,  
       ROW_NUMBER() OVER(PARTITION BY f.IDFaktury ORDER BY p.CenaSprz DESC) AS 'Numer',  
       RANK() OVER(PARTITION BY f.IDFaktury ORDER BY p.CenaSprz DESC) AS 'RangaCeny',  
       DENSE_RANK() OVER(PARTITION BY f.IDFaktury ORDER BY p.CenaSprz DESC) AS 'Gęst_Ranga',  
       NTILE(4) OVER(PARTITION BY f.IDFaktury ORDER BY p.CenaSprz DESC) AS 'KwartylCeny'  
FROM tbFaktury f  
JOIN tbPozycjeFaktur p ON f.IDFaktury = p.FakturaID  
ORDER BY  
       f.IDFaktury, p.CenaSprz DESC;
```

ZIP Code	ROW_NUMBER	RANK	DENSE_RANK
123456	1	1	1
123456	2	1	1
123456	3	1	1
654321	4	4	2
654321	5	4	2
234567	6	6	3
345678	7	7	4
456789	8	8	5
456789	9	8	5

# FETCH, OFFSET

```
-- Wybieranie z wyniku wierszy 21-30
SELECT *
FROM tblPracownicy
ORDER BY Nazwisko
OFFSET 5 ROWS           -- Pomiń pierwsze 5 wierszy
FETCH NEXT 10 ROWS ONLY; -- Następnie pobierz 10 wierszy
```

## CASE - wyrażenie warunkowe

```
SELECT ID_OpisSprzedazy,
       Towar_ID,
       CASE
           WHEN Ilosc <= 5 THEN 'Pojedyncza'
           WHEN Ilosc <= 15 THEN 'Detaliczna'
           ELSE 'Hurtowa'
       END AS typSprzedazy
FROM tblOpisSprzedazy
```

# zaawansowane agregacje

# Funkcję agregującą

```
SELECT
    AVG(CenaSprz) AS ŚredniaCena,
    SUM(CenaSprz) AS SumaWszystkichCen,
    COUNT(1) AS IlośćWierszy,
    MAX(CenaSprz) AS MaksymalnaCena,
    MIN(CenaSprz) AS MinimalnaCena,
    ROUND(AVG(CenaSprz), 2) AS ŚredniaCenaZaokrąglona, -- zaokrąglenie wyniku
    AVG(COALESCE(CenaSprz, 0)) AS ŚredniaCenaZNull -- jeśli w danych mogą
    pojawić się wartości NULL
FROM
    tbPozycjeFaktur;
```

Name	Value
A	10
A	20
B	40
C	20
C	50



$\Sigma$

```
SELECT
    Name,
    SUM(Value)
FROM
    sample_table
GROUP BY
    Name;
```



Name	SUM(Value)
A	30
B	40
C	70

```
SELECT STRING_AGG(nazwa, ', ')
FROM tbKlienci
```

Employee Names
Scott
Williams
Mary
Mike
John
George



Teammates
Scott, Williams, Mary, Mike, John, George

# Grupowanie

```
SELECT genre, AVG(price)
FROM books
GROUP BY genre
HAVING AVG(price) > 8
```

UWAGA! przy GROUP BY trzeba dodożyć do SELECT  
wszystkie kolumny które chcemy wyświetlić

title	genre	price
book 1	adventure	11.90
book 2	fantasy	8.49
book 3	romance	9.99
book 4	adventure	9.99
book 5	fantasy	7.99
book 6	romance	5.88

genre	avg_price
adventure	$(11.90 + 9.99)/2$ 10.945
fantasy	$(8.49 + 7.99)/2$ 8.24
romance	$(9.99 + 5.88)/2$ 7.935

# GROUP BY ROLLUP

Generuje agregacje na wszystkich poziomach hierarchii.

-- zwraca podsumowanie dla każdego TowarID, dla każdej FakturaID oraz ogólne podsumowanie.

## SELECT

```
p.TowarID,  
p.FakturaID,  
SUM(p.CenaSprz) AS SumaCen
```

## FROM

```
tbPozycjeFaktur p
```

## GROUP BY

```
ROLLUP(p.TowarID, p.FakturaID);
```

wszystkie towary i faktury  
wszystkie faktury dla towaru 1

wszystkie faktury dla towaru 2

Results		Messages	
	TowarID	FakturaID	SumaCen
1	NULL	NULL	18366,50
2	1	NULL	102,00
3	1	1	13,00
4	1	2	13,00
5	1	3	13,00
6	1	4	13,00
7	1	7	12,50
8	1	33	12,50
9	1	55	12,50
10	1	57	12,50
11	2	NULL	290,00
12	2	1	29,00
13	2	9	29,00
14	2	11	29,00
15	2	12	29,00
16	2	29	29,00
17	2	22	29,00

UWAGA! ROLLUP i następne komendy (CUBE, GROUPING SETS) nie obsługują funkcji STRING\_AGG!



# GROUP BY CUBE

Generuje wszystkie możliwe agregacje dla podanych kolumn.

```
-- Zwraca podsumowanie dla każdego  
TowarID, każdej FakturaID, obu  
razem oraz ogólne podsumowanie.
```

**SELECT**

```
    p.TowarID,  
    p.FakturaID,  
    SUM(p.CenaSprz) AS SumaCen
```

**FROM**

```
    tbPozycjeFaktur p
```

**GROUP BY**

```
    CUBE (p.TowarID, p.FakturaID);
```

wszystkie towary per faktura

	TowarID	FakturaID	SumaCen
574	41	97	45,00
575	47	97	36,00
576	NULL	97	163,50
577	31	98	69,00
578	NULL	98	69,00
579	46	99	45,00
580	NULL	99	45,00
581	26	100	29,00
582	28	100	56,00
583	NULL	100	85,00
584	NULL	NULL	18366,50
585	1	NULL	102,00
586	2	NULL	290,00
587	3	NULL	180,00
588	4	NULL	333,00
589	5	NULL	234,00
590	6	NULL	506,00

wszystkie towary i faktury

wszystkie faktury per towar

Query executed successfully.

# GROUPING SETS

Pozwala określić wiele zestawów grupowania w jednym zapytaniu, najbardziej elastyczny sposób

-- Możemy grupować oddzielnie po TowarID, po FakturaID oraz po obu naraz.

**SELECT**

```
p.TowarID,  
p.FakturaID,  
SUM(p.CenaSprz) AS SumaCen
```

**FROM**

```
tbPozycjeFaktur p
```

**GROUP BY**

**GROUPING SETS(**

```
(p.TowarID),  
(p.FakturaID),  
(p.TowarID, p.FakturaID)
```

```
);
```

EFEKT - bardzo podobny jak przy CUBE, tyle że brak ogólnego podsumowania (wszystkie faktury, wszystkie towary) Nie ma możliwości zrobienia takiego podsumowania tą komendą

	TowarID	FakturaID	SumaCen
574	41	97	45,00
575	47	97	36,00
576	NULL	97	163,50
577	31	98	69,00
578	NULL	98	69,00
579	46	99	45,00
580	NULL	99	45,00
581	26	100	29,00
582	28	100	56,00
583	NULL	100	85,00
584	NULL	NULL	18366,50
585	1	NULL	102,00
586	2	NULL	290,00
587	3	NULL	180,00
588	4	NULL	333,00
589	5	NULL	234,00
590	6	NULL	506,00

Query executed successfully.

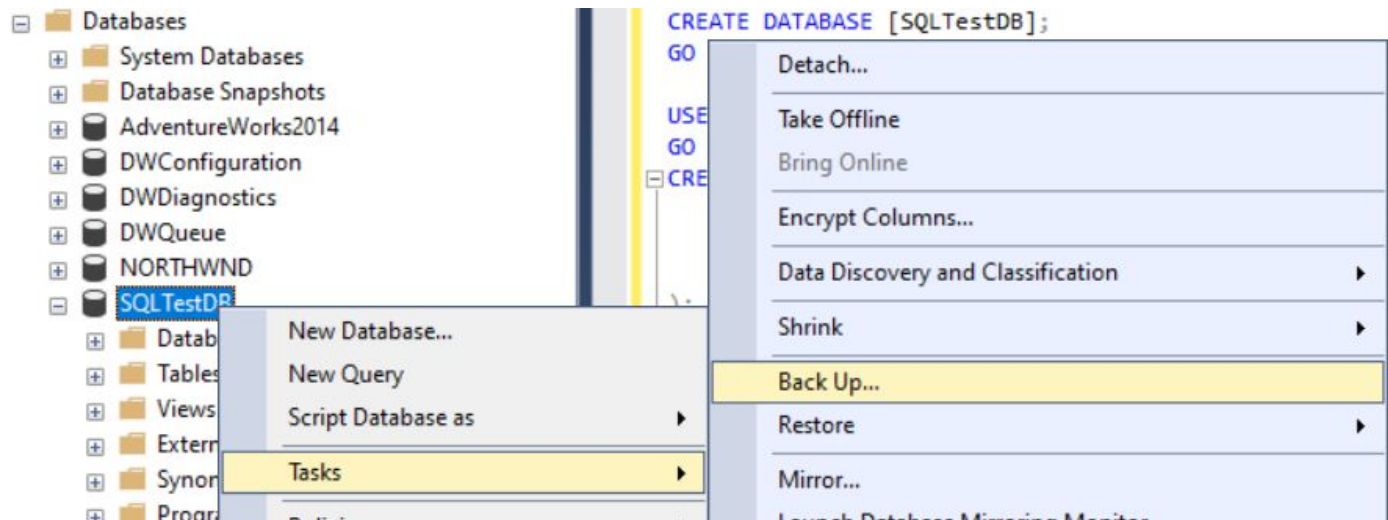
# SQL DML

## zapytania modyfikujące dane

```
ae-ruby-debug
ae-search-in-files
ae-shell
ae-subprocess-inspector
ae-term
test-shutdown-after-startup
lib
tcl
test
README
arcadia.gemspec
arcadia.todo
```

# Bezpieczeństwo danych przy operacjach DML

Backup - zapis aktualnego stanu bazy (wszystkich /wybranych obiektów i danych)



Używanie transakcji, które potem można zatwierdzić (COMMIT), bądź cofnąć (ROLLBACK)

```
BEGIN TRANSACTION
-- operacje zmieniające dane/obiekty
COMMIT TRANSACTION -- zatwierdzenie
-- lub wycofanie
ROLLBACK TRANSACTION
```

# INSERT INTO SELECT

```
INSERT INTO some_table (column1, column2, column3...)
VALUES (value1, value2, value3...)
```

możemy pominąć nawias z kolumnami, wtedy **musimy** podać wartości dla **wszystkich kolumn tabeli**. Z drugiej strony, jeśli uzupełniamy tylko część kolumn, to **nie** możemy pominąć tych, oznaczonych w tabeli jako **NOT NULLABLE**

```
INSERT INTO some_table
VALUES (value1, value2, value3...)
```

Nazwy kolumn można pominąć, wtedy trzeba podać wartości dla wszystkich kolumn, w kolejności jak w definicji tabeli

**W wielu bazach klucze główne są zdefiniowane jako samouzupełniające się** - wtedy podanie klucza głównego można pominąć. Dodatkowo nie podajemy go jeśli nie definiujemy listy kolumn:

```
INSERT INTO tblTowary (ID_Towar, Kategoria_ID, NazwaTowaru,
Cena_Katalogowa) VALUES (36, 5, 'Papaja', 5.5)
```

```
INSERT INTO tblTowary
VALUES (36, 5, 'Papaja', 5.5) -- błąd
```

```
INSERT INTO tblTowary - zaleca się dodać nazwy kolumn, tylko bez id
VALUES (5, 'Papaja', 5.5)
```

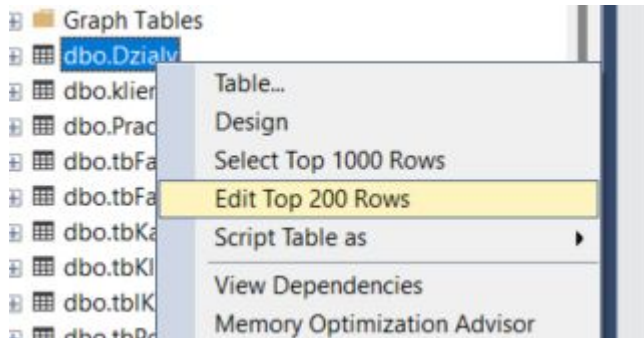
# INSERT INTO SELECT

Wstawienie w kolejne kolumny w tabeli wierszy zwróconych przez zapytanie select poniżej.

```
INSERT INTO HM.dbo.tbKlienci
SELECT NazwaFirmy, Miasto, Kod, Ulica, 'N/A'
FROM BS.dbo.tblKlienci
```

**Zapytanie insert into select jest jedną transakcją - znaczy to, że w momencie kiedy nie uda się wstawienie chociaż jednego z wierszy wszystkie inserty zostają wycofane.**

Dodawać, edytować i usuwać dane w tabeli możemy także przez interface w SQL Management Studio



# UPDATE

```
UPDATE tblTowary  
SET NazwaTowaru = 'Zaktualizowana nazwa produktu'  
WHERE NazwaTowaru = 'Stara nazwa produktu';
```

```
UPDATE tblOpisSprzedazy  
SET Ilosc = Ilosc + 3  
WHERE Towar_ID = 103
```

```
UPDATE tblOpisSprzedazy  
SET Ilosc = Ilosc * 1.1  
WHERE Towar_ID = 103
```

```
UPDATE tblOpisSprzedazy  
SET Ilosc = (SELECT MAX(Ilosc) FROM tblOpisSprzedazy)  
WHERE Towar_ID = 103 AND Sprzedaz_ID =  
(SELECT ID_Sprzedaz FROM tblSprzedaz WHERE Klient_ID = 4);
```

# DELETE

-- PK rekordu **nie jest** FK w innej tabeli

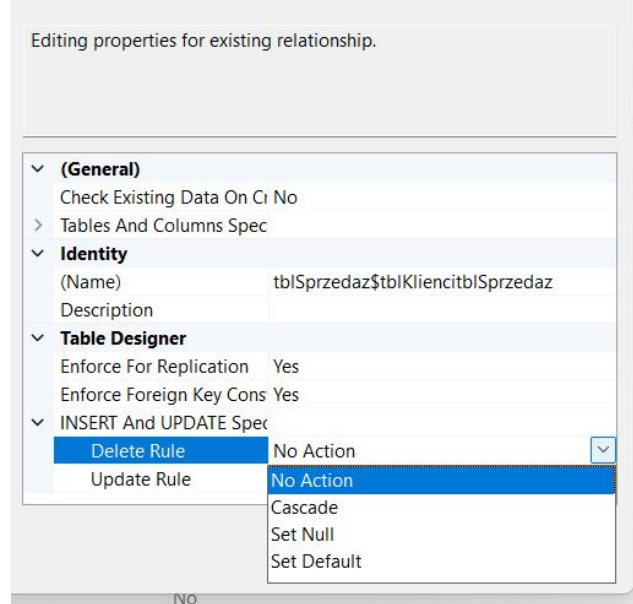
```
DELETE FROM tblTowary  
WHERE ID_Towar = 36
```

-- PK rekordu **jest** FK w innej tabeli, przy próbie usunięcia pojawi się błąd

```
DELETE FROM tblTowary  
WHERE ID_Towar = 15
```

-- możliwe rozwiązanie - Zasymulowanie akcji set null/  
cascade/ wstawienie innej wartości w powiązane pola

Najbezpieczniejszym ustawieniem jest **NO ACTION**. Jeśli chcemy usunąć rekord odwołujący się do innej tabeli musimy ręcznie zrobić DELETE/UPDATE na wszystkich tabelach z odwołaniami





# Podzapytania, CTE

# Podzapytania (subselect)

```
SELECT MAX(cena) FROM tbTowary;  
-- Do powyższego nie da się dodać niezgrupowanych pól  
SELECT cena , wykonawca + ' - ' + tytuł  
FROM tbTowary  
WHERE cena = (SELECT MAX(cena) FROM tbTowary);
```

100 %

Results Messages

	(No column name)
1	99,00

	cena	(No column name)
1	99,00	George Michael - Twenty Five

```
SELECT Sprzedaz_ID, CenaSprzedazy -  
(SELECT AVG(CenaSprzedazy)  
FROM tblOpisSprzedazy)  
FROM tblOpisSprzedazy
```

00 %

Results Messages

	Sprzedaz_ID	(No column name)
1	22	-11,8985
2	58	10,1015

```
SELECT Miasto FROM tblPracownicy  
SELECT KodKlienta, Miasto FROM tblKlienci  
WHERE Miasto IN  
(SELECT Miasto FROM tblPracownicy)
```

100 %

Results Messages

	Miasto
1	Warszawa
2	Sochaczew

	KodKlienta	Miasto
1	AL	WARSZAWA
2	CR	WARSZAWA

Podzapytanie musi działać jako niezależna kwerenda. Może się pojawić w dowolnej sekcji, zwykle SELECT, WHERE, FROM/JOIN

# Podzapytania skorelowane

```
SELECT
    f.IDFaktury,
    f.KlientID,
    pf.TotalCena
FROM
    tbFaktury f
JOIN
    (SELECT
        FakturaID,
        SUM(CenaSprz) AS TotalCena
        FROM tbPozycjeFaktur
        WHERE Ilosc > 5 -- Pre-filtrowanie
        GROUP BY FakturaID
    ) pf
ON f.IDFaktury = pf.FakturaID;
```

```
SELECT Sprzedaz_ID, CenaSprzedazy -
    (SELECT AVG(CenaSprzedazy)
     FROM tblOpisSprzedazy)
FROM tblOpisSprzedazy
```

00 %

Results Messages

	Sprzedaz_ID	(No column name)
1	22	-11,8985
2	58	10,1015

Podzapytania, które łączą się z głównym zapytaniem. Ważne by polom i tabelom, oraz całemu podzapytaniu nadać aliasy inne niż w głównym zapytaniu

# CTE (Common Table Expression)

```
-- wersja z subselect
SELECT cena , wykonawca + ' - ' + tytuł
FROM tbTowary
WHERE cena = (SELECT MAX(cena) FROM tbTowary);

-- wersja z CTE
WITH NajdrozszyTowar AS (
    SELECT MAX(cena) AS MaxCena
    FROM tbTowary
)
SELECT cena, wykonawca + ' - ' + tytuł
FROM tbTowary
WHERE cena = (SELECT MaxCena FROM NajdrozszyTowar)
```

Wspólne wyrażenie tabelowe (CTE) można stosować zamiennie z subselect. Na czas wykonania zapytania utworzona jest wirtualna tabela tymczasowa zawierająca widok zapytania zawartego w CTE. Używamy jej jak każdej innej tabeli. W CTE **wszystkie pola muszą mieć nazwy**

Przerobienie subselect na CTE:

1. Zapisujemy to co było subselectem jako CTE w WITH
2. Potrzebne pola wyciągamy z CTE jak z każdej innej tabeli

```
SELECT KodKlienta, Miasto FROM tblKlienci
WHERE Miasto IN
    (SELECT Miasto FROM tblPracownicy)

-- wersja z CTE
WITH miasta_pracownicy AS(
    SELECT miasto from tblPracownicy
)
SELECT KodKlienta, Miasto FROM tblKlienci
WHERE Miasto IN (SELECT * FROM miasta_pracownicy)
```

# CTE (Common Table Expression)

```
-- Użycie CTE dla prefiltrowania przed JOIN
WITH PrefiltrowanePozycje AS (
    SELECT FakturaID, SUM(CenaSprz) AS TotalCena
    FROM tbPozycjeFaktur
    WHERE Ilosc > 5
    GROUP BY FakturaID
)
SELECT f.IDFaktury, f.KlientID, p.TotalCena
FROM tbFaktury f
JOIN PrefiltrowanePozycje p ON f.IDFaktury = p.FakturaID

-- Użycie CTE dla dodatkowej kolumny
WITH SumaCen AS (
    SELECT FakturaID, SUM(CenaSprz) AS TotalCena
    FROM tbPozycjeFaktur
    GROUP BY FakturaID
)
SELECT f.IDFaktury, f.KlientID, p.TotalCena
FROM tbFaktury f
LEFT JOIN SumaCen p ON f.IDFaktury = p.FakturaID
```

CTE łączymy z głównym zapytaniem tak jak każdą inną tabelę; najczęściej nadajemy CTE dłuższą nazwę, opisującą dane z zapytania którego używa. Używając jej w JOIN możemy nadać inny alias.

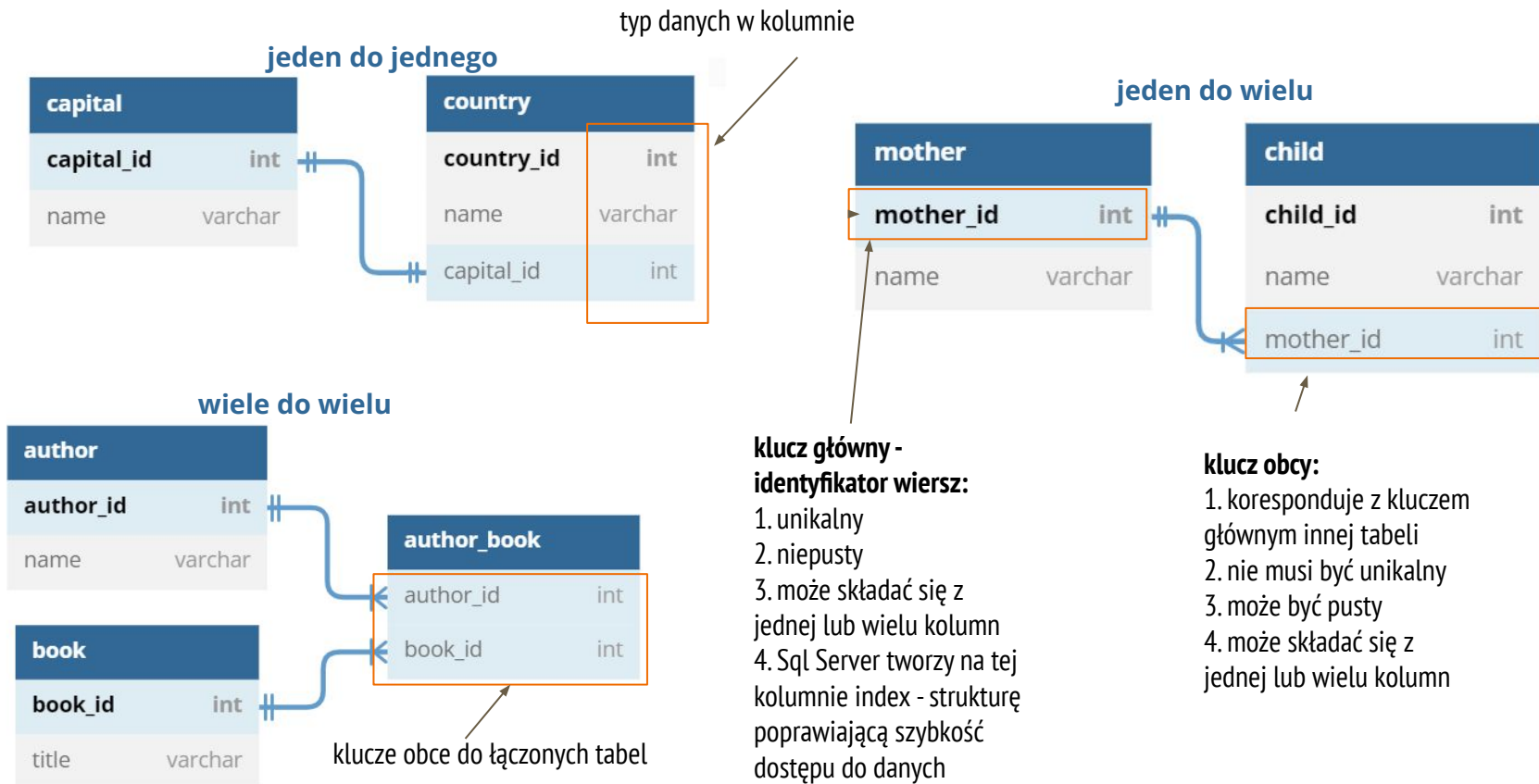
CO działa szybciej?

- CTE sprawdza się lepiej, gdy wykorzystujemy je wielokrotnie (w select, w where)
- Subselect sprawdza się lepiej przy małych zapytaniach, używane jednokrotnie w zapytaniu
- Przy problemach z wydajnością warto porównać czas wykonania obu metod



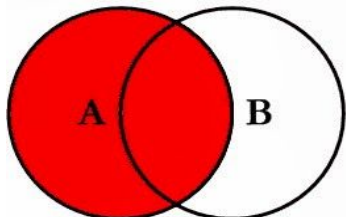
# UNIE, APPLY, PIVOT Złączenia tabel

# Relacje między tabelami w bazie danych

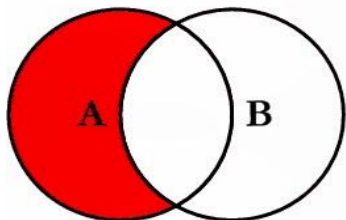


# Złączenia JOIN

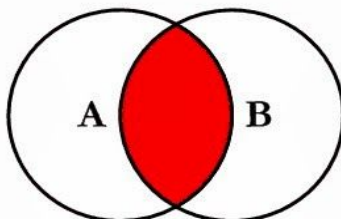
## SQL JOINS



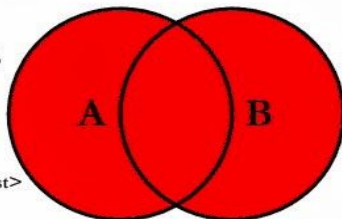
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



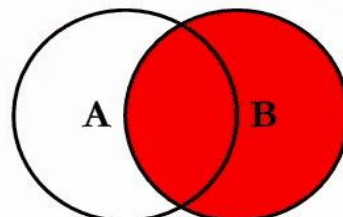
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



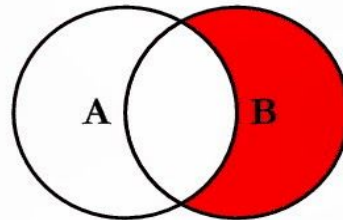
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



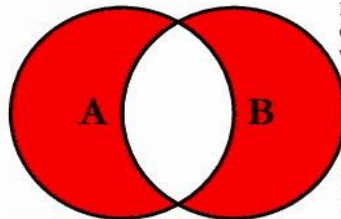
```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```



# Złączenia pionowe

Query1

A
B
C

C
D
E

Query2

UNION

A
B
C
D
E

UNION ALL

A
B
C
C
D
E

EXCEPT

A
B
C
D
E

INTERSECT

A
B
C
D
E

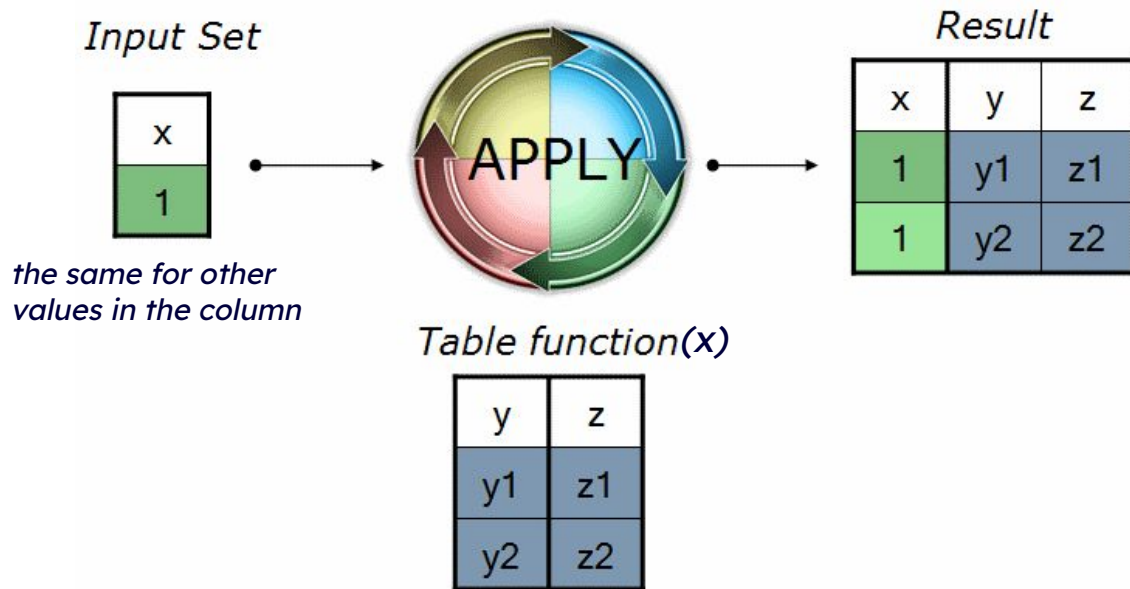
```
SELECT UPPER(Miasto)  
FROM tblPracownicy
```

```
INTERSECT  
-- UNION  
-- UNION ALL  
-- EXCEPT
```

```
SELECT UPPER(Miasto)  
FROM tblKlienci;
```

**WAŻNE** Wszystkie połączone zapytania muszą mieć tą samą liczbę kolumn takiego samego typu

# Złączenia APPLY

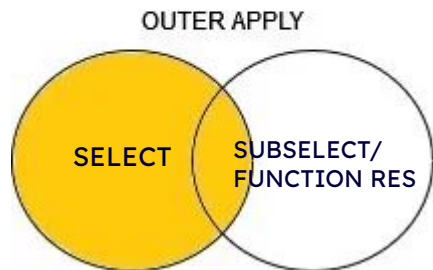
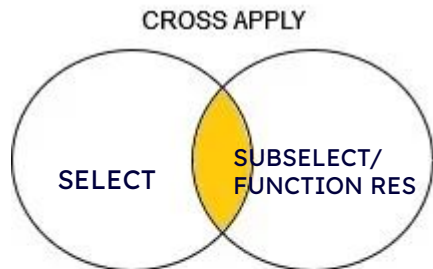


Łączenie zapytań za pomocą APPLY

Często przydaje się do uproszczenia zapytań.

Klauzula APPLY to potężne narzędzie, które pozwala na **wykonywanie funkcji tabelarycznych (takich jak STRING\_SPLIT) lub zapytań zwracających tabelę dla każdego wiersza w lewej tabeli**. Istnieją dwa rodzaje APPLY: CROSS APPLY i OUTER APPLY.

# OUTER APPLY, CROSS APPLY



```
CREATE TABLE SplitString (string NVARCHAR(100))

INSERT INTO SplitString VALUES ('2022-11-24')
INSERT INTO SplitString VALUES ('2022-04-01')

SELECT * FROM SplitString

SELECT * FROM SplitString s
CROSS APPLY STRING_SPLIT(s.string, '-')

```

10 %

Results Messages

	string
1	2022-11-24
2	2022-04-01

	string	value
1	2022-11-24	2022
2	2022-11-24	11
3	2022-11-24	24
4	2022-04-01	2022
5	2022-04-01	04
6	2022-04-01	01

# PIVOT

dbo.Orders TestPivot

Orderid	Custid	Orderdate	OrderYear	Shippe...	OrderValue	Freight	Shipname	Shipaddress	Shipcity	Shipregion	Shipcountry
10298	37	2018-09-05 00:00:00.000	2018	2	3127.000000	168.22	Destination AT...	4567 Johnstown Road	Cork	Co. Cork	Ireland
10309	37	2018-09-19 00:00:00.000	2018	1	1762.000000	47.30	Destination AT...	4567 Johnstown Road	Cork	Co. Cork	Ireland
10332	51	2018-10-17 00:00:00.000	2018	2	2233.600000	52.84	Ship to 51-B	7890 rue St. Laurent	Montréal	Québec	Canada
10335	37	2018-10-22 00:00:00.000	2018	2	2545.200000	42.11	Destination AT	4567 Johnstown Road	Cork	Co. Cork	Ireland



Table expression

Shipcountry	Freight	OrderYear
Ireland	168.22	2018
Ireland	47.30	2018
Canada	52.84	2018
Ireland	42.11	2018
Canada	15.66	2018
Ireland	124.12	2018
Canada	20.39	2018
Ireland	35.03	2018
Norway	93.63	2018
Canada	47.42	2018



PIVOT

Result table

Shipcountry	2018	2019	2020
Argentina	NULL	117.66	480.92
Canada	136.31	1687.76	374.02
Ireland	416.78	980.77	1357.69
Norway	93.63	52.01	129.86

Spreading element  
(distinct values)

Aggregate elements  
SUM(Freight)

Grouping element  
Group By(Shipcountry)

Derived table version

```

SELECT 3 PVT.Shipcountry
        ,PVT.[2018]
        ,PVT.[2019]
        ,PVT.[2020]
FROM (
    1 SELECT 1 ord.Shipcountry
          2 ,ord.Freight
          3 ,OrderYear
        FROM dbo.Orders_testPivot ord
    ) AS tabExpr
PIVOT (
    2 SUM(Freight)
        FOR OrderYear
        IN ([2018],[2019],[2020])
    ) AS PVT;
    
```

Annotations for Derived table version:

- 3: Result table (SELECT clause)
- 1: Table expression (FROM clause)
- 1: Grouping element (ord.Shipcountry)
- 2: Aggregate element (SUM(Freight))
- 3: Spreading element (OrderYear)
- 2: Result table's column names (IN clause)

CTE version

```

;WITH tabExpr AS
(
    1 SELECT 1 ord.Shipcountry
          2 ,ord.Freight
          3 ,OrderYear
        FROM dbo.Orders testPivot ord
    )
SELECT 3 PVT.Shipcountry
        ,PVT.[2018]
        ,PVT.[2019]
        ,PVT.[2020]
FROM tabExpr
PIVOT (
    2 SUM(Freight)
        FOR OrderYear
        IN ([2018],[2019],[2020])
    ) AS_PVT;
    
```

Annotations for CTE version:

- 1: Table expression (WITH clause)
- 3: Result table (SELECT clause)
- 2: Result table's column names (IN clause)

# Widoki, procedury składowane

# Widoki

Widokami nazywamy obiekty bazy danych, które są zapisanymi zapytaniami do bazy. Warto tworzyć widoki z zapytań, które często wykonujemy na bazie. Można je potem stosować w **SELECT** identycznie jak tabele.

```
-- tworzenie widoku (lub update opcjonalnie)
-- alter używamy, gdy chcemy zmienić istniejący widok, jeśli już istnieje
CREATE OR ALTER VIEW avg_sprzedaz AS (SELECT avg(CenaSprz) AS srednia_sp FROM
tbPozycjeFaktur);
```

```
SELECT * FROM avg_sprzedaz
SELECT * from tbPozycjeFaktur WHERE CenaSprz > (SELECT TOP 1 srednia_sp FROM
avg_sprzedaz)
```

```
DROP VIEW avg_sprzedaz -- usunięcie widoku
```

Komendy **CREATE**, **ALTER** i **DROP** są komendami języka DDL (data definition language). Służą też do tworzenia innych struktur w bazie, jak tabele, funkcje, indeksy itp.

# PROCEDURE Składowane

```
CREATE PROCEDURE StatystykiKlienta @SumaCen NUMBER, @IloscTransakcji INT
AS
BEGIN
    WITH StatystykiKlienta AS (
        -- full QUERY at script
    )
    SELECT *
    FROM tbKlienci k
    JOIN StatystykiKlienta sk ON k.IDKlienta = sk.KlientID
    WHERE sk.SumaZakupow > @SumaCen
    AND sk.LiczbaTransakcji > @IloscTransakcji;
END;
GO

EXECUTE StatystykiKlienta 10, 1
```

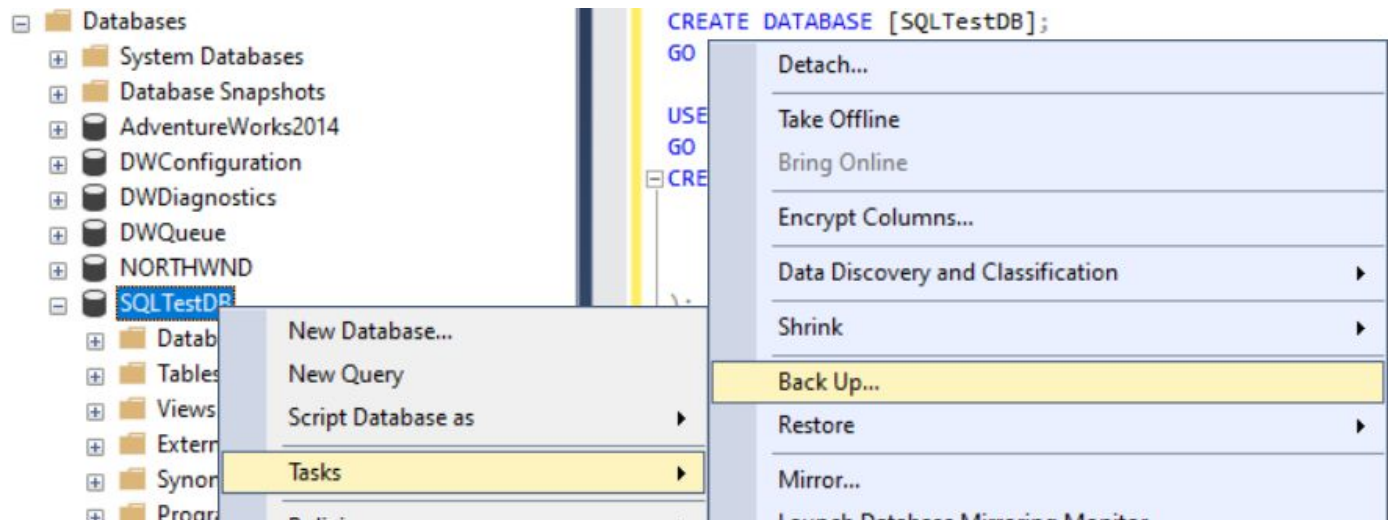


# Przechowywanie danych z bazy Integracja z innymi systemami



# Bezpieczeństwo danych przy operacjach DML

Backup - zapis aktualnego stanu bazy (wszystkich /wybranych obiektów i danych)



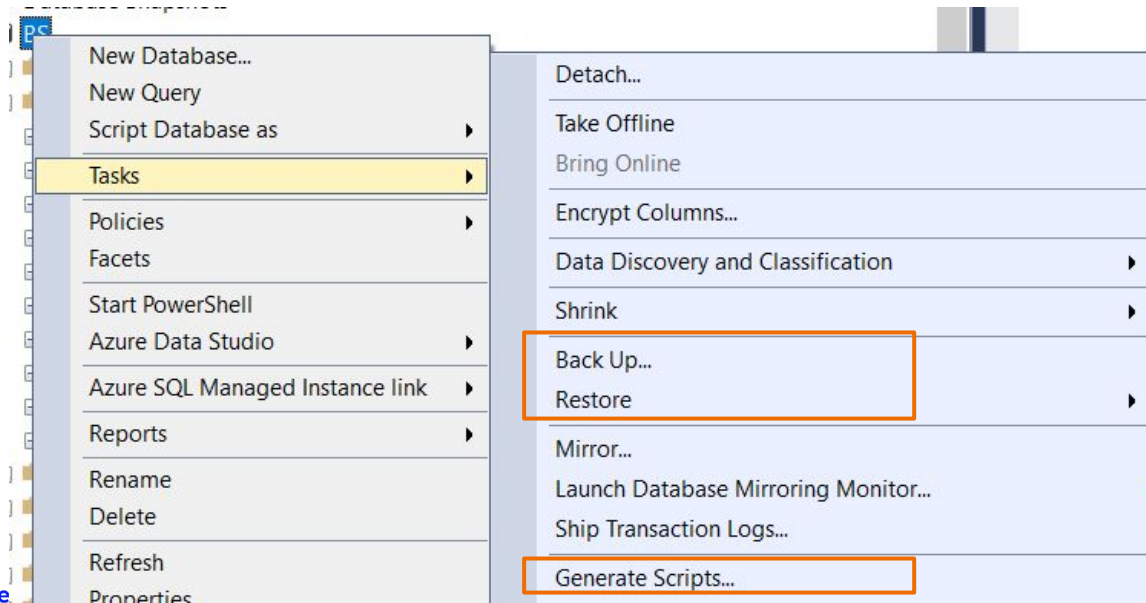
Używanie transakcji, które potem można zatwierdzić (COMMIT), bądź cofnąć (ROLLBACK)

```
BEGIN TRANSACTION
-- operacje zmieniające dane/obiekty
COMMIT TRANSACTION -- zatwierdzenie
-- lub wycofanie
ROLLBACK TRANSACTION
```

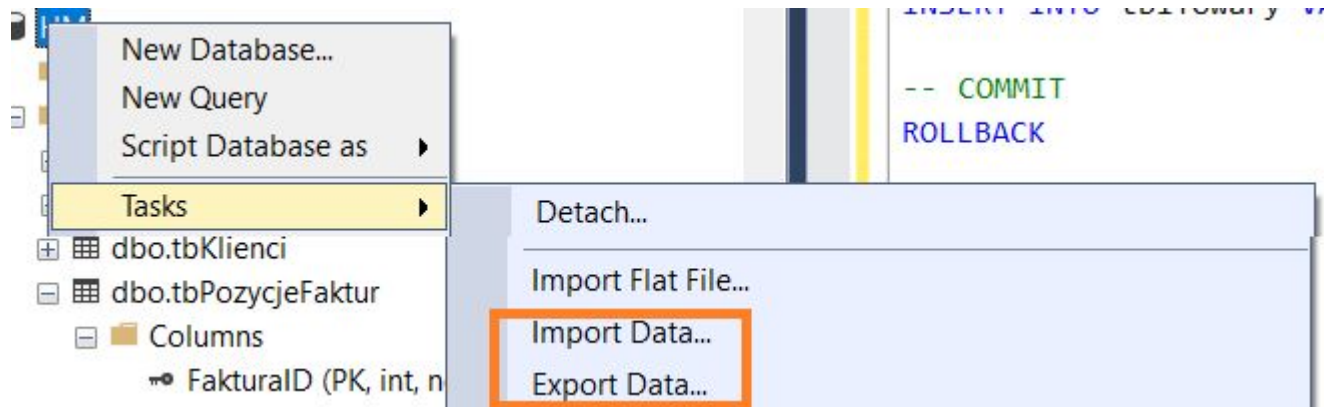
# Kopie zapasowe, Zapis obiektów z bazy do skryptu

Tworzenie Kopii zapasowych danych umożliwia nam Odtworzenie struktury bazy i danych z momentu stworzenia backupu.

Obiekty z bazy jak i dane (wybrane) możemy także zapisać w formie skryptu .sql Z operacjami SQL DDL oraz SQL DML (polecenia tworzące obiekty i wstawiające dane)



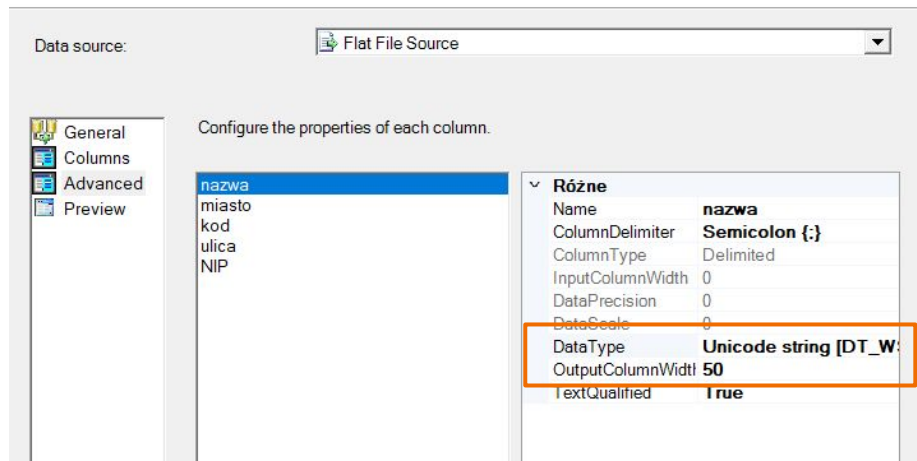
# Import/Export danych do SQL Server (Import And Export Wizard)



Umożliwia import i export danych z Excela oraz między bazami przez Microsoft OLE DB Provider (dawniej Microsoft native client 11). Określamy źródło oraz miejsce docelowe danych, dane jakie chcemy przekazać oraz mapowanie.

# Import plików płaskich (.csv, .txt)

- by dodać do istniejącej tabeli używaj import data -> flat file. Opcja flat file pozwala tylko na dodanie danych do nowej tabeli
- ułatwimy sobie zadanie wstawiając w pierwszej linii plikku csv nazwy docelowych kolumn
- w zaawansowanych opcjach importu odpowiednio uzupełnij długości kolumn oraz typy. Ważne -> typ string [DT\_STR] koresponduje z varchar w SQL, typ [DT\_WSTR] z nvarchar.
- Jeśli na końcu pliku znajdują się puste linie zostaną one zinterpretowane jako kolejne wiersze do wstawienia
- Jeśli operacja zakończy się błędem przeczytaj uważnie wiadomość opisującą problem



# Połączenie z Excel do bazy danych SQL Server przez Power Query

Dane -> pobierz dane -> Z bazy danych SQL Server. Ustawiamy dane z bazy oraz co chcemy importować - dane z tabeli, widoku, bądź pobrane za pomocą Zapytania Select SQL

## Baza danych programu SQL Server

Serwer ⓘ

DESKTOP-C3IH1IF\SQLEXPRESS

Baza danych (opcjonalna)

HM

⌵ Opcje zaawansowane

Limit czasu polecenia w minutach (opcjonalnie)

Instrukcja języka SQL (opcjonalna, wymaga bazy danych)

```
SELECT ID_Klient,  
       KodKlienta,  
       NazwaFirmy,  
       Telefon,  
FROM HM.dbo.tblKlienci
```

# Narzędzia w Power Query

Power Query umożliwia zaawansowane przekształcanie danych. Wszystkie operacje zostaną zapisane w liście zastosowanych kroków, co w łatwy sposób pozwala wycofywać i edytować wykonane operacje.

The screenshot displays the Power Query interface. The ribbon at the top includes tabs: Plik, Narzędzia główne, Przekształć, Dodaj kolumnę, and Widok. The 'Przekształć' tab is active, showing various transformation options like 'Transponuj', 'Odwroc wiersze', 'Zlicz wiersze', 'Typ danych: Tekst', 'Zamienianie wartości', 'Anuluj przestawienie kolumn', 'Wydziel kolumny', 'Format', 'Scal kolumny', 'Statystyka', 'Trygonometryczne', 'Data', 'Godzina', 'Czas trwania', and 'Kolumna strukturalna'.

The main area shows a table with the following data:

	A <sup>B</sup> NazwaFirmy	A <sup>B</sup> Telefon	I <sup>2</sup> AK	I <sup>2</sup> AL	I <sup>2</sup> BR	I <sup>2</sup>
1	ALLADYN SP Z o.o.	228906754		null	2	null
2	ALOJZY KOWALCZUK	445568907		1	null	null
3	CHACIŃSKA GABRIELA	123469811		null	null	null
4	CHACIŃSKI RYSZARD	223335609		null	null	null
5	DAMIS	123456768		null	null	null
6	ELENA-PRES	443434545		null	null	null
7	Firma S.A	23 454545		null	null	null
8	IRLANDZKIE ROWERY AUTOMATYCZNE S.A	143333333		null	null	null
9	KRAK&FIK KANCELARIA PRAWNICZA	558-22-78		null	null	null
10	LEKSUX S.A.	45 777788		null	null	null
11	LIF-TEX LIPIŃSKI FRANCISZEK	456678888		null	null	null

The formula bar shows the following formula:

```
= Table.TransformColumns("#Kolumna przestawna",{{"Telefon", Text.Lower, type text}})
```

The 'Ustawienia zapytania' pane on the right shows the 'Zastosowane kroki' (Applied Steps) list, which includes:

- Źródło
- Zamieniono wartość
- Kolumna przestawna
- × Tekst pisany małymi literami**

# Użyteczne linki

<https://learn.microsoft.com/en-us/sql/t-sql/functions/functions?view=sql-server-ver16> - dokumentacja; funkcje wbudowane

<https://blog.sqlauthority.com/category/sql-tips-and-tricks/> - ciekawy blog z mnustwem porad w zakresie MSSQL

<https://learn.microsoft.com/en-us/sql/samples/sql-samples-where-are?view=sql-server-ver16> Treningowe bazy danych na SQL Server od Microsoftu

<https://www.sqlops.com/find-missing-indexes/> - analiza indeksów które warto dodać do bazy

<https://eitanblumin.com/2018/10/28/the-loop-hash-and-merge-join-types/>