NINJA DATA

SKŁADNIA: SQL SERVER (T-SQL)

SQL Cheat Sheet Mistrz

CTE, funkcje okna i wzorce, o które pytają na rozmowach rekrutacyjnych.

ŚCIĄGA | NOWICJUSZ | WOJOWNIK | MISTRZ

01 CTE
WITH, czyli zapytanie krok po kroku

- **WITH**: nazwany wynik pośredni. Czytasz od góry do dołu jak przepis.

```
WITH Sprzedaz AS (
  SELECT ProduktID, SUM(Ilosc) AS Sztuki
  FROM Zamowienia
  GROUP BY ProduktID
),
Hity AS (
  SELECT * FROM Sprzedaz WHERE Sztuki > 1000
)
SELECT p.Nazwa, h.Sztuki
FROM Hity h
JOIN Produkty p ON p.ID = h.ProduktID;
```

- **CTE rekurencyjne**: hierarchie (szef i podwładni) albo ciąg dat.

```
WITH Dni AS (
  SELECT CAST('2026-01-01' AS date) AS Dzien
  UNION ALL
  SELECT DATEADD(DAY, 1, Dzien) FROM Dni
  WHERE Dzien < '2026-12-31'
)
SELECT Dzien FROM Dni OPTION (MAXRECURSION 400);
```

02 WZORCE
Zadania z rekrutacji

- **Top N w każdej grupie**: ROW_NUMBER w CTE, potem filtr.

```
WITH r AS (
  SELECT *, ROW_NUMBER() OVER (
    PARTITION BY Kategoria
    ORDER BY Sprzedaz DESC) AS nr
  FROM Produkty
)
SELECT * FROM r WHERE nr <= 3;
```

- **Duplikaty**: zostawiasz najnowszy wiersz każdego klienta.

```
WITH d AS (
  SELECT *, ROW_NUMBER() OVER (
    PARTITION BY Email
    ORDER BY DataDodania DESC) AS nr
  FROM Klienci
)
DELETE FROM d WHERE nr > 1;
```

WSKAZÓWKA Przed DELETE uruchom to samo z SELECT zamiast DELETE i zobacz, co zniknie.

03 FUNKCJE OKNA
OVER (PARTITION BY ... ORDER BY ...)

- **Okno a GROUP BY**: wiersze zostają, a obok każdego masz wynik dla jego grupy.

```
SELECT Nazwa, Kategoria, Sprzedaz,
  SUM(Sprzedaz) OVER (PARTITION BY Kategoria)
    AS SumaKat
FROM Produkty;
```

- **ROW_NUMBER, RANK, DENSE_RANK**: numer bez remisów, ranking z dziurą po remisie (1, 1, 3), ranking bez dziur (1, 1, 2).

```
SELECT Nazwa,
  RANK() OVER (ORDER BY Sprzedaz DESC) AS Miejsce
FROM Produkty;
```

- **NTILE**: dzieli wiersze na N równych grup, np. kwartyle klientów.

```
NTILE(4) OVER (ORDER BY Wartosc DESC) AS Kwartyl
```

- **Suma narastająca**: kolejność plus ramka okna.

```
SUM(Wartosc) OVER (ORDER BY Data
  ROWS BETWEEN UNBOUNDED PRECEDING
       AND CURRENT ROW)
```

- **Średnia krocząca 7 dni**: bieżący wiersz i 6 poprzednich.

```
AVG(Wartosc) OVER (ORDER BY Data
  ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)
```

- **LAG, LEAD**: wartość z poprzedniego albo następnego wiersza, np. zmiana miesiąc do miesiąca.

```
SELECT Miesiac, Sprzedaz,
  Sprzedaz - LAG(Sprzedaz) OVER (ORDER BY Miesiac)
    AS Zmiana
FROM SprzedazMies;
```

- **FIRST_VALUE**: pierwsza wartość w oknie, np. pierwszy produkt klienta.

```
FIRST_VALUE(Produkt) OVER (
  PARTITION BY KlientID ORDER BY Data)
```

- **Udział w całości**: wiersz podzielony przez sumę okna.

```
Sprzedaz * 1.0 / SUM(Sprzedaz) OVER () AS Udzial
```

04 EXISTS
Czy coś istnieje

- **EXISTS**: klienci, którzy złożyli co najmniej jedno zamówienie.

```
SELECT k.Nazwa FROM Klienci k
WHERE EXISTS (SELECT 1 FROM Zamowienia z
              WHERE z.KlientID = k.ID);
```

- **NOT EXISTS**: klienci bez zamówień. Bezpieczniejsze niż NOT IN, gdy w kolumnie są NULL.

```
WHERE NOT EXISTS (SELECT 1 FROM Zamowienia z
                  WHERE z.KlientID = k.ID)
```

- **Podzapytanie skorelowane**: liczone dla każdego wiersza osobno.

```
SELECT Nazwa, Cena FROM Produkty p1
WHERE Cena > (SELECT AVG(Cena) FROM Produkty p2
              WHERE p2.Kategoria = p1.Kategoria);
```

- **CROSS APPLY**: dla każdego wiersza wynik innego zapytania, np. 3 ostatnie zamówienia klienta.

```
SELECT k.Nazwa, o.Data
FROM Klienci k
CROSS APPLY (SELECT TOP 3 Data FROM Zamowienia z
             WHERE z.KlientID = k.ID
             ORDER BY Data DESC) o;
```

05 GRUPOWANIE+
Sumy częściowe

- **ROLLUP**: sumy dla każdego poziomu i suma całkowita.

```
SELECT Region, Miasto, SUM(Sprzedaz)
FROM Sklepy
GROUP BY ROLLUP (Region, Miasto);
```

- **CUBE**: wszystkie kombinacje kolumn. GROUPING SETS: tylko te, które wymienisz.

```
GROUP BY GROUPING SETS ((Region), (Kategoria), ())
```

- **GROUPING**: 1 w wierszu sumy, przydaje się do etykiety „Razem”.

```
IIF(GROUPING(Miasto) = 1, 'Razem', Miasto)
```

06 PIVOT
Wiersze na kolumny

- **PIVOT**: regiony jako kolumny.

```
SELECT * FROM (
  SELECT Produkt, Region, Sprzedaz FROM Sprzedaz
) AS t
PIVOT (SUM(Sprzedaz)
  FOR Region IN ([Polska], [Niemcy], [Francja])
) AS p;
```

- **Ręczny PIVOT**: CASE w SUM, działa w każdej bazie.

```
SELECT Produkt,
  SUM(CASE WHEN Region = 'Polska'
           THEN Sprzedaz END) AS Polska,
  SUM(CASE WHEN Region = 'Niemcy'
           THEN Sprzedaz END) AS Niemcy
FROM Sprzedaz GROUP BY Produkt;
```

- **UNPIVOT**: kolumny z powrotem na wiersze.

```
SELECT Produkt, Region, Sprzedaz
FROM Tabela
UNPIVOT (Sprzedaz FOR Region
  IN ([Polska], [Niemcy], [Francja])) AS u;
```

07 WYDAJNOŚĆ
Szybsze zapytania

- **Filtr bez funkcji na kolumnie**: indeks działa tylko wtedy.

```
-- wolno:  WHERE YEAR(Data) = 2026
-- szybko:
WHERE Data >= '2026-01-01'
  AND Data <  '2027-01-01'
```

- **Tylko potrzebne kolumny**: SELECT * na dużej tabeli czyta wszystko.
- **Plan wykonania**: w SSMS Ctrl+L przed uruchomieniem pokazuje, gdzie baza traci czas.
- **Najpierw zawężasz, potem łączysz**: filtr i agregacja w CTE przed JOIN zmniejszają liczbę wierszy.

Chcesz mieć te komendy w głowie, a nie na kartce? **Fiszki SQL Podstawy** sklep.ninjadata.pl/produkt/sql-podstawy/