3. Podstawy SQL

Zanim zaczniesz korzystać z relacyjnych baz danych i prawdopodobnie jeszcze bardziej skomplikowanych rzeczy, musisz znać standardowy język używany do interakcji z nimi. Chociaż możesz obejść się bez znajomości języka SQL i pracy z bazami danych (pomyśl o programie Microsoft Access), prędzej czy później będziesz musiał się go nauczyć. W tej części zobaczymy wprowadzenie do SQL. Skoncentrujemy się tylko na podstawowych tematach, dzięki czemu nie zgubisz się w szczegółach, które nie są potrzebne w pierwszych etapach Twojego projektu. Ci, którzy mają już przyzwoite umiejętności SQL, mogą przejść do następnej części, ale nawet w takim przypadku zdecydowanie zalecamy zrobienie krótkiego przeglądu.

Co to jest SQL?

SQL oznacza ustrukturyzowany język zapytań. Jak sama nazwa wskazuje, jest to język używany do wysyłania zapytań do bazy danych, ale oprócz wykonywania zapytań ma wiele innych funkcji, co zobaczymy w tej części. W tej części zobaczymy wprowadzenie do SQL. Skoncentrujemy się tylko na podstawowych tematach, dzięki czemu nie zgubisz się w szczegółach, które nie są potrzebne w pierwszych etapach Twojego projektu. Jedną rzeczą, którą musimy wiedzieć o SQL, jest to, że nie jest on podobny do innych języków programowania, takich jak Java, C# i Python. Dzieje się tak, ponieważ jest to język deklaratywny, a nie proceduralny. Oznacza to, że zamiast opisywać obliczenia, które należy wykonać, aby uzyskać to, czego chcesz, po prostu określasz, czego chcesz, bez określania, jak to uzyskać. SQL został pierwotnie stworzony z myślą o modelu relacyjnym. Jednak późniejsze dodatki dodały funkcje, które nie są częścią modelu relacyjnego, więc w razie potrzeby możesz robić rzeczy w inny sposób. Wiele aktualnych wersji baz danych obsługuje zarówno metody relacyjne, jak i obiektowo-relacyjne, które obejmują obsługę programowania obiektowego i bardziej złożonych typów danych niż zwykłe podstawowe. Wkrótce zobaczymy, czym jest tryb relacyjny i jakie są jego implikacje; nie martw się. Tam, gdzie może się to wydawać nieco ograniczające i prawdopodobnie masz rację, ponieważ sam SQL prawdopodobnie nie wystarczy do wykonania wszystkich obliczeń, o których myślisz, zwłaszcza jeśli myślisz o aplikacjach dla dużych przedsiębiorstw, jest łatwy w użyciu w połączeniu z innymi językami , zwłaszcza te proceduralne. Prawie każdy język programowania ma jakieś złącze specyficzne dla bazy danych do interakcji lub używa jakiegoś rodzaju sterownika ODBC, który pozwala łączyć się z relacyjną bazą danych i wykonywać zapytania. Zróbmy krótkie wprowadzenie do historii SQL. SQL pojawił się po raz pierwszy w 1970 r.1, ale najnowsza wersja pochodzi z 2011 r. Nie wszystkie bazy danych mają tę samą wersję; nawet nie wszystkie z nich implementują całą wersję, ale czasami tylko jej podzbiór. Czasami powoduje to problemy podczas próby przeniesienia kodu, który działa dla określonej wersji bazy danych, do kodu pochodzącego od innego dostawcy.

Model relacyjny

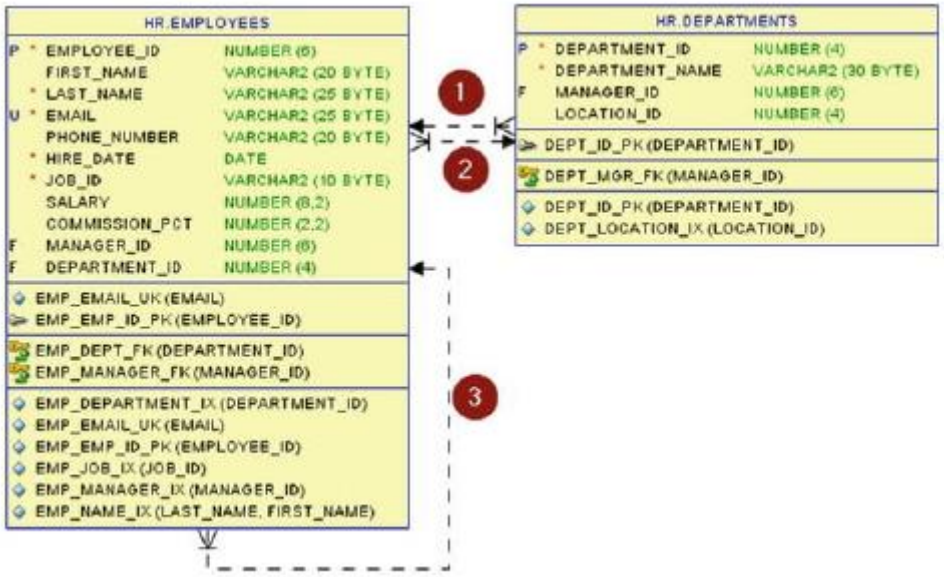
SQL jest standardowym językiem używanym do interakcji z relacyjnymi bazami danych, ale nie implementuje wszystkich koncepcji modelu relacyjnego. Po wielu latach nie ma jeszcze w pełni udanej implementacji modelu relacyjnego. Model relacyjny ma zasadniczo trzy koncepcje:

* Tabele, które służą do przedstawiania informacji oraz przechowywania kolumn i wierszy, których celem jest reprezentowanie atrybutów i rekordów (pomyśl o arkuszu kalkulacyjnym Excel).

* Klucze, aby skutecznie zarządzać i wdrażać relacje oraz jednoznacznie identyfikować rekordy w tabeli (więcej później, gdy omawiamy klucze podstawowe i obce).

* Operacje, które wykorzystują model i mogą generować inne relacje i obliczenia spełniające zapytania.

Aby pokazać przykład dotyczący tych pojęć, weźmy próbkę relacyjnej bazy danych; równie dobrze mogą to być typowe tabele Pracowników i Działów, wyjaśniające nieco niektóre z tych pojęć przy użyciu dwóch tabel ze schematu Oracle HR. Procedura ich instalacji polega zasadniczo na klonowaniu repozytorium github i uruchomieniu kilku poleceń, które uruchomią skrypt. Pełne instrukcje masz na stronie powitalnej repozytorium. Jednakże, jeśli chcesz przetestować, możesz także zdecydować się na pobranie bazy danych Oracle Enterprise Edition ze strony internetowej Oracle, która jest bezpłatna według strony internetowej właściciela, gdy tylko użycie ma na celu „opracowywanie, testowanie, prototypowanie i demonstrowanie twojej aplikacji, a nie w jakimkolwiek innym celu."



Na rysunku możesz zauważyć, że mamy dwie tabele. Pierwsza część wylicza kolumny tabel i ich typy danych (zobaczymy je za chwilę). Dla działów widzimy, że mamy kolumnę o nazwie DEPARTMENT_ID, składającą się z liczby 4 cyfr. „P" z przodu oznacza, że ta kolumna jest kluczem podstawowym. Następnie mamy kolumnę DEPARTMENT_NAME, która jest nazwą działu; MANAGER_ID to kolumna identyfikująca identyfikator pracownika (w tabeli Pracownicy), który jest kierownikiem działu, który jest kluczem obcym łączącym tabelę działów z tabelą pracowników i wymusza, aby kierownik działu musi być prawidłowym pracownikiem znalezionym w tabeli Pracownicy; oraz LOCATION_ID, który jest identyfikatorem lokalizacji działu (jak możesz pomyśleć, że może to być prawdopodobnie klucz obcy do „brakującej" tabeli lokalizacji. Tak, masz rację!). Kolumny, które są używane w relacjach z innymi tabelami lub identyfikują wartości tabeli, tutaj DEPARTMENT_ID i MANAGER_ID są zwykle przedstawiane osobno na diagramie, w postaci kluczy z nazwami ograniczeń, przedstawiającymi nazwę kolumny, do której się odnoszą w nawiasy. Również wszelkie indeksy znalezione w tabeli potrzebne do przyspieszenia zapytań lub zapewnienia prawidłowej implementacji relacji są również przedstawione w ostatniej części definicji tabeli. Są jeszcze inne ważne rzeczy, których można się nauczyć z tej liczby. Są to kropkowane linie między tabelami, które definiują relacje. To są:

1. DEPT_MGR_FK, czyli w zasadzie relacja, która mówi nam, że pracownik o numerze MANAGER_ID jest jedynym managerem z Tabeli Pracownicy. Ta relacja jest kluczem obcym i zwykle jest zgodna z konwencją nazywania jej z zakończeniem „_FK".

2. EMP_DEPT_FK, który jest w przeciwnym kierunku, mówi nam, że DEPARTMENT_ID znaleziony w tabeli Pracownicy jest jednym i jedynym DEPATMENT_ID znalezionym w tabeli departamentów.

3. EMP_MANAGER_FK. Jest to jak dotąd najbardziej skomplikowane do zrozumienia, ale definiuje relację menedżera, tak że KIEROWNIK pracownika musi być jednocześnie członkiem tabeli PRACOWNIK, co uniemożliwia przypisanie menedżera, który nie jest częścią tabeli PRACOWNIK przy stole, czy w naszym świecie domenowym, pracownik spoza firmy.

To jest zrozumiałe, czyli w zasadzie wszystko, co musimy wiedzieć o modelu relacyjnym. Oczywiście temat jest znacznie szerszy, ale dla naszych celów zaufaj nam, że to w zasadzie wszystko, co musisz wiedzieć.

Bazy danych i dostawcy baz danych

Istnieje wiele typów baz danych: relacyjne, obiektowe, bez SQL… ale generalnie hurtownie danych zostały utworzone z relacyjnymi bazami danych. Wynika to z faktu, że łatwo jest nauczyć się języka SQL i przeprowadzać z nim analizy. W dzisiejszych czasach wiele systemów, zwłaszcza tych, które zajmują się dużymi ilościami danych, ma tendencję do korzystania z innego typu baz danych, baz danych NoSQL, które są inne i zwykle używają pary klucz/wartość jako sposobu przechowywania danych. Podczas gdy te bazy danych mogą być bezschematowe i bardziej praktyczne, nie przestrzegają zasad ACID dla transakcji, których przestrzega większość relacyjnych baz danych. Jest to kompromis, który trzeba zapłacić, aby zyskać prędkość i móc zapewnić lepszą wydajność podczas pracy z dużą ilością danych oraz móc działać w sposób rozproszony.

Uwaga: obecnie NoSQL i specjalnie dystrybuowane bazy danych są popularnym tematem wraz z nadejściem BigData i faktycznie używają innych języków niż ich relacyjne odpowiedniki. Ale nawet przy tym znajdziesz wiele podobieństw między nimi a językiem SQL. Tak więc, pomimo tego, że myślisz o zaimplementowaniu bazy danych NOSQL do swojego projektu lub innego typu bazy danych, ten rozdział powinien być wart przeczytania.

Głównymi dostawcami baz danych jest firma Oracle, która twierdzi, że ma udział w rynku wynoszący 98,5% firm z listy Fortune 500. Choć może się to wydawać zdumiewające, należy podkreślić, że zazwyczaj duże firmy mają różnych dostawców baz danych. Prawdopodobnie łatwo jest zobaczyć duże firmy posiadające Oracle, SQL Server, a nawet MySQL, Postgree i bazy danych typu open source, takie jak MariaDB i wiele innych. DB2 firmy IBM i SQL Server firmy Microsoft to również ważne komercyjne relacyjne bazy danych, na które należy zwrócić uwagę. Znajdziesz je w wielu firmach choć pewnie nie są tak rozbudowane jak Oracle. Warto zauważyć, że wszystkie komercyjne bazy danych mają zwykle jakąś zmniejszoną lub ograniczoną wersję do bezpłatnego użytku, a nawet z pozwoleniem na dystrybucję udzielonym przez ich własną firmę. Wersje te mają zwykle pewne ograniczenia, ale jako mała lub średnia firma nie powinno to stanowić dla Ciebie problemu i zawsze możesz przejść na wersję płatną, jeśli Twoja firma się rozwinie. Jeśli odejdziesz od komercyjnych baz danych, Oracle ma inny RDBMS, który jest open source, o nazwie MySQL. Chociaż MySQL został zakupiony przez Oracle, nadal możesz go pobrać i używać za darmo. Tę bazę danych zobaczysz głównie w wielu projektach internetowych, zwłaszcza, że łatwiej jest ją zintegrować z językami internetowymi, takimi jak php i tym podobne. Istnieją inne bardzo interesujące bazy danych typu open source. Jednym z nich jest postgresql, który w dzisiejszych czasach zyskuje coraz większą popularność. Kolejnym jest rozwidlenie MySQL, zwane MariaDB, które było kontynuowane przez głównych ludzi, którzy stworzyli MySQL i których projekt rozpoczął się, gdy Oracle kupił oryginalną bazę danych. Jedną z najlepszych rzeczy MariaDB jest to, że jest w 100% kompatybilna z MySQL. Od kilku lat do MariaDB dodano różne i nowe funkcje, które odbiegają od oryginalnego MySQL. Te zwykle nie mają ograniczeń, tylko te wynikające z ograniczeń implementacji i architektury. Wszystkie te bazy danych mogą współpracować z wieloma systemami operacyjnymi, w tym Linux i Windows w głównych częściach ich odpowiednich wersji.

Uwaga: Wybierając bazę danych do realizacji swojego projektu, nie kieruj się wyłącznie pieniędzmi, ale również umiejętnościami swoich pracowników. Oracle jest bardzo dobry i wersja Express może ci się przydać, ale jeśli z jakiegoś powodu po jakimś czasie będziesz musiał przejść na edycje płatne, licencje nie są tanie. Migracja bazy danych od jednego dostawcy do innego może być uciążliwa, jeśli Twój system jest duży lub Twoje aplikacje korzystają z określonych funkcji jednego dostawcy. W takich przypadkach możesz rozważyć skorzystanie z rozwiązania w chmurze z płatną bazą danych, w której cena może być kontrolowana. Bazy danych typu open source są również bardzo dobrą alternatywą.

Zgodność z ACID

Relacyjne bazy danych zwykle obsługują tak zwany zestaw właściwości ACID, jeśli chodzi o transakcje. Baza danych zgodna z ACID oznacza, że gwarantuje:

* Niepodzielność, co oznacza, że instrukcje mogą grupować się w transakcje, co oznacza, że możesz kontrolować, że jeśli jedna się nie powiedzie, transakcja się nie powiedzie, co oznacza, że żadne zmiany nie zostaną zapisane w bazie danych.

* Spójność, co oznacza, że baza danych jest zawsze w prawidłowym stanie, nawet po awarii lub nieprzestrzeganiu ograniczeń. Zwykle najłatwiejszym sposobem myślenia o tym jest tabela z ograniczeniem, które sprawdza, czy żaden pracownik nie może mieć pensji większej niż 10 000 USD. Jeśli zaczniesz wprowadzać pracowników i wynagrodzenia do tej tabeli jako część transakcji, jeśli dodasz jedną z pensją 20 000 USD, cała transakcja powinna zakończyć się niepowodzeniem, a dane tabeli powinny zostać przywrócone tak, jak przed rozpoczęciem tej transakcji. To samo dotyczy przypadku niepowodzenia podczas realizacji transakcji. Albo transakcja została zatwierdzona i utrwalona, lub jeśli nie została jeszcze ukończona, zostanie wycofana.

* Izolacja, co oznacza, że za każdym razem, gdy wysyłasz zapytanie do bazy danych, pobierane jest to, co było przed rozpoczęciem jakiejkolwiek uruchomionej transakcji lub po zakończeniu dowolnej uruchomionej transakcji. Pomyśl na przykład, że masz duży stół i wysłałeś do bazy danych instrukcję aktualizacji wynagrodzeń pracowników, aby ustalić dwukrotność ich wynagrodzenia. Jeśli transakcja nie została jeszcze zakończona podczas zapytania do tabeli, powinna zostać wyświetlona poprzednia wersja tabeli, w której tabela nie została jeszcze zaktualizowana. Po zakończeniu transakcji wszelkie kolejne zapytania do tabeli powinna zostać wyświetlona zaktualizowana tabela, w której pracownicy mają podwójną pensję, ale generalnie nie jest pożądane uzyskiwanie miksu, ponieważ prowadzi to do błędów.

* Trwałość oznacza, że raz zatwierdzona transakcja jest zatwierdzona na zawsze, nawet w przypadku niepowodzenia. Tak więc baza danych musi zapewnić, dzięki różnym mechanizmom, że zmiany są zawsze zapisywane na dysku, albo poprzez bezpośrednie przechowywanie danych (zwykle nieefektywne), albo przez przechowywanie pliku wektorowego definiującego zmiany zastosowane do danych, aby baza danych po przywróceniu , jest w stanie odtworzyć te zmiany.

Rodzaje instrukcji SQL

W tym momencie prawdopodobnie już zastanawiasz się, co zrobisz, aby sprawdzić sumę pieniędzy zafakturowanych dla konkretnego klienta lub określonego regionu. To dobrze, a wkrótce nauczysz się, jak to robić. Jednak podczas pracy z relacyjną bazą danych istnieją inne stwierdzenia, być może równie ważne jak zapytania. Oprócz wysyłania zapytań do bazy danych, istnieją inne rodzaje instrukcji, które są zwykle używane przed zapytaniem, aby przygotować bazę danych do przechowywania danych, które będą później wyszukiwane. Zróbmy szybkie podsumowanie i przejrzyjmy je:

* Instrukcje DML lub Data Manipulation Language. Należy zauważyć, że ta grupa zawiera instrukcje Select do wykonywania zapytań w tabelach, ale nie ogranicza się tylko do zapytań, ponieważ istnieją inne zestawy instrukcji, które należą do tej grupy, takie jak INSERT, używane do wstawiania rekordów w tabeli; UPDATE do aktualizacji tych zapisów; i DELETE, aby usunąć rekordy na podstawie jednego lub wielu warunków. W wielu miejscach zobaczysz komunikat MERGE. Jest to specyficzna instrukcja, która nie została zaimplementowana we wszystkich bazach danych i jest mieszanką INSERT + UPDATE. Czasami jest to również nazywane i UPSERT z tego powodu.

* Instrukcje DDL lub Data Definition Language, które obejmują instrukcje używane do modyfikowania struktury tabeli, tworzenia ich lub usuwania. Należą do nich instrukcja CREATE do tworzenia obiektów, takich jak tabele i indeksy, ale także wiele innych, instrukcja DROP do ich usuwania, ALTER do ich modyfikowania oraz specjalna instrukcja o nazwie TRUNCATE używana do usuwania wszystkich danych z tabeli.

* Wyciągi transakcji, takie jak COMMIT, aby zatwierdzić wszystkie zmiany dokonane w ramach transakcji, ROLLBACK, aby cofnąć wszystkie wprowadzone zmiany. Należy zauważyć, że jeśli polecenie COMMIT nie zostało wykonane na końcu transakcji, po rozłączeniu, ze względu na mechanizmy spójności zaimplementowane przez prawie wszystkie bazy danych, zmiany zostaną utracone i nie zostaną utrwalone w plikach bazy danych. Niektóre bazy danych implementują również polecenie CHECKPOINT, które wyznacza punkty kontrolne w bazie danych, ale obecnie nie jest to dla nas zbyt przydatne.

Uwaga: Należy zachować ostrożność w przypadku opcji automatycznego zatwierdzania, która jest domyślnie włączona w niektórych relacyjnych bazach danych, takich jak MySQL i SQL Server. Może to być dobre, jeśli zapomnisz zatwierdzić transakcję na koniec transakcji, ale może mieć straszne konsekwencje, jeśli zepsujesz ją instrukcją aktualizacji lub usunięcia. Naszą sugestią jest wyłączenie automatycznego zatwierdzania i pamiętaj, aby zawsze zatwierdzać. Instrukcja TRUNCATE, ponieważ wszystkie instrukcje DDL mają niejawne polecenie COMMIT. Jeśli przez pomyłkę obetniesz tabelę, nie ma możliwości jej odzyskania, chyba że przywrócisz bazę danych do poprzedniego punktu kontrolnego. Podczas gdy w przypadku niektórych baz danych możliwe jest nawet odzyskanie aktualizacji lub usunięcia wprowadzonych przez pomyłkę, nie jest to możliwe w przypadku obciętych baz danych. Bądź bardzo ostrożny podczas uruchamiania instrukcji TRUNCATE!

Typy danych SQL

Język SQL definiuje zestaw typów danych dla wartości kolumn. Nie wszystkie bazy danych implementują te same typy danych i nie wszystkie bazy danych używają tej samej nazwy dla zgodnych typów danych. Większość baz danych obsługuje jednak najpopularniejsze typy danych. W kolejnych podrozdziałach dokonujemy ich przeglądu, wyjaśniamy, kiedy ich używać i analizujemy ich specyfikę. Ważne jest, aby użyć poprawnego typu danych, ponieważ wybranie nieprawidłowego typu, oprócz marnowania miejsca, może prowadzić do niskiej wydajności z powodu jawnych lub nawet niejawnych konwersji typów.

Liczbowe typy danych

Istnieje wiele liczbowych typów danych. W większości baz danych są one reprezentowane przez typy Integer, Float, Real, Decimal i BigInt. Istnieją inne typy danych, takie jak Smallint dla małych liczb. Wszystkie one są używane do przechowywania liczb i musisz wybrać, która jest bardziej odpowiednia

dla każdej kolumny tabeli, zasadniczo biorąc pod uwagę maksymalną wartość, jaką można przejść, oraz czy potrzebne są pozycje dziesiętne i do jakiego stopnia precyzja jest dla Ciebie ważna.

Tekstowe typy danych

Tekst jest zwykle reprezentowany przez typy danych Char (stała długość) lub nowsze typy danych Varchar (zmienna długość). Typy danych String i CLOB (Character LOB) są również dostępne w innych bazach danych. Zalecamy trzymanie się typu danych Varchar, chyba że istnieje bardzo ważny powód, aby używać innych typów danych, ponieważ zazwyczaj nowe bazy danych nie marnują miejsca na przechowywanie mniejszej wartości w kolumnie varchar zdefiniowanej do przechowywania większej liczby znaków.

Typy danych daty

Wszystkie wartości daty, godziny i znacznika czasu należą do tej kategorii. Prawdopodobnie tutaj pojawia się najwięcej różnic w stosunku do różnych dostawców baz danych. Ważne jest, aby sprawdzić dokumentację swojej bazy danych, aby upewnić się, że używasz właściwego typu. Większość baz danych obsługuje również znaczniki czasu ze strefami czasowymi, więc jeśli planujesz obsługiwać różne regiony w swojej aplikacji i chcesz korzystać z tych różnych stref czasowych, przed wybraniem typu danych zapoznaj się z instrukcją.

Inne typy danych

W tym obszarze możemy znaleźć inne ważne typy danych, takie jak typ danych binarny, bool lub bitowy używany do przechowywania informacji binarnych (prawda lub fałsz); typy danych LOB lub RAW używane do przechowywania dużych informacji, takich jak duże fragmenty tekstów, obrazów lub plików w formacie binarnym; typy XML do przechowywania informacji XML w obsługujących je bazach danych, dzięki czemu można wydajnie wyszukiwać dane; typy zdefiniowane przez użytkownika poprzez łączenie typów podstawowych; oraz wiele innych typów danych, które nie mieszczą się w żadnej z poprzednich kategorii.

Pobieranie danych z tabeli

Kilka akapitów temu przedstawiliśmy tabele Pracownicy i Działy. Wykorzystamy je w tej części do wykonania kilku instrukcji i sprawdzenia oczekiwanych rezultatów. Zaczniemy od najbardziej podstawowej operacji, jaką można wykonać na tabeli, za pomocą instrukcji SELECT. Najpierw chcemy pokazać przegląd wybranej instrukcji. W kolejnych podrozdziałach przyjrzymy się każdemu blokowi po kolei i omówimy jego zastosowanie. Wszystkie typowe instrukcje wyboru SQL mają następującą postać:

SELECT list_of_columns (separated by commas, or *)

FROM list_of_tables (separated by commas)

[WHERE set_of_conditions (separated by operators)]

[GROUP BY grouping_of_columns (separated by

commas)]

[HAVING

set_of_conditions_applied_to_the_grouping_of_columns

(separated by operators)]

[ORDER BY ordering_conditions (separated by

commas)] ;

Jeśli to jest wystarczająco jasne, możemy zacząć od uzyskania wszystkich informacji z tabeli za pomocą specjalnej instrukcji SELECT *, a następnie zobaczymy inne sposoby użycia opcji z instrukcją Select i użyjemy projekcji, aby wybrać tylko określone kolumny z bazy danych.v

Instrukcja *SELECT

Składnia instrukcji select jest bardzo prosta. Zwykle musimy określić, z jakich kolumn chcemy pobrać informacje i z której tabeli lub tabel, a następnie opcjonalny predykat w klauzuli where, aby zastosować dowolny filtr w danych, których chcemy użyć, aby wiersze, które nie pasują do tych warunki zostaną odfiltrowane z wyniku. Typowa instrukcja select wygląda następująco:

SELECT

column1, column2, columnn

FROM

schema.table1

WHERE

column1 = 'Albert';

Wybrane kolumny są częścią klauzuli projekcji. Kolumny w klauzuli where są filtrami lub kolumnami używanymi do łączenia z innymi tabelami. Zobaczmy najpierw najbardziej podstawowe stwierdzenie:

```
   SELECT
         *
   FROM
         hr.departments;

   DEPARTMENT_ID
DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
   -------------- ------------------------------ ------
---- -----------
             10
Administration                      200      . 1700
             20
Marketing                           201        1800
             30
Purchasing                          114        1700
             40 Human
Resources                      203        2400
             50
Shipping                            121        1500
             60
IT                                  103        1400
   ... (output truncated)
   27 rows selected
```

Za pomocą tej instrukcji select * mówimy bazie danych, aby pobierała wszystkie kolumny i wszystkie dane z tabeli departamentów bez stosowania żadnego filtra. Ponieważ pobieramy wszystkie dane, zostanie wykonane PEŁNE SKANOWANIE tabeli, co oznacza, że cała tabela zostanie odczytana przez bazę danych i zwrócona. Żaden indeks, bez względu na to, czy istniał, nie zostanie użyty do odzyskania tego, chyba że zdarzy się nietypowa sytuacja, w której wszystkie kolumny tabeli będą znajdować się w indeksie (więcej na ten temat w dalszej części rozdziału dotyczącego wydajności).

Instrukcja select column

Instrukcja select column jest szczególnym przypadkiem instrukcji select *. W takim przypadku używasz predykatu projekcji, aby wybrać tylko te kolumny, które Cię interesują. Jeśli to możliwe, używaj go zamiast select *, ponieważ ten ostatni jest bardzo niebezpieczny. Pomyśl, co może się stać, jeśli dodasz kolumnę do tabeli. Teraz wybór zamiast zwracania n kolumn zwróci n+1. To samo, jeśli usuniesz niektóre. Na przestrzeni lat widzieliśmy wiele błędów w procesach polegających na używaniu gwiazdki zamiast nazywania kolumn. Również z punktu widzenia wydajności użycie select * nie jest zalecanym rozwiązaniem, ponieważ uniemożliwisz optymalizatorowi użycie indeksów, jeśli są dostępne, ale zostanie to omówione później. Typowa instrukcja select służąca do pobierania nazw działów z tabeli departamentów jest następująca:

SELECT

department_name

FROM

hr.departments;

DEPARTMENT_NAME

Administration

Marketing

Purchasing

Human Resources

Shipping

IT

..... (output truncated)

27 rows selected

Instrukcja select count (*) lub count (column).

Podczas opracowywania możemy być zainteresowani łatwym policzeniem liczby wierszy zwracanych przez zapytanie. Można to osiągnąć za pomocą instrukcji select count. Jeśli użyjesz gwiazdki, policzy wszystkie wiersze z tabeli, ale z kolumną pominie wartości NULL, czyli wartości, które nie istnieją dla tego konkretnego wiersza. Zdania, w przypadku gdy istnieją wszystkie wartości, powinny dawać ten sam wynik, ale różne bazy danych implementują się wewnętrznie inaczej, a niektóre z nich mogą mieć lepszą wydajność niż inne.

SELECT

COUNT(department_name)

FROM

hr.departments;

COUNT(DEPARTMENT_NAME)

27

Whereas:

SELECT

COUNT(*)

FROM

hr.departments;

COUNT(*)

27

Otrzymujemy więc dokładnie ten sam wynik.

Wybierz odrębną klauzulę

Gdy nie chcemy brać pod uwagę odrębnych wartości dla określonego wyboru, możemy użyć klauzuli odrębnej. Spowoduje to pobranie tylko jednego na parę kolumn. Jeśli wybierzemy tylko jedną kolumnę, pobierzemy tylko różne wartości dla tej kolumny; ale jeśli zapytanie składa się z projekcji kilku kolumn, zwróci różne krotki, co oznacza, że zwróci różne wartości, ale biorąc pod uwagę wszystkie wybrane kolumny. Oto przykład:

SELECT

COUNT(first_name)

FROM

hr.employees;

FIRST_NAME

107

But, for example:

SELECT

COUNT(distinct first_name)

FROM

hr.employees;

COUNT(DISTINCTFIRST_NAME)

91

Jeśli jednak zapytamy parę imię, nazwisko, zobaczymy, że nie ma pracowników o dokładnie takim samym nazwisku (imię + nazwisko są takie same):

```
SELECT DISTINCT

first_name, last_name

FROM

hr.employees;

FIRST_NAME LAST_NAME

-------------------- -------------------------

Ellen Abel

Sundar Ande

Mozhe Atkinson

... (output truncated)

107 rows selected
```

Sortowanie

Czasami chcesz, aby wynik zapytania był sortowany według określonej kolumny. Sortowanie może odbywać się rosnąco lub malejąco. W SQL osiąga się to za pomocą słów kluczowych ORDER BY. Klauzula ORDER BY zawsze znajduje się na końcu i sortuje wszystkie rekordy, które zostały wybrane i nie zostały odfiltrowane. Rzućmy okiem na kilka przykładów: Jeśli nie określono, domyślnie kolejność sortowania jest rosnąca:

```
SELECT

department_name

FROM

hr.departments

ORDER BY

department_name;

DEPARTMENT_NAME

------------------------------

Accounting

Administration

Benefits

Construction

... (output truncated)

27 rows selected
```

Uwaga: Domyślnie sortowanie jest numeryczne w przypadku pól liczbowych, alfabetyczne w przypadku pól tekstowych i chronologiczne podczas sortowania pola daty. Ale możemy określić kierunek sortowania:

```
SELECT
department_name
FROM
hr.departments
ORDER BY
department_name desc;
```

```
DEPARTMENT_NAME
------------------------------
Treasury
Shipping
Shareholder Services
Sales
... (output truncated)
27 rows selected
```

Możemy sortować wiersze według wielu kolumn jednocześnie, więc pierwsza kolumna w klauzuli zostanie posortowana, potem druga i tak dalej:

```
SELECT
first_name, last_name
FROM
hr.employees
ORDER BY
first_name, last_name;
```

```
FIRST_NAME LAST_NAME
-------------------- -------------------------
Adam Fripp
Alana Walsh
Alberto Errazuriz
Alexander Hunold
Alexander Khoo
```

... (output truncated)

107 rows selected

Możemy nawet zdecydować się na sortowanie niektórych kolumn rosnąco, a innych malejąco:

```
SELECT
first_name, last_name
FROM
hr.employees
ORDER BY
first_name asc, last_name DESC;
```

```
FIRST_NAME LAST_NAME
-------------------- -------------------------
Adam Fripp
Alana Walsh
Alberto Errazuriz
Alexander Khoo
Alexander Hunold
... (output truncated)
107 rows selected
```

Czasami jednak łatwiej jest po prostu określić pozycję kolumny w predykacie projekcji, więc poniższy przykład będzie podobny do poprzedniego przykładu:

```
SELECT
first_name, last_name
FROM
hr.employees
ORDER BY 1 ASC, 2 DESC;
```

```
FIRST_NAME LAST_NAME
-------------------- -------------------------
Adam Fripp
Alana Walsh
Alberto Errazuriz
Alexander Khoo
```

Alexander Hunold

... (output truncated)

107 rows selected

Jeśli mamy w kolumnach jakieś wartości null i chcemy je przenieść na początek lub na koniec, możemy użyć NULLS LAST:

SELECT

First_name, last_name, commission_pct

FROM

hr.employees

ORDER BY

3 DESC NULLS LAST,1,2;

FIRST_NAME LAST_NAME COMM

ISSION_PCT

-------------------- ------------------------- ---- ----------

John Russell

,4

Allan McEwen

,35

Janette King

,35

Patrick Sully

,35

Alberto Errazuriz

,3

... (output truncated)

Vance Jones

William Gietz

Winston Taylor

107 rows selected

I to na razie wszystko, co musimy wiedzieć o sortowaniu.

Uwaga: Oprócz porządkowania, relacyjne bazy danych zwykle implementują pewne predykaty, aby wybrać górne X wierszy. Może to być bardzo przydatne w środowisku hurtowni danych, jeśli chcesz

wyświetlić 5 największych klientów lub 10 regionów z największą sprzedażą. W zależności od bazy danych jest to realizowane jako klauzule TOP, LIMIT, FETCH FIRST X ROWS ONLY; ponieważ nie ma jasnego standardu, nie będziemy go tutaj omawiać, ale zachęcamy do sprawdzenia instrukcji dostawcy, aby uzyskać więcej informacji.

Filtracja

Do tej pory widzieliśmy kilka podstawowych zapytań, ale nie są one jeszcze zbyt przydatne. Po prostu wybieramy wszystkie rekordy z tabeli. Ale zwykle chcesz wybrać tylko niektóre rekordy, które spełniają warunek. Obliczenie tego warunku lub zestawu warunków może być czasem skomplikowane. Przez większość czasu będziesz pytać o swoje dane, ile wystawiłem faktury za dany region? Kim są pracownicy pracujący w części łańcucha dostaw firmy? Jaki procent zysku mam dla określonej grupy produktów? Wszystkie te pytania wymagają pewnego rodzaju filtrowania, ponieważ kierujesz obliczenia do określonej grupy.

Klauzula Where

Predykatem do filtrowania wierszy, które zostaną uwzględnione w wyniku, jest klauzula WHERE. Istnieje jednak ogromny zestaw operatorów, z których można korzystać. Przejrzyjmy je. Jeśli pamiętasz z poprzednich przykładów, mieliśmy w tabeli 107 pracowników, zobaczmy, jak obliczyć pracowników, którzy zarabiają więcej niż 10 000 USD za pomocą zapytania. Zapytanie będzie wyglądać następująco, wykorzystując to, czego się do tej pory nauczyliśmy:

SELECT

count(*)

FROM

hr.employees

WHERE

salary>10000;

15 rows selected

COUNT(*)

15

Możemy więc teraz wywnioskować, że mamy 15 pracowników na 107, którzy zarabiają więcej niż $10,000$.

Operatory

W naszym przykładzie użyliśmy operatora większego niż, aby porównać wartość wynagrodzenia w tabeli z określoną liczbą. Ale jest wielu innych operatorów. Oto lista najczęściej używanych:

Operator : Znaczenie : Przykład

= : Równe wynagrodzenie : = 10000

< : Mniej niż pensja : < 10000

<= : Mniejsze lub równe Wynagrodzeniu : <=10000

> : Większa niż pensja : > 10000

>= : Większe lub równe Wynagrodzeniu : >= 10000

<> : Inaczej niż Wynagrodzenie : <> 10000

IN () : Jedna z wartości na liście : Wynagrodzenie w (10000, 11000, 12000)

NOT IN () : Żadna z wartości na liście : Wynagrodzenie nie w (10000, 11000,12000)

BETWEEN x i y : Pomiędzy tym zakresem : Wynagrodzenie między 10000 i 11000

LIKE : Dopasowuje częściowe słowo (tekst). Pobiera wszystkich pracowników, których imię to Pete, Peter lub PeteXXX. Symbol wieloznaczny % będzie pasował do dowolnego znaku (znaków): Imię, na przykład „Pete%"

NOT LIKE : Wyklucz częściowe dopasowania: Imię inne niż „Pete%"

IS NULL : Określona kolumna ma wartość NULL : Wynagrodzenie ma wartość NULL

IS NOT NULL : Określona kolumna ma wartość : Wynagrodzenie nie jest puste

Uwaga: pamiętaj, że jeśli porównujesz z ciągiem tekstowym lub znakiem, musisz użyć pojedynczych cudzysłowów, aby ująć ciąg lub znak, z którym porównujesz. Poniższe porównanie nie jest poprawnym porównaniem: gdzie imię = Piotr. Właściwym sposobem jest ujęcie tekstu w pojedyncze cudzysłowy: gdzie imię = „Piotr".

Oprócz nich istnieje kilka innych operatorów, które są podobne do operatora IN(). Są to operatory, które porównują grupę wierszy: ANY, ALL, EXISTS, a nazwy są oczywiste. W podrozdziale dotyczącym podzapytań zobaczymy pewne zastosowanie.

Operatory logiczne

Istnieje inny zestaw operatorów zwanych operatorami logicznymi. Ten zestaw operatorów jest szeroko stosowany, ponieważ działają one jak klej między różnymi warunkami lub innymi operatorami. Dla czytelników przyzwyczajonych do programowania w dowolnym języku wyniki te będą bardzo znajome

Operator : Znaczenie: Przykład

AND: Oba warunki muszą być spełnione, aby zwróciły wartość true: Wynagrodzenie > 10000 AND imię, takie jak „Pete%"

OR: Przynajmniej jeden z warunków musi być spełniony, aby zwrócić wartość true:

Wynagrodzenie > 10000 OR imię jak „Pete%"

NOT: Warunek musi być fałszywy, aby zwrócił prawdę: NOTNIE (wynagrodzenie > 10000)

Jeśli pamiętasz ze szkoły, musisz znać pierwszeństwo operatorów. Oznacza to, że czasami musimy użyć nawiasów, aby określić, który operator występuje jako pierwszy. Na przykład w operatorze NOT określiliśmy nawiasy, aby nakazać bazie danych wykonanie najpierw porównania Wynagrodzenie > 10000, a następnie operatora NOT. W tym przypadku nie jest to konieczne, ponieważ domyślnie najpierw stosowane są operatory porównania, a następnie operator logiczny NOT. Ale musisz znać

zasady pierwszeństwa. W każdym razie zalecamy używanie nawiasów, jeśli to możliwe, ponieważ kod jest znacznie bardziej przejrzysty i można łatwo wiedzieć, które warunki należy sprawdzić w pierwszej kolejności, i uniknąć błędów. Możemy zagwarantować, że zaoszczędzi to dużo czasu na debugowaniu źle działających zapytań. Zwykle najpierw wykonywane są operacje arytmetyczne. Tak więc dodawanie, odejmowanie, mnożenie lub dzielenie zostanie wykonane przed jakimkolwiek innym warunkiem. Później stosowane są operatory porównania. Po nich LIKE, IN i NULL. Następnie instrukcja Between, następnie operator porównania <>, a na samym końcu operatory NOT, AND i OR w tej kolejności. Pozostawienie tych operatorów logicznych na koniec ma sens, ponieważ, jak wyjaśniliśmy wcześniej, są one używane głównie jako łącznik między innymi operatorami lub grupami warunków. Jeśli dotarłeś do tego momentu, to jest to bardzo dobra wiadomość. Nauczyłeś się prawdopodobnie najważniejszej części pisania zapytań. To oczywiście jeszcze nie wystarczy, ponieważ sprawy mogą się znacznie skomplikować, a pisanie dobrych, wydajnych i przejrzystych zapytań SQL wymaga trochę czasu. Ale jesteśmy na dobrej drodze.

Grupowanie danych

Czasami nie chcesz pobierać pojedynczych wierszy z bazy danych. Chcesz pogrupować według określonych danych i pobrać tylko sumy, liczbę grup, średnie i tak dalej. Jest to bardzo typowe w środowisku hurtowni danych. Pomyśl o menedżerze, który chce odzyskać sprzedaż z określonego dnia. W zależności od wielkości Twojej firmy pobieranie danych sprzedaży pojedynczo nie będzie miało sensu, ponieważ prawdopodobnie bardziej interesuje Cię konkretny szczegół niż weryfikacja wszystkich zakupów klientów. W takich przypadkach musimy mieć możliwość grupowania rekordów według określonych grup, aw SQL osiąga się to za pomocą klauzuli GROUP BY. Istnieje lista funkcji agregujących lub zestawów, których również musisz się nauczyć. Określają one operację, którą chcesz obliczyć na grupie.

Operator : Znaczenie : Przykład

MAX () : Zwraca wartość maksymalną zestawu. : MAKS (wynagrodzenie)

MIN () : Zwraca minimum zestawu. : MIN (pensja)

SUMA () : Zwraca sumę lub sumę częściową zestawu. : SUMA (wynagrodzenie)

AVG () : Zwraca średnią zestawu. : średnia (wynagrodzenie)

COUNT () : Zlicza rekordy. Możesz również określić warunek wewnątrz. : LICZBA (pensja > 10000)

Aby móc uwzględnić kolumny w klauzuli group by, musimy upewnić się, że znajdują się one również w części projekcyjnej zapytania. Oznacza to, że nie możemy dodać kolumny w klauzuli select, że nie ma jej w klauzuli group by. Niektóre bazy danych pozwalają to zrobić, modyfikując niektóre parametry, ale zwykle jest to coś, czego chcemy uniknąć, więc miej to na uwadze. Wiedząc o tym, możemy obliczyć np. średnią wynagrodzeń pracowników według ich działów, aby zobaczyć, w którym dziale firmy najlepiej pracować, jeśli oczywiście zależy nam tylko na pieniądzach!

```
SELECT
      department_id, trunc(avg(salary))
FROM
      hr.employees
GROUP BY
      department_id
ORDER BY


      2 DESC;

DEPARTMENT_ID                      TRUNC(AVG(SALARY
))
------------ ------------------------------------
--
          90                                   193
33
         110                                   101
50
          70                                   100
00
... (output truncated)
          10                                    44
00
          30                                    41
50
          50                                    34
75
12 rows selected
```

Jest więc jasne, że osoby przypisane do działu o id = 90 są wyraźnie tymi, które zarabiają więcej, podczas gdy te przypisane do działu o id = 50 otrzymują niższe odcinki wypłat. Można to łatwo wytłumaczyć, ponieważ dział o id = 90 to dział Kierownictwa.

Uwaga: Zapomnij na chwilę o używaniu funkcji TRUNC() . Dodaliśmy je, aby uzyskać wyraźny wynik i nie mieć wyników z dużymi miejscami po przecinku. W zależności od każdej bazy danych ta funkcja ma różne nazwy, ale w zasadzie instruuje bazę danych, aby pozbyła się części dziesiętnej liczby poprzez obcięcie danych wyjściowych. Uwaga, to nie oznacza tego samego, co Runda. Jeszcze raz; sprawdź instrukcję sprzedawcy, aby uzyskać więcej informacji.

Oczywiście czasami to nie wystarcza i chcesz przefiltrować konkretną grupę, bo np. interesują Cię tylko te, które mają co najmniej 5 pracowników. Wymaga to użycia klauzuli HAVING, tak jakbyś myślał, że nie ma możliwości filtrowania tego za pomocą klauzuli where. Kluczową kwestią jest tutaj zrozumienie, że każdy filtr w klauzuli Where filtruje wiersze, ale w tym przypadku musimy odfiltrować grupy z danych wyjściowych, a nie pojedyncze wiersze. Potrzebna jest więc klauzula posiadania. Oto przykład: Po pierwsze, liczba pracowników na dział:

```
SELECT
      department_id, COUNT(*)
FROM
      hr.employees
GROUP BY
      department_id
ORDER BY
      2 DESC;

DEPARTMENT_ID   COUNT(*)
------------- ----------
           50         45
           80         34
          100          6
           30          6
           60          5
           90          3
           20          2
          110          2
           40          1
           10          1
                       1
           70          1
12 rows selected
```

A następnie spójrzmy na nasze działy, które mają przydzielonych więcej niż 5 pracowników:

```
SELECT
      department_id, COUNT(*)
FROM
      hr.employees
GROUP BY
      department_id


HAVING
      COUNT(*) > 5
ORDER BY 2 DESC;

DEPARTMENT_ID   COUNT(*)
------------- ----------
           50         45
           80         34
           30          6
          100          6
 4 rows selected
```

Ważne jest również, aby zrozumieć, że klauzula „having” jest stosowana po zastosowaniu wszystkich filtrów w klauzuli „where”. Tak więc w poprzednim przykładzie, jeśli odfiltrujemy pracowników, którzy zarabiają więcej niż 9 000, a następnie usuniemy grupy, które mają mniej niż 5 pracowników, wynik może być inny, ponieważ niektórzy pracownicy mogli już zostać usunięci z grup na podstawie warunku Wynagrodzenie > =9.000, wpływając na całkowitą liczbę pracowników dla tej konkretnej grupy. Zobaczmy przykład:

```
SELECT
      department_id, COUNT(*)
FROM
      hr.employees
WHERE
      salary < 9000
GROUP BY
      department_id
HAVING
      COUNT(*) >= 5
ORDER BY 2 DESC;

DEPARTMENT_ID   COUNT(*)
------------- ----------
           50         45
           80         17
           30          5
3 rows selected
```

Jak więc widać, straciliśmy dwie grupy. Wynika to z faktu, że id_działu 100 i 60 miały co najmniej dwóch i jednego pracownika, z których każdy zarabiał co najmniej 9 000 USD. Ponadto działy 80 i 30 zmniejszyły liczbę członków spełniających warunek wynagrodzenia <9000 USD, zmieniając wynik zapytania.

Korzystanie z podzapytań

Wiele razy chcesz określić filtr, ale ten filtr jest obliczany na podstawie jakiegoś parametru, który nie jest oczywisty lub nie można go wcześniej ustawić, ponieważ zależy od twoich własnych danych. Jak widzieliście w poprzednich przykładach, możemy znaleźć dla pracowników zarabiających ponad 10 000 USD. To jest ok, ale co się dzieje, jeśli chcemy szukać pracowników zarabiających więcej niż średnia w firmie, aby zobaczyć, czy na to zasługują? Najwyraźniej to, czego nauczyliśmy się do tej pory, nie wystarczy, ponieważ średnie wynagrodzenie w firmie może się zmieniać od czasu do czasu. Pomyśl o podwyżkach płac, zatrudnionych lub zwolnionych pracownikach itd., które zmienią tę liczbę. Jest to bardzo powszechna sytuacja i musimy wprowadzić pojęcie podzapytania. Aby to ułatwić, najpierw będziemy szukać średniej pensji firmy, a następnie przetestujemy wszystkich pracowników pod kątem tej pensji, jeśli wynosiła ona około 10 000 USD. Więc jedyne, co musimy zrobić, to znaleźć sposób, aby najpierw obliczyć tę średnią pensję, a potem reszta będzie taka, jak pokazano wcześniej. Pokażmy przykład, jak to obliczyć. Na początek napiszmy zapytanie, jak znaleźć wszystkich pracowników zarabiających powyżej 10 000 USD. Będzie to coś podobnego do następującego:

```
  SELECT
        first_name, last_name, salary
  FROM
        hr.employees
  WHERE
        salary > 10000
  ORDER BY 3 DESC;

  FIRST_NAME           LAST_NAME
SALARY
  -------------------- ------------------------- ----
------
  Steven               King
 24000
```

```
  Neena                Kochhar
17000
  Lex                  De
Haan                       17000
  John                 Russell
14000
  Karen                Partners
13500
  Michael              Hartstein
13000
  Shelley              Higgins
12000
  Alberto              Errazuriz
12000
  Nancy                Greenberg
12000
  Lisa                 Ozer
11500
  Gerald               Cambrault
11000
  Den                  Raphaely
11000
  Ellen                Abel
11000
  Eleni                Zlotkey
10500
  Clara                Vishney
10500
  15 rows selected
```

To połowa pracy, ponieważ teraz musimy zmodyfikować nasze zapytanie, aby zamiast tego kierować się na kwotę 10 000 USD obliczającą pracowników, którzy zarabiają więcej niż średnia firmy. Zrobienie tego jest łatwiejsze niż myślenie o tym, jak obliczyć to drugie. Jeśli do tej pory zrozumiałeś wszystko, co omówiliśmy, powinno to być dla ciebie łatwe

SELECT

TRUNC(AVG(salary))

FROM

hr.employees;

TRUNC(AVG(SALARY))

6461

Uwaga: w tym przypadku nie potrzebujemy grupy według wyrażenia, ponieważ nie chcemy żadnej grupy, rozważamy firmę jako całość. Gdybyśmy chcieli wykonać te same obliczenia według działów, potrzebowalibyśmy trochę pogrupowania według id_działu, ale zobaczymy to później, ponieważ podzapytanie zwraca więcej niż jeden wiersz (w rzeczywistości jeden na dział), a nadal musimy zobaczyć coś innego aby móc odpowiedzieć na to pytanie.

Ok, to wszystko! Mamy średnią pensję całej firmy, więc teraz pozostaje tylko kwestia zastosowania koncepcji podzapytania. Jak widać mamy tutaj zapytanie główne, to które oblicza pracowników, których pensja jest większa niż jedna kwota oraz podzapytanie, które jest potrzebne do obliczenia średniej firmy. Pozostaje więc tylko kwestia ich wspólnego spisywania. Użyjemy nawiasów w miejscu gdzie i dodamy podzapytanie tak, jakby miało jakąkolwiek wartość:

```
SELECT
      first_name, last_name, salary
FROM
      hr.employees
WHERE
      salary > (SELECT
                        TRUNC(AVG(salary))
                  FROM
                        employees)
ORDER BY 3 DESC;
```

```
And the result is the following:
FIRST_NAME           LAST_NAME
SALARY
-------------------- ------------------------- ----
------
Steven               King
 24000
Neena                Kochhar



 17000
  Lex                  De
Haan                       17000
  ... (output truncated)
  David                Lee
  6800
  Susan                Mavris
  6500
  Shanta               Vollman
  6500
  51 rows selected
```

Oblicz liczbę pracowników, którzy przekraczają średnią pensję

Wykluczanie pracowników z Department_id = 90 (Kierownictwo) Chcemy, abyś ty też spróbował. Jak myślisz, jak moglibyśmy policzyć liczbę pracowników według działów, którzy przekraczają średnią pensję? Chcielibyśmy wyłączyć z średniej kadry kierowniczej. Spróbuj zastosować tę samą metodologię, którą wyjaśniliśmy w tym punkcie, aby znaleźć rozwiązanie.

1. Spróbuj wymyślić zapytanie, które to spełni. WSKAZÓWKA: Powinno to być zapytanie zawierające identyfikator działu i liczbę pracowników.

2. Spróbuj pomyśleć o podzapytaniu, które jest do tego potrzebne. WSKAZÓWKA: Tym razem musimy obliczyć średnią wynagrodzeń firmy, filtrując najpierw pracowników należących do id_działu = 90.

3. Połącz oba zapytania i napisz wymagane zapytanie. Nie oszukuj i spróbuj samodzielnie przemyśleć rozwiązanie! Jeśli po pewnym czasie nie możesz, tutaj jest dla odniesienia:

```
SELECT
      department_id, count(*)
FROM
      hr.employees
WHERE
      salary > (SELECT
                        TRUNC(AVG(salary))


                  FROM
                        hr.employees
                  WHERE
                        department_id <>90)
GROUP BY
      department_id
ORDER BY 2 DESC;
DEPARTMENT_ID   COUNT(*)
------------- ----------
           80         34
          100          6
           50          4
           90          3
          110          2
           40          1
           60          1
           20          1
                       1
           30          1
           70          1

 11 rows selected
```

Łączenie tabel

Do tej pory nauczyliśmy się całkiem sporo o tym, jak pracować z relacyjną bazą danych. Jednak dość często zdarza się, że musimy pracować z kilkoma tabelami jednocześnie. Więc to, co widzieliśmy do tej pory, jest mało przydatne. Zwykle w hurtowni danych będziesz mieć kilka tabel. Niektóre z nich będą tabelami wyszukiwania, zawierającymi podstawowe dane Twoich klientów, dostawców, produktów itd., podczas gdy inne będą tabelami faktów, zawierającymi informacje o sprzedaży, kosztach, odcinkach wypłat pracowników itd. Więcej na ten temat dowiemy się w następnych rozdziałach, ale zacznij o tym myśleć. Na razie koncentrujemy się na naszych dwóch przykładowych tabelach, Pracownicy i Działy. Prawdopodobnie w pewnym momencie chcesz policzyć na przykład liczbę pracowników w każdym dziale. W tym momencie możesz pomyśleć o rozwiązaniu niezbyt eleganckim, ale to zadziała, czyli wybraniu kolumny DEPARTMENT_ID z tabeli pracowników i dodaniu funkcji agregacji, takiej jak count (*), a następnie zastosowaniu grupy w kolumnie DEPARTMENT_ID. Chociaż to zadziała, mamy tutaj pewne wady (i pewne zalety, szczerze mówiąc). Najważniejsze jest to, że to rozwiązanie nie podaje nam nazw działów. Mamy tylko identyfikatory, więc w zależności od tego, jakie informacje chcemy przedstawić, jest to niedopuszczalne. Zaletą jest to, że wysyłanie zapytań tylko do jednej tabeli jest zawsze szybsze niż pobieranie danych z więcej niż jednej, oczywiście biorąc pod uwagę, że stosowane są te same filtry. Istnieje wiele rodzajów sprzężeń, ale składnia jest bardzo podobna. Przedstawimy składnię ANSI 92 Join i poprzednią, ANSI89. Chociaż ANSI 92 jest obsługiwany we wszystkich bazach danych, poprzedni może nie być, zwłaszcza jeśli chodzi o łączenia zewnętrzne. Zależy to całkowicie od tego, którego używasz, i chociaż zalecamy trzymanie się standardu ANSI 92, który jest obsługiwany we wszystkich bazach danych, prawdą jest również, że jesteśmy bardziej przyzwyczajeni do starego. Składnia zapisu łączenia obejmującego dwie tabele w ANSI89 jest następująca:

SELECT list_of_columns (separated by commas, or *)

FROM table1, table2

[WHERE table1.column1=table2.column1 ...]

Whereas in the new syntax the format is as follows:

SELECT list_of_columns (separated by commas, or *)

FROM table1

JOIN table2

ON (table1.column1=table2.column1)

Jak widać, oba formaty są podobne. Aby wygenerować łączenie, w starej składni używamy przecinka, natomiast w nowszej składni używamy słowa kluczowego JOIN (lub podobnego, w zależności od typu łączenia, więcej później), a następnie dodajemy klauzulę ON, aby określić łączone kolumny z dwóch stolików. Zamiast tego, w pierwszym przypadku połączone kolumny są określone w tej samej klauzuli gdzie, co może powodować zamieszanie, ponieważ czasami może być trudno zobaczyć, które kolumny są połączone i jakie są warunki zastosowane w klauzuli gdzie. Większość zwolenników ANSI92 używa tego argumentu jako głównego w swojej obronie nowszej składni.

Rodzaje sprzężeń

Jak powiedzieliśmy wcześniej, istnieje wiele rodzajów sprzężeń. Bardzo ważne jest, aby wiedzieć, co chcemy wybrać, ponieważ zmusi nas to do użycia jednego lub drugiego rodzaju łączenia. Zaczniemy od najbardziej podstawowego, a te najbardziej skomplikowane przyjrzymy się przykładom z tabel Pracownicy i Działy, które widzieliśmy wcześniej.

Połączenie kartezjańskie

W rzeczywistości nie jest to łączenie, ale jest również nazywane łączeniem (łączenie krzyżowe). Jest to relacja „bez relacji" między dwiema tabelami. Z tego powodu jest to rodzaj łączenia, którego najczęściej chcesz uniknąć, ponieważ zwykle dochodzi do niego przez pomyłkę, przy błędnym określeniu klauzul łączenia. Jak sama nazwa wskazuje, ten typ łączenia jest iloczynem kartezjańskim, który obejmuje łączenie (lub łączenie) każdego pojedynczego wiersza z pierwszej tabeli z każdym pojedynczym wierszem drugiej tabeli. Jak już zauważyłeś, całkowita liczba rekordów jest iloczynem liczby rekordów z pierwszej tabeli przez liczbę rekordów z drugiej. Chociaż prawdą jest, że zwykle będziesz chciał uniknąć tego łączenia, może to być przydatne w niektórych przypadkach, na przykład, gdy wiesz, że dwa stoły nie mają ze sobą nic wspólnego, ale mimo to chcesz do nich dołączyć, lub gdy jeden ze stołów, który nie ma nic wspólnego z innymi, zawiera tylko jeden wiersz i chcesz dołączyć dwie tabele razem, generując nową tabelę z kolumnami obu tabel. Mając kilka rekordów z tabeli pracowników

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17/06/87	AD_PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21/09/89	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13/01/93	AD_VP	17000	(null)	100	90

i kilka rekordów z tabeli departamentów ,

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400

możemy utworzyć między nimi sprzężenie krzyżowe:

SELECT

*

FROM

hr.employees, hr.departments;

2.889 records selected

Or using the newer ANSI92 syntax:

SELECT

*

FROM

hr.employees

CROSS JOIN

hr.departments;

2.889 records selected

Jak widać na ostatnim rysunku, kolumna ID_działu pochodząca z tabeli pracowników i kolumna ID_działu pochodząca z tabeli działów nie pasują do siebie. Jest to oczywiste, ponieważ nie dodaliśmy takiego warunku, więc silnik SQL zasadniczo łączy każdego pracownika, niezależnie od jego działu, ze wszystkimi działami w firmie. Na rysunku widać, że pracownicy przypisani do działów 90 i 60 są powiązani z działem 10 i tak dalej.

Połączenie wewnętrzne

Najczęściej używanym łączeniem jest łączenie wewnętrzne. Ten rodzaj łączenia polega na łączeniu dwóch tabel, które mają co najmniej jedną wspólną kolumnę. W klauzuli where lub w klauzuli ON dodajemy warunek table1.column1 = table2.column1 i tak dalej, aby określić wszystkie kolumny, których chcemy użyć do łączenia. Spowoduje to utworzenie wyniku z kolumnami, które wybraliśmy w

projekcji z dwóch (lub więcej) tabel połączonych za pomocą kolumn, które określiliśmy w klauzulach Where lub ON. Zobaczmy przykład: wiemy, że identyfikator działu jest wspólny dla dwóch kolumn i powiedziano nam, że mamy uzyskać listę pracowników i przypisane im nazwy działów. Można to osiągnąć za pomocą następujących zapytań:

```
SELECT
employees.first_name, employees.last_name,
departments.department_name
FROM
hr.employees, hr.departments
WHERE
employees.department_id =
departments.department_id;
```

or using the ANSI92 syntax:

```
SELECT
employees.first_name, employees.last_name,
departments.department_name
FROM
hr.employees
JOIN
hr.departments
ON
(employees.department_id =
departments.department_id);
```

Rezultatem, jak widać, tym razem nie jest już robienie iloczynu kartezjańskiego, ponieważ wprowadzamy poprawną klauzulę łączenia, więc całkowita liczba pobranych rekordów będzie zwykle mniejsza niż wynik mnożenia. Jeśli jedna z tabel (najmniejsza) ma unikalne rejestry dla danej kolumny, liczba zwróconych rekordów będzie równa liczbie rekordów, które mamy w większej tabeli, chyba że jedna z tabel zawierała rekord pusty lub wartość identyfikator działu nie istnieje w tabeli działów, ponieważ wtedy dopasowanie jest niemożliwe. W rzeczywistości mamy pracownika z nieprzypisanym działem w naszej tabeli, więc suma zamiast 107 wynosi 106, jak widać w wynikach zapytania.

```
   FIRST_NAME           LAST_NAME                 DEPA
RTMENT_NAME
   -------------------- ------------------------- ----
-------------------------
   Jennifer             Whalen                    Admi
nistration
   Pat                  Fay                       Mark
eting
   Michael              Hartstein                 Mark


 eting
    Sigal                Tobias                    Purc
 hasing
    106 rows selected
```

Istnieje specjalny przypadek łączenia wewnętrznego, zwany łączeniem naturalnym. Naturalne łączenie wykonuje wewnętrzne równoważne połączenie dwóch tabel bez określania kolumn łączenia. Być może zastanawiasz się, jak silnik to osiąga. To jest łatwe. Używa nazw kolumn i typów danych kolumn, więc każda kolumna, która ma dokładnie taką samą nazwę i ten sam typ danych, jest automatycznie dodawana do łączenia przez silnik. Ponieważ w naszym przykładzie kolumna id_działu jest wspólna dla obu tabel i ma ten sam typ danych, możemy przepisać poprzednie łączenie, aby zamiast tego użyć składni łączenia naturalnego:

SELECT

*

FROM

hr.employees

NATURAL JOIN

hr.departments;

32 rows selected

Jednak zdecydowanie odradzamy używanie naturalnych połączeń, ponieważ zdasz sobie sprawę, że numery rekordów nie pasują, a my starannie wybraliśmy tę próbkę, aby ostrzec Cię o niebezpieczeństwach związanych z naturalnym połączeniem i niepożądanymi skutkami ubocznymi. Jeśli wrócimy do definicji tabeli pracowników i działów, zobaczymy, że oprócz kolumny id_działu, współdzielą one również kolumnę id_kierownika. Więc ponieważ używamy naturalnego łączenia, to łączenie kolumny manager_id jest również automatycznie dodawane w klauzuli join i jest to coś, co nie ma tutaj żadnego sensu, ponieważ kierownik działu niekoniecznie oznacza, że jest kierownikiem wszystkich pracowników zatrudnionych w tym samym dziale.

Uwaga : Używanie naturalnych połączeń może wprowadzać w błąd i być źródłem błędów. Używaj ich tylko wtedy, gdy masz pewność, że mogą być bezpiecznie używane. Zachęcamy do nieużywania ich z następującego powodu: zastanów się, czy mamy dwie tabele, które mają tylko jedną kolumnę, ale w którymś momencie ktoś zmodyfikuje jedną z kolumn, wprowadzając kolumnę, która wcześniej istniała

w drugiej tabeli: Spowoduje to, że join, aby dodać do niego nowe kolumny, powodując potencjalnie niechciany wynik. To samo można zastosować w przypadku połączeń krzyżowych. Używaj go ostrożnie.

Połączenie zewnętrzne

Czasami przydatne jest łączenie tabel, które mają częściowo wspólne wiersze, a w wyniku łączenia chcesz mieć wszystkie wiersze, które istniały w jednej z tabel źródłowych lub w obu z nich, bez względu na to, czy nie mają one odpowiednika w inny połączony stół. Zwłaszcza, gdy pracujemy na hurtowniach danych, jest to zwykle źródłem problemów (brakujące rekordy po złączeniu) i chociaż nasze procesy i skrypty ETL powinny sobie z tym poradzić, w niektórych obliczeniach może być konieczne upewnienie się, że nie stracimy rekordów, które nie istnieją w żadnej z zaangażowanych tabel w złączeniu. Wyobraź sobie, że Twoja firma notuje sprzedaż od początku swojej działalności. Z jakiegokolwiek powodu masz sprzedaż powiązaną z produktami, których już nie sprzedajesz, ale robiłeś to w przeszłości. Wyobraź sobie, że straciłeś wszystkie szczegóły tego produktu, a nawet nie masz już wpisu w swoim systemie ERP lub transakcyjnym. Jeśli połączysz tabelę produktów z tabelą sprzedaży, nie będzie korespondencji między sprzedażą tych produktów a informacjami o produkcie. Oznacza to, że jeśli dołączysz do tych dwóch stołów, stracisz wszystkie rekordy sprzedaży. Doprowadzi to do problemów i zamieszania, ponieważ jeśli obliczysz sprzedaż zagregowaną według kategorii produktów lub całości, a nawet według produktu, „zapomnisz" policzyć te sprzedaże. Jest to czasem niepożądane. Jak możesz pomyśleć, możesz stworzyć fałszywy wpis produktu w tabeli produktów, z poprawnym identyfikatorem produktu, więc wtedy zadziała łączenie wewnętrzne. Może to być rozwiązanie, ale czasami nie jest to łatwe ze względu na liczbę utraconych rekordów lub może to być niepraktyczne. W takich konkretnych przypadkach na ratunek przychodzą łączenia zewnętrzne. Zasadniczo istnieją trzy typy sprzężeń zewnętrznych: lewe sprzężenia zewnętrzne (lub lewe złączenia dla zwięzłości), prawe sprzężenia zewnętrzne (lub prawe złączenia) i pełne złączenia zewnętrzne. Te trzy mają tę samą składnię, z wyjątkiem słów kluczowych join, ale zachowują się zupełnie inaczej.

Uwaga : Złączenia zewnętrzne są szczególnie interesujące do wykorzystania w procesie ETL, aby nie stracić żadnego rejestru podczas ładowania, ale raczej nie zaleca się używania fikcyjnych wpisów dla tabel końcowych w celu ich uzupełnienia.

Lewe złącze zewnętrzne (lub złącze lewe)

W lewym łączeniu rekordy, które nie pasują z pierwszej tabeli, pojawiają się w wyniku łączenia, natomiast te, które nie pasują z drugiej tabeli, nie pojawiają się. Wynikiem złączenia będzie wówczas kompozycja między złączeniem wewnętrznym (wszystkie rekordy, które pojawiają się w obu tabelach) i tymi, które pojawiają się tylko w pierwszej tabeli. Aby to lepiej zrozumieć, skorzystajmy jeszcze raz z naszych zaprzyjaźnionych stolików pracowników i działów. Jeśli przypomnisz sobie tabelę Działy, każdy dział ma kierownika. Cóż, to nie do końca prawda. Istnieją działy, które nie mają menedżera, więc zamiast tego w kolumnie identyfikator_kierownika w tabeli działów występuje wartość pusta. Jeśli klucz obcy nie odwołuje się do wartości, jak w tym przypadku, jeśli wykonamy połączenie wewnętrzne, te działy zostaną odfiltrowane, ponieważ wartość null nie będzie pasować do żadnego pracownika w naszej tabeli (nie mamy pracownika zerowego, prawda?). Zobaczmy przykład lewego sprzężenia zewnętrznego, w którym zwrócimy wszystkie działy, które mają menedżerów, wraz z nazwiskiem menedżera, ale także wszystkie działy, które nie mają menedżera. Tak więc wynikiem połączenia powinno być tych samych 27 działów wraz z nazwiskiem kierownika dla tych, które je mają.

```
SELECT

d.department_name, e.First_Name || ' ' ||
```

e.Last_Name as Manager

FROM

hr.departments d

LEFT JOIN

hr.employees e

ON (d.manager_id=e.employee_id);

DEPARTMENT_NAME MANAGER

------------------------------ --------------------

Executive Steven King

IT Alexander Hunold

Finance Nancy Greenberg

Purchasing Den Raphaely

Shipping Adam Fripp

..... (output truncated)

Benefits

Shareholder Services

Control And Credit

Corporate Tax

Treasury

27 rows selected

Jak widać z poprzedniego fragmentu, mamy 27 rekordów, ponieważ mamy 27 działów, a niektóre z nich pokazują zerowego menedżera. Dzieje się tak, ponieważ Outer Join dodaje rekordy, które mają wpis w tabeli działów, ale nie w tabeli pracowników (menedżer).

Prawe łączenie zewnętrzne (lub prawe łączenie)

W ten sam sposób mamy lewe łączenie, mamy prawe łączenie. Pomysł jest ten sam, ale tym razem rekordy, które zostaną dodane do wyniku łączenia wewnętrznego, to te, które istnieją w prawej tabeli (drugiej tabeli) łączenia, podczas gdy te, które są obecne tylko w lewej tabeli (pierwsza tabela złączenia) zostaną utracone. Wyobraźmy sobie, że teraz chcemy wiedzieć, w jakim dziale pracownik jest kierownikiem. Jak myślisz, nie wszyscy pracownicy są menedżerami, więc jeśli powtórzymy to samo złączenie, które zrobiliśmy w poprzednim przykładzie, tym razem będziemy mieć wszystkie rekordy z prawej części złączenia (tabela pracowników) i z którego działu zarządzanie:

```
   SELECT
         d.Department_name, e.First_Name || ' ' ||
e.Last_Name as Manager
   FROM
         hr.departments d
   RIGHT JOIN
         hr.employees e
   ON (d.manager_id=e.employee_id);

   DEPARTMENT_NAME                MANAGER
   ------------------------------ --------------------
--------------------------
   Administration                 Jennifer Whalen
   Marketing                      Michael Hartstein
   Purchasing                     Den Raphaely
   Human Resources                Susan Mavris
   Shipping                       Adam Fripp
   IT                             Alexander Hunold
   Public Relations               Hermann Baer
   Sales                          John Russell
   Executive                      Steven King



 Finance                        Nancy Greenberg
 Accounting                     Shelley Higgins
 ... (output truncated)
                                Mozhe Atkinson
                                Alberto Errazuriz
                                Allan McEwen
                                Douglas Grant

   107 rows selected.
```

Jak widać tym razem są pracownicy, którzy niczym nie zarządzają.

Pełne połączenie zewnętrzne (lub pełne połączenie)

Wyobraź sobie, że chcesz, aby zarówno Left Join, jak i Right Join zostały wykonane w tym samym czasie. Wtedy na ratunek przychodzi Full Outer Join lub Full Join. Wyobraź sobie, że chcesz listę działów i ich kierowników, ale jednocześnie chcesz wszystkich pracowników i dział, którym zarządzają. Oczywiście potrzebujesz kombinacji obu. Zobaczmy przykład:

```
SELECT
      d.Department_name, e.First_Name || ' ' ||
e.Last_Name as Manager
FROM
      hr.departments d
FULL OUTER JOIN
      hr.employees e
ON (d.manager_id=e.employee_id);

DEPARTMENT_NAME                MANAGER
------------------------------ --------------------
-------------------------
Executive                      Steven King
                               Neena Kochhar
                               Lex De Haan
IT                             Alexander Hunold
                               Bruce Ernst
... (output truncated)
Payroll
Recruiting


Retail Sales

123 rows selected.
```

Jak widać mamy teraz Działy z Kierownikiem, Pracowników, którzy nie zarządzają żadnym działem oraz Działy bez Kierownika.

Uwaga: lewe i prawe łączenie jest znacznie częściej używane niż pełne łączenie zewnętrzne, zwłaszcza w środowisku hurtowni danych. Ale ważne jest również, aby wiedzieć, że zawsze istnieje możliwość połączenia obu w jedno oświadczenie.

Aliasy tabeli

Czasami my, jako ludzie, jesteśmy trochę leniwi. Ciągłe odwoływanie się do nazw tabel za pomocą ich nazw jest trudne, a ponadto, jeśli ta sama tabela jest używana kilka razy w instrukcji select, jest myląca. Na szczęście język SQL rozwiązuje ten problem, umożliwiając programistom nadawanie tabelom pseudonimów lub aliasów. Następujące stwierdzenia są podobne:

SELECT

first_name, last_name, department_name

FROM

hr.employees, hr.departments

WHERE

employees.department_id =

departments.department_id;

and

```
SELECT

first_name, last_name, department_name

FROM

hr.employees e, hr.departments d

WHERE

e.department_id = d.department_id;
```

Jedyna różnica polega na tym, że dodaliśmy dwa aliasy tabel, aby odwoływać się do ich oryginalnych tabel przy użyciu aliasu lub pseudonimu. Alias należy dodać po nazwie tabeli, a następnie można go użyć w klauzuli where i kolejnych klauzulach, aby odnieść się do oryginalnych tabel.

Skorelowane podzapytania

Widzieliśmy wcześniej, jak działa podzapytanie. Poradziliśmy jednak, że niektóre zapytania podrzędne muszą używać sprzężeń, aby zewnętrzna tabela mogła obliczyć określone obliczenia. Wyobraźmy sobie, że chcemy odzyskać pracowników, których wynagrodzenie jest powyżej średniej wynagrodzeń wszystkich osób w ich działach. Nie możemy tego zrobić bezpośrednio za pomocą podzapytania, ponieważ musimy obliczyć średnią pensję dla każdego działu, a następnie porównać każdego pracownika z tą średnią pensją. Ale te dwa zapytania są ze sobą powiązane, ponieważ dział, w którym przebywa pracownik, musi być taki sam, jak ten, który obliczamy dla wynagrodzenia, więc skutecznie potrzebujemy łączenia. To jeden z najczęstszych przykładów. Zobaczmy, jak to rozwiązać. Zapytanie będzie się składało, jak wspomniano, z dwóch części. Jedno, zwane zapytaniem zewnętrznym, wybierze pracowników spełniających warunek, a drugie zapytanie, zwane zapytaniem wewnętrznym, będzie zapytaniem obliczającym średnie wynagrodzenie na dział. Relacja między zapytaniem wewnętrznym i zewnętrznym zostanie określona w klauzuli where zapytania wewnętrznego, ponieważ zapytanie zewnętrzne nie może odwoływać się do kolumn zapytania wewnętrznego, chyba że znajdują się one w klauzuli FROM, co nie ma miejsca (są one w WHERE lub klauzula filtrująca). Zapytanie będzie wyglądać mniej więcej tak. Zwróć uwagę na aliasy tabeli wewnętrznej i zewnętrznej, jak wyjaśniono w poprzednim akapicie:

```
   SELECT
         first_name || ' ' || last_name EMP_NAME,
salary
   FROM
         hr.employees emp_outer
   WHERE
         salary > (SELECT
                           AVG(salary)
                  FROM
                           employees emp_inner
                  WHERE
                           emp_inner.department_id =
emp_outer.department_id)
   ORDER BY 2 DESC;

   EMP_NAME
SALARY


   ---------------------------------------- ----
------
   Steven
King                                      24000
   John
Russell                                    14000
   Karen
Partners                                   13500
   Michael
Hartstein                                 13000
   ... (output truncated)
   Renske
Ladwig                                     3600
   Jennifer
Dilly                                     3600
   Trenna
Rajs                                       3500

    38 rows selected
```

Ustaw operacje

Łączenie danych z różnych tabel jest bardzo przydatne, ale są też inne operacje, których trzeba się nauczyć. Co się stanie, jeśli chcesz połączyć dane z dwóch tabel, które mają identyczny układ? Pomyśl na przykład o dwóch różnych tabelach zawierających dane sprzedażowe z 2016 i 2017 roku. Musi istnieć sposób na ich „połączenie” i wykorzystanie jako jednej tabeli. Na szczęście jest jeden. Przedstawimy kilka operatorów zbiorów, które ułatwią te i inne zadania. Zacznijmy od operatora unii.

Union i Union Wszystkich Operatorów

Operator Unii, jak wprowadzono wcześniej, łączy wynik jednego zapytania z wynikiem innego. Warto jednak wiedzieć, że są pewne wymagania. Dwie tabele lub zapytania muszą zwracać identyczną liczbę kolumn, a także mieć te same typy danych w każdej kolumnie. Gwarantuje to, że wynik może być konkatenacją wyników obu zapytań lub tabel, a dane zostaną wyrównane i umieszczone w kolumnie, która musi być. Przykład sprzedaży przedstawiony w poprzednim akapicie jest najbardziej przejrzysty, aby zrozumieć, jak zachowuje się instrukcja UNION lub UNION ALL.

Aby pokazać to na przykładzie, musimy trochę popracować nad danymi HR. Wróćmy do naszej tabeli pracowników i utworzymy dwie nowe tabele na podstawie wynagrodzenia, jakie otrzymuje pracownik. Stworzymy jedną tabelę dla pracowników zarabiających mniej lub równo niż 6000 USD, a drugą dla pracowników zarabiających powyżej 6000 USD.

Uwaga: używamy tutaj instrukcji tworzenia, aby pokazać ten przykład. To stwierdzenie zostanie omówione później w następnych sekcjach, więc nie martw się, jeśli nie rozumiesz go dobrze w tym momencie; po prostu wykonaj przykładową instrukcję i postępuj zgodnie z instrukcjami.

Uruchommy następujące dwie instrukcje:

```
CREATE TABLE

hr.employeesLTE6000

AS SELECT

*

FROM

hr.employees

WHERE

salary <=6000;

and

CREATE TABLE

hr.employeesGT6000

AS SELECT

*

FROM

hr.employees

WHERE

salary >6000;
```

the output should be something like that:

```
Table HR.EMPLOYEESLTE6000 created.

Table HR.EMPLOYEESGT6000 created.
```

Mamy więc teraz w naszym schemacie HR dwie nowe tabele, jedną dla pracowników zarabiających więcej niż 6000 USD, a drugą dla pracowników zarabiających mniej niż 6000 USD. Ponieważ obie tabele mają taką samą liczbę kolumn i te same typy danych, możemy użyć instrukcji union, aby połączyć je z powrotem:

```
SELECT

*

FROM

hr.employeesLTE6000

UNION
```

SELECT

*

FROM

hr.employeesGT6000;

Wynikiem selekcji jest 107 pracowników, których mamy w naszej oryginalnej tabeli pracowników. Wyobraź sobie teraz następujący nowy zestaw tabel:

-HR.EMPLOYEESLTE6000, który obejmuje wszystkich pracowników zarabiających mniej niż lub równo 6000 USD. Oraz nową tabelę o nazwie

-HR.EMPLOYEESGTE6000, która zawiera pracowników zarabiających co najmniej 6000 USD. Stwórzmy brakującą tabelę:

CREATE TABLE

hr.employeesGTE6000

AS SELECT

*

FROM

hr.employees

WHERE

salary >=6000;

Table HR.EMPLOYEESGTE6000 created.

And amend slightly our previous query to use the new table:

SELECT

*

FROM

hr.employeesLTE6000

UNION

SELECT

*

FROM

hr.employeesGTE6000;

Wynik jest taki sam. W porządku. Ale co się stanie, gdy zamiast tego użyjemy tego samego zapytania z operatorem UNION ALL? Zobaczmy:

SELECT

*

FROM

hr.employeesLTE6000

UNION ALL

SELECT

*

FROM

hr.employeesGTE6000;

109 rows selected;

Ups! Mamy problem. Mamy dwa rekordy więcej; mamy zduplikowane dane! Może się zdarzyć, że naprawdę chciałeś to zrobić, ale prawdopodobnie tak nie będzie. Właśnie zduplikowaliśmy dane dla pracowników, którzy zarabiają dokładnie 6000 USD, ponieważ są w obu tabelach. Tak więc NION usuwa duplikaty, podczas gdy UNION ALL nie. Być może zastanawiasz się, używajmy UNION zawsze zamiast UNION ALL. To częściowo prawda. Większość ludzi tak robi, ale zazwyczaj nie jest to dobra decyzja. Ponieważ UNION usuwa duplikaty, jest to proces najdroższy do wykonania dla silnika bazy danych UNION ALL. Używaj ich więc mądrze. Jeśli nie zależy Ci na duplikatach, zawsze używaj UNION ALL, ponieważ zawsze będzie szybciej. Jeśli zależy Ci na zduplikowanych rekordach, użyj UNION, która zawsze odrzuca powtarzające się wiersze.

Uwaga: Union statement nie gwarantuje, że wiersze będą dodawane z jednej tabeli po drugiej ani sortowane w zależności od tabeli, z której pochodzą. Wiersze z pierwszej tabeli mogą pojawiać się na początku, na końcu lub mieszać się z wierszami z drugiej tabeli. Jeśli z jakiegokolwiek powodu chcesz zlecić konkretne zlecenie, musisz użyć klauzuli ORDER BY na końcu instrukcji.

Operator przecięcia

Operator przecięcia działa w taki sam sposób jak operator logiczny o tej samej nazwie. Zasadniczo odczytuje dane z obu zapytań lub tabel i przechowuje tylko te wiersze, które pojawiają się dokładnie w obu tabelach. Ponownie konieczne jest spełnienie tych samych warunków wstępnych, co w przypadku deklaracji Union i Union All. Zobaczmy przykład wykorzystujący wcześniej utworzone tabele:

```
SELECT
      *
FROM
      hr.employeesLTE6000
INTERSECT
SELECT
      *
FROM
      hr.employeesGTE6000;
```

As you may see, two employees are returned, which are exactly the two that appear in both tables, having $6,000 as a salary.

```
EMPLOYEE_ID FIRST_NAME           LAST_NAME
----------- -------------------- ------------------
-------
EMAIL                     PHONE_NUMBER         HIRE
_DATE JOB_ID         SALARY
------------------------- -------------------- ----
----- ---------- ----------
COMMISSION_PCT MANAGER_ID DEPARTMENT_ID
-------------- ---------- -------------
         104 Bruce                Ernst
BERNST                    590.423.4568         21/0
5/07  IT_PROG          6000
                        103            60

         202 Pat                  Fay
PFAY                      603.123.6666         17/0
8/05  MK_REP           6000



                        201            20
```

Operator minusa

Operator minus , znany również jako wyjątkiem w niektórych silnikach baz danych (ale uwaga, ponieważ składnia może się nieco zmienić), jest operatorem, który odejmuje od pierwszej tabeli rekordy, które są również zawarte w drugiej tabeli. Zobaczmy przykład, który to zilustruje: będziemy odejmować od jednej z naszych poprzednich tabel, PRACOWNICYGTE600, która zawiera wszystkich pracowników zarabiających 6000 lub więcej, te z tabeli PRACOWNICYGT6000, która zawierała Pracowników zarabiających więcej niż 6000. Tak więc wynik operacja odejmowania powinna być nowym zestawem wierszy zawierających tylko pracowników, którzy zarabiają dokładnie 6000, czyli tych, którzy będą obecni tylko w pierwszej tabeli. Sprawdźmy to:

```
  SELECT
        *
  FROM
        employeesGTE6000
  MINUS
  SELECT
        *
  FROM
        employeesGT6000;
  and the result are the two employees that exactly
earn 6000:
  EMPLOYEE_ID
FIRST_NAME           LAST_NAME                 EMAIL
                   PHONE_NUMBER         HIRE_DATE
JOB_ID         SALARY COMMISSION_PCT MANAGER_ID
DEPARTMENT_ID
  ----------- -------------------- ------------------
------- ------------------------- --------------------
--------- ---------- ---------- -------------- -------


--- -------------
          104
Bruce                Ernst                     BERNST
                   590.423.4568         21/05/07  IT_P
ROG          6000                       103
 60
          202
Pat                  Fay                       PFAY
                   603.123.6666         17/08/05  MK_R
EP           6000                       201
  20
```

Praca z datami

Daty są ważne w hurtowniach danych. Chociaż możemy z nimi pracować i przechowywać je tak, jakby były dowolnymi ciągami tekstowymi, nie jest to dobre z punktu widzenia wydajności, ponieważ w pewnym momencie może zaistnieć potrzeba operowania nimi i przekonwertowania ich na format daty. Widzieliśmy, że wszystkie bazy danych obsługują różne typy danych w definicjach kolumn. Tutaj, w tym akapicie, zobaczymy, jak operować funkcjami związanymi z datami. W zależności od dostawcy bazy danych funkcje mogą mieć inną nazwę, ale zasadniczo robią to samo.

Funkcje SYSDATE, NOW(), CURRENT_DATE i GETDATE().

Wszystkie silniki relacyjnych baz danych posiadają funkcję pobierania aktualnej daty systemu. Istnieją różne funkcje zwykle do pobierania tylko części daty, daty i części czasu, milisekund od określonego punktu w czasie i tym podobnych. W zależności od bazy danych funkcja może być wywoływana inaczej, ale wszystkie działają w ten sam sposób. Znajomość bieżącej daty jest zwykle ważna, gdy chcemy wykonywać operacje na datach, takie jak dodawanie, odejmowanie, filtrowanie, a jeszcze ważniejsza, gdy chcemy odświeżyć lub ponownie załadować dane na podstawie kryteriów daty. Wyobraźmy sobie, że chcemy odświeżyć dane tylko dla bieżącego miesiąca, czyli ostatnich 30 dni. Musimy porównać daty w naszych hurtowniach danych z rzeczywistą datą bieżącą. Najpierw musimy znaleźć sposób na pobranie aktualnych danych z naszej bazy danych. Na przykład w Oracle jest to realizowane za pomocą zarezerwowanego słowa sysdate. W SQL Server można to zrobić za pomocą funkcji GETDATE(), podczas gdy w mysql możemy użyć funkcji NOW(), aw Postgres możemy użyć wyrażenia ANSI CURRENT_DATE. Spójrzmy na kilka przykładów:

SELECT

SYSDATE

FROM

DUAL; --(This is Oracle's)

SYSDATE

18/06/16

SELECT

NOW(); --(This is MYSQL's, but you can also

do select SYSDATE();)

2016-06-18 12:46:42

SELECT

CURRENT_DATE --(This is Postgres)

date

2016-06-18

Dodawanie dni do daty

W ten sam sposób, w jaki możemy uzyskać aktualną datę, możemy operować polami daty. Możemy dodawać dni, miesiące i lata, odejmować je i usuwać części daty. Ponownie, zależy to od każdej implementacji bazy danych, dlatego zachęcamy do sprawdzenia instrukcji dostawcy bazy danych lub sprawdzenia jej w Internecie w celu uzyskania bardziej szczegółowych informacji o tym, jak z nimi pracować. W tym rozdziale pokażemy kilka przykładów użycia Oracle Express Edition i mysql. W przypadku Oracle uruchamiamy zapytanie takie jak to:

SELECT

SYSDATE, SYSDATE + INTERVAL '1' DAY

FROM

dual;

SYSDATE SYSDATE+INTERVAL'1'DAY

-------- ----------

18/06/16 19/06/16

Podczas gdy w przypadku mysql musimy go nieco zmienić:

SELECT

SYSDATE(), SYSDATE() + INTERVAL '1' DAY ;

SYSDATE() SYSDATE() + INTERVAL '1' DAY

---------- ----------

2016-06-18 15:22:33 2016-06-19 15:22:33

Za pomocą tej samej procedury możemy odjąć dowolną wartość. Zwróć uwagę, że słowo kluczowe INTERVAL jest bardzo wygodne, ponieważ możemy zmienić interwał DAY o dowolny potrzebny nam przedział czasu: MIESIĄC, ROK, GODZINA, MINUTA, SEKUNDA…

Wyrażenia warunkowe

Język SQL ma wbudowane pewne wyrażenia warunkowe. Wyrażenia warunkowe sprawdzają warunek i wykonują akcję lub inną w zależności od tego, czy wynik testu jest prawdziwy, czy fałszywy. Podobnie jak w przypadku każdego języka programowania, warunek testu musi być warunkiem boolowskim, więc wynik jego oceny zawsze będzie prawdziwy lub fałszywy.

Wyrażenie przypadku

Wyrażenie Case jest bardzo przydatną instrukcją sterującą do częściowego wykonywania obliczeń lub zbierania danych z określonej kolumny na podstawie warunku. Składnia instrukcji case jest następująca:

CASE

WHEN BooleanExpression1 THEN Branch1

...

WHEN BooleanExpressionN THEN BranchN

ELSE BranchN+1

END (alias for the column)

Wyobraźmy sobie teraz, że chcemy pobrać w kolumnie z wartością zależną od daty. Zapytamy o numer miesiąca i przetłumaczymy go na nazwę miesiąca. Oczywiście istnieją lepsze podejścia do tego, ale dla celów ilustracyjnych, to będzie dobre.

SELECT

CASE

WHEN to_char(sysdate,'mm')=06 THEN

'June' ELSE 'Another Month'

END CurrentMonth

FROM

dual;

CURRENTMONTH

June

This is Oracle syntaxis using sysdate and the dual pseudotable but the same

can be written in for example, mysql/mariadb:

SELECT

CASE

WHEN month(now())=06 THEN 'June' ELSE

'Another Month'

END CurrentMonth;

CurrentMonth

June

Wyrażenia Decode() lub IF().

Wyrażenie decode jest również używane jako instrukcja warunkowa. Nie zachęcamy do korzystania z nich, chyba że jesteś do tego przyzwyczajony lub jesteś przyzwyczajony do programowania w językach, które mają klauzule if/else, ponieważ czasami może to być nieco tajemnicze, co utrudnia debugowanie. Można go użyć w zamian za obudowę, a tam, gdzie jest bardziej zwarty, zwykle trudno jest zrozumieć, czy jest ich wiele zagnieżdżonych. Składnia jest następująca: DECODE (Statement,result1,branch1, …resultn, branchn, [else_branch]). Zobaczmy przykład, odpowiednik tego, który zrobiliśmy dla poprzedniego polecenia CASE:

SELECT

DECODE(to_char(sysdate,'mm'), 06, 'June',

'Another Month') CurrentMonth FROM

dual;

CURRENTMONTH

1

June

And in mysql/mariadb instead of decode we will be using

IF(BooleanExpression,if_case,else_case):

SELECT

IF(month(now())=06, 'June', 'Another Month')

CurrentMonth;

CurrentMonth

June

Wniosek

Jedna część do nauki języka SQL to za mało. Chcieliśmy jednak, abyś przedstawił powszechnie używany język interakcji z relacyjnymi bazami danych, abyś mógł zacząć myśleć o pisaniu własnych zapytań. W tym rozdziale zobaczyliśmy krótkie wprowadzenie do relacyjnych baz danych, rodzaje dostępnych instrukcji, typy danych, sposób pobierania i zliczania danych, sortowania, filtrowania i grupowania, a następnie bardziej zaawansowane instrukcje, w tym instrukcje zagnieżdżone lub skorelowane podzapytania. Zalecamy przeczytanie nieco więcej o języku SQL lub zakup jednej z wielu dostępnych książek, które dadzą Ci przewagę w pierwszym kroku potrzebnym do pomyślnego zbudowania rozwiązania BI. Jeśli twój kod jest dobry i dobrze napisany oraz wykonuje obliczenia, do których jest uprawniony, kolejne kroki będą znacznie łatwiejsze i będą działać znacznie szybciej niż kiepski projekt lub wadliwy kod. W kolejnych rozdziałach zaczniemy widzieć komponenty, które będą zgodne z naszą hurtownią danych, jak logicznie definiować encje, a następnie przełożymy to na wymagania do zbudowania schematu potrzebnego do przechowywania naszych danych. Wiemy, że ten rozdział był trochę gęsty, ale obiecujemy, że kolejne będą bardziej praktyczne, ponieważ zaczniemy pracować nad naszym rozwiązaniem od zera i jesteśmy pewni, że wiele się z niego nauczysz i że pomogą zbudujesz własne rozwiązanie BI.