Bazy danych
przed 6 wykładem

Andrzej Giniewicz

12.04.2024

W niniejszej porcji materiałów zajmiemy się korzystaniem z więcej niż jednej tabeli.

1 Iloczyn kartezjański tabel

Pomyślmy przez chwilę o iloczynie kartezjańskim, jaki znamy z matematyki. Jeśli mamy dwa zbiory, A i B, iloczyn kartezjański $A \times B$ definiujemy jako zbiór

$$\{(a, b) : a \in A \wedge b \in B\},$$

czyli zbiór wszystkich par. Jeśli zbioru A i B są z przestrzeni wielowymiarowymi odpowiednio o n i m wymiarach, jak tabele w SQL, gdzie każda kolumna to jeden wymiar krotki, to

$$A \times B = \{((a_1, \ldots, a_n), (b_1, \ldots, b_m)) : (a_1, \ldots, a_n) \in A \wedge (b_1, \ldots, b_m) \in B\}.$$

W SQL będziemy dodatkowo spłaszczać krotki, tak aby uzyskać zbiór krotek wartości, a nie zbiór par krotek wartości. Wprowadźmy operację

$$A \otimes B \coloneqq \{(a_1, \ldots, a_n, b_1, \ldots, b_m) : (a_1, \ldots, a_n) \in A \wedge (b_1, \ldots, b_m) \in B\}.$$

W efekcie, jeśli zbiór A był n wymiarowy i miał N elementów (n kolumn i N wierszy) a zbiór B był m wymiarowy i miał M elementów (m kolumn i M wierszy), to zbiór $A \otimes B$ będzie miał $n + m$ wymiarów i $N \cdot M$ elementów ($n + m$ kolumn i $N \cdot M$ wierszy).

Aby w MySQL i MariaDB zapisać tabelę, która jest iloczynem kartezjańskim dwóch tabel `tabela1` i `tabela2`, napiszemy w miejscu jej nazwy `tabela1 CROSS JOIN tabela2`, na przykład

```
SELECT * FROM tabela1 CROSS JOIN tabela2 LIMIT 10;
```

Zwróćmy uwagę, że wyrażenie `tabela1 CROSS JOIN tabela2` traktujemy tak samo, jak nazwę tabeli. Przy wykonaniu powyższej operacji, zależnie od tabel, może pojawić się błąd mówiący, że nazwy nie są unikatowe. Ponieważ każda tabela musi mieć unikatowe nazwy kolumn, jeśli na liście kolumn drugiej tabeli pojawi się taka sama jak w pierwszej tabeli, zapytanie z „giwazdką” się nie wykona. Musimy wtedy ręcznie wskazać te kolumny, które chcemy

wyświetlić, wypisując je. W przypadku duplikatów musimy podać pełną nazwę kolumny. Pełna nazwa kolumny to `nazwa_tabeli.nazwa_kolumny`, przy czym, jeśli w nazwach są spacje, używamy znaku ‘ na nazwie tabeli i kolumny osobno ‘`nazwa tabeli`‘.‘`nazwa kolumny`‘. Taka składnia jest wymagana, jeśli w dwóch tabelach istnieje kolumna o tej samej nazwie. Jeśli nie ma duplikatu nazwy, nazwę tabeli można pominąć. Dlatego wcześniej, gdy rozważaliśmy zapytania do tylko jednej tabeli, nie musieliśmy korzystać z nazw tabeli przed nazwami kolumn.

Możemy też nadać aliasy zarazem kolumnom, jak i tabelom. Na przykład

```
SELECT pierwsza.kolumna as ‘kolumna z A‘,
       druga.kolumna as ‘kolumna z B‘
FROM ‘tabela A‘ AS pierwsza CROSS JOIN ‘tabela B‘ AS druga
LIMIT 10;
```

Po wykonaniu takiego zapytania, uzyskamy tabelę mającą dwie kolumny: `kolumna z A` i `kolumna z B`, pomimo iż oryginalnie miały tę samą nazwę.

Iloczyn kartezjański ma jednak wąskie zastosowanie, ponieważ rzadko zależy nam na łączeniu wierszy „każdy z każdym”. O wiele częściej chcemy połączyć odpowiadające sobie wiersze. Na przykład, jeśli w jednej tabeli znajdują się informacje o zamówieniach i w nich są jakieś identyfikatory użytkowników, natomiast w drugiej tabeli znajdują się informacje o użytkownikach opatrzonych tymi samymi identyfikatorami, łącząc te tabele będziemy chcieli wyfiltrować tylko niektóre wiersze. Oczywiście moglibyśmy to zrobić warunkiem

```
SELECT zamówienia.‘numer zamówienia‘ as zamówienie,
       klienci.‘imię i nazwisko‘ as klient
FROM zamówienia CROSS JOIN klienci
WHERE zamówienia.‘id klienta‘ = klienci.‘id klienta‘;
```

Niestety, jest to bardzo nieefektywne, ponieważ najpierw tworzymy ogromną tabelę zawierającą wszystkie kombinacje, a potem wybieramy tylko niektóre wiersze. Załóżmy uproszczoną sytuację, w której tabela `zamówienia` zawiera wiersze

‘numer zamówienia‘	‘id klienta‘
1	3
2	4
3	2

Natomiast w tabeli klienci mamy

‘id klienta‘	‘imię i nazwisko‘
1	Jan Czarnowidz
2	Janina Nosiwoda
3	Joachim Krzywouch
4	Janusz Wyrwiząb

Iloczyn kartezjański tych tabel ma aż $3 \cdot 4 = 12$ wierszy

‘numer zamówienia‘	zamówienia.‘id klienta‘	klienci.‘id klienta‘	‘imię i nazwisko‘
1	3	1	Jan Czarnowidz
1	3	2	Janina Nosiwoda
1	3	3	Joachim Krzywouch
1	3	4	Janusz Wyrwiząb
2	4	1	Jan Czarnowidz
2	4	2	Janina Nosiwoda
2	4	3	Joachim Krzywouch
2	4	4	Janusz Wyrwiząb
3	2	1	Jan Czarnowidz
3	2	2	Janina Nosiwoda
3	2	3	Joachim Krzywouch
3	2	4	Janusz Wyrwiząb

Następnie z tabeli tej usuwane są wiersze, która mają różną wartość `‘id klienta‘`, zgodnie z warunkiem `WHERE zamówienia.‘id klienta‘ = klienci.‘id klienta‘`.

‘numer zamówienia‘	zamówienia.‘id klienta‘	klienci.‘id klienta‘	‘imię i nazwisko‘
~~1~~	~~3~~	~~1~~	~~Jan Czarnowidz~~
~~1~~	~~3~~	~~2~~	~~Janina Nosiwoda~~
1	3	3	Joachim Krzywouch
~~1~~	~~3~~	~~4~~	~~Janusz Wyrwiząb~~
~~2~~	~~4~~	~~1~~	~~Jan Czarnowidz~~
~~2~~	~~4~~	~~2~~	~~Janina Nosiwoda~~
~~2~~	~~4~~	~~3~~	~~Joachim Krzywouch~~
2	4	4	Janusz Wyrwiząb
~~3~~	~~2~~	~~1~~	~~Jan Czarnowidz~~
3	2	2	Janina Nosiwoda
~~3~~	~~2~~	~~3~~	~~Joachim Krzywouch~~
~~3~~	~~2~~	~~4~~	~~Janusz Wyrwiząb~~

Co pozostawia tylko trzy wiersze

‘numer zamówienia‘	zamówienia.‘id klienta‘	klienci.‘id klienta‘	‘imię i nazwisko‘
1	3	3	Joachim Krzywouch
2	4	4	Janusz Wyrwiząb
3	2	2	Janina Nosiwoda

Następnie zgodnie z treścią zapytania, z tego rezultatu są wybierane dwie kolumny i odpowiednio nazywane

zamówienie	klient
1	Joachim Krzywouch
2	Janusz Wyrwiząb
3	Janina Nosiwoda

Zastosowanie to jest tak częste, że w SQL opracowano szereg operacji do scalania dwóch tabel od razu uwzględniających tak zwany „warunek łączenia”.

2 Łączenie tabel z warunkiem

Niniejsza sekcja dotyczy SQL w ogólności, nie MySQL i MariaDB. Zacznijmy od przykładu. W naszym systemie użytkowników reprezentujemy liczbami naturalnymi. W jednej tabeli przechowujemy chińskojęzyczne książki przeczytane przez konkretne osoby, w drugiej tabeli miasta w Chinach odwiedzone przez osoby. Naszym celem jest połączenie informacji z tych dwóch tabel w ten sposób, aby w przyszłości móc wnioskować o tym, czy któraś książka o Chinach motywuje do odwiedzenia konkretnych miast.

Załóżmy najpierw, że w obu tabelach każda osoba ma przynajmniej jeden rekord. Na przykład w tabeli `książki` mamy

`osoba`	`książka`
1	Opowieści znad brzegów rzek
1	Sen czerwonego pawilonu
2	Opowieść o trzech królestwach

natomiast w tabeli `miasta`

`osoba`	`miasto`
1	Pekin
1	Tiencin
2	Wuhan

Połączenie tych tabel z warunkiem `książki.osoba = miasta.osoba` wygląda następująco (dla uproszczenia w wyniku pomijamy duplikat kolumny `osoba`)

`osoba`	`książka`	`miasto`
1	Opowieści znad brzegów rzek	Pekin
1	Opowieści znad brzegów rzek	Tiencin
1	Sen czerwonego pawilonu	Pekin
1	Sen czerwonego pawilonu	Tiencin
2	Opowieść o trzech królestwach	Wuhan

Zwróćmy uwagę, że połączenia „każdy z każdym” w ramach tej samej osoby są tu oczekiwanym działaniem, jeśli weźmiemy pod uwagę, że operacja łączenia wywodzi się z pojęcia iloczynu kartezjańskiego z filtrowaniem. Połączenie to jest jednoznaczne, ponieważ każda osoba występująca w pierwszej tabeli, jest również obecna w drugiej tabeli i odwrotnie.

W SQL operacja taka nazywa się `INNER JOIN` i ma dwa rodzaje składni

```
tabela1 INNER JOIN tabela2 ON warunek
```

lub

```
tabela1 INNER JOIN tabela2 USING (kolumny)
```

Wróćmy do przykładu z poprzedniej sekcji.

```
SELECT zamówienia.‘numer zamówienia‘ as zamówienie,
klienci.‘imię i nazwisko‘ as klient
FROM zamówienia CROSS JOIN klienci
WHERE zamówienia.‘id klienta‘ = klienci.‘id klienta‘;
```

Ten sam rezultat uzyskamy pisząc

```
SELECT zamówienia.‘numer zamówienia‘ as zamówienie,
klienci.‘imię i nazwisko‘ as klient
FROM zamówienia INNER JOIN klienci ON zamówienia.‘id klienta‘ = klienci.‘id klienta‘;
```

lub

```
SELECT zamówienia.‘numer zamówienia‘ as zamówienie,
klienci.‘imię i nazwisko‘ as klient
FROM zamówienia INNER JOIN klienci USING(‘id klienta‘);
```

Oba zapisy są nieznacznie krótsze, ale większą zaletą jest to, że SQL od razu wie, że filtrowanie z warunkiem jest naszą intencją i nie tworzy dużej tabeli rozmiaru $N \times M$ wierszy, tylko dla każdego wiersza z lewej tabeli (w przykładzie `zamówienia`) w prawej tabeli (w przykładzie `klienci`) wyszukuje wierszy pasujących do niego. W najgorszym przypadku[1] zajmie to tyle samo czasu, ale o wiele mniej pamięci.

Jeśli jednak dopuścimy możliwość występowania rozbieżności i dodamy osobę, która przeczytała chińską książkę, ale nie była w Chinach i osobę, która była w Chinach, ale nie czytała ani jednej chińskiej książki, na przykład Załóżmy najpierw, że w obu tabelach każda osoba ma przynajmniej jeden rekord. Na przykład w tabeli `książki` mamy

`osoba`	`książka`
1	Opowieści znad brzegów rzek
1	Sen czerwonego pawilonu
2	Opowieść o trzech królestwach
3	Wędrówka na Zachód

natomiast w tabeli `miasta`

`osoba`	`miasto`
1	Pekin
1	Tiencin
2	Wuhan
4	Tongli

Połączenie tych tabel z warunkiem `książki.osoba = miasta.osoba` za pomocą `CROSS JOIN` lub `INNER JOIN` da ten sam rezultat, co przykład bez dodatkowych wierszy.

[1] Później, gdy poznamy koncepcję indeksów, dowiemy się, że jeśli kolumna używana do łączenia, jest indeksem w tabeli po prawej stronie lub precyzyjniej prefiksem pewnego indeksu, złożoność czasowa dodatkowo zmniejszy się z $\Theta(NM)$ do $\Theta(N \log(M))$.

osoba	książka	miasto
1	Opowieści znad brzegów rzek	Pekin
1	Opowieści znad brzegów rzek	Tiencin
1	Sen czerwonego pawilonu	Pekin
1	Sen czerwonego pawilonu	Tiencin
2	Opowieść o trzech królestwach	Wuhan

Efekt jest identyczny, ponieważ iloczyn kartezjański łączy elementy parami „każdy z każdym” — a ponieważ nowe wiersze nie mają pary w drugiej tabeli, dla żadnego wiersza warunek filtrujący nie będzie spełniony.

Niekiedy jednak informacja, że dany wiersz nie ma pary w drugiej tabeli, jest interesująca. Na przykład możemy być zainteresowani tym, którzy klienci się zarejestrowali w sklepie, ale nie zrobili jeszcze zakupów, zrobili zakupy, ale nie napisali komentarza, i temu podobne. Oprócz wyniku jak powyżej, z instrukcji `INNER JOIN`, sens mają następujące operacje:

1. `LEFT OUTER JOIN`, który oprócz wierszy w obu tabelach, poda informacje o tych wierszach, które są tylko w tabeli po lewej stronie operatora łączenia,

2. `RIGHT OUTER JOIN`, który oprócz wierszy w obu tabelach, poda informacje o tych wierszach, które są tylko w tabeli po prawej stronie operatora łączenia,

3. `FULL OUTER JOIN`, który oprócz wierszy w obu tabelach, poda informacje o tych wierszach, które są tylko w tabeli po dowolnej stronie operatora łączenia.

Wynikiem działania `LEFT OUTER JOIN` w przypadku byłoby

osoba	książka	miasto
1	Opowieści znad brzegów rzek	Pekin
1	Opowieści znad brzegów rzek	Tiencin
1	Sen czerwonego pawilonu	Pekin
1	Sen czerwonego pawilonu	Tiencin
2	Opowieść o trzech królestwach	Wuhan
3	Wędrówka na Zachód	NULL

w przypadku `RIGHT OUTER JOIN`

osoba	książka	miasto
1	Opowieści znad brzegów rzek	Pekin
1	Opowieści znad brzegów rzek	Tiencin
1	Sen czerwonego pawilonu	Pekin
1	Sen czerwonego pawilonu	Tiencin
2	Opowieść o trzech królestwach	Wuhan
4	NULL	Tongli

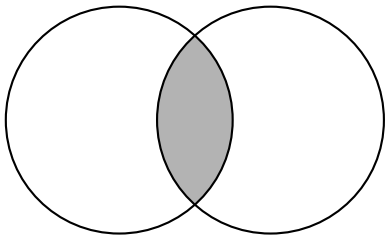
a w przypadku `FULL OUTER JOIN`

osoba	książka	miasto
1	Opowieści znad brzegów rzek	Pekin
1	Opowieści znad brzegów rzek	Tiencin
1	Sen czerwonego pawilonu	Pekin
1	Sen czerwonego pawilonu	Tiencin
2	Opowieść o trzech królestwach	Wuhan
3	Wędrówka na Zachód	NULL
4	NULL	Tongli

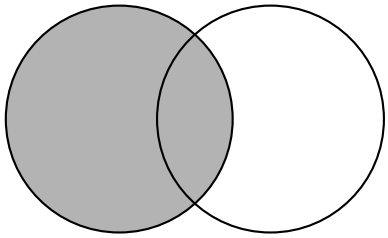
Zwróćmy uwagę, że wiersze, które nie mają pary, mają ustalone `NULL`, czyli brak danych, we wszystkich kolumnach, które pochodzą z tabeli, w której nie ma pasującego wiersza.

Operacje `JOIN` można zwizualizować za pomocą diagramów Venna.

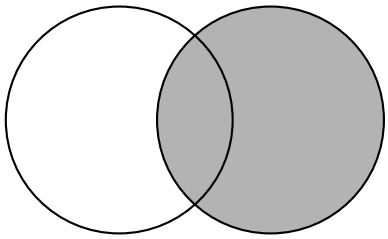
1. `INNER JOIN` ma tylko te wiersze, które są w obu tabelach.



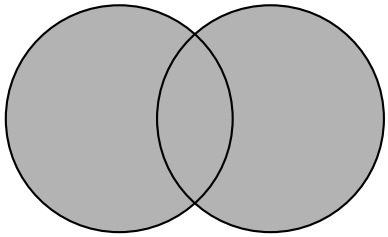
2. `LEFT OUTER JOIN` ma te wiersze, które są w obu tabelach i w lewej.



3. `RIGHT OUTER JOIN` ma te wiersze, które są w obu tabelach i w prawej.



4. `FULL OUTER JOIN` ma wiersze z obu tabel.



3 JOIN w MySQL i MariaDB

Choć w wielu silnikach baz danych (na przykład PostgreSQL, MSSQL, Oracle) dostępne są wszystkie opisane tutaj rodzaje operacji `JOIN`, w MySQL i MariaDB nie ma `FULL OUTER JOIN`. Brak ten możemy jednak uzupełnić, łącząc dwa zapytania za pomocą `UNION ALL`. Niedziałający kod

```
SELECT zamówienia.‘numer zamówienia‘ as zamówienie,
       klienci.‘imię i nazwisko‘ as klient
FROM zamówienia FULL OUTER JOIN klienci USING(‘id klienta‘);
```

możemy zastąpić kodem

```
(SELECT zamówienia.‘numer zamówienia‘ as zamówienie,
        klienci.‘imię i nazwisko‘ as klient
 FROM zamówienia LEFT OUTER JOIN klienci USING(‘id klienta‘))
UNION ALL
(SELECT zamówienia.‘numer zamówienia‘ as zamówienie,
        klienci.‘imię i nazwisko‘ as klient
 FROM zamówienia RIGHT OUTER JOIN klienci USING(‘id klienta‘)
 WHERE zamówienia.‘id klienta‘ IS NULL);
```

Pierwsze z zapytań w `UNION ALL` jest zapytaniem `LEFT OUTER JOIN`, więc zawiera w sobie już wiersze znajdujące się w obu tabelach i po lewej, zatem brakuje tylko tych, które są jedynie w prawej. W tym celu stosujemy tak zwany wzorzec „anty-łączenia” — wykonujemy łączenie `RIGHT OUTER JOIN` zwracające wiersze z obu tabel i tych tylko w prawej. Ponieważ jednak wiersze z obu tabel znajdują się już w wynikach pierwszego zapytania, wykluczamy je warunkiem `WHERE`, pozostawiając jedynie te wiersze, które znajdują się jedynie w prawej tabeli.

Kolejną różnicą w zapytaniach `JOIN` jest to, że różnie traktowane są warunki `USING` oraz `ON`. Pozornie warunek `USING(x, y, z)` jest tożsamy warunkowi

```
ON tabela1.x = tabela2.x AND tabela1.y = tabela2.y AND tabela1.z = tabela2.z
```

jednakże w pierwszym przypadku w wyniku będą trzy kolumny (bez duplikatów) a w drugim przypadku będzie sześć kolumn (z duplikatami).

Istnieje również wiele skrótów. Na przykład:

1. zamiast `INNER JOIN` możemy zapisać `JOIN` lub przecinek,
2. zamiast `LEFT OUTER JOIN` możemy napisać `LEFT JOIN`,
3. zamiast `RIGHT OUTER JOIN` możemy napisać `RIGHT JOIN`.

Co więcej, `CROSS JOIN` jest synonimem dla `INNER JOIN` bez podanego warunku `ON` lub `USING`, możemy więc również zamiast `CROSS JOIN` użyć `JOIN` lub przecinka.

Na przykład

```
SELECT książka, miasto FROM książki, miasta;
```

jest skrótem dla

```
SELECT książka, miasto FROM książki CROSS JOIN miasta;
```

natomiast

```
SELECT książka, miasto FROM książki, miasta USING(osoba);
```

skrótem dla

```
SELECT książka, miasto FROM książki INNER JOIN miasta USING(osoba);
```

Skróty są czasem pomocne, jednakże niekiedy zaciemniają intencję piszącego zapytanie. Jeśli z powodu stosowania skrótów, trzeba się będzie zastanawiać „co autor miał na myśli", lepiej użyć pełnej formy.

Kolejnym skrótem są tak zwane naturalne łączenia. Jeśli w obu tabelach znajdują się kolumny o tych samych nazwach i wszystkie te kolumny umieścilibyśmy w warunku `USING`, na przykład

```
SELECT książka, miasto FROM książki INNER JOIN miasta USING(osoba);
```

zamiast tego możemy napisać

```
SELECT książka, miasto FROM książki NATURAL INNER JOIN miasta;
```

MySQL lub MariaDB sprawdzi wtedy, jakie kolumny mają te same nazwy i wykona łączenie z ich użyciem (ponieważ jest to skrót do `USING` a nie `ON`, kolumny wyniku nie będą miały duplikatów). Kolejnymi skrótami są

1. zamiast `NATURAL INNER JOIN` możemy zapisać `NATURAL JOIN`,
2. zamiast `NATURAL LEFT OUTER JOIN` możemy napisać `NATURAL LEFT JOIN`,
3. zamiast `NATURAL RIGHT OUTER JOIN` możemy napisać `NATURAL RIGHT JOIN`.

Inne połączenia słowa `NATURAL` z `JOIN` nie mają sensu, w szczególności połączenie to nie ma sensu dla iloczynu kartezjańskiego.