1. Konstrukcje z użyciem CASE.

Instrukcja CASE jest instrukcją warunkową. Ma zastosowanie SQL w miejscach wymagających sprawdzenie wartości wyrażenia, czy spełnia to wrażenie określone warunki. Konstrukcje takie spotykane są w wielu językach programowania :

if (..) {.. } else {..}
IF .. THEN .. ELSE .. END
switch (..) { case ..: ..break; case ..: break;}

Można ją użyć po słowie SELECT, ale także w wyrażeniach po WHERE i HAVING lub w klauzuli GROUP BY i ORDER BY.
Konstrukcja polecenia CASE występuje w dwóch postaciach. W pierwszej porównujemy wartość lub zawartość kolumny z wartością po słowie THEN, w drugim przypadku wyznaczamy wartość logiczną wyrażenia. Poniżej obydwie formy instrukcji CASE.

Porównanie z wartością danej kolumny.

```
SELECT a1, a2,
  CASE wartość_lub_kolumna
    WHEN wartosc_1 THEN wynik_1
    WHEN wartosc_2 THEN wynik_2
    WHEN wartosc_3 THEN wynik_3
    [ ELSE wynik_gdy_brak_na_liscie ]
  END
FROM nazwa_tabeli
```

Porównanie z wyznaczoną wartością logiczną.

```
SELECT a1, a2,
  CASE
    WHEN wyrazenie_logiczne_1 THEN wynik_1
    WHEN wyrazenie_logiczne_2 THEN wynik_2
    WHEN wyrazenie_logiczne_3 THEN wynik_3
    [ ELSE wynik_gdy_brak_na_liscie ]
  END
FROM nazwa_tabeli
```

Przykłady z laboratorium.

```
SELECT imie, nazwisko,
  CASE
    WHEN akt.punkty >= 50 and akt.punkty < 60 THEN 'dostateczny'
    WHEN akt.punkty >= 60 and akt.punkty < 70 THEN 'dobry'
    WHEN akt.punkty >= 90 THEN 'bardzo dobry'
    ELSE 'brak oceny'
  END
FROM lab04.uczestnik u JOIN lab04.uczest_kurs uk ON
u.id_uczestnik=uk.id_uczest
JOIN lab04.aktywnosc akt ON akt.id_uczestnik = uk.id_uczest
WHERE uk.id_kurs=1
ORDER BY 2,1 ;
```

```
SELECT imie, nazwisko, punkty,
  CASE punkty
    WHEN 98 THEN 'premia 1'
    WHEN 96 THEN 'premia 2'
    WHEN 94 THEN 'premia 3'
    ELSE 'brak premii'
  END
FROM lab04.uczestnik u JOIN lab04.uczest_kurs uk ON
u.id_uczestnik=uk.id_uczest
JOIN lab04.aktywnosc oc ON oc.id_uczestnik = uk.id_uczest
WHERE uk.id_kurs IN (1,2)
ORDER BY 3 DESC, 2,1 ;
```

2. Wyrażenie CTE (Common Table Expression).

Wyrażenia CTE (wspólne wyrażenia tablicowe) upraszczają i zwiększają przejrzystość kodu polecenia SQL. Raz zdefiniowane struktury w ramach polecenia WITH można wykorzystać wiele razy. Wyrażenia CTE można wykorzystać również w ramach widoku, funkcji czy procedury składowanej. Realizacja wyrażeń CTE zaczyna się od słowa kluczowego WITH po którym następuje deklaracja tabel CTE. Widoczność zdefiniowanych tabel CTE jest w ramach wyrażenia WITH oraz zapytania związanego z wyrażeniem. Wyrażenia CTE są szczególnie przydatne w przypadku rozbudowanych zapytań, łączących wiele tabel, które chcemy użyć w kolejnym kroku, wykonując na nich dodatkowe operacje. Poniżej struktura wyrażenia CTE.

```
WITH table1_CTE [( atrybuty )] AS ( definicja zapytania )   -- definicja tabeli CTE
    [ , table2_CTE, ... ]                                   -- definicje kolejnych tabel CTE
SELECT [ atrybuty ] FROM tables_CTE ;
```

Przykłady z laboratorium.

```
-- Proste demo.
WITH myCTE AS ( select nazwisko, imie, email from lab04.uczestnik )
SELECT * FROM myCTE;

-- Lista kursów z opłatami.

WITH sumCTE AS ( select id_kurs, sum(oplata) as suma from lab04.uczest_kurs group
by id_kurs )
SELECT ko.opis, suma from sumCTE JOIN lab04.kurs k USING ( id_kurs)
        JOIN lab04.kurs_opis ko ON k.id_nazwa = ko.id_kurs;


-- Procentowy udział osób w poszczególnych kursach.

WITH totalCTE as ( select count(*)::float as tot from lab04.uczest_kurs ),

  kursCTE as ( select id_kurs, count(*)::float as num from lab04.uczest_kurs group by
id_kurs )

SELECT ko.opis, ((kCTE.num/(totalCTE.tot))* 100)::decimal(5,1)

FROM totalCTE, lab04.kurs k JOIN kursCTE kCTE USING (id_kurs)

    JOIN lab04.kurs_opis ko ON k.id_nazwa=k.id_kurs;
```

3. Rekurencyjne wyrażenie CTE.

Interesującą właściwością wspólnych wyrażeń tablicowych jest możliwość stosowania rekurencji w ich wnętrzu. Tego typu funkcjonalność, wykorzystujemy w np. zbiorach z określoną hierarchią elementów.

Definicja struktury wyrażeń rekurencyjnych WITH składa się z trzech elementów:

- Określenia zapytania zakotwiczającego, jest to zazwyczaj zbiór elementów stanowiących korzeń (lub korzenie).
- Zapytania rekursywnego – skorelowanego z wynikiem zwracanym przez zapytanie poprzednie. Odwołujemy się tu do struktury hierarchicznej. Operator UNION ALL łączy wszystkie przebiegi w finalny zbiór wynikowy. Ważne jest, aby zrozumieć, że w każdym kroku działamy tylko na zbiorze zwracanym przez krok poprzedni.
- Niejawnego warunku zakończenia rekurencji. Jeśli zapytanie rekurencyjne, skorelowane, nie zwróci żadnego elementu, działanie CTE zostaje przerwane.

Składnia.

```
WITH RECURSIVE cte_name (
    CTE_definicja_zapytania     -- część nierekursywna ( zapytanie
zakotwiczające )
    UNION [ALL]
    CTE_definicja_zapytania     -- cześć rekursywna ( zapytanie rekursywne,
                                -- skorelowane z wynikiem poprzedniego
zapytania )
) SELECT * FROM cte_name;
```

Przykłady z laboratorium.

-- proste demo

```
WITH RECURSIVE nieparzyste(n) AS ( VALUES(1)
            UNION
            SELECT 2+n FROM nieparzyste WHERE n < 10 )
SELECT * FROM nieparzyste ORDER BY n;
```

-- silnia

```
WITH RECURSIVE Silnia (n, silnia) AS (
   SELECT 1, CAST(1 AS BIGINT)
   UNION ALL
   SELECT n + 1, (n + 1) * silnia
   FROM Silnia WHERE n<20)
SELECT n, silnia FROM Silnia;
```

```
-- hierarchia

-- Tworzymy tablicę pracownik zawierającą id pracownika,
-- nazwisko i id bezpośredniego szefa
-- Tabela zawiera połączenie typu self-join
CREATE TABLE emp ( empno INT, empname VARCHAR(20), mgrno INT ) ;
-- Wprowadzamy przykładowe dane
INSERT INTO emp VALUES ( 100, 'Kowalski',    null),
                       ( 101, 'Abacki',      100),
                       ( 102, 'Cabacki',     101),
                       ( 103, 'Dadacki',     102),
                       ( 104, 'Zazadzki',    101),
                       ( 105, 'Stachera',    104),
                       ( 106, 'Flisikowski', 100),
                       ( 107, 'Olech',       106),
                       ( 108, 'Płochocki',   106),
                       ( 109, 'Stachyra',    107),
                       ( 110, 'Sztuka',      109),
                       ( 111, 'Sosin',       110),
                       ( 112, 'Głowala',     110),
                       ( 113, 'Straszewski', 110),
                       ( 114, 'Dwojak',      100),
                       ( 115, 'Kotulski',    114),
                       ( 116, 'Łaski',       115),
                       ( 117, 'Iwanowicz',   115)  ;

-- Zapytanie zwracające nazwisko pracownika i jego przełożonego
SELECT e.empno, e.empname, e.mgrno, m.empname
FROM emp e JOIN emp m ON e.mgrno = m.empno;
```

```
-- Zapytanie zwracające nazwisko pracownika,
-- nazwisko bezpośredniego przełożonego i poziom w hierarchii
WITH RECURSIVE cte
      AS ( SELECT empno, empname, mgrno, ''::varchar(20) as mgrname, 1 lvl from emp
where mgrno is null
           UNION ALL
           SELECT e.empno, e.empname, e.mgrno, c.empname, c.lvl+1
           FROM emp e inner join cte c on e.mgrno = c.empno
           WHERE e.mgrno is not null )
SELECT empname, mgrname, lvl FROM cte ORDER BY lvl;


-- Zapytanie zwracające nazwisko pracownika,
-- poziom w hierarchii i listę przełożonych

WITH RECURSIVE cte
      AS ( SELECT empno, empname, mgrno, 1 lvl, ''::text as path from emp where mgrno is
null
           UNION ALL
           SELECT e.empno, e.empname, e.mgrno, c.lvl+1, concat( c.path, '->', c.empname )
           FROM emp e inner join cte c on e.mgrno = c.empno
           WHERE e.mgrno is not null )
SELECT empname, lvl, path FROM cte ORDER BY lvl;
```