1. Operatory na zbiorach - UNION, INTERSECT, EXCEPT.

Do operacji na zbiorach będących wynikiem poleceń SELECT możemy użyć operatorów UNION, UNION ALL, EXCEPT, INTERSECT.
UNION tworzy sumę zbiorów - elementy nie powtarzają się,
UNION ALL tworzy sumę zbiorów ,
EXCEPT - zwraca różnicę,
INTERSECT - daje w wyniku część wspólną.
Warunkami poprawności jest zgodność list kolumn zwracanych przez polecenia SELECT co do ilości, kolejności i typu kolumny.

select_statement UNION [ALL | DISTINCT] **select_statement**

select_statement INTERSECT [ALL | DISTINCT] **select_statement**

select_statement EXCEPT [ALL | DISTINCT] **select_statement**

Przykłady z laboratorium.

Operator UNION.

```
SELECT 'W' AS "Typ", nazwisko, imie, nazwa FROM lab04.wykladowca W
JOIN lab04.uczestnik_organizacja UO USING ( id_wykladowca )
JOIN lab04.organizacja O USING (id_organizacja)
WHERE O.strona_www = 'www.agh.edu.pl'
UNION
SELECT 'U' AS "Typ", nazwisko, imie, nazwa FROM lab04.uczestnik W
JOIN lab04.uczestnik_organizacja UO USING ( id_uczestnik )
JOIN lab04.organizacja O USING (id_organizacja)
WHERE O.strona_www = 'www.agh.edu.pl';
```

Operator INTERSECT.

```
SELECT nazwisko, imie FROM lab04.wykladowca W
JOIN lab04.uczestnik_organizacja UO USING ( id_wykladowca )
JOIN lab04.organizacja O USING (id_organizacja)
WHERE O.strona_www = 'www.uj.edu.pl'
INTERSECT
SELECT nazwisko, imie FROM lab04.wykladowca W
JOIN lab04.uczestnik_organizacja UO USING ( id_wykladowca )
JOIN lab04.organizacja O USING (id_organizacja)
WHERE O.strona_www = 'www.agh.edu.pl';
```

Operator EXCEPT.

```
SELECT nazwisko, imie FROM lab04.wykladowca W
JOIN lab04.uczestnik_organizacja UO USING ( id_wykladowca )
JOIN lab04.organizacja O USING (id_organizacja)
WHERE O.strona_www = 'www.uj.edu.pl'
EXCEPT
SELECT nazwisko, imie FROM lab04.wykladowca W
JOIN lab04.uczestnik_organizacja UO USING ( id_wykladowca )
JOIN lab04.organizacja O USING (id_organizacja)
WHERE O.strona_www = 'www.agh.edu.pl';
```

2. Podzapytania.

Termin **podzapytanie** używamy do określenia kompletnego polecenia SELECT otoczonego nawiasami i zazwyczaj nazywanego aliasem za pomocą klauzuli AS poza nawiasami. Podzapytania możemy użyć w innym poleceniu SELECT, UPDATE, INSERT czy DELETE. W zależności od zwróconego wyniku podzapytania można go wykorzystać w różnych miejscach podstawowego zapytania.
Podzapytanie tabelaryczne można użyć tam, gdzie można użyć nazwy tabeli lub widoku ale też w procedurze przechowywanej lub funkcji zwracającej tabelę.
Podzapytanie tabelaryczne z jedną kolumną może użyć tam, gdzie używamy podzapytanie tabelaryczne lub jako lista wartości w predykacie IN.
Podzapytanie skalarne może być użyć tam, gdzie używamy nazwy kolumny, wyrażenia lub nazwy kolumn.
Kolejnym formalizmem związanym z podzapytaniami są podzapytania skorelowane i nieskorelowane.
Podzapytanie skorelowane jest wtedy, gdy jakiś warunek wewnątrz podzapytania (w klauzuli WHERE lub HAVING) zależy od wartości w rekordzie aktualnie przetwarzanym przez zapytanie zewnętrzne.
Podzapytanie nieskorelowane nie jest zależne od zewnętrznej wartości, może być uruchomione jako odrębne zapytanie.
Podzapytania można też rozpatrywać ze względu na miejsce wykorzystania.
Podzapytanie w miejscu listy tabel, określa źródło danych.
Podzapytanie w miejscu warunku, staje się częścią kryteriów selekcji zarówno dla klauzuli WHERE jak i HAVING.
Podzapytanie w miejscu listy atrybutów tworzy najczęściej wartość obliczaną.

W podzapytaniach wykorzystywane są predykaty EXISTS, ALL i ANY (SOME)

Przykłady z laboratorium.

```
SELECT U.nazwisko, U.imie, SuQu.id, SuQu.wynik FROM
(SELECT id_uczest id, sum(oplata) wynik
FROM lab04.uczest_kurs GROUP BY id_uczest) SuQu
JOIN lab04.uczestnik U ON suqu.id = U.id_uczestnik
ORDER BY suqu.wynik DESC;
```

```
SELECT nazwisko, imie FROM lab04.uczestnik WHERE id_uczestnik IN
( SELECT id_uczestnik FROM lab04.uczestnik_organizacja WHERE id_organizacja =
1 )
ORDER BY 1,2;


SELECT (SELECT SUM(oplata) FROM lab04.uczest_kurs) razem,
 (SELECT COUNT(oplata) FROM lab04.uczest_kurs) ilosc,
 (SELECT AVG(oplata) FROM lab04.uczest_kurs) srednia;
```

przykład podzapytania skorelowanego

```
SELECT KO.opis,
( SELECT COUNT(*) FROM lab04.wykladowca W
JOIN lab04.wykl_kurs WK ON wk.id_wykl = W.id_wykladowca
  WHERE WK.id_kurs=KU.id_kurs ) AS ilosc_wykladowcy
FROM lab04.kurs KU JOIN lab04.kurs_opis KO ON KU.id_nazwa = KO.id_kurs;
```

Operator ALL

```
SELECT UO.id_organizacja FROM lab04.uczestnik UC
JOIN lab04.uczestnik_organizacja UO USING(id_uczestnik)
WHERE UO.id_organizacja
=ALL (SELECT id_organizacja  FROM lab04.organizacja O WHERE
UO.id_organizacja = O.id_organizacja);
```

Operator ANY

```
SELECT u.imie, u.nazwisko FROM uczestnik u
WHERE u.id_uczestnik = ANY ( SELECT id_uczest FROM uczest_kurs )
ORDER BY 2 ;
```

Operator EXISTS i NOT EXISTS

```
SELECT ko.opis FROM lab04.kurs KU
JOIN lab04.kurs_opis KO ON KU.id_nazwa = ko.id_kurs
WHERE EXISTS ( SELECT 1 FROM lab04.wykl_kurs WK WHERE wk.id_kurs =
ku.id_kurs );

SELECT ko.opis FROM lab04.kurs KU
JOIN lab04.kurs_opis KO ON KU.id_nazwa = ko.id_kurs
WHERE NOT EXISTS ( SELECT 1 FROM lab04.wykl_kurs WK WHERE wk.id_kurs
= ku.id_kurs );
```

3. Perspektywa (widok).

Perspektywa powstaje w wyniku polecenia SELECT dając w rezultacie nową dynamiczną tabelę wynikową. Może być tabelą wirtualną, standardowo nie istnieje fizyczna postać (nie zajmuje miejsca) w bazie danych. Istnieją także widoki zmaterializowane. Użycie perspektywy powoduje każdorazowo wykonanie polecenia SELECT będącego definicją widoku. Dla polecenia SELECT użycie widoku jest identyczne jak zwykłej tabeli. Widoki tworzymy w następujących sytuacjach:

pozwalają uprościć złożone zapytania - brak konieczności pisania długich poleceń SELECT,
umożliwiają wielokrotne użycia zdefiniowanego zapytania,
umożliwiają zmianę formatowania danych, np. wykorzystanie funkcji CAST,
ukrywanie efektów normalizacji - łączenie tabel z wykorzystaniem JOIN,
możemy tworzyć kolumny obliczeniowe,
ograniczenie dostępu do danych - usuwanie poprzez selekcję i projekcję chronionych danych,
tworzenie warstwy abstrakcji - przykrywanie oryginalnej struktury bazy danych, zmiana nazw.

Składnia poleceń z dokumentacji Postgresql'a.

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
```

```
CREATE VIEW name [ ( column_name [, ...] ) ]
  AS query
  [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Polecenie tworzące perspektywę może posiadać dodatkowe parametry istotne dla widoków, które mogą modyfikować dane w tabelach tworzących widok.

WITH CHECK OPTION
WITH LOCAL CHECK OPTION
WITH CASCADED CHECK OTION

Pewne perspektywy pozwalają na operacje INSERT i UPDATE. Takie perspektywy nazywa się modyfikowalnymi. Modyfikowalność zakłada przeniesienie każdego każdej modyfikacji z perspektywy do tabeli źródłowej (bazowej). Standard ISO narzuca na takie obiekty pewne wymagania. W definicji widoku:

nie występuje opcja DISTINCT,
każdy atrybut w poleceniu definiującym widok jest atrybutem z tabeli źródłowej (nie ma stałych, wyrażeń, czy funkcji agregujących,
w klauzuli FROM określona jest tylko jedna tabela źródłowa (wykluczone są złączenia, sumy, przecięcia czy różnica),
klauzula WHERE nie zawiera żądnego podzapytania,
nie występuję klauzula GROUP BY i HAVING.

Przykład z laboratorium.

```
CREATE VIEW vw_kurs_wykladowca (opis, nazwisko, imie)
AS
SELECT
ko.opis, wy.nazwisko, wy.imie FROM kurs KU JOIN kurs_opis KO ON KU.id_kurs = KO.id_kurs
JOIN wykl_kurs WK ON WK.id_kurs = ku.id_kurs AND WK.id_grupa = ku.id_grupa
JOIN wykladowca WY ON wy.id_wykladowca = wk.id_wykl;
```