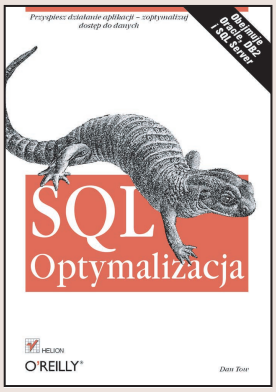
SQL. Optymalizacja



Autor: Dan Tow
Tłumaczenie: Marek Pałczyński (rozdz. 1 –; 6, dod. C),
Tomasz Pędziwiatr (rozdz. 7 –; 10, dod. A, B)
ISBN: 83-7361-423-0
Tytuł oryginału: SQL Tuning
Format: B5, stron: 348

Instrukcje SQL są kluczowymi elementami typowych aplikacji bazodanowych, a więc efektywność ich wykonywania decyduje w głównym stopniu o wydajności samych aplikacji. Twórcy aplikacji bazodanowych i administratorzy baz danych często spędzają długie godziny w celu upewnienia się, że dostęp do danych istotnie odbywa się po najszybszych ścieżkach, czyli że plany wykonywania wyrażeń SQL są optymalne. Wiąże się z tym między innymi rozważanie wzajemnego związku między strukturą wyrażeń SQL a planami ich wykonywania.

ka ta poświęcona jest jednemu z kluczowych aspektów tego związku, często niedocenianemu, aczkolwiek niezmiernie istotnemu — wyborowi odpowiedniego planu wykorzystywanego przez określone zapytanie. Autor prezentuje matematyczną metodą optymalizacji wyrażeń SQL, opierającą się na dobrze zdefiniowanym algorytmie postępowania i prowadzącą do znajdowania optymalnych (lub niemal optymalnych) planów wykonania dla określonych wyrażeń; jest to naprawdę atrakcyjna alternatywa dla poszukiwań metodą prób i błędów, rzadko dającą optymalne rezultaty. Czytelnik znajdzie w niniejszej książce opis wielu szczegółowych zagadnień związanych z optymalizacją wyrażeń SQL i baz danych w ogólności, między innymi takich jak:

- Buforowanie danych i zarządzanie tabelami
- Indeksowanie, implementowanie indeksów i związane z tym koszty
- Filtrowanie zawartości tabel i jego związek z indeksowaniem
- Złączenia tabel i metody ich realizacji
- Analiza planów wykonywania zapytań i zarządzanie tymi planami w bazach danych Oracle, MS SQL Server i DB2
- Sporządzanie diagramów zapytań i wykorzystywanie ich do celów optymalizacji złożonych zapytań oraz do wyznaczania najlepszych planów wykonywania
- Specjalne metody optymalizacji szczególnych przypadków, w których standardowe postępowanie okazuje się niewystarczające: buforowanie wielokrotnie wydawanych zapytań, łączenie i upodobnianie zapytań itp.

Treści poszczególnych rozdziałów towarzyszą ćwiczenia kontrolne, a całość wieńczy prezentacja zastosowania opisywanych koncepcji w (kompletnie opisanym) procesie optymalizowania konkretnej aplikacji.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

7

Tworzenie diagramów i optymalizacja złożonych zapytań SQL

Tworzenie diagramów i optymalizacja złożonych zapytań SQL Jak dotąd, nauczyliśmy się optymalizować zapytania na rzeczywistych tabelach oraz tworzyć dla nich diagramy, które spełniają różne wymagania odnoszące się do normalnych zapytań biznesowych:

- Zapytanie przedstawione jest na jednym drzewie.
- Drzewo ma jedno źródło, dokładnie jedną tabelę bez złączeń z jej kluczami głównymi. Wszystkie węzły, inne niż węzeł źródłowy, mają pojedyncze, skierowane ku nim połączenia ze znajdującymi się powyżej węzłami szczegółowymi, ale każdy węzeł może być na szczycie dowolnej ilości skierowanych ku dołowi połączeń.
- Wszystkie złączenia mają skierowane ku dołowi strzałki (złączenia, które są unikalne na jednym z końców).
- Złączenia zewnętrzne są niefiltrowane, skierowane ku dołowi, kolejne złączenia znajdujące się poniżej są także zewnętrzne.
- Pytanie, na które zapytanie SQL udziela nam odpowiedzi jest w gruncie rzeczy pytaniem o encję znajdującą się na samej górze (źródło) drzewa lub odnośnie agregacji tej encji.
- Pozostałe tabele dostarczają jedynie referencyjnych danych, umieszczonych w określonej strukturze jedynie przez wzgląd na normalizację.

Nazwałem zapytania spełniające powyższe warunki *zapytaniami prostymi*, chociaż jak mogliśmy się przekonać w rozdziale 6., mogą one zawierać dowolną ilość złączeń, a ich optymalizacja może być całkiem trudna, zwłaszcza w rzadkich przypadkach węzłów o podobnych współczynnikach filtrowania lub kiedy istnieje ukryte filtrowanie złączeń.

Zapytania, które nie mają tak prostej formy będziemy nazywać *zapytaniami złożonymi*. Jak pokażę w tym rozdziale, niektóre złożone zapytania są wynikiem *błędów*: w projekcie bazy danych, aplikacji lub implementacji. Błędy tego typu powodują, że bardzo łatwo jest utworzyć nieprawidłowe zapytanie. W tym rozdziale nauczymy się, jak rozpoznawać anomalie występujące w diagramie zapytania, które mogą być ostrzeżeniem o istnieniu błędu w konstrukcji zapytania. Nauczymy się również, jak naprawić te funkcjonalne lub związane z projektem błędy, przy czym niejednokrotnie zwiększenie wydajności będzie efektem ubocznym naszych działań. Poprawki te przekształcają zazwyczaj zapytanie do prostej formy lub na tyle do niej zbliżonej, aby można zastosować metody przedstawione w książce wcześniej.

Niektóre złożone zapytania wykraczają poza wszystkie formy, dla których opisywałem tworzenie diagramów, wykorzystując podzapytania, widoki lub klauzule, takie jak `UNION` i `UNION ALL`. Takie złożone zapytania mają zazwyczaj dużą funkcjonalność i są często spotykane, musimy więc znaleźć metodę na stworzenie dla nich diagramu oraz na ich optymalizację. Cel ten osiągniemy poprzez rozwinięcie przedstawionych wcześniej metod adekwatnych dla prostych zapytań.

Niestandardowe diagramy złączeń

Jeśli zapytanie zawiera jedynie proste tabele (nie widoki), bez podzapytań, bez klauzul operacji grupowych, takich jak `UNION`, zawsze można utworzyć jakiś diagram zapytania przez stosowanie metod przedstawionych w rozdziale 5. Jednakże zdarza się, że diagram ma pewne cechy, które nie pozwalają na stosowanie opisanego wcześniej, zwykłego szablonu drzewa złączeń. Opiszę te nieprawidłowości pojedynczo i pokażę jak sobie z nimi radzić.

Aby zilustrować anomalie, przedstawię niecałkowite diagramy złączeń, w których części nie mające znaczenia dla dyskusji są ukryte za szarymi obłokami. Skupi to naszą uwagę na tę część, która odgrywa decydującą rolę, pokaże również uniwersalność przykładu. Jako konwencję przyjmę ukrywanie nie mających znaczenia dla dyskusji części szkieletu połączeń. Liczba szarych połączeń i w ogóle ich istnienie nie ma znaczenia dla przykładu, pokazuje jedynie możliwość wykonania dodatkowych złączeń w realnych przypadkach. Sporadycznie występować będą czarne połączenia z ukrytymi częściami szkieletu zapytania. Połączenia te mają znaczenie dla dyskusji, ale ukryte części — nie.

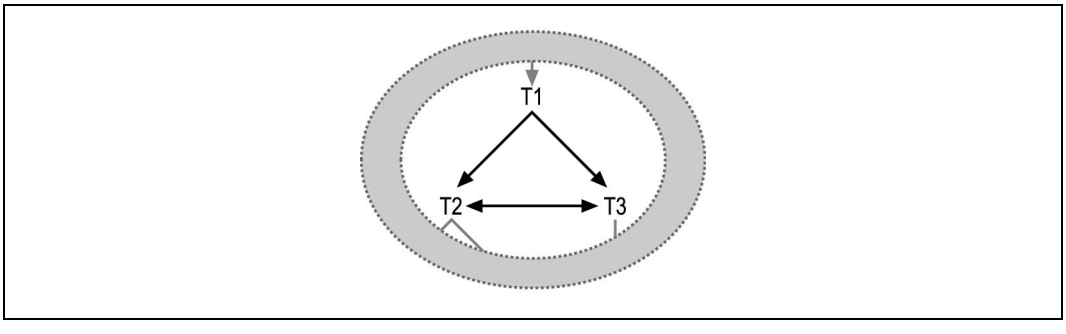
Cykliczne grafy złączeń

Jak poradzić sobie ze złączeniami, które nie mapują się na proste drzewo, ale zawierają połączenia tworzące w pewnym miejscu szkieletu pętlę? Jest kilka okoliczności, w których można napotkać tego typu diagram. W następnym podrozdziale omówię cztery przypadki z różnymi rozwiązaniami.

Teoria grafów jest gałęzią matematyki opisującą abstrakcyjne twory zwane *grafami*, które składają się z połączeń i węzłów, takich jak diagramy zapytań wykorzystywane w tej książce. W teorii grafów, graf *cykliczny* ma połączenia w formie zamkniętej pętli. W przedstawianych dalej przykładach, aż do rysunku 7.8, na diagramach występują pętle, sprawiając, że są one diagramami cyklicznymi.

Przypadek 1. Dwie tabele nadrzędne (złączone jeden-do-jednego) współdzielą tabelę szczegółową

Rysunek 7.1 ilustruje pierwszy przypadek, w którym pojedynczy klucz obcy łączy się z kluczami głównymi dwóch różnych tabel nadrzędnych.



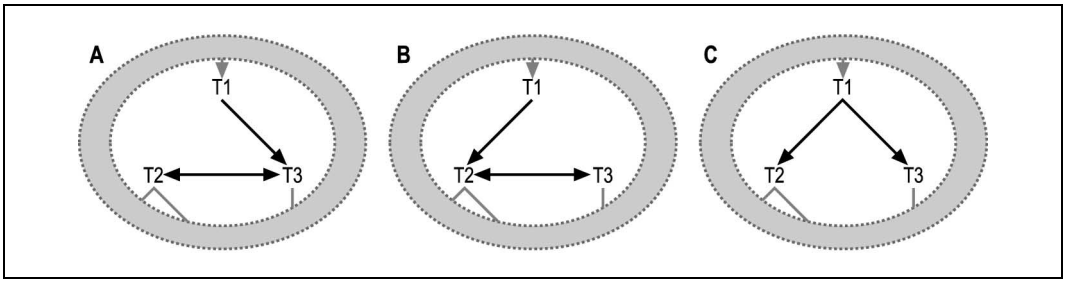
Rysunek 7.1. Przypadek 1. cyklicznego diagramu zapytania

Na podstawie tego przypadku możemy wnioskować, że SQL może wyglądać w następujący sposób:

```
SELECT ...
FROM ... T1, ... T2, ... T3, ...
WHERE ... T1.FKey1=T2.PKey2
    AND T1.FKey1=T3.PKey3
    AND T2.PKey2=T3.PKey3 ...
```

Nazwałem klucz obcy z tabeli `T1`, wskazujący na obie tabele przez `FKey1`, a klucze główne tabeli `T2` i `T3` odpowiednio przez `PKey2` i `PKey3`. Ponieważ wszystkie trzy złączenia występują jawnie w SQL-u, złączenie cykliczne jest oczywiste, ale należy zwrócić uwagę, że dowolne z tych połączeń mogłoby być nieobecne. W takim przypadku przechodniość (jeśli $a = b$ i $b = c$ to $a = c$) sugerowałaby istnienie tego nieobecnego warunku złączenia. To samo zapytanie moglibyśmy wówczas przedstawić w jednej z trzech postaci z rysunku 7.2.

Zwróćmy uwagę, że w wersji A i B zapytania możemy wnioskować o istnieniu nieobecnego połączenia z faktu, że połączenia między `T2` a `T3` mają strzałki na obu końcach, co oznacza złączenie jeden-do-jednego. Wersja C, dla odmiany wygląda jak zwykłe drzewo złączeń i można się nie zorientować, że ma ono cykliczne złączenia, chyba że zauważymy, iż `T1` wykorzystuje ten sam klucz obcy do połączenia z `T2` i `T3`.



Rysunek 7.2. To samo zapytanie cykliczne, w którym brakuje jednego z trzech przechodnich warunków złączenia

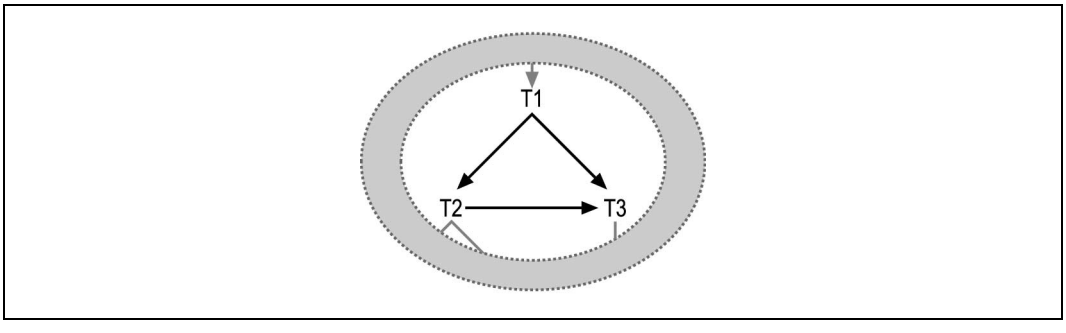
W przypadku występowania tabel jeden-do-jednego, złączenia cykliczne, takie jak na rysunku 7.1, są częste. Nie są one problemami funkcjonalnymi, chociaż ta kwestia pozostaje do rozważenia przy napotkaniu złączenia jeden-do-jednego. Można je nawet postrzegać jako dodatkowe możliwości. Jeśli osiągnęliśmy węzeł `T1`, dobrze jest mieć możliwość wyboru następnego węzła, czy to `T2`, czy `T3`, ponieważ każdy z nich może mieć lepszy współczynnik filtrowania od drugiego lub może prowadzić do dobrych filtrów leżących niżej w drzewie. W przypadku dojścia w porządku złączenia do `T2` lub `T3` przed `T1`, najbardziej użyteczne byłoby złączenie jeden-do-jednego z pozostałym węzłem (z `T2` do `T3` lub z `T3` do `T2`). Pozwala to na uzyskanie dostępu do każdego filtra wykorzystanego w tej drugiej tabeli, przed złączeniem z tabelą `T1`. Bez poziomego połączenia, połączenie `T2` z `T3` możliwe byłoby jedynie poprzez `T1`, przy prawdopodobnie większym koszcie.

Niektóre optymalizatory są wystarczająco sprytne, aby wykorzystać przechodniość do uzupełnienia brakującego złączenia, a nawet wykorzystać dodatkową swobodę w porządku złączenia. Najbezpieczniej jest jednak jawnie uwzględnić wszystkie trzy warunki złączeń, jeśli dwa spośród nich wskazują przez przechodniość na istnienie trzeciego. Z całą pewnością trzeba jawnie uwzględnić w SQL-u wszystkie złączenia potrzebne dla optymalnego planu wykonania.

Istnieją szczególne przypadki, w których dwie tabele złączone jeden-do-jednego, tak jak `T2` i `T3`, są tą samą tabelą! W takiej sytuacji, każdy wiersz tabeli `T1` łączy się dwukrotnie z tym samym wierszem tabeli `T2`, co jest bardzo nieefektywne. Najbardziej oczywiste przypadki, w których ta sama tabela wykorzystana jest, poprzez użycie aliasów, dwukrotnie w klauzuli `FROM`, są jednak mało prawdopodobne. Dzieje się tak właśnie dlatego, że są one zbyt oczywiste, aby pozostać niezauważonymi. Jednakże dwukrotne złączenie z tą samą tabelą może być bardziej subtelne i można je pominąć w przeglądzie kodu. Może się to zdarzyć, gdy synonim lub widok ukrywa prawdziwą tożsamość tabeli za co najmniej jednym aliasem zapytania. Tak czy inaczej, najlepiej usunąć z zapytania zbyteczną referencję do tabeli i przepisać wszystkie odniesienia kolumn oraz wszystkie złączenia znajdujące się poniżej, tak aby odnosiły się do pozostałego aliasu.

Przypadek 2. Każda z nadrzędnych tabel szczegółowych ma kopie klucza obcego wskazującego na klucz główny trzeciej tabeli

Rysunek 7.3 przedstawia drugi co do ważności przypadek złączeń cyklicznych, w którym identyczne klucze obce tabel `T1` i `T2` wskazują na ten sam klucz główny tabeli `T3`.



Rysunek 7.3. Złączenie cykliczne sugerujące denormalizację

Tym razem SQL ma następującą postać:

```
SELECT ...
FROM ... T1, ... T2, ... T3, ...
WHERE T1.FKey1=T2.PKey2
  AND T1.FKey2=T3.PKey3
  AND T2.FKey2=T3.PKey3 ...
```

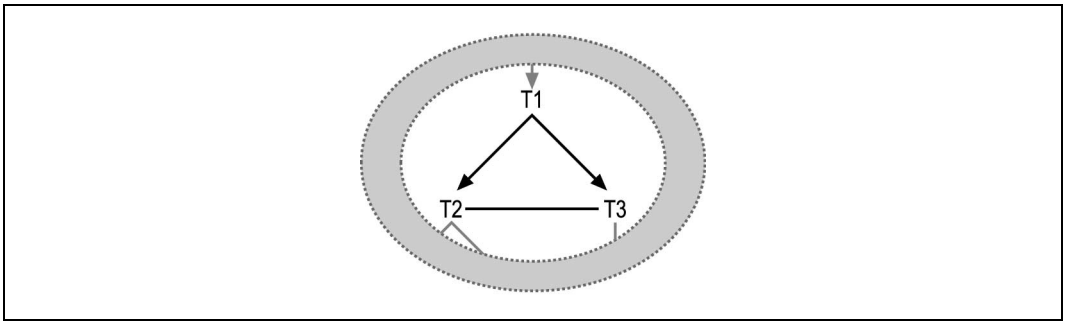
W tym wyrażeniu oznaczyłem klucze obce wskazujące z `T1` do `T2` i `T3` odpowiednio przez `FKey1` i `FKey2`. Z przechodniości wynika, że kolumna klucza obcego `T2.FKey2` ma taką samą wartość, co `T1.FKey2`, ponieważ obie łączą się z `T3.PKey3`. Klucze główne tabel `T2` i `T3` oznaczyłem odpowiednio przez `PKey2` i `PKey3`. Najbardziej prawdopodobnym wytłumaczeniem faktu, że `T1` i `T2` łączą się z tą samą tabelą `T3` poprzez jej klucz główny jest to, że klucze obce z tabel `T1` i `T2` są nadmiarowe. W tym scenariuszu, kolumna `FKey2` tabeli szczegółów `T1` denormalizuje dane jej tabeli nadrzędnej. Dane te zawsze pasują do wartości `FKey2` w odpowiadającym wierszu `T2`.

Ewentualnie wartości `FKey2` powinny pasować, ale czasem niestety tak nie jest, ponieważ zdenormalizowane dane bardzo często nie są zsynchronizowane.

W rozdziale 10. opisano wszystkie za i przeciw denormalizacji w przypadkach takich jak ten. W skrócie, jeśli denormalizacja jest uzasadniona, można przypuszczać, że dodatkowe połączenie w diagramie zapytania da dostęp do lepszego planu wykonania. Jednak bardziej prawdopodobne jest, że denormalizacja jest błędem, za którym idą większe koszty i ryzyko z nią związane niż spodziewane zyski. Wyeliminowanie denormalizacji usunęłoby klucz obcy `FKey2` z tabeli `T1`, a tym samym usunęłoby także połączenie `T1` z `T3`, sprawiając, że diagram zapytania stałby się drzewem.

Przypadek 3. Filtr dwuwęzłowy (nieunikalny na obu końcach) pomiędzy węzłami, które są już połączone poprzez normalne złączenia

Rysunek 7.4 pokazuje trzeci przypadek złączenia cyklicznego. Tym razem mamy normalne połączenia ku dołowi od `T1` do `T2` i `T3`, ale mamy także trzeci, niezwykły warunek złączenia pomiędzy `T2` i `T3`, który nie wykorzystuje kluczy głównych żadnej z tabel.



Rysunek 7.4. Złączenie cykliczne z filtrem dwuwęzłowym

Ponieważ żaden klucz główny nie jest wykorzystany przy złączeniu `T2` z `T3`, połączenie pomiędzy nimi nie ma strzałki na żadnym z końców.

SQL dla rysunku 7.4 ma postać:

```
SELECT ...
FROM ... T1, ... T2, ... T3, ...
WHERE ... T1.FKey1=T2.PKey2
   AND T1.FKey2=T3.PKey3
   AND T2.Kol2<JestJakosPorownywalneZ>T3.Kol3 ...
```

Jeśli na przykład `T1` byłoby tabelą `Orders` łączącą się z tabelami `Customers` — `T2` i `Salespersons` — `T3`, zapytanie mogłoby zwrócić zamówienia, w których klienci są przypisani do innych regionów niż sprzedawcy odpowiedzialni za zamówienie.

```
SELECT ...
FROM Orders T1, Customers T2, Salespersons T3
WHERE T1.Customer_ID=T2.Customer_ID
   AND T1.Salesperson_ID=T3.Salesperson_ID
   AND T2.Region_ID!=T3.Region_ID
```

W tym przypadku, warunek `T2.Region_ID!=T3.Region_ID` jest technicznie złączeniem, ale lepiej postrzegać go jako warunek filtrowania, którego działanie wymaga wierszy z dwóch różnych tabel. Jeśli zignorujemy bezstrzałkowe połączenie pomiędzy `T2` i `T3`, osiągniemy `T1` zanim będziemy mogli zastosować dwuwęzłowy filtr na `Region_ID`. Jedynymi dozwolonymi porządkami złączeń, które unikają bezpośredniego, niestandardowego połączenia pomiędzy `T2` i `T3` są:

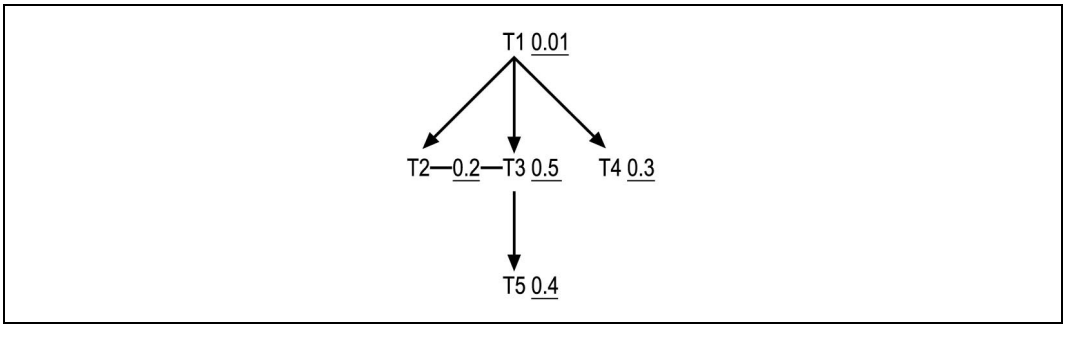
```
(T1, T2, T3)
(T1, T3, T2)
```

```
(T2, T1, T3)
(T3, T1, T2)
```

Dowolny porządek, inny niż powyższe cztery (taki jak `(T2, T3, T1)`), spowodowałby katastrofalny nawał danych po złączeniu wiele-do-wielu z drugą tabelą. Byłby to niemal iloczyn kartezjański wierszy tabeli `T2` i `T3`. We wszystkich tych bezpiecznych porządkach złączeń tabela `T1` znajduje się na pierwszym bądź drugim miejscu, zanim osiągniemy `T2` i `T3`. Te porządki złączeń tworzą zatem dwa zwykłe złączenia wiele-do-jednego pomiędzy szczegółową tabelą `T1` a jej nadrzędnymi tabelami `T2` i `T3`.

Rzadki filtr dwuwęzłowy nie ma żądanego działania w momencie osiągnięcia pierwszej z filtrowanych tabel, dopiero potem, po osiągnięciu drugiej tabeli, odrzuca część wierszy, tak jak to czyni zwykły filtr. Z takiej perspektywy, radzenie sobie z tym przypadkiem jest proste — załóżmy, że filtr nie istnieje (lub nie jest bezpośrednio dostępny) aż do momentu złączenia z jedną z tabel, do których filtr się odnosi. Jednak gdy tylko osiągniemy dowolny koniec dwuwęzłowego filtra, koniec przeciwny wchodzi w posiadanie lepszego współczynnika filtrowania i staje się bardziej atrakcyjny jako następny węzeł do złączenia.

Rysunek 7.5 pokazuje specyficzny przykład z filtrem dwuwęzłowym, w którym ułamek wierszy zwykłego złączenia `T1` z `T2` i `T3`, które spełniają dodatkowo dwuwęzłowy warunek filtrowania, jest równy 0,2. W tym przypadku wybralibyśmy pierwotnie porządek złączeń niezależny od istnienia filtru dwuwęzłowego, uwzględniając jedynie zwykłe złączenia. Jednak gdy tylko osiągniemy złączenie z tabelą `T2` lub `T3`, ta druga jeszcze nie złączona tabela otrzymuje nowy, znacznie bardziej odpowiedni współczynnik złączenia równy współczynnikowi pierwotnemu (1,0 dla `T2` i 0,5 dla `T3`) pomnożonemu przez 0,2.



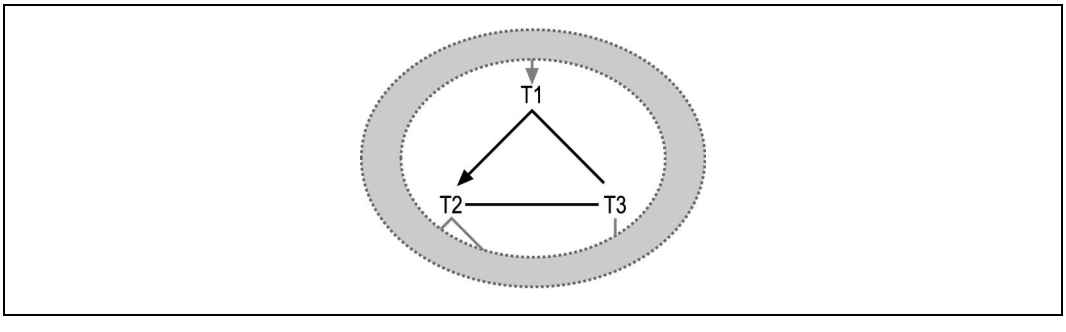
Rysunek 7.5. Filtr dwuwęzłowy z jawnym współczynnikiem filtrowania

Przy optymalizacji zapytania przedstawionego na rysunku 7.5 należy postępować zgodnie z normalną procedurą, ignorując dwuwęzłowy filtr pomiędzy `T2` i `T3`, oczywiście, aż do momentu osiągnięcia jednej z tych tabel. Tabela wejściowa to `T1`, po niej należy dodać tabelę `T4` ze względu na najlepszy zwykły filtr poniżej tabeli `T1`. `T3` ma następny w kolejności współczynniki filtrowania równy 0,5, tak więc wystąpi ona po `T4` w porządku złączenia. Mamy teraz wybór pomiędzy `T2` i `T5`. Ale ponieważ `T2`, ze względu na wcześniejsze osiągnięcie `T3`, ma aktywowany filtr dwuwęzłowy, jego współczynnik filtrowania jest równy 0,2, i to jest on korzystniejszy niż `T5`. Dlatego łączymy go jako następny w kolejności. W efekcie mamy najlepszy porządek złączenia `(T1, T4, T3, T2, T5)`.

Złączenie z T2 w przytoczonym przykładzie jest zwykłym złączeniem wykorzystującym zagnieżdżone pętle w indeksie na kluczu głównym tabeli T2, wychodząc od klucza obcego z T1. Unikajmy zagnieżdżonych pętli w tabeli z filtrem dwuwęzłowym. Odnosząc się do SQL-a tuż przed rysunkiem 7.5, byłoby znacznie lepiej dosięgnąć tabeli Customers, wykorzystując pętle zagnieżdżone przy złączeniu T1.Customer_ID=T2.Customer_ID niż przy złączeniu dwuwęzłowym T2.Region_ID!=T3.Region_ID.

Przypadek 4. Złączenie wieloczęściowe z dwóch kluczy obcych jest rozłożone na złożone klucze główne dwóch tabel

Na zakończenie rozważań na rysunku 7.6 pokazano czwarty przypadek złączeń cyklicznych. Występują tu dwa niestandardowe złączenia z T3. Żadne z nich nie wykorzystuje całego klucza głównego tej tabeli ani kluczy głównych tabel na przeciwnych końcach tych złączeń. Jeśli taki przypadek braku połączenia z całym kluczem głównym na co najmniej jednym końcu każdego złączenia jest „zły", wówczas przypadek 4. jest zazwyczaj takim, w którym dwukrotne zło staje się dobrem!



Rysunek 7.6. Złączenie cykliczne z dwoma niestandardowymi złączeniami

W sytuacji, jaką zaprezentowano na rysunku 7.6, zapytanie SQL ma postać:

```
SELECT ...
FROM ... T1, ... T2, ... T3, ...
WHERE ... T1.FKey1=T2.PKey2
   AND T1.FKey2=T3.PKeyColumn1
   AND T2.FKey3=T3.PKeyColumn2 ...
```

Taki SQL pojawia się zazwyczaj, kiedy tabela T3 ma dwuczęściowy klucz główny, a dwuczęściowy klucz obcy jest rozłożony na dwie tabele w zależności tabela nadrzędna-tabela szczegółowa.

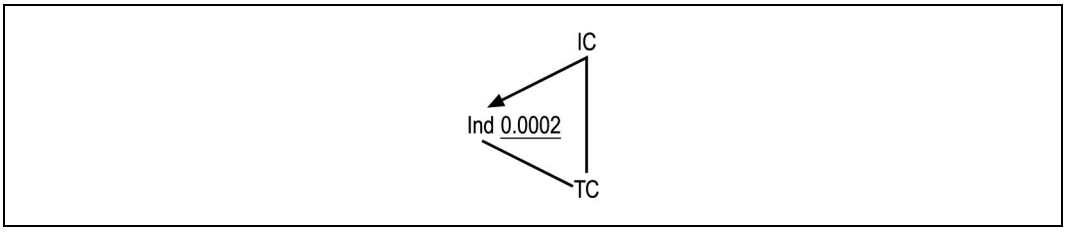
Konkretny przykład wyjaśni tę kwestię. Rozważmy tabele słownikowe oznaczone Tables, Indexes, Table_Columns i Index_Columns. Moglibyśmy wybrać dwuczęściowy klucz główny (Table_ID, Column_Number) tabeli Table_Columns, gdzie Column_Number oznacza miejsce, które zajmuje kolumna w naturalnym porządku kolumn tabeli — 1 dla pierwszej kolumny, 2 dla drugiej i tak dalej. Tabela Indexes miałaby klucz obcy do tabeli Tables na kolumnie Table_ID, a tabela Index_Columns miałaby dwuczęściowy klucz

główny (Index_ID, Column_Number). Wartość Column_Number w tabeli Index_Columns ma to samo znaczenie, co kolumna Column_Number w tabeli Table_Columns — miejsce jakie kolumna zajmuje w naturalnym porządku kolumn tabeli (nie jej miejsce w porządku indeksów, która to wartość znajduje się w Index_Position). Jeśli znalibyśmy nazwę indeksu i chcielibyśmy znaleźć listę nazw kolumn, które tworzą indeks według porządku z Index_Position, moglibyśmy napisać zapytanie:

```
SELECT TC.Column_Name
FROM Indexes Ind, Index_Columns IC, Table_Columns TC
WHERE Ind.Index_Name='PRACOWNICY_X1'
   AND Ind.Index_ID=IC.Index_ID
   AND Ind.Table_ID=TC.Table_ID
   AND IC.Column_Number=TC.Column_Number
ORDER BY IC.Index_Position ASC
```

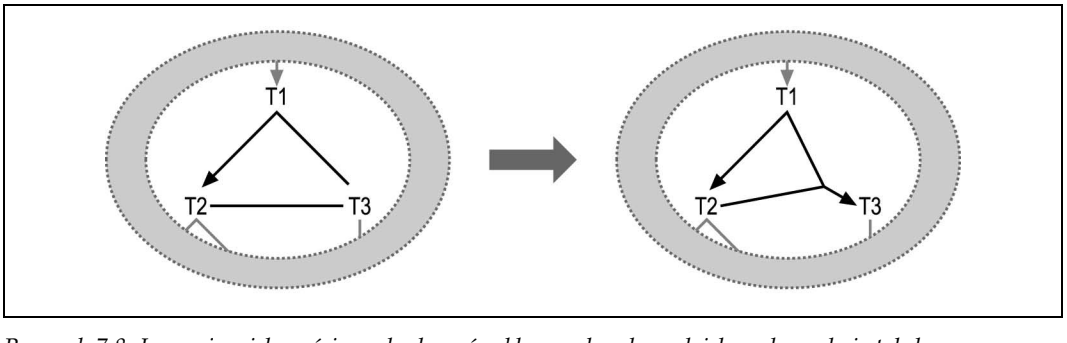
W ramach powtórki, spróbujmy utworzyć szkielet diagramu dla tego zapytania.

Jeśli warunek na Index_Name miałby współczynnik filtrowania 0,0002, diagram zapytania pozbawiony pozostałych współczynników wyglądałby jak na rysunku 7.7.



Rysunek 7.7. Konkretny przykład czwartego przypadku złączeń cyklicznych

W tym przypadku, dwukrotne „zło" (dwa złączenia, które nie łączą się z całym kluczem głównym po żadnej stronie połączenia) łączy się, tworząc w efekcie „dobro", kiedy rozpatrujemy złączenia z TC razem, ponieważ razem łączą się one z pełnym kluczem głównym tej tabeli. Możemy przetworzyć diagram tego nietypowego przypadku w sposób widoczny na rysunku 7.8.



Rysunek 7.8. Łączenie wieloczęściowych złączeń z kluczy obcych rozdzielonych na dwie tabele

Jeśli przestrzegamy reguły, aby łączyć „do" lub „z" pełnych kluczy głównych, najlepszy porządek złączenia dla rysunku 7.7 staje się jasny. Wyjdźmy z filtru na `Ind` i podążajmy za połączeniem górnym ku `IC`. Jest to tak naprawdę najlepszy plan wykonania dla tego przykładu. W przypadkach takich jak ten, należy uwzględniać niestandardowe złączenia prowadzące do wieloczęściowych kluczy głównych tylko do momentu, w którym baza danych osiągnie wszystkie węzły górne niezbędne do użycia pełnego klucza głównego.

Podsumowanie złączeń cyklicznych

Następująca lista podsumowuje metody wykorzystywane przy rozwiązywaniu problemów ze złączeniami cyklicznymi:

Przypadek 1. Dwie tabele nadrzędne (złączone jeden-do-jednego) współdzielą tabelę szczegółową

Mamy tu okazję do optymalizacji, zwiększając stopień swobody w porządku złączenia. Powinniśmy jednak rozważyć także inne rozwiązania przedstawione w dalszej części tego rozdziału. Pozwalają nam one dobrze radzić sobie ze złączeniami jeden-do-jednego.

Przypadek 2. Każda z nadrzędnych tabel szczegółowych ma kopie klucza obcego wskazującego na klucz główny trzeciej tabeli

Tutaj także mamy możliwość zwiększenia swobody w porządku złączenia, ale ten przypadek oznacza denormalizację, która zazwyczaj nie znajduje odpowiedniego uzasadnienia. Jeśli mamy wybór, powinniśmy usunąć denormalizację, chyba że zysk w tym lub innych zapytaniach potwierdza jej trafność.

W dotychczas prowadzonych rozważaniach sugerowałem idealne rozwiązania, przy założeniu posiadania całkowitej kontroli nad aplikacją, projektem bazy danych oraz SQL-em. Pokazywałem także kompromisowe rozwiązania, które mają zastosowanie w sytuacji, gdy ma się mniejszą kontrolę. Jednak czasem, gdy ma się do czynienia z nieuzasadnioną denormalizacją, w przypadku gotowego systemu, którego nie posiadamy czy na który nie mamy nawet wpływu, jedynym wyjściem kompromisowym jest nic nie robić.

Przypadek 3. Filtr dwuwęzłowy (nieunikalny na obu końcach) pomiędzy węzłami, które są już połączone poprzez normalne złączenia

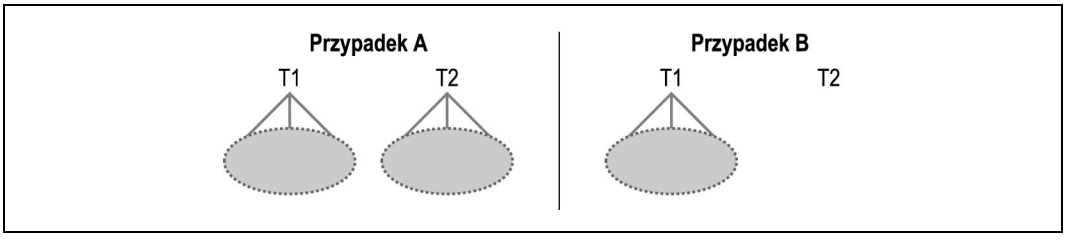
Ten przypadek należy traktować, jak gdyby nie występował w nim żaden filtr aż do momentu osiągnięcia jednego z węzłów. Wówczas w poszukiwaniu pozostałej części porządku złączenia, węzeł ów należy traktować jako mający lepszy współczynnik filtrowania.

Przypadek 4. Złączenie wieloczęściowe z dwóch kluczy obcych jest rozdzielone na złożone klucze główne dwóch tabel

Ten przypadek złączenia powinien być wykorzystywany, jedynie gdy posiadamy obie części klucza.

Rozłączone diagramy zapytań

Rysunek 7.9 pokazuje dwa przypadki rozłączonych diagramów zapytania: szkielet zapytania, w którym nie udało się połączyć wszystkich tabel zapytania w pojedynczą, połączoną strukturę. W każdym z tych przypadków mamy w pewnym sensie do czynienia z dwoma niezależnymi zapytaniami, każde z osobnym diagramem zapytania, który można optymalizować w oderwaniu od pozostałych diagramów.



Rysunek 7.9. Rozłączone diagramy zapytań

W przypadku A przedstawiłem zapytanie, które składa się z dwóch, wyglądających na niezależne, zapytań — każde z własnymi złączeniami. W przypadku B pokazałem praktycznie zwykłe zapytanie, którego jedna z tabel (tabela `T2`) jest odłączona od drzewa złączenia (tzn. nie jest złączona z żadną inną tabelą). Każdy z tych dwóch przypadków mapuje się na dwa oddzielne zapytania, które są wykonywane w obrębie pojedynczego zapytania. Co się stanie, jeśli połączymy dwa niezależne zapytania w jedno? Kiedy dwie tabele są połączone w jedno zapytanie bez jakichkolwiek warunków złączenia, baza danych zwraca iloczyn kartezjański — każdą możliwą kombinację wierszy pierwszej tabeli z wierszami tabeli drugiej. W przypadku rozłączonych diagramów, należy myśleć o wynikach zapytania reprezentowanego przez każdy niezależny szkielet zapytania (lub izolowany węzeł) jako o wirtualnej tabeli. Z tej perspektywy widać, że baza danych zwróci wszystkie kombinacje wierszy tych dwóch niezależnych zapytań. Tak więc w wyniku otrzymamy iloczyn kartezjański.

Kiedy spotkamy się z iloczynem kartezjańskim, tak jak pokazano na rysunku 7.9, powinniśmy zbadać przyczynę jego zaistnienia. Gdy ją już poznamy, będziemy mogli zadecydować, które z wymienionych poniżej działań podjąć. Będzie to uzależnione od tego, z którym spośród czterech przypadków mamy do czynienia:

Przypadek 1. W zapytaniu brakuje złączenia, które łączyłoby rozdzielone części

Dodać brakujące złączenie.

Przypadek 2. Zapytanie składa się z dwóch niezależnych zapytań, a każde z nich zwraca wiele wierszy

Wyeliminować iloczyn kartezjański poprzez osobne wykonywanie oddzielnych zapytań.

Przypadek 3. Jedno z niezależnych zapytań jest zapytaniem zwracającym jeden wiersz

Rozważyć rozdzielenie zapytań, aby zmniejszyć przepływ danych z bazy danych, szczególnie jeśli jedno z niezależnych zapytań zwraca wiele wierszy. Wykonać najpierw zapytanie jednowierszowe.

Przypadek 4. Oba niezależne zapytania są zapytaniami jednowierszowymi

Zachować zapytania w formie połączonej, chyba że jest to kłopotliwe lub trudne w konserwacji.

Zanim rozdzielimy rozłączone zapytanie na dwa niezależne zapytania, rozważmy, czy programista mógł przez nieuwagę pozostawić zapytanie bez złączenia. Na początku cyklu tworzenia systemu, najczęstszą przyczyną rozłączonych diagramów zapytań jest to, iż programiści zapominają dodać niektóre warunki złączenia, które łączą dwa rozłączone poddrzewa. W takim przypadku, powinniśmy po prostu dołożyć brakujące złączenie, dzięki czemu pozbędziemy się rozłącznych drzew. Za każdym razem, kiedy jedna z tabel w drzewie ma klucz obcy wskazujący na klucz główny tabeli źródłowej drugiego drzewa, niemal pewne jest, że brakujące złączenie zostało opuszczone przez przypadek.

Jeśli każde niezależne zapytanie zwraca wiele wierszy, liczba kombinacji przekroczy liczbę wierszy, które otrzymalibyśmy w przypadku wykonania obu zapytań oddzielnie. Jednakże zbiór kombinacji z dwóch tabel nie zawiera więcej informacji, niż można by uzyskać poprzez wykonanie zapytań oddzielnie. Dlatego generowanie nadmiarowych danych w kombinacji jest po prostu stratą czasu, przynajmniej jeśli zamierza się otrzymać czystą informację. Wobec tego lepsze może być wykonanie tych dwóch zapytań oddzielnie.

Generowanie kombinacji w iloczynie kartezjańskim rzadko bywa w jakiś sposób uzasadnione, z perspektywy wygody programowania. Jednakże zawsze istnieją metody alternatywne, pozwalające uniknąć zbędnych danych, jeśli koszt ich uzyskania jest zbyt wysoki.

Jeśli rozpatrywać problem tylko pod kątem fizycznych operacji wejścia-wyjścia, iloczyn kartezjański nie sprawiałby żadnego problemu, ponieważ nadmiarowe odczyty danych z powtórzonych zapytań byłyby najprawdopodobniej w pełni buforowane już po pierwszym odczycie. Słyszałem nawet opinie, które broniły długo wykonujących się zapytań tego typu, opierając się o niewielką ilość fizycznych operacji wejścia-wyjścia. Takie zapytania są dobrym sposobem na spalenie procesora i wygenerowanie ogromnej ilości logicznych operacji wejścia-wyjścia, jeśli kiedykolwiek zaszłaby potrzeba przeprowadzenia jakiegoś testu wytrzymałości czy innego doświadczenia. Nie mają one jednak zastosowania w aplikacjach biznesowych.

Jeśli jedno z niezależnych zapytań zwraca tylko jeden wiersz, zagwarantowane jest, że przynajmniej iloczyn kartezjański jest bezpieczny i że zwrócona liczba wierszy będzie nie większa niż liczba wierszy zwróconych przez większe z niezależnych zapytań. Jednakże istnieje wciąż niewielki koszt połączenia zapytań. Jest on związany z przesyłem danych, ponieważ lista `SELECT` połączonego zapytania może zwrócić dane z mniejszego zapytania

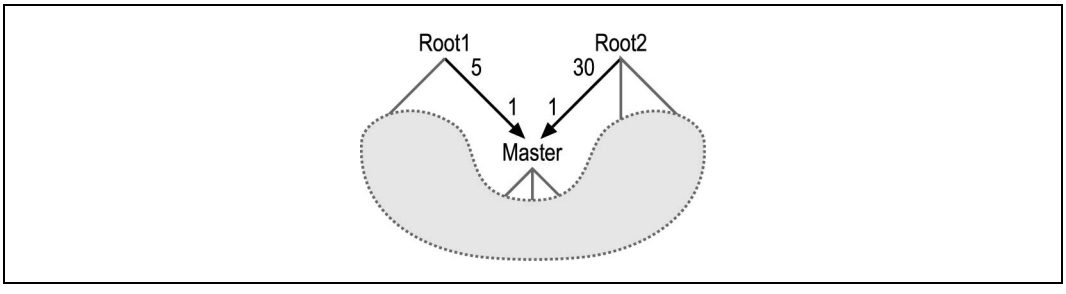
wielokrotnie, raz dla każdego wiersza zapytania wielowierszowego. Powoduje to przesył większej ilości zbędnych danych niż w przypadku rozdzielenia obu zapytań. Koszt przesyłu jest z drugiej strony kontrowany poprzez oszczędności w opóźnieniach wynikłych z przesyłu każdego pakietu — zapytanie połączone oszczędza ciągłych odwołań sieciowych do bazy danych, tak więc najlepszy wybór zależy od szczegółów. Jeśli nie rozdzielimy zapytań, optymalny plan wykonania jest prosty — najpierw należy uruchomić optymalny plan wykonania dla zapytania zwracającego pojedynczy wiersz. Następnie, w pętli zagnieżdżonej, która wykonana będzie tylko raz, trzeba uruchomić optymalny plan wykonania dla zapytania wielowierszowego. Ten połączony plan wykonania ma koszt taki sam, jak wykonanie dwóch zapytań oddzielnie. Jeśli plan zapytania wielowierszowego uruchomimy najpierw, plan pętli zagnieżdżonych będzie wymagał powtarzania dla każdego zapytania jednowierszowego tylekroć razy, ile wierszy zwróciłoby zapytanie wielowierszowe.

Połączenie zapytania jednowierszowego z zapytaniem wielowierszowym jest czasem wygodne i usprawiedliwione. Istnieje specjalny przypadek, zobrazowany na prawej połówce rysunku 7.9, w którym zapytanie jednowierszowe jest po prostu odczytem jedynego wiersza izolowanej tabeli `T2` nie mającej żadnych złączeń. Iloczyn kartezjański z izolowaną tabelą jest czasem użyteczny w pobieraniu parametrów przechowywanych w jednowierszowej tabeli parametrów, szczególnie kiedy parametry te występują jedynie w klauzuli `WHERE`, a nie w liście `SELECT`. Kiedy zapytanie zwraca dane z tabeli parametrów, okazuje się, że tańsze jest wykonanie odpowiednio połączonego zapytania niż wykonanie dwóch osobnych zapytań.

Z jeszcze rzadszym przypadkiem mamy do czynienia, gdy oba izolowane zapytania zwracają pojedynczy wiersz. Z punktu widzenia wydajności jest całkowicie uzasadnione i bezpieczne połączenie takich dwóch zapytań. Jest ono pozbawione niebezpieczeństw związanych z innymi przypadkami. Jednakże z perspektywy programowania i konserwacji oprogramowania, łączenia takich zapytań może być mylące, a oszczędności są raczej niewielkie.

Diagram zapytania z wieloma tabelami źródłowymi

Rysunek 7.10 pokazuje przykład diagramu zapytania, który nie spełnia oczekiwania odnośnie jednego źródła. Przypadek ten jest spokrewniony z poprzednim (rozłączone diagramy zapytań). Tutaj, dla każdego wiersza tabeli `Master` spełniającego warunki zapytania, zapytanie zwróci wszystkie kombinacje odpowiednich szczegółów z `Root1` i `Root2`. Mając dane współczynniki złączeń tabeli szczegółów, możemy spodziewać się wszystkich kombinacji 5 szczegółów tabeli `Root1` i 30 szczegółów tabeli `Root2`, co da nam 150 kombinacji dla każdego wiersza tabeli `Master`. Te 150 wierszy kombinacji nie zawiera więcej danych niż 5 szczegółów `Root1` w połączeniu z 30 szczegółami `Root2`, zatem szybciej jest odczytać te 5 i 30 wierszy oddzielnie, unikając iloczynu kartezjańskiego. Podczas gdy rozłączony diagram zapytania tworzy pojedynczy, duży iloczyn kartezjański, liczne węzły-źródła tworzą całą serię mniejszych iloczynów kartezjańskich, po jednym dla każdego odpowiedniego wiersza tabeli głównej.



Rysunek 7.10. Diagram zapytania o wielu tabelach źródłowych

Istnieją cztery przyczyny wystąpienia diagramu zapytania o wielu źródłach. Następująca lista pokazuje te przyczyny i opisuje odpowiadające im rozwiązania:

Przypadek 1. Brakujący warunek

W zapytaniu brakuje warunku, który zamieniłby jedną z tabel źródłowych w tabelę główną oraz złączenie jeden-do-wielu na jeden-do-jednego.

Rozwiązanie: dodać brakujący warunek złączenia.

Przypadek 2. Iloczyn kartezjański wiele-do-wielu

Zapytanie reprezentuje iloczyn kartezjański wiele-do-wielu na każdy wiersz tabeli głównej, pomiędzy tabelami szczegółów dzielącymi wspólną tabelę główną. Przypadek ten ma miejsce, gdy współczynnik złączenia tabeli szczegółów z pojedynczej współdzielonej tabeli głównej do dwóch różnych tabel źródłowych jest większy niż 1,0.

Rozwiązanie: usunąć iloczyn kartezjański poprzez rozdzielenie zapytania na dwa niezależne zapytania czytające niezależnie z tabeli źródłowej.

Przypadek 3. Współczynnik złączenia tabeli szczegółów jest mniejszy niż 1,0

Jedna ze źródłowych tabel szczegółowych łączy się ze współdzieloną tabelą główną, przy współczynniku tabeli szczegółowej złączenia mniejszym niż 1,0.

Rozwiązanie: chociaż nie jest to problem wydajnościowy, należy rozważyć odseparowanie części zapytania lub optymalizację jednej z części zapytania, tak by stała się podzapytaniem.

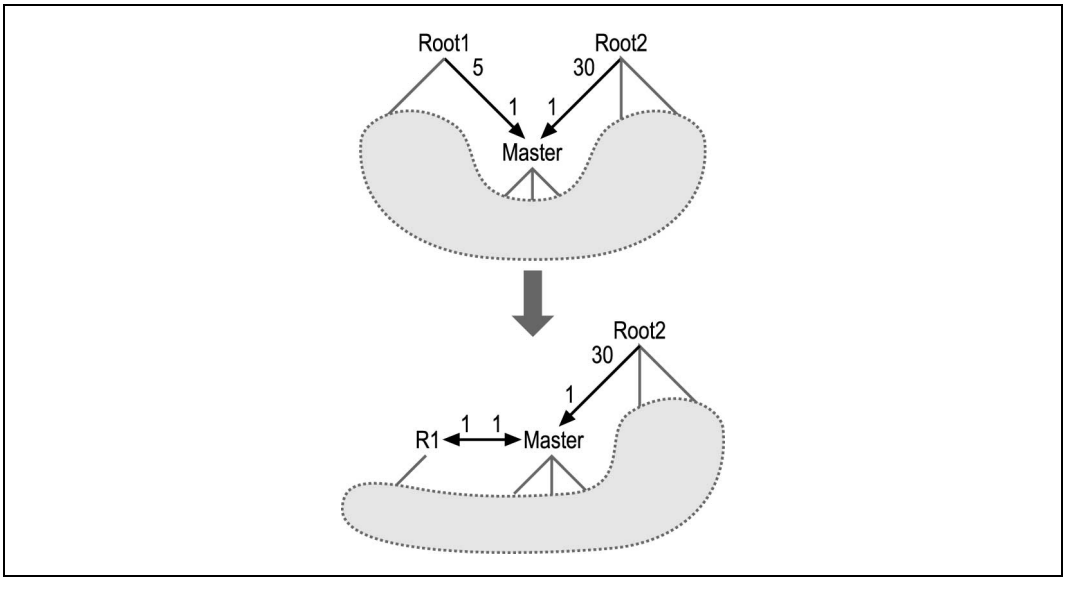
Przypadek 4. Tabela jest używana jedynie dla sprawdzenia obecności

Jedna ze źródłowych tabel szczegółów nie dostarcza żadnych danych potrzebnych w liście polecenia `SELECT` i jest włączona do zapytania jedynie dla sprawdzenia obecności.

Rozwiązanie: zamienić sprawdzenie obecności na podzapytanie.

Przypadek 1. Brakujący warunek złączenia

Występowanie drugiego węzła źródłowego najczęściej wskazuje na brak warunku złączenia, który zamieniłby jeden ze źródłowych węzłów w węzeł główny. Rysunek 7.11 pokazuje transformację, w której złączenie z tabeli `Master` do `Root1` zostało zamienione



Rysunek 7.11. Naprawianie zapytania z mnogimi węzłami źródłowymi

na złączenie jeden-do-jednego poprzez dodanie (lub rozpoznanie) dodatkowego warunku na tabeli `Root1` (przemianowana na `R1`), co zapewniło, że baza danych znajdzie co najmniej jeden wiersz w `R1` dla każdego wiersza tabeli `Master`. Jest to szczególnie prawdopodobne, jeśli `R1` zawiera szczegółowe dane związane z przedziałami czasowymi (jak np. zmieniający się podatek), które łączą rekord główny (jak encja podatku) i warunek na datę (np. żądanie *obecnej* stawki podatkowej), tworząc złączenie jeden-do-jednego.

Niejednokrotnie warunek powodujący, że złączenie staje się jeden-do-jednego, już istnieje. Wówczas powinniśmy odkryć kombinację złączenia wiele-do-jednego i tego warunku, która to kombinacja zmieni współczynnik złączenia szczegółów.

W przykładzie, w którym współczynnik złączenia tabeli szczegółów `Root1` był równy 5, współczynnik filtrowania dla takiego filtra byłby równy 0,2, lub inaczej 1/5.

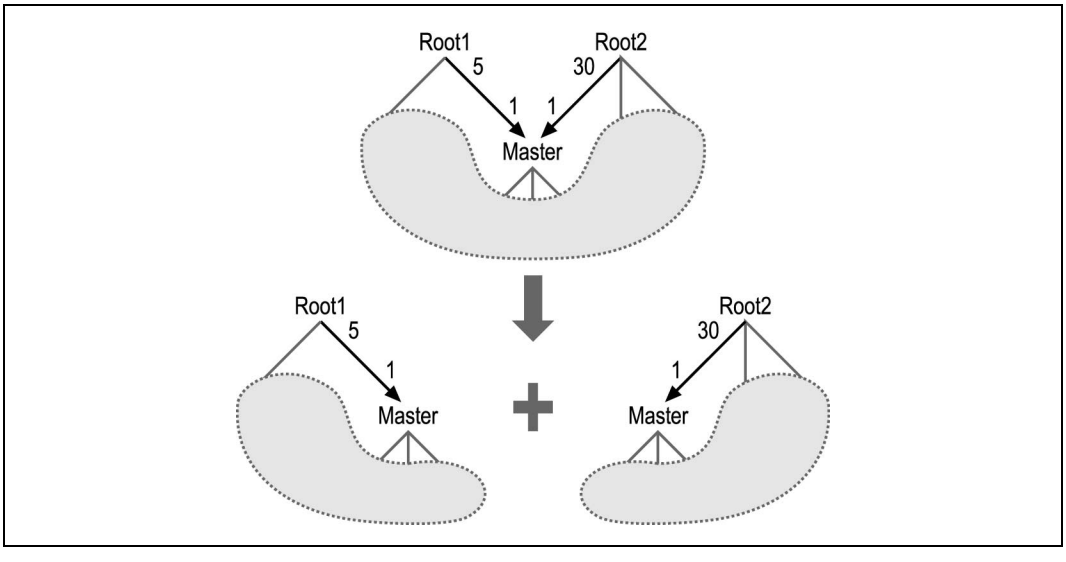
Ewentualnie, warunek, który powoduje, że złączenie jest typu jeden-do-jednego może nie być uwzględniony w zapytaniu, szczególnie jeśli rozwój aplikacji odbywał się w systemie testowym, w którym relacja jeden-do-wielu była ukryta ze względu na konkretne dane.

Poprzedni przykład dotyczący zmieniających się stóp podatkowych jest tu dobrą ilustracją. W systemie produkcyjnym może się bowiem okazać, że istnieją rekordy jedynie do obecnej stopy podatku, co ukrywa w ten sposób błąd nieuwzględnienia warunku na datę na tabeli stóp.

Niezależnie od tego czy brakuje warunku tworzącego złączenie jeden-do-jednego, czy też nie jest on rozpoznany jako połączony z tym złączeniem, powinniśmy włączyć ten warunek i rozpoznać go jako część złączenia, a nie niezależny warunek złączenia. Taki brakujący warunek złączenia jest szczególnie prawdopodobny, gdy jeden z kluczy obcych jednej z tabel źródłowych wskazuje ku dołowi na wspólną tabelę główną i jest jednocześnie częścią wieloczęściowego klucza głównego tej tabeli źródłowej.

Przypadek 2. Rozdzielanie iloczynu kartezjańskiego na wiele zapytań

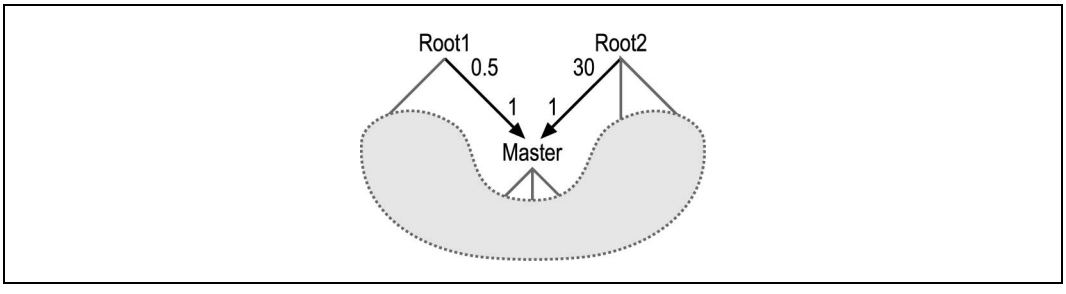
Rysunek 7.12 wskazuje inne rozwiązanie dla problemu diagramów o wielu węzłach źródłowych. To rozwiązanie jest podobne do tego omówionego wcześniej, dotyczącego jawnie wykonywanych nie połączonych zapytań, i tutaj rozdzielamy iloczyn kartezjański i zastępujemy go przez dwa rozłączne zbiory. W naszym przykładzie, zapytanie, które zwróciłoby 150 wierszy dla każdego wiersza tabeli `Master` zostało zamienione na dwa zapytania, zwracające w połączeniu po 35 wierszy na każdy wiersz tabeli `Master`. W każdym przypadku wystąpienia relacji jeden-do-wielu z tabeli głównej do dwóch różnych tabel źródłowych, możemy otrzymać dokładnie te same dane, przy dużo mniejszej liczbie odczytanych wierszy, za pomocą oddzielnych zapytań, tak jak jest to widoczne na ilustracji. Ponieważ rezultat przyjmuje zmienioną formę, potrzeba także zmienić logikę aplikacji, która obsługuje tę nową formę.



Rysunek 7.12. Rozwiązywanie problemu iloczynu kartezjańskiego za pomocą oddzielnych zapytań

Przypadek 3. Źródłowe tabele szczegółów, które łączą się zazwyczaj w stosunku nie więcej niż jeden-do-jednego

Rysunek 7.13 pokazuje przypadek wielokrotnych tabel źródłowych, w którym wydajność zapytania nie jest problemem nawet w niezmienionej postaci. Ponieważ współczynnik złączenia tabeli szczegółów z `Master` do `Root1` jest równy 0,5, nie pojawia się kartezjańska



Rysunek 7.13. Iloczyn kartezjański z niskim współczynnikiem złączenia detali

eksplozja wierszy podczas łączenia odpowiednich wierszy tabeli `Root1` z `Root2`, dla przeciętnego rekordu tabeli `Master`. Można traktować `Root1`, jak gdyby był złączony ku dołowi, faworyzując go nawet poprzez poprawienie jego współczynnika filtrowania przez owo 0,5 (w myśl specjalnej reguły z rozdziału 6. dla współczynników złączeń tabel szczegółów mniejszych niż 1,0).

Chociaż zapytanie to nie stanowi problemu z punktu widzenia optymalizacji, może ono być niepoprawne. Złączenie jeden-do-zera lub jeden-do-wielu z tabeli `Master` do `Root1` ma zazwyczaj typ jeden-do-zera lub jeden-do-jednego, co prowadzi do dobrego zachowania iloczynu kartezjańskiego. Jednakże jeśli złączenie jest zawsze typu jeden-do-wielu, trzeba wziąć pod uwagę, że rezultat może być iloczynem kartezjańskim z powtórzeniami dla danego wiersza tabeli `Root2`. Ponieważ przypadek ten jest rzadki, można z dużym prawdopodobieństwem powiedzieć, że zapytanie było zaprojektowane i przetestowane tak, by zwracało rezultaty, które mapują jeden-do-jednego z wierszami z `Root2`, a aplikacja może nawet nie działać w innych rzadkich przypadkach.

Im rzadszy jest przypadek jeden-do-wielu, tym bardziej prawdopodobne jest, że przypadek taki był zupełnie zaniedbany w projektowaniu aplikacji.

Na przykład, jeśli aplikacja wymieni dane w `Root2` po odczytaniu ich za pomocą powyższego zapytania i będzie próbowała przesłać zmiany z powrotem do bazy danych, musi ona rozważyć, która kopia powtórzonych wierszy `Root2` powinna być zapisana ponownie do bazy danych. Czy aplikacja powinna ostrzec użytkownika końcowego, że próbowała wysłać niespójne kopie? Jeśli agreguje ona dane tabeli `Root2` z zapytania, czy unika dodawania danych z powtórzonych wierszy tabeli `Root2`?

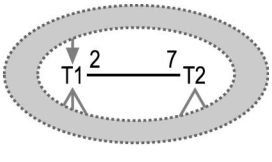
Przypadek 4. Zamiana kontroli istnienia na jawne podzapytanie

Jedno z rozwiązań funkcjonalnego problemu z rysunku 7.13 pokazane jest na rysunku 7.12 — rozłączenie zapytania na dwie części. Innym, zadziwiająco częstym rozwiązaniem jest wyizolowanie gałęzi z `Root1` w podzapytanie, zazwyczaj za pomocą warunku `EXISTS`. Rozwiązanie to jest szczególnie łatwe w zastosowaniu, jeśli oryginalne zapytanie nie wybierało kolumn z `Root1` (lub dowolnej tabeli połączonej poniżej niej poprzez ukryte, szare połączenia na rysunku 7.13). W tym dość popularnym, szczególnym przypadku,

tak naprawdę jesteśmy zainteresowani tylko tym, czy odpowiedni wiersz z `Root1` istnieje i być może spełnia jakieś warunki filtrowania. Nie interesuje nas jego zawartość czy też liczba pasujących wierszy (poza pierwszym). W dalszej części rozdziału zobaczymy, jak tworzyć diagram i optymalizować zapytania z podzapytaniami tego typu.

Złączenia bez kluczy głównych

Połączenia bez strzałek na końcach symbolizują złączenia, które nie zawierają kluczy głównych. Zazwyczaj reprezentują one niezwykłe złączenia wiele-do-wielu, chociaż w niektórych przypadkach mogą się zamieniać w złączenia wiele-do-zera lub wiele-do-jednego. W przypadku gdy nigdy nie mają one postaci wiele-do-wielu, unikalny warunek na jednym z końców jest po prostu nierozpoznany, więc trzeba dodać strzałkę na tym unikalnym końcu. Jeśli — chociaż czasami — są one typu wiele-do-wielu, wówczas spotykamy się z tymi samymi problemami (i tymi samymi rozwiązaniami), które są charakterystyczne dla diagramów zapytań o wielu węzłach źródłowych. Rysunek 7.14 przedstawia złączenie wiele-do-wielu pomiędzy `T1` a `T2`, gdzie współczynnik złączenia detali na każdym końcu jest większy niż 1,0. (Współczynniki złączenia tabel głównych istnieją jedynie na unikalnych końcach złączenia, tych ze strzałkami, więc to złączenie ma dwa współczynniki złączenia detali).



Rysunek 7.14. Złączenie wiele-do-wielu

Okazuje się, że taki przypadek jest znacznie częściej spotykany niż poprzednie przykłady niestandardowych diagramów złączeń. Chociaż współdzieli on te same przyczyny problemów i te same rozwiązania, co problem z wieloma węzłami źródłowymi, znaczna większość złączeń wiele-do-wielu występuje ze względu na brak jakiegoś warunku złączenia. Zacząć należy od sprawdzenia, czy warunki filtrowania, które istnieją już w zapytaniu powinny być traktowane jako część złączenia, ponieważ dopełniają one specyfikacji pełnego klucza głównego na jednym z końców złączenia. Przykład 5.2 z rozdziału 5. mógł być potencjalnie takim przypadkiem, z brakującym warunkiem `OT.Code_Type='STATUS_ZAMOWIENIA'` niezbędnym, aby złączenie z `OT` stało się unikalne. Gdybyśmy traktowali ten warunek jedynie jako warunek filtrowania na aliasie `OT`, złączenie z `OT` wyglądałoby jak wiele-do-wielu. Nawet jeśli nie znaleźlibyśmy brakującej części złączenia pośród warunków filtrowania tego zapytania, powinniśmy podejrzewać, że została ona opuszczona przez pomyłkę.

Ten przypadek brakujących warunków złączenia jest szczególnie częsty, kiedy projekt bazy danych pozwala na wiele typów encji lub partycji w obrębie tabeli, a programista zapomniał w zapytaniu uwzględnić warunek na typ lub partycję. Wcześniejszy przykład tabeli `Code_Translation` ma różne typy encji translacji dla każdego `Code_Type` i nie

uwzględnienie warunku na `Code_Type` spowodowałoby, że złączenie z `Code_Translation` byłoby typu wiele-do-wielu. Często zdarza się, że problem tego typu nie jest dość wcześnie zauważony w fazie testów. Dzieje się tak dlatego, że nawet jeśli projekt bazy danych pozwala na wiele typów partycji, środowisko testowe może mieć dane tylko jednego typu. Na ten stan rzeczy projektanci bardzo często się godzą. Nawet jeśli w prawdziwych danych istnieje wiele typów partycji, inna bardziej wybiórcza część klucza może sama powodować unikalność. Jest to jednocześnie szczęśliwy i nieszczęśliwy zbieg okoliczności — z jednej strony nie pozwala, aby brakujący warunek złączenia sprawił natychmiastowe problemy, z drugiej powoduje, że problem jest znacznie trudniejszy do znalezienia i naprawienia, oraz daje fałszywe wrażenie, że aplikacja nie zawiera błędów. Odnalezienie i wstawienie brakującego warunku złączenia może zwiększyć wydajność tylko nieznacznie poprzez uczynienie złączenia bardziej selektywnym, ale może także mieć ogromną wartość, jeśli naprawi niebezpieczny w skutkach, niewidoczny błąd.

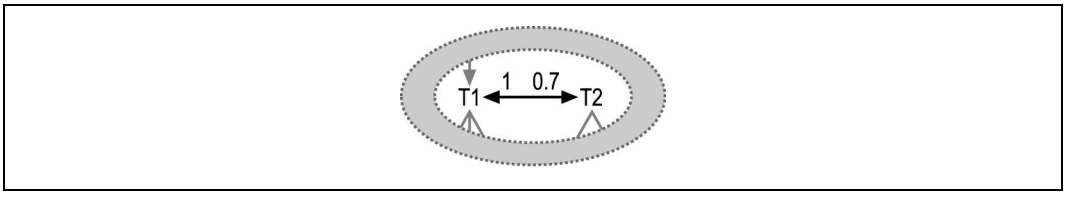
Poprzez bezpośrednią analogię z diagramem zapytania o wielu węzłach źródłowych, rozwiązania problemu złączenia wiele-do-wielu mapuje się na podobne rozwiązania diagramu wielu źródeł.

Złączenia jeden-do-jednego

Jest taki dowcip o człowieku, który skarżył się, że musi codziennie chodzić do szkoły 5 mil *pod górę w obie strony*. W pewnym sensie, złączenia jeden-do-jednego zamieniają ten obraz na sytuację przeciwną — ze względu na heurystyczne reguły wybierania następnej tabeli złączenia, złączenia jeden-do-jednego są *w obie strony z górki*! Jako takie, tego typu złączenia nie powodują żadnych problemów z optymalizacją i są najmniej kłopotliwymi elementami diagramów zapytania. Jednakże wskazują one czasami na pewne sprzyjające okoliczności do poprawienia projektu bazy danych, jeśli jesteśmy na etapie rozwoju aplikacji, w którym projekt bazy danych nie jest jeszcze zamrożony. Użytecznie jest także mieć standardowe sposoby na reprezentację złączeń jeden-do-jednego na diagramie. Opiszę więc sposoby przedstawiania takich przypadków.

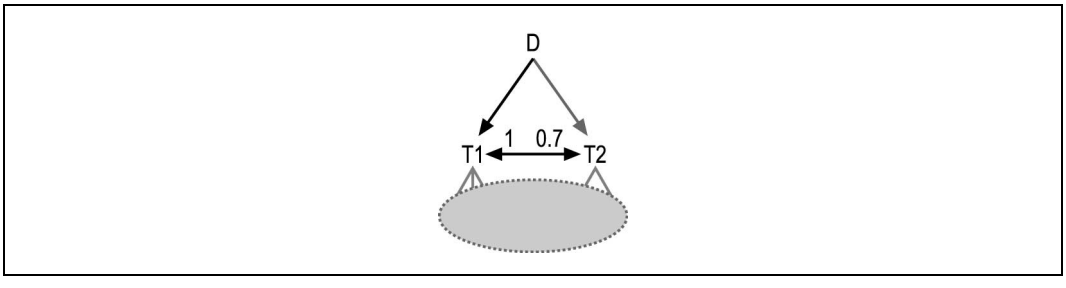
Złączenie jeden-do-jednego z tabelą zawierającą podzbiór danych

Rysunek 7.15 pokazuje typowe złączenie jeden-do-jednego osadzone w większym zapytaniu. Podczas gdy złączenie wiele-do-wielu posiadało współczynniki złączenia szczegółów na obu końcach, złączenie jeden-do-jedengo posiada współczynniki złączenia jedynie z tabelą nadrzędną. Współczynnik ten pokazuje nam, że złączenie pomiędzy `T1` a `T2` jest rzeczywiście typu jeden-do-zera lub jeden-do-jednego — złączenie zero-do-jednego zdarza się w 30% wierszy tabeli `T1`.



Rysunek 7.15. Typowe złączenie jeden-do-jednego

Ponieważ jest to złączenie wewnętrzne, przypadki jeden-do-zera pomiędzy `T1` a `T2` tworzą ukryty warunek złączenia, który powinien być obsłużony tak, jak opisano to pod koniec rozdziału 6. Należy także zwrócić uwagę, że może to być ukryty przypadek złączenia cyklicznego, co często się zdarza, gdy tabela główna łączy się jeden-do-jednego z inną tabelą. Jeśli tabela szczegółów znajduje się powyżej `T1`, jak wskazuje na to szare połączenie i jeśli ta tabela szczegółów łączy się z `T1` przez ten sam klucz unikalny, który został użyty przy złączeniu z `T2`, wówczas przez przechodniość tworzy się złączenie prowadzące od tabeli szczegółów do tabeli `T2`. Rysunek 7.16 pokazuje to implikowane złączenie poprzez połączenie zaznaczne szarym kolorem. O tym, jak radzić sobie z takimi przypadkami napisano we wcześniejszej części tego rozdziału poruszającej zagadnienie złączeń cyklicznych.



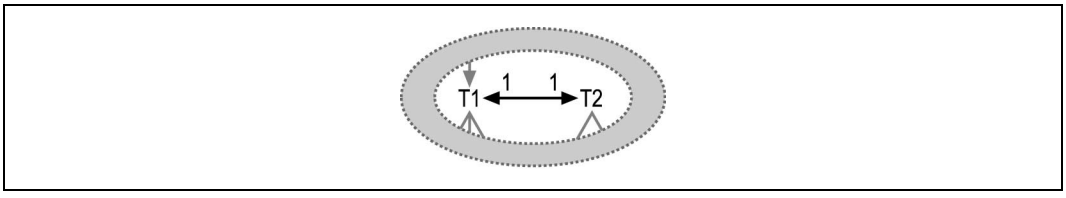
Rysunek 7.16. Złączenie implikowane tworzące złączenie cykliczne

Niezależnie od istnienia złączenia cyklicznego, może istnieć okazja do poprawienia projektu bazy danych. Przypadek z rysunku 7.16 sugeruje istnienie zbioru encji, które mapują się jeden-do-jednego z `T1` oraz podzbioru tych samych encji, które mapują się jeden-do-jednego z `T2`, gdzie `T2` jest zbudowane z kluczy głównych `T1` i kolumn mających zastosowanie jedynie dla tego podzbioru. W tym przypadku nie ma istotnych powodów, aby konieczne były dwie tabele — wystarczy po prostu dodać kolumnę do `T1` i pozostawić ją pustą dla wszystkich elementów większego zbioru, które nie należą do zbioru mniejszego! Sporadycznie zdarzają się sytuacje, w których ze względu na wygodę, można pozostawić w projekcie dwie tabele. Z perspektywy optymalizacji, połączenie tych dwóch tabel jest niemal zawsze pomocne, tak więc należy się choćby nad nim zastanowić, jeśli tylko mamy wpływ na projekt bazy danych.

Złączenia ściśle typu jeden-do-jednego

Rysunek 7.17 pokazuje szczególnie ważny przypadek na połączenie dwóch tabel w jedną. Współczynnik złączenia z tabelą główną jest tu dokładnie równy 1,0, a relacja pomiędzy tabelami jest ściśle typu jeden-do-jednego. W związku z tym, obie tabele mapują się na ten sam zbiór encji, a złączenie jest niepotrzebnym wydatkiem w porównaniu z wykorzystaniem tabeli połączonej.

Biorąc pod uwagę wydajność, jedynym powodem rozdzielenia tych tabel mogłaby być sytuacja, w której zapytanie niemal zawsze potrzebowałoby danych tylko z jednej spośród nich i bardzo rzadko wymagałoby wykonania złączenia. Najczęściej zdarza się, że

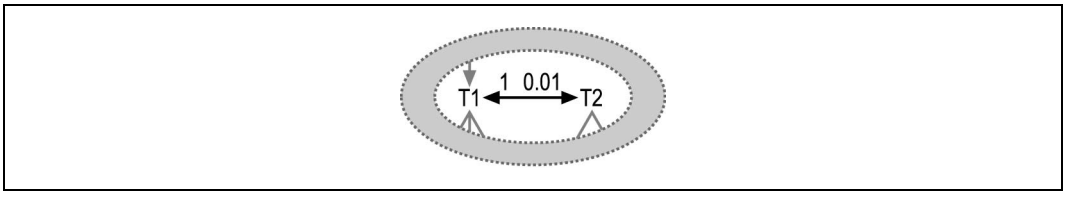


Rysunek 7.17. Złączenie dokładnie jeden-do-jednego

jedna z tabel zawiera dane potrzebne sporadycznie, w porównaniu z drugą tabelą. W tym przypadku — zwłaszcza gdy rzadko wykorzystywane dane zajmują dużo miejsca na każdy wiersz, a wierszy jest wiele — może się okazać, że większa zwartość częściej przeszukiwanej tabeli, w wyniku której jest ona lepiej buforowana, uzasadni koszt rzadko potrzebnego złączenia. Nawet z funkcjonalnego punktu widzenia lub perspektywy tworzenia oprogramowania jest całkiem prawdopodobne, że koszt kodowania jednoczesnego dodawania i usuwania wierszy z obu tabel (a czasem i aktualizowania) jest wysoki. Dlatego łatwiejsze może się okazać utrzymanie pojedynczej, połączonej tabeli. Zazwyczaj pojawiające się złączenia dokładnie typu jeden-do-jednego są rezultatem jakiejś dodanej funkcjonalności, która wymaga nowych kolumn w już istniejących encjach, a ograniczenia w projektowaniu lub po prostu czyjaś wizja nie pozwoliły zmienić oryginalnej tabeli. Jeśli to tylko możliwe, lepiej jest rozwiązać problem przez usunięcie tego typu ograniczeń.

Złączenie jeden-do-jednego ze znacznie mniejszym podzbiorem

Inny przypadek został pokazany na rysunku 7.18. Jest to złączenie jeden-do-zera lub jeden-do-jednego, które niemal zawsze jest typu jeden-do-zera. W takim przypadku argumenty za rozdzieleniem tabel są bardzo mocne. Mały zbiór encji reprezentowany przez `T2` może mieć zupełnie inne wymogi optymalizacyjne niż nadzbiór reprezentowany przez `T1`. Najprawdopodobniej tabela `T1` jest zazwyczaj przeszukiwana bez złączenia z `T2`. Wówczas bardzo istotny jest fakt, że nie zawiera ona niepotrzebnych kolumn z `T2`, a indeksy na niej założone mają zastosowanie przy zapytaniach do często używanych danych. Ukryty warunek złączenia, reprezentowany przez mały współczynnik złączenia z tabelą główną po stronie `T2`, jest bardzo dobry. W rzeczywistości jest on na tyle dobry, że można zdecydować się na wyjście od pełnego przeszukania tabeli `T2` i ponownie znaleźć najlepszą ścieżkę prowadzącą do pozostałych danych. Trudno byłoby powtórzyć taki plan wykonania bez tworzenia, zbędnych gdzie indziej, indeksów, gdybyśmy mieli połączyć te dwie tabele w jedną.

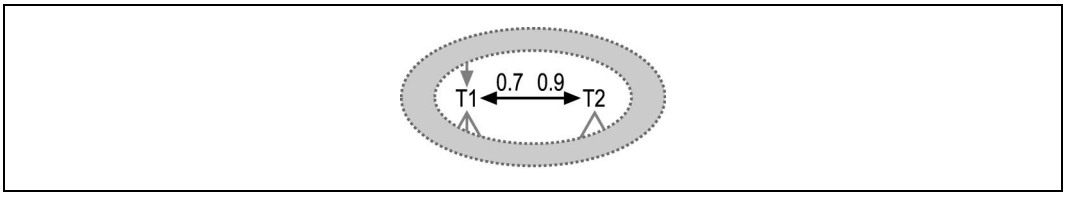


Rysunek 7.18. Złączenie jeden-do-zera lub jeden-do-jednego pomiędzy tabelami o bardzo dużej różnicy rozmiarów

W tym przypadku najważniejsze jest, aby wziąć pod uwagę ukryty warunek złączenia z tabeli `T1` do `T2`, niezależnie od tego, czy wychodzimy ze strony zapytania, po której znajduje się `T2`, czy też osiągniemy ją jak najwcześniej, aby uzyskać dostęp do ukrytego filtru.

Złączenie jeden-do-jednego z ukrytymi filtrami złączenia po obu stronach

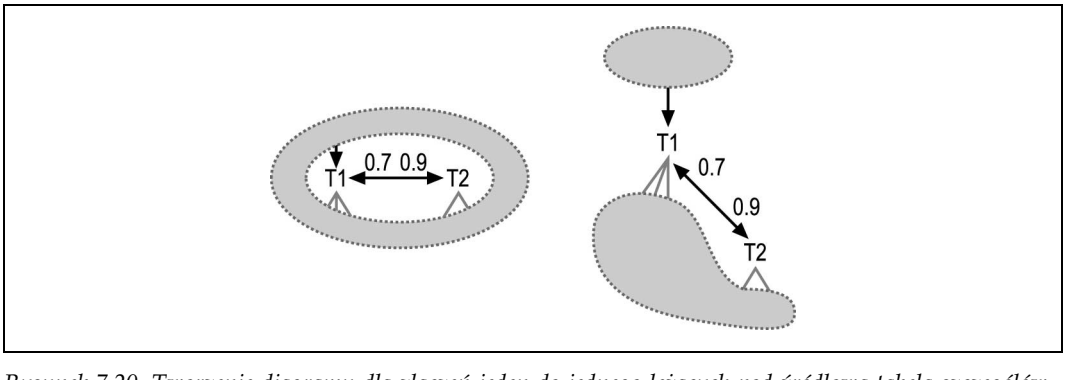
Rysunek 7.19 pokazuje wyjątkowy przykład złączenia [zero lub jeden]-do-[zero lub jeden], w którym filtr ma zastosowanie w obu kierunkach. Jeśli złączenia jeden-do-jednego są skierowane w dół w obu kierunkach, wówczas złączenia [zero lub jeden]-do-[zero lub jeden] opadają bardzo stromo. Jednakże jeśli tylko dane nie są uszkodzone (w jednej tabeli brakuje danych), taki przypadek wskazuje na istnienie trzeciej tabeli lub na to, że powinna istnieć, zawierając nadzbiór tych dwóch zachodzących na siebie zbiorów. Jeśli znajdziemy lub utworzymy taką tabelę, te same argumenty, które przytoczono wcześniej mają zastosowanie w łączeniu jej z jedną lub obiema tabelami zawierającymi podzbiór danych.



Rysunek 7.19. Złączenie [zero lub jeden]-do-[zero lub jeden]

Konwencje pokazywania złączeń jeden-do-jednego

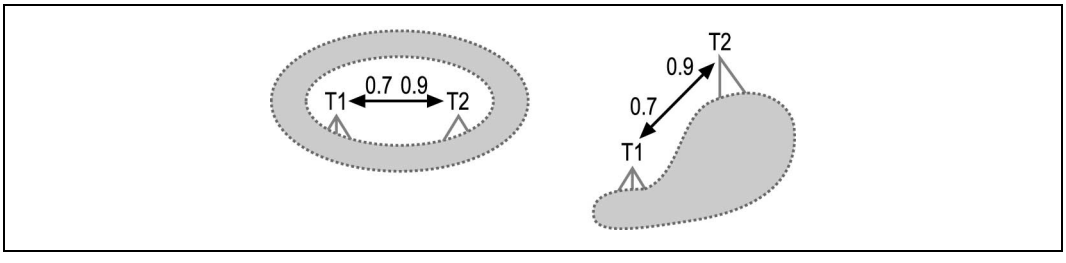
Bardzo pomocne jest ustalenie konwencji odnośnie pokazywania diagramów zapytań. Pomagają one prezentować ważne informacje w uniwersalny sposób. Połączenia z pojedynczymi strzałkami zawsze wskazują ku dołowi. Rysunek 7.20 pokazuje dwie możliwości, które dobrze funkcjonują w złączeniach jeden-do-jednego leżących poniżej źródłowej tabeli szczegółów. Pierwsza możliwość wskazuje na połączenie dwustrzałkowe, gdzie żaden z węzłów nie leży powyżej drugiego. Druga używa standardowego przepływu ku dołowi ze źródłowej tabeli szczegółów. Obie metody okażą się dobre, jeśli tylko będziemy pamiętać, że oba kierunki złączenia jeden-do-jednego są w istocie *skierowane ku dołowi*.



Rysunek 7.20. Tworzenie diagramu dla złączeń jeden-do-jednego leżących pod źródłową tabelą szczegółów

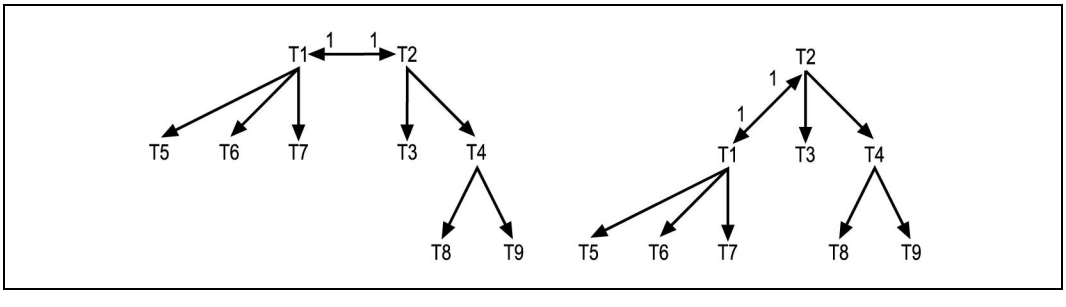
W razie gdy obie złączone tabele znajdują się poniżej źródła, należy pamiętać o tym, że jeśli tabele złączone jeden-do-jednego współdzielą klucz główny, wówczas połączenie od góry do `T1` może przez przechodniość równie dobrze prowadzić do `T2`, chyba że prowadzi do jakiegoś innego klucza unikalnego występującego w `T1`, a nieistniejącego w `T2`. Tworzy się tu implikowane złączenie cykliczne przedstawione na rysunku 7.2 B.

Rysunek 7.21 przedstawia inne możliwe diagramy dla złączeń jeden-do-jednego tabel, które można zakwalifikować jako główne tabele szczegółów (nie mają złączeń od góry), jeśli chociaż jeden z kierunków złączenia jeden-do-jednego ma współczynnik złączenia z tabelą nadrzędną mniejszy niż 1,0. Ponownie można podkreślić złączenie jeden-do-jednego poprzez wykorzystanie poziomego układu lub też podkreślić, która tabela jest większa (i który kierunek złączenia jest bardziej stromy) poprzez umieszczenie wyżej węzła z większym współczynnikiem złączenia z tabelą nadrzędną. Węzeł z większym współczynnikiem złączenia z tabelą nadrzędną reprezentuje tabelę zawierającą więcej wierszy w tym [zero lub jeden]-do-[zero lub jeden] złączeniu.



Rysunek 7.21. Różne możliwe diagramy dla złączenia [zero lub jeden]-do-[zero lub jeden] głównych tabeli szczegółów

Rysunek 7.22 pokazuje przypadek podobny do tego z rysunku 7.21, ale z węzłami będącymi dokładnie w relacji jeden-do-jednego (tabel, które zawsze się łączą). Ponownie możemy wzmocnić równość obu kierunków złączenia poprzez ułożenie węzłów w poziomie. Ewentualnie, możemy wybrać kierunek, który pozwala na stworzenie lepiej zbalansowanego drzewa, tj. takiego, które lepiej mieści się na stronie, umieszczając węzły z bardziej rozbudowanymi gałęziami wyżej. To, które rozwiązanie wybierzemy, ma mniejsze znaczenie, jeśli tylko pamiętamy, że oba kierunki są w zasadzie skierowane ku dołowi, niezależnie od tego jak są przedstawione na diagramie.



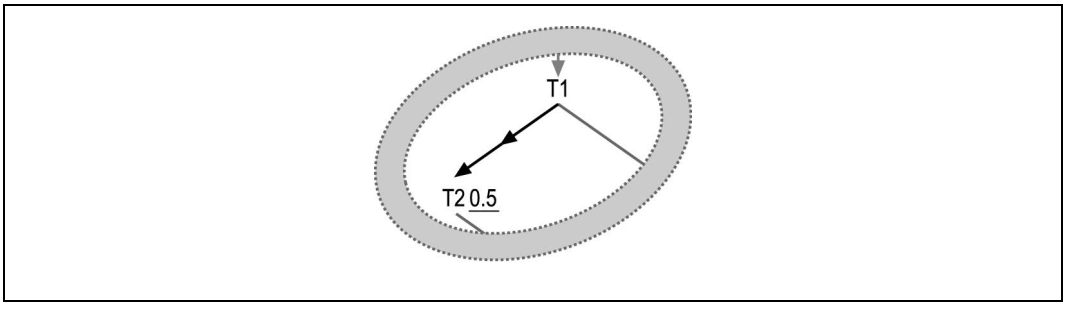
Rysunek 7.22. Różne możliwości przedstawienia diagramu dla głównych tabel szczegółów złączonych dokładnie jeden-do-jednego

Złączenia zewnętrzne

Niemal zawsze sensem i celem złączenia zewnętrznego jest zapobieżenie utracie pożądanych danych z tabeli, z której wychodzi złączenie. *Nieprawidłowe* złączenia zewnętrzne, które opiszę w następnych podrozdziałach, ogólnie rzecz biorąc, pokazują pewne sprzeczności z podanymi powyżej przyczynami istnienia złączeń zewnętrznych.

Filtrowane złączenia zewnętrzne

Przyjrzyjmy się rysunkowi 7.23, na którym zobrazowano złączenie zewnętrzne z tabelą zawierającą warunek filtrowania. W przypadku gdy tabela `T1` ma wiersz nie znajdujący odpowiednika w tabeli `T2`, baza danych przypisuje w rezultacie wartość `null` do każdej kolumny `T2`. Dlatego — za wyjątkiem `T2.JakaśKolumna IS NULL` — praktycznie każdy warunek filtrowania na `T2` wyłączyłby wynikowy wiersz pochodzący z zewnętrznej części złączenia zewnętrznego.



Rysunek 7.23. Złączenie zewnętrzne z filtrowanym węzłem

Nawet warunki takie, jak `T2.Unpaid_Flag != 'T'` lub `NOT T2.Unpaid_Flag = 'Y'`, które powinny być prawdziwe, w zewnętrznym przypadku nie są.

W przypadku warunków w klauzuli `WHERE`, baza danych interpretuje wartości `null` w mało intuicyjny sposób. Jeśli patrzymy na `null` jako na reprezentację wartości „niewiadoma" w odniesieniu do kolumny tabeli, a nie w bardziej standardowym znaczeniu — „nie ma zastosowania", możemy zacząć rozumieć jak baza danych traktuje wartości `null` w klauzuli `WHERE`. Za wyjątkiem pytania konkretnie o to, czy kolumna ma wartość `null`, niemal każde pytanie jakie można zadać, zwróci rezultat „nieznany", co w istocie jest *prawdziwą wartością* większości warunków z wartościami `null`. W razie odrzucenia wierszy zapytania, baza danych traktuje prawdziwą wartość „nieznany" jak `FALSE`, nie przyjmując wierszy z nieznanymi prawdziwymi wartościami w klauzuli `WHERE`. I o ile `NOT FALSE = TRUE`, okazuje się, że `NOT "nieznane" = "nieznane"`!

Ponieważ większość filtrów na tabeli zewnętrznej odrzuca zewnętrzną część złączenia zewnętrznego i ponieważ głównym celem złączenia zewnętrznego jest zachowanie tej części, trzeba szczególnie uważać na każdy filtr na tabeli zewnętrznej. Jeden z poniższych scenariuszy ma zazwyczaj zastosowanie i dobrze jest poświęcić chwilę, aby ustalić który:

- Filtr jest jednym z rzadko występujących filtrów, jak na przykład `JakaśKolumna IS NULL`, który może zwrócić wartość `TRUE` dla wartości `null` występujących w części zewnętrznej, a sam filtr jest funkcjonalnie poprawny.
- Programista nie miał zamiaru pozbyć się części zewnętrznej, więc warunek filtrowania musi zostać usunięty.
- Warunek filtrowania miał w zamierzeniach odrzucić część zewnętrzną, a złączenie mogłoby być równie dobrze wewnętrzne. W takim przypadku nie ma funkcjonalnych różnic pomiędzy zapytaniem ze złączeniem wyrażonym w formie zewnętrznej czy formie wewnętrznej. Jednak poprzez jawne zaznaczenie go jako złączenia wewnętrznego, baza danych ma większą swobodę w tworzeniu planu wykonania i złączenie to może być wykonane w obie strony. Jeśli najlepszy filtr znajduje się po tej samej stronie złączenia co poprzednio złączona zewnętrznie tabela, dodatkowy stopień swobody może udostępnić lepszy plan wykonania. Niemniej zamiana złączenia na wewnętrzne może spowodować, że optymalizator popełni błąd, którego mógłby uniknąć w przypadku złączenia zewnętrznego. Złączenia zewnętrzne są jedną z metod na wymuszenie porządku złączenia, jeśli tylko świadomie tego chcemy, nawet gdy nie potrzebujemy uchronić zewnętrznej części złączenia.

Warunki złączenia zewnętrznego na pojedynczej tabeli

W starej notacji bazy danych Oracle, warunek filtrowania czynimy częścią złączenia przez dodanie (+). Na przykład, dwuczęściowe złączenie zewnętrzne z tabelą `Code_Translations`, użytą we wcześniejszych przykładach, mogłoby wyglądać następująco:

```
WHERE ...
   AND O.Order_Type=OType_Trans.Code (+)
   AND OTypeTrans.Type(+)='TYP_ZAMOWIENIA'
```

Według nowszej notacji ANSI, która jest jedyną dozwoloną w DB2, warunek filtrowania przenosi się z klauzuli `FROM` i staje się jawnym warunkiem złączenia:

```
FROM ... Orders O ...
LEFT OUTER JOIN Code_Translations OTypeTrans
                           ON O.Order_Type_Code=OTypeTrans.Code
                       AND OTypeTrans.Code_Type='TYP_ZAMOWIENIA'
```

W oryginalnej notacji złączenia zewnętrznego, pochodzącej z bazy danych SQL Server, baza danych po prostu zakłada, że warunek filtrowania jest częścią złączenia:

```
WHERE ...
      AND O.Order_Type_Code*=OTypeTrans.Code
      AND OTypeTrans.Type='TYP_ZAMOWIENIA'
```

Zauważmy, że opisany problem przestaje istnieć dla złączeń zewnętrznych starego SQL Servera. Baza danych automatycznie uczyniła filtr częścią zapytania. Zwróćmy także uwagę, że w rzadkich przypadkach, w których warunek filtrowania jest naprawdę filtrem, musimy bądź stosować nową notację złączeń zewnętrznych, aby otrzymać pożądane rezultaty, bądź też zamienić złączenie w równoważne podzapytanie `NOT EXISTS`, tak jak to zaraz opiszę.

- Warunek filtrowania był zamierzony, ale powinien być częścią złączenia! Dzięki uczynieniu warunku filtrowania częścią złączenia, nakazujemy bazie danych co następuje: „Dla każdego wiersza tabeli szczegółów dostarcz odpowiadający mu wiersz z tabeli, do której pasuje filtr, jeśli w ogóle taki istnieje. W przeciwnym wypadku połącz z pseudowierszem o wszystkich wartościach `null`".

Rozważmy dogłębniej pierwszy scenariusz. Spójrzmy na zapytanie:

```
SELECT ...
FROM Employees E
             LEFT OUTER JOIN Departments D
                          ON E.Department_ID=D.Department_ID
WHERE E.Manager_ID IS NULL
```

O co tak naprawdę zapytanie pyta bazę danych? Semantycznie jest to żądanie o dwa, raczej różne, zbiory wierszy — zbiór wszystkich pracowników, którzy nie mają żadnych oddziałów i zbiór wszystkich pracowników, których menadżerowie nie mają departamentów. Możliwe jest, że aplikacja rzeczywiście potrzebuje dwóch takich niezależnych zbiorów jednocześnie, aczkolwiek wydaje się bardziej prawdopodobnym, że programista nie zauważył, że tak proste zapytanie zwraca tak złożone rezultaty, i wcale nie potrzebował jednego z tych zbiorów.

Rozważmy nieco inny przykład:

```
SELECT ... FROM Employees E
         LEFT OUTER JOIN Departments D
                      ON E.Department_ID=D.Department_ID
WHERE D.Department_ID IS NULL
```

Na pierwszy rzut oka zapytanie to może się wydawać dziwne, ponieważ klucz główny `(Department_ID)` tabeli `Department` nie może być równy `null`. Pomimo tego, taki klucz wartości `null` nie mógłby nigdy złączyć się z żadną inną tabelą złączeniem takim, jak zaprezentowane (ponieważ warunek `NULL = NULL` zwraca wartość „nieznany"). Jednakże, ponieważ jest to złączenie zewnętrzne, istnieje sensowna interpretacja tego zapytania: „Znajdź pracowników, którzy nie mają przypisanych departamentów". W zewnętrznej części tego zewnętrznego zapytania, każda kolumna tabeli `Departments`, włączając nawet obligatoryjne różne od `null` kolumny, jest zamieniana na `null`. Dlatego warunek `D.Department_ID IS NULL` jest prawdziwy jedynie w zewnętrznym przypadku. Istnieje jednak znacznie popularniejszy i łatwiejszy do odczytania sposób wyrażenia tego zapytania:

```
SELECT ...
FROM Employees E
WHERE NOT EXISTS (SELECT *
                    FROM Departments D
                   WHERE E.Department_ID=D.Department_ID)
```

Chociaż forma `NOT EXISTS` w tego typu zapytaniu jest bardziej naturalna, łatwiejsza do odczytania i zrozumienia, forma pierwsza (najlepiej jeśli jest otoczona przez odpowiednie komentarze) ma swoje miejsce w optymalizacji SQL. Przewaga wyrażenia `NOT EXISTS` jako złączenia zewnętrznego, po którym występuje wyrażenie `Klucz_Główny IS NULL`, polega na tym, że daje ono większą kontrolę w momencie wystąpienia złączenia

w planie wykonania oraz kiedy selektywność warunku zostanie uwzględniona. Zazwyczaj warunki `NOT EXISTS` są wykonywane po wszystkich zwykłych złączeniach, w każdym razie dzieje się tak w bazie danych Oracle. To przykład, w którym filtr (nie będący częścią złączenia zewnętrznego) na złączonej zewnętrznie tabeli jest umieszczony celowo i poprawnie.

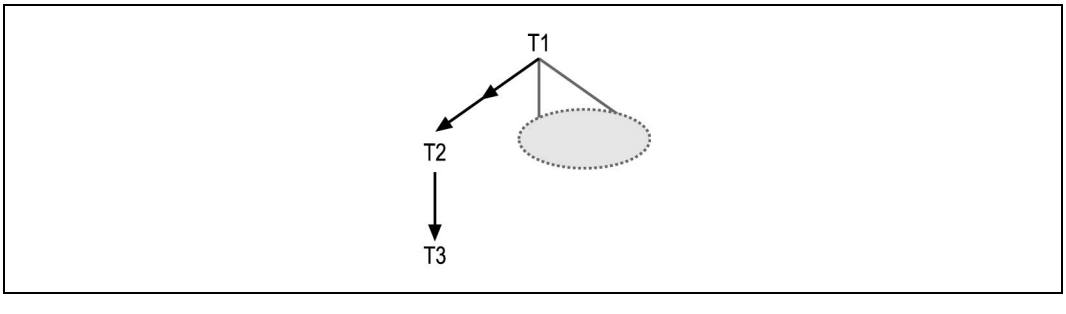
W notacji zgodnej ze starą wersją SQL Servera, kombinacja złączenia zewnętrznego i warunku `IS NULL` nie działała. Przepisanie przykładu w notacji zgodnej z SQL Serverem mogłoby wyglądać w taki sposób:

```
SELECT ...
FROM Employees E, Departments D
WHERE E.Department_ID*=D.Department_ID
  AND D.Department_ID IS NULL
```

Jednakże rezultat nie będzie taki, jakiego byśmy sobie życzyli! Przypomnijmy, że SQL Server interpretuje wszystkie warunki filtrowania na tabeli złączonej zewnętrznie jako część złączenia i będzie próbował utworzyć połączenie z tabelą `Departments`, która ma wartości `null` klucza głównego (wartości `null` w kolumnie `D.Department_ID`). Nawet jeśli takie wiersze istnieją z naruszeniem poprawnego projektu bazy danych, nigdy nie złączą się poprawnie z tablicą `Employees`, ponieważ warunek równości nie może być prawdziwy dla wartości `null` klucza. Zamiast tego zapytanie nie przefiltruje żadnych wierszy, zwracając wszystkich pracowników z wszystkimi złączeniami wpadającymi w przypadek zewnętrzny.

Złączenia zewnętrzne prowadzące do złączeń wewnętrznych

Rozważmy rysunek 7.24, w którym złączenie zewnętrzne prowadzi do złączenia wewnętrznego.



Rysunek 7.24. Złączenie zewnętrzne prowadzące do złączenia wewnętrznego

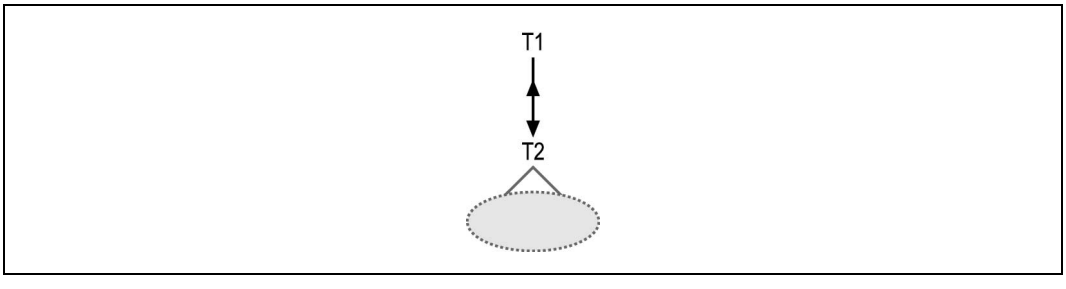
Powyższe złączenie w starej wersji bazy danych Oracle miałoby następujący zapis:

```
SELECT ...
FROM Table1 T1, Table3 T2, Table3 T3
WHERE T1.FKey2=T2.PKey2(+)
  AND T2.FKey3=T3.PKey3
```

W zewnętrznym przypadku pierwszego złączenia, baza danych wygeneruje pseudowiersz tabeli `T2`, wypełniony samymi wartościami `null` dla wszystkich kolumn, włączając w to wartość `T2.FKey3`. Jednakże klucz obcy o wartości `null` nigdy nie złączy się z inną tabelą, tak więc wiersze reprezentujące przypadek zewnętrzny będą odrzucone podczas próby złączenia z `T3`. Dlatego wynik złączenia zewnętrznego prowadzącego do złączenia wewnętrznego jest dokładnie taki sam, jak rezultat jaki otrzymalibyśmy, gdyby oba złączenia były wewnętrzne. Jego otrzymanie jest bardziej kosztowne, ponieważ baza danych odrzuca wiersze, które nie połączyły się w dalszej części planu wykonania. Taka sytuacja zawsze świadczy o błędzie. Jeśli zamiarem jest utrzymanie zewnętrznego przypadku, należy wymienić złączenie zewnętrzne prowadzące do złączenia wewnętrznego przez złączenie zewnętrzne prowadzące do innego złączenia zewnętrznego. W przeciwnym razie należy użyć złączenia wewnętrznego prowadzącego do złączenia wewnętrznego.

Złączenie zewnętrzne wskazujące na tabele szczegółów

Przeanalizujmy rysunek 7.25, w którym strzałka pośrednia pokazuje złączenie zewnętrzne wskazujące na tabelę szczegółów.



Rysunek 7.25. Złączenie zewnętrzne wskazujące na tabelę szczegółów

Gdyby wykorzystać nową notację ANSI, zapytanie mogłoby wyglądać następująco:

```
SELECT ...
FROM Departments D
     LEFT OUTER JOIN Employees E
                  ON D.Department_ID=E.Department_ID
```

Zapis w starej wersji bazy danych Oracle przedstawiałby się w ten sposób:

```
SELECT ...
FROM Departments D, Employees E
WHERE D.Department_ID=E.Department_ID(+)
```

zaś w starej wersji SQL Servera, w taki:

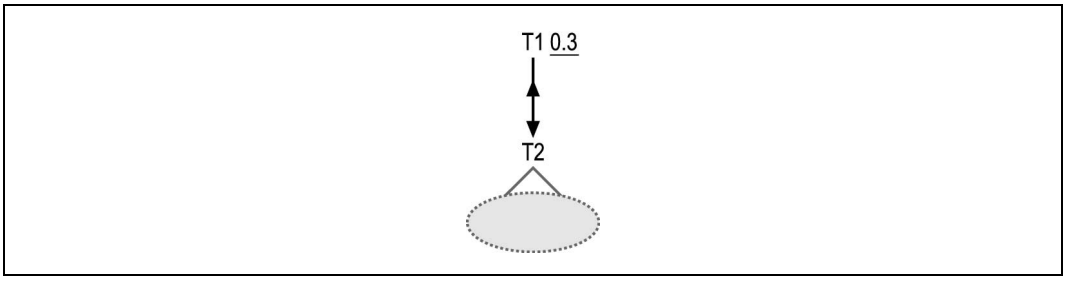
```
SELECT ...
FROM Departments D, Employees E
WHERE D.Department_ID*=P.Department_ID
```

O co pyta każde z tych zapytań? Istotę pytania można by sformułować w mniej więcej taki sposób: „Podaj mi informacje dotyczące wszystkich pracowników, którzy mają departamenty (przypadek wewnętrzny) wraz z danymi ich departamentów, a także informacje

dotyczące departamentów, które nie posiadają pracowników (przypadek zewnętrzny)". W przypadku wewnętrznym, rezultat mapuje każdy wiersz na encję detali (pracownik przynależący do departamentu), podczas gdy w przypadku zewnętrznym, rezultat mapuje każdy wiersz na encję źródłową (departament, który nie przynależy do żadnego pracownika). Jest mało prawdopodobne, żeby taki dziwny splot encji był użyteczny jako rezultat jednego zapytania, stąd zapytania takie jak te, ze złączeniami zewnętrznymi do tabeli szczegółów, rzadko są poprawne. Najbardziej popularny przypadek takiego błędu to złączenie z tabelą szczegółów, które zazwyczaj nie zwróci żadnego lub jeden szczegół na wiersz źródłowy i rzadko jest to złączenie wiele-do-jednego z tabelą źródłową. Programiści czasem nie zauważają implikacji związanych z rzadkimi przypadkami wiele-do-jednego, a mogą się one nie ujawnić w czasie testowania.

Złączenia zewnętrzne z tabelami szczegółów posiadającymi filtry

Rysunek 7.26 pokazuje złączenie zewnętrzne z tabelą szczegółów, która dodatkowo ma warunek filtrowania. I znowu podwójne „zło" daje w wyniku „dobro". Złączenie zewnętrzne z tabelą szczegółów, która posiada dodatkowo filtr może przysporzyć wszystkich problemów opisanych w poprzednich dwóch podrozdziałach. Czasem filtr znosi efekt problematycznego złączenia zewnętrznego, zamieniając je funkcjonalnie w złączenie wewnętrzne. W takich przypadkach trzeba unikać usunięcia filtra, chyba że uczynimy jednocześnie złączenie zewnętrzne — wewnętrznym.



Rysunek 7.26. Złączenie zewnętrzne do filtrowanej tabeli szczegółów

Najbardziej interesującym przypadkiem, widocznym na rysunku 7.26, jest sytuacja, w której istnienie filtru ma sens jedynie w kontekście złączenia zewnętrznego. Wówczas warunek filtrowania na `T1` jest prawdziwy jedynie w przypadku zewnętrznym, na przykład `T1.FKey_ID IS NULL`. (`T1.FKey_ID` jest kluczem obcym wskazującym na `T2.PKey_ID` na diagramie złączenia). Podobnie jak w poprzednim przykładzie z warunkiem wartość-klucza-łączącego `IS NULL` (klucz główny we wcześniejszym przypadku), przypadek ten jest równoważny podzapytaniu `NOT EXISTS`. To alternatywne wyrażenie dla warunku `NOT EXISTS` oferuje czasami użyteczny poziom dodatkowej kontroli w momencie, kiedy baza danych wykona złączenie i odrzuci wiersze, które nie spełniają warunku. Ponieważ wszystkie złączone wewnętrznie wiersze są odrzucone przez warunek `IS NULL`, pozbywamy się standardowego problemu związanego ze złączeniem zewnętrznym z tabelą szczegółów — zmieszania ze sobą różnych encji pochodzących z wewnętrznej i zewnętrznej części złączenia. I w taki oto sposób dwukrotne „zło" daje w efekcie „dobro"!

Zapytania z podzapytaniami

Niemal wszystkie prawdziwe zapytania z podzapytaniami nakładają specjalny rodzaj warunku na wiersze w zewnętrznym, głównym zapytaniu — muszą pasować albo też nie do odpowiednich wierszy powiązanego zapytania. Na przykład, jeśli potrzebujemy danych odnośnie departamentów, które mają pracowników, zapytanie mogłoby wyglądać następująco:

```
SELECT ...
FROM Departments D
WHERE EXISTS (SELECT NULL
              FROM Employees E
              WHERE E.Department_ID=D.Department_ID)
```

Ewentualnie moglibyśmy zapytać o departamenty, które nie posiadają pracowników:

```
SELECT ... FROM Departments D
WHERE NOT EXISTS (SELECT NULL
                  FROM Employees E
                  WHERE E.Department_ID=D.Department_ID)
```

Złączenie `E.Department_ID=D.Department_ID` w każdym z tych zapytań jest złączeniem skorelowanym, które dopasowuje do siebie tabele zapytania zewnętrznego i podzapytania. Zapytanie `EXISTS` ma alternatywną, równoważną postać:

```
SELECT ...
FROM Departments D
WHERE D.Department_ID IN (SELECT E.Department_ID FROM Employees E)
```

Ponieważ obie formy są funkcjonalnie równoważne i ponieważ diagram nie powinien sugerować jednego rozwiązania, jest on dla obu form taki sam. Dopiero po jego utworzeniu należy wybrać odpowiednią formę w zależności od tego, która z nich najlepiej rozwiązuje problem wydajności i w najlepszy sposób przedstawia oczekiwaną ścieżkę dostępu do danych.

Tworzenie diagramów zapytań z podzapytaniami

Ignorując złączenie pomiędzy zapytaniem zewnętrznym a podzapytaniem, można utworzyć niezależne diagramy zapytań dla każdego z nich osobno. Pozostaje jedynie pytanie, jak przedstawić połączenie tych dwóch diagramów, tak by utworzyły jeden. Na podstawie formy `EXISTS` poprzedniego zapytania jasno widać, że zapytanie zewnętrzne łączy się z podzapytaniem poprzez złączenie skorelowane. Takie złączenie ma szczególną właściwość: dla każdego wiersza zapytania zewnętrznego, już po pierwszym razie, kiedy baza danych odnajdzie pasujące do siebie (według złączenia) wiersze, zaprzestaje dalszych poszukiwań, uważając warunek `EXISTS` za spełniony i podaje wiersz zapytania zewnętrznego do następnego kroku planu wykonania. Jeśli znajdzie pasujące wiersze dla podzapytania powiązanego przez `NOT EXISTS`, zatrzymuje się, ze względu na niespełnienie tego warunku, a następnie odrzuca odpowiedni wiersz zapytania zewnętrznego.

Takie zachowania sugerują, że diagram zapytania powinien odpowiadać na następujące cztery pytania odnośnie złączeń skorelowanych, które nie mają zastosowania w przypadku zwykłych złączeń:

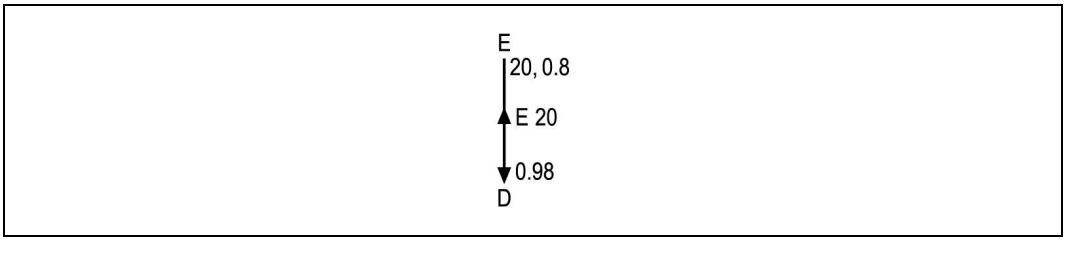
- Czy złączenie jest zwykłe? (Nie, złączenie jest skorelowane z podzapytaniem.)
- Która strona złączenia jest podzapytaniem, a która zapytaniem zewnętrznym?
- Przez którą formę zapytania da się wyrazić, `EXISTS` czy `NOT EXISTS`?
- Jak wcześnie w planie wykonania podzapytanie powinno być wykonane?

Pracując z podzapytaniami i ustalając odpowiedzi na te pytania, trzeba pamiętać, że tak samo jak dla zwykłych zapytań, musimy pokazać, który koniec jest tabelą główną i jak duże są współczynniki złączeń.

Tworzenie diagramów dla podzapytań EXISTS

Rysunek 7.27 pokazuje moją konwencję tworzenia diagramów z podzapytaniami typu `EXISTS` (które mogą być wyrażone przez równoważne podzapytanie `IN`). Rysunek oparty jest o wcześniejsze podzapytanie typu `EXISTS`:

```
SELECT ... FROM Departments D
WHERE NOT EXISTS (SELECT NULL
                  FROM Employees E
                  WHERE E.Department_ID=D.Department_ID)
```



Rysunek 7.27. Proste zapytanie z podzapytaniem

Dla *złączeń skorelowanych* (znanych jako złączenia typu *semi-join*, jeśli odnoszą się do podzapytań typu `EXISTS`) z tabeli `Departments` do `Employees`, diagram rozpoczyna się z takimi samymi statystykami złączenia, jak pokazano na rysunku 5.1.

Jak każde inne złączenie, złączenie typu semi-join, które łączy zapytanie wewnętrzne z zewnętrznym jest przedstawione w postaci strzałki na jednym z końców połączenia, wskazującej na klucz główny. Podobnie na każdym końcu znajdują się współczynniki złączenia, które reprezentują te same właściwości, które złączenie to miałoby w zwykłym zapytaniu. Używam strzałki pośredniej, aby wskazać przejście od węzła zewnętrznego zapytania skorelowanego do węzła podzapytania. Umieściłem *E* obok strzałki pośredniej, aby pokazać, że jest to złączenie typu semi-join dla elementu `EXISTS` lub `IN` podzapytania.

W tym przypadku, tak jak w wielu przypadkach podzapytań, część diagramu z podzapytaniem jest pojedynczym węzłem reprezentującym podzapytanie bez jego własnych złączeń. Rzadziej, jednak podobnie jak w tym przykładzie, zapytanie zewnętrzne jest pojedynczym węzłem, reprezentującym to zapytanie bez złączeń własnych. Składnia jest zasadniczo nieograniczona, z możliwością wystąpienia wielu podzapytań złączonych z zapytaniem zewnętrznym lub też podzapytań ze złożoną wewnętrzną strukturą złączeń lub nawet podzapytań wskazujących na głębiej zagnieżdżone w nich samych podzapytania.

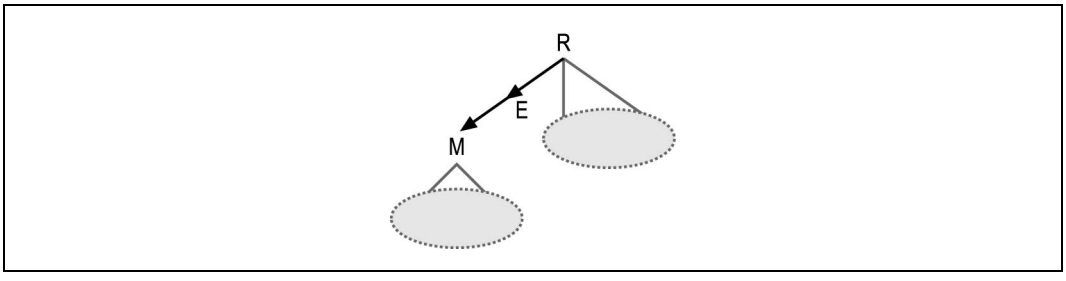
Złączenie typu semi-join wymaga także do dwóch liczb, aby pokazać właściwości podzapytania, dzięki czemu można wybrać najlepszy plan. Rysunek 7.27 pokazuje obie te wartości, które czasem są niezbędne do wybrania optymalnego planu radzenia sobie z podzapytaniem.

Pierwsza wartość, tuż obok *E* (równa 20 na rysunku 7.27) jest *współczynnikiem korelacji*. Współczynnik korelacji jest współczynnikiem *I*/*E*. *E* jest szacunkowym lub zmierzonym czasem wykonania najlepszego planu prowadzącego z zapytania zewnętrznego do podzapytania (podążając za logiką zapytania `EXISTS`). *I* jest szacunkowym lub zmierzonym czasem wykonania najlepszego planu prowadzącego z zapytania wewnętrznego do zapytania zewnętrznego (podążając za logiką zapytania typu `IN`). Możemy zawsze wyznaczyć ten współczynnik bezpośrednio poprzez zmierzenie czasu wykonania obu form, co zazwyczaj jest dość łatwą czynnością, chyba że mamy do czynienia z dużą liczbą połączonych podzapytań. W dalszej części książki opiszę kilka reguł pomocnych w oszacowaniu wartości *I*/*E* mniej lub bardziej celnie, ale nawet przybliżona estymacja jest wystarczająca do wyboru planu, kiedy — jak ma to często miejsce — wartość jest bądź znacznie mniejsza, bądź znacznie większa niż 1,0. Kiedy współczynnik korelacji jest większy niż 1,0, należy wybrać podzapytanie skorelowane z warunkiem `EXISTS` oraz plan, który wychodzi od zapytania zewnętrznego do podzapytania.

Następną nową wartością jest *skorygowany współczynnik filtrowania podzapytania* (równy 0,8 na rysunku 7.27) zaraz obok współczynnika złączenia detali. Jest to szacunkowa wartość, która jest pomocna w wyborze najlepszego miejsca w porządku złączenia do sprawdzenia warunku podzapytania. Ma ona jedynie zastosowanie do zapytań, które zaczynają się od zapytania zewnętrznego, tak więc należy ją wykluczyć z każdego złączenia typu semi-join (ze współczynnikiem korelacji mniejszym niż 1,0), które może być zamienione na źródłowe zapytanie w planie wykonania.

Jeśli mamy więcej niż jedno złączenie typu semi-join ze współczynnikiem korelacji mniejszym niż 1,0, powinniśmy wyjść od podzapytania o najmniejszym współczynniku korelacji, ale wciąż będą nam potrzebne skorygowane współczynniki filtrowania pozostałych podzapytań.

Zanim wyjaśnię, jak obliczyć współczynnik korelacji i skorygowany współczynnik korelacji podzapytania, zastanówmy się, kiedy będziemy ich potrzebować. Rysunek 7.28 pokazuje część diagramu zapytania dla podzapytania typu `EXISTS` ze źródłową tabelą szczegółów na końcu złączenia typu semi-join odpowiadającemu kluczowi głównemu.



Rysunek 7.28. Złączenie semi-join z kluczem głównym

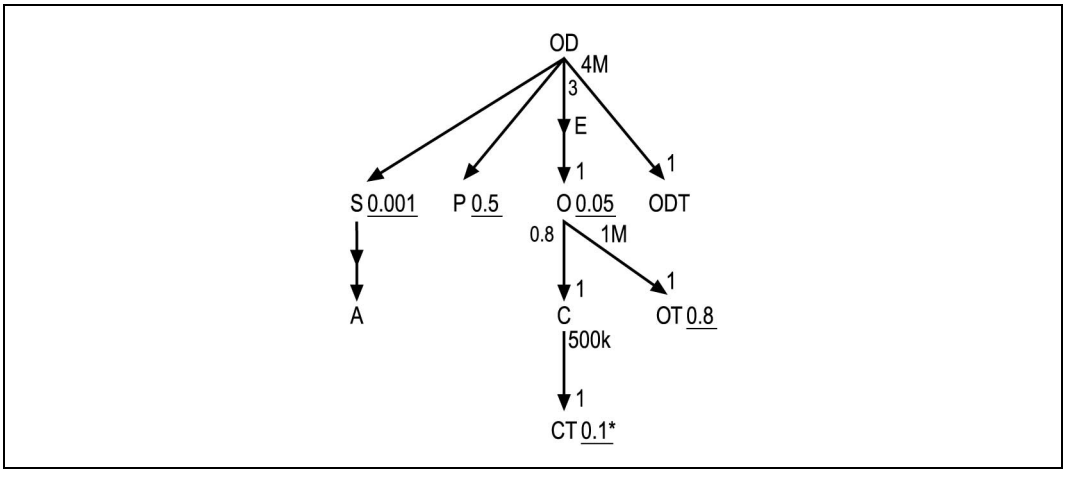
Złączenie semi-join na rysunku nie różni się funkcjonalnie od zwykłego złączenia, ponieważ zapytanie nigdy nie znajdzie więcej niż jednego pasującego wiersza z tabeli `M` dla dowolnego wiersza zapytania zewnętrznego.

Zakładam tutaj, że całe poddrzewo poniżej `M` ma *formę normalną* (ze wszystkimi połączeniami wskazującymi ku dołowi na klucze główne), tak że całe podzapytanie mapuje się jeden-do-jednego z wierszami ze źródłowej tabeli szczegółów `M` tego poddrzewa.

Ponieważ złączenie typu semi-join nie różni się funkcjonalnie od zwykłego złączenia, możemy uzyskać dodatkowy stopień swobody w planie wykonania poprzez jawne wyeliminowania warunku `EXISTS` oraz włączenie podzapytania do zapytania zewnętrznego. Rozważmy następujące zapytanie:

```
SELECT <Kolumny jedynie z zapytania zewnętrznego>
FROM Order_Details OD, Products P, Shipments  S,
     Addresses A, Code_Translations CT
WHERE OD.Product_ID = P.Product_ID
  AND P.Unit_Cost > 100
  AND OD.Shipment_ID = S.Shipment_ID
  AND S.Address_ID = A.Address_ID(+)
  AND OD.Status_Code = CT.Code
  AND CT.Code_Type = 'Dokladny_Status_Zamowienia'
  AND S.Shipment_Date > :now - 1
  AND EXISTS (SELECT null
              FROM Orders O, Customers C, Code_Translations OT,
                   Customer_Types CT
              WHERE C.Customer_Type_ID = CT.Customer_Type_ID
                AND CT.Text = 'Rząd'
                AND OD.Order_ID = O.Order_ID
                AND O.Customer_ID = C.Customer_ID
                AND O.Status_Code = OT.Code
                AND O.Completed_Flag = 'N'
                AND OT.Code_Type = 'STAN_ZAMOWIENIA'
                AND OT.Text!= 'Odwolane')
ORDER BY <Kolumny jedynie z zapytania zewnętrznego>
```

Używając nowej notacji złączeń typu semi-join możemy to zapytanie przedstawić na diagramie, tak jak pokazano je na rysunku 7.29



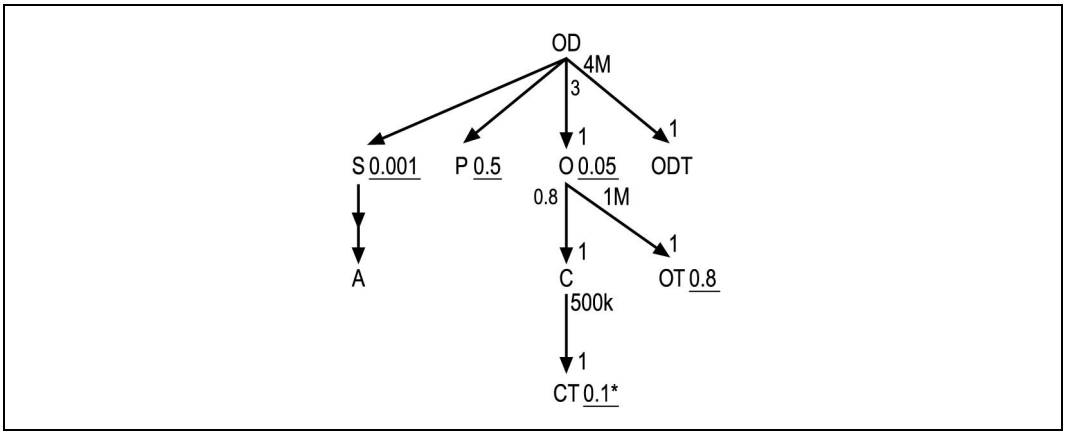
Rysunek 7.29. Złożony przykład złączenia semi-join skorelowanego z kluczem głównym źródłowej tabeli szczegółów podzapytania

Jeśli po prostu przepiszemy to zapytanie, przenosząc złączenia tabel i wszystkie warunki z podzapytania do zapytania zewnętrznego otrzymamy zapytanie, które ma tą samą funkcjonalność, ponieważ złączenie typu semi-join jest skierowane ku kluczowi głównemu, a podzapytanie jest złączone jeden-do-jednego ze swoją źródłową tabelą szczegółów:

```
SELECT <Kolumny jedynie z oryginalnego zapytania zewnętrznego>
FROM Order_Details OD, Products P, Shipments S,
     Addresses A, Code_Translations ODT,
     Orders O, Customers C, Code_Translations OT, Customer_Types CT
WHERE OD.Product_ID = P.Product_ID
  AND P.Unit_Cost > 100
  AND OD.Shipment_ID = S.Shipment_ID
  AND S.Address_ID = A.Address_ID(+)
  AND OD.Status_Code = ODT.Code
  AND ODT.Code_Type = 'Dokladny_Status_Zamowienia'
  AND S.Shipment_Date > :now - 1
                AND C.Customer_Type_ID = CT.Customer_Type_ID
                AND CT.Text = 'Rząd'
                AND OD.Order_ID = O.Order_ID
                AND O.Customer_ID = C.Customer_ID
                AND O.Status_Code = OT.Code
                AND O.Completed_Flag = 'N'
                AND OT.Code_Type = 'STAN_ZAMOWIENIA'
                AND OT.Text!= 'Odwolane'
ORDER BY <Kolumny jedynie z oryginalnego zapytania zewnętrznego>
```

Aby transformacja z jednej formy w drugą stała się oczywista, zrobiłem odpowiednie wcięcia w kodzie. Diagram zapytania jest niemal identyczny, jak można zobaczyć na rysunku 7.30.

Ta nowa forma ma duży stopień dodatkowej swobody, pozwalając na przykład na plan, który łączy średnio filtrowany węzeł `P` tuż po złączeniu z wysoce filtrowanym węzłem `O`, ale przed złączeniem z niemal nieprzefiltrowanym węzłem `OT`. W oryginalnej formie baza danych byłaby zmuszona do wykonania całego podzapytania, zanim mogłaby rozważyć złączenia z węzłami zapytania zewnętrznego. Ponieważ w przypadkach takich jak ten



Rysunek 7.30. To samo zapytanie zmienione przez włączenie podzapytania

włączanie podzapytań w zapytania zewnętrzne może tylko pomóc i ponieważ takie włączenie tworzy diagram, który umiemy już optymalizować, oprę się na założeniu obowiązującym w pozostałej części podrozdziału, że ten typ podzapytania będziemy zawsze włączali w zapytanie zewnętrzne. Opiszę jedynie, jak tworzyć diagramy i optymalizować inne przypadki.

W teorii, ten sam trik może być zaadoptowany do włączania podzapytań typu `EXISTS`, w których złączenie typu semi-join jest także skierowane do końca związanego z tabelą szczegółów, ale wykonanie go jest trudniejsze i z mniejszym prawdopodobieństwem przyniesie jakąś poprawę wydajności. Przeanalizujmy wcześniejsze zapytanie odnośnie departamentów z warunkiem `EXISTS` na `Employees`:

```
SELECT ...
FROM Departments D
WHERE EXISTS (SELECT NULL
              FROM Employees E
              WHERE E.Department_ID=D.Department_ID)
```

A oto na jakie problemy natrafimy przy próbie zastosowania powyżej przedstawionego triku:

- Oryginalne zapytanie zwraca co najwyżej jeden wiersz dla każdego wiersza tabeli nadrzędnej i każdego departamentu. Aby uzyskać te same rezultaty z zapytania po transformacji, ze zwykłym złączeniem z tabelą szczegółów (`Employees`), musielibyśmy wymienić liście `SELECT` unikalny klucz tabeli nadrzędnej oraz wykonać operację `DISTINCT` na wierszach wynikowych.
  Poczynienie tych kroków usuwa duplikaty, które pojawiają się, gdy jeden wiersz tabeli nadrzędnej posiada wiele odpowiadających mu wierszy tabeli szczegółów.
- Jeśli wiersz nadrzędny ma wiele pasujących wierszy tabeli szczegółów, często bardziej kosztowną operacją jest znalezienie ich wszystkich niż wstrzymanie złączenia semi-join, tuż po znalezieniu pierwszej pasującej pary.

Dlatego jeśli złączenie semi-join jest skierowane ku górze, nie powinno się zamieniać złączeń tego typu na zwykłe złączenia, za wyjątkiem sytuacji, gdy współczynnik złączenia szczegółów dla złączenia semi-join jest bliski 1,0 lub nawet nieco mniejszy.

Aby dokończyć tworzenie diagramu podzapytania typu `EXISTS`, musimy mieć kilka reguł pozwalających nam oszacować współczynnik korelacji oraz skorygowany współczynnik filtrowania podzapytania. W oszacowaniu współczynnika korelacji może nam pomóc opisana poniżej procedura.

1. Dla złączenia typu semi-join, niech D będzie współczynnikiem złączenia szczegółów, M —współczynnikiem złączenia tabeli nadrzędnej. Niech S będzie najlepszym (najmniejszym) współczynnikiem filtrowania spośród wszystkich węzłów podzapytania oraz niech R będzie najlepszym (najmniejszym) współczynnikiem filtrowania wszystkich węzłów zapytania zewnętrznego.
2. Jeśli $D*S < M*R$ to współczynnik korelacji będzie równy $(D*S)/(M*R)$.
3. Natomiast jeśli $S > R$, współczynnik korelacji będzie równy S/R.
4. W przeciwnym wypadku, niech E będzie zmierzonym czasem wykonania najlepszego planu, który wiedzie od zapytania zewnętrznego do podzapytania (podążając za logiką działania `EXISTS`). Niech I będzie zmierzonym czasem wykonania najlepszego planu prowadzącego z zapytania wewnętrznego do zewnętrznego (w zgodzie z logiką `IN`). Wówczas współczynnik korelacji będzie równy I/E. Jeśli krok 2. lub 3. pozwoli nam znaleźć przybliżony współczynnik korelacji, możemy całkiem bezpiecznie określić, w którym kierunku poprowadzić podzapytanie bez mierzenia czasów wykonania.

Szacunkowa wartość uzyskana w kroku 2. i 3. może nie być tak celna, jak bezpośrednie zmierzenie czasu wykonania. Jednak szacowanie jest wystarczające, jeśli tylko dokonywane jest ostrożnie, tak aby uniknąć wartości prowadzących do niewłaściwego wyboru źródła — zapytania zewnętrznego lub podzapytania. Reguły przedstawione w kroku 2. oraz 3. są zaprojektowane specjalnie dla przypadków, w których takie ostrożne, zachowawcze szacowania są wykonalne.

Jeśli w kroku 2. lub 3. nie uda się uzyskać estymacji, wartością, która będzie najbezpieczniejsza i najłatwiejsza do uzyskania jest po prostu wartość zmierzona. W takim przypadku, który występuje dość rzadko, znalezienie dobrej wartości na podstawie obliczeń byłoby na tyle skomplikowane, że nie jest warte wysiłku.

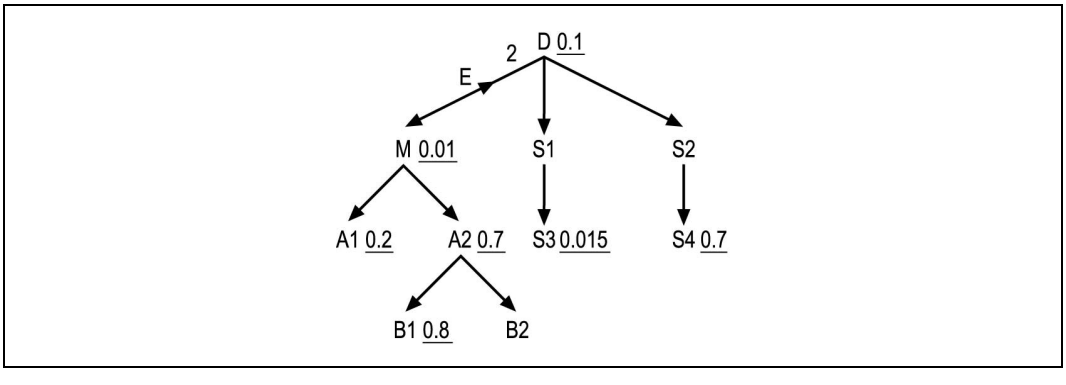
W momencie znalezienia współczynnika korelacji musimy sprawdzić, czy potrzebujemy skorygowanego współczynnika filtrowania podzapytania i w takim przypadku spróbować go ustalić:

1. Jeśli współczynnik korelacji jest mniejszy niż 1,0 oraz mniejszy niż wszystkie pozostałe współczynniki korelacji (w wypadku wielu podzapytań) musimy się zatrzymać. Wówczas nie potrzebujemy współczynnika filtrowania podzapytania, ponieważ jest on pomocny jedynie w przypadku, gdy ustalamy, kiedy wyjdziemy ze złączenia zewnętrznego, co nie ma w tej sytuacji zastosowania.

2. Jeśli podzapytanie jest zapytaniem na jednej tabeli bez warunków filtrowania, jedynie ze skorelowanym warunkiem złączenia, musimy zmierzyć q (liczba wierszy zapytania zewnętrznego z usuniętym warunkiem podzapytania) i t (liczba wierszy pełnego zapytania, włączając w to podzapytanie). Wówczas skorygowany współczynnik filtrowania podzapytania będzie równy t/q. W tym przypadku, warunek `EXISTS` jest szczególnie łatwy do sprawdzenia — baza danych szuka w indeksie złączenia pierwszego dopasowania.
3. W przeciwnym przypadku, dla złączenia typu semi-join, niech D będzie współczynnikiem złączenia tabeli szczegółowej, a s — współczynnikiem filtrowania skorelowanego węzła (węzła przyłączonego do złączenia semi-join) na szczegółowym końcu podzapytania.
4. Jeśli $D \leq 1$, skorygowany współczynnik filtrowania podzapytania będzie równy $s*D$.
5. W przeciwnym przypadku, jeśli $s*D < 1$, skorygowany współczynnik filtrowania podzapytania będzie równy $(D–1+(s*D))/D$.
6. W przeciwnym przypadku, niech skorygowany współczynnik filtrowania podzapytania będzie równy 0,99. Nawet najsłabszy warunek filtrowania `EXISTS` zapobiegnie rozmnożeniu wierszy i oferuje większą siłę filtrowania na jednostkę niż złączenie dolne bez żadnych filtrów. Ta ostatnia reguła obsługuje wszystkie pozostałe przypadki tego typu. Lepsze to niż nic (zresztą niewiele lepsze).

Tak jak pozostałe reguły zawarte w tej książce, reguły obliczania współczynnika korelacji oraz skorygowanego współczynnika filtrowania podzapytania są regułami heurystycznymi. Ponieważ dokładne wartości są rzadko potrzebne, aby wybrać dobry plan wykonania, ta starannie zaprojektowana, wydajna heurystyka daje dobre rozwiązanie w co najmniej 90% i praktycznie nigdy nie prowadzi do nieoptymalnego wyboru. Podobnie jak w innych częściach tej książki, idealna kalkulacja złożonego zapytania jest poza zasięgiem manualnych metod optymalizacji.

Sprawdźmy, czy powyższe reguły są zrozumiałe w wystarczającym stopniu, aby ustalić współczynnik korelacji oraz skorygowany współczynnik filtrowania podzapytania oraz uzupełnić rysunek 7.31, na którym brakuje tych dwóch liczb.



Rysunek 7.31. Złożone zapytanie z brakującymi wartościami dla złączenia semi-join

Wyniki naszych działań możemy porównać z następującym rozwiązaniem problemu:

1. Niech $D = 2$ *i* $M = 1$ (brak wartości na diagramie oznacza 1). Niech $S = 0{,}015$ (najlepszy współczynnik filtrowania z podzapytania należy do węzła `S3`, który znajduje się dwa poziomy poniżej źródłowej tabeli szczegółów podzapytania — tabeli `D`). Następnie ustalmy $R = 0{,}01$, które jest najlepszym filtrem na przestrzeni wszystkich węzłów drzewa poniżej źródłowej tabeli szczegółów zapytania zewnętrznego `M`, z nią samą włącznie.
2. Ponieważ $D*S = 0{,}03$ oraz $M*R = 0{,}01$, $D*S > M*R$, przechodzimy do kroku 3.
3. Ponieważ $S > R$, współczynnik korelacji jest równy S/R, czyli 1,5.

Aby znaleźć skorygowany współczynnik filtrowania podzapytania, wykonajmy następujące czynności:

1. Zauważamy, że współczynnik korelacji jest większy niż 1, więc przechodzimy do kroku 2.
2. Zauważamy, że podzapytanie złożone jest z wielu tabel i filtrów, toteż przechodzimy do punktu 3.
3. Znajdujemy $D = 2$ oraz współczynnik filtrowania na węźle D — $s = 0{,}1$.
4. Zauważamy, że $D > 1$, więc przechodzimy do kroku 5.
5. Obliczamy $s*D = 0{,}2$, co jest wartością mniejszą niż 1, a więc szacujemy skorygowany współczynnik filtrowania podzapytania przez $(D -1+(s*D))/D = (2–1+(0{,}1*2))/2 = 0{,}6$.

W jednym z następnych podrozdziałów, dotyczącym optymalizacji podzapytań `EXISTS`, pokażę optymalizację całego diagramu z rysunku 7.32.

Tworzenie diagramów dla podzapytań NOT EXISTS

Warunek podzapytania, który można wyrazić przez `NOT EXISTS` lub też `NOT IN` są pod jednym względem prostsze niż podzapytania typu `EXISTS` — nie można wyjść z podzapytania ku zapytaniu zewnętrznemu. Eliminuje to potrzebę znajdowania współczynnika korelacji. Litera *E*, symbolizująca podzapytanie typu `EXISTS`, jest zamieniona przez *N*, oznaczające warunek podzapytania typu `NOT EXISTS`, a współczynnik korelacji jest znany jako złączenie typu *anti-join*, ponieważ poszukuje ono przypadku, w którym złączenie z wierszami podzapytania nie znajdzie żadnych elementów pasujących.

Okazuje się, że niemal zawsze lepiej jest wyrażać zapytania typu `NOT EXISTS`, warunkiem `NOT EXISTS` niż warunkiem `NOT IN`. Rozważmy następujący szablon podzapytania `NOT IN`:

```
SELECT ...
FROM ... Outer_Anti_Joined_Table_Order Outer
WHERE ...
  AND Outer.Some_Key NOT IN (SELECT Inner.Some_Key
                             FROM ... Subquery_Anti_Joined_Table Inner WHERE

<Warunki_I_Zlaczenia_Jedynie_Na_Tabelach_Podzapytania>)
...
```

Możemy i powinniśmy wyrazić to zapytanie w równoważnej formie `NOT EXISTS`:

```
SELECT ...
FROM ... Outer_Anti_Joined_Table_Order Outer
WHERE ...
  AND Outer.Some_Key IS NOT NULL
  AND NOT EXISTS (SELECT null
                  FROM ... Subquery_Anti_Joined_Table Inner WHERE
                  <Warunki_I_Zlaczenia_Jedynie_Na_Tabelach_Podzapytania>)
                  AND Outer.Some_Key = Inner.Some_Key)
```

Aby zamienić zapytanie `NOT IN` na `NOT EXISTS` bez zmian funkcjonalności, musimy dodać warunek `NOT NULL` na kluczu złączenia skorelowanego w tabeli zewnętrznej. Dzieje się tak, ponieważ warunek `NOT IN` jest równoważny serii warunków różny-od połączonych za pomocą `OR`, ale według bazy danych warunek `NULL!=` ➥*<JakasWartosc>* nie jest prawdziwy, a więc forma `NOT IN` odrzuca wszystkie wiersze zapytania zewnętrznego dzięki kluczowi skorelowanemu o wartości `null`. Ten fakt nie jest powszechnie znany, więc możliwe jest, że prawdziwą intencją twórcy takiego zapytania było włączenie w rezultat zapytania tych wierszy, które forma `NOT IN` odrzuca. Po zamianie form mamy dobrą okazję, aby wyszukać i naprawić tego typu błędy.

Oba warunki podzapytania, typu `EXISTS` i `NOT EXISTS`, zaprzestają poszukiwań pasujących wierszy tuż po znalezieniu pierwszego dopasowania, jeśli tyko takie istnieje.

Warunki podzapytań typu `NOT EXISTS` są potencjalnie bardziej pomocne we wczesnych fazach planu wykonania, ponieważ kiedy odpowiednie wiersze znalezione są wcześnie, zapytanie odrzucając je sprawia, że późniejsze kroki planu wykonania odbywają się szybciej. W przypadku warunku `EXISTS`, baza danych musi sprawdzić każdą możliwość pasujących do siebie wierszy, po czym wszystkie je odrzuca, co jest dużo kosztowniejszą operacją, gdy istnieje wiele szczegółów na wiersz nadrzędny, złączone przez złączenie semi-join. Zapamiętajmy następujące reguły, które porównują warunki `EXISTS` i `NOT EXISTS` wskazujące na tabele detali z nadrzędnej tabeli zapytania zewnętrznego:

- *Nieselektywny* warunek `EXISTS` nie jest kosztowny przy testowaniu (ponieważ łatwo znajduje dopasowanie, zazwyczaj na pierwszym wierszu złączonym przez semi-join), ale odrzuca niewiele wierszy z zapytania zewnętrznego. Im więcej wierszy zwróciłoby izolowane podzapytanie, tym mniej kosztowny i selektywny jest warunek `EXISTS`. Warunek `EXISTS`, który cechowałby się dużą selektywnością, byłby także kosztowny w sprawdzaniu, gdyż musi on odrzucić dopasowanie na każdym wierszu szczegółów.
- *Selektywny* warunek `NOT EXISTS` nie jest kosztowny przy sprawdzaniu (ponieważ łatwo znajduje dopasowania, zazwyczaj już na pierwszym wierszu złączonym przez semi-join) i odrzuca wiele wierszy z zapytania zewnętrznego. Im więcej wierszy zwróciłoby izolowane podzapytanie tym mniej kosztowny i bardziej selektywny jest warunek `EXISTS`. Z drugiej strony, mało selektywne warunki `NOT EXISTS` są kosztowne w sprawdzaniu, gdyż muszą potwierdzić brak dopasowania dla każdego wiersza szczegółów.

Ponieważ konwersja warunków podzapytania typu NOT EXISTS na równoważne proste zapytanie nie zawierające podzapytań, jest trudna i mało wynagradzająca, często używa się podzapytań NOT EXISTS na obu końcach złączenia anti-join — na końcu tabeli nadrzędnej i tabeli szczegółów. Rzadko istnieje potrzeba poszukiwania innych sposobów wyrażenia warunku NOT EXISTS.

Ponieważ selektywne warunki NOT EXISTS nie są kosztowne przy sprawdzaniu, okazuje się, że oszacowanie skorygowanego współczynnika filtrowania podzapytania jest całkiem łatwe:

1. Należy zmierzyć q (liczba wierszy zapytania zewnętrznego z usuniętym warunkiem podzapytania typu NOT EXISTS) i t (liczba wierszy pełnego zapytania, włączając w to podzapytanie). Niech C będzie liczbą tabel w klauzuli FROM podzapytania (zazwyczaj dla warunków NOT EXISTS jest to jedna tabela).
2. Ustalmy skorygowany współczynnik filtrowania podzapytania na $(C–1+(t/q))/C$.

Optymalizacja zapytań zawierających podzapytania

Podobnie jak w przypadku prostych zapytań, optymalizacja zapytań złożonych z podzapytaniami jest całkiem prosta, jeśli tylko dysponujemy odpowiednim diagramem zapytania. Poniżej przedstawiono niezbędne kroki postępowania dla optymalizacji złożonych zapytań z podzapytaniami, które pozwalają utworzyć kompletny diagram zapytania.

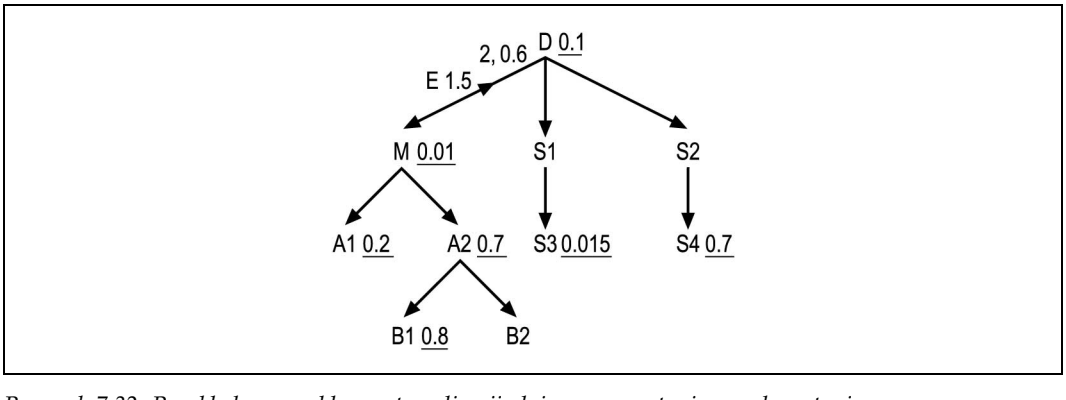
1. Należy zamienić każdy warunek NOT IN na równoważny warunek NOT EXISTS, według wcześniejszego szablonu.
2. Jeżeli złączenie skorelowane jest typu EXISTS, a podzapytanie znajduje się na nadrzędnym końcu tego złączenia (strzałka pośrednia jest skierowana ku dołowi), trzeba zamienić zapytanie złożone na proste, tak jak pokazano to wcześniej oraz zoptymalizować je, używając zwykłych reguł dla prostych zapytań.
3. W przeciwnym przypadku, jeśli złączenie skorelowane jest typu EXISTS, należy znaleźć najniższy współczynnik korelacji spośród wszystkich zapytań typu EXISTS (jeśli tylko istnieje więcej niż jeden). Jeśli ten najmniejszy współczynnik korelacji jest mniejszy niż 1,0, musimy zamienić ten warunek podzapytania na równoważny warunek IN i wyrazić pozostałe warunki podzapytań typu EXISTS poprzez jawne wykorzystanie warunku EXISTS. Następnie musimy zoptymalizować nieskorelowane podzapytanie typu IN, tak jakby było ono samodzielnym zapytaniem — jest to punkt początkowy planu wykonania całego zapytania. Po zakończeniu wykonywania podzapytania nieskorelowanego, baza danych przeprowadzi operację sortowania, aby odrzucić z listy wygenerowanej przez podzapytanie, zduplikowane klucze skorelowania. Następnym krokiem w złączeniu, po ukończeniu pierwszego podzapytania, jest skorelowany klucz w zapytaniu zewnętrznym, wykorzystując do tego indeks na kluczu tego złączenia. Od tego momentu zapytanie zewnętrzne należy traktować jak gdyby źródłowe podzapytanie nie istniało i jakby pierwszym węzłem była źródłowa tabela zapytania zewnętrznego.

4. Jeśli wszystkie współczynniki korelacji są większe lub równe 1,0 lub jeśli warunki podzapytania są jedynie typu `NOT EXISTS`, tabelę źródłową wybieramy z zapytania zewnętrznego, tak jakby zapytanie to nie miało żadnych warunków podzapytania, zgodnie ze zwykłymi regułami odnoszącymi się do prostych zapytań.
5. W momencie osiągnięcia węzłów zapytania zewnętrznego, które zawiera złączenia semi-join lub anti-join z jeszcze niewykonanymi podzapytaniami, całe podzapytanie należy traktować, jakby było ono pojedynczym, podłączonym od dołu węzłem (nawet jeśli złączenie skorelowane jest w rzeczywistości górne). Przy wyborze momentu wykonania pozostałych podzapytań powinno się przyjąć, że ten wirtualny węzeł ma współczynnik filtrowania równy skorygowanemu współczynnikowi filtrowania podzapytania.

Ponieważ podzapytania skorelowane zatrzymują się na pierwszym pasującym wierszu — jeśli takowy istnieje — zapobiegają ryzyku rozmnożenia wierszy, charakterystycznego dla normalnych złączeń górnych, i mogą zmniejszyć liczbę wierszy będących w obiegu. Jednak ponieważ często zachodzi konieczność sprawdzenia wielu wierszy, aby uzyskać taki efekt filtrowania, prawdziwa wartość skorygowanego współczynnika filtrowania podzapytania czyni ten wirtualny węzeł równoważny węzłowi niemal nieprzefiltrowanemu (jeśli rozpatruje się współczynnik zysku do kosztu).

6. Tuż po umieszczeniu złączenia skorelowanego w porządku złączenia w oparciu o krok 5., natychmiast należy wykonać pozostałą część tego skorelowanego podzapytania, optymalizując plan wykonania podzapytania z węzłem skorelowanym jako węzłem źródłowym. Po zakończeniu podzapytania wracamy do zapytania zewnętrznego i kontynuujemy optymalizację pozostałej części porządku złączenia.

Jako przykład, rozważmy rysunek 7.32, który jest po prostu uzupełnionym rysunkiem 7.31, z wypełnionymi wartościami współczynnika korelacji oraz skorygowanego współczynnika filtrowania podzapytania.



Rysunek 7.32. Przykładowy problem optymalizacji złożonego zapytania z podzapytaniem

Ponieważ złączenie skorelowane jest typu `EXISTS`, krok 1. nie ma tu zastosowania. Podobnie, ponieważ strzałka pośrednia złączenia semi-join jest skierowana ku górze, krok 2. także nie ma tu racji bytu. Najniższy (jedyny) współczynnik korelacji jest równy 1,5 (tuż obok *E*), tak więc i krok 3. nie można tu zrealizować. Przechodząc do kroku 4., znajdujemy, że najlepszym węzłem źródłowym zapytania zewnętrznego jest `M`. W kroku 5. wybieramy pomiędzy złączeniami dolnymi z `A1` i `A2`, ze współczynnikami filtrowania równymi odpowiednio 0,2 i 0,7, a złączeniem dolnym z wirtualnym węzłem reprezentującym całe podzapytanie, o wirtualnym współczynniku filtrowania równym 0,6. `A1` jest najlepszym spośród tych trzech kandydatów, ponieważ ma najodpowiedniejszy współczynnik filtrowania, a więc łączymy się z nim w następnej kolejności. Ponieważ nie osiągniemy żadnego węzła idąc ku dołowi od `A1`, następnym kandydatem do złączenia jest podzapytanie (wykorzystując krok 5.), a więc wykonujemy złączenie semi-join z węzłem `D`.

Stosując się do kroku 6. procedury (gdyż rozpoczęliśmy podzapytanie), musimy dalej podążać tą ścieżką, wychodząc od `D` jako od węzła źródłowego. Wykorzystując w obrębie tego podzapytania reguły dla prostych zapytań, łączymy się z `S1`, `S3`, `S2` i `S4`, w tym porządku. Wracamy do zapytania zewnętrznego, stosując reguły dla prostych zapytań i znajdujemy pozostały porządek złączenia — `A2`, `B1`, `B2`. Zatem cały, optymalny plan złączenia, uwzględniając złączenia semi-join, ma postać `(M, A1, D, S1, S3, S2, S4, A2, B1, B2)`.

Czy trzeba się przejmować podzapytaniami?

Reguły odnoszące się do podzapytań są tak skomplikowane, że aż zniechęcające do ich stosowania, dlatego chciałbym się przyznać do czegoś, co być może polepszy samopoczucie Czytelnika. Na przestrzeni 10 lat zoptymalizowałem zapytania SQL zawierające podzapytania bez konieczności stosowania takich formalnych reguł, podążając wyłącznie za intuicją i generalną regułą wyboru pomiędzy wyjściem z podzapytania lub z zapytania zewnętrznego, która to reguła opiera się na wyborze najbardziej selektywnego warunku. Zazwyczaj nie przejmuję się zbytnio momentem wykonania podzapytań, za wyjątkiem przypadków gdy powinny one być źródłem całego zapytania, ponieważ taka kontrola jest zazwyczaj bardzo utrudniona a zysk okazuje się niewielki, w porównaniu z sytuacją, gdy wybór pozostawimy automatycznemu optymalizatorowi.

Czasami (rzadziej jednak niż mogłoby się wydawać) udaje mi się rozpoznać przypadki leżące na pograniczu, dla których wypróbowanie różnych alternatyw ma sens. To samo podejście może zadziałać i u Czytelnika. Jeśli jednak nie mamy zaufania do swojej intuicji lub chcemy mieć dla niej jakieś oparcie, te formalne reguły okażą się przydatne. Reguły zawarte w tym rozdziale, które stworzyłem specjalnie na potrzeby tej książki, są efektywne w pełnym zakresie rzeczywistych zapytań oraz zapewniają sensowny kompromis pomiędzy matematyczną perfekcją a użytecznością. Jedynie pełna kalkulacja może rozwiązać pełną złożoność problemu w sposób idealny, bez wykorzystania metody prób i błędów. Jednak powyższe reguły dają coś w rodzaju automatycznej intuicji, która, jeśli tylko się chce, jest równie dobra, co ta stosowana przeze mnie.

Zapytania z widokami

Widok może spowodować, że dowolnie skomplikowane zapytanie będzie wyglądało jak prosta tabela, z punktu widzenia osoby piszącej zapytanie wykorzystujące ten widok. Kiedy wiele zapytań współdzieli jakąś część planu wykonania SQL, współdzielone widoki wielokrotnego użytku mogą być bardzo silnym mechanizmem redukującym złożoność kodu aplikacji. Niestety, ukrywanie pewnych działań przed okiem programisty nie zmniejsza skomplikowania kroków niezbędnych do osiągnięcia rzeczywistych danych. Z drugiej strony, ukrywanie złożoności przed programistą najprawdopodobniej nie zwiększy trudności problemu, który optymalizator, czy to automat, czy też człowiek, musi pokonać, aby znaleźć szybki plan wykonania. W dyskusji tej odnosić się będę do dwóch zapytań ważnych dla problemu optymalizacji. Są to zatem:

Zapytania tworzące widoki

Zapytania, które stanowią podstawę dla widoków (zapytania używane do tworzenia widoków za pomocą `CREATE VIEW <NazwaWidoku> AS <ZapytanieDefiniujaceWidok>`).

Zapytania wykorzystujące widoki

Są to zapytania, które należy optymalizować i które są wykonywane przez bazę danych. Odwołują się one do widoków w klauzuli `FROM` (na przykład `SELECT ... FROM Widok1 V1, Widok2 V2, ... WHERE ...`).

Często jestem proszony o optymalizację lub oszacowanie wydajności zapytania definiującego widok bez listy zapytań z niego korzystających. Nierzadko też prosi się mnie o optymalizację zapytań wykorzystujących widoki bez wiedzy odnośnie zapytań tworzących widoki. Żadna z próśb nie jest możliwa do spełnienia — żadne zapytanie definiujące widok, które jest bardziej skomplikowane niż `SELECT <ListaProstychKolumn> FROM <PojedynczaTabela>`, nie zadziała dobrze w każdym możliwym zapytaniu wykorzystującym ten widok. Podobnie, żadne zapytanie wykorzystujące jakiś widok nie zadziała poprawnie, jeśli zapytanie tworzące widok koliduje z wydajną drogą do osiągnięcia potrzebnych danych.

Dla danego widoku musimy znać i zoptymalizować każde zapytanie wykorzystujące ten widok, abyśmy mogli powiedzieć, że zapytanie je definiujące jest w tym kontekście całkowicie poprawne. Analogicznie, musimy znać zapytanie definiujące każdy użyty widok, abyśmy mogli stwierdzić, czy zapytanie wykorzystujące te widoki jest poprawne.

Kiedy optymalizujemy SQL, widoki zwiększają zazwyczaj złożoność na jeden z trzech sposobów:

- W celu utworzenia optymalnego diagramu złączeń, zachodzi potrzeba zamiany zapytania wykorzystującego widok na zapytanie równoważne, wykorzystujące rzeczywiste tabele.

- Zapytania do widoków często zawierają w szkielecie zapytania zbędne węzły. Każdy widok niesie ze sobą całe zapytanie, które je definiuje, wraz z całym poddrzewem złożonym z wszystkich definiujących widok węzłów i złączeń. Użycie widoku oznacza użycie całego poddrzewa. Jednakże programista wykorzystujący ten widok może potrzebować tylko kilku kolumn widoku i może ominąć jakąś liczbę węzłów i złączeń w zapytaniu definiującym widok, jeśli napisze równoważne zapytanie do zwykłych tabel. Kiedy aplikacja potrzebuje wszystkich węzłów widoku, wciąż zachodzi prawdopodobieństwo, że zapytanie wykorzystujące widok dotyka te węzły w mało efektywny sposób, łącząc się z tymi samymi wierszami tych samych, leżących u podstawy widoku, tabel w wielu ukrytych kontekstach. Przykład będzie przedstawiony w części „Nadmiarowe operacje odczytu w zapytaniach wykorzystujących widoki".
- Czasami zapytania wykorzystujące widoki nie mogą być w prosty sposób wyrażone poprzez równoważne zapytania do zwykłych tabel. Zazwyczaj takie przypadki, w których zapytania do widoków zwracają inne rezultaty niż zapytania do tabel, są sporadyczne. Jednakże rezultaty poprawne w takich przypadkach nie są uzyskiwane z zapytań do widoków! Kiedy zapytanie do widoków nie pozwala się dobrze rozłożyć na proste, równoważne zapytanie do tabel, wydajność takiego zapytania jest zazwyczaj niska, a te sporadyczne przypadki, które definiują funkcjonalność specyficzną dla poszczególnych widoków, są zazwyczaj błędne. Tak czy inaczej, aby poprawić wydajność za pomocą niemal równoważnego prostego zapytania do tabel, wymagane są zwykle drobne zmiany w funkcjonalności i trzeba być szczególnie ostrożnym, aby nie wprowadzić jakiegoś nowego błędu. (Pomimo tego, że poprawia się więcej błędów niż się ich tworzy, ich ilość i tak wzrasta!) Odpowiedni przykład będzie przedstawiony w podrozdziale „Złączenia zewnętrzne z widokami".

Tworzenie diagramów dla zapytań wykorzystujących widoki

Tworzenie diagramów dla zapytań wykorzystujących widoki jest relatywnie proste, chociaż czasem męczące. Poniżej przedstawiono instrukcję postępowania w takich wypadkach.

1. Utworzyć diagram każdego zapytania definiującego widok, tak jakby było ono zapytaniem samodzielnym. Każdy diagram, który definiuje widok powinien być zwyczajny, tzn. powinien mieć jedną źródłową tabelę szczegółów, a złączenia powinny być typu wiele-do-jednego i powinny być skierowane ku dołowi, od węzła głównego. Jeśli zapytanie definiujące widok nie mapuje się na szkielet normalnego zapytania, zapytania wykorzystujące ten widok będą najprawdopodobniej miały niską wydajność i będą zwracały błędne rezultaty. Klucz główny źródłowej tabeli szczegółów dla zapytania definiującego widok należy traktować jak wirtualny klucz główny całego widoku.
2. Utworzyć diagram zapytania wykorzystującego widok, tak jakby każdy widok był zwykłą tabelą. Złączenie z widokiem powinno mieć strzałkę na końcu przy widoku (widok powinien się znajdować na niższym końcu połączenia)

tylko wtedy, gdy łączy się z wirtualnym kluczem głównym widoku. Symbolicznie można pokazać warunki filtrowania widoku w zapytaniu wykorzystującym widoki jako literę *F*, nie zaprzątając sobie głowy dokładnym wyznaczaniem współczynników filtrowania. Warto narysować przerywaną linią kółko wokół każdego widoku.

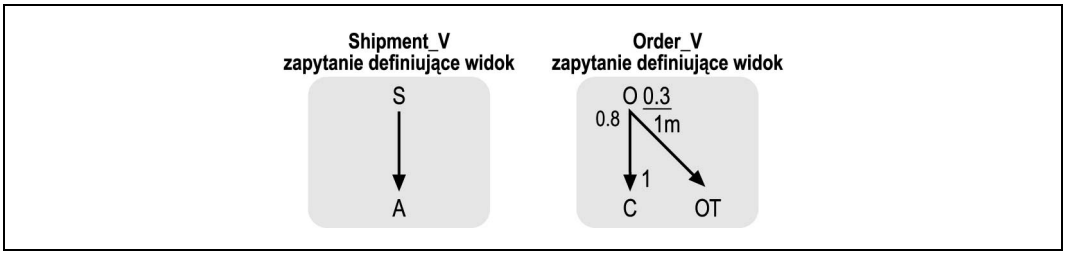
3. Rozszerzyć diagram zapytania wykorzystującego widoki z kroku 2. procedury, zastępując każdy węzeł reprezentujący widok przez cały diagram zapytania definiującego widok z kroku 1., wraz z przerywaną linią dokoła tego diagramu. Każde złączenie od góry będzie złączone z poddrzewem definiującym widok w źródłowym węźle szczegółów. Złączenia wychodzące ku dołowi od widoków mogą się łączyć z dowolnym węzłem widoku, w zależności od tego, która spośród tabel z zapytania definiującego widok ma klucz obcy złączenia (w liście `SELECT` definiującej widok). Dowolny warunek filtrowania na widoku staje się warunkiem filtrowania na odpowiadającym węźle zapytania definiującego widok, w zależności od tego, którą kolumnę węzła warunek filtrowania uwzględnia. Dla każdego z tych warunków znajdujemy współczynnik filtrowania w standardowy sposób, tj. rozwijając symbol *F* na wynikowym diagramie zapytania. Ewentualnie można połączyć współczynnik filtrowania zapytania definiującego widok oraz zapytania go wykorzystującego, jeśli te dwa zapytania posiadają dwa różne filtry na tym samym węźle.

Reguły te mogą się wydawać abstrakcyjne i skomplikowane, ale przykład powinien sprawić, że cały proces stanie się jaśniejszy. Rozważmy następujące dwie definicje widoków:

```
CREATE VIEW Shipment_V AS
SELECT A.Address_ID Shipment_Address_ID, A.Street_Addr_Line1
       Shipment_Street_Address_Line1, A.Street_Addr_Line2
       Shipment_Street_Address_Line2, A.City_Name Shipment_City_Name,
       A.State_Abbreviation Shipment_State, A.ZIP_Code Shipment_ZIP,
       S.Shipment_Date, S.Shipment_ID
FROM Shipments S, Addresses A
WHERE S.Address_ID = A.Address_ID

CREATE VIEW Recent_Orders_V AS
SELECT O.Order_ID, O.Order_Date, O.Customer_ID,
       C.Phone_Number Customer_Main_Phone, C.First_Name Customer_First_Name,
       C.Last_Name Customer_Last_Name,
       C.Address_ID Customer_Address_ID, OT.Text Order_Status
FROM Orders O, Customers C, Code_Translations OT
WHERE O.Customer_ID = C.Customer_ID
  AND O.Status_Code = OT.Code
  AND OT.Code_Type = 'STATUS_ZAMOWIENIA'
  AND O.Order_Date > SYSDATE - 366
```

W pierwszym kroku postępowania musimy utworzyć diagramy tych definiujących widoki zapytań, tak jak pokazano to na rysunku 7.33. Te diagramy zapytań zostały utworzone w oparciu o reguły opisane w rozdziale 5. przy użyciu tych samych współczynników filtrowania i tych samych statystyk złączenia, co dla odpowiedniego przykładu z rysunku 5.5.

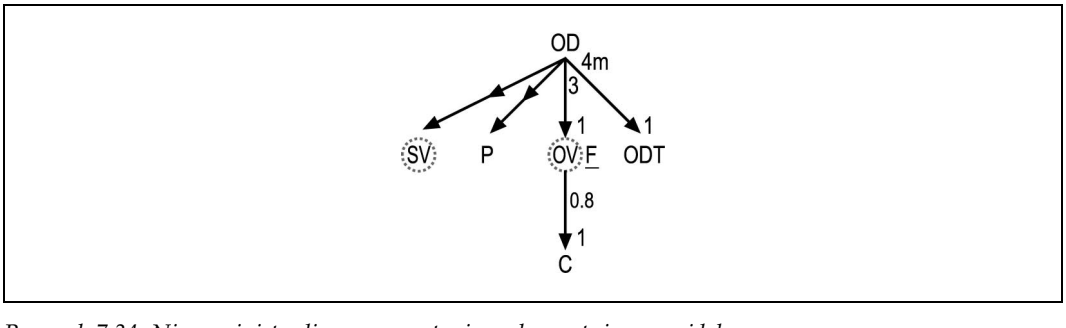


Rysunek 7.33. Diagramy zapytań dla przykładowych zapytań definiujących widoki

Zapytanie wykorzystujące utworzone widoki może mieć zatem postać:

```
SELECT OV.Customer_Main_Phone, C.Honorific, OV.Customer_First_Name,
       OV.Customer_Last_Name, C.Suffix, OV.Customer_Address_ID,
       SV.Shipment_Address_ID, SV.Shipment_Street_Address_Line1,
       SV.Shipment_Street_Address_Line2, SV.Shipment_City_Name,
       SV.Shipment_State, SV.Shipment_ZIP, OD.Deferred_Ship_Date,
       OD.Item_Count, ODT.Text, P.Prod_Description, SV.Shipment_Date
FROM Recents_Order_V OV, Order_Details OD, Products P, Shipment_V SV,
            Code_Translations ODT, Customers C
WHERE UPPER(OV.Customer_Last_Name) LIKE :last_name||'%'
  AND UPPER(OV.Customer_First_Name) LIKE :first_name||'%'
  AND OD.Order_ID = OV.Order_ID
  AND OV.Customer_ID = C.Customer_ID
  AND OD.Product_ID = P.Product_ID(+)
  AND OD.Shipment_ID = SV.Shipment_ID(+)
  AND OD.Status_Code = ODT.Code
  AND ODT.Code_Type = 'DOKLADNY_STATUS_ZAMOWIENIA'
ORDER BY OV.Customer_ID, OV.Order_ID Desc, OD.Shipment_ID,
OD.Order_Details_ID
```

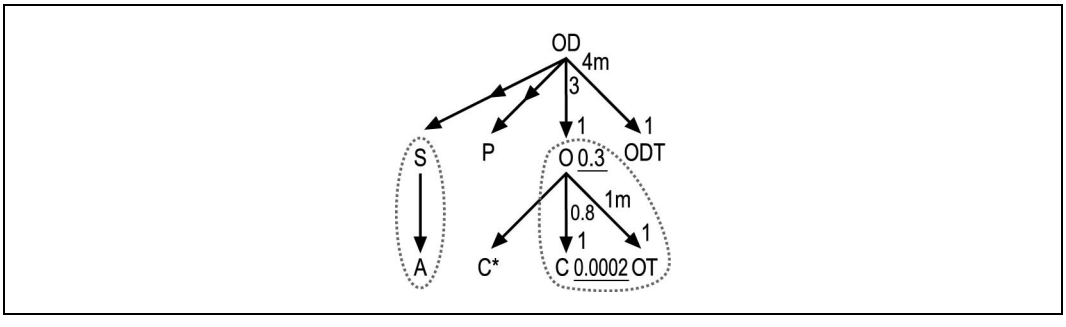
Przechodząc do kroku 2. procedury, tworzymy początkowy diagram zapytania, traktując widoki jak zwykłe tabele, w taki sposób, jak na rysunku 7.34.



Rysunek 7.34. Nierozwinięty diagram zapytania wykorzystującego widok

Zamieńmy każdy węzeł odpowiadający widokowi na rysunku 7.34 przez diagramy zapytań definiujących te widoki z rysunku 7.33 oraz otoczmy każde zapytanie definiujące widok przerywaną linią, tak aby pokazać granice widoku. Uwzględnijmy zapytania definiujące widoki w pozostałej części całego diagramu, podłączając je do odpowiednich węzłów-tabel, w zależności od tego, która tablica w zapytaniu tworzącym widok zawiera klucz łączący. Normalnie, każde złączenie z widokiem, przychodzące z góry, będzie

zakotwiczone w źródłowej tabeli szczegółów zapytania definiującego ten widok. Jednakże węzły tabel nadrzędnych, które połączone są w widokiem od dołu (na przykład węzeł C na rysunku 7.34) mogą się łączyć z dowolnym węzłem szkieletu definiującego widok, w zależności od tego, która tabela zawiera klucz obcy wskazujący na węzeł nadrzędny. Dodajmy liczbowe współczynniki filtrowania do każdego węzła szkieletu zapytania, niezależnie od tego czy należą do zapytania definiującego czy też wykorzystującego widok. Na rysunku 7.35 współczynnik filtrowania równy 0,3 tuż obok węzła O pochodzi z filtra występującego w zapytaniu definiującym widok, podczas gdy współczynnik 0,0002 tuż obok węzła C pochodzi z warunku (na imię i nazwisko klienta) zapytania korzystającego z widoku.



Rysunek 7.35. Rozwinięty diagram zapytania wykorzystującego widok

Graficzny rezultat naszego przykładu powinien wyglądać jak na rysunku 7.35, na którym dodałem gwiazdkę do umieszczonego najbardziej na lewo węzła C, aby podkreślić różnicę pomiędzy dwoma tak samo oznaczonymi węzłami. Ponownie statystyki dla filtra na węźle klienta zapożyczyłem z przykładu pokazanego wcześniej na rysunku 5.5, uzyskując w ten sposób współczynnik 0,0002 tuż obok C w najbardziej na prawo wysuniętym szkielecie.

Te czynności pozwalają nam ukończyć diagram, którego utworzenie jest niezbędne, aby kontynuować optymalizację zapytania wykorzystującego widok, tak byśmy mogli zadecydować, czy do otrzymania optymalnego planu konieczne są zmiany w zapytaniu definiującym widok, czy też w zapytaniu wykorzystującym widok.

Optymalizacja zapytań z widokami

Przeważnie optymalny plan wykonania dla zapytania wykorzystującego widoki jest dokładnie tym samym planem, który uzyskalibyśmy, gdyby występowały w nim tylko zwykłe tabele. Jednakże istnieją cztery utrudnienia, z którymi być może będziemy musieli się zmierzyć:

- Niektóre złączenia ze złożonymi widokami są trudne do wyrażenia w postaci prostych złączeń do prostych tabel. W szczególności są to złączenia zewnętrzne z widokami, które same mają złączenia w zapytaniu je definiującym. Problem ten dotyka także naszego przypadku, a jego wyjaśnienie będzie można znaleźć się w podrozdziale „Złączenia zewnętrzne z widokami".

- Niektóre widoki odnoszą się do dokładnie tych samych wierszy, tej samej tabeli, co jakaś inna tabela w zapytaniu wykorzystującym te widoki, czego rezultatem jest wykonywanie zbytecznej pracy przez bazę danych (efekt, który powinniśmy usunąć). Taki przypadek występuje na diagramie 7.35 dla węzłów `C*` oraz `C`. Problemem tym zajmiemy się również dalszym fragmencie tej publikacji.

  Zaletą widoków, z perspektywy tworzenia oprogramowania jest to, że ukrywają złożoność problemu, ale niestety dokładnie ta zaleta sprawia, że tworzenie nadmiarowych złączeń jest bardzo łatwe. Przy wykorzystaniu jedynie zwykłych tabel sytuacja taka byłaby łatwa do zauważenia i wymagałaby dużo więcej pracy, gdybyśmy chcieli ją ponowić.

- Węzły znajdujące się w obrębie zapytania definiującego widok oraz złączenia do nich prowadzące są często zbyteczne w uzyskiwaniu rezultatów zapytania, które wykorzystuje ten widok.

- Używanie widoków ogranicza naszą kontrolę nad planem wykonania. Jeśli zmienimy zapytanie definiujące widok, tak aby poprawić plan wykonania zapytania wykorzystującego ten widok, możemy przez przypadek obniżyć wydajność innych zapytań go wykorzystujących. Zawsze możemy utworzyć nowy widok do użycia tylko w obrębie jednego zapytania, ale powoduje to zaprzepaszczenie korzyści płynących z możliwości współdzielenia widoków. Ogólnie rzecz biorąc wskazówki SQL oraz inne zmiany w zapytaniu wykorzystującym widoki nie wpływają na sposób w jaki baza danych uzyskuje dostęp do tabel tego zapytania. Czasem, aby otrzymać wydajny plan wykonania, niezbędne jest usunięcie widoków.

Złączenia zewnętrzne z widokami

Powracając do wcześniejszego przykładu, zastanówmy się, co tak naprawdę znaczy złączenie zewnętrzne z widokiem `Shipment_V`, który jest z kolei złączeniem wewnętrznym pomiędzy tabelami `Shipments` i `Addresses`. Ponieważ baza danych musi zachowywać się tak, jak gdyby widok był rzeczywistą tabelą o dokładnie takich wierszach jakie ma widok, złączenie znajduje wewnętrzny przypadek tych `Shipment_ID`, które istnieją w tabeli `Shipments` i wskazują na dostawy o wartościach `Address_ID` pasujących do tabeli `Addresses`. Jeśli baza danych nie będzie mogła znaleźć złączenia z tabelami `Shipments` i `Addresses`, złączenie z widokiem stanie się całkowicie złączeniem zewnętrznym (z oboma tabelami), nawet jeśli pierwotne złączenie z tabelą `Shipments` nie zakończyło się sukcesem. W poszukiwaniu planu zagnieżdżonych pętli, baza danych nie może wiedzieć, czy złączenie zewnętrzne znajdzie przypadek wewnętrzny aż do momentu połączenia się z powodzeniem do obu tabel w zapytaniu definiującym widok.

Niestety, przypadek ten jest zbyt skomplikowany, aby mógł być obsłużony automatycznie, dlatego też baza danych może po prostu zaprzestać poszukiwania planu zagnieżdżonych pętli. Co więcej, baza danych powinna uwzględnić, że niezależnie od tego jak bardzo skomplikowana jest leżąca u podstaw logika, nie może popełnić błędu w takich złożonych przypadkach jak ten, jeśli pobierze każdy wiersz z zapytania definiującego

widok i będzie traktować rezultat jakby pochodził z rzeczywistej tabeli. Dla złączeń zewnętrznych z widokami, baza danych wykorzystuje zazwyczaj do połączenia z tą tymczasową tabelą złączenie sortująco-scalające lub mieszające. Takie działanie jest zazwyczaj bezpieczne, gdy chodzi o funkcjonalność, ale dla wydajności jest nierzadko katastrofalna w skutkach, chyba że zapytanie definiujące widok jest szybkie jako zapytanie samodzielne.

Jako generalną regułę dla utrzymania wydajności warto przyjąć, iż należy unikać złączeń zewnętrznych z każdym widokiem, który jest bardziej skomplikowany niż proste `SELECT <ListaProstychKolumn> FROM <PojedynczaTabela>`.

Podobne problemy występują w przypadku wszystkich złączeń z widokami, które zawierają `UNION` lub `GROUP BY` w zapytaniu definiującym widok. Chociaż trzeba przyznać, iż złączenia *wychodzące* od takich widoków zazwyczaj działają dobrze, pod warunkiem, że zawierają tabelę, którą wybralibyśmy jako źródłową tabelę zapytania.

Przeanalizujmy ponownie zapytanie wykorzystujące widoki z poprzedniego podrozdziału. Jeśli włączymy zapytanie definiujące widok `Shipment_V` do zapytania wykorzystującego widok w celu rozwiązania problemów wydajnościowych związanych ze złączeniem zewnętrznym, możemy spodziewać się następujących rezultatów:

```
SELECT OV.Customer_Main_Phone, C.Honorific, OV.Customer_First_Name,
       OV.Customer_Main_Phone, C.Suffix, OV.Customer_Address_ID,
       A.Address_ID Shipment_Address_ID,
       A.Street_Addr_Line1 Shipment_Street_Address_Line1,
       A.Street_Addr_Line2 Shipment_Street_Address_Line2,
       A.City_Name Shipment_City_Name, A.State_Abbreviation Shipment_State,
       A.ZIP_Code Shipment_ZIP, OD.Deferred_Ship_Date, OD.Item_Count,
       ODT.Text, P.Prod_Description, S.Shipment_Date
FROM Recent_Orders_V OV, Order_Details OD, Products P, Shipments S,
     Addresses A, Code_Translations ODT, Customers C
WHERE UPPER(OV.Customer_Last_Name) LIKE :last_name||'%'
  AND UPPER(OV.Customer_First_Name) LIKE :first_name||'%'
  AND OD.Order_ID = OV.Order_ID
  AND OV.Customer_ID = C.Customer_ID
  AND OD.Product_ID = P.Product_ID(+)
  AND OD.Shipment_ID = S.Shipment_ID(+)
  AND S.Address_ID = A.Address_ID(+)
  AND OD.Status_Code = ODT.Code
  AND ODT.Code_Type = 'DOKLADNY_STATUS_ZAMOWIENIA'
ORDER BY OV.Customer_ID, OD.Order_ID Desc, S.Shipment_ID, OD.Order_Detail_ID
```

Niestety, wynik tego zapytania nie jest dokładnie taki, jak zapytania początkowego, ze względu na szczególne cechy złączeń zewnętrznych z widokami. W szczególności zapytanie oryginalne zwraca `Shipment_Date` o wartości `null` za każdym razem, gdy złączenie całego widoku (włączając w to złączenie z tabelą `Addresses`) z tabelą `Order_Details` nie powiedzie się. Dlatego za każdym razem, gdy dostawa nie ma poprawnego, niepustego `Address_ID`, oryginalne zapytanie zwróci dla `Shipment_Date` wartość `null` pomimo tego, że złączenie z `Shipments` jest poprawne.

Z dużą pewnością można powiedzieć, że to dziwne zachowanie nie było intencją programisty i nie jest funkcjonalnie niezbędne, dlatego powyższa forma najprawdopodobniej zadziała poprawnie, a nawet lepiej niż oryginał w tym szczególnym przypadku. Jednak jakakolwiek zmiana funkcjonalności dla poprawienia wydajności jest działaniem niebezpiecznym. Dlatego przed dokonaniem zmiany, takiej jak przed chwilą opisanej, która włącza zapytanie definiujące widok w główne zapytanie SQL, należy się upewnić, czy nowe zachowanie w takich krańcowych przypadkach jest poprawne oraz ostrzec programistów informując ich, że zmiana może spowodować, iż testy zwrócą inne rezultaty. W mało prawdopodobnym przypadku, gdy oryginalne zachowanie jest naprawdę potrzebne lub gdy nie chcemy zagłębiać się w analizę poprawności pierwotnego zachowania przypadkach krańcowych, można idealnie emulować oryginalną funkcjonalność zapytania w następujący sposób:

```
SELECT OV.Customer_Main_Phone, C.Honorific, OV.Customer_First_Name,
       OV.Customer_Main_Phone, C.Suffix, OV.Customer_Address_ID,
       A.Address_ID Shipment_Address_ID,
       A.Street_Addr_Line1 Shipment_Street_Address_Line1,
       A.Street_Addr_Line2 Shipment_Street_Address_Line2,
       A.City_Name Shipment_City_Name, A.State_Abbreviation Shipment_State,
       A.ZIP_Code Shipment_ZIP, OD.Deferred_Ship_Date, OD.Item_Count,
       ODT.Text, P.Prod_Description,
       DECODE(A.Address_ID, NULL, TO_DATE(NULL),
                                  S.Shipment_Date) Shipment_Date
FROM Recent_Orders_V OV, Order_Details OD, Products P, Shipments S,
     Addresses A, Code_Translations ODT, Customers C
WHERE UPPER(OV.Customer_Last_Name) LIKE :last_name||'%'
  AND UPPER(OV.Customer_First_Name) LIKE :first_name||'%'
  AND OD.Order_ID = OV.Order_ID
  AND OV.Customer_ID = C.Customer_ID
  AND OD.Product_ID = P.Product_ID(+)
  AND OD.Shipment_ID = S.Shipment_ID(+)
  AND S.Address_ID = A.Address_ID(+)
  AND OD.Status_Code = ODT.Code
  AND ODT.Code_Type = 'DOKLADNY_STATUS_ZAMOWIENIA'
ORDER BY OV.Customer_ID, OD.Order_ID Desc,
    DECODE(A.Address_ID, NULL, TO_NUMBER(NULL), S.Shipment_ID),
      OD.Order_Detail_ID
```

Powyższe zapytanie zawiera dwie zmiany, które powodują że zwrócone rezultaty są takie, jak gdyby złączenie z `Shipments` tworzyło przypadek zewnętrzny, niezależnie od tego, czy złączenie z tabelą `Addresses` tworzyło przypadek zewnętrzny. Bez widoku, zapytanie będzie traktowało złączenie z tabelą `Shipments` niezależnie od złączenia z tabelą `Addresses`. Jednakże wyrażenie `DECODE` na obu końcach listy `SELECT` oraz środkowa część listy `ORDER BY` sprawiają, że wewnętrzny przypadek pierwszego złączenia emuluje przypadek zewnętrzny (produkując `NULL` w miejsce `Shipment_Date` i `Shipment_ID`) za każdym razem, kiedy złączenie z `Addresses` znajdzie przypadek zewnętrzny.

Czasami będzie występowała jakaś funkcjonalna potrzeba zastosowania widoku zamiast zwykłych tabel. Najczęstszym powodem jest obejście ograniczeń automatycznie wygenerowanego SQL-a. Może się okazać, że dla wypełnienia zadanej funkcjonalności niezbędne są bardziej złożone konstrukcje SQL, których generator może nie obsługiwać. Popularnym rozwiązaniem jest ukrycie tej złożoności w ręcznie napisanym zapytaniu

tworzącym widok i pozwolenie generatorowi na traktowanie tego widoku jak zwykłej tabeli, zatajając przed nim tę złożoność. W takich przypadkach możliwe jest, że nie uda się pozbyć widoków, tak jak sugerowałem to we wcześniejszych rozwiązaniach. Alternatywnym wyjściem z tej sytuacji jest rozszerzenie zastosowania widoków, ukrywając większą część SQL w ich definicji. Ponieważ poprzedni problem był związany ze złączeniem zewnętrznym z widokiem, moglibyśmy go rozwiązać poprzez wciągnięcie złączenia zewnętrznego do zapytania definiującego widok. W takim przypadku zamienilibyśmy użycie `Shipment_V` na `Order_Details_V`, wykorzystując następujące zapytanie definiujące widok:

```
CREATE VIEW Order_Details_V AS
SELECT A.Address_ID Shipment_Address_ID,
       A.Street_Addr_Line1 Shipment_Street_Address_Line1,
       A.Street_Addr_Line2 Shipment_Street_Address_Line2,
       A.City_Name Shipment_City_Name, A.State_Abbreviation Shipment_State,
       A.ZIP_Code Shipment_ZIP, S.Shipment_Date, S.Shipment_ID,
       OD.Deferred_Ship_Date, OD.Item_Count, OD.Order_ID,
       OD.Order_Detail_ID, OD.Product_ID, OD.Status_Code
FROM Shipments S, Addresses A, Order_Details OD
WHERE OD.Shipment_ID = S.Shipment_ID(+)
  AND S.Address_ID = A.Address_ID(+)
```

Zapytanie wykorzystujące ten widok przybiera taką postać:

```
SELECT OV.Customer_Main_Phone, C.Honorific, OV.Customer_First_Name,
       OV.Customer_Last_Name, C.Suffix, OV.Customer_Address_ID,
       ODV.Shipment_Address_ID, ODV.Shipment_Street_Address_Line1,
       ODV.Shipment_Street_Address_Line2, ODV.Shipment_City_Name,
       ODV.Shipment_State, ODV.Shipment_ZIP, ODV.Deferred_Ship_Date,
       ODV.Item_Count, ODT.Text, P.Prod_Description, ODV.Shipment_Date
FROM Recent_Orders_V OV, Order_Details_V ODV, Products P,
     Code_Translations ODT, Customers C
WHERE UPPER(OV.Customer_Last_Name) LIKE :last_name||'%'
  AND UPPER(OV.Customer_First_Name) LIKE :first_name||'%'
  AND ODV.Order_ID = OV.Order_ID
  AND OV.Customer_ID = C.Customer_ID
  AND ODV.Product_ID = P.Product_ID(+)
  AND ODV.Status_Code = ODT.Code
  AND ODT.Code_Type = 'DOKLADNY_STATUS_ZAMOWIENIA'
ORDER BY OV.Customer_ID, OV.Order_ID Desc, ODV.Shipment_ID,
ODV.Order_Detail_ID
```

Nadmiarowe operacje odczytu w zapytaniach wykorzystujących widoki

Przejdźmy teraz do przypadku złączenia z węzłami `C*` oraz `C`, z rysunku 7.35. Węzły te reprezentują tę samą tabelę, z identycznymi składniami złączenia, więc każdy plan wykonania, który korzysta z nich obu odczytuje te same wiersze dwukrotnie i prawdopodobnie korzysta z tych samych indeksów także dwukrotnie. Drugi zbyteczny odczyt nie jest najprawdopodobniej związany z fizycznymi operacjami wejścia-wyjścia, ponieważ odczyt pierwszy, który miał miejsce milisekundy wcześniej, powinien bezpiecznie umieścić tabelę lub blok indeksu na początku współdzielonego bufora pamięci podręcznej. Jeśli plan wykonania jest wysoce filtrowany zanim osiągnie drugi, zbyteczny węzeł, nadmiarowość logicznych operacji wejścia-wyjścia może nie mieć znaczenia, jednak dla

dużych zapytań lub zapytań, które odfiltrowują większość wierszy dopiero po osiągnięciu drugiego węzła, dodatkowy koszt takich logicznych operacji wejścia-wyjścia może mieć znaczenie.

Jeśli programista oryginalnie pisał zapytanie do prostych tabel, ten rodzaj błędu raczej się nie pojawi — musiałby zejść z właściwej sobie ścieżki, aby uwzględnić nadmiarowe złączenie, a ta redundancja byłaby oczywista przy przeglądzie kodu. Jednak przy wykorzystaniu widoków, tego typu błędy są łatwe do popełnienia i dobrze ukryte.

Jak pozbyć się nadmiarowego złączenia z tabelą `Customers`? Mamy trzy możliwości:

- Możemy dodać nowe kolumny do listy `SELECT` zapytania definiującego widok i w zapytaniu wykorzystującym ten widok będziemy ich używać zamiast kolumn odnoszących się do zbytecznej tabeli. Takie rozwiązanie jest bezpieczne dla innych zapytań wykorzystujących ten sam widok, ponieważ nie zmienia ono diagramu zapytania definiującego widok.
- Wyeliminujemy zbyteczne złączenie z zapytania definiującego widok i w zapytaniu wykorzystującym widok będziemy używać jedynie kolumn z prostych tabel. Jednak w razie gdy istnieją inne zapytania wykorzystujące ten sam widok, mogą się one odnosić do usuniętych kolumn.
- Nie będziemy wykorzystywać widoku w zapytaniu, które go wykorzystywało, zamieniając go przez równoważne, nienadmiarowe złączenia z prostymi tabelami.

Niepotrzebne węzły i złączenia

Rozważmy złączenie z węzłem `OT` w ostatnim zapytaniu wykorzystującym widok. Oryginalne zapytanie definiujące widok uwzględnia to złączenie w celu uzyskania statusu zamówienia, ale zapytanie wykorzystujące widok nawet nie odnosi się do niego, więc można sobie zadać pytanie o konieczność tego węzła. Jeśli nawet nie zauważyliśmy takiego nieużywanego węzła, moglibyśmy go wykryć, jeśli zwrócilibyśmy uwagę na odwołanie do indeksu klucza głównego tej tabeli, ale bez odczytów samej tabeli. Takie odczyty wyłącznie indeksów klucza głównego wskazują zazwyczaj na niepotrzebne złączenie.

Usunięcie niepotrzebnych złączeń w bezpieczny sposób nie jest łatwe, gdyż mają one czasami pewne funkcjonalne efekty uboczne. Ponieważ to złączenie jest złączeniem wewnętrznym, jest bardzo możliwe, nawet przy braku filtra na tym węźle, że samo złączenie odrzuci wiersze, których zapytanie nie powinno zwrócić. Rezultatem może być usunięcie wierszy, dla których `Order.Status_Code IS NULL` lub dla których `Status_Code` wskazuje na niewłaściwe wartości kodów nie mających odpowiedników w tabeli `Code_Translations`. Ta ostatnia możliwość jest mało prawdopodobna lub powinna być usunięta przez naprawienie spójności odwołań. Trzeba zauważyć, że wartości `null` kluczy obcych są częste i jeśli kolumna może mieć wartość `null`, powinniśmy rozważyć jawne dodanie warunku `Status_Code IS NOT NULL` przed usunięciem złączenia, aby zasymulować ukryty efekt filtrowania złączenia wewnętrznego. Bardziej prawdopodobne jest, że programista używający tego widoku nie pomyślał nawet o niejawnej funkcji filtrowania widoku i ukryty

filtr był całkowicie niepożądany i niezgodny z intencją. Dlatego przed podjęciem decyzji o symulowaniu pierwotnego działania poprzez zapytanie oparte jedynie o tabele i eliminujące niepotrzebne złączenie, należy sprawdzić, czy to pierwotne działanie było w ogóle poprawne. Jeśli zamiana spowoduje nawet drobną zmianę zachowania zapytania, nawet na lepsze, należy ostrzec testerów, że testy regresyjne mogą w takich krańcowych przypadkach przynieść inne rezultaty.

Zapytania z operacjami na zbiorach

Niekiedy zachodzi potrzeba optymalizacji wieloczęściowego zapytania, które używa operatorów, takich jak `UNION`, `UNION ALL`, `INTERSECT` i `EXCEPT`, aby połączyć rezultaty dwóch lub większej liczby prostych zapytań. Rozszerzenie metody optymalizacji poprzez tworzenie diagramów SQL na takie wieloczęściowe zapytania jest zazwyczaj proste — należy stworzyć diagramy i optymalizować każdą część osobno, tak jakby była samodzielnym zapytaniem. Kiedy części te wykonują się szybko, połączenie rezultatów za pomocą operatorów zbiorów zazwyczaj działa dobrze.

Słowo kluczowe `EXCEPT` jest wyspecyfikowane przez standard ANSI SQL dla operatorów zbiorowych jako operacja wyszukania różnic pomiędzy zbiorami. DB2 i SQL Server podążają za standardem poprzez wspieranie `EXCEPT`. Jednak Oracle wykorzystuje dla tej samej operacji słowo `MINUS`, najprawdopodobniej dlatego, że wspierał on tę operację jeszcze zanim pojawiła się ona w standardzie.

Niektóre z tych operatorów zbiorów zasługują na chwilę uwagi. Operator `UNION` nie tylko łączy części wyniku, ale musi je także posortować i odrzucić duplikaty. Ta ostatnia czynność jest często niezbędna, szczególnie jeśli projekt uwzględnia podział na części głównie, aby odrzucić powtórzenia. W bazie danych Oracle operator `UNION` może być zamieniony przez `UNION ALL`, jeśli stwierdzimy, że powtórzenia nie są możliwe lub nie powinny być usunięte. W bazach danych nie wspierających `UNION ALL` można przeskoczyć eliminujące działanie przez zastąpienie pojedynczego zapytania `UNION` dwoma lub większą liczbą prostych zapytań i łącząc rezultaty w warstwie aplikacji zamiast w bazie danych.

Operacja `INTERSECT` może zazwyczaj być z powodzeniem zamieniona przez podzapytanie typu `EXISTS` poszukującego pasującego wiersza, który byłby wynikiem działania drugiej części zapytania z `INTERSECT`. Na przykład, jeśli mielibyśmy dwie tabele odnoszące się do pracowników, moglibyśmy poszukać rekordów pracownika współdzielonego w następujący sposób:

```
SELECT Employee_ID FROM Employees1
INTERSECT
SELECT Employee_ID FROM Employees2
```

Zawsze moglibyśmy zamienić zapytanie INTERSECT przez:

```
SELECT DISTINCT Employee_ID
FROM Employees E1
WHERE EXISTS (SELECT null
              FROM Employees2 E2
              WHERE E1.Employee_ID=E2.Employee_ID)
```

Używając metod zaprezentowanych w rozdziale „Zapytania z widokami" można określić, czy podzapytanie EXISTS powinno być wyrażone w formie EXISTS lub IN, czy też zamienione na proste złączenie. Zauważmy, że warunki złączenia skorelowanego stają się całkiem liczne, jeśli lista SELECT zawiera wiele elementów. Weźmy także pod uwagę, że INTERSECT dopasuje listy kolumn z wartościami null, podczas gdy złączenie skorelowane — nie, chyba że użyjemy warunku złączenia specjalnie do tego przeznaczonego. Przykładowo gdy nieujemny klucz główny Manager_ID może mieć wartość null (ale Employee_ID — nie) równoważna postać zapytania w wersji Oracle:

```
SELECT Employee_ID, Manager_ID FROM Employees1
INTERSECT
SELECT Employee_ID, Manager_ID FROM Employees2
```

jest następująca:

```
SELECT DISTINCT Employee_ID, Manager_ID
FROM Employees1 E1
WHERE EXISTS (SELECT null
              FROM Employees E2
              WHERE E1.Employee_ID=E2.Employee_ID
              AND NVL(E1.Manager_ID,-1)=NVL(E2.Manager_ID,-1))
```

Wyrażenie NVL(...,-1) w drugim warunku złączenia skorelowanego zamienia wartości null na kolumnie null-owej, tak aby złączyły się one poprzez dopasowanie null z null.

Operacja EXCEPT (lub MINUS) może zazwyczaj być zamieniona z zyskiem na podzapytanie NOT EXISTS. Gdybyśmy poszukiwali rekordów pracowników w pierwszej tabeli, pod warunkiem, że nie istniały w tabeli drugiej, moglibyśmy użyć poniższego zapytania:

```
SELECT Employee_ID, Manager_ID FROM Employees1
MINUS
SELECT Employee_ID, Manager_ID FROM Employees2
```

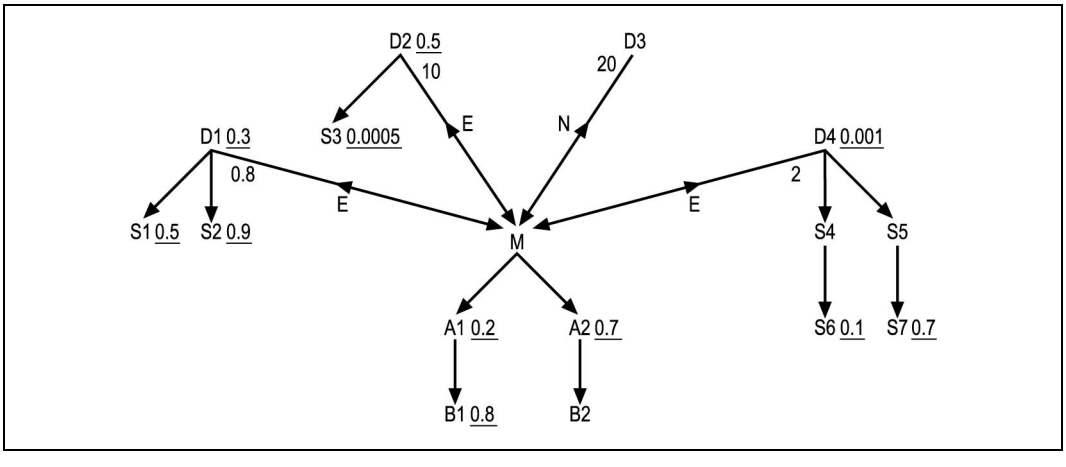
Zawsze moglibyśmy też zastąpić tę postać następującą:

```
SELECT DISTINCT Employee_ID
FROM Employees1 E1
WHERE NOT EXISTS (SELECT null
                  FROM Employees2 E2
                  WHERE E1.Employee_ID=E2.Employee_ID)
```

Rozwiązalibyśmy wówczas to zapytanie wykorzystując metody opisane we wcześniejszym rozdziale „Zapytania z podzapytaniami".

Ćwiczenie

Poniższy przykład przedstawiający nierealistycznie skomplikowane zapytanie przygotowano tak, aby Czytelnik mógł sprawdzić, czy dobrze zrozumiał zasady optymalizacji zapytań z podzapytaniami. Rysunek 7.36 obrazuje zapytanie bardziej skomplikowane i trudniejsze niż można by spotkać po roku intensywnej pracy nad optymalizacją SQL. Osoba, która poradzi sobie z tym zadaniem, poradzi sobie także z łatwością z dowolnym scenariuszem, które przyniesie życie.



Rysunek 7.36. Złożony problem z wieloma podzapytaniami

Jeśli ktoś nie zdoła rozwiązać tego zadania za pierwszym razem, powinien powrócić do tego ćwiczenia po nabyciu większej praktyki.

Na diagramie należy uzupełnić brakujące współczynniki dla złączeń skorelowanych. Załóżmy że $t = 5$ (liczba wierszy całego zapytania, uwzględniając podzapytanie `NOT EXISTS`) oraz $q = 50$ (liczba wierszy zapytania z usuniętym warunkiem `NOT EXISTS`). Następnie trzeba znaleźć najlepszy porządek złączenia biorąc pod uwagę wszystkie tabele podzapytań oraz zapytań zewnętrznych.