Bazy danych

6. SQL- złączenia

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

2020

Złączenia “teoriomnogościowe”

```
mysql> CREATE DATABASE JanMaria;
Query OK, 1 row affected (0.02 sec)

mysql> USE JanMaria;
Database changed

mysql> CREATE TABLE Jan (Data DATE, Miasto VARCHAR(12),
                         PRIMARY KEY(Data,Miasto)
                        );
Query OK, 0 rows affected (0.14 sec)

mysql> CREATE TABLE Maria LIKE Jan;
Query OK, 0 rows affected (0.07 sec)
```

Ostatni przykład pokazuje jak utworzyć tabelę o takiej samej strukturze jak inna, istniejąca już tabela.

W wykładach dotyczących SQL często przedstawiam dodatkowe możliwości składni, mimo iż są one “poboczne” do omawianego właśnie zagadnienia.

```
mysql> SELECT * FROM Jan;
+------------+---------+
| Data       | Miasto  |
+------------+---------+
| 2017-04-16 | Kalisz  |
| 2017-04-28 | Poznań  |
| 2017-05-12 | Gniezno |
+------------+---------+
3 rows in set (0.01 sec)
```

```
mysql> SELECT * FROM Maria;
+------------+-----------+
| Data       | Miasto    |
+------------+-----------+
| 2017-04-03 | Kalisz    |
| 2017-04-16 | Toruń     |
| 2017-04-28 | Poznań    |
| 2017-05-08 | Bydgoszcz |
+------------+-----------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Jan
    -> UNION
    -> SELECT * FROM Maria;
+------------+-----------+
| Data       | Miasto    |
+------------+-----------+
| 2017-04-16 | Kalisz    |
| 2017-04-28 | Poznań    |
| 2017-05-12 | Gniezno   |
| 2017-04-03 | Kalisz    |
| 2017-04-16 | Toruń     |
| 2017-05-08 | Bydgoszcz |
+------------+-----------+
6 rows in set (0.01 sec)
```

```
mysql> SELECT * FROM Jan
    -> UNION ALL
    -> SELECT * FROM Maria;
+------------+-----------+
| Data       | Miasto    |
+------------+-----------+
| 2017-04-16 | Kalisz    |
| 2017-04-28 | Poznań    |
| 2017-05-12 | Gniezno   |
| 2017-04-03 | Kalisz    |
| 2017-04-16 | Toruń     |
| 2017-04-28 | Poznań    |
| 2017-05-08 | Bydgoszcz |
+------------+-----------+
7 rows in set (0.00 sec)
```

```
mysql> SELECT Jan.Miasto AS jm, Jan.Data AS jd, Maria.Miasto AS mm, Maria.Data AS md
    -> FROM Jan CROSS JOIN Maria;
+---------+------------+-----------+------------+
| jm      | jd         | mm        | md         |
+---------+------------+-----------+------------+
| Kalisz  | 2017-04-16 | Kalisz    | 2017-04-03 |
| Poznań  | 2017-04-28 | Kalisz    | 2017-04-03 |
| Gniezno | 2017-05-12 | Kalisz    | 2017-04-03 |
| Kalisz  | 2017-04-16 | Toruń     | 2017-04-16 |
| Poznań  | 2017-04-28 | Toruń     | 2017-04-16 |
| Gniezno | 2017-05-12 | Toruń     | 2017-04-16 |
| Kalisz  | 2017-04-16 | Poznań    | 2017-04-28 |
| Poznań  | 2017-04-28 | Poznań    | 2017-04-28 |
| Gniezno | 2017-05-12 | Poznań    | 2017-04-28 |
| Kalisz  | 2017-04-16 | Bydgoszcz | 2017-05-08 |
| Poznań  | 2017-04-28 | Bydgoszcz | 2017-05-08 |
| Gniezno | 2017-05-12 | Bydgoszcz | 2017-05-08 |
+---------+------------+-----------+------------+
12 rows in set (0.00 sec)

mysql> SELECT * FROM Jan
    -> INTERSECT
    -> SELECT * FROM Maria;
```

Nie działa w MySQL ☹

Złączenie wewnętrzne

Definicja: Złączenie *wewnętrzne* to podzbiór iloczynu kartezjańskiego tabel, spełniający podane warunki złączenia.

`T1` $\bowtie_{\texttt{warunek}}$ `T2` — wybierz te i tylko te krotki, dla których spełniony jest `warunek`.

Przykład

```
mysql> SELECT * FROM T1;
+------+------+
| x    | y    |
+------+------+
| 1    | 1    |
| 1    | 5    |
| -2   | 8    |
| 4    | 7    |
| 9    | -15  |
| 0    | 23   |
+------+------+
6 rows in set (0.08 sec)
```

```
mysql> SELECT * FROM T2;
+------+------+
| p    | q    |
+------+------+
| 1    | 5    |
| 4    | 6    |
| 9    | -15  |
| 8    | 1    |
| -3   | 7    |
+------+------+
5 rows in set (0.00 sec)
```

```
mysql> select * FROM T1 JOIN T2 ON x=p;
+------+------+------+------+
| x    | y    | p    | q    |
+------+------+------+------+
| 1    | 1    | 1    | 5    |
| 1    | 5    | 1    | 5    |
| 4    | 7    | 4    | 6    |
| 9    | -15  | 9    | -15  |
+------+------+------+------+
4 rows in set (0.06 sec)

mysql> select * FROM T1 JOIN T2 ON y=q;
+------+------+------+------+
| x    | y    | p    | q    |
+------+------+------+------+
| 1    | 1    | 8    | 1    |
| 1    | 5    | 1    | 5    |
| 4    | 7    | -3   | 7    |
| 9    | -15  | 9    | -15  |
+------+------+------+------+
4 rows in set (0.00 sec)
```

```
mysql> select * FROM T1 JOIN T2 ON x=p AND y=q;
+------+------+------+------+
| x    | y    | p    | q    |
+------+------+------+------+
| 1    | 5    | 1    | 5    |
| 9    | -15  | 9    | -15  |
+------+------+------+------+
2 rows in set (0.02 sec)

mysql> select * FROM T1 JOIN T2 ON x > p AND y < q;
+------+------+------+------+
| x    | y    | p    | q    |
+------+------+------+------+
| 1    | 1    | -3   | 7    |
| 1    | 5    | -3   | 7    |
| 9    | -15  | 1    | 5    |
| 9    | -15  | 4    | 6    |
| 9    | -15  | 8    | 1    |
| 9    | -15  | -3   | 7    |
+------+------+------+------+
6 rows in set (0.03 sec)
```

```
mysql> CREATE TABLE T3 (x TINYINT, y TINYINT)
    -> SELECT p AS x, q AS y FROM T2;
Query OK, 5 rows affected (0.12 sec)
Records:  5 Duplicates:  0 Warnings:  0
```

Utworzenie tabeli z jednoczesnym wypełnieniem jej danymi pobranymi z innej tabeli. W podanym przykładzie konieczne jest użycie aliasów.

```
mysql> SELECT * FROM T1;
+------+------+
| x    | y    |
+------+------+
| 1    | 1    |
| 1    | 5    |
| -2   | 8    |
| 4    | 7    |
| 9    | -15  |
| 0    | 23   |
+------+------+
6 rows in set (0.08 sec)
```

```
mysql> SELECT * FROM T3;
+------+------+
| x    | y    |
+------+------+
| 1    | 5    |
| 4    | 6    |
| 9    | -15  |
| 8    | 1    |
| -3   | 7    |
+------+------+
5 rows in set (0.00 sec)
```

(Niektóre) nazwy kolumn w tabelach `T1`, `T3` powtarzają się. Wówczas można użyć szczególnej postaci złączenia po powtarzającej się kolumnie.

```
mysql> SELECT * FROM T1 JOIN T3
    -> ON T1.x=T3.x;
+------+------+------+------+
| x    | y    | x    | y    |
+------+------+------+------+
| 1    | 1    | 1    | 5    |
| 1    | 5    | 1    | 5    |
| 4    | 7    | 4    | 6    |
| 9    | -15  | 9    | -15  |
+------+------+------+------+
4 rows in set (0.07 sec)
```

```
mysql> SELECT * FROM T1 JOIN T3
    -> USING (x);
+------+------+------+
| x    | y    | y    |
+------+------+------+
| 1    | 1    | 5    |
| 1    | 5    | 5    |
| 4    | 7    | 6    |
| 9    | -15  | -15  |
+------+------+------+
4 rows in set (0.06 sec)
```

```
mysql> SELECT *
    -> FROM T1 NATURAL JOIN T3;
+------+------+
| x    | y    |
+------+------+
| 1    | 5    |
| 9    | -15  |
+------+------+
2 rows in set (0.02 sec)
```

Złączenie naturalne — równość *wszystkich* powtarzających się kolumn.

Złączenie naturalne

Jan $\bowtie$ Maria

```
mysql> SELECT * FROM Jan NATURAL JOIN Maria;
+------------+--------+
| Data       | Miasto |
+------------+--------+
| 2017-04-28 | Poznań |
+------------+--------+
1 row in set (0.00 sec)
```

Jan $\bowtie_{\text{Miasto}}$ Maria

```
mysql> SELECT Jan.Miasto, Jan.Data AS jd, Maria.Data AS md
    -> FROM Jan JOIN Maria ON Jan.Miasto = Maria.Miasto;
+--------+------------+------------+
| Miasto | jd         | md         |
+--------+------------+------------+
| Kalisz | 2017-04-16 | 2017-04-03 |
| Poznań | 2017-04-28 | 2017-04-28 |
+--------+------------+------------+
2 rows in set (0.00 sec)
```

Złączenie ⊖ (theta)

```
mysql> SELECT Jan.Miasto AS JM, Jan.Data AS JD, Maria.Miasto AS MM, Maria.Data AS MD
    -> FROM Jan, Maria
    -> WHERE Jan.Miasto=Maria.Miasto;
+--------+------------+--------+------------+
| JM     | JD         | MM     | MD         |
+--------+------------+--------+------------+
| Kalisz | 2017-04-16 | Kalisz | 2017-04-03 |
| Poznań | 2017-04-28 | Poznań | 2017-04-28 |
+--------+------------+--------+------------+
2 rows in set (0.05 sec)
```

Warunek złączenia zapisany w klauzuli `WHERE`.

Składnia typowa dla Oracle.

```
mysql> SELECT Jan.Miasto AS JM, Jan.Data AS JD, Maria.Miasto AS MM, Maria.Data AS MD
    -> FROM Jan, Maria
    -> WHERE Jan.Miasto=Maria.Miasto AND (DATEDIFF(Jan.Data,Maria.Data) > 10);
+--------+------------+--------+------------+
| JM     | JD         | MM     | MD         |
+--------+------------+--------+------------+
| Kalisz | 2017-04-16 | Kalisz | 2017-04-03 |
+--------+------------+--------+------------+
1 row in set (0.00 sec)
```

Składnia “konwencjonalna”:

```
mysql> SELECT Miasto, Jan.Data AS JD, Maria.Data AS MD
    -> FROM Jan JOIN Maria USING (Miasto)
    -> WHERE DATEDIFF(Jan.Data, Maria.Data) > 10;
+--------+------------+------------+
| Miasto | JD         | MD         |
+--------+------------+------------+
| Kalisz | 2017-04-16 | 2017-04-03 |
+--------+------------+------------+
1 row in set (0.00 sec)
```

Przykład klasyczny — złączenie tabel rozbitych na skutek normalizacji

```
mysql> CREATE TABLE Klienci
    -> (NrKlienta TINYINT UNSIGNED NOT NULL PRIMARY KEY,
    -> Nazwa VARCHAR(32) NOT NULL,
    -> Miasto VARCHAR(32) NOT NULL)
    -> CHARACTER SET cp852;
Query OK, 0 rows affected (0.19 sec)

mysql> LOAD DATA LOCAL INFILE 'c://wyklady//bazy11//klienci.txt'
    -> INTO TABLE Klienci
    -> LINES TERMINATED BY '\r\n';
Query OK, 12 rows affected (0.07 sec)
Records: 12  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SET CHARACTER SET cp852;
Query OK, 0 rows affected (0.00 sec)
```

`LOCAL` oznacza, że podajemy ścieżkę do pliku zlokalizowanego na kliencie, nie na serwerze. Pola domyślnie odzielane są znakami tabulacji.
`LINES TERMINATED BY '\r\n'` i podany charset to idiosynkrazje DOS/Windows.

```
mysql> SELECT * FROM Klienci;
+-----------+-----------------------------+-------------+
| NrKlienta | Nazwa                       | Miasto      |
+-----------+-----------------------------+-------------+
|         1 | Benedetto                   | Kraków      |
|         2 | Paolo                       | Bolesławiec |
|         3 | Cuda Niewidy                | Kraków      |
|         4 | Najsłynniejsza Firma        | Leszno      |
|         5 | Nowoczesny Sklep            | Lublin      |
|         6 | Niesamowita Sprawa          | Kraków      |
|         7 | Nowe Życie                  | Bolesławiec |
|         8 | Nibynóżka                   | Puck        |
|         9 | Ciao-Ciao-Ciacho            | Kraków      |
|        10 | Wymyślanie Nazw Jest Trudne | Lublin      |
|        11 | Bee Mee Kukuryku            | Kraków      |
|        12 | This Is The End             | Bolesławiec |
+-----------+-----------------------------+-------------+
12 rows in set (0.00 sec)
```

```
mysql> DESCRIBE Zamowienia;
+--------------+----------------------+------+-----+---------+----------------+
| Field        | Type                 | Null | Key | Default | Extra          |
+--------------+----------------------+------+-----+---------+----------------+
| NrZam        | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| NrKlienta    | smallint(5) unsigned | NO   |     |         |                |
| Kwota        | float unsigned       | NO   |     |         |                |
| DataZlozenia | date                 | NO   |     |         |                |
| DataZaplaty  | date                 | YES  |     | NULL    |                |
+--------------+----------------------+------+-----+---------+----------------+
5 rows in set (0.27 sec)

mysql> DESCRIBE Klienci;
+-----------+--------------------+------+-----+---------+-------+
| Field     | Type               | Null | Key | Default | Extra |
+-----------+--------------------+------+-----+---------+-------+
| NrKlienta | tinyint(3) unsigned | NO   | PRI |         |       |
| Nazwa     | varchar(32)        | NO   |     |         |       |
| Miasto    | varchar(32)        | NO   |     |         |       |
+-----------+--------------------+------+-----+---------+-------+
3 rows in set (0.08 sec)
```

Jest wspólna kolumna — `NrKlienta`.

```
mysql> SELECT * FROM Zamowienia NATURAL JOIN Klienci
    -> WHERE NrKlienta > 10
    -> LIMIT 12;
+-----------+-------+---------+--------------+-------------+------------------+-------------+
| NrKlienta | NrZam | Kwota   | DataZlozenia | DataZaplaty | Nazwa            |Miasto       |
+-----------+-------+---------+--------------+-------------+------------------+-------------+
|        11 |     2 | 4702.25 | 2017-01-11   | 2017-01-13  | Bee Mee Kukuryku |Kraków       |
|        12 |     4 | 7298.23 | 2017-02-13   | NULL        | This Is The End  |Bolesławiec  |
|        11 |     6 | 1122.14 | 2017-02-26   | 2017-02-28  | Bee Mee Kukuryku |Kraków       |
|        12 |    14 | 2196.22 | 2017-02-17   | NULL        | This Is The End  |Bolesławiec  |
|        12 |    21 | 2239.01 | 2017-02-23   | 2017-03-23  | This Is The End  |Bolesławiec  |
|        12 |    24 |  1779.4 | 2017-04-19   | NULL        | This Is The End  |Bolesławiec  |
|        12 |    35 | 7552.24 | 2017-03-12   | NULL        | This Is The End  |Bolesławiec  |
|        12 |    36 | 8296.06 | 2017-03-04   | NULL        | This Is The End  |Bolesławiec  |
|        12 |    45 | 8840.35 | 2017-03-05   | 2017-03-07  | This Is The End  |Bolesławiec  |
|        12 |    46 |  891.25 | 2017-02-10   | 2017-03-22  | This Is The End  |Bolesławiec  |
|        11 |    55 | 9963.39 | 2017-02-13   | 2017-04-25  | Bee Mee Kukuryku |Kraków       |
|        11 |    56 | 6663.54 | 2017-02-19   | NULL        | Bee Mee Kukuryku |Kraków       |
+-----------+-------+---------+--------------+-------------+------------------+-------------+
12 rows in set (0.03 sec)
```

```
mysql> SELECT * FROM Zamowienia NATURAL JOIN Klienci
    -> WHERE MONTH(DataZaplaty) = 4 AND Kwota > 8400.0;
+-----------+-------+---------+--------------+-------------+----------------------+--------+
| NrKlienta | NrZam | Kwota   | DataZlozenia | DataZaplaty | Nazwa                | Miasto |
+-----------+-------+---------+--------------+-------------+----------------------+--------+
|         6 |    17 | 9082.96 | 2017-03-07   | 2017-04-23  | Niesamowita Sprawa   | Kraków |
|        11 |    55 | 9963.39 | 2017-02-13   | 2017-04-25  | Bee Mee Kukuryku     | Kraków |
|         1 |    77 | 8853.25 | 2017-04-16   | 2017-04-29  | Benedetto            | Kraków |
|         4 |    84 | 8448.24 | 2017-04-25   | 2017-04-29  | Najsłynniejsza Firma | Leszno |
|        11 |    86 | 8641.58 | 2017-03-06   | 2017-04-30  | Bee Mee Kukuryku     | Kraków |
|         9 |   127 |  9015.9 | 2017-04-02   | 2017-04-13  | Ciao-Ciao-Ciacho     | Kraków |
|        11 |   138 | 9340.57 | 2017-03-14   | 2017-04-30  | Bee Mee Kukuryku     | Kraków |
+-----------+-------+---------+--------------+-------------+----------------------+--------+
7 rows in set (0.00 sec)
```

Jak baza danych realizuje złączenia (wewnętrzne)

Złączenia zdefiniowane są jako podzbiory iloczynu kartezjańskiego odpowiednich tabel, jednak *na ogół* nie są one realizowane w ten sposób, iż najpierw wykonywany jest iloczyn kartezjański, potem zaś wybierane są odpowiednie wiersze. Sposób realizacji złączenia nie może wpłynąć na ostateczny wynik zapytania, ale może wpłynąć (i wpływa!) na czas realizacji zapytania, zajętość pamięci itp. Jak zatem baza danych realizuje złączenie? Najczęściej używa się następujących trzech algorytmów:

1. Złączenie pętli zagnieżdżonych (*nested loops*)

1. Baza przegląda pierwszą tabelę wejściową. Wiersze nie spełniające filtru nałożonego tylko na tą tabelę odrzuca, wiersze spełniające filtr przekazuje dalej.
2. Do każdego wiersza z pierwszej tabeli, który pozostał po pierwszym kroku, dopasowywane są wiersze z drugiej tabeli, spełniające warunek złączenia (złączenie wewnętrzne) lub wartości `NULL`, jeśli wierszy takowych nie ma (złączenie zewnętrzne). Odrzucane są wiersze nie spełniające warunków dla dotychczas wykorzystanych tabel, czyli filtru dla drugiej tabeli i warunków obejmujących łącznie pierwszą i drugą tabelę.
3. Analogicznie postępujemy dla trzeciej i każdej następnej tabeli.

Takie złączenie ma postać zagnieżdżonych pętli — najbardziej zewnętrzna obiega pierwszą tabelę wejściową, najbardziej wewnętrzna — ostatnią. Z tego względu istotne jest, aby *pierwsza*, najbardziej zewnętrzna pętla, odrzucała możliwie dużo wierszy oraz żeby połączenia następowały po kolumnach indeksowanych, wówczas bowiem łatwo jest znaleźć wiersze pasujące do aktualnego klucza złączenia.

Na każdym etapie wymagana jest jedynie informacja o aktualnie przetwarzanej pozycji oraz zawartość konstruowanego w danej chwili wiersza wynikowego — cały proces nie wymaga dużej pamięci.

Złączenie pętli zagnieżdżonych może mieć warunki złączenia w postaci nierówności. Wiele RDBMS wyraźnie preferuje ten typ złączenia.

2. Złączenie haszujące (mieszające, *hash join*)

Stosuje się tylko do złączeń wewnętrznych, w których warunki złączenia mają postać równości. *Teoretycznie* jest to wówczas najszybszy algorytm złączenia, ale *praktycznie* tak wcale nie musi być.

Złączane tabele przetwarzane są niezależnie. Cały algorytm przebiega w dwu fazach:

- W *fazie budowania* dla mniejszej (po zastosowaniu filtru) tabeli tworzona jest *tablica haszująca* (tablica mieszająca, *hash table*), powstała przez zastosowanie *funkcji haszującej* do kluczy złączenia. Teoretycznie rozmieszcza on "hasze" przyporządkowane różnym kluczom równomiernie w pamięci. Algorytm działa szczególnie szybko, jeśli cała tablica haszująca mieści się w pamięci.

- W *fazie wyszukiwania* sekwencyjnie przeglądana jest większa tabela. Na kluczu złączenia każdego wiersza wykonywana jest ta sama funkcja haszująca; jeżeli odpowiedni element znajduje się w tablicy haszującej dla *pierwszej* tabeli, wiersze są łączone. Jeżeli nie, wiersz drugiej tabeli jest odrzucany. Jeżeli tablica haszująca znajduje się w całości w pamięci, średni czas wyszukiwania elementów jest stały i niezależny od rozmiarów tablicy — to właśnie stanowi o efektywności tego algorytmu.

Problemy ze złączeniem haszującym

Efektywność złączenia haszującego silnie zależy od doboru funkcji haszującej. Idealna funkcja haszująca ma tę własność, że zmiana pojedynczego bitu w kluczu, zmienia połowę bitów hasza, i zmiana ta jest niezależna od zmian spowodowanych przez zmianę dowolnego innego bitu w kluczu. Idealne funkcje haszujące są trudne do zaprojektowania lub kosztowne obliczeniowo, stosuje się więc funkcje "gorsze", co jednak prowadzić może do *kolizji*: Funkcja haszująca dwóm różnym kluczom usiłuje przypisać tę samą wartość hasza. Aby tego uniknąć, stosuje się różne techniki, co jednak powiększa koszt obliczeniowy algorytmu.

Innym problemem jest *grupowanie*: Wbrew założeniom, funkcje haszujące mają tendencję do nierównomiernego rozmieszczania haszy w pamięci, co zmniejsza efektywnośc wykorzystania pamięci i powiększa czas wyszukiwania.

3. Złączenie sortująco-scalające (*sort and merge*)

Tabele odczytuje się niezależnie i stosuje do nich właściwe filtry, po czym wynikowe zbiory wierszy sortuje się względem klucza złączenia. Następnie dwie posortowane listy zostają scalone. Baza danych odczytuje na przemian wiersze z każdej listy. Jeżeli warunek złączenia ma postać równości, baza porównuje górne wiersze i odrzuca te, które znajdują się na posortowanej liście wcześniej niż górny wiersz drugiej tabeli, zwraca zaś te, które wzajemnie sobie odpowiadają. Procedura scalania jest szybka, ale procedura wstępnego sortowania jest wolna, o ile nie ma gwarancji, że *oba* zbiory wierszy mieszczą się w pamięci.

Uwaga!

Jeżeli ze względów estetycznych lub innych *wynikowy* zbiór wierszy pewnego zapytania ma być posortowany, to jeśli ten wynikowy zbiór w całości nie mieści się w pamięci, może to *znacznie* spowolnić czas wykonywania złączenia.

Złączenia zewnętrzne

Mogą obejmować krotki, które *nie* należą do iloczynu kartezjańskiego tabel wejściowych.

```
mysql> CREATE DATABASE Ludzie CHARACTER SET cp1250;
Query OK, 1 row affected (0.05 sec)

mysql> USE Ludzie;
Database changed

mysql> CREATE TABLE Kobiety
    -> (Nr SMALLINT UNSIGNED, Imie VARCHAR(16), Wiek SMALLINT UNSIGNED);
Query OK, 0 rows affected (0.73 sec)

mysql> CREATE TABLE Mezczyzni LIKE Kobiety;
Query OK, 0 rows affected (0.13 sec)
```

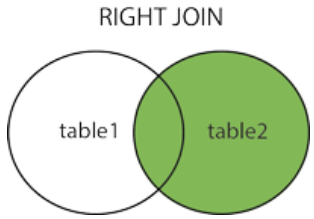
```
mysql> INSERT INTO Kobiety VALUES
    -> (1,'Karolina',21),(2,'Patrycja',24),(3,'Zuzia',23),(4,'Aśka',22),
    -> (5,'Małgorzata',22),(6,'Anna',23),(7,'Martyna',19),(8,'Sabina',21);
Query OK, 8 rows affected (0.21 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> INSERT INTO Mezczyzni VALUES
    -> (1,'Jan',24),(2,'Piotrek',28),(3,'Hubert',18),(4,'Grzegorz',22),
    -> (5,'Jan',21),(6,'Krzysztof',26),(9,'Dominik',22);
Query OK, 7 rows affected (0.12 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM Kobiety
    -> ORDER BY Imie;
+------+------------+------+
| Nr   | Imie       | Wiek |
+------+------------+------+
|    6 | Anna       |   23 |
|    4 | Aśka       |   22 |
|    1 | Karolina   |   21 |
|    5 | Małgorzata |   22 |
|    7 | Martyna    |   19 |
|    2 | Patrycja   |   24 |
|    8 | Sabina     |   21 |
|    3 | Zuzia      |   23 |
+------+------------+------+
8 rows in set (0.11 sec)
```

```
mysql> SELECT * FROM Mezczyzni
    -> ORDER BY Imie;
+------+-----------+------+
| Nr   | Imie      | Wiek |
+------+-----------+------+
|    9 | Dominik   |   22 |
|    4 | Grzegorz  |   22 |
|    3 | Hubert    |   18 |
|    1 | Jan       |   24 |
|    5 | Jan       |   21 |
|    6 | Krzysztof |   26 |
|    2 | Piotrek   |   28 |
+------+-----------+------+
7 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Mezczyzni RIGHT JOIN Kobiety
    -> ON Mezczyzni.Nr = Kobiety.Nr;
+------+-----------+------+------+------------+------+
| Nr   | Imie      | Wiek | Nr   | Imie       | Wiek |
+------+-----------+------+------+------------+------+
|    1 | Jan       |   24 |    1 | Karolina   |   21 |
|    2 | Piotrek   |   28 |    2 | Patrycja   |   24 |
|    3 | Hubert    |   18 |    3 | Zuzia      |   23 |
|    4 | Grzegorz  |   22 |    4 | Aśka       |   22 |
|    5 | Jan       |   21 |    5 | Małgorzata |   22 |
|    6 | Krzysztof |   26 |    6 | Anna       |   23 |
| NULL | NULL      | NULL |    7 | Martyna    |   19 |
| NULL | NULL      | NULL |    8 | Sabina     |   21 |
+------+-----------+------+------+------------+------+
8 rows in set (0.00 sec)
```



Prawe złączenie: prawie* jak zwykłe złączenie, z tym, że wiersze z prawej tabeli nie mające odpowiedników w lewej tabeli, są uzupełniane wartościami `NULL`.

Kolejność tabel jest istotna.

*„Prawie” robi różnicę ☺

`LEFT JOIN` działa tak samo, jak `RIGHT JOIN`, z tą różnicą, że wiersze z tabeli stojącej po *lewej* stronie operacji `JOIN`, które nie mają swoich odpowiedników w tabeli stojącej po prawej stronie, zostaną uzupełnione wartościami `NULL`.

Przykład — poszukiwanie wierszy, które nie mają odpowiedników w innej tabeli.

```
mysql> SELECT Kobiety.*
    -> FROM Kobiety JOIN Mezczyzni
    -> ON Kobiety.Nr = Mezczyzni.Nr
    -> WHERE ISNULL(Mezczyzni.Nr);
Empty set (0.06 sec)
```

```
mysql> SELECT Kobiety.*
    -> FROM Kobiety LEFT JOIN Mezczyzni
    -> ON Kobiety.Nr = Mezczyzni.Nr
    -> WHERE ISNULL(Mezczyzni.Nr);
+------+---------+------+
| Nr   | Imie    | Wiek |
+------+---------+------+
|    7 | Martyna |   19 |
|    8 | Sabina  |   21 |
+------+---------+------+
2 rows in set (0.00 sec)
```

Zauważmy, że kolumna `Nr` *nie jest* zadeklarowana jako `NOT NULL`.

Pełne złączenie zewnętrzne

Złączenia typu `LEFT JOIN` i `RIGHT JOIN` łączą te wiersze tabel, które mają swoje odpowiedniki, natomiast brakujące wiersze z jednej tabeli zostaną w wyniku wypełnione wartościami `NULL`. Przypuśćmy jednak, że dwie tabele mają pewien wspólny zakres kluczy złączenia, ale w obu są klucze niepasujące do żadnego wiersza drugiej tabeli. Chcemy zobaczyć wszystko, co pasuje i wszystko, co nie pasuje, z obu tabel. W tej sytuacji trzeba użyć pełnego złączenia zewnętrznego.

Przykład

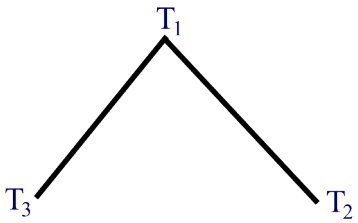
```
mysql> SELECT * FROM Mezczyzni;
+------+-----------+------+
| Nr   | Imie      | Wiek |
+------+-----------+------+
|    1 | Jan       |   24 |
|    2 | Piotrek   |   28 |
|    3 | Hubert    |   18 |
|    4 | Grzegorz  |   22 |
|    5 | Jan       |   21 |
|    6 | Krzysztof |   26 |
|    9 | Dominik   |   22 |
+------+-----------+------+
7 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Kobiety;
+------+------------+------+
| Nr   | Imie       | Wiek |
+------+------------+------+
|    1 | Karolina   |   21 |
|    2 | Patrycja   |   24 |
|    3 | Zuzia      |   23 |
|    4 | Aśka       |   22 |
|    5 | Małgorzata |   22 |
|    6 | Anna       |   23 |
|    7 | Martyna    |   19 |
|    8 | Sabina     |   21 |
+------+------------+------+
8 rows in set (0.01 sec)
```

```
mysql> SELECT Mezczyzni.Nr AS 'Numer Faceta', Mezczyzni.Imie AS Facet,
    -> Kobiety.Imie AS Baba, Kobiety.Nr AS "Numer Baby"
    -> FROM Mezczyzni RIGHT JOIN Kobiety ON Mezczyzni.Nr = Kobiety.Nr
    -> UNION
    -> SELECT Mezczyzni.Nr AS 'Numer Faceta', Mezczyzni.Imie AS Facet,
    -> Kobiety.Imie AS Baba, Kobiety.Nr AS "Numer Baby"
    -> FROM Mezczyzni LEFT JOIN Kobiety ON Mezczyzni.Nr = Kobiety.Nr;
+--------------+-----------+------------+------------+
| Numer Faceta | Facet     | Baba       | Numer Baby |
+--------------+-----------+------------+------------+
|            1 | Jan       | Karolina   |          1 |
|            2 | Piotrek   | Patrycja   |          2 |
|            3 | Hubert    | Zuzia      |          3 |
|            4 | Grzegorz  | Aśka       |          4 |
|            5 | Jan       | Małgorzata |          5 |
|            6 | Krzysztof | Anna       |          6 |
|         NULL | NULL      | Martyna    |          7 |
|         NULL | NULL      | Sabina     |          8 |
|            9 | Dominik   | NULL       |       NULL |
+--------------+-----------+------------+------------+
9 rows in set (0.08 sec)
```

Typy złączeń


```
SELECT ... FROM T1
JOIN T2 ON T1.kp=T2.kq
JOIN T3 ON T2.kr=T3.ks
WHERE ...;
```




```
SELECT ... FROM T1
JOIN T2 ON T1.kp=T2.kq
JOIN T3 ON T1.kr=T3.ks
WHERE ...;
```

Każdy wierzchołek grafu reprezentuje tabelę, każda krawędź złączenie

```
JOIN ... ON ...
```

Tego typu złączenia “drzewiaste”, mogące obejmować kilka, kilkanaście, kilkadziesiąt lub nawet więcej ☺ tabel, są charakterystyczne dla złączeń, które obejmują dane z wielu tabel, porozdzielane pomiędzy nimi głównie ze względów normalizacyjnych.

W SQL daje się jednak realizować także złączenia o innej strukturze, w szczególności złączenia, w których pojawiają się cykle.

Zupełnie inny typ złączenia

Przypuśćmy, że mamy trzy tabele z takimi oto przykładowymi danymi:

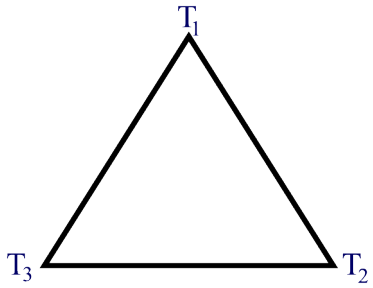
```
mysql> SELECT * FROM T1;
+------+------+
| I    | J    |
+------+------+
| 1    | 1    |
| 2    | 1    |
| 3    | 2    |
+------+------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM T2;
+------+------+
| K    | L    |
+------+------+
| 1    | A    |
| 2    | B    |
| 3    | C    |
+------+------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM T3;
+------+------+
| M    | N    |
+------+------+
| A    | 1    |
| A    | 2    |
| B    | 1    |
| C    | 2    |
| C    | 3    |
+------+------+
5 rows in set (0.00 sec)
```

Dla takich tabel można zaprojektować złączenie *“zapętlone”*:

```
mysql> SELECT Jeden.I AS I, Jeden.J AS J, K, L, M, N FROM T1 AS Jeden
    -> JOIN T2 ON Jeden.I=T2.K
    -> JOIN T3 ON T2.L=T3.M
    -> JOIN T1 As Dwa ON T3.N=Dwa.J
    -> WHERE Jeden.I=Dwa.I AND Jeden.J=Dwa.J;
+------+------+------+------+------+------+
| I    | J    | K    | L    | M    | N    |
+------+------+------+------+------+------+
| 1    | 1    | 1    | A    | A    | 1    |
| 2    | 1    | 2    | B    | B    | 1    |
| 3    | 2    | 3    | C    | C    | 2    |
+------+------+------+------+------+------+
3 rows in set (0.00 sec)
```



Ponieważ tabela `T1` występuje w tym złączeniu dwa razy, każde wystąpienie musi mieć unikalny alias. Warunek `WHERE` stanowi, że oba wystąpienia tabeli `T1` odnoszą się do tych samych wierszy.

Wymuszanie kolejności wykonywania złączeń

Przypuśćmy, że łaczymy więcej niż dwie tabele i że warunek złączenia ma postać

$$\texttt{... AND } T_1.k_2 = T_2.k_2 \texttt{ AND } T_1.k_3 = T_3.k_3 \texttt{ ...}$$

Chcemy wykonać pętle zagnieżdżone po tabelach T_1 i T_2 *przed* odwołaniem do tabeli T_3. Aby odroczyć wykonanie złączenia, *trzeba uczynić je zależnym* od danych ze złączenia, które powinno zostać wykonane wcześniej.

```
... AND T1.k2=T2.k2 AND T1.k3+0*T2.k2=T3.k3 ...
```

Druga wersja jest logicznie równoważna pierwszej, jednak baza interpretując je, po lewej stronie drugiego złączenia trafia na wyrażenie, które zależy tak od tabeli $\texttt{T}_1$, jak i $\texttt{T}_2$ (nie ma znaczenia, że wartość z tabeli $\texttt{T}_2$ nie może wpłynąć na wynik), nie wykona więc złączenia z $\texttt{T}_3$ przed złączeniem z $\texttt{T}_2$.

Uwaga: Oczywiście to samo stosuje się do złączeń sformułowanych w postaci

```
... T1 JOIN T2 ON T1.k2=T2.k2 JOIN T3 ON T1.k3=T3.k3 ...
```