1. SELECT

z dokumentacji

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    [ * | expression [ [ AS ] output_name ] [, ...] ]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY grouping_element [, ...] ]
    [ HAVING condition ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ]
[, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } ONLY ]
    [ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [, ...] ] [
NOWAIT | SKIP LOCKED ] [...] ]
```

następnie opisano każdy element składni.

Dla naszych celów ograniczymy składnię do postaci:

SELECT [DISTINCT] lista nazw kolumn, wyrażeń, funkcji oddzielonych przecinkami

FROM nazwy tabel lub widoków oddzielonych przecinkami

WHERE wyrażenie filtrujące - wybierające krotki

GROUP BY lista nazw kolumn

HAVING warunek wybieranych wierszy po grupowaniu

[UNION, INTERSECT, EXCEPT] operacje na zbiorach wynikowych zapytań

ORDER BY lista nazw kolumn lub lista pozycji kolumn [ASC | DESC]

Klauzula SELECT wskazuje listę kolumn, które zostaną zwrócone w tabeli wynikowej.

Klauzula FROM jest odpowiedzialna za wskazanie tabel z których będą pobierane dane.

Klauzula WHERE określa warunki, jakie mają być spełnione, aby rekord (krotka) pojawiła się w tabeli wynikowej.

Klauzula GROUP BY grupuje wiersze tablicy wynikowej według identycznych wartości w kolumnach o podanych nazwach.

Klauzula HAVING określa warunki dla rekordów uzyskanych w wyniku przetworzenia klauzulą GROUP BY.

Klauzula ORDER BY określa sposób uporządkowania tabeli wynikowej.

dokumentacja Postgresql - polecenie select
https://www.postgresql.org/docs/9.5/sql-select.html

2. Złączenia - użycie operatorów WHERE i JOIN .

Iloczyn kartezjański tabel (relacji) to zbiór krotek (rekordów) powstających z złączenia wszystkich krotek, kombinacja każdej z każdą. Jest to złączenie krzyżowe.

```
SELECT lista_nazw_kolumn FROM nazwa_tabeli1, nazwa_tabeli2 ...;
```

Niezbyt często zbiór będący wynikiem takiego polecenia spełnia nasze potrzeby.

Przykład z laboratorium.

```
SELECT  opis, nazwisko, imie FROM kurs, kurs_opis, wykl_kurs, wykladowca;
```

```
SELECT * FROM uczestnik;
```

Jeżeli między tabelami istnieją związki, które możemy wykorzystać do odrzucenia zbędnych lub nadmiarowych krotek to możemy wykorzystać ten fakt i zastosować inną składnię polecenia SELECT. Związkami, które wykorzystujemy, a wręcz tworzymy w tym celu są powiązania typu klucz kandydujący - klucz obcy. Tworzenie takiego powiązania np.

```
ALTER TABLE "DB1lab01"."lab04"."uczest_kurse" ADD CONSTRAINT
uczest_kurs_id_kurs_fkey FOREIGN KEY (id_kurs)
   REFERENCES lab04.kurs(id_kurs)
   ON DELETE NO ACTION
   ON UPDATE NO ACTION
   NOT DEFERRABLE,
  CONSTRAINT uczest_kurs_id_uczest_fkey FOREIGN KEY (id_uczest)
   REFERENCES lab04.uczestnik(id_uczestnik)
   ON DELETE NO ACTION
   ON UPDATE NO ACTION
   NOT DEFERRABLE
```

nie jest niezbędne do odrzucenia niepożądanych krotek, jest jednak bardzo pożyteczne, pozwala na utrzymanie w bazie danych zweryfikowanych danych na etapie ich modyfikacji, w rezultacie polecenie SELECT nie dostarczy zaskakującego bądź nieoczekiwanego wyniku.

Do filtrowania danych używamy operatora **WHERE**.
Przykłady poleceń:

```
SELECT  opis, data_rozpoczecia, nazwisko, imie, email
FROM lab04.kurs, lab04.kurs_opis, lab04.wykl_kurs, lab04.wykladowca
WHERE kurs.id_nazwa = kurs_opis.id_kurs
AND kurs.id_kurs = wykl_kurs.id_kurs
AND wykladowca.id_wykladowca = wykl_kurs.id_wykl;
```

Do tego celu możemy także użyć składni z operatorem **JOIN.**

Przykłady złączenia tabel (JOIN) .

```
SELECT  opis, data_rozpoczecia, data_zakonczenia, nazwisko, imie, email
FROM lab04.kurs KU
JOIN lab04.kurs_opis KO ON ku.id_nazwa = ko.id_kurs
JOIN lab04.wykl_kurs WK ON wk.id_kurs = ku.id_kurs
JOIN lab04.wykladowca WY ON WY.id_wykladowca = wk.id_wykl;

SELECT  opis, data_rozpoczecia, data_zakonczenia, nazwisko, imie, email
FROM lab04.kurs KU
JOIN lab04.kurs_opis KO ON ku.id_nazwa = ko.id_kurs
LEFT OUTER JOIN lab04.wykl_kurs WK ON wk.id_kurs = ku.id_kurs
LEFT OUTER JOIN lab04.wykladowca WY ON WY.id_wykladowca = wk.id_wykl;

SELECT  opis, data_rozpoczecia, data_zakonczenia, nazwisko, imie, email
FROM lab04.kurs KU
JOIN lab04.kurs_opis KO ON ku.id_nazwa = ko.id_kurs
RIGHT OUTER JOIN lab04.wykl_kurs WK ON wk.id_kurs = ku.id_kurs
RIGHT OUTER JOIN lab04.wykladowca WY ON WY.id_wykladowca = wk.id_wykl;

SELECT  opis, data_rozpoczecia, data_zakonczenia, nazwisko, imie, email
FROM lab04.kurs KU
JOIN lab04.kurs_opis KO ON ku.id_nazwa = ko.id_kurs
FULL OUTER JOIN lab04.wykl_kurs WK ON wk.id_kurs = ku.id_kurs
FULL OUTER JOIN lab04.wykladowca WY ON WY.id_wykladowca = wk.id_wykl;
```

2. AGREGATY - przykłady operatorów

Standardowe funkcje agregujące dostępne w języku SQL.

COUNT() - zwraca liczbę krotek

SUM(nazwa_atrybutu) - zwraca sumę wartości dla zadanego atrybutu dla wybranej grupy krotek

AVG(nazwa_atrybutu) - zwraca średnią arytmetyczną dla zadanego atrybutu nazwa_atrybutu dla wybranej grupy krotek

MIN(nazwa_atrybutu) - zwraca wartość minimalną dla zadanego atrybutu nazwa_atrybutu dla wybranej grupy krotek

MAX(nazwa_atrybutu) - zwraca wartość maksymalną dla zadanego atrybutu nazwa_atrybutu dla wybranej grupy krotek

nazwa_atrybutu jest najczęściej nazwą kolumny, a wybrana grupa krotek jest określona przez operatory **WHERE** i **GROUP BY.**
Końcowy rezultat może być dodatkowo filtrowany przez operator **HAVING**.

Przykłady.

```
SELECT COUNT(*) FROM lab04.uczestnik ;

SELECT COUNT(oplata) FROM lab04.uczest_kurs ;
SELECT SUM(oplata) FROM lab04.uczest_kurs ;
SELECT AVG(oplata) FROM lab04.uczest_kurs ;
SELECT MIN(oplata) FROM lab04.uczest_kurs ;
SELECT MAX(oplata) FROM lab04.uczest_kurs ;

SELECT SUM(oplata)/COUNT(oplata) AS srednia FROM lab04.uczest_kurs ;

SELECT id_kurs, COUNT(*) FROM lab04.uczest_kurs
GROUP BY id_kurs ORDER BY 1 ;

SELECT id_kurs, COUNT(*) FROM lab04.uczest_kurs
GROUP BY id_kurs HAVING COUNT(*) > 6 ORDER BY 1 ;


SELECT w.nazwisko, w.imie, ko.opis
FROM lab04.wykladowca w LEFT JOIN lab04.wykl_kurs wk ON w.id_wykladowca =
wk.id_wykl
        LEFT JOIN lab04.kurs k USING (id_kurs)
        LEFT JOIN lab04.kurs_opis ko ON ko.id_kurs = k.id_nazwa
ORDER BY w.nazwisko ;

SELECT w.nazwisko, w.imie, ko.opis
FROM lab04.wykladowca w RIGHT JOIN lab04.wykl_kurs wk ON w.id_wykladowca =
wk.id_wykl
        RIGHT JOIN lab04.kurs k USING (id_kurs)
        RIGHT JOIN lab04.kurs_opis ko ON ko.id_kurs = k.id_nazwa
ORDER BY w.nazwisko ;
```