Katedra
Informatyki i Automatyki
Politechniki Rzeszowskiej

Bazy Danych

1. SQL

Opracował: Maciej Penar

Spis treści

1. Zanim zaczniemy

Zrelaksować się i przyswoić sobie teorię dot. Algebry relacji.

Materiały:

- Google: https://www.google.pl/search?q=algebra+relacji&oq=algebra+relacji
- Podstawowy kurs systemów baz danych, rozdział 2 oraz 5.2, J. Ullman, J. Widom

Oprogramowanie:

- SQLite: https://www.sqlite.org/index.html
- SQLite (link 2):
- https://github.com/mpenarprz/BazyDanychI4/tree/master/Laboratorium/tools
- GUI do SQLite: http://sqlitebrowser.org/

DRZEWO OPERATORÓW ALGEBRY RELACJI

Jak komuś nie chce się otwierać książki „Podstawowy kurs systemów baz danych" to zamieszczam krótkie info o co chodzi z zapytaniami w „algebrze relacji" w formie drzewa operatorów. Trzeba znać operatory żeby zrozumieć o co tu chodzi.

Załóżmy relację np. Ziemniaki(Dojrzały, Rozmiar, Waga)

Niech atrybut Dojrzały opisuje czy ziemniak należący do relacji jest dojrzały lub nie (true/false). Z kolei atrybut Rozmiar niech ma zdefiniowaną dziedzinę {„Mały", „Średni", „Duży"} i opisuje jakościowo naszego ziemniaka. Atrybut Waga opisuje ilościowo ziemniaka. Prawidłowe wartości są większe od 0 (Waga >=0) – przyjmijmy że to waga gramach.

Załóżmy że instancja relacji Ziemniaki to np.:

Ziemniaki		
Dojrzały	**Rozmiar**	**Waga**
True	Duży	180
True	Średni	120
True	Średni	160
False	Mały	50

Zastanówmy się nad znaczeniem operatorów algebry relacji – otóż wyznaczają one pewien podzbiór relacji nad którą operują. I tak wyrażenie $\pi(Ziemniaki)_{Dojrzały}$ wyznacza podzbiór relacji Ziemniaki zawierający jedynie atrybut Dojrzały. Instancja (wystąpienie) relacji: $\pi(Ziemniaki)_{Dojrzały}$ to:

$\boldsymbol{\pi(Ziemniaki)_{Dojrzały}}$
Dojrzały
True
True
True
False

Zapis w formie: $\pi(Ziemniaki)_{Dojrzały}$ nazywamy liniowym

Nas będzie interesował zapis w formie drzewa:

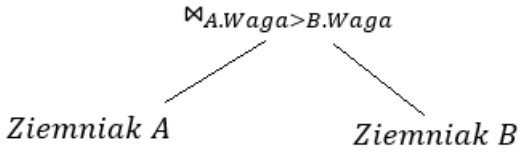
Weźmy bardziej skomplikowane zapytanie np.

$\pi(\sigma(Ziemniaki)_{Waga>150})_{Dojrzały,Rozmiar}$ które wyznacza podzbiór relacji Ziemniaki zawierający jedynie atrybut Dojrzały oraz Rozmiar, w którym ziemniaki ważą 150g.

To zapis w formie drzewa przyjąłby postać:

Niektóre operatory są dwuargumentowe np. ∩,∪, ⋈ co powoduje rozgałęzianie się drzewa. Weźmy ultra trudne zapytanie np.⋈ $(Ziemniak\ A, Ziemniak\ B)_{A.Waga>B.Waga}$ które wybiera wszystkie pary ziemniaków których waga pierwszego jest większa od wagi drugiego.

Drzewo wygląda tak:



Ciekawostka: relacja wynikowa:

Ziemniaki					
A.Dojrzały	**A.Rozmiar**	**A.Waga**	**B.Dojrzały**	**B.Rozmiar**	**B.Waga**
True	Duży	180	True	Średni	160
True	Duży	180	True	Średni	120
True	Duży	180	False	Mały	50
True	Średni	160	True	Średni	120
True	Średni	160	False	Mały	50
True	Średni	120	False	Mały	50

Do Algebry Relacji i jej związku z wykonywaniem zapytań wrócimy na liście 4.

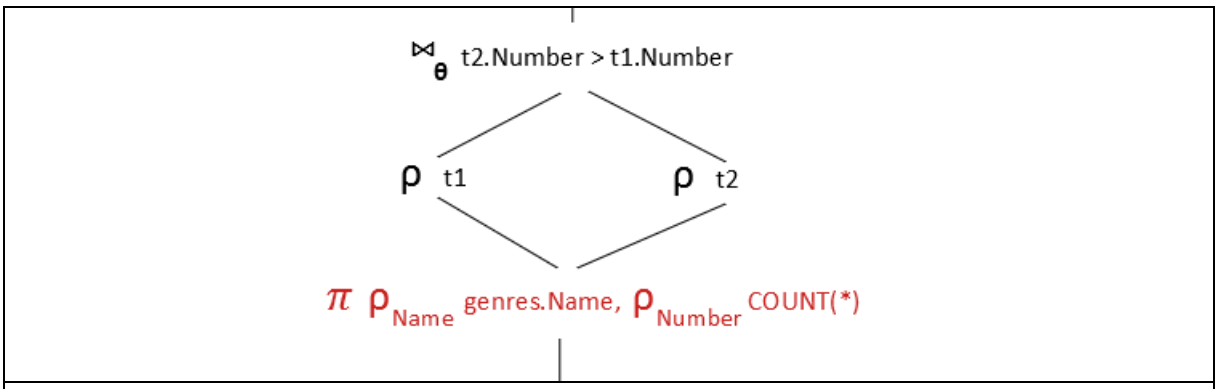
Znane operatory to:

Nazwa	***Symbol***	***SQL***
Projekcja	π	SELECT
Selekcja	σ	WHERE/HAVING
Przemianowanie	ρ	SELECT … **AS**
Eliminacja Duplikatów	δ	DISTINCT
Okno	ω	WINDOW/OVER
Unia	$\cup$	UNION (ALL)
Przecięcie	$\cap$	INTERSECT (ALL)
Różnica	$-$	MINUS/EXCEPT (ALL)
Grupowanie	γ	GROUP BY
Złączenie	$\bowtie$	(INNER/OUTER/LEFT/RIGHT) JOIN
Sortowanie	τ	ORDER BY

DRZEWO OPERATORÓW ALGEBRY RELACJI - UWAGI

Pamiętajcie, że:

1) **drzewo** operatorów to **drzewo** (a nie graf). To oznacza, że jeśli wyjdzie Wam rysunek taki jak:



Przykład 1: cykl w grafie operatorów

… i powstaje cykl, to nie mamy drzewa, a raczej graf – czyli jest błąd.

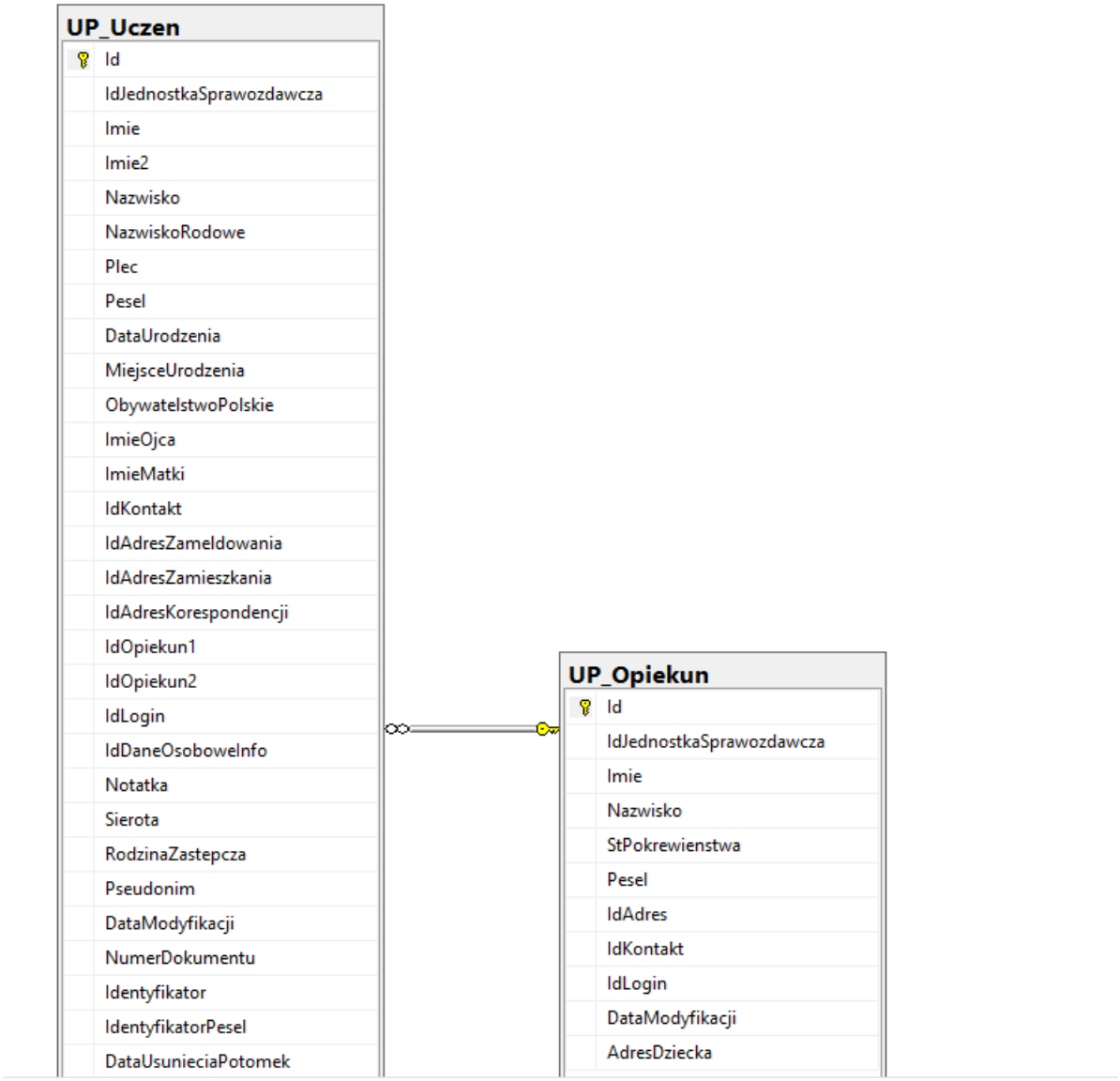
2) Wynikiem działania każdego operatora AR (niezależnie czy jest to operator jednoargumentowy [np. selekcja], czy dwuargumentowy [np. złączenie]) jest relacja. Przez to, że zapytanie SQL-owe składa się z operatorów Algebry Relacji to zapytanie **SQL-a możemy też traktować jako relację**.

To oznacza, że dowolny argument dowolnego operatora AR możemy podmienić na SQL.

3) Zawsze liściami drzewa operatorów Algebry Relacji są **tabele** (relacje)

DRZEWO OPERATORÓW ALGEBRY RELACJI, A SQL

Załóżmy tabele Opiekun oraz Uczeń – zawierającą dane dot. opiekunów uczniów oraz dane opisujące uczniów:



Konwersja zapytań na drzewa wykonujemy za pomocą tabeli powyżej. Poniżej zamieszczam kilka przykładów drzewek.

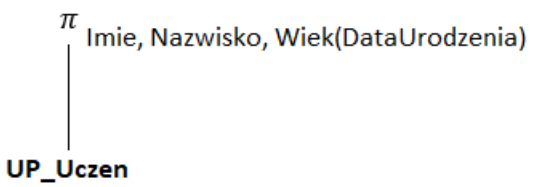
PRZYKŁAD 2

Najprostszym zapytaniem jest scan tabeli. To jest całe drzewo – tylko trzeba zajrzeć do źródła. Nie ma żadnych dodatkowych ograniczeń (bo *):

SELECT * FROM UP_Uczen	**UP_Uczen**
Przykład 2: najprostsze zapytanie	

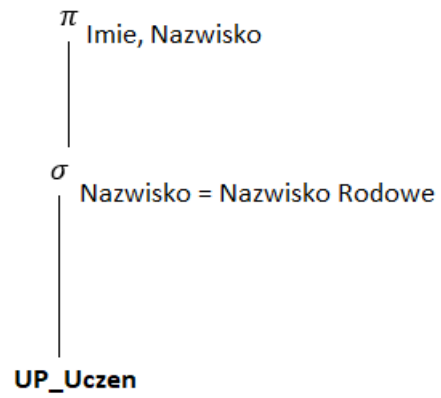
PRZYKŁAD 3

Bardziej skomplikowane jest użycie operatora projekcji – słowo kluczowe SELECT z podanymi argumentami. Operator projekcji ma na wejściu jeden argument będący relacją i w tym przypadku jest to UP_Uczen – zazwyczaj jest to relacja będąca wynikiem złączenia wszystkich pozostałych relacji (klauzula FROM) – ale jest tu pewna dowolność.

SELECT Imie, Nazwisko, Wiek(DataUrodzenia) FROM UP_Uczen	
Przykład 3: projekcja	

PRZYKŁAD 4

Operator selekcji (słowo kluczowe WHERE, lub HAVING) powoduje utworzenie następującego drzewa:

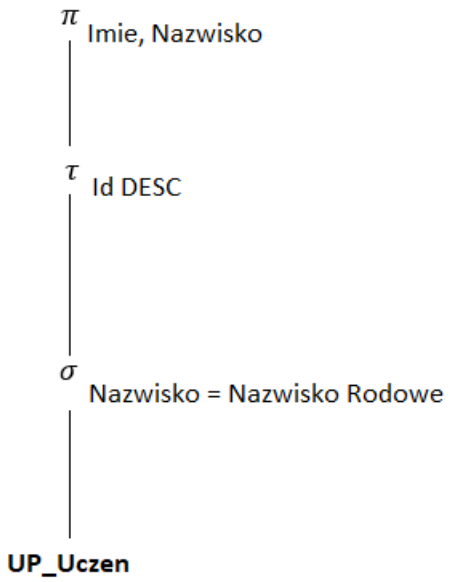
SELECT Imie, Nazwisko FROM UP_Uczen WHERE NazwiskoRodowe = Nazwisko	
Przykład 4: projekcja i selekcja	

Kiedyś, wyżej był przykład z ziemniakami i były podane 2 drzewa – jedno z nich miało najpierw projekcje, a potem selekcje. To była herezja. Po zastosowaniu projekcji tracimy informacje na temat pewnych kolumn. W tym przykładzie po zastosowaniu projekcji mamy tylko Imie i Nazwisko – dlatego przeniesienie warunku PO projekcji jest niemożliwe – bo warunek dot. Nazwiska Rodowego.

PRZYKŁAD 5

Sortowanie tworzy następujące drzewo:

```
SELECT Imie, Nazwisko
FROM UP_Uczen
WHERE NazwiskoRodowe = Nazwisko
ORDER BY Id DESC
```



Przykład 5: projekcja i selekcja i sortowanie

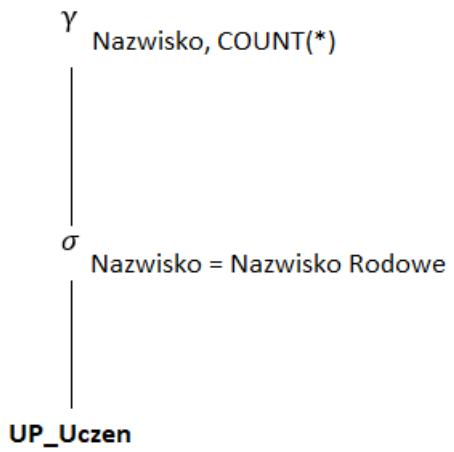
Ogólna zasada: najpierw selekcja. Co ciekawe to drzewo dałoby ten sam wynik, ale w skrajnie nieefektywny sposób:

Selekcja bez indeksów ma złożoność O(N) – trzeba wykonać skan całej tabeli. Sortowanie ma złożoność O(N logN). **Więc zawsze lepiej jest „odsiać" niepotrzebne rekordy przed operacją sortowania.** Na razie dopuszczam oba drzewa.

PRZYKŁAD 6

Grupowanie tworzy takie drzewo:

```
SELECT Nazwisko, COUNT(*)
FROM UP_Uczen
WHERE NazwiskoRodowe = Nazwisko
GROUP BY Nazwisko
```



Przykład 6: projekcja i selekcja i grupowanie

Grupowanie jest od razu selekcją i zawiera w definicji funkcje agregujące. Selekcja będąca efektem klauzuli WHERE **musi być przed (poniżej)** operatorem GRUPOWANIA.

PRZYKŁAD 7

HAVING tworzy takie drzewo:

```
SELECT Nazwisko, COUNT(*)
FROM UP_Uczen
WHERE NazwiskoRodowe = Nazwisko
GROUP BY Nazwisko
HAVING COUNT(*) > 10
```

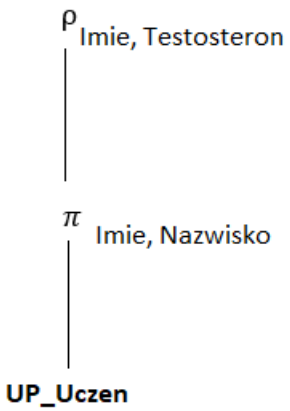
Przykład 7: projekcja i selekcja i grupowanie i having

HAVING jest operatorem drugorzędnego filtrowania i może być tylko wykorzystane gdy jest operator grupowania (klauzula GROUP BY). Na drzewie operatorów jest to **selekcja po (powyżej)** operatora grupowania.

PRZYKŁAD 8

Przemianowanie objawia się w prostym przypadku tak:

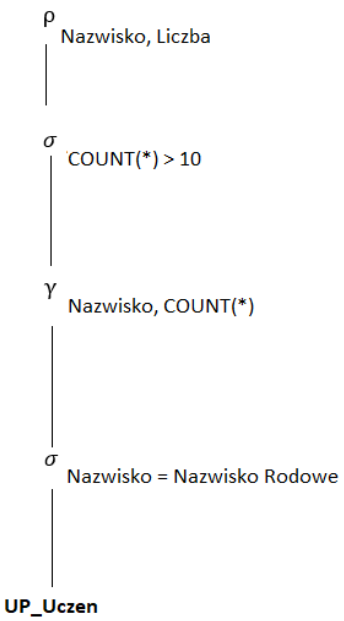
```
SELECT
       Imie,
       Nazwisko AS 'Testosteron'
FROM UP_Uczen
```



Przykład 8A: przemianowanie

W bardziej skomplikowanym przypadku:

```
SELECT Nazwisko, COUNT(*) AS Liczba
FROM UP_Uczen
WHERE NazwiskoRodowe = Nazwisko
GROUP BY Nazwisko
HAVING COUNT(*) > 10
```



Przykład 8B: przemianowanie

Przykład 8B pokazuje dlaczego w niektórych bazach danych nie można wykonać:

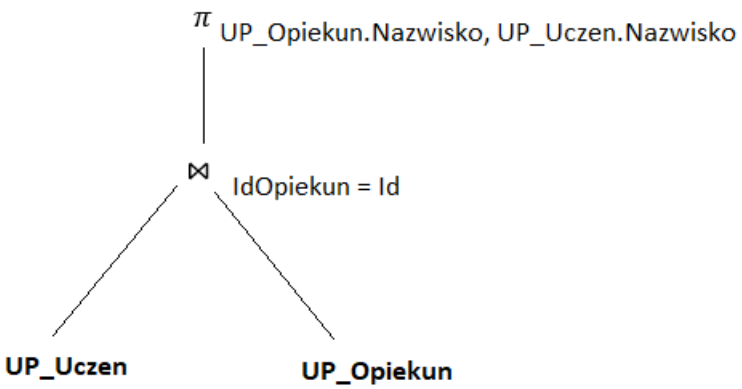
```
SELECT Nazwisko, COUNT(*) AS Liczba
FROM UP_Uczen
WHERE NazwiskoRodowe = Nazwisko
GROUP BY Nazwisko
HAVING Liczba > 10
```

Ponieważ w niektórych BD przemianowanie jest wykonywane po HAVING – więc w HAVING musimy odwoływać się po COUNT(*), a nie przyjaznym `Liczba`.

PRZYKŁAD 9

Złączenia są podchwytliwe przy tworzeniu drzewek. Przede wszystkim, trzeba pamiętać o kolejności złączeń. Nie ma to znaczenia dla zapytań z dwoma tabelami na przykład:

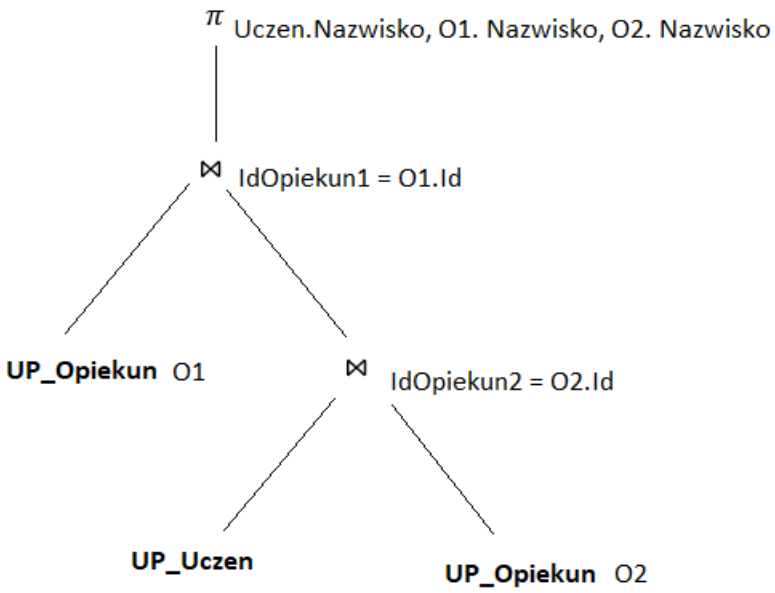
```
SELECT
       O.Nazwisko,
       U.Nazwisko
FROM
       UP_Uczen U
       INNER JOIN
UP_Opiekun O ON
U.IdOpiekun1 = O.Id
```



Przykład 9A: złączenie

Dla trzech tabel sprawa się komplikuje, poprawne drzewo może być takie:

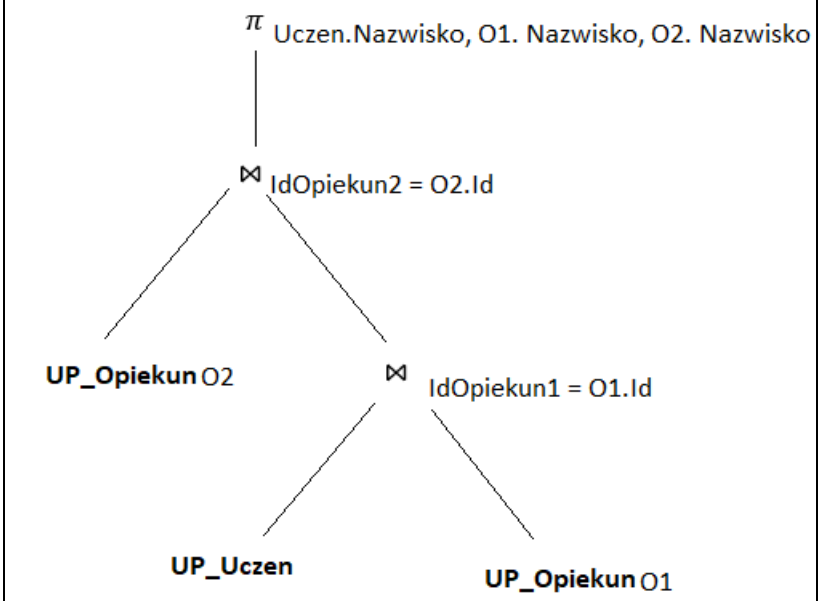
```
SELECT
       O1.Nazwisko,
       O2.Nazwisko,
       U.Nazwisko
FROM
       UP_Uczen U
       INNER JOIN
       UP_Opiekun O1 ON
       U.IdOpiekun1 =
       O1.Id
       INNER JOIN
       UP_Opiekun O2 ON
       U.IdOpiekun2 =
       O2.Id
```



Przykład 9B: złączenie z trzema tabelami

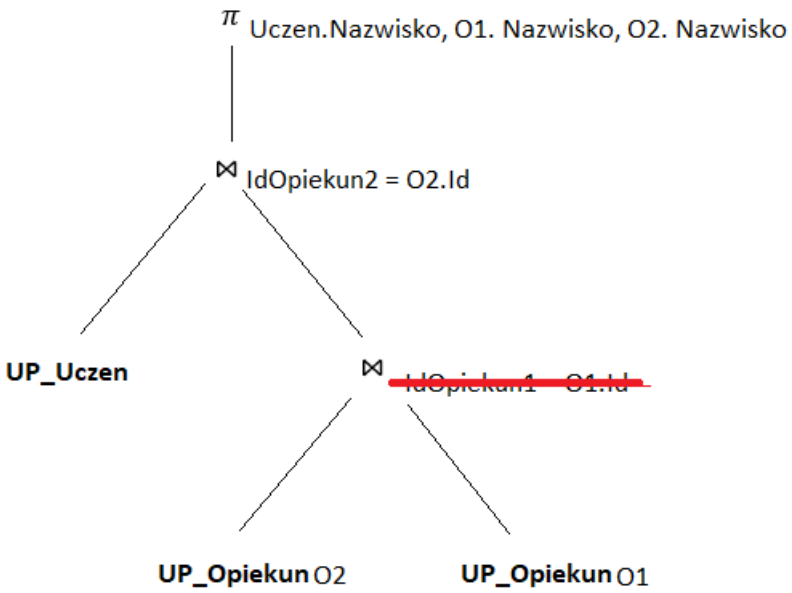
Albo takie:

```
SELECT
        O1.Nazwisko,
        O2.Nazwisko,
        U.Nazwisko
FROM
        UP_Uczen U
        INNER JOIN
        UP_Opiekun O1 ON
        U.IdOpiekun1 =
        O1.Id
        INNER JOIN
        UP_Opiekun O2 ON
        U.IdOpiekun2 =
        O2.Id
```



Przykład 9C: złączenie z trzema tabelami

Ale już nie takie:



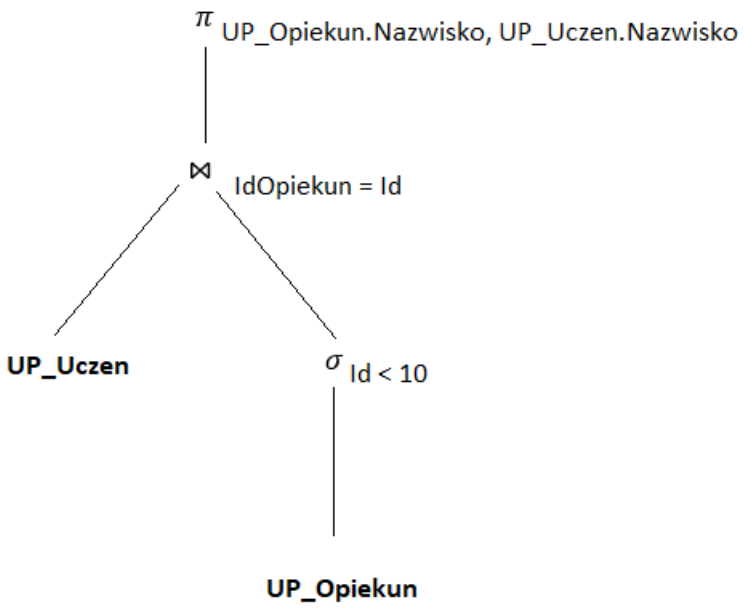
Złączenie Opiekunów O1 i O2 nie da się przeprowadzić bez tabeli Ucznia (wg. podanego warunku).

JESZCZE MAŁA NOTKA ODNOŚNIE ZŁĄCZEŃ

Nie będę za to (jeszcze) bił po łapach, ale wykonywanie złączeń oznacza zazwyczaj sortowanie, czyli wprowadza złożoność O(N logN). Co oznacza, że wszystkie „tanie” operacje optymalizatory będą starać się zaaplikować **przed** wykonaniem złączenia. Co oznacza, że takie zapytanie:

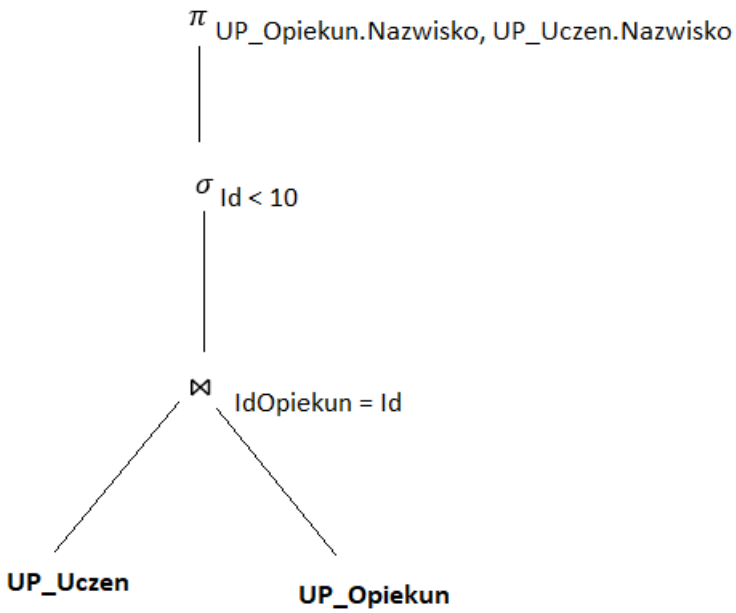
```
SELECT
	O.Nazwisko,
	U.Nazwisko
FROM
	UP_Uczen U
	INNER JOIN UP_Opiekun O ON U.IdOpiekun1 = O.Id
WHERE
	O.Id < 10;
```

Raczej (tj. w 99,99% przypadków) skutkować takim logicznym planem:



A nie takim:

#NIEWYDAJNEASF*CK



…. ale jak optymalizator nie zdąży – co może się zdarzyć przy skomplikowanych zapytaniach, to można „defensywnie” napisać SQL w taki sposób:

```
SELECT
	O.Nazwisko,
	U.Nazwisko
FROM
	UP_Uczen U
	INNER JOIN (SELECT Id, Nazwisko FROM UP_Opiekun WHERE Id < 10) O ON U.IdOpiekun1
= O.Id;
```

Osiągając prawie identyczny plan zapytania, a wymuszając selekcje przed złączeniem.

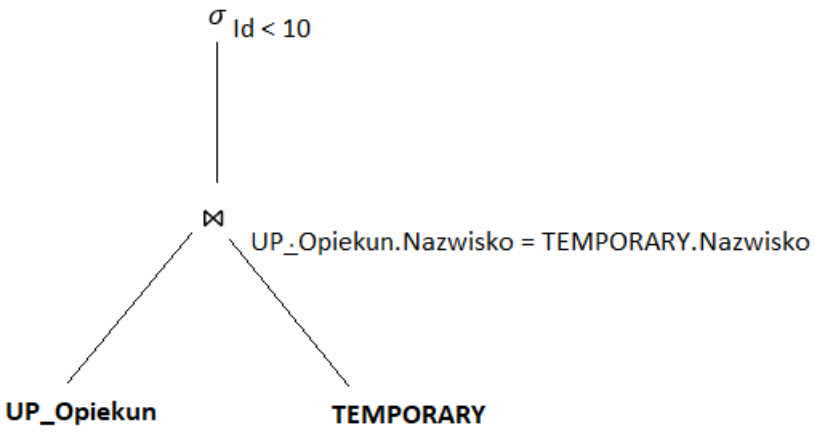
CO PROWADZI NAS DO WITH (I WIDOKÓW) … (I PODZAPYTAŃ)

Jakby się ktoś zastanawiał jak tworzyć drzewa kiedy korzystacie z CTE (klauzuli WITH), to schemat jest taki jak zawsze: **WITH/widoki i podzapytania traktujemy jako czarną skrzynkę.**

Załóżmy nadmiernie skomplikowane zapytanie, które nie mam pojęcia co robi (ale ma WITH):

```
WITH TEMPORARY AS(
	SELECT Nazwisko, COUNT(*) AS Liczba
	FROM UP_Uczen
	WHERE NazwiskoRodowe = Nazwisko
	GROUP BY Nazwisko
	HAVING COUNT(*) > 10
)
SELECT
	*
FROM
	UP_Opiekun O
	INNER JOIN TEMPORARY T ON T.Nazwisko = O.Nazwisko
WHERE
	O.Id < 10;
```

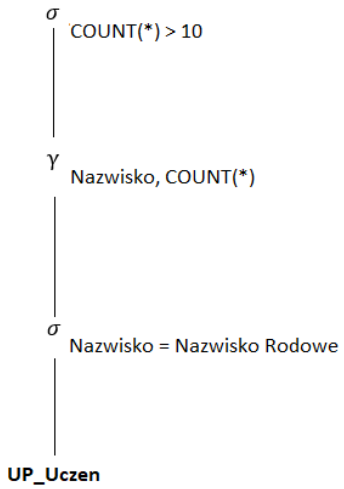
To co istotne jest za WITH, czyli plan wykonania wygląda tak:



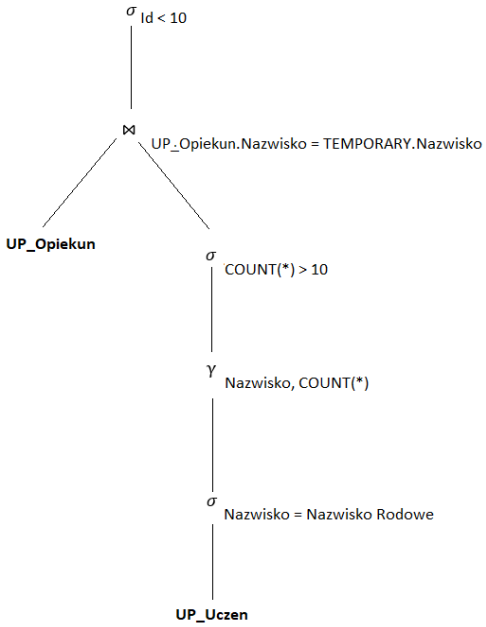
No dobra mister-magister… a jak wygląda plan dla TEMPORARY? No tak jak plan dla takiego zapytania:

```
SELECT Nazwisko, COUNT(*) AS Liczba
FROM UP_Uczen
WHERE NazwiskoRodowe = Nazwisko
GROUP BY Nazwisko
HAVING COUNT(*) > 10
```

Czyli tak:



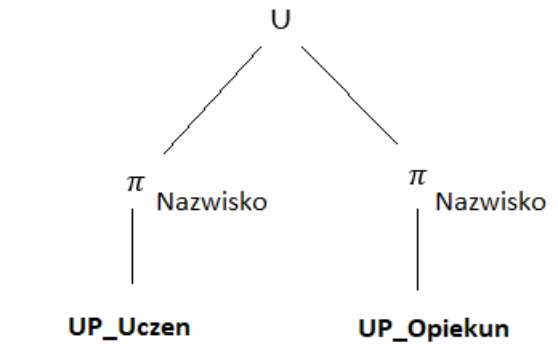
I teraz metodą copy-paste-a wklejamy ten plan w miejsce TEMPORARY:



PRZYKŁAD 9

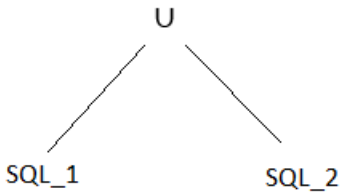
Analogicznie mając zapytania korzystające ze zbiorów plan wygląda następująco:

```
SELECT Nazwisko FROM UP_Uczen
UNION ALL
SELECT Nazwisko FROM UP_Opiekun
```



Przykład 10: Unia

Sprawne oko zauważa, że to nic innego jak podmiana takiego drzewa:



Gdzie pod SQL_1 wsadzono plan: `SELECT Nazwisko FROM UP_Uczen`

A pod SQL_2 wsadzono plan: `SELECT Nazwisko FROM UP_Opiekun`

ŚCIĄGA SQL

Ściąga DQL w SQL – w miarę uniwersalna. Wytłuszczoną czcionką zaznaczono słowa kluczowe.

Przykład	Co oznacza
SELECT * **FROM** MY_TABLE	Pobiera wszystko z tabeli MY_TABLE
SELECT * **FROM** MY_TABLE **ORDER BY** ATT	Pobiera wszystko z tabeli MY_TABLE, sortuje po atrybucie ATT rosnąco
ORDER BY ATT **ASC**, ATT2 **DESC**	Sortowanie po kilku atrybutach. Specyfikacja sortowania rosnąco ASC, malejąco DESC.
SELECT TOP 10 * **FROM** MY_TABLE	Wybranie pierwszych 10 rekordów. Wynik niedeterministyczny. To chyba że użyte z ORDER BY.

SELECT
  *
FROM
  MY_TABLE
LIMIT 10
	Wybranie pierwszych 10 rekordów. Wynik niedeterministyczny. To chyba że użyte z ORDER BY.

SELECT DISTINCT
  *
FROM
  MY_TABLE
	Pobiera wszystkie unikatowe rekordy z tabeli MY_TABLE

SELECT
  MY_ATTRIBUTE AS A,
  MY_ATTRIBUTE2
FROM
  MY_TABLE
	Pobiera atrybuty MY_ATTRIBUTE, który zostaje przemianowany na A, oraz atrybut MY_ATTRIBUTE2 z tabeli MY_TABLE

SELECT
  *
FROM
  MY_TABLE AS TTT
  INNER JOIN YOUR_TABLE AS KKK **ON** TTT.ATT = KKK.ATT
	Pobiera wszystko ze złączenia pomiędzy tabelą MY_TABLE oraz YOUR_TABLE. Obu tabelom nadano aliasy (odpowiednio TTT/KKK). Złączenie jest po warunku równościowym na atrybucie ATT

INNER JOIN **LEFT OUTER JOIN** **RIGHT OUTER JOIN** **FULL OUTER JOIN** **CROSS JOIN**	Rodzaje złączeń w SQL

SELECT
  *
FROM
  MY_TABLE,
  MY_TABLE2,
  MY_TABLE3
	Iloczyn kartezjański (CROSS JOIN) table MY_TABLE, MY_TABLE2, MY_TABLE3

SELECT
  *
FROM
  ([SQL]) ALIAS
	Opakowanie zapytania. W klauzuli FROM można użyć zapytania.

SELECT
  *
FROM
  MY_TABLE
WHERE
  A > 0
	Pobiera wszystkie atrybuty z odfiltrowanej tabeli MY_TABLE. Filtrowanie zachodzi na warunku A > 0.

WHERE
  [warunek]
  AND [warunek]
	Łączenie warunków w klauzuli where – logiczne AND

WHERE
  [warunek]
  OR [warunek]
	Łączenie warunków w klauzuli where – logiczne OR

NOT [warunek]	Negacja warunku
WHERE **ATT IN (1,2,3,10)**	Sprawdzenie czy atrybut ATT posiada wartość ze zbioru {1,2,3,10}
WHERE **ATT IN ([SQL])**	Sprawdzenie czy atrybut ATT posiada wartość ze zbioru – dynamicznie wyliczony zbiór

WHERE **EXISTS ([SQL])**	Sprawdzenie niepustości dynamicznie wyliczonego zbioru
WHERE MY_TEXT_ATTRIBUTE **LIKE [wzorzec]**	Sprawdzenie czy wartość atrybutu MY_TEXT_ATTRIBUTE pasuje do wzorca
? (czasem _) – dowolny znak (regexp: '.') % - dowolny ciąg znaków (regexp: '.'*)	Specjalny znaki we wzorcach
SELECT ATT**,** **COUNT(*)** **FROM** MY_TABLE **GROUP BY** ATT	Utworzenie grup po wartościach atrybutu ATT oraz wyliczenie agregacji typu COUNT.
SELECT ATT**,** **COUNT(*)** **FROM** MY_TABLE **WHERE** A > 0 **GROUP BY** ATT	Utworzenie grup po wartościach atrybutu ATT oraz wyliczenie agregacji typu COUNT. Do agregacji wliczane są **tylko** rekordy spełniające warunek A>0
COUNT **SUM** **MIN** **MAX** **AVG**	Rodzaje funkcji agregujących w SQL – podstawowe
COUNT(*)	Wyjątkowa agregacja – ile jest wartości
AVG(WIEK)	Średnia wartość atrybutu WIEK
COUNT(DISTINCT WIEK)	Wyjątkowa agregacja – ile różnych wartości znajduje się w grupie
SELECT ATT**,** **COUNT(*)** **FROM** MY_TABLE **GROUP BY** ATT **HAVING** **AVG(**TTT**)** > 10	Utworzenie grup po wartościach atrybutu ATT oraz wyliczenie agregacji typu COUNT. Odfiltrowanie tych **grup** dla których agregacja AVG(TTT) osiąga wartość większą niż 10.
[SQL] **UNION** **[SQL]**	Suma wyników dwóch zapytań SQL. Jako zbiór.
[SQL] **UNION ALL** **[SQL]**	Suma wyników dwóch zapytań SQL. Jako multizbiór.
UNION **UNION ALL** **MINUS (EXCEPT)** **MINUS (EXCEPT) ALL** **INTERSECT**	Możliwe operacje na zbiorach w SQL.
WITH [nazwa X] AS (**[dowolny SQL]** **)** **[dowolny SQL, w tym odwołujący się do `nazwa X`]**	Common Table Expression (CTE)

3. SQL

JAK PISAĆ SQL-E?

Klucz do sukcesu w pisaniu SQL-a (i jego ocenianiu) to piękne FORMATOWANIE ZAPYTAŃ.

Ogólnie przyjęty przeze mnie sposób formatowania jest następujący: wyobrażamy sobie kilka kolumn do których stosujemy kilka reguł:

- W pierwszej kolumnie umieszczamy **tylko** główne słowa kluczowe / grupy: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
- W każdej kolejnej nieparzystej kolumnie umieszczamy zazwyczaj główne słowa kluczowe / grupy: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY – ale wyjątkiem jest łamanie warunków w JOIN-ach (przykład 2)
- W parzystach zamieszczamy wszystko inne łamiąc wiersze wyrażeniami: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN, AND, OR (Przykład 2) – choć nie musimy łamać wierszy łącząc warunki w JOINACH (Przykład 4)
- Wyjątkiem gdy zapytanie możemy wpisać całości in-line jest przypadek gdy jest podzapytaniem z 1 kolumną w SELECT i 1 tabelą we FROM
- Wyjątkiem gdy słowo kluczowe i jego argumenty możemy wpisać w jednej linijce jest przypadek gdy słowo kluczowe i argument stanowią łącznie 2 wyrazy (z pominięciem przemianowania) (Przykład 3)

Przykładowe zapytania sformatowane w ten sposób:

Przykład 1:

```
SELECT
	p.Id AS [Id],
	p.IdJednostkaSprawozdawcza AS [UnitId],
	p.Imie AS [Name],
	p.Nazwisko AS [Surname],
	RTRIM(LTRIM(p.Imie + ' ' + p.Nazwisko)) AS [DisplayName],
	p.Aktywny AS [Active],
	p.DataModyfikacji AS [SyncDate]
FROM
	dbo.Pracownik p
```

Przykład 2:

```
SELECT
        f.[Id] AS [Id]
        ,t.[Id] AS [LessonTimeId]
        ,te.[Id] AS [TeacherId]
        ,cat.[Id] AS [CategoryId]
        ,cat.[Active] AS [CategoryActive]
FROM
        [dbo].[UP_UczenFrekwencja] f
        INNER JOIN [dbo].[UP_V_Mobile_LessonTime] t ON t.UnitId = f.IdJednostkaSprawozdawcza
                AND t.Id = f.IdPoraLekcji
        INNER JOIN [dbo].[UP_V_Mobile_Employee] te ON te.UnitId = f.IdJednostkaSprawozdawcza
                AND te.Id = f.IdPracownikModyfikujacy
                AND f.IdTypWpisuFrekwencji = cat.Id
```

Przykład 3:

```
SELECT f.[Id] AS [Id]
FROM [dbo].[UP_UczenFrekwencja] f
```

Przykład 4:

```
SELECT
        Id,
        IdLogin
FROM
        [dbo].[Uczen]
WHERE
        IdLogin IS NOT NULL
UNION ALL
SELECT
        U.Id,
        O.IdLogin
FROM
        [dbo].[Uczen] U
        INNER JOIN [dbo].[Opiekun]  O ON U.IdOpiekun1 = O.Id OR U.IdOpiekun2 = O.Id
WHERE
        O.IdLogin IS NOT NULL
        AND O.Id > 0
```

NO DOBRZE PANIE MAGISTRZE, W CZYM MA MI TO POMÓC?

Może nie jest to ewidentne na początku – ale SQL ma dużo śmieci. Najczęściej błędne działanie SQL-a wynika z klauzuli **WHERE.** Na ogół fragment FROM nie zawiera błędów – jego postać wynika z kluczy obcych w BD. Dostając od kogoś zapytanie takie jak z Przykładu 4 moje (i liczę na to, że w przyszłości Wasze) oczy widzą coś w tym stylu:

Przykład 4:

```
SELECT
        U.Id,
        O.IdLogin
FROM
        [dbo].[Uczen] U
        INNER JOIN [dbo].[Opiekun]  O ON U.IdOpiekun1 = O.Id OR U.IdOpiekun2 = O.Id
WHERE
        O.IdLogin IS NOT NULL
        AND O.Id > 0
```

BAZA DANYCH

SQL-a dobrze ćwiczy się na najmniejszym silniku BD: Sqlite. Binarkę SQLite-a wrzuciłem na repo tutaj. Plik bazy danych chinook.db tutaj. Plik ze schematem ERD tutaj.

SQlite posiada graficzny interfejs użytkownika – do pobrania tutaj.

ROZGRZEWKA

W tej sekcji zamieszczam zapytania na rozgrzewkę – te z chęcią skonsultuję:

1. Wykonać dump tabeli (SELECT *): Tabeli media_types
2. Wyświetlić pierwsze alfabetycznie tytuły pierwszych 5 rekordów z tabeli albums
3. Znaleźć kompozytora utworu ('tracks') o nazwie 'No Futuro'
4. Ile jest albumów?
5. Znaleźć nazwy utworów oraz czasy trwania (w minutach) utworów które zajmują więcej niż 900000000 bajtów
6. Wyświetlić albumy artysty 'Van Halen'
7. (Wyświetlić pierwsze alfabetycznie tytuły pierwszych 5 rekordów z tabeli albums kończący się '(Remastered)'
8. Wyświetlić alfabetycznie nazwy albumów które posiadają utwory z gatunku 'Rock' oraz 'Metal
9. Ile jest utworów bez kompozytora?
10. Ile jest kompozytorów (nie artystów)?
11. Policzyć zestawienie ile utworów ma album. Na zestawieniu są wszystkie albumy?
12. Policzyć ile jest utworów których autorem jest autor albumu do którego należą te utwory
13. Wyświetlić nazwę albumu oraz tytuł najdłuższego utworu tego albumu
14. Wyświetlić 10 rekordów. Po 5 najdłuższych płyt w gatunkach Pop oraz Electronica/Dance
15. Wyświetlić wszystkie pary utworów z albumu 'Chemical Wedding' dla których pierwszy utwór z pary jest krótszy od drugiego utworu z pary

(10 PKT) ZADANIE

Z repozytorium pobrać szablon sprawozdania – uzupełnić – odesłać.

Uwagi:

- Termin do godziny 8:00 dnia 2020-04-06
- Każde zapytanie ma 3 sekcje (Semantyka, Drzewko i SQL) – jedną sekcję uzupełniłem ja, resztę musicie dopisać. W przypadku 8-mym usunąłem potrzebę rysowania Drzewka.
- W tym roku nie pobłażam plagiatom – jeśli zauważę reużyty diagram to pracę od razu oceniam w całości na 0 pkt – zadanie możecie mi utrudnić poprzez konsekwentne formatowanie zapytań
- Na ocenę nie będzie tylko wpływać logiczna poprawność zapytania, ale też sposób jego formatowania
- Proszę sprawozdania podesłać na mail: mpenar@kia.prz.edu.pl, plik **nazwać [numer_indeksu]_BD.pdf**