

SQL - podstawy

Definiowanie podstawowych
elementów tabel i proste
zapytania

Tworzenie tabel

Nazwa

CREATE TABLE Nazwa(...);

Tworzenie tabel

Nazwa

kolumna1	kolumna2	kolumna3

CREATE TABLE Nazwa(k1,k2,...);

- nazwy kolumn

Tworzenie tabel

Nazwa

kolumna1	kolumna2	kolumna3
typ danych 1	typ danych 2	typ danych 3

CREATE TABLE Nazwa(k1 +1,k2 +2,...);

- nazwy kolumn
- typy danych w kolumnach: CHAR, VARCHAR, TEXT, INT, REAL, DATE, TIMESTAMP ;

Tworzenie tabel

Nazwa

kolumna1	kolumna2	kolumna3
typ danych 1	typ danych 2	typ danych 3
klucz w relacji: PRIMARY KEY	klucz obcy: REFERENCES...	inne: NOT NULL, DEFAULT, CHECK

CREATE TABLE Nazwa(k1 +1 f11 f12,k2 +2 f2,f3...);

- nazwy kolumn
- typy danych w kolumnach: CHAR, VARCHAR, TEXT, INT, REAL, DATE, TIMESTAMP ;
- więzy kolumn - klucze, klucze obce, NOT NULL, UNIQUE, DEFAULT, [CONSTRAINT nazwa] CHECK

Tworzenie tabel - przykład


```
CREATE TABLE ();
```

Tworzenie tabel - przykład

nick	grupa	data_rej

```
CREATE TABLE Os (
nick ,
grupa ,
data_rej );
```

Tworzenie tabel - przykład

nick	grupa	data_rej
nazwa - słowo	nazwa - słowo	dzien rejestracji

```
CREATE TABLE Os (
nick ,
grupa ,
data_rej );
```

Tworzenie tabel - przykład

nick	grupa	data_rej
varchar(20)	varchar(20)	date

```
CREATE TABLE Os (
nick varchar(20),
grupa varchar(20),
data_rej date );
```

Tworzenie tabel - typy kolumn

- **Typy tekstu:** CHAR, CHAR(5), VARCHAR, VARCHAR(15), TEXT,
- **Typy liczbowe:** INT, REAL, FLOAT, ...
- **Typy daty i czasu:** DATE, TIME, TIMESTAMP, ...
- **Typy logiczne:** typ BOOLEAN przyjmuje wartości TRUE, FALSE i UNKNOWN
- **Typy binarne** (zdjęcia, muzyka,...): BLOB (BYTEA), ...
- **SERIAL** - licznik automatycznie zwiększający wartość przy każdym odwołaniu;

Typy liczbowe

Typ	# bajt	Opis	Zakres
smallint	2	całkowita	-32768 .. 32767
integer	4	całkowita	-2147483648 .. 2147483647
bigint	8	całkowita	-9223372036854775808 .. 9223372036854775808
decimal		dokładna	(131072 cyfry) . (16383 cyfry)
numeric		dokładna	(131072 cyfry) . (16383 cyfry)
real	4	zmiennoprz.	6 cyfr dziesiętnych
double precision	8	zmiennoprz.	15 cyfr dziesiętnych
small-serial	2	autoincrement	1 .. 32767
serial	4	autoincrement	1 .. 2147483647
bigserial	8	autoincrement	1 .. 9223372036854775808

Typy liczbowe

Typ	# bajt	Opis	Zakres
smallint	2	całkowita	-32768 .. 32767
integer	4	całkowita	-2147483648 .. 2147483647
bigint	8	całkowita	-9223372036854775808 .. 9223372036854775808
decimal		dokładna	(131072 cyfry) . (16383 cyfry)
numeric		dokładna	(131072 cyfry) . (16383 cyfry)
real	4	zmiennoprz.	6 cyfr dziesiętnych
double precision	8	zmiennoprz.	15 cyfr dziesiętnych
small-serial	2	autoincrement	1 .. 32767
serial	4	autoincrement	1 .. 2147483647
bigserial	8	autoincrement	1 .. 9223372036854775808

Typy znakowe (tekstowe)

Typ	Opis
character varying(n) , varchar(n)	zmienna długość do podanej maksymalnej n
character(n) char(n) char = char(1)	stała długość n, krótsze teksty uzupełnione spacjami
text, varachar	dowolna, zmienna długość tekstu

Typy znakowe (tekstowe)

Typ	Opis
character varying(n) , varchar(n)	zmienna długość do podanej maksymalnej n
character(n) char(n) char = char(1)	stała długość n, krótsze teksty uzupełnione spacjami
text, varachar	dowolna, zmienna długość tekstu

Typy daty i czasu

Typ	Opis	Bajty
timestamp [without time zone]	data i czas z dokładnością do 1 mikrosek.	8
date	data (bez godziny)	4
time [without time zone]	godz:min:sek:mikrosek	12
interval	przedział czasu: 3 days 4 hours 5 minutes 6.25 seconds	12

Tworzenie tabel - typy kolumn

- **Typy tekstu:** CHAR, CHAR(5), VARCHAR, VARCHAR(15), TEXT,
- **Typy liczbowe:** INT, REAL, FLOAT, ...
- **Typy daty i czasu:** DATE, TIME, TIMESTAMP, ...
- **Typy logiczne:** typ BOOLEAN przyjmuje wartości TRUE, FALSE i UNKNOWN
- **Typy binarne** (zdjęcia, muzyka,...): BLOB (BYTEA), ...
- **SERIAL** - licznik automatycznie zwiększający wartość przy każdym odwołaniu;

Tworzenie tabel - typy kolumn

- Typy tekstu: CHAR(5), VARCHAR, VARCHAR(15), TEXT
- Typy liczbowe: REAL, FLOAT,...
- Typy daty i czasu: DATE, TIME, TIMESTAMP,...
- **Typy logiczne:** typ BOOLEAN przyjmuje wartości TRUE, FALSE i UNKNOWN
- **Typy binarne** (zdjęcia, muzyka,...): BLOB (BYTEA),...
- **SERIAL** - licznik automatycznie zwiększający wartość przy każdym odwołaniu;

Tworzenie tabel - typy kolumn

- Typy tekstu: CHAR(5), VARCHAR, VARCHAR(15), TEXT
- Typy liczbowe: REAL, FLOAT,...
- Typy daty i czasu: DATE, TIME, TIMESTAMP,...
- **Typy logiczne:** typ BOOLEAN przyjmuje wartości TRUE, FALSE i UNKNOWN
- **Typy binarne** (zdjęcia, muzyka,...): BLOB (BYTEA),...
- **SERIAL** - licznik automatycznie zwiększający wartość przy każdym odwołaniu;
- **Typ wyliczeniowy** (wyk,cw,pr,rep)
- **Struktury:** rekord, tablica, obiekt geometryczny...
- **Inne:** adresy internetowe, XML, JSON

Tworzenie tabel - przykład

nick	grupa	data_rej
varchar(20)	varchar(20)	date
klucz główny	klucz obcy do tabeli Gr	wymagana data rejestracji

```
CREATE TABLE Os (
nick varchar(20),
grupa varchar(20),
data_rej date );
```

Tworzenie tabel - przykład

nick	grupa	data_rej
varchar(20)	varchar(20)	date
primary key	klucz obcy do tabeli Gr	wymagana data rejestracji

```
CREATE TABLE Osoba (
nick varchar(20) primary key,
grupa varchar(20),
data_rej date );
```

Tworzenie tabel - przykład

nick	grupa	data_rej
varchar(20)	varchar(20)	date
primary key	references Gr(id)	wymagana data rejestracji

```
CREATE TABLE Osoba (
nick varchar(20) primary key,
grupa varchar(20) references Gr(id),
data_rej date );
```

Tworzenie tabel - przykład

nick	grupa	data_rej
varchar(20)	varchar(20)	date
primary key	references Gr(id) on delete set null	wymagana data rejestracji

```
CREATE TABLE Osoba (
nick varchar(20) primary key,
grupa varchar(20) references Gr(id) on delete set null,
data_rej date );
```

Tworzenie tabel - przykład

nick	grupa	data_rej
varchar(20)	varchar(20)	date
primary key	references Gr(id) on delete set null	not null

```
CREATE TABLE Osoba (
    nick varchar(20) primary key,
    grupa varchar(20) references Gr(id) on delete set null,
    data_rej date not null);
```

Tworzenie tabel - przykład

nick	grupa	data_rej
varchar(20)	varchar(20)	date
primary key	references Gr(id) on delete set null	not null default current_date

```
CREATE TABLE Osoba (
    nick varchar(20) primary key,
    grupa varchar(20) references Gr(id) on delete set null,
    date_rej date not null default current_date);
```

Tworzenie tabel - więzy

Tworzenie tabel - więzy

Klucz obcy:

FOREIGN KEY grupa REFERENCES Gr(id)

Tworzenie tabel - więzy

Klucz obcy:

FOREIGN KEY grupa REFERENCES Gr(id)

ON DELETE [set null | **restrict** | set default| cascade]

ON UPDATE [set null | restrict |set default| **cascade**]

Tworzenie tabel - więzy

Klucz obcy:

FOREIGN KEY grupa REFERENCES Gr(id)

ON DELETE [set null | **restrict** | set default| cascade]

ON UPDATE [set null | restrict |set default| **cascade**]

Wartość domyślna:

DEFAULT 0, DEFAULT now()

Tworzenie tabel - więzy

Klucz obcy:

FOREIGN KEY grupa REFERENCES Gr(id)

ON DELETE [set null | **restrict** | set default| cascade]

ON UPDATE [set null | restrict |set default| **cascade**]

Wartość domyślna:

DEFAULT 0, DEFAULT now()

Pola niepuste:

NOT NULL;

Tworzenie tabel - więzy

Klucz obcy:

FOREIGN KEY grupa REFERENCES Gr(id)

ON DELETE [set null | **restrict** | set default| cascade]

ON UPDATE [set null | restrict |set default| **cascade**]

Wartość domyślna:

DEFAULT 0, DEFAULT now()

Pola niepuste:

NOT NULL;

Inne warunki:

CHECK (data_rej<=current_date);

CHECK (pewnosc BETWEEN 0 AND 100);

Tworzenie tabel - więzy nazwane

Tworzenie tabel - więzy nazwane

Nadawanie nazwy więzom:

CONSTRAINT pwnsc_lim CHECK (pewnosc BETWEEN
0 AND 100);

CONSTRAINT ref_grupa FOREIGN KEY (grupa)
REFERENCES Gr(id);

Tworzenie tabel - więzy nazwane

Nadawanie nazwy więzom:

CONSTRAINT pwnsc_lim CHECK (pewnosc BETWEEN
0 AND 100);

CONSTRAINT ref_grupa FOREIGN KEY (grupa)
REFERENCES Gr(id);

Więz nazwany można usunąć:

- ALTER TABLE Os DROP CONSTRAINT ref_grupa

Tworzenie tabel - więzy nazwane

Nadawanie nazwy więzom:

CONSTRAINT pwnsc_lim CHECK (pewnosc BETWEEN 0 AND 100);

CONSTRAINT ref_grupa FOREIGN KEY (grupa)
REFERENCES Gr(id);

Więz nazwany można usunąć:

- ALTER TABLE Os DROP CONSTRAINT ref_grupa

Więz nazwany można zawiesić do końca transakcji:

- SET CONSTRAINT pwnsc_lim DEFERRABLE
- SET CONSTRAINT pwnsc_lim IMMEDIATE

Tworzenie tabel - więzy nazwane

Nadawanie nazwy więzom:

CONSTRAINT pwnsc_lim CHECK (pewnosc BETWEEN 0 AND 100) **INITIALLY DEFERRABLE**;

CONSTRAINT ref_grupa FOREIGN KEY (grupa)
REFERENCES Gr(id);

Więz nazwany można usunąć:

- ALTER TABLE Os DROP CONSTRAINT ref_grupa

Więz nazwany można zawiesić do końca transakcji:

- SET CONSTRAINT pwnsc_lim DEFERRABLE
- SET CONSTRAINT pwnsc_lim IMMEDIATE

Tworzenie tabel - więzy nazwane

Nadawanie nazwy więzom:

CONSTRAINT pwnsc_lim CHECK (pewnosc BETWEEN 0 AND 100) **INITIALLY IMMEDIATE**;
CONSTRAINT ref_grupa FOREIGN KEY (grupa)
REFERENCES Gr(id);

Więz nazwany można usunąć:

- ALTER TABLE Os DROP CONSTRAINT ref_grupa

Więz nazwany można zawiesić do końca transakcji:

- SET CONSTRAINT pwnsc_lim DEFERRABLE
- SET CONSTRAINT pwnsc_lim IMMEDIATE

Klucze złożone i niuanse CHECK

Klucze złożone i niuanse CHECK

nick: varchar(20)	moment: timestamp <i>default now()</i>	wpis: text

```
CREATE TABLE Wpisy(
    nick varchar(20) references Os(nick),
    moment timestamp not null default now()
        CHECK (moment=now()),
    wpis text CHECK (NOT substr('*' ,wpis)));
```

Klucze złożone i niuanse CHECK

nick: varchar(20)	moment: timestamp default now()	wpis: text
jeden fragment klucza	drugi fragment klucza	

```
CREATE TABLE Wpisy(  
nick varchar(20) references Os(nick),  
moment timestamp not null default now()  
    CHECK (moment=now()),  
wpis text CHECK (NOT substr('*' ,wpis)));
```

Klucze złożone i niuanse CHECK

nick: varchar(20)	moment: timestamp default now()	wpis: text
jeden fragment klucza	drugi fragment klucza	

```
CREATE TABLE Wpisy(
    nick varchar(20) references Os(nick),
    moment timestamp not null default now()
        CHECK (moment=now()),
    wpis text CHECK (NOT substr('*' ,wpis)),
    CONSTRAINT klucz_wpis_os PRIMARY KEY (nick,moment));
```

Klucze złożone i niuanse CHECK

nick: varchar(20)	moment: timestamp default now()	wpis: text	id: SERIAL

```
CREATE TABLE Wpisy(  
nick varchar(20) references Os(nick),  
moment timestamp not null default now()  
    CHECK (moment=now()),  
wpis text CHECK (NOT substr('*' ,wpis)),  
id SERIAL PRIMARY KEY);
```

Klucze złożone i niuanse CHECK

nick: varchar(20)	moment: timestamp default now()	wpis: text	id: SERIAL

```
CREATE TABLE Wpisy(  
nick varchar(20) references Os(nick),  
moment timestamp not null default now()  
    CHECK (moment=now()),  
wpis text CHECK (NOT substr('*' ,wpis)),  
id SERIAL PRIMARY KEY);
```

Zmiana schematu tablicy

Zmiana schematu tablicy

- ALTER TABLE Os ADD [COLUMN] adres text;
- ALTER TABLE Os DROP [COLUMN] grupa;

Zmiana schematu tablicy

- ALTER TABLE Os ADD [COLUMN] adres text;
- ALTER TABLE Os DROP [COLUMN] grupa;
- ALTER TABLE Os ALTER adres SET DEFAULT 'Ełk';
- ALTER TABLE Os ALTER grupa DROP NOT NULL;

Zmiana schematu tablicy

- ALTER TABLE Os ADD [COLUMN] adres text;
- ALTER TABLE Os DROP [COLUMN] grupa;
- ALTER TABLE Os ALTER adres SET DEFAULT 'Ełk';
- ALTER TABLE Os ALTER grupa DROP NOT NULL;
- DROP TABLE Gr [CASCADE]; - konsekwencje dla tablic, które odwołują się do Gr przez klucz obcy;

Zmiana schematu tablicy

- ALTER TABLE Os ADD [COLUMN] adres text;
- ALTER TABLE Os DROP [COLUMN] grupa;
- ALTER TABLE Os ALTER adres SET DEFAULT 'Ełk';
- ALTER TABLE Os ALTER grupa DROP NOT NULL;
- DROP TABLE Gr [CASCADE]; - konsekwencje dla tablic, które odwołują się do Gr przez klucz obcy;
- ALTER TABLE Os ADD CONSTRAINT...
- ALTER TABLE Os DROP CONSTRAINT...
- SET CONSTRAINT ... DEFERRABLE

Wypełnianie tablicy

Wypełnienie tablicy

- `INSERT INTO Os VALUES('Ala',NULL,now());`
- `INSERT INTO Os(nick) VALUES('Ola');`
- `INSERT INTO Os SELECT...`

Wypełnienie tablicy

- `INSERT INTO Os VALUES('Ala',NULL,now());`
- `INSERT INTO Os(nick) VALUES('Ola');`
- `INSERT INTO Os SELECT...`

Wypełnienie tablicy

- `INSERT INTO Os VALUES('Ala',NULL,now());`
- `INSERT INTO Os(nick) VALUES('Ola');`
- `INSERT INTO Os SELECT...`

- `DELETE FROM Os;`
- `DELETE FROM Os WHERE grupa IS NULL;`

Wypełnienie tablicy

- `INSERT INTO Os VALUES('Ala',NULL,now());`
- `INSERT INTO Os(nick) VALUES('Ola');`
- `INSERT INTO Os SELECT...`

- `DELETE FROM Os;`
- `DELETE FROM Os WHERE grupa IS NULL;`

Wypełnienie tablicy

- `INSERT INTO Os VALUES('Ala',NULL,now());`
- `INSERT INTO Os(nick) VALUES('Ola');`
- `INSERT INTO Os SELECT...`
- `DELETE FROM Os;`
- `DELETE FROM Os WHERE grupa IS NULL;`
- `UPDATE Os SET nick='Ala95' WHERE nick='Ala';`
- `UPDATE wpis SET moment=moment+100;`

Wypełnienie tablicy

- `INSERT INTO Os VALUES('Ala',NULL,now());`
- `INSERT INTO Os(nick) VALUES('Ola');`
- `INSERT INTO Os SELECT...`
- `DELETE FROM Os;`
- `DELETE FROM Os WHERE grupa IS NULL;`
- `UPDATE Os SET nick='Ala95' WHERE nick='Ala';`
- `UPDATE wpis SET moment=moment+100;`

Zapytania SELECT

Zapytania SELECT

```
SELECT [DISTINCT] A,C,B  
FROM R, S  
WHERE F  
ORDER BY B,C;
```

Zapytania SELECT

```
SELECT [DISTINCT] A,C,B  
FROM R, S -- (1) złączenie  
WHERE F  
ORDER BY B,C;
```

1. Obliczanie (rozumienie) zapytania rozpoczynamy od złączenia relacji wymienionych po FROM

Zapytania SELECT

```
SELECT [DISTINCT] A,C,B  
FROM R, S  
WHERE F  
ORDER BY B,C;
```

-- (1) złączenie
-- (2) selekcja

1. Obliczanie (rozumienie) zapytania rozpoczynamy od złączenia relacji wymienionych po FROM
2. Następnie wybieramy krotki złączenia spełniające warunek selekcji F podany w klauzuli WHERE

Zapytania SELECT

```
SELECT [DISTINCT] A,C,B      -- (3) rzutowanie,  
FROM R, S                    -- (1) złączenie,  
WHERE F                      -- (2) selekcja  
ORDER BY B,C;
```

1. Obliczanie (rozumienie) zapytania rozpoczynamy od złączenia relacji wymienionych po FROM
2. Następnie wybieramy krotki złączenia spełniające warunek selekcji F podany w klauzuli WHERE
3. Wynik rzutujemy na kolumny wskazane w klauzuli SELECT ewentualnie usuwając duplikaty (DISTINCT)

Zapytania SELECT

```
SELECT DISTINCT A,C,B      -- (3) rzutowanie,  
FROM R, S                  -- (1) złączenie,  
WHERE F                     -- (2) selekcja  
ORDER BY B,C;
```

1. Obliczanie (rozumienie) zapytania rozpoczynamy od złączenia relacji wymienionych po FROM
2. Następnie wybieramy krotki złączenia spełniające warunek selekcji F podany w klauzuli WHERE
3. Wynik rzutujemy na kolumny wskazane w klauzuli SELECT ewentualnie usuwając duplikaty (DISTINCT)

Zapytania SELECT

```
SELECT [DISTINCT] A,C,B      -- (3) rzutowanie,  
FROM R, S                    -- (1) złączenie,  
WHERE F                      -- (2) selekcja  
ORDER BY B,C;                -- (4) sortowanie
```

1. Obliczanie (rozumienie) zapytania rozpoczynamy od złączenia relacji wymienionych po FROM
2. Następnie wybieramy krotki złączenia spełniające warunek selekcji F podany w klauzuli WHERE
3. Wynik rzutujemy na kolumny wskazane w klauzuli SELECT ewentualnie usuwając duplikaty (DISTINCT)
4. Wynik porządkujemy wg klauzuli ORDER BY

SELECT - klauzula WHERE

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10;
```

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10;
```

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10;
```

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10;
```

Można wykonywać operacje arytmetyczne na datach i liczbach: data-data to liczba dni, data +- liczba to data; w przypadku dat ważna jest także konwersja do odpowiedniego formatu (timestamp::date lub cast);

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,
których nick zaczyna się na "ab",

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10;
```

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,
których nick zaczyna się na "ab",

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10 AND  
nick LIKE 'ab%';
```

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,
których nick zaczyna się na "ab",

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10 AND  
nick LIKE 'ab%';
```

Dla tekstów mamy operacje porównywania przybliżonego LIKE, znaki zastępcze (% zastępuje dowolny ciąg znaków a _ dowolny znak), konkatenację || i wiele innych funkcji...

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,
których nick zaczyna się na "ab",
którzy nie mają przypisanej grupy

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10 AND  
nick LIKE 'ab%';
```

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,
których nick zaczyna się na "ab",
którzy nie mają przypisanej grupy

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10 AND  
nick LIKE 'ab%' AND grupa IS NULL;
```

SELECT - klauzula WHERE

Zarejestrowani w ciągu ostatnich 10 dni,
których nick zaczyna się na "ab",
którzy nie mają przypisanej grupy

```
SELECT nick FROM Os  
WHERE data_rej>=current_date-10 AND  
nick LIKE 'ab%' AND grupa IS NULL;
```

Sprawdzenie, czy pole jest puste, wykonujemy za pomocą operatora IS NULL (niepuste - IS NOT NULL).

SELECT - klauzula SELECT

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'

SELECT nick

FROM Os WHERE grupa LIKE '%temp%';

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'
z adresem,

SELECT nick

FROM Os WHERE grupa LIKE '%temp';

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'
z adresem,

SELECT nick, adres

FROM Os WHERE grupa LIKE '%temp%';

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'
z adresem,

```
SELECT nick||' z'||adres  
FROM Os WHERE grupa LIKE '%temp%';
```

SELECT - klauzula SELECT

Nicki osób z grupy LIKE '%temp%'
z adresem,

```
SELECT nick||' z '||adres  
FROM Os WHERE grupa LIKE '%temp%';
```

Z wybranych kolumn (i stałych) możemy wyliczyć nowe kolumny pokazywane w relacji wynikowej.

SELECT - klauzula SELECT

Nicki osób z grupy LIKE '%temp%'
z adresem,
oraz proponowaną datą wyrejestrowania

SELECT nick||' z'||adres

FROM Os WHERE grupa LIKE '%temp%';

Z wybranych kolumn (i stałych) możemy wyliczyć nowe kolumny pokazywane w relacji wynikowej.

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'
z adresem,
oraz proponowaną datą wyrejestrowania

```
SELECT nick||' z'||adres,  
       data_rej+30 as "Data wypisu"  
  FROM Os WHERE grupa LIKE '%temp%';
```

Z wybranych kolumn (i stałych) możemy wyliczyć nowe kolumny pokazywane w relacji wynikowej.

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'
z adresem,
oraz proponowaną datą wyrejestrowania

```
SELECT nick||' z'||adres,  
       data_rej+30 as "Data wypisu"  
  FROM Os WHERE grupa LIKE '%temp%';
```

Z wybranych kolumn (i stałych) możemy wyliczyć nowe kolumny pokazywane w relacji wynikowej.
Dodajemy jeszcze jedną kolumnę wyliczaną i nadajemy jej nazwę.

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'
z adresem,
oraz proponowaną datą wyrejestrowania

```
SELECT nick||' z'||adres,  
       data_rej+30 as "Data wypisu"  
FROM Os WHERE grupa LIKE '%temp%';
```

Pełne rekordy grup LIKE '%temp%'

```
SELECT id, nr, status, nazwa, założyciel  
FROM Gr WHERE id LIKE '%temp%';
```

SELECT - klauzula SELECT

Nicki osób z grup LIKE '%temp%'
z adresem,
oraz proponowaną datą wyrejestrowania

```
SELECT nick||' z'||adres,  
       data_rej+30 as "Data wypisu"  
FROM Os WHERE grupa LIKE '%temp%';
```

Pełne rekordy grup LIKE '%temp%'

```
SELECT *  
FROM Gr WHERE id LIKE '%temp%';
```

Klauzula FROM

Klauzula FROM

Wypiszmy nicki i adresy wszystkich osób

```
SELECT nick, adres  
FROM Os;
```

Klauzula FROM

Wypiszmy nicki i adresy wszystkich osób
wraz z datami zamieszczenia wpisów

```
SELECT nick, adres  
FROM Os;
```

1. Potrzebujemy danych z tablicy Wpisy

Klauzula FROM

Wypiszmy nicki i adresy wszystkich osób
wraz z datami zamieszczenia wpisów

```
SELECT nick, adres  
FROM Os, Wpisy;
```

1. Potrzebujemy danych z tablicy Wpisy
2. Dopisanie na liście FROM tworzy iloczyn kartezjański

Klauzula FROM

Wypiszmy nicki i adresy wszystkich osób
wraz z datami zamieszczenia wpisów

```
SELECT nick, adres  
FROM Os, Wpisy  
WHERE Os.nick = Wpisy.nick;
```

1. Potrzebujemy danych z tablicy Wpisy
2. Dopisanie na liście FROM tworzy iloczyn kartezjański
3. Warunek selekcji powoduje, że złączenie ma sens

Klauzula FROM

Wypiszmy nicki i adresy wszystkich osób
wraz z datami zamieszczenia wpisów

```
SELECT nick, adres, (moment)::date  
FROM Os, Wpisy  
WHERE Os.nick = Wpisy.nick;
```

1. Potrzebujemy danych z tablicy Wpisy
2. Dopisanie na liście FROM tworzy iloczyn kartezjański
3. Warunek selekcji powoduje, że złączenie ma sens
4. Teraz na listę wynikową możemy dodać datę wpisu

Klauzula FROM

Wypiszmy nicki i adresy wszystkich osób
wraz z datami zamieszczenia wpisów

```
SELECT DISTINCT nick, adres, (moment)::date
FROM Os, Wpisy
WHERE Os.nick = Wpisy.nick;
```

1. Potrzebujemy danych z tablicy Wpisy
2. Dopisanie na liście FROM tworzy iloczyn kartezjański
3. Warunek selekcji powoduje, że złączenie ma sens
4. Teraz na listę wynikową możemy dodać datę wpisu
5. Dodajemy DISTINCT, by usunąć powtórzenia z wyniku

Klauzula FROM

Wypiszmy nicki i adresy wszystkich osób
wraz z datami zamieszczenia wpisów

```
SELECT DISTINCT nick, adres, (moment)::date
FROM Os, Wpisy
WHERE Os.nick = Wpisy.nick;
```

1. Potrzebujemy danych z tablicy Wpisy
2. Dopisanie na liście FROM tworzy iloczyn kartezjański
3. Warunek selekcji powoduje, że złączenie ma sens
4. Teraz na listę wynikową możemy dodać datę wpisu
5. Dodajemy DISTINCT, by usunąć powtórzenia z wyniku
6. Nazwa kolumny nick nie jest unikalna w zapytaniu!!!

Klauzula FROM

Wypiszmy nicki i adresy wszystkich osób
wraz z datami zamieszczenia wpisów

```
SELECT DISTINCT OS.nick, adres, (moment)::date
FROM Os, Wpisy
WHERE Os.nick = Wpisy.nick;
```

1. Potrzebujemy danych z tablicy Wpisy
2. Dopisanie na liście FROM tworzy iloczyn kartezjański
3. Warunek selekcji powoduje, że złączenie ma sens
4. Teraz na listę wynikową możemy dodać datę wpisu
5. Dodajemy DISTINCT, by usunąć powtórzenia z wyniku
6. Nazwa kolumny nick nie jest unikalna w zapytaniu!!!

Klauzula FROM - aliasy relacji

Wypiszmy pary osób (ich nicki),
które zarejestrowały się tego samego dnia

Klauzula FROM - aliasy relacji

Wypiszmy pary osób (ich nicki),
które zarejestrowały się tego samego dnia

```
SELECT Os.nick, Os.nick  
FROM Os, Os  
WHERE Os.data_rej=Os.data_rej;
```

1. Potrzebujemy dwóch kopii relacji Os: Os i Os

Klauzula FROM - aliasy relacji

Wypiszmy pary osób (ich nicki),
które zarejestrowały się tego samego dnia

```
SELECT Os.nick, Os.nick  
FROM Os, Os  
WHERE Os.data_rej=Os.data_rej;
```

1. Potrzebujemy dwóch kopii relacji Os: Os i Os
2. Aby je rozróżnić nadajemy im nazwy (aliasy) o1 i o2

Klauzula FROM - aliasy relacji

Wypiszmy pary osób (ich nicki),
które zarejestrowały się tego samego dnia

```
SELECT Os.nick, Os.nick  
FROM Os o1, Os o2  
WHERE Os.data_rej=Os.data_rej;
```

1. Potrzebujemy dwóch kopii relacji Os: Os i Os
2. Aby je rozróżnić nadajemy im nazwy (aliasy) o1 i o2

Klauzula FROM - aliasy relacji

Wypiszmy pary osób (ich nicki),
które zarejestrowały się tego samego dnia

```
SELECT o1.nick, o2.nick  
FROM Os o1, Os o2  
WHERE o1.data_rej=o2.data_rej;
```

1. Potrzebujemy dwóch kopii relacji Os: Os i Os
2. Aby je rozróżnić nadajemy im nazwy (aliasy) o1 i o2

Klauzula FROM - aliasy relacji

Wypiszmy pary osób (ich nicki),
które zarejestrowały się tego samego dnia

```
SELECT o1.nick, o2.nick  
FROM Os o1, Os o2  
WHERE o1.data_rej=o2.data_rej;
```

1. Potrzebujemy dwóch kopii relacji Os: **Os** i **Os**
2. Aby je rozróżnić nadajemy im nazwy (aliasy) o1 i o2

Klauzula FROM - aliasy relacji

Wypiszmy pary osób (ich nicki),
które zarejestrowały się tego samego dnia

```
SELECT o1.nick, o2.nick  
FROM Os o1, Os o2  
WHERE o1.data_rej=o2.data_rej;
```

1. Potrzebujemy dwóch kopii relacji Os: **Os** i **Os**
2. Aby je rozróżnić nadajemy im nazwy (aliasy) o1 i o2
3. Poprawiamy jeszcze zapytanie, by uniknąć par zwrotnych i symetrycznych;

Klauzula FROM - aliasy relacji

Wypiszmy pary osób (ich nicki),
które zarejestrowały się tego samego dnia

```
SELECT DISTINCT o1.nick, o2.nick  
FROM Os o1, Os o2  
WHERE o1.data_rej=o2.data_rej AND  
o1.nick <> o2.nick;
```

1. Potrzebujemy dwóch kopii relacji Os: Os i Os
2. Aby je rozróżnić nadajemy im nazwy (aliasy) o1 i o2
3. Poprawiamy jeszcze zapytanie, by uniknąć par zwrotnych i symetrycznych;

Złączenia w klaузuli FROM

Złączenia w klauzuli FROM

Os JOIN Wpis USING(nick)

złącz po jednakowych wartościach nick

Złączenia w klauzuli FROM

Os JOIN Wpis USING(nick)

złącz po jednakowych wartościach nick

Os JOIN Gr ON grupa=id

złącz według warunku grupa=id

Złączenia w klauzuli FROM

Os JOIN Wpis USING(nick)

złącz po jednakowych wartościach nick

Os JOIN Gr ON grupa=id

złącz według warunku grupa=id

Os NATURAL JOIN Wpis

złącz według wszystkich wspólnych kolumn

Złączenia w klauzuli FROM

Os JOIN Wpis USING(nick)

złącz po jednakowych wartościach nick

Os JOIN Gr ON grupa=id

złącz według warunku grupa=id

Os NATURAL JOIN Wpis

złącz według wszystkich wspólnych kolumn =

= kolumn o takich samych nazwach

Złączenia w klauzuli FROM

Os JOIN Wpis USING(nick)

złącz po jednakowych wartościach nick

Os JOIN Gr ON grupa=id

złącz według warunku grupa=id

Os NATURAL JOIN Wpis

złącz według wszystkich wspólnych kolumn =

= kolumn o takich samych nazwach

Os LEFT JOIN Wpis USING(nick)

złącz dodając do wyniku osoby bez wpisów z NULL w kolumnach wpisów;

Złączenia w klauzuli FROM

Os JOIN Wpis USING(nick)

złącz po jednakowych wartościach nick

Os JOIN Gr ON grupa=id

złącz według warunku grupa=id

Os NATURAL JOIN Wpis

złącz według wszystkich wspólnych kolumn =

= kolumn o takich samych nazwach

Os LEFT JOIN Wpis USING(nick)

złącz dodając do wyniku osoby bez wpisów z NULL w kolumnach wpisów (RIGHT i FULL OUTER JOIN);

Złączenia w klaузuli FROM

Złączenie jest łączne lewostronnie i złączenia zewnętrzne nie są wzajemnie przemienne.

Złączenia w klauzuli FROM

Złączenie jest łączne lewostronnie i złączenia zewnętrzne nie są wzajemnie przemienne.

Os LEFT JOIN Wpis USING(nick) JOIN Zal USING(id)

Złączenia w klauzuli FROM

Złączenie jest łączne lewostronnie i złączenia zewnętrzne nie są wzajemnie przemienne.

Os LEFT JOIN Wpis USING(nick) JOIN Zal USING(id)
(Os LEFT JOIN Wpis USING(nick)) JOIN Zal USING
(id)

Złączenia w klauzuli FROM

Złączenie jest łączne lewostronnie i złączenia zewnętrzne nie są wzajemnie przemienne.

Os LEFT JOIN Wpis USING(nick) JOIN Zal USING(id)
(Os LEFT JOIN Wpis USING(nick)) JOIN Zal USING
(id)

Złączenia w klauzuli FROM

Złączenie jest łączne lewostronne i złączenia zewnętrzne nie są wzajemnie przemienne.

Os LEFT JOIN Wpis USING(nick) JOIN Zal USING(id)
(Os LEFT JOIN Wpis USING(nick)) JOIN Zal USING(id)

Tylko osoby, które mają wpis z załącznikiem

Os LEFT JOIN (Wpis USING(nick)) JOIN Zal USING(id))

Osoby, które mają wpis z załącznikiem, są z tym wpisem pozostałe są z pustymi polami wpisu

Złączenia w klauzuli FROM

Złączenie jest łączne lewostronne i złączenia zewnętrzne nie są wzajemnie przemienne.

Os LEFT JOIN Wpis USING(nick) JOIN Zal USING(id)
(Os LEFT JOIN Wpis USING(nick)) JOIN Zal USING(id)

Tylko osoby, które mają wpis z załącznikiem

Os LEFT JOIN (Wpis USING(nick)) JOIN Zal USING(id))

Osoby, które mają wpis z załącznikiem, są z tym wpisem; pozostałe są z pustymi polami wpisu

Operacje na zbiorach

Operacje na zbiorach

(SELECT nick,data_rej as "data" FROM Os WHERE
data_rej BETWEEN '2009.01.01' AND '2009.12.31')

Operacje na zbiorach

(SELECT nick,data_rej as "data" FROM Os WHERE
data_rej BETWEEN '2009.01.01' AND '2009.12.31')

(SELECT nick,data1 as "data" FROM ArchOs WHERE
data1 BETWEEN '2009.01.01' AND '2009.12.31' AND
data2>'2009.12.31');

Operacje na zbiorach

```
(SELECT nick,data_rej as "data" FROM Os WHERE  
data_rej BETWEEN '2009.01.01' AND '2009.12.31')  
UNION ALL  
(SELECT nick,data1 as "data" FROM ArchOs WHERE  
data1 BETWEEN '2009.01.01' AND '2009.12.31' AND  
data2>'2009.12.31');
```

Operacje na zbiorach

```
(SELECT nick,data_rej as "data" FROM Os WHERE  
data_rej BETWEEN '2009.01.01' AND '2009.12.31')  
UNION ALL  
(SELECT nick,data1 as "data" FROM ArchOs WHERE  
data1 BETWEEN '2009.01.01' AND '2009.12.31' AND  
data2>'2009.12.31');
```

- Wykonując sumę musimy zwrócić uwagę na **zgodność typów** dodawanych relacji;

Operacje na zbiorach

```
(SELECT nick,data_rej as "data" FROM Os WHERE  
data_rej BETWEEN '2009.01.01' AND '2009.12.31')  
UNION ALL
```

```
(SELECT nick,data1 as "data" FROM ArchOs WHERE  
data1 BETWEEN '2009.01.01' AND '2009.12.31' AND  
data2>'2009.12.31');
```

- Wykonując sumę musimy zwrócić uwagę na zgodność typów dodawanych relacji;
- Suma (UNION) automatycznie usuwa duplikaty; jeśli chcemy je pozostawić stosujemy **UNION ALL**;

Operacje na zbiorach

```
(SELECT nick,data_rej as "data" FROM Os WHERE  
data_rej BETWEEN '2009.01.01' AND '2009.12.31')  
UNION
```

```
(SELECT nick,data1 as "data" FROM ArchOs WHERE  
data1 BETWEEN '2009.01.01' AND '2009.12.31' AND  
data2>'2009.12.31');
```

- Wykonując sumę musimy zwrócić uwagę na zgodność typów dodawanych relacji;
- Suma (**UNION**) automatycznie usuwa duplikaty; jeśli chcemy je pozostawić stosujemy UNION ALL;

Operacje na zbiorach

```
(SELECT nick,data_rej as "data" FROM Os WHERE  
data_rej BETWEEN '2009.01.01' AND '2009.12.31')  
UNION  
(SELECT nick,data1 as "data" FROM ArchOs WHERE  
data1 BETWEEN '2009.01.01' AND '2009.12.31' AND  
data2>'2009.12.31');
```

- Wykonując sumę musimy zwrócić uwagę na zgodność typów dodawanych relacji;
- Suma (UNION) automatycznie usuwa duplikaty; jeśli chcemy je pozostawić stosujemy UNION ALL;
- Pozostałe operacje to EXCEPT (MINUS) i INTERSECT

Operacje na zbiorach

```
(SELECT nick,data_rej as "data" FROM Os WHERE  
data_rej BETWEEN '2009.01.01' AND '2009.12.31')  
UNION  
(SELECT nick,data1 as "data" FROM ArchOs WHERE  
data1 BETWEEN '2009.01.01' AND '2009.12.31' AND  
data2>'2009.12.31');
```

- Wykonując sumę musimy zwrócić uwagę na zgodność typów dodawanych relacji;
- Suma (UNION) automatycznie usuwa duplikaty; jeśli chcemy je pozostawić stosujemy UNION ALL;
- Pozostałe operacje to EXCEPT (MINUS) i INTERSECT
- EXCEPT i INTERSECT też domyślnie usuwają duplikaty

Porządkowanie wyniku

Porządkowanie wyniku

- Usuwanie duplikatów DISTINCT
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: ORDER BY nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

Porządkowanie wyniku

- Usuwanie duplikatów DISTINCT
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: ORDER BY nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

```
SELECT DISTINCT nick,  
    EXTRACT(hour FROM moment) AS "data_wyp"  
FROM Os JOIN Wpis USING(nick)  
WHERE (moment::date)=current_date-1  
ORDER BY "data_wp" DESC, nick  
LIMIT 10;
```

Porządkowanie wyniku

- Usuwanie duplikatów **DISTINCT**
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: ORDER BY nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

```
SELECT DISTINCT nick,  
    EXTRACT(hour FROM moment) AS "data_wyp"  
FROM Os JOIN Wpis USING(nick)  
WHERE (moment::date)=current_date-1  
ORDER BY "data_wp" DESC, nick  
LIMIT 10;
```

Porządkowanie wyniku

- Usuwanie duplikatów DISTINCT
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: ORDER BY nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

```
SELECT DISTINCT nick,  
    EXTRACT(hour FROM moment) AS "data_wyp"  
FROM Os JOIN Wpis USING(nick)  
WHERE (moment::date)=current_date-1  
ORDER BY "data_wp" DESC, nick  
LIMIT 10;
```

Porządkowanie wyniku

- Usuwanie duplikatów DISTINCT
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: ORDER BY nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

```
SELECT DISTINCT nick,  
    EXTRACT(hour FROM moment) AS "data_wyp"  
FROM Os JOIN Wpis USING(nick)  
WHERE (moment::date)=current_date-1  
ORDER BY "data_wp" DESC, nick  
LIMIT 10;
```

Porządkowanie wyniku

- Usuwanie duplikatów DISTINCT
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: ORDER BY nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

```
SELECT DISTINCT nick,  
    EXTRACT(hour FROM moment) AS "data_wyp"  
FROM Os JOIN Wpis USING(nick)  
WHERE (moment::date)=current_date-1  
ORDER BY "data_wp" DESC, nick  
LIMIT 1 OFFSET 100;
```

Porządkowanie wyniku

- Usuwanie duplikatów DISTINCT
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: **ORDER BY** nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

```
SELECT DISTINCT nick,  
    EXTRACT(hour FROM moment) AS "data_wyp"  
FROM Os JOIN Wpis USING(nick)  
WHERE (moment::date)=current_date-1  
ORDER BY "data_wp" DESC, nick  
LIMIT 10;
```

Porządkowanie wyniku

- Usuwanie duplikatów DISTINCT
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: ORDER BY nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

```
SELECT DISTINCT nick,  
    EXTRACT(hour FROM moment) AS "data_wyp"  
FROM Os JOIN Wpis USING(nick)  
WHERE (moment::date)=current_date-1  
ORDER BY "data_wp" DESC, nick  
LIMIT 10;
```

Porządkowanie wyniku

- Usuwanie duplikatów DISTINCT
- Nazywanie kolumn AS "data_wyp"
- Sortowanie: ORDER BY nazwa lub liczba i DESC/ASC
- Ograniczenie liczby krotek: LIMIT, OFFSET

```
SELECT DISTINCT nick,  
    EXTRACT(hour FROM moment) AS "data_wyp"  
FROM Os JOIN Wpis USING(nick)  
WHERE (moment::date)=current_date-1  
ORDER BY 2 DESC, 1  
LIMIT 10;
```

SQL - podstawy

Podsumowanie

DDL (definiowanie tabel)

```
CREATE Os (
    id SERIAL PRIMARY KEY,
    nazwisko text NOT NULL,
    adres text,
    data_rej date DEFAULT current_date);
```

```
CREATE Grupa (
    id SERIAL PRIMARY KEY,
    nazwa text UNIQUE NOT NULL,
    założyciel integer REFERENCES Os ON DELETE SET NULL,
    typ char CHECK typ IN ('z','s','p'));
```

```
DROP TABLE Os [CASCADE];
```

DDL (definiowanie tabel)

```
CREATE Os (
    id SERIAL PRIMARY KEY,
    nazwisko text NOT NULL,
    adres text,
    data_rej date DEFAULT current_date);
```

SERIAL to nie tyle typ danych, co dodatkowy obiekt w bazie - sekwencja generująca na żądanie nową wartość: nextval("sek"), setval ("sek");

```
CREATE Grupa (
    id SERIAL PRIMARY KEY,
    nazwa text UNIQUE NOT NULL,
    założyciel integer REFERENCES Os ON DELETE SET NULL,
    typ char CHECK typ IN ('z','s','p'));
```

```
DROP TABLE Os [CASCADE];
```

DDL (definiowanie struktury)

```
CREATE Os (
    id SERIAL PRIMARY KEY,
    nazwisko text NOT NULL,
    adres text,
    data_rej date DEFAULT current
```

PRIMARY KEY jest unikalny, niepusty i powoduje utworzenie indeksu (np. B-drzewa)

UNIQUE sprawdza unikalność wartości niepustych; też powoduje utworzenie indeksu.

UNIQUE i NOT NULL to klucz alternatywny.

```
CREATE Grupa (
    id SERIAL PRIMARY KEY,
    nazwa text UNIQUE NOT NULL,
    założyciel integer REFERENCES Os ON DELETE SET NULL,
    typ char CHECK typ IN ('z','s','p'));
```

```
DROP TABLE Os [CASCADE];
```

DDL (definicja)

```
CREATE Os (
    id SERIAL PRIMARY KEY,
    nazwisko text NOT NULL,
    adres text,
    data_rej date DEFAULT cur
```

```
CREATE Grupa (
    id SERIAL PRIMARY KEY,
    nazwa text UNIQUE NOT NULL,
    założyciel integer REFERENCES Os ON DELETE SET NULL,
    typ char CHECK typ IN ('z','s','p'));
```

```
DROP TABLE Os [CASCADE];
```

FOREIGN KEY... REFERENCES (klucz obcy) wymaga, by w tabeli nadzędnej istniała krotka wskazywana przez klucz. Usunięcie krotki nadzędnej może zostać:

- zablokowane (RESTRICT|NO ACTION)
- kaskadowo usunąć krotki podrzędne
- wykasować lub zmienić na wartość domyślną klucze obce w krotkach podrzędnych.

Analogiczne efekty wywołuje modyfikacja krotki nadzędnej.

DDL (definicja)

```
CREATE Os (
    id SERIAL PRIMARY KEY,
    nazwisko text NOT NULL,
    adres text,
    data_rej date DEFAULT cur
```

```
CREATE Grupa (
    id SERIAL PRIMARY KEY,
    nazwa text UNIQUE NOT NULL,
    założyciel integer REFERENCES Os ON DELETE SET NULL,
    typ char CHECK typ IN ('z','s','p'))
```

```
DROP TABLE Os [CASCADE];
```

FOREIGN KEY... REFERENCES (klucz obcy) wymaga, by w tabeli nadzędnej istniała krotka wskazywana przez klucz. Usunięcie krotki nadzędnej może zostać:

- zablokowane (RESTRICT|NO ACTION)
- kaskadowo usunąć krotki podrzędne
- wykasować lub zmienić na wartość domyślną klucze obce w krotkach podrzędnych.

Analogiczne efekty wywołuje modyfikacja krotki nadzędnej.

Akcja referencyjna:

Usunięcie tabeli, na którą inne wskazują przez FOREIGN KEY, powoduje ostrzeżenie.

Usunięcie z opcją CASCADE powoduje usunięcie tabeli i więzu klucza obcego.

DDL (definiowanie tabel)

```
CREATE Os (
    id SERIAL PRIMARY KEY,
    nazwisko text NOT NULL,
    adres text,
    data_rej date DEFAULT current_date);
```

```
CREATE Grupa (
    id SERIAL PRIMARY KEY,
    nazwa text UNIQUE NOT NULL,
    założyciel integer REFERENCES Os ON DELETE SET NULL,
    typ char CHECK typ IN ('z','s','p'));
```

```
DROP TABLE Os [CASCADE];
```

Więz **CHECK** dotyczy jednej krotki.
Jest sprawdzany w momencie jej
wstawienia lub modyfikacji.

DDL (zmiany schematu)

ALTER TABLE tabela [ADD|DROP] COLUMN opis_kolumny;

ALTER TABLE tabela [ADD|DROP] CONSTRAINT nazwa_więzu

- CHECK (...)
- FOREIGN KEY ... REFERENCES ...
- PRIMARY KEY ...

ALTER TABLE tabela

- ALTER COLUMN kolumna
 - [SET|DROP] DEFAULT
 - [SET|DROP] NOT NULL
 - TYPE nowy_typ
- RENAME TO nowa_nazwa;

ALTER tabela RENAME TO nowa_nazwa;

DML (zmiany stanu)

INSERT INTO tabela VALUES('...','...','...');

INSERT INTO tabela(kol1,kol2) VALUES('...','...');

INSERT INTO tabela[(kol1,kol2)]

 SELECT kol1,kol2 FROM ... WHERE ...;

DELETE FROM tabela WHERE ...;

UPDATE tabela SET pole = wyrazenie WHERE ...;

Query Language

SELECT [DISTINCT]

- *, tabela.*
- atrybuty
- stałe
- wyrażenia (kolumny) wyliczane z atrybutów i stałych
- przemianowane kolumny

FROM

- lista relacji R, S, ... (R X S X...)
- aliasy relacji R r, S s
- łączenia relacji:
 - R JOIN S ON(warunek)
 - R JOIN S USING (wspólne kolumny)
 - R NATURAL JOIN S
 - R [LEFT|RIGHT|FULL] [OUTER] JOIN S [ON|USING]

WHERE

- warunek selekcji: R.id=S.id OR S.nazwa LIKE '%temp%' OR typ IS NULL

SQL - zapytania złożone

Podzapytania, grupowanie i
agregacja

Podzapytania

Podzapytania

SELECT A,B FROM R, S WHERE

Podzapytania

SELECT A,B FROM R, S WHERE

SELECT A,B FROM (SELECT...) X, S WHERE

Podzapytania w klauzuli WHERE

SELECT A,B FROM R, S WHERE

W **klauzuli WHERE** możemy stosować operatory, których argumentem jest zbiór - może to być zbiór stały (jawnie wymienione wartości) lub wynik (pod)zapytania SELECT.

Podzapytania w klauzuli WHERE

SELECT A,B FROM R, S WHERE

W **klauzuli WHERE** możemy stosować operatory, których argumentem jest zbiór - może to być zbiór stały (jawnie wymienione wartości) lub wynik (pod)zapytania SELECT.

- wartość > ANY (SELECT... FROM ... WHERE ...)
- wartość > SOME (SELECT... FROM ... WHERE ...)
- wartość > ALL (SELECT... FROM ... WHERE ...)
- wartość IN (SELECT... FROM ... WHERE ...)
- EXISTS (SELECT * FROM ... WHERE ...)

Podzapytania w klauzuli WHERE

SELECT A,B FROM R, S WHERE

W **klauzuli WHERE** możemy stosować **operatorы**, których argumentem jest zbiór - może to być zbiór stały (jawnie wymienione wartości) lub wynik (pod)zapytania SELECT.

- wartość > ANY (SELECT... FROM ... WHERE ...)
- wartość > SOME (SELECT... FROM ... WHERE ...)
- wartość > ALL (SELECT... FROM ... WHERE ...)
- wartość IN (SELECT... FROM ... WHERE ...)
- EXISTS (SELECT * FROM ... WHERE ...)

Podzapytania w klauzuli WHERE

SELECT A,B FROM R, S WHERE

W **klauzuli WHERE** możemy stosować **operatorы**, których argumentem jest zbiór - może to być zbiór stały (jawnie wymienione wartości) lub wynik (pod)zapytania SELECT.

- wartość > ANY (SELECT... FROM ... WHERE ...)
- wartość > SOME (SELECT... FROM ... WHERE ...)
- wartość > ALL (SELECT... FROM ... WHERE ...)
- wartość IN (SELECT... FROM ... WHERE ...)
- EXISTS (SELECT * FROM ... WHERE ...)

Podzapytania w klauzuli WHERE

SELECT A,B FROM R, S WHERE

W **klauzuli WHERE** możemy stosować **operatorы**, których argumentem jest **zbiór** - może to być zbiór stały (jawnie wymienione wartości) lub wynik (pod)zapytania SELECT.

- wartość > ANY (SELECT... FROM ... WHERE ...)
- wartość > SOME (SELECT... FROM ... WHERE ...)
- wartość > ALL (SELECT... FROM ... WHERE ...)
- wartość IN (SELECT... FROM ... WHERE ...)
- EXISTS (SELECT * FROM ... WHERE ...)

Podzapytania w klauzuli WHERE

SELECT A,B FROM R, S WHERE

W **klauzuli WHERE** możemy stosować **operatorы**, których argumentem jest **zbiór** - może to być zbiór stały (jawnie wymienione wartości) lub wynik (pod)zapytania **SELECT**.

- wartość < ANY (SELECT... FROM ... WHERE ...)
- wartość >= SOME (SELECT... FROM ... WHERE ...)
- wartość != ALL (SELECT... FROM ... WHERE ...)
- wartość NOT IN (SELECT... FROM ... WHERE ...)
- NOT EXISTS (SELECT * FROM ... WHERE ...)

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ... ;
```

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ... ;
```

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (wyniki Anny Abackiej);
```

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (wyniki Anny Abackiej);
```

```
SELECT wynik FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna';
```

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (
```

```
SELECT wynik FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna';
```

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (  
SELECT wynik FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna';
```

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytanie jest wyliczane "w kontekście" zapytania, ale użyte w nim nazwy (relacje, zmienne, atrybuty) przesyłają nazwy z kontekstu zapytania.

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytanie jest wyliczane "w kontekście" zapytania, ale użyte w nim nazwy (relacje, zmienne, atrybuty) przesyłają nazwy z kontekstu zapytania.

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytanie jest wyliczane "w kontekście" zapytania, ale użyte w nim nazwy (relacje, zmienne, atrybuty) przesyłają nazwy z kontekstu zapytania.

To wszystko?

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytanie jest wyliczane "w kontekście" zapytania, ale użyte w nim nazwy (relacje, zmienne, atrybuty) przesyłają nazwy z kontekstu zapytania.

To wszystko?

Pytalismy o osoby, a nie o krotki złączenia Osoba i Wynik!

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT * FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytanie jest wyliczane "w kontekście" zapytania, ale użyte w nim nazwy (relacje, zmienne, atrybuty) przesyłają nazwy z kontekstu zapytania.

To wszystko?

Pytalismy o osoby, a nie o krotki złączenia Osoba i Wynik!

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT Osoby.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytanie jest wyliczane "w kontekście" zapytania, ale użyte w nim nazwy (relacje, zmienne, atrybuty) przesyłają nazwy z kontekstu zapytania.

To wszystko?

Pytalismy o osoby, a nie o krotki złączenia Osoba i Wynik!

Podzapytania w klauzuli WHERE - przykład 1

Osoby mające jakiś wynik lepszy niż jakiś wynik Anny Abackiej

```
SELECT Osoby.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ANY (SELECT wynik  
FROM Wynik JOIN Osoba ON id=osoba  
WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytanie jest wyliczane "w kontekście" zapytania, ale użyte w nim nazwy (relacje, zmienne, atrybuty) przesyłają nazwy z kontekstu zapytania.

To wszystko?

Pytalismy o osoby, a nie o krotki złączenia Osoba i Wynik!

Teraz to wszystko

Podzapytania w klauzuli WHERE - przykład 2

Osoby mające jakiś wynik lepszy niż wszystkie wyniki Anny Abackiej

Podzapytania w klauzuli WHERE - przykład 2

Osoby mające jakiś wynik lepszy niż **wszystkie** wyniki Anny Abackiej

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ALL (...);
```

Podzapytania w klauzuli WHERE - przykład 2

Osoby mające jakiś wynik lepszy niż **wszystkie wyniki Anny Abackiej**

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE wynik > ALL (SELECT wynik  
                      FROM Wynik JOIN Osoba ON id=osoba  
                      WHERE nazwisko='Abacka' AND imie='Anna');
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
ALL(wyniki tej osoby dla tego zadania w ubiegłych latach);
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
    ALL(wyniki tej osoby dla tego zadania w ubiegłych latach);
```

```
SELECT wynik FROM Wynik
WHERE osoba=? AND zad=?
    AND EXTRACT(year FROM czasWyk)<2011;
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
    ALL(wyniki tej osoby dla tego zadania w ubiegłych latach);
```

```
SELECT wynik FROM Wynik
WHERE osoba=? AND zad=?
    AND EXTRACT(year FROM czasWyk)<2011;
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
    ALL(wyniki tej osoby dla tego zadania w ubiegłych latach);
```

```
SELECT wynik FROM Wynik
WHERE osoba=w1.osoba AND zad=w1.zad
    AND EXTRACT(year FROM czasWyk)<2011;
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
    ALL(
```

```
SELECT wynik FROM Wynik
WHERE osoba=w1.osoba AND zad=w1.zad
    AND EXTRACT(year FROM czasWyk)<2011;
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>  
ALL(
```

```
SELECT wynik FROM Wynik  
WHERE osoba=w1.osoba AND zad=w1.zad  
AND EXTRACT(year FROM czasWyk)<2011);
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
ALL(SELECT wynik FROM Wynik
      WHERE osoba=w1.osoba AND zad=w1.zad
      AND EXTRACT(year FROM czasWyk)<2011);
```

Podzapytania w klauzuli WHERE - przykład 3

Osoby, które od zeszłego roku poprawiły jakiś swój wynik
czyli

Osoby, które mają jakiś wynik za zadanie zrobiony w 2011 roku
lepszy niż ich wszystkie wyniki za to zadanie w ubiegłych latach.

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba
WHERE EXTRACT(year FROM czasWyk)=2011 AND wynik>
ALL(SELECT wynik FROM Wynik
      WHERE osoba=w1.osoba AND zad=w1.zad
      AND EXTRACT(year FROM czasWyk)<2011);
```

W tym przykładzie mamy przykład podzapytania zależnego - w jego treści pojawiają się odwołania do krotki, dla której jest obliczane i musi być obliczane wielokrotnie.

Podzapytania w klauzuli WHERE - przykład 4

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością ≥ 50)

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością ≥ 50)

```
SELECT * FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (...)
```

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT * FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (...)
```

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT * FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (...)
```

```
SELECT Zad.id  
FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50;
```

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT * FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (
```

```
SELECT Zad.id  
FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT * FROM Osoba JOIN Wynik ON osoba=id
WHERE wynik=100 AND zad IN (
    SELECT Zad.id
    FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat
    WHERE kat.nazwa='BD' and trafnosc>=50);
```

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT * FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Zad.id  
FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT * FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Zad.id  
FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Zad.id  
FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Zad.id  
FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?
To wszystko?

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Zad.id  
FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?

To wszystko?

Łączenie z tabelą Zad w podzapytaniu jest zbędne!

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Zad.id  
FROM Zad JOIN Klas ON id=zad JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?

To wszystko?

Łączenie z tabelą Zad w podzapytaniu jest zbędne!

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Zad.id  
FROM Klas JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?

To wszystko?

Łączenie z tabelą Zad w podzapytaniu jest zbędne!

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Zad.id  
FROM Klas JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?

To wszystko?

Łączenie z tabelą Zad w podzapytaniu jest zbędne!

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Klas.zad  
FROM Klas JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?

To wszystko?

Łączenie z tabelą Zad w podzapytaniu jest zbędne!

Podzapytania w klauzuli WHERE - przykład 4

Osoby, które uzyskały wynik 100 za jakieś zadanie z kategorii BD
(sklasyfikowane jako takie z pewnością >= 50)

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON osoba=id  
WHERE wynik=100 AND zad IN (SELECT Klas.zad  
FROM Klas JOIN Kat ON kat.id=klas.kat  
WHERE kat.nazwa='BD' and trafnosc>=50);
```

To wszystko?

To wszystko?

Łączenie z tabelą Zad w podzapytaniu jest zbędne!

Teraz to wszystko!

Podzapytania w klauzuli WHERE - przykład 5

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie.

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(...)
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje **ich gorszy wynik za to samo zadanie z ubiegłych lat**

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(...)
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(...)
```

```
SELECT * FROM Wynik WHERE  
EXTRACT(year FROM czasWyk)<2011 AND  
osoba=osoba AND zad=zad AND wynik<wynik
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(...)
```

```
SELECT * FROM Wynik WHERE  
EXTRACT(year FROM czasWyk)<2011 AND  
osoba=w1.osoba AND zad=w1.zad AND wynik<w1.wynik
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(...)
```

```
SELECT * FROM Wynik WHERE  
EXTRACT(year FROM czasWyk)<2011 AND  
osoba=w1.osoba AND zad=w1.zad AND wynik<w1.wynik
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie.

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(...)
```

```
SELECT * FROM Wynik WHERE  
EXTRACT(year FROM czasWyk)<2011 AND  
osoba=w1.osoba AND zad=w1.zad AND wynik<w1.wynik
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(...)
```

```
SELECT * FROM Wynik w2 WHERE  
EXTRACT(year FROM czasWyk)<2011 AND  
w2.osoba=w1.osoba AND w2.zad=w1.zad AND w2.wynik<w1.wynik
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(
```

```
SELECT * FROM Wynik w2 WHERE  
EXTRACT(year FROM czasWyk)<2011 AND  
w2.osoba=w1.osoba AND w2.zad=w1.zad AND w2.wynik<w1.wynik);
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(  
SELECT * FROM Wynik w2 WHERE  
EXTRACT(year FROM czasWyk)<2011 AND  
w2.osoba=w1.osoba AND w2.zad=w1.zad AND w2.wynik<w1.wynik);
```

Podzapytania w klauzuli WHERE - przykład 5

Osoby, które od zeszłego roku poprawiły swój wynik za jakieś zadanie

czyli

istnieje ich gorszy wynik za to samo zadanie z ubiegłych lat

```
SELECT Osoba.* FROM Osoba JOIN Wynik w1 ON id=osoba  
WHERE EXTRACT(year FROM czasWyk)=2011 AND EXISTS(  
SELECT * FROM Wynik w2 WHERE  
EXTRACT(year FROM czasWyk)<2011 AND  
w2.osoba=w1.osoba AND w2.zad=w1.zad AND w2.wynik<w1.wynik);
```

W podzapytaniu nie ma sensu bawić się w sortowanie, usuwanie duplikatów (ew. w rzutowanie przy EXISTS). To zbędna praca.

Alternatywne postaci zapytań

Alternatywne postaci zapytań

To samo pytanie można zadać na kilka istotnie różnych sposobów. Posłużymy się teraz przykładami z bazy Zapisów. Osoby, które chodziły na wykład z Baz Danych można znaleźć:

- wybierając tych, którzy łączą się z wpisem do jakiejś grupy wykładowej z Baz danych (JOIN);
- wybierając osoby, których kody są w zbiorze tych, które dotyczą wyboru grupy wykładowej... (IN);
- wybierając osoby, dla których istnieje wybór, dla którego istnieje grupa, dla której istnieje.... (EXISTS)

Alternatywne postaci zapytań - przykład

Wybierzmy nazwiska osób, które zapisali się (kiedykolwiek) na wykład z Baz danych - zastosujmy JOIN

```
SELECT DISTINCT nazwisko FROM
    użytkownik
    JOIN wybór USING(kod_uz)
    JOIN grupa USING(kod_grupy)
    JOIN przedmiot_semestr USING(kod_przed_sem)
    JOIN przedmiot using(kod_przed)
WHERE rodzaj_zajec='w' AND nazwa='Bazy danych';
```

Alternatywne postaci zapytań - przykład

Wybierzmy nazwiska osób, które zapisali się (kiedykolwiek) na wykład z Baz danych - zastosujmy IN

```
SELECT DISTINCT nazwisko FROM użytkownik  
WHERE kod_uz IN  
  (SELECT kod_uz FROM wybor  
   WHERE kod_grupy IN  
     (SELECT kod_grupy FROM grupa  
      WHERE rodzaj_zajec='w' AND kod_przed_sem IN  
        (SELECT kod_przed_sem FROM przedmiot_semestr  
         WHERE kod_przed IN  
           (SELECT kod_przed FROM przedmiot  
            WHERE nazwa='Bazy danych'))));
```

Alternatywne postaci zapytań - przykład

Wybierzmy nazwiska osób, które zapisali się (kiedykolwiek) na wykład z Baz danych - zastosujmy EXISTS

```
SELECT DISTINCT nazwisko FROM uzytkownik u WHERE
EXISTS(SELECT * FROM wybor w WHERE u.kod_uz=w.kod_uz
AND
    EXISTS (SELECT * FROM grupa g WHERE rodzaj_zajec='w'
        AND g.kod_grupy=w.kod_grupy AND
    EXISTS (SELECT * FROM przedmiot_semestr ps
        WHERE ps.kod_przed_sem=g.kod_przed_sem AND
    EXISTS (SELECT * FROM przedmiot p WHERE
        p.nazwa='Bazy danych' AND
        p.kod_przed=ps.kod_przed))));
```

Alternatywne postaci zapytań - przykład

Aby porównać działanie tych wersji możemy je wykonać poprzedzając poleceniem EXPLAIN lub EXPLAIN ANALYZE:

```
explain analyze select nazwisko from uzytkownik join  
wybor using(kod_uz) join ....
```

Alternatywne postaci zapytań - przykład

Aby porównać działanie tych wersji możemy je wykonać poprzedzając poleceniem EXPLAIN lub EXPLAIN ANALYZE:

QUERY PLAN

```
HashAggregate (cost=130.48..130.84 rows=36 width=9) (actual time=6.391..6.522 rows=255 loops=1)
  -> Nested Loop (cost=51.22..130.39 rows=36 width=9) (actual time=2.126..6.091 rows=307 loops=1)
    -> Nested Loop (cost=51.22..120.13 rows=36 width=4) (actual time=2.116..4.257 rows=307 loops=1)
      -> Hash Join (cost=51.22..115.94 rows=2 width=4) (actual time=1.986..3.522 rows=8 loops=1)
        Hash Cond: (grupa.kod_przed_sem = przedmiot_semestr.kod_przed_sem)
        -> Seq Scan on grupa (cost=0.00..61.90 rows=745 width=8) (actual time=0.017..1.298 rows=745 loops=1)
          Filter: (rodzaj_zajec = 'w'::bpchar)
        -> Hash (cost=51.17..51.17 rows=4 width=4) (actual time=1.877..1.877 rows=16 loops=1)
          -> Hash Join (cost=13.74..51.17 rows=4 width=4) (actual time=0.247..1.867 rows=16 loops=1)
            Hash Cond: (przedmiot_semestr.kod_przed = przedmiot.kod_przed)
            -> Seq Scan on przedmiot_semestr (cost=0.00..30.47 rows=1847 width=8) (actual time=0.007..
0.795 rows=1847 loops=1)
              -> Hash (cost=13.72..13.72 rows=1 width=4) (actual time=0.181..0.181 rows=1 loops=1)
                -> Seq Scan on przedmiot (cost=0.00..13.72 rows=1 width=4) (actual time=0.020..0.178 rows=1
loops=1)
                  Filter: (nazwa = 'Bazy danych'::text)
                -> Index Scan using wybor_key on wybor (cost=0.00..1.81 rows=23 width=8) (actual time=0.008..0.060
rows=38 loops=8)
                  Index Cond: (wybor.kod_grupy = grupa.kod_grupy)
                -> Index Scan using uzytkownik_key on uzytkownik (cost=0.00..0.27 rows=1 width=13) (actual time=0.003..0.004
rows=1 loops=307)
                  Index Cond: (uzytkownik.kod_uz = wybor.kod_uz)
Total runtime: 6.696 ms
(19 rows)
```

Alternatywne postaci zapytań - przykład

Aby porównać działanie tych wersji możemy je wykonać poprzedzając poleceniem EXPLAIN lub EXPLAIN ANALYZE:

QUERY PLAN

```
-> Nested Loop (cost=51.22..130.39 rows=36 width=9) (actual time=2.126..6.091 rows=307 loops=1)
    -> Nested Loop (cost=51.22..120.13 rows=36 width=4) (actual time=2.116..4.257 rows=307 loops=1)
        -> Hash Join (cost=51.22..115.94 rows=2 width=4) (actual time=1.986..3.522 rows=8 loops=1)
            Hash Cond: (grupa.kod_przed_sem = przedmiot_semestr.kod_przed_sem)
            -> Seq Scan on grupa (cost=0.00..61.90 rows=745 width=8) (actual time=0.017..1.298 rows=745 loops=1)
                Filter: (rodzaj_zajec = 'w'::bpchar)
            -> Hash (cost=51.17..51.17 rows=4 width=4) (actual time=1.877..1.877 rows=16 loops=1)
                -> Hash Join (cost=13.74..51.17 rows=4 width=4) (actual time=0.247..1.867 rows=16 loops=1)
                    Hash Cond: (przedmiot_semestr.kod_przed = przedmiot.kod_przed)
                    -> Seq Scan on przedmiot_semestr (cost=0.00..30.47 rows=1847 width=8) (actual time=0.007..0.795 rows=1847 loops=1)
                        -> Hash (cost=13.72..13.72 rows=1 width=4) (actual time=0.181..0.181 rows=1 loops=1)
                            -> Seq Scan on przedmiot (cost=0.00..13.72 rows=1 width=4) (actual time=0.020..0.178 rows=1 loops=1)
                                Filter: (nazwa = 'Bazy danych'::text)
                            -> Index Scan using wybor_key on wybor (cost=0.00..1.81 rows=23 width=8) (actual time=0.008..0.060 rows=38 loops=8)
                                Index Cond: (wybor.kod_grupy = grupa.kod_grupy)
                            -> Index Scan using uzytkownik_key on uzytkownik (cost=0.00..0.27 rows=1 width=13) (actual time=0.003..0.004 rows=1 loops=307)
                                Index Cond: (uzytkownik.kod_uz = wybor.kod_uz)
Total runtime: 6.696 ms
(19 rows)
```

Alternatywne postaci zapytań - przykład

Aby porównać działanie tych wersji możemy je wykonać poprzedzając poleceniem EXPLAIN lub EXPLAIN ANALYZE:

QUERY PLAN

```
-> Nested Loop (cost=51.22..120.13 rows=36 width=4) (actual time=2.116..4.257 rows=307 loops=1)
    -> Hash Join (cost=51.22..115.94 rows=2 width=4) (actual time=1.986..3.522 rows=8 loops=1)
        Hash Cond: (grupa.kod_przed_sem = przedmiot_semestr.kod_przed_sem)
        -> Seq Scan on grupa (cost=0.00..61.90 rows=745 width=8) (actual time=0.017..1.298 rows=745 loops=1)
            Filter: (rodzaj_zajec = 'w'::bpchar)
        -> Hash (cost=51.17..51.17 rows=4 width=4) (actual time=1.877..1.877 rows=16 loops=1)
            -> Hash Join (cost=13.74..51.17 rows=4 width=4) (actual time=0.247..1.867 rows=16 loops=1)
                Hash Cond: (przedmiot_semestr.kod_przed = przedmiot.kod_przed)
                -> Seq Scan on przedmiot_semestr (cost=0.00..30.47 rows=1847 width=8) (actual time=0.007..0.795 rows=1847 loops=1)
                    -> Hash (cost=13.72..13.72 rows=1 width=4) (actual time=0.181..0.181 rows=1 loops=1)
                        -> Seq Scan on przedmiot (cost=0.00..13.72 rows=1 width=4) (actual time=0.020..0.178 rows=1 loops=1)
                            Filter: (nazwa = 'Bazy danych'::text)
                -> Index Scan using wybor_key on wybor (cost=0.00..1.81 rows=23 width=8) (actual time=0.008..0.060 rows=38 loops=8)
                    Index Cond: (wybor.kod_grupy = grupa.kod_grupy)
            -> Index Scan using uzytkownik_key on uzytkownik (cost=0.00..0.27 rows=1 width=13) (actual time=0.003..0.004 rows=1 loops=307)
                Index Cond: (uzytkownik.kod_uz = wybor.kod_uz)
Total runtime: 6.696 ms
(19 rows)
```

Alternatywne postaci zapytań - przykład

Aby porównać działanie tych wersji możemy je wykonać poprzedzając poleceniem EXPLAIN lub EXPLAIN ANALYZE:

QUERY PLAN

```
-> Hash Join (cost=51.22..115.94 rows=2 width=4) (actual time=1.986..3.522 rows=8 loops=1)
    Hash Cond: (grupa.kod_przed_sem = przedmiot_semestr.kod_przed_sem)
    -> Seq Scan on grupa (cost=0.00..61.90 rows=745 width=8) (actual time=0.017..1.298 rows=745 loops=1)
        Filter: (rodzaj_zajec = 'w'::bpchar)
    -> Hash (cost=51.17..51.17 rows=4 width=4) (actual time=1.877..1.877 rows=16 loops=1)
        -> Hash Join (cost=13.74..51.17 rows=4 width=4) (actual time=0.247..1.867 rows=16 loops=1)
            Hash Cond: (przedmiot_semestr.kod_przed = przedmiot.kod_przed)
            -> Seq Scan on przedmiot_semestr (cost=0.00..30.47 rows=1847 width=8) (actual time=0.007..
0.795 rows=1847 loops=1)
            -> Hash (cost=13.72..13.72 rows=1 width=4) (actual time=0.181..0.181 rows=1 loops=1)
                -> Seq Scan on przedmiot (cost=0.00..13.72 rows=1 width=4) (actual time=0.020..0.178 rows=1
loops=1)
                    Filter: (nazwa = 'Bazy danych'::text)
-> Index Scan using wybor_key on wybor (cost=0.00..1.81 rows=23 width=8) (actual time=0.008..0.060
rows=38 loops=8)
    Index Cond: (wybor.kod_grupy = grupa.kod_grupy)
-> Index Scan using uzytkownik_key on uzytkownik (cost=0.00..0.27 rows=1 width=13) (actual time=0.003..0.004
rows=1 loops=307)
    Index Cond: (uzytkownik.kod_uz = wybor.kod_uz)
Total runtime: 6.696 ms
(19 rows)
```

Alternatywne postaci zapytań - przykład

Aby porównać działanie tych wersji możemy je wykonać poprzedzając poleceniem EXPLAIN lub EXPLAIN ANALYZE:

QUERY PLAN

```
-> Seq Scan on grupa (cost=0.00..61.90 rows=745 width=8) (actual time=0.017..1.298 rows=745 loops=1)
    Filter: (rodzaj_zajec = 'w'::bpchar)
-> Hash (cost=51.17..51.17 rows=4 width=4) (actual time=1.877..1.877 rows=16 loops=1)
    -> Hash Join (cost=13.74..51.17 rows=4 width=4) (actual time=0.247..1.867 rows=16 loops=1)
        Hash Cond: (przedmiot_semestr.kod_przed = przedmiot.kod_przed)
        -> Seq Scan on przedmiot_semestr (cost=0.00..30.47 rows=1847 width=8) (actual time=0.007..
0.795 rows=1847 loops=1)
            -> Hash (cost=13.72..13.72 rows=1 width=4) (actual time=0.181..0.181 rows=1 loops=1)
                -> Seq Scan on przedmiot (cost=0.00..13.72 rows=1 width=4) (actual time=0.020..0.178 rows=1
loops=1)
                    Filter: (nazwa = 'Bazy danych'::text)
-> Index Scan using wybor_key on wybor (cost=0.00..1.81 rows=23 width=8) (actual time=0.008..0.060
rows=38 loops=8)
    Index Cond: (wybor.kod_grupy = grupa.kod_grupy)
-> Index Scan using uzytkownik_key on uzytkownik (cost=0.00..0.27 rows=1 width=13) (actual time=0.003..0.004
rows=1 loops=307)
    Index Cond: (uzytkownik.kod_uz = wybor.kod_uz)
Total runtime: 6.696 ms
(19 rows)
```

Alternatywne postaci zapytań - przykład

Aby porównać działanie tych wersji możemy je wykonać poprzedzając poleceniem EXPLAIN lub EXPLAIN ANALYZE:

QUERY PLAN

```
-> Hash (cost=51.17..51.17 rows=4 width=4) (actual time=1.877..1.877 rows=16 loops=1)
    -> Hash Join (cost=13.74..51.17 rows=4 width=4) (actual time=0.247..1.867 rows=16 loops=1)
        Hash Cond: (przedmiot_semestr.kod_przed = przedmiot.kod_przed)
        -> Seq Scan on przedmiot_semestr (cost=0.00..30.47 rows=1847 width=8) (actual time=0.007..0.795 rows=1847 loops=1)
            -> Hash (cost=13.72..13.72 rows=1 width=4) (actual time=0.181..0.181 rows=1 loops=1)
                -> Seq Scan on przedmiot (cost=0.00..13.72 rows=1 width=4) (actual time=0.020..0.178 rows=1 loops=1)
                    Filter: (nazwa = 'Bazy danych'::text)
-> Index Scan using wybor_key on wybor (cost=0.00..1.81 rows=23 width=8) (actual time=0.008..0.060 rows=38 loops=8)
    Index Cond: (wybor.kod_grupy = grupa.kod_grupy)
-> Index Scan using uzytkownik_key on uzytkownik (cost=0.00..0.27 rows=1 width=13) (actual time=0.003..0.004 rows=1 loops=307)
    Index Cond: (uzytkownik.kod_uz = wybor.kod_uz)
Total runtime: 6.696 ms
(19 rows)
```

Alternatywne postaci zapytań - przykład

Aby porównać działanie tych wersji możemy je wykonać poprzedzając poleceniem EXPLAIN lub EXPLAIN ANALYZE:

QUERY PLAN

```
-> Hash (cost=51.17..51.17 rows=4 width=4) (actual time=1.877..1.877 rows=16 loops=1)
    -> Hash Join (cost=13.74..51.17 rows=4 width=4) (actual time=0.247..1.867 rows=16 loops=1)
        Hash Cond: (przedmiot_semestr.kod_przed = przedmiot.kod_przed)
        -> Seq Scan on przedmiot_semestr (cost=0.00..30.47 rows=1847 width=8) (actual time=0.007..0.795 rows=1847 loops=1)
            -> Hash (cost=13.72..13.72 rows=1 width=4) (actual time=0.181..0.181 rows=1 loops=1)
                -> Seq Scan on przedmiot (cost=0.00..13.72 rows=1 width=4) (actual time=0.020..0.178 rows=1 loops=1)
                    Filter: (nazwa = 'Bazy danych'::text)
    -> Index Scan using wybor_key on wybor (cost=0.00..1.81 rows=23 width=8) (actual time=0.008..0.060 rows=38 loops=8)
        Index Cond: (wybor.kod_grupy = grupa.kod_grupy)
    -> Index Scan using uzytkownik_key on uzytkownik (cost=0.00..0.27 rows=1 width=13) (actual time=0.003..0.004 rows=1 loops=307)
        Index Cond: (uzytkownik.kod_uz = wybor.kod_uz)
```

Total runtime: 6.696 ms

(19 rows)

Alternatywne postaci zapytań - przykład

- JOIN -
- IN -
- EXISTS -

Alternatywne postaci zapytań - przykład

- JOIN - Total runtime: 6.696 ms
- IN -
- EXISTS -

Alternatywne postaci zapytań - przykład

- JOIN - Total runtime: 6.696 ms
- IN - Total runtime: 27.340 ms
- EXISTS -

Alternatywne postaci zapytań - przykład

- JOIN - Total runtime: 6.696 ms
- IN - Total runtime: 27.340 ms
- EXISTS - Total runtime: 2968.883 ms

Alternatywne postaci zapytań - przykład 2

Zapytanie z negacją ma inne możliwe postaci. Pytając o osoby, które nigdy na wykład z BD nie chodziły, możemy:

Alternatywne postaci zapytań - przykład 2

Zapytanie z negacją ma inne możliwe postaci. Pytając o osoby, które nigdy na wykład z BD nie chodziły, możemy:

- wybrać osoby, które nie należą (NOT IN) do grupy uczestników wykładów z Baz danych;

Alternatywne postaci zapytań - przykład 2

Zapytanie z negacją ma inne możliwe postaci. Pytając o osoby, które nigdy na wykład z BD nie chodziły, możemy:

- wybrać osoby, które nie należą (NOT IN) do grupy uczestników wykładów z Baz danych;
- wybrać osoby, dla których nie istnieje (NOT EXISTS) wpis na wykład z Baz danych;

Alternatywne postaci zapytań - przykład 2

Zapytanie z negacją ma inne możliwe postaci. Pytając o osoby, które nigdy na wykład z BD nie chodziły, możemy:

- wybrać osoby, które nie należą (NOT IN) do grupy uczestników wykładów z Baz danych;
- wybrać osoby, dla których nie istnieje (NOT EXISTS) wpis na wykład z Baz danych;
- od wszystkich osób możemy odjąć (EXCEPT) te, które chodziły na wykład z Baz danych;

Alternatywne postaci zapytań - przykład 2

Zapytanie z negacją ma inne możliwe postaci. Pytając o osoby, które nigdy na wykład z BD nie chodziły, możemy:

- wybrać osoby, które nie należą (NOT IN) do grupy uczestników wykładów z Baz danych;
- wybrać osoby, dla których nie istnieje (NOT EXISTS) wpis na wykład z Baz danych;
- od wszystkich osób możemy odjąć (EXCEPT) te, które chodziły na wykład z Baz danych;
- złączyć lewostronnie (LEFT JOIN) wszystkie osoby z wpisami na wykład z Baz danych i znaleźć te, które z niczym się nie połączły (np. Wybor.kod_uz IS NULL).

Alternatywne postaci zapytań - przykład 2

Wybieramy osoby, które nie należą (NOT IN) do grupy uczestników wykładów z Baz danych.

Alternatywne postaci zapytań - przykład 2

Wybieramy osoby, które nie należą (NOT IN) do grupy uczestników wykładów z Baz danych.

```
select nazwisko from użytkownik  
where kod_uz not in  
(select wybór.kod_uz from wybór join grupa  
using(kod_grupy) join przedmiot_semestr using  
(kod_przed_sem) join przedmiot using  
(kod_przed) where nazwa='Bazy danych' and  
rodzaj_zajec='w');
```

Alternatywne postaci zapytań - przykład 2

Wybieramy osoby, które nie należą (**NOT IN**) do grupy uczestników wykładów z Baz danych.

```
select nazwisko from użytkownik  
where kod_uz not in  
(select wybór.kod_uz from wybór join grupa  
using(kod_grupy) join przedmiot_semestr using  
(kod_przed_sem) join przedmiot using  
(kod_przed) where nazwa='Bazy danych' and  
rodzaj_zajec='w');
```

Alternatywne postaci zapytań - przykład 2

Wybieramy osoby, które nie należą (**NOT IN**) do grupy uczestników wykładów z Baz danych.

```
select nazwisko from użytkownik  
where kod_uz not in  
(select wybor.kod_uz from wybor join grupa  
using(kod_grupy) join przedmiot_semestr using  
(kod_przed_sem) join przedmiot using  
(kod_przed) where nazwa='Bazy danych' and  
rodzaj_zajec='w');
```

Total runtime: 7.396 ms

Alternatywne postaci zapytań - przykład 2

Wybieramy osoby, dla których nie istnieje (NOT EXISTS) wpis na wykład z Baz danych.

Alternatywne postaci zapytań - przykład 2

Wybieramy osoby, dla których nie istnieje (NOT EXISTS) wpis na wykład z Baz danych.

```
select nazwisko from użytkownik u
where not exists(select * from wybor join
grupa using(kod_grupy) join przedmiot_semestr
using(kod_przed_sem) join przedmiot using
(kod_przed) where nazwa='Bazy danych' and
rodzaj_zajec='w' and wybor.kod_uz=u.kod_uz);
```

Alternatywne postaci zapytań - przykład 2

Wybieramy osoby, dla których nie istnieje (**NOT EXISTS**) wpis na wykład z Baz danych.

```
select nazwisko from użytkownik u
where not exists(select * from wybor join
grupa using(kod_grupy) join przedmiot_semestr
using(kod_przed_sem) join przedmiot using
(kod_przed) where nazwa='Bazy danych' and
rodzaj_zajec='w' and wybor.kod_uz=u.kod_uz);
```

Alternatywne postaci zapytań - przykład 2

Wybieramy osoby, dla których nie istnieje (**NOT EXISTS**) wpis na wykład z Baz danych.

```
select nazwisko from użytkownik u
where not exists(select * from wybor join
grupa using(kod_grupy) join przedmiot_semestr
using(kod_przed_sem) join przedmiot using
(kod_przed) where nazwa='Bazy danych' and
rodzaj_zajec='w' and wybor.kod_uz=u.kod_uz);
```

Total runtime: 8.641 ms

Alternatywne postaci zapytań - przykład 2

Od wszystkich osób możemy odjąć (EXCEPT) te, które chodziły na wykład z Baz danych.

Alternatywne postaci zapytań - przykład 2

Od wszystkich osób możemy odjąć (EXCEPT) te, które chodziły na wykład z Baz danych.

```
(select nazwisko, kod_uz from uzytkownik)
except
(select nazwisko, kod_uz from uzytkownik join
wybor using(kod_uz) join grupa using
(kod_grupy) join przedmiot_semestr using
(kod_przed_sem) join przedmiot using
(kod_przed) where nazwa='Bazy danych' and
rodzaj_zajec='w');
```

Alternatywne postaci zapytań - przykład 2

Od wszystkich osób możemy odjąć (EXCEPT) te, które chodziły na wykład z Baz danych.

```
(select nazwisko, kod_uz from uzytkownik)
except
(select nazwisko, kod_uz from uzytkownik join
wybor using(kod_uz) join grupa using
(kod_grupy) join przedmiot_semestr using
(kod_przed_sem) join przedmiot using
(kod_przed) where nazwa='Bazy danych' and
rodzaj_zajec='w');
```

Alternatywne postaci zapytań - przykład 2

Od wszystkich osób możemy odjąć (EXCEPT) te, które chodziły na wykład z Baz danych.

```
(select nazwisko, kod_uz from uzytkownik)
except
(select nazwisko, kod_uz from uzytkownik join
wybor using(kod_uz) join grupa using
(kod_grupy) join przedmiot_semestr using
(kod_przed_sem) join przedmiot using
(kod_przed) where nazwa='Bazy danych' and
rodzaj_zajec='w');
```

Total runtime: 19.039 ms

Alternatywne postaci zapytań - przykład 2

Łączymy lewostronnie (LEFT JOIN) wszystkie osoby z wpisami na wykład z Baz danych i znajdujemy te osoby, które z niczym się nie połączły.

Total runtime: 8.606 ms

Alternatywne postaci zapytań - przykład 2

Łączymy lewostronnie (LEFT JOIN) wszystkie osoby z wpisami na wykład z Baz danych i znajdujemy te osoby, które z niczym się nie połączły.

```
select nazwisko from użytkownik left join  
(wybor.kod_uz from wybor join grupa using  
(kod_grupy) join przedmiot_semestr using  
(kod_przed_sem) join przedmiot using  
(kod_przed) where nazwa='Bazy danych' and  
rodzaj_zajec='w')) AA using(kod_uz) where AA.  
kod_uz IS NULL;
```

Total runtime: 8.606 ms

Alternatywne postaci zapytań - przykład 2

Łączymy lewostronnie (LEFT JOIN) wszystkie osoby z wpisami na wykład z Baz danych i znajdujemy te osoby, które z niczym się nie połączły.

```
select nazwisko from użytkownik left join  
(wybor.kod_uz from wybor join grupa using  
(kod_grupy) join przedmiot_semestr using  
(kod_przed_sem) join przedmiot using  
(kod_przed) where nazwa='Bazy danych' and  
rodzaj_zajec='w')) AA using(kod_uz) where AA.  
kod_uz IS NULL;
```

Total runtime: 8.606 ms

Alternatywne postaci zapytań - przykład 2

Łączymy lewostronnie (LEFT JOIN) wszystkie osoby z wpisami na wykład z Baz danych i znajdujemy te osoby, które z niczym się nie połączły.

```
select nazwisko from użytkownik left join  
(wybor.kod_uz from wybor join grupa using  
(kod_grupy) join przedmiot_semestr using  
(kod_przed_sem) join przedmiot using  
(kod_przed) where nazwa='Bazy danych' and  
rodzaj_zajec='w')) AA using(kod_uz) where AA.  
kod_uz IS NULL;
```

Total runtime: 8.606 ms

Alternatywne postaci zapytań - przykład 2

Łączymy lewostronnie (LEFT JOIN) wszystkie osoby z wpisami na wykład z Baz danych i znajdujemy te osoby, które z niczym się nie połączły.

```
select nazwisko from użytkownik left join  
(wybor.kod_uz from wybor join grupa using  
(kod_grupy) join przedmiot_semestr using  
(kod_przed_sem) join przedmiot using  
(kod_przed) where nazwa='Bazy danych' and  
rodzaj_zajec='w')) AA using(kod_uz) where AA.  
kod_uz IS NULL;
```

Total runtime: 8.606 ms

Alternatywne postaci zapytań - przykład 2

Łączymy lewostronnie (LEFT JOIN) wszystkie osoby z wpisami na wykład z Baz danych i znajdujemy te osoby, które z niczym się nie połączły.

```
select nazwisko from użytkownik left join  
(wybor.kod_uz from wybor join grupa using  
(kod_grupy) join przedmiot_semestr using  
(kod_przed_sem) join przedmiot using  
(kod_przed) where nazwa='Bazy danych' and  
rodzaj_zajec='w')) AA using(kod_uz) where AA.  
kod_uz IS NULL;
```

Total runtime: 8.606 ms

Podzapytania w klauzuli FROM

Podzapytania w klauzuli FROM

O tym było właśnie przed chwilą!

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...
- Wyrażenia może odwoływać się do innych tabel, ale nie do innych aliasów, czyli zmiennych krotkowych;

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...
- Wyrażenia może odwoływać się do innych tabel, ale nie do innych aliasów, czyli zmiennych krotkowych;
- Wyrażenia można łączyć przez JOIN, UNION,

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...
- Wyrażenia może odwoływać się do innych tabel, ale nie do innych aliasów, czyli zmiennych krotkowych;
- Wyrażenia można łączyć przez JOIN, UNION,

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...
- Wyrażenia może odwoływać się do innych tabel, ale nie do innych aliasów, czyli zmiennych krotkowych;
- Wyrażenia można łączyć przez JOIN, UNION,
- Wyrażenia muszą mieć aliasy!

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...
- Wyrażenia może odwoływać się do innych tabel, ale nie do innych aliasów, czyli zmiennych krotkowych;
- Wyrażenia można łączyć przez JOIN, UNION,
- Wyrażenia muszą mieć **aliasy!**

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...
- Wyrażenia może odwoływać się do innych tabel, ale nie do innych aliasów, czyli zmiennych krotkowych;
- Wyrażenia można łączyć przez JOIN, UNION,
- Wyrażenia muszą mieć aliasy!
- Wyrażenia pozwalają nam zapisać w SQL praktycznie dowolnie skomplikowane zapytanie algebry relacji lub relacyjnego rachunku krotek.

Podzapytania w klauzuli FROM

```
SELECT nazwisko FROM Uzytkownik  
    JOIN (wpisy na wykład z BD) AA USING (kod_uz)  
    JOIN (grupy prowadzone przez PKA) BB ON (...)  
WHERE semestr>5;
```

- Wyrażenia są obliczane przed rozpoczęciem obliczania klauzul WHERE i SELECT...
- Wyrażenia może odwoływać się do innych tabel, ale nie do innych aliasów, czyli zmiennych krotkowych;
- Wyrażenia można łączyć przez JOIN, UNION,
- Wyrażenia muszą mieć aliasy!
- Wyrażenia pozwalają nam zapisać w SQL praktycznie dowolnie skomplikowane zapytanie algebry relacji lub relacyjnego rachunku krotek.
- Nie należy przesadzać z wyrażeniami tabelowymi!

Grupowanie i funkcje agregujące

Grupowanie i funkcje agregujące

W zapytaniu SQL w klawiszu `SELECT` można użyć funkcji `MIN`, `MAX`, `SUM`, `AVG`, `COUNT`, która zwraca odpowiednio minimum,... z określonej kolumny (może być wyliczana).

Grupowanie i funkcje agregujące

W zapytaniu SQL w klawiszu `SELECT` można użyć funkcji `MIN`, `MAX`, `SUM`, `AVG`, `COUNT`, która zwraca odpowiednio minimum,... z określonej kolumny (może być wyliczana)

```
SELECT max(kod_grupy) FROM Grupa;
```

Grupowanie i funkcje agregujące

W zapytaniu SQL w klawiszu `SELECT` można użyć funkcji `MIN`, `MAX`, `SUM`, `AVG`, `COUNT`, która zwraca odpowiednio minimum,... z określonej kolumny (może być wyliczana)

```
SELECT max(kod_grupy) FROM Grupa;
```

Grupowanie i funkcje agregujące

W zapytaniu SQL w kolumnie SELECT można użyć funkcji MIN, MAX, SUM, AVG, COUNT, która zwraca odpowiednio minimum,... z określonej kolumny (może być wyliczana)

```
SELECT max(kod_grupy) FROM Grupa;
```

```
SELECT count(*) FROM wybor JOIN grupa USING  
(kod_grupy) JOIN przedmiot_semestr USING  
(kod_przed_sem) JOIN przedmiot USING(kod_przed)  
WHERE nazwa='Bazy danych' AND semestr_id=39;
```

Grupowanie i funkcje agregujące

W zapytaniu SQL w kolumnie SELECT można użyć funkcji MIN, MAX, SUM, AVG, COUNT, która zwraca odpowiednio minimum,... z określonej kolumny (może być wyliczana)

```
SELECT max(kod_grupy) FROM Grupa;
```

```
SELECT count(*) FROM wybor JOIN grupa USING  
(kod_grupy) JOIN przedmiot_semestr USING  
(kod_przed_sem) JOIN przedmiot USING(kod_przed)  
WHERE nazwa='Bazy danych' AND semestr_id=39;
```

Grupowanie i funkcje agregujące

W zapytaniu SQL w kolumnie SELECT można użyć funkcji MIN, MAX, SUM, AVG, COUNT, która zwraca odpowiednio minimum,... z określonej kolumny (może być wyliczana)

```
SELECT max(kod_grupy) FROM Grupa;
```

```
SELECT count(*) FROM wybor JOIN grupa USING  
(kod_grupy) JOIN przedmiot_semestr USING  
(kod_przed_sem) JOIN przedmiot USING(kod_przed)  
WHERE nazwa='Bazy danych' AND semestr_id=39;
```

```
SELECT count(grupa.kod_uz) FROM grupa JOIN  
przedmiot_semestr USING(kod_przed_sem) JOIN  
przedmiot USING(kod_przed) WHERE nazwa='Bazy  
danych';
```

Grupowanie i funkcje agregujące

W zapytaniu SQL w kolumnie SELECT można użyć funkcji MIN, MAX, SUM, AVG, COUNT, która zwraca odpowiednio minimum,... z określonej kolumny (może być wyliczana)

```
SELECT max(kod_grupy) FROM Grupa;
```

```
SELECT count(*) FROM wybor JOIN grupa USING  
(kod_grupy) JOIN przedmiot_semestr USING  
(kod_przed_sem) JOIN przedmiot USING(kod_przed)  
WHERE nazwa='Bazy danych' AND semestr_id=39;
```

```
SELECT count(grupa.kod_uz) FROM grupa JOIN  
przedmiot_semestr USING(kod_przed_sem) JOIN  
przedmiot USING(kod_przed) WHERE nazwa='Bazy  
danych';
```

Grupowanie i funkcje agregujące

W zapytaniu SQL w kolumnie SELECT można użyć funkcji MIN, MAX, SUM, AVG, COUNT, która zwraca odpowiednio minimum,... z określonej kolumny (może być wyliczana)

```
SELECT max(kod_grupy) FROM Grupa;
```

```
SELECT count(*) FROM wybor JOIN grupa USING  
(kod_grupy) JOIN przedmiot_semestr USING  
(kod_przed_sem) JOIN przedmiot USING(kod_przed)  
WHERE nazwa='Bazy danych' AND semestr_id=39;
```

```
SELECT count(DISTINCT grupa.kod_uz) FROM grupa JOIN  
przedmiot_semestr USING(kod_przed_sem) JOIN  
przedmiot USING(kod_przed) WHERE nazwa='Bazy  
danych';
```

Grupowanie i funkcje agregujące

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...)

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

```
SELECT *
FROM Wybor JOIN Grupa Using(kod_grupy)
WHERE rodzaj_zajec='w';
```

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

```
SELECT *      -- to są zapisy do grup wykładowych  
FROM Wybor JOIN Grupa Using(kod_grupy)  
WHERE rodzaj_zajec='w';
```

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

```
SELECT *
FROM Wybor JOIN Grupa Using(kod_grupy)
WHERE rodzaj_zajec='w';
```

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

```
SELECT *
FROM Wybor JOIN Grupa Using(kod_grupy)
WHERE rodzaj_zajec='w'
GROUP BY Wybor.kod_grupy;
```

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

```
SELECT *
FROM Wybor JOIN Grupa Using(kod_grupy)
WHERE rodzaj_zajec='w'
GROUP BY Wybor.kod_grupy; -- tworzymy grupy z krotek
                           o tych samych wartościach
                           Wybor.kod_grupy
```

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

```
SELECT count(*) -- dla każdej grupy liczymy krotki  
FROM Wybor JOIN Grupa Using(kod_grupy)  
WHERE rodzaj_zajec='w'  
GROUP BY Wybor.kod_grupy; -- tworzymy grupy dla grup
```

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

```
SELECT wybor.kod_grupy, count(*) -- warto dodać id grupy  
FROM Wybor JOIN Grupa Using(kod_grupy)  
WHERE rodzaj_zajec='w'  
GROUP BY Wybor.kod_grupy; -- tworzymy grupy dla grup
```

Grupowanie i funkcje agregujące

Można pogrupować zawartość tabeli wynikowej i dla każdej grupy wyliczyć agregat (MIN, MAX, ...).

Policzymy liczbę zapisanych do każdej grupy wykładowej.

```
SELECT wybor.kod_grupy, count(*) -- warto dodać id grupy  
FROM Wybor JOIN Grupa Using(kod_grupy)  
WHERE rodzaj_zajec='w'  
GROUP BY Wybor.kod_grupy -- tworzymy grupy dla grup  
HAVING count(*)>100; -- możemy wybrać tylko duże grupy
```

Grupowanie i funkcje agregujące

1. Tworzymy "zwykłe" zapytanie (bez agregatów i GROUP BY).
2. Dodajemy klauzulę GROUP BY A,B,... co dzieli relację wynikową na grupy wg jednakowych wartości A,B,... atrybutów grupowania.
3. Możemy dodać klauzulę HAVING określającą, które grupy przejdą dalej; w warunku klauzuli możemy "pytać" o wartości agregatów dla grupy i/lub atrybuty grupowania.
4. W klauzuli SELECT dla każdej grupy możemy do wyniku przekazać wartości atrybutów grupowania lub agregatów.

```
SELECT nazwisko count(*)
FROM użytkownik JOIN grupa USING(kod_uz) JOIN
wybor USING (kod_grupy)
WHERE rodzaj_zajec='w'
GROUP BY nazwisko, grupa.kod_grupy
HAVING count(*)>100
```

Grupowanie i funkcje agregujące

1. Tworzymy "zwykłe" zapytanie (bez agregatów i GROUP BY).
2. Dodajemy klauzulę GROUP BY A,B,... dzielącą relację wynikową na grupy wg jednakowych wartości A,B,... atrybutów grupowania.
3. Możemy dodać klauzulę HAVING określającą, które grupy przejdą dalej; w warunku klauzuli możemy "pytać" o wartości agregatów dla grupy i/lub atrybuty grupowania.
4. W klauzuli SELECT dla każdej grupy możemy do wyniku przekazać wartości atrybutów grupowania lub agregatów.

```
SELECT nazwisko count(*)  
FROM użytkownik JOIN grupa USING(kod_uz) JOIN  
wybor USING (kod_grupy)  
WHERE rodzaj_zajec='w'  
GROUP BY nazwisko, grupa.kod_grupy  
HAVING count(*)>100
```

Grupowanie i funkcje agregujące

1. Tworzymy "zwykłe" zapytanie (bez agregatów i GROUP BY).
2. Dodajemy klauzulę GROUP BY A,B,... dzielącą relację wynikową na grupy wg jednakowych wartości A,B,... atrybutów grupowania.
3. Możemy dodać klauzulę HAVING określającą, które grupy przejdą dalej; w warunku klauzuli możemy "pytać" o wartości agregatów dla grupy i/lub atrybuty grupowania.
4. W klauzuli SELECT dla każdej grupy możemy do wyniku przekazać wartości atrybutów grupowania lub agregatów.

```
SELECT nazwisko count(*)
FROM użytkownik JOIN grupa USING(kod_uz) JOIN
wybor USING (kod_grupy)
WHERE rodzaj_zajec='w'
GROUP BY nazwisko, grupa.kod_grupy
HAVING count(*)>100
```

Grupowanie i funkcje agregujące

1. Tworzymy "zwykłe" zapytanie (bez agregatów i GROUP BY).
2. Dodajemy klauzulę GROUP BY A,B,... dzielącą relację wynikową na grupy wg jednakowych wartości A,B,... atrybutów grupowania.
3. Możemy dodać klauzulę HAVING określającą, które grupy przejdą dalej; w warunku klauzuli możemy "pytać" o wartości agregatów dla grupy i/lub atrybuty grupowania.
4. W klauzuli SELECT dla każdej grupy możemy do wyniku przekazać wartości atrybutów grupowania lub agregatów.

```
SELECT nazwisko, count(*)
FROM użytkownik JOIN grupa USING(kod_uz) JOIN
wybor USING (kod_grupy)
WHERE rodzaj_zajec='w'
GROUP BY nazwisko, grupa.kod_grupy
HAVING count(*)>100
```

Upraszczanie zapytań

Upraszczanie zapytań

Podzapytania w klauzuli FROM

`SELECT ... FROM (...) AA, R, S WHERE F(R,S,AA)`

-- takie sobie uproszczenie

Tabele tymczasowe

`CREATE TEMP TABLE AA (...); INSERT INTO AA SELECT...`

`SELECT ... FROM AA, R, S WHERE F(R,S,AA)`

-- tabel można używać wielokrotnie; są kasowane na koniec sesji

Perspektywy (widoki)

`CREATE VIEW AA AS SELECT ...`

`SELECT ... FROM AA, R, S WHERE F(R,S,AA)`

-- perspektywy są obiektem trwałym, ale wirtualnym

Instrukcja WITH

`WITH AA AS (...)`

`SELECT ... FROM AA, BB, R, S WHERE F(R,S,AA)`

-- na pierwszy rzut oka wygląda jak podzapytanie, ale pozwala na rekursję!

Upraszczanie zapytań - TEMP TABLE

```
CREATE TEMPORARY TABLE trudneBD LIKE(zad);
INSERT INTO trudneBD
    SELECT zad.*
        FROM zad JOIN klas ON zad.id=klad.zad JOIN kat ON kat.id=klas.kat
        WHERE trudnosc=100 AND trafnosc>=50 AND kat.nazwa='Bazy
danych';
```

```
CREATE TEMPORARY TABLE masters2011
    (id int, imie text, nazwisko text, wsp real);
INSERT INTO masters2010
    SELECT id,imie,nazwisko,avg(wynik)
        FROM osoba JOIN wynik ON id=osoba
        WHERE EXTRACT(year FROM dataWyn)=2011
        GROUP BY id, imie, nazwisko
        HAVING count(DIST zad.id)>100 AND avg(wynik)>75;
```

Upraszczanie zapytań - TEMP TABLE

Korzystając z `trudneBD` i `masters2011` możemy znaleźć "mistrzów 2011", którzy robili trudne zadania z baz danych:

```
SELECT masters2011.id, imie, nazwisko, avg(wynik)
FROM masters2011 JOIN wynik ON id=osoba JOIN
    trudneBD ON trudneBD.id=wynik.zad
GROUP BY masters2011.id, imie, nazwisko
ORDER BY 4 DESC;
```

Upraszczanie zapytań - **VIEWS**

```
CREATE VIEW trudneBD AS  
SELECT zad.*  
FROM zad JOIN klas ON zad.id=klad.zad JOIN kat ON kat.id=klas.kat  
WHERE trudnosc=100 AND trafnosc>=50 AND kat.nazwa='Bazy  
danych';
```

```
CREATE VIEW masters2011(id, imie, nazwisko, wsp) AS  
SELECT id,imie,nazwisko,avg(wynik)  
FROM osoba JOIN wynik ON id=osoba  
WHERE EXTRACT(year FROM dataWyn)=2011  
GROUP BY id, imie, nazwisko  
HAVING count(DIST zad.id)>100 AND avg(wynik)>75;
```

Upraszczanie zapytań - **VIEWS**

Korzystając z **trudneBD** i **masters2011** możemy znaleźć "mistrzów 2011", którzy robili trudne zadania z baz danych:

```
SELECT masters2011.id, imie, nazwisko, avg(wynik)
FROM masters2011 JOIN wynik ON id=osoba JOIN
    trudneBD ON trudneBD.id=wynik.zad
GROUP BY masters2011.id, imie, nazwisko
ORDER BY 4 DESC;
```

TEMP TABLE versus VIEWS

id	nazwisko	imie	wsp
.	.	.	.

id	nazwisko	imie	wsp
.	.	.	.

id	imie	...
1	Anna	...
2	Basia	...

osoba	zad	czasWyk	wynik
1	13	2011-...	80
2	15	2011-...	70

TEMP TABLE versus VIEWS

id	nazwisko	imie	wsp
1	Abacka	Anna	80

id	nazwisko	imie	wsp
1	Abacka	Anna	80

id	imie	...
1	Anna	...
2	Basia	...

osoba	zad	czasWyk	wynik
1	13	2011-...	80
2	15	2011-...	70

TEMP TABLE versus VIEWS

id	nazwisko	imie	wsp
1	Abacka	Anna	80

id	nazwisko	imie	wsp
1	Abacka	Anna	80

id	imie	...
1	Anna	...
2	Basia	...

osoba	zad	czasWyk	wynik
1	13	2011-...	80
2	15	2011-...	70
2	13	2011-...	100

TEMP TABLE versus VIEWS

id	nazwisko	imie	wsp
1	Abacka	Anna	80

id	nazwisko	imie	wsp
1	Abacka	Anna	80
2	Babacka	Basia	85

id	imie	...
1	Anna	...
2	Basia	...

osoba	zad	czasWyk	wynik
1	13	2011-...	80
2	15	2011-...	70
2	13	2011-...	100

TEMP TABLE versus VIEWS

id	nazwisko	imie	wsp
1	Abacka	Anna	80

id	nazwisko	imie	wsp
1	Abacka	Anna	80
2	Babacka	Basia	85

id	imie	...
1	Anna	...
2	Basia	...

osoba	zad	czasWyk	wynik
1	13	2011-...	80
2	15	2011-...	70
2	13	2011-...	100
1	15	2011-...	30

TEMP TABLE versus VIEWS

id	nazwisko	imie	wsp
1	Abacka	Anna	80

id	nazwisko	imie	wsp
1	Abacka	Anna	80
2	Babacka	Basia	85

id	imie	...
1	Anna	...
2	Basia	...

osoba	zad	czasWyk	wynik
1	13	2011-...	80
2	15	2011-...	70
2	13	2011-...	100
1	15	2011-...	30

Klauzula WITH

```
WITH trudneBD AS (SELECT zad.* FROM zad JOIN klas ON zad.id=klad.zad JOIN kat ON kat.id=klas.kat WHERE trudnosc=100 AND trafnosc>=50 AND kat.nazwa='Bazy danych'),  
masters2011 AS (SELECT id,imie,nazwisko,avg(wynik) AS "wsp"  
FROM osoba JOIN wynik ON id=osoba  
WHERE EXTRACT(year FROM dataWyn)=2011  
GROUP BY id, imie, nazwisko  
HAVING count(DISTINCT zad.id)>100 AND avg(wynik)>75)
```

```
SELECT masters2011.id, imie, nazwisko, avg(wynik)  
FROM masters2011 JOIN wynik ON id=osoba JOIN  
trudneBD ON trudneBD.id=wynik.zad  
GROUP BY masters2011.id, imie, nazwisko  
ORDER BY 4 DESC;
```

Klauzula WITH - rekursja

WITH subkat AS

(SELECT id AS "pod", nadziedna AS "nad" FROM kat
UNION

SELECT id AS "pod", nad

FROM kat JOIN subkat ON (nadziedna=pod))

SELECT pod FROM subkat JOIN kat ON (nad=id)

WHERE nazwa='Bazy danych';

Klauzula WITH - rekursja

WITH **subkat** AS

(SELECT id AS "pod", nadziedna AS "nad" FROM kat
UNION

SELECT id AS "pod", nad

FROM **kat** JOIN **subkat** ON (nadziedna=pod))

SELECT pod FROM subkat JOIN kat ON (nad=id)

WHERE nazwa='Bazy danych';

Klauzula WITH - rekursja

```
WITH subkat AS
  (SELECT id AS "pod", nadziedna AS "nad" FROM kat
   UNION
   SELECT id AS "pod", nad
     FROM kat JOIN subkat ON (nadziedna=pod))
  SELECT pod FROM subkat JOIN kat ON (nad=id)
 WHERE nazwa='Bazy danych';
```

KONIEC