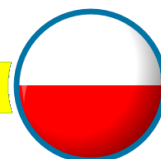


SQL dla początkujących

PostgreSQL

Podręcznik kursowy

Mobilo © 2020



W tym miejscu zwykle pojawia się informacja o tym kto i jak może posługiwać się tym podręcznikiem. Bez regułek prawnych odwołam się po prostu do kilku prostych zasad, które oddają sens kto, kiedy i jak może wg zamysłu autora korzystać z tego materiału:

- Ten podręcznik jest integralnym elementem kursu online.
- Możesz z niego korzystać będąc uczestnikiem tego kursu. Podręcznik jest dla Ciebie i korzystaj z niego do woli – drukuj, wypełniaj, uzupełniaj, przeglądaj, póki Twoim celem jest samodzielne opanowanie tematu.
- Proszę nie umieszczaj go w publicznie dostępnych miejscach, jak blogi, repozytoria git hub, chomik itp.
- Nie wykorzystuj go w innych celach, np. organizacji własnych szkoleń, gdzie występujesz np. jako instruktor.
- Jeśli nie posłuchasz moich próśb, to jako autor zapiszę się do ZAKS-u i następnym razem kupując smartfona zapłacisz za niego kilkaset złotych więcej 😊, więc może lepiej po prostu przestrzegaj praw autorskich 😊

Z góry dziękuję!

Rafał Mobilo © 2020

Zapraszam do odwiedzenia strony:

<http://www.kursyonline24.eu/>

[Review 2021-04-19](#)

Spis treści

Wstęp - Dlaczego PostgreSQL?	5
Jak się uczyć?	6
Instalacja PostgreSQL.....	7
PgAdmin - program do pracy z bazą danych.....	9
Tworzenie bazy danych.....	11
Tworzenie tabeli.....	13
Wstawianie i wyświetlanie rekordów	15
Odtworzenie bazy danych.....	17
Wprowadzenie do SELECT	19
Klauzula WHERE (filtrowanie danych)	21
Operator LIKE (filtrowanie tekstów)	23
Operatory logiczne AND OR NOT.....	25
Klauzula DISTINCT (tylko wartości unikalne).....	27
LIMIT i OFFSET - ograniczenie liczby zwracanych rekordów	29
ORDER BY - sortowanie rekordów	31
Agregowanie wartości.....	33
Klauzula GROUP BY - tworzenie grup z agregacjami.....	35
Klauzula HAVING - filtrowanie agregowanych grup	37
Klauzula BETWEEN zamiast operatorów porównań	39
Operator IN - zamiast wielu OR	41
Korzystanie z funkcji i obliczeń	43
NULL czyli brak wartości	45
Eliminacja NULL - funkcja COALESCE	47
Funkcje tekstowe	49
Funkcje daty i czasu	51
Przegląd typów danych	53
Relacje w relacyjnej bazie danych.....	55
Złączenie tabel przez INNER JOIN	57
Złączenie tabel przez OUTER JOIN	59
Podzapytania skalarne	61
Podzapytanie a join.....	63
Podzapytania skorelowane	65

Złączenie wyników zapytań przez UNION.....	67
Wprowadzenie do widoków - CREATE VIEW	69
Dodawanie rekordów - INSERT	71
Modyfikacja rekordów - UPDATE.....	73
Usuwanie rekordów - DELETE	75
Tworzenie tabel poleceniem SELECT. Tabele tymczasowe.....	77
Przepisywanie rekordów za pomocą INSERT INTO ... SELECT.....	79
Dodatek - Diagram bazy danych Northwind	81
Dodatek - Diagram bazy danych DVD Rental.....	82
Dodatek – Instalacja PostgreSQL 13 na Linux Centos 8	83
Dodatek – Instalacja pgAdmin 4 na Linux Centos 8	84
Notatki	85
Spróbuj też!.....	86

Wstęp - Dlaczego PostgreSQL?

Przechowywanie danych w uporządkowanej postaci to dla aplikacji konieczność. Dane przechowujemy w bazach danych, a najlepszą na świecie bazą danych jest... niestety nie wiem! Każdy rodzaj bazy danych jest trochę inny od pozostałych.

PostgreSQL jest w 100% darmową relacyjną bazą danych, która dzięki swojej ponad 30-letniej historii oferuje szerokie możliwości, jest skalowalna, stabilna i wydajna. Może przechowywać najróżniejsze typy danych, świetnie integruje się z zewnętrznymi narzędziami i językami programowania. Łatwo zainstalować ją na lokalnym systemie, ale jednocześnie jest jedną z popularniejszych baz implementowanych w chmurze. Wszystkie te czynniki sprawiają, że PostgreSQL ciągle rośnie w siłę i to mimo tego, że nie stoi za nią żadna wielka korporacja, która w ramach swojego biznesu byłaby jakoś szczególnie odpowiedzialna za marketing tego rozwiązania. Baza PostgreSQL zdobywa coraz to większy kawałek rynku. To developerzy i użytkownicy sami ją wybierają i to chyba najlepiej świadczy o tym, że pozycja PostgreSQL w światowej czołówce relacyjnych systemów baz danych jest zasłużona.

Jeśli więc chcesz zainwestować w swój rozwój w zakresie pracy z bazą danych, to PostgreSQL wydaje się dobrym kierunkiem. Chmura i open source to przyszłość, a PostgreSQL już tam jest! Amazon, Microsoft, IBM – każda z tych korporacji ma PostgreSQL w swojej ofercie.

Na tym kursie nauczysz się pisać zapytań do bazy danych PostgreSQL.

- Kurs zaczynamy od instalacji bazy danych na systemie,
- Poznajemy pgAdmin – narzędzie klienckie do pracy z SQL
- Uczymy się podstaw relacyjnej bazy danych
- Odtwarzamy przykładową bazę danych
- Piszemy zapytania zaczynając od tych najprostszych, poprzez takie, które wykorzystują funkcje obliczeniowe, filtrują rekordy, konwertują, agregują dane, a kończąc na złączeniach tabel (joinach), wykorzystywaniu podzapytań (subquery)
- Uczymy się jak modyfikować dane i tworzyć proste tabele i widoki

Ponieważ najlepiej uczyć się praktycznie, to do szkolenia dołączony jest podręcznik kursowy (właśnie go czytasz). Dla każdej lekcji znajdziesz w nim krótkie podsumowanie wiadomości – notatkę z lekcji. Do tego jest również zestaw zadań z rozwiązaniami i kilka pytań pozwalających samodzielnie rozważyć działanie prezentowanych poleceń i funkcji. Przedstawiane przykłady nawiązują do prawdziwych problemów, które można spotkać pracując z danymi. Nie stronimy również od błędów. Kiedy już wiesz, jak spowodować jakiś błąd, to automatycznie wiesz też, jak go rozwiązać.

Zapraszam do nauki PostgreSQL – 100% darmowego relacyjnego systemu bazo-danowego

Powodzenia!

Rafał

Jak się uczyć?

Skoro tutaj zaglądasz, to znaczy, że planujesz poznać PostgreSQL i budowanie zapytań do tej bazy danych. To świetnie!

Naukę oczywiście zorganizujesz sobie po swojemu, ale pozwól, że zaproponuję kilka sposobów nauki, a Ty sam/a wybierzesz, co z tego Ci się podoba, a co wolisz zrobić po swojemu

1. **Co za dużo to niezdrowo** – nie rób na raz za dużo materiału. Nie od razu Kraków zbudowano. Jedna lub dwie lekcje na dzień powinny wystarczyć.
2. **Liczy się regularność** – niekoniecznie uczyć trzeba się codziennie, ale jeśli postanowisz przerabiać lekcje we wtorki, czwartki i soboty to już coś!
3. **Wykonuj zadania praktyczne**. Od samego oglądania filmów nie staniesz się programistą. Trzeba tworzyć zapytania samodzielnie
4. **Zmieniaj treść poleceń na własną rękę**. Wykonaj podobny przykład na innej tabeli, zmień dane. Im więcej kreatywności podczas nauki, tym więcej się zapamiętuje
5. Uczę się uczyć, a do głowy nie wchodzi – wszyscy tak mamy i to pewnie dlatego nauka w szkole trwa aż tyle lat! **Od czasu do czasu zrób sobie powtórkę**. Przecież nikt Cię nie goni i nie rozlicza z postępów.
 - a. Sugeruję wrócić do zadań. Jeśli potrafisz je rozwiązać – świetnie przerabiaj następną lekcję
 - b. Jeśli zadania sprawiają problem, wróć do notatki lub lekcji – zobaczysz, że słuchając drugi raz tego samego, materiał nie będzie już taki trudny
6. Notatki w podręczniku są dla Twojej wygody. Niestety wygoda leży blisko lenistwa. Nie bądź leniem. Przygotuj sobie zeszyt lub kilka luźnych kartek i **zapisuj to czego się uczysz**. To co wejdzie oczami lub uszami, będzie wychodzić rękami i... nie ma wyjścia – po drodze zahaczy o mózg 😊
7. Jeśli możesz – **wydrukuj sobie podręcznik, dopisuj do niego własne notatki**, uwagi itp.
8. Kiedy osiągniesz jakiś „kamień milowy”, ukończysz sekcję kursu, a może nawet cały kurs – **daj sobie nagrodę** – to niesamowicie zwiększa motywację!
9. **Nie bój się korzystać z innych materiałów**: książek, blogów, forów itp. Część z nich na początku może być nieco za trudna, ale nic nie stoi na przeszkodzie żeby na początku nic nie mówić, tylko słuchać 😊. Szczególnie polecam polską inicjatywę – **Warsaw PostgreSQL Users Group**:
<https://www.meetup.com/Warsaw-PostgreSQL-Users-Group/>
10. Kiedy już ukończysz kurs – **zaktualizuj CV na LinkedIn**, pochwal się swoim certyfikatem, daj się odnaleźć rekrutom, zaprosz mnie do znajomych (link w profilu). Chętnie potwierdzę Twoją nową umiejętność!

Powodzenia!

Rafał

Instalacja PostgreSQL

Notatka:

- PostgreSQL jest Open Source i jest dostępny za darmo
- PostgreSQL jest dostępny na wszystkie popularne systemy operacyjne
- Aktualnie najnowsza wersja wspiera tylko 64-bitowe wersje systemu operacyjnego
- Najnowsza wspierająca system 32-bitowy to 10
- Aktualnie najnowsza wersja PostgreSQL to (uzupełnij)
- Komponenty instalacji:
 - PostgreSQL Server – silnik bazy danych
 - pgAdmin 4 – narzędzie dla programisty/administratora
 - StackBuilder – dodatkowe narzędzia, sterowniki itp.
 - Command line tools – narzędzia linii komend pozwalające pracować z postgresem
- Data Directory wskazuje na katalog, w którym są przechowywane pliki bazy danych
- Najważniejszym użytkownikiem silnika bazy danych jest postgres i jego hasło jest definiowane podczas instalacji
- Domyślny port PostgreSQL to 5432
- Domyślna baza danych na PostgreSQL nazywa się postgres
- pgAdmin to narzędzie do pracy z PostgreSQL, które działa w przeglądarce

The following settings will be used for the installation::

```
Installation Directory: C:\Program Files\PostgreSQL\13
Server Installation Directory: C:\Program Files\PostgreSQL\13
Data Directory: C:\Program Files\PostgreSQL\13\data
Database Port: 5432
Database Superuser: postgres
Operating System Account: NT AUTHORITY\NetworkService
Database Service: postgresql-x64-13
Command Line Tools Installation Directory: C:\Program Files\PostgreSQL\13
pgAdmin4 Installation Directory: C:\Program Files\PostgreSQL\13\pgAdmin 4
```

Laboratorium

11. Pobierz i zainstaluj PostgreSQL na swoim komputerze.
12. Udokumentuj (dla własnego dobra) wybrane opcje instalacji
13. Przyjrzyj się wykonanej instalacji (np. przejrzyj pliki, jakie zostały wgrane na system)

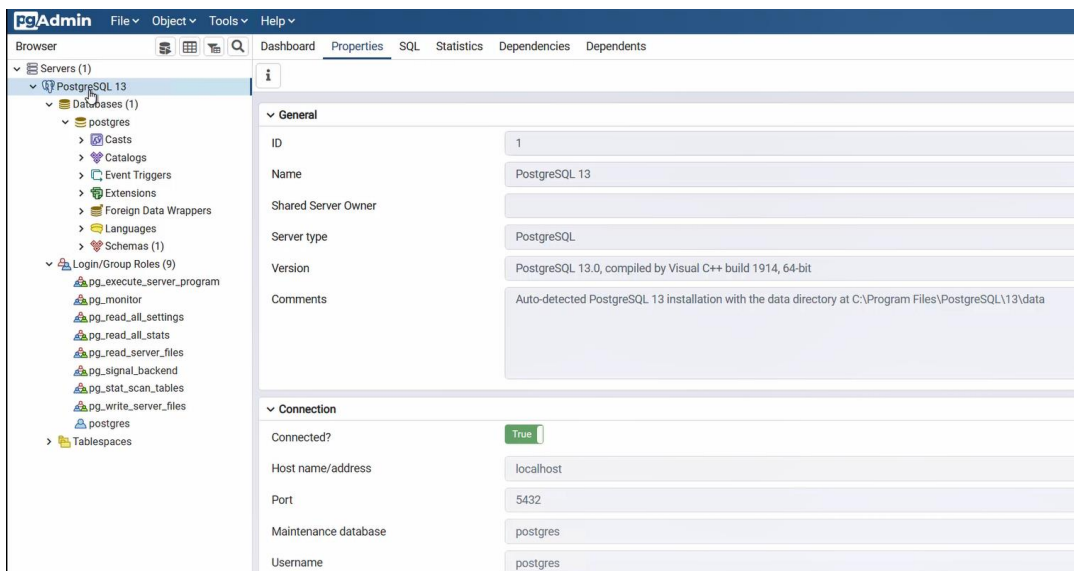
Sprawdź się!

1. Strona domowa PostgreSQL
2. Najnowsza wersja PostgreSQL
3. Jakie komponenty składają się na instalację Postgres?
4. Do czego one służą?
5. Które z nich są najważniejsze?
6. Co jest definiowane przez „Data Directory”?
7. Czy korzystanie z dysku systemowego, jako „Data Directory” to dobry pomysł? Uzasadnij!
8. Jak nazywa się najważniejszy użytkownik administracyjny w PostgreSQL?
9. Jak nazywa się domyślna baza danych?
10. Na jakim porcie domyślnie nasłuchuje PostgreSQL?
11. Co uzasadniałoby zmianę numeru portu?
12. Co można zrobić, jeśli na komputerze brakuje pamięci dla innych programów, a w danej chwili nie korzystasz z PostgreSQL?
13. Jak nazywa się program graficzny pozwalający w wygodny sposób pracować z PostgreSQL?
14. Co dzieje się po uruchomieniu tego programu?
15. Co zrobić, jeśli na systemie masz starszą przeglądarkę ustawioną jako domyślną i nie można podłączyć się do PgAdmin?

PgAdmin - program do pracy z bazą danych

Notatka:

- Klikając w pgAdmin w drzewku po lewej stronie zmienia się zawartość okna po prawej
- Po prawej stronie można przeglądać:
 - Dashboard – informacje natury wydajnościowej
 - Properties – właściwości zaznaczonego obiektu
 - SQL – kod SQL, jakim dany obiekt jest utworzony
 - Statistics – statystyki dla zaznaczonego obiektu
 - Dependencies / Dependents – informacje o powiązaniach między obiektami
- Podstawowym sposobem definiowania obiektów w bazie danych jest SQL
- Wygląd programu pgAdmin można modyfikować przechodząc do File >> Preferences
- Modyfikowanie pewnych opcji wymaga dodatkowo kliknięcia polecenia Refresh
- Większość poleceń jest dostępnych w menu górnym, albo w menu kontekstowym (pod prawym przyciskiem myszy)
- Dużo przydatnych poleceń odnajdziesz w menu Tools
- Narzędzie do pisania zapytań to Query Tool



Laboratorium

1. Na własną rękę przekonaj się jak można pracować z pgAdmin
 - Zajrzyj do opcji
 - Przejrzyj drzewko po lewej
 - Przejrzyj panel po prawej
 - Zapoznaj się z menu głównym i kontekstowym

Sprawdź się!

1. Chcesz zobaczyć właściwości loginu. Co zrobisz?
2. Została wykryta luka zabezpieczeń dotycząca tylko jednej konkretnej wersji Postgresa, a dokładniej tylko jednego konkretnego numeru build. Jak sprawdzisz numer build serwera?
3. Zgubiłeś notatki z zapisanym katalogiem przeznaczonym na pliki bazy danych. Gdzie znajdziesz taką informację?
4. Dane są niepoprawnie sortowane. Jaki parametr jest odpowiedzialny za definiowanie sposobu sortowania?
5. Jak ograniczyć liczbę użytkowników, którzy jednocześnie mogą się podłączyć do bazy danych?
6. Kiedy w interfejsie GUI klikniesz jakąś akcję, to tak naprawdę co dzieje się „pod spodem”?
7. W bazie danych masz tabelę customers_old. Jak sprawdzić, czy ta tabela nie jest już wykorzystywana i co za tym idzie, czy można ją usunąć?
8. Do czego służy Query Tool?

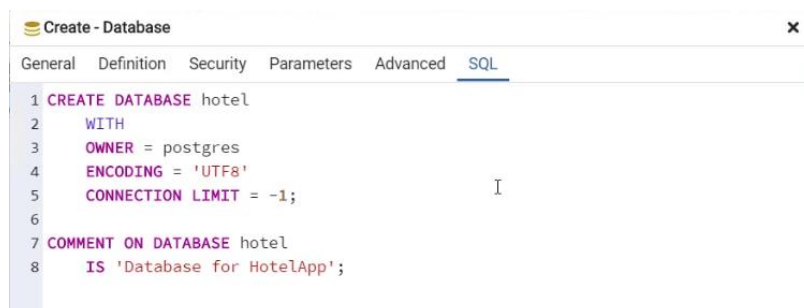
Tworzenie bazy danych

Notatka:

- Tworzenie bazy danych można wykonać w interfejsie graficznym lub korzystając z języka SQL poleceniem

CREATE DATABASE db_name

- Warto stosować konwencję nazewnictwa dla obiektów bazodanowych (np. tylko małe litery i znak „_” jako separator poszczególnych słów)
- Najważniejsze atrybuty i opcje bazy danych:
 - Nazwa
 - Właściciel
 - Komentarz
 - Kodowanie znaków narodowych (encoding)
 - Collation (sortowanie i porównywanie tekstów)
 - Liczba użytkowników, którzy mogą jednocześnie podłączyć się do bazy
 - Uprawnienia użytkowników
- Po utworzeniu bazy danych w interfejsie graficznym jest jednocześnie generowany kod SQL, który można wykorzystać do utworzenia bazy danych. Odpowiedni kod dla już wcześniej utworzonych obiektów też można podejrzeć na zakładce SQL.
- Jeśli po wykonaniu polecenia SQL nie widać zmian w interfejsie graficznym, to należy odświeżyć ten widok (to nie bug, to feature!)
- Tytuł zakładki Query Tool informuje o tym jaki użytkownik, na jakiej bazie i serwerze pracuje
- Każda baza danych ma również swój ID. Pozwala on odnaleźć na dysku pliki tej bazy danych
- Kod tworzący bazę danych musi być uruchamiany w bazie postgres
- Skrypty tworzone w pgAdmin można zapisywać i w ten sposób dokumentować swoją pracę.



```
1 CREATE DATABASE hotel
2 WITH
3 OWNER = postgres
4 ENCODING = 'UTF8'
5 CONNECTION LIMIT = -1;
6
7 COMMENT ON DATABASE hotel
8 IS 'Database for HotelApp';
```

Laboratorium

1. Utwórz bazę danych **hospital**. Wykorzystaj interfejs graficzny. Ze zrozumieniem (!) zaakceptuj domyślne opcje podpowiadane przez pgAdmin.
9. Utwórz bazę danych **shop**. Odpowiednie polecenie napisz w Query Tool. Nie zapomnij odświeżyć drzewka w interfejsie graficznym.
2. Odczytaj identyfikatory obu baz i odszukaj na dysku katalog, w którym są zapisywane pliki tych baz danych

Sprawdź się!

1. Jakie opcje bazy danych można określić podczas jej tworzenia?
2. W jakiej sytuacji może się przydać określenie maksymalnej liczby połączeń do bazy danych?
3. Jakie polecenie języka SQL tworzy nową bazę danych?
4. Tworzysz bazę danych korzystając z GUI. Czy baza danych automatycznie wyświetli się w drzewku?
5. Tworzysz bazę danych korzystając z SQL. Czy baza danych automatycznie wyświetli się w drzewku?

Propozycja rozwiązania

```
CREATE DATABASE shop;
```

Tworzenie tabeli

Notatka:

- Tworzenie tabel można wykonywać w interfejsie graficznym, ale „pod spodem” i tak jest generowany i wykonywany kod SQL
- Tworząc table definiuje się kolumny. Kolumny posiadają typ danych:
 - Serial – liczba autonumerowana
 - Character varying – napis o zmiennej długości (maksymalną długość trzeba zdefiniować)
 - Boolean – wartość logiczna
 - Numeric – wartość liczbowo zmiennoprzecinkowa (należy określić liczbę cyfr i liczbę cyfr po przecinku)
- Kolumny mogą mieć dodatkowe modyfikatory:
 - Primary key – ta kolumna jednoznacznie identyfikuje rekord
 - NOT NULL – pole musi posiadać wartość (nie może być puste)
- Polecenie tworzące table to np.:

```
CREATE TABLE rooms
(
    room_id SERIAL PRIMARY KEY,
    room_number INTEGER NOT NULL
);
```

	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
	listprice_id	serial			<input type="checkbox"/> No <input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes <input type="checkbox"/> No
	name	character varying	50		<input checked="" type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input checked="" type="checkbox"/> No
	is_active	boolean			<input checked="" type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input checked="" type="checkbox"/> No
	pice	numeric	10	2	<input checked="" type="checkbox"/> Yes <input type="checkbox"/> No	<input type="checkbox"/> Yes <input checked="" type="checkbox"/> No

Laboratorium

1. W bazie **hospital** utwórz tabelę **doctors** z następującymi kolumnami:
 2. **doctor_id** – pole automatycznie numerowane, klucz podstawowy
 3. **first_name** – imię – napis max. 50 znaków, musi być wprowadzone
 4. **last_name** – nazwisko – napis max 50 znaków, musi być wprowadzone
 5. **specjalization** – specjalizacja – napis max 50 znaków, może być puste
 6. **room_number** – numer gabinetu, w którym lekarz ma przyjmować pacjentów, może być puste
- Tabelę utwórz jak Ci wygodniej – w interfejsie znakowym lub graficznym

Sprawdź się!

1. Co oznacza typ danych SERIAL?
2. Jaka jest funkcja PRIMARY KEY?
3. Jaki typ odpowiada polom napisowym/znakowym?
4. Czym charakteryzuje się pole oznaczone jako NOT NULL?
5. Jaki typ posiadają pola przechowujące wartość logiczną True/False?
6. Jaki typ danych należy przypisywać kolumnie, która pozwoli na przechowywanie liczby całkowitej lub dziesiętnej?
7. Jaka komenda (dwa początkowe słowa) służy do tworzenia tabel?

Propozycja rozwiązania

```
CREATE TABLE doctors
(
    doctor_id SERIAL PRIMARY KEY,
    first_name CHARACTER VARYING (50) NOT NULL,
    last_name CHARACTER VARYING (50) NOT NULL,
    specialization CHARACTER VARYING (50),
    room_number INT
)
```

Wstawianie i wyświetlanie rekordów

Notatka:

- Zawartość tabeli można wyświetlić w pgAdmin po kliknięciu tabeli prawym przyciskiem myszy i wybraniu View/Edit Data
 - Po dokonaniu edycji, dane można zapisać wykonując polecenie Save data changes
 - Wartości dla typu SERIAL nie trzeba wprowadzać
- Dane do tabeli można wprowadzać poleceniem INSERT:

```
INSERT INTO listprice(name, is_active, price)
VALUES('hostel-room', False, 50)
```

- Aplikacje pracujące z bazą danych też wysyłają polecenia SQL
- Rekordy z tabeli można wyświetlić poleceniem SELECT, np.
SELECT * FROM listprice
- Gwiazdka w powyższym poleceniu oznacza wszystkie kolumny. Zamiast tego można też wymieniać nazwy kolumn, np.

```
SELECT name, is_active, price FROM listprice
```



	listprice_id [PK] integer	name character varying (50)	is_active boolean	price numeric (10,2)
1	1	single person room	true	100.00
2	2	double room	true	180.00

Laboratorium

1. Do tabeli dodaj kilka rekordów. Stosuj interfejs graficzny, ale też kod SQL:
 - a. Gregory / House / ALL / 112
 - b. James / Wilson / oncology / 113
 - c. Eric / Foreman / neurology / brak(Jeśli lubisz serial „Doctor House”, to możesz dodawać tych rekordów więcej:
<https://www.filmweb.pl/serial/Dr+House-2004-130177/cast/actors>)
2. Wyświetl wszystkie kolumny i wszystkie wiersze stosując odpowiednie polecenie SQL
3. Wyświetl tylko imię, nazwisko i numer gabinetu

Sprawdź się!

1. Jeśli tabela ma kolumnę zdefiniowaną jako SERIAL, to czy podczas tworzenia nowego wiersza trzeba podawać jej wartość?
2. Jak wprowadza się wartość logiczną prawda/fałsz do tabeli w poleceniu wstawiającym rekord?
3. Jak wprowadza się tekst do tabeli w poleceniu wstawiającym rekord?
4. Jak wprowadza się liczbę do tabeli w poleceniu wstawiającym rekord?
5. Jaka jest ogólna składnia dla polecenia INSERT?
6. Jak wygląda najprostsze polecenie wyświetlające wszystkie rekordy z tabeli?
7. Jak zmodyfikować to polecenie, aby wyświetlone zostały tylko niektóre kolumny?

Propozycja rozwiązania

```
INSERT INTO doctors(first_name, last_name, specialization, room_number)
VALUES ('Gregory', 'House', 'ALL', 112);

INSERT INTO doctors(first_name, last_name, specialization, room_number)
VALUES ('James', 'Wilson', 'oncology', 113);

INSERT INTO doctors(first_name, last_name, specialization)
VALUES ('Eric', 'Foreman', 'neurology');

SELECT * FROM doctors;

SELECT first_name, last_name, room_number
FROM doctors;
```


Odtworzenie bazy danych

Notatka:

- Istnieją różnego rodzaju przykładowe bazy danych do pobrania w celu nauki postgreSQLa
- Relacyjne bazy danych to w dużym uproszczeniu zbiór tabel wraz z relacjami, jakie te tabele wiążą
- Mamy relacje
 - Jeden do wielu (jedno państwo – wiele miast)
 - Wiele do jednego (wiele miast – jedno państwo)
 - Jeden do jednego (jeden film – jeden reżyser)
 - Wiele do wielu (wielu aktorów w wielu filmach)
- Rozbicie bazy na mniejsze entity/tabele ma na celu między innymi optymalizację użycia pamięci
- Pracując z bazą danych warto mieć jej dokumentację
- Odtworzenie bazy danych odbywa się w dwóch głównych krokach:
- Utworzenie bazy danych
- Wczytanie pliku kopii (plik może być w różnych formatach)
- Po odtworzeniu bazy danych pamiętaj o odświeżeniu drzewka w pgAdmin



Laboratorium

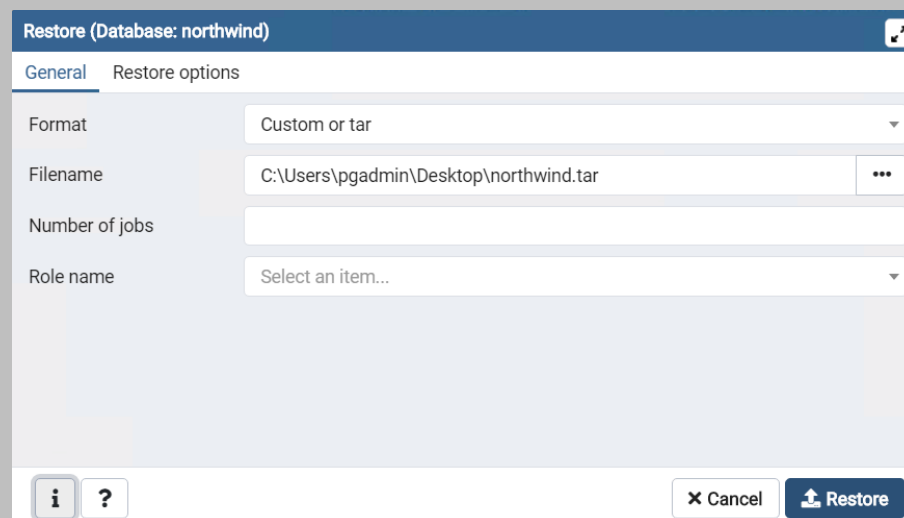
1. Z dołączonych do kursu materiałów pobierz i rozpakuj plik **northwind.zip**. Jego zawartością jest **northwind.tar**.
2. Odtwórz bazę danych **northwind**, której kopia znajduje się w pliku:
3. Utwórz bazę danych **northwind**
4. Odtwórz plik kopii (pamiętaj o zmianie rozszerzenia na „all files”
5. W bazie danych **northwind** wyświetl wszystkie rekordy i kolumny z tabeli **customers**

Sprawdź się!

1. Tabele w relacyjnej bazie danych są powiązane relacjami. Co oznacza, że między tabelą autor, a książką istnieje relacja jeden do wielu?
2. Tabele zawierają liczne kolumny wśród nich często pojawia się kolumna ID Do czego jest zwykle wykorzystywana?
3. Czy bazy danych zawsze muszą obsługiwać wszystkie możliwe scenariusze użycia danych i przewidywać, jak określona baza danych będzie wykorzystywana?
4. Korzystanie z relacyjnej bazy danych ma pozytywny czy negatywny wpływ na użycie pamięci RAM i na dysku?
5. Z jakich dwóch głównych kroków składa się proces odtwarzania bazy danych z pliku tar?

Propozycja rozwiązania

```
CREATE DATABASE northwind;
```



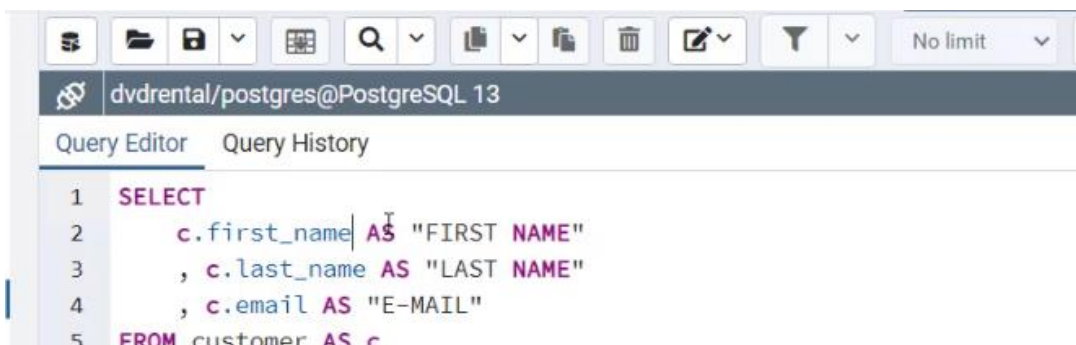
```
-- podłącz się do bazy northwind
```

```
SELECT * FROM customers;
```

Wprowadzenie do SELECT

Notatka:

- Rekordy są:
 - Wstawiane poleceniem **INSERT**
 - Aktualizowane poleceniem **UPDATE**
 - Usuwane poleceniem **DELETE**
 - Wybierane poleceniem **SELECT**
- Podczas wpisywania poleceń SQL w pgAdmin można korzystać ze skrótów:
 - **CTRL + space** – uruchomienie auto-uzupełniania
 - **F5** – uruchomienie wpisanego zapytania
- Schemat to początkowa część dwuczłonowej nazwy obiektów w bazie danych. Schematy służą do grupowania tych obiektów ze względu np. na tematykę opisywaną przez te obiekty
- Schemat public jest schematem domyślnym
- Warto dbać o staranne formatowanie pisanych zapytań SQL
- Rozdzielanie nazw kolumn przecinkiem na początku linijki pozwala na łatwiejszą modyfikację zapytania w przyszłości
- Symbol „—” pozwala na wprowadzenie komentarza jednolinijkowego
- Tabela może otrzymać krótki alias poprzez dodanie po jej nazwie „**AS my_table**”. Kiedy tabela ma alias, można za pomocą tego aliasu określać jakie kolumny mają być pobierane. Napisz **my_table.column_name**
- Aliasować można też zwracane kolumny korzystając ze składni **AS my_column**. Dzięki temu kolumny z wygenerowanego result set otrzymują nazwy zgodne z aliasem
- Tworząc alias można opuścić słowo AS



```
1 SELECT
2     c.first_name AS "FIRST NAME"
3     , c.last_name AS "LAST NAME"
4     , c.email AS "E-MAIL"
5 FROM customer AS c
```

Laboratorium

1. W bazie **northwind** wyświetl z tabeli **customers** kolumny:
 - a. **company_name** (alias **firma**)
 - b. **contact_name** (alias representative)
2. Z tabeli **products** wyświetl
 - a. **product_name** (alias **product**),
 - b. **unit_price** (alias **price**)
 - c. **units_in_stock** (alias **amount**)
3. Na własną rękę możesz przejrzeć inne tabele z tej bazy danych
4. Przejrzyj diagram bazy northwind (dostępny na końcu tego podręcznika) i spróbuj zinterpretować znaczenie najważniejszych tabel. Zinterpretuj relacje między kilkoma tabelami.

Sprawdź się!

1. Jakie polecenie wstawia rekord, jaka go modyfikują, jaka usuwa, a jaka wybiera?
2. Jakim poleceniem wyświetlisz wszystkie rekordy i wszystkie wiersze z tabeli drinks?
3. Co to jest schemat i do czego on służy? Jaki schemat jest schematem domyślnym?
4. Czy sposób pisania polecenia SQL pod względem łamania linii ma w SQL znaczenie czy nie?
5. Jaki symbol oznacza jednolinijkowy komentarz w SQL?
6. Język SQL pozwala na tworzenie aliasów. Co można aliasować?
7. Co oznacza błąd „Relation ... does not exists”
8. W systemie pomocy, w artykule dotyczącym składni odnajdujesz pewne słowo w nawiasie kwadratowym. Co to znaczy?
9. W systemie pomocy, w artykule dotyczącym składni odnajdujesz pewne wyrażenia rozdzielone znakiem pionowej kreseczki |. Co to znaczy?

Propozycja rozwiązania


```
SELECT
    c.company_name AS firma
    , c.contact_name AS representative
FROM customers c;

SELECT
    p.product_name AS product
    , p.unit_price AS price
    , p.units_in_stock AS amount
FROM products AS p
```

Klauzula WHERE (filtrowanie danych)

Notatka:

- Polecenie „SELECT *” nie jest zalecane – lepiej wyraźnie wybierać tylko te kolumny, które są istotne
- Do wybierania określonych wierszy stosuje się klauzulę WHERE, za którą występuje wyrażenie logiczne
- Niektóre z operatorów, które można stosować w WHERE to =, <, <=, >, >=, <>, !=
- Kiedy budujesz wyrażenie filtrujące wg pola napisowego, to ten napis należy umieścić w apostrofie
- Polecenie SELECT z WHERE może zwracać zero, jeden lub więcej wierszy
- Jeśli chcesz znaleźć rekord w oparciu o PRIMARY KEY, to w klauzuli WHERE odwołaj się do kolumn(y) definiującej ten klucz i użyj operatora „=”
- Jeśli kolumna sprawdzana w WHERE jest kolumną logiczną (boolean), to można tą kolumnę porównywać do **True** lub **False**, ale równie dobrze można po prostu odwołać się do wartości tej kolumny. Ponieważ w tej kolumnie znajduje się wartość logiczna to nadaje się do wykorzystania w klauzuli WHERE bez żadnych dodatkowych wyrażeń
- Porównując kolumnę z datą można przyrównywać ją z napisem opisującym datę, np. rental_date > '2016-02-14'



```
1 SELECT
2   *
3 FROM customer AS c
4 WHERE c.create_date >= '2006-02-14'
```

The screenshot shows a PostgreSQL Query Editor window. The title bar indicates the connection is to 'dvdrental/postgres@PostgreSQL 13'. The window has two tabs: 'Query Editor' (active) and 'Query History'. The query editor contains a SQL query with four lines: 'SELECT', '*', 'FROM customer AS c', and 'WHERE c.create_date >= '2006-02-14''. The query is syntax-highlighted, with keywords in purple, the table name in blue, and the date string in red. The toolbar at the top includes icons for file operations, search, and execution, along with a 'No limit' dropdown menu.

Laboratorium

1. Napisz polecenia do tabeli **customers** w bazie **northwind**, które wyświetlą:
 - a. Klientów z Niemiec (kolumna **country** ma mieć wartość 'Germany')
 - b. Klientów, gdzie mamy kontakt z właścicielem firmy (kolumna **contact_title** ma mieć wartość 'Owner')
2. Napisz polecenia do tabeli **products**, które wyświetlą:
 - a. Produkty należące do kategorii 2 (**category_id**)
 - b. Produkty, których mamy w magazynie co najmniej w ilości 100 sztuk (kolumna **units_in_stock**)
 - c. Produkty wycofywane (kolumna **discontinued** ma mieć wartość 1)
 - d. Produkty, których mamy w magazynie mniej niż powinniśmy mieć (wartość w kolumnie **units_in_stock** jest mniejsza niż wartość w kolumnie **reorder_level**)

Sprawdź się!

1. Co jest lepsze w zapytaniu SELECT? Wybieranie poszczególnych kolumn, czy stosowanie gwiazdki? Dlaczego?
2. Jaka klauzula pozwala wybrać z tabeli tylko określone wiersze?
3. Jakie operatory porównania zapamiętałeś/aś z lekcji?
4. Ile wierszy może być zwróconych w wyniku wykonania polecenia, które w klauzuli WHERE narzuca warunek równości na kolumnę klucza podstawowego (PRIMARY KEY)?
5. Tabela ma kolumnę logiczną (boolean) **is_local**. Jak najprościej skonstruować zapytanie, które wyświetli tylko rekordy, które w kolumnie **is_local** mają wartość True?

Propozycja rozwiązania

```
SELECT * FROM customers WHERE country = 'Germany';
SELECT * FROM customers WHERE contact_title = 'Owner';
SELECT * FROM products WHERE category_id = 2;
SELECT * FROM products WHERE units_in_stock >= 100;
SELECT * FROM products WHERE discontinued = 1;
SELECT * FROM products WHERE units_in_stock < reorder_level
```

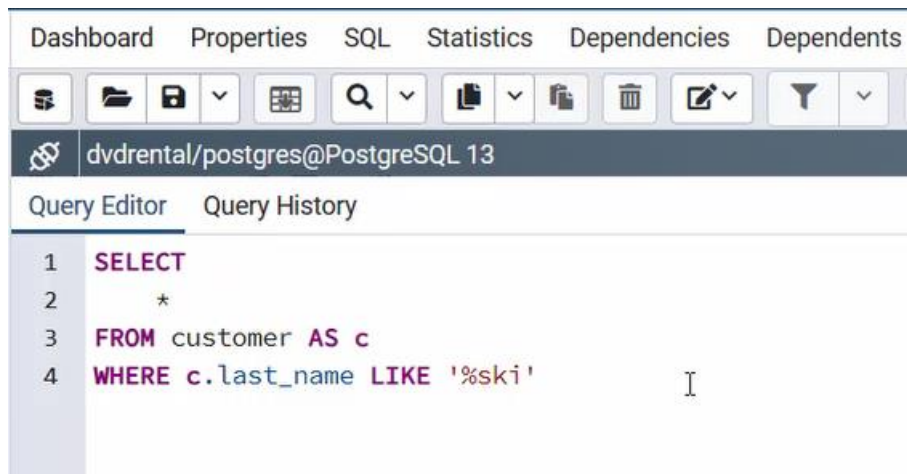
Operator LIKE (filtrowanie tekstów)

Notatka:

- Operator LIKE pozwala przeprowadzić bardziej zaawansowane wyszukiwania bazując na polach tekstowych
- Można dzięki niemu porównywać zawartość kolumn do maski, która może zawierać tzw. metaznaki
- Dostępne metaznaki, które mogą występować w masce wyszukiwania to:
 - % - dowolna liczba dowolnych znaków
 - _ - jeden dowolny znak
- Składnia to

```
last_name LIKE '%ski%
```

- % odpowiada także za brak znaków (dowolna liczba dowolnych znaków – w tym przypadku ta dowolna liczba znaków to zero)
- Domyślnie operatory porównań (w tym operator LIKE) rozróżniają wielkość liter, np. last_name LIKE '%SKI' to nie jest to samo co last_name LIKE '%ski'. Jeśli chcesz żeby LIKE nie rozróżniał wielkości liter, to użyj operatora ILIKE



Laboratorium

1. Z bazy **northwind**, z tabeli **products** wyświetl produkty, które:
 - a. Są pakowane w paczki (kolumna **quantity_per_unit** zawiera słowo 'bag')
 - b. Są pakowane w butelki (kolumna **quantity_per_unit** zawiera słowo 'bottle')
 - c. Nazwa zawiera słowo hot lub Hot lub HOT (kolumna **product_name** zawiera słowo hot zapisane obojętnie jakimi literami – zobacz do notatek powyżej, jaki operator na to pozwala)
2. Z bazy **northwind**, z tabeli **suppliers** (dostawcy) wyświetl firmy, które:
 - a. Jako osobę kontaktową mają managera (kolumna **contact_title** zawiera słowo 'manager' zapisane jakimikolwiek literami)
 - b. Jako osobę kontaktową ma Peter-a (kolumna **contact_name** zawiera Peter – wielkość liter nie ma znaczenia)
 - c. Swoją siedzibę mają w mieście o nazwie składającej się dokładnie z 5 liter (kolumna **city** ma mieć nazwę złożoną z jakichkolwiek pięciu liter)

Sprawdź się!

1. Operator LIKE „rozumie” dwa metaznaki. Jakie to metaznaki i jakie jest ich znaczenie?
2. W tabeli masz pole factory, o długości 5 znaków. Czy wyrażenie 'PFWC%' i 'PFWC_' wykorzystywane z operatorem LIKE w klauzuli WHERE są równoważne?

Propozycja rozwiązania

```
SELECT * FROM products WHERE quantity_per_unit LIKE '%bag%';
SELECT * FROM products WHERE quantity_per_unit LIKE '%bottle%';
SELECT * FROM products WHERE product_name ILIKE '%hot%';

SELECT * FROM suppliers WHERE contact_title ILIKE '%Manager%';
SELECT * FROM suppliers WHERE contact_name ILIKE '%Peter%';
SELECT * FROM suppliers WHERE city LIKE '_____';
```


Operatory logiczne AND OR NOT

Notatka:

- Do budowania złożonych warunków można posługiwać się operatorami logicznymi:
 - OR – jeden lub drugi warunek mają być spełnione

```
firstname LIKE 'J%' OR firstname LIKE 'K%'
```

- AND – oba warunki mają być spełnione:

```
firstname LIKE 'J%' AND firstname LIKE 'K%'
```

- NOT – to zaprzeczenie kolejnego warunku:

```
NOT firstname LIKE 'J%'
```

- Takie warunki można ze sobą łączyć budując stosunkowo skomplikowane kryteria.
- AND wiąże mocniej niż OR
- NOT wiąże mocniej niż AND
- Jeśli warunki mają być wyznaczane w innej (nie domyślnej) kolejności, to warunki należy grupować korzystając z nawiasów
- Wyrażenia logiczne w kryteriach podlegają tym samym prawom, co przekształcenia wyrażeń logicznych (np. negacja alternatywy jest równoważna koniunkcji negacji – to pojęcia i prawa tzw. Logiki matematycznej – np. prawa de Morgana)



Laboratorium

1. Zamrożone pieniądze! Szukamy drogich produktów zalegających w magazynie. Znajdź produkty, których cena (**unit_price**) jest większa równa 100 i mamy ich (**units_in_stock**) co najmniej 10
2. Dlaczego nie zamawiamy wyczerpanych zapasów! Szukamy produktów, których nie ma w magazynie i błędnie nie zaznaczono, kiedy należy je zamawiać – kolumna **units_in_stock** wynosi 0 i **reorder_level** też wynosi 0
3. Dlaczego nie zamówiliśmy produktów, które powinny być już zamówione! Szukamy produktów, których nie ma w magazynie, ale powinny być zamawiane odpowiednio wcześniej (**units_in_stock** ma być równe 0, a **reorder_level** ma być różne od 0)
4. Zawężamy poszukiwania wyłącznie do produktów pakowanych w butelki (kolumna **quantity_per_unit** zawiera słowo 'bottle'). Znajdź drogie produkty, które znajdują się w magazynie (**unit_price** >=100 i **units_in_stock**>0) lub produkty, których mamy dużo nawet jeśli są tańsze (**units_in_stock**>100)

Sprawdź się!

1. Jakie operatory odpowiadają za (uwaga terminy logiczne!): koniunkcję, alternatywę i negację?
2. Uzupełnij tabelki wyznaczając wartości wynikowe:

a	b	a AND b	a OR b
TRUE	TRUE		
TRUE	FALSE		
FALSE	FALSE		
FALSE	TRUE		

A	b	a AND NOT b	(NOT a OR NOT b) AND NOT a
TRUE	TRUE		
TRUE	FALSE		
FALSE	FALSE		
FALSE	TRUE		

a	NOT a
TRUE	
FALSE	

Propozycja rozwiązania

```
SELECT * FROM products WHERE unit_price >= 100 AND units_in_stock >=10;
SELECT * FROM products WHERE units_in_stock = 0 AND reorder_level = 0;
SELECT * FROM products WHERE units_in_stock = 0 AND reorder_level <> 0;
SELECT * FROM products WHERE (unit_price >= 100 AND units_in_stock > 0 OR
units_in_stock > 100) AND quantity_per_unit ILIKE '%bottle%';
```

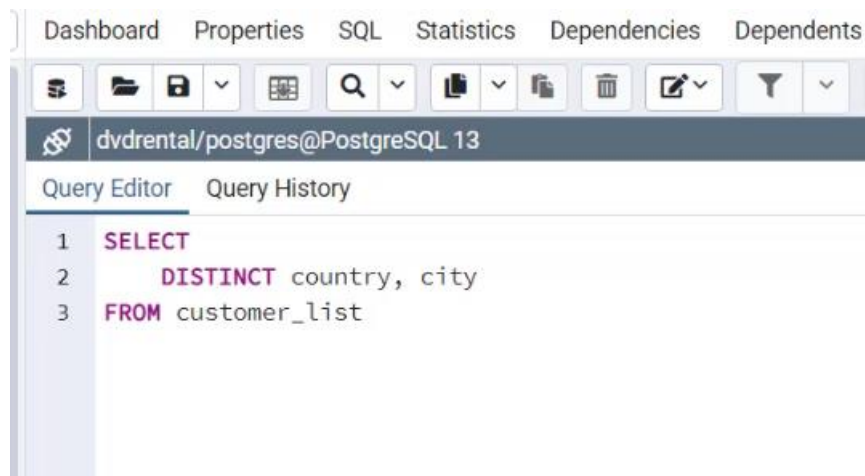
Klauzula DISTINCT (tylko wartości unikalne)

Notatka:

- Dane w tabelach mogą służyć do przygotowywanie zestawień uogólnionych (bez informacji szczegółowej)
- Widok to zapytanie utrwalone w bazie danych. Na tym etapie nauki możemy traktować widoki, jakby to były tabele
- Zapytania wybierające tylko niektóre kolumny z tabel mogą prezentować wiersze, które mają powtarzające się wiersze. Dodanie w liście SELECT słowa DISTINCT spowoduje pominięcie duplikatów
- DISTINCT można stosować względem zapytań zwracających jedną kolumnę, ale również względem zapytań, które zwracają tych kolumn więcej
- Składnia:

```
SELECT DISTINCT country, city FROM customer_list
```

- Nie należy przesadzać z liczbą wybieranych kolumn kiedy stosujesz DISTINCT. Grozi to tym, że w oryginalnym zapytaniu wszystkie rekordy już i tak są unikalne. Dodanie słowa DISTINCT nic nie zmieni, ale spowoduje dodatkowe obciążenie serwera bazy danych. Dokładnie w ten sposób zachowuje się polecenie SELECT DISTINCT, gdy wśród pobieranych kolumn znajduje się PRIMARY KEY



Laboratorium

1. Manager chce przeprowadzić promocję i zastanawia się skąd pochodzą klienci. Napisz zapytanie, które zwróci informację o państwie (**country**) i mieście (**city**) z tabeli **customers**. Każda para państwo- miasto ma być wyświetlone tylko raz.
2. Manager chce wiedzieć na jakich stanowiskach pracują reprezentanci klientów. Napisz zapytanie, które wyświetli unikalne stanowiska (**contact_title**) z tabeli **customers**.
3. Dla twojej firmy będzie przygotowany raport prezentujący sprzedaż z podziałem na regiony określone kodem pocztowym. Na potrzeby tego raportu napisz zapytanie zwracające unikalne kody pocztowe (**ship_postal_code**) z tabeli **orders**
4. Okazało się, że zapytanie z poprzedniego punktu powinno dodatkowo zwracać informacje o kraju (**ship_country**). Popraw je

Sprawdź się!

1. Tabela books zawiera informacje o tytule książki i autorze. Jeden autor mógł napisać wiele książek. Chcesz otrzymać tylko listę autorów. Czy polecenie

```
SELECT author FROM books
```

wylistuje każde nazwisko tylko raz?

2. Tabela books zawiera informacje o tytule książki, autorze i wydawnictwie. Jeden autor mógł napisać wiele książek i jedna książka mogła być wydawana przez wiele wydawnictw. Polecenie

```
SELECT DISTINCT author, title FROM books
```

wyświetli unikalne pary author-title, czy tylko unikalne nazwiska?

3. Tabela books ma kolumnę book_id, title, author. W tej tabeli kluczem podstawowym jest book_id. Jest tu 1000 rekordów. Mamy też 800 unikalnych autorów książek. Ile rekordów zwróci polecenie

```
SELECT DISTINCT book_id, author FROM books
```

Propozycja rozwiązania

```
SELECT DISTINCT country, city FROM customers;
SELECT DISTINCT contact_title FROM customers;
SELECT DISTINCT ship_postal_code FROM orders;
SELECT DISTINCT ship_country, ship_postal_code FROM orders;
```

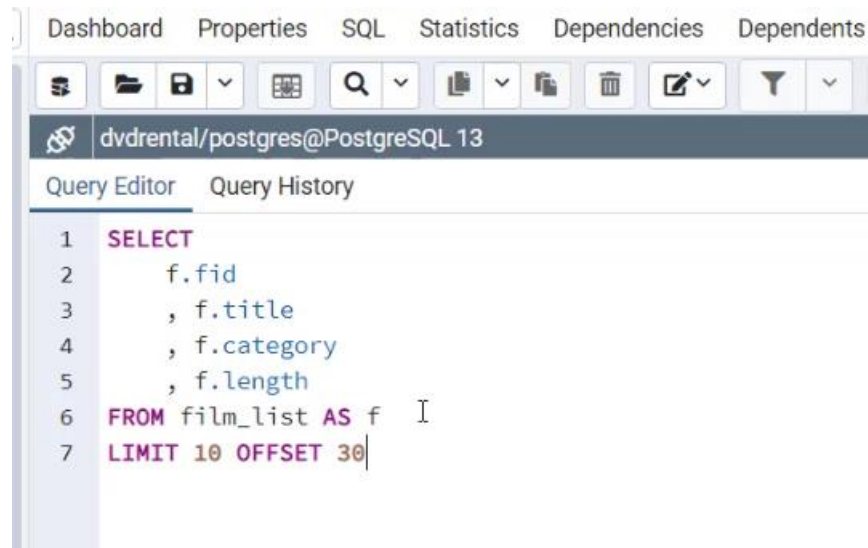
LIMIT i OFFSET - ograniczenie liczby zwracanych rekordów

Notatka:

- Z punktu widzenia wydajności, lepiej jest, jeśli zapytania zwracają mniej danych, aniżeli gdy te zapytania zwracają dużo danych. Osiąga się to przez
 - Wybór tylko określonych kolumn zamiast wszystkich (*)
 - Definiowanie selektywnych warunków w klauzuli WHERE
 - Stosowanie słów kluczowych LIMIT i OFFSET
- LIMIT definiuje, ile wierszy ma być zwróconych przez zapytanie
- OFFSET definiuje, ile wierszy należy opuścić podczas zwracania wyników
- Składnia polecenia wygląda tak:

```
SELECT col1, col2,... FROM table LIMIT 10 OFFSET 30
```

- LIMIT i OFFSET muszą być wykonywane w określonej kolejności Domyślnie zapytania zwracają wyniki w niezdefiniowanej kolejności (nawet jeśli dane wyglądają na posortowane, to nie można na tej kolejności polegać). Do sortowania danych należy posługiwać się poleceniem ORDER BY, o którym mowa w kolejnej lekcji



The screenshot shows a web-based PostgreSQL interface. At the top, there are tabs for 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', and 'Dependents'. Below these is a toolbar with various icons. The main area shows the connection 'dvdrental/postgres@PostgreSQL 13'. Below the connection name are two tabs: 'Query Editor' (active) and 'Query History'. The 'Query Editor' contains a SQL query with line numbers 1 through 7. The query is:
1 SELECT
2 f.fid
3 , f.title
4 , f.category
5 , f.length
6 FROM film_list AS f
7 LIMIT 10 OFFSET 30

Laboratorium

1. Programista przygotowuje stronę webową, która ma za zadanie wyświetlać wybrane informacje o produktach strona po stronie. Ponieważ programista nie ma doświadczenia z budowaniem poprosił cię o pomoc. Kryteria są następujące:
 - a. Zapytanie ma pracować na tabeli **products**
 - b. Należy wyświetlić nazwę produktu (**product_name**), cenę (**unit_price**), informacje o ilości jednostkowej (**quantity_per_unit**)
 - c. Ma być zwróconych
 - i. Zapytanie 1 – 10 pierwszych rekordów
 - ii. Zapytanie 2 – 10 rekordów zaczynając od jedenastego
 - iii. Zapytanie 3 – 10 rekordów zaczynając od dwudziestego pierwszego

Sprawdź się!

1. Jakie słowa kluczowe odpowiadają za określenie liczby rekordów do zwrócenia przez zapytanie, a jakie za opuszczenie pewnej liczby rekordów?
2. Jak zdefiniować wartości LIMIT i OFFSET jeśli chcesz wyświetlić 12 rekordów zaczynając od 53-go?

Propozycja rozwiązania

```
SELECT product_name, unit_price, quantity_per_unit FROM products LIMIT 10 OFFSET 0;  
SELECT product_name, unit_price, quantity_per_unit FROM products LIMIT 10 OFFSET 10;  
SELECT product_name, unit_price, quantity_per_unit FROM products LIMIT 10 OFFSET 20;
```

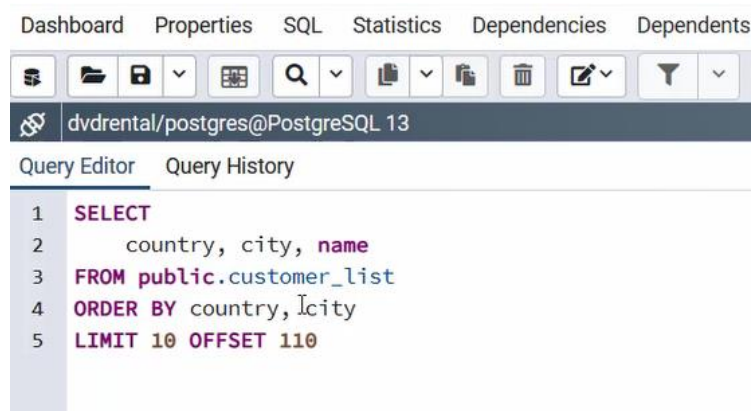
ORDER BY - sortowanie rekordów

Notatka:

- Domyślnie dane są zwracane przez serwer bez ustalonej kolejności
- Sortowanie można zmienić korzystając z klauzuli ORDER BY
- Sortować można wg jednej kolumny lub większej liczby kolumn
- Sortować można rosnąco (ASC) i malejąco (DESC)
- Sortując wg kilku kolumn, można każdą kolumnę niezależnie sortować malejąco lub rosnąco
- Sortować można wg kolumn, które nie są wyświetlane
- Jeśli stosujesz LIMIT/OFFSET to użyj też ORDER BY, aby zapewnić zdefiniowaną kolejność kolumn
- Składnia polecenia wygląda tak:

```
SELECT col1, col2,...  
FROM table  
ORDER BY col1 ASC, col2 DESC  
LIMIT 10 OFFSET 30
```

- Uwaga – zła praktyka: w order by można przekazać numery kolumn z listy SELECT. Sortowanie odbędzie się wg tych kolumn. Wiedz o tym, ale raczej nie używaj
- Na marginesie: stosując aliasy i nazwy zawierające spację, należy je umieszczać w cudzysłowie np. "zip code"



Laboratorium

1. Wiedząc o braku domyślnego sortowania wyników popraw zapytanie zaproponowane programiście w poprzednim zadaniu. Napisz zapytanie do tabeli **products**, które zwróci **product_name**, **unit_price**, **quantity_per_unit**, uporządkuje wg **product_name** i wyświetli 10 pierwszych rekordów
2. W aplikacji ma być wyświetlona alfabetyczna lista klientów. Wyświetl wszystkie informacje uporządkowane wg **company_name**
3. Przygotowujesz zestawienie stanów magazynowych. Wyświetl wszystkie informacje z tabeli **products** posortowane wg **units_in_stock** malejąco oraz **reorder_level** malejąco
4. Wyświetl wszystkie informacje o wszystkich zamówieniach z tabeli zamówień (**orders**) uporządkowane wg daty zamówienia (**order_date**)

Sprawdź się!

1. Jakie jest działanie polecenia ORDER BY a jakie ORDER BY ... ASC a jakie ORDER BY DESC ?
2. Tabela produkty zawiera między innymi kolumny Category i Price. Chesz posortować dane alfabetycznie wg Category, ale w ramach kategorii produkty mają być ułożone w kolejności malejącej wg ceny. Jak będzie wyglądała klauzula ORDER BY?
3. Co oznacza zapis ORDER BY 4,2

Propozycja rozwiązania

```
SELECT product_name, unit_price, quantity_per_unit FROM products
ORDER BY product_name LIMIT 10 OFFSET 0;

SELECT * FROM customers ORDER BY company_name;

SELECT * FROM products ORDER BY units_in_stock DESC, reorder_level DESC;

SELECT * FROM orders ORDER BY order_date
```


Agregowanie wartości

Notatka:

- Najpopularniejsze funkcje agregujące to
 - AVG – średnia
 - COUNT – policz
 - MIN, MAX – wartość minimalna, maksymalna
 - SUM – suma
- Jeśli chcesz policzyć, ile jest rekordów napisz

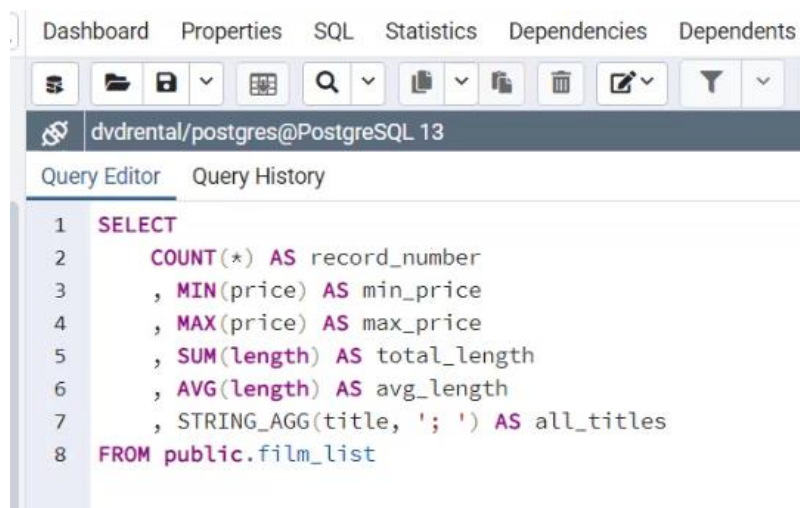
```
SELECT COUNT(*) FROM tabela
```

- W przypadku pozostałych funkcji agregujących, obliczenia wykonywane są na określonej kolumnie. Dlatego nazwę tej kolumny przekazuje się jako argument do funkcji agregującej np.

```
SELECT SUM(length) FROM film_list
```

- W jednym poleceniu można wyliczyć kilka różnych wartości agregujących
- Do połączenia napisów znajdujących się w wierszach określonej kolumny można skorzystać z funkcji str_aggregate, np.:

```
SELECT string_agg(title, ';' ) FROM film_list
```



Laboratorium

1. Firma consultingowa będzie doradzać w zakresie polityki cenowej. Ponieważ doradca nic nie wie o produktach twojej firmy przygotuj zapytanie, które zwróci z tabeli **products** informacje o:
 - a. Liczbie produktów
 - b. Średniej cenie produktów (kolumna **unit_price**)
 - c. Minimalnej cenie
 - d. Maksymalnej cenie
 - e. Sumie **units_in_stock**
2. Do folderu reklamowego potrzebna jest lista nazw produktów z tabeli **products** rozdzielona przecinkami (i spacją). Napisz zapytanie, które połączy **product_name** w taki długi napis

Sprawdź się!

1. Jakie funkcje pozwolą na
 - a. wyznaczenie liczby rekordów w tabeli
 - b. najmniejszej lub największej wartości w kolumnie
 - c. sumie wartości w kolumnie
 - d. średniej wartości w kolumnie
 - e. połączenia wartości tekstowych z wielu wierszy w jedno większe pole

Propozycja rozwiązania

```
SELECT
    COUNT(*) AS "Number of products"
    , AVG(unit_price) AS "Average price"
    , MIN(unit_price) AS "Min price"
    , MAX(unit_price) AS "Max price"
    , SUM(units_in_stock) AS "Total units"
FROM products;

SELECT
    string_agg(product_name, ', ')
FROM products;
```

Klauzula GROUP BY - tworzenie grup z agregacjami

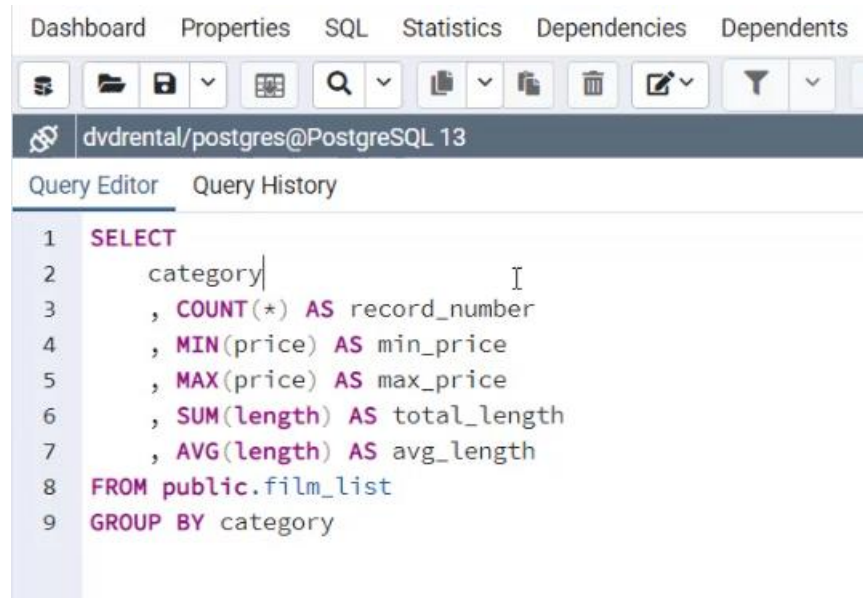
Notatka:

- Aby wykonać agregacje na mniejszych podzbiorach w tabeli stosuje się klauzulę GROUP BY
- W tym przypadku do listy SELECT dodaje się nazwę jednej lub większej liczby kolumn, których wartości dzielą zbiór na mniejsze grupy
- Wszystkie dodatkowe kolumny, które nie są argumentami dla funkcji agregujących muszą być wymienione w klauzuli GROUP BY
- Jeśli jakaś kolumna jest wymieniona w liście SELECT, ale nie jest argumentem funkcji agregującej ani nie jest wymieniona w GROUP BY, to dochodzi do błędu:

Column ... must appear in the GROUP BY clause be used in the aggregate function

- Składnia polecenia SELECT z GROUP BY wygląda tak:

```
SELECT category, SUM(length) AS total_length
FROM film_list
GROUP BY category
ORDER BY total_length DESC
```



Laboratorium

1. Napisz zapytanie do tabeli **customers**, które wyznaczy liczbę klientów z każdego kraju (kolumna **country**). Uporządkuj malejąco wg tej wyznaczonej liczby
2. Napisz zapytanie, które z tabeli **orders** wyznaczy liczbę zamówień wysyłanych do każdego kraju (kolumna **ship_country**). Uporządkuj malejąco wg tej wyznaczonej liczby.
3. Szukamy drogich dostawców. Napisz zapytanie, które podzieli tabelę **products** ze względu na dostawcę (kolumna **supplier_id**) i dla każdej grupy wyznaczy średnią cenę dostarczanych produktów (**unit_price**). Uporządkuj malejąco wg tej wyznaczonej liczby.
4. Do poprzedniego zapytania dodaj **category_id**, dzięki czemu będzie wiadomo jaka jest średnia cena produktów dostarczanych przez wybranego dostawcę w ramach określonej kategorii.

Sprawdź się!

1. Jak poradzić sobie z błędem „column xxx must appear in the GROUP BY clause or be used in an aggregate function”?
2. Produkty są przypisane do kategorii, podkategorii i typu. Chcesz wyznaczyć średnią cenę ze względu na kategorię. Jakie kolumny muszą się znaleźć w klauzuli SELECT, a jakie w GROUP BY?
3. Produkty są przypisane do kategorii, podkategorii i typu. Chcesz wyznaczyć średnią cenę ze względu na kategorię i podkategorię. Jakie kolumny muszą się znaleźć w klauzuli SELECT, a jakie w GROUP BY?
4. Czy zastosowanie w zapytaniu klauzuli GROUP BY automatycznie spowoduje, że dane zostaną posortowane?

Propozycja rozwiązań

```
SELECT country, COUNT(*) AS cnt
FROM customers
GROUP BY country
ORDER BY cnt DESC;
```

```
SELECT ship_country, COUNT(*) AS cnt
FROM orders
GROUP BY ship_country
ORDER BY cnt DESC;
```

```
SELECT supplier_id, AVG(unit_price) AS avg_price
FROM products
GROUP BY supplier_id
ORDER BY avg_price DESC;
```

```
SELECT category_id, supplier_id, AVG(unit_price) AS avg_price
FROM products
GROUP BY category_id, supplier_id
ORDER BY avg_price DESC
```

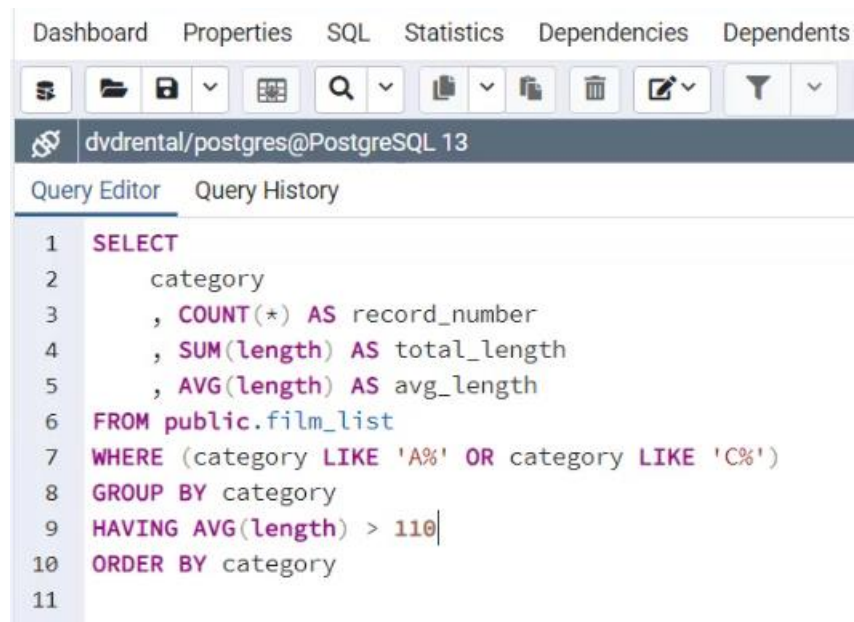
Klauzula HAVING - filtrowanie agregowanych grup

Notatka:

- Klauzula HAVING pozwala filtrować wynik zwrócony przez zapytanie korzystające z klauzuli GROUP BY.
- O ile klauzula WHERE filtruje dane przed przekazaniem danych do GROUP BY, o tyle HAVING to drugi filtr, który wybiera tylko te wyniki zwrócone przez GROUP BY, które spełniają określony warunek
- Częsty błąd, to umieszczenie warunku w WHERE zamiast w HAVING
- Składnia polecenia SELECT z GROUP BY i HAVING:

```
SELECT category, AVG(length) AS total_length
FROM film_list
WHERE category IN ('comedy', 'action', 'animated')
GROUP BY category
HAVING AVH(length) > 100.
ORDER BY total_length DESC
```

- Zapamiętaj kolejność słów w powyższym zapytaniu: SELECT, FROM, WHERE, GROUP BY, HAVING i ORDER BY
- Jeśli analizować by w jakiej kolejności te instrukcje są wykonywane, to kolejność będzie: FROM, WHERE, SELECT, GROUP BY, HAVING, ORDER BY



The screenshot shows a web-based PostgreSQL interface. At the top, there are tabs for 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', and 'Dependents'. Below these is a toolbar with icons for various database actions. The main area is titled 'Query Editor' and 'Query History'. The SQL query being edited is as follows:

```
1 SELECT
2     category
3     , COUNT(*) AS record_number
4     , SUM(length) AS total_length
5     , AVG(length) AS avg_length
6 FROM public.film_list
7 WHERE (category LIKE 'A%' OR category LIKE 'C%')
8 GROUP BY category
9 HAVING AVG(length) > 110
10 ORDER BY category
11
```

Laboratorium

1. Zastanawiamy się nad zapasami w magazynach. Z jakiej kategorii produktów mamy najwięcej zapasów. Napisz zapytanie do tabeli **products**, które wyświetli identyfikator kategorii (**category_id**) i sumę zapasów (**units_in_stock**). Wyświetl tylko te kategorie, w których suma jest większa lub równa 500.
2. W naszej międzynarodowej firmie przygotowujemy promocję i zastanawiamy się, jakie kraje są naszymi największymi odbiorcami. Napisz zapytanie, które z tabeli **customers** wyświetli kraj (**country**) i liczbę klientów. Wyświetl tylko te kraje, gdzie mamy co najmniej 10 klientów. Wynik uporządkuj wg tej liczby malejąco.
3. W ramach promocji przygotujemy spersonalizowaną ofertę próbek produktów dla przedstawicieli klientów. Żeby spersonalizować upominki, musimy wiedzieć na jakich stanowiskach pracują reprezentanci firm. Napisz zapytanie, które z tabeli **customers** zwróci nazwę stanowiska (**contact_title**) i ilość osób. Wyświetl tylko te stanowiska, na których jest co najmniej 5 osób. Dane uporządkuj wg tej liczby malejąco

Sprawdź się!

1. Uporządkuj słowa kluczowe wg kolejności zapisu poprawnego polecenia SELECT
 - a. SELECT
 - b. HAVING
 - c. GROUP BY
 - d. WHERE
 - e. ORDER BY
 - f. FROM
2. Uporządkuj te słowa wg kolejności wykonywania polecenia SELECT
3. Czy stosowanie kryteriów w WHERE I HAVING jest równoważne?

Propozycja rozwiązania

```
SELECT
    p.category_id, SUM(units_in_stock)
FROM products p
GROUP BY p.category_id
HAVING SUM(units_in_stock) >= 500;

SELECT
    c.country, count(*) FROM customers c
GROUP BY c.country
HAVING count(*) >=10
ORDER BY count(*) DESC;

SELECT
    c.contact_title, count(*) FROM customers c
GROUP BY c.contact_title
HAVING count(*) >= 5
ORDER BY count(*) DESC
```

Klauzula BETWEEN zamiast operatorów porównań

Notatka:

- BETWEEN nie jest kluczowym elementem języka SQL, ale pozwala uprościć zapis eliminując dwa porównania
- Poniższe dwa warunki są równoważne:

```
length >= 100 AND length <= 110
```

```
length BETWEEN 100 AND 110
```

- BETWEEN dokonuje porównania „miękkiego” (mniejsze równe i większe równe). Nie jest to porównanie „ostre” (mniejsze i większe)
- Można obejść problem „miękości” porównania BETWEEN poprzez określenie wartości brzegowych jako o jedną jednostkę mniejszą lub większą. Do takiego zdefiniowania wartości brzegowych trzeba mieć wiedzę na temat rodzaju przechowywanych danych. Jeśli np. kolumna length jest liczbą całkowitą, to zadziała warunek

```
length BETWEEN 101 AND 109
```

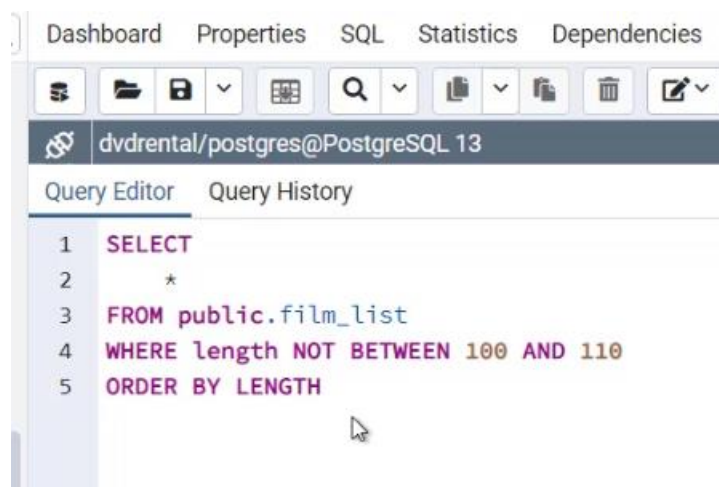
ale jeśli length to liczba dziesiętna z dwoma miejscami po przecinku, to użyj:

```
length BETWEEN 100.001 AND 109.991
```

- BETWEEN można łączyć z NOT na dwa sposoby –poniższe zapisy są równoważne:

```
NOT length BETWEEN 100 AND 110
```

```
length NOT BETWEEN 100 AND 110
```



Laboratorium

1. Kontrola Urzędu Skarbowego prosi o przedstawienie informacji o zamówieniach z numerami od **10400** do **10500**. Napisz zapytanie do tabeli **orders**, które zwróci rekordy z identyfikatorem zamówienia (**order_id**) z wybranego przez Urząd Skarbowy zakresu.
2. Przeprowadzasz analizę sprzedaży za 1997 rok (o rety kiedy to było 😊). Z tabeli **orders** wyświetl te zamówienia, których data zamówienia (**order_date**) jest z 1997 roku. Skorzystaj z operatora BETWEEN
3. Analityk przegląda informacje o cenach produktów porcjami, bazując na cenie. Najpierw chce zobaczyć produkty o cenie do 10 włącznie, potem od 10 (bez 10) do 20 włącznie itd. Napisz zapytanie, które z tabeli **products** wyświetli produkty o cenie (**unit_price**) ostro większej od 10 i mniejszej równej 20. Mimo tego, że pewnie korci Cię w tym momencie użycie zwykłych operatorów porównania, spróbuj skorzystać z odpowiedniego wyrażenia BETWEEN.

Sprawdź się!

1. Kolumna Price zawiera cenę produktu (wartość określona z dokładnością do groszy). Jak inaczej można by zapisać warunki
 - a. Price >=10 AND Price <= 20
 - b. Price > 10 AND Price < 20
 - c. Price > 10 AND Price <= 20
 - d. Price < 10 AND Price > 20
 - e. Price <= 10 AND Price >= 20

Propozycja rozwiązania

```
SELECT * FROM orders WHERE order_id BETWEEN 10400 AND 10500;
SELECT * FROM orders WHERE order_date BETWEEN '1997-01-01' AND '1997-12-31';
SELECT * FROM products WHERE unit_price BETWEEN 10.001 AND 20 ORDER BY unit_price;
```


Operator IN - zamiast wielu OR

Notatka:

- IN nie jest kluczowym elementem języka SQL, ale pozwala uprościć zapis eliminując wyrażenia logiczne OR/AND
- Poniższe dwa warunki są równoważne:

```
rating = 'R' OR rating = 'G' OR rating = 'PG'
```

```
rating IN ('R', 'G', 'PG')
```

- Operator IN można łączyć z operatorem NOT na dwa sposoby. Poniższe polecenia są równoważne:

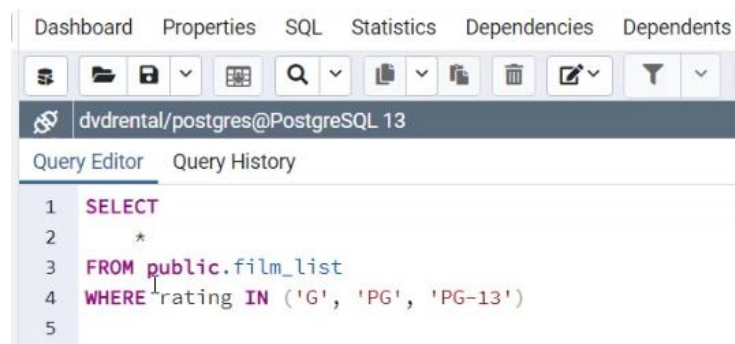
```
NOT rating IN ('R', 'G', 'PG')
```

```
rating NOT IN ('R', 'G', 'PG')
```

- Operator NOT IN pozwala wykluczyć wiele wyrażen AND z zaprzeczeniem:

```
rating <> 'R' AND rating <> 'G' AND rating <> 'PG'
```

```
rating NOT IN ('R', 'G', 'PG')
```



Laboratorium

- Wyświetl te rekordy z tabeli **customers**, które w kolumnie **city** mają Berlin lub London lub Madrid
- Wyświetl te rekordy z tabeli **customers**, które w kolumnie **country** mają wartość inną niż Brazil, USA i Venezuela
- Wyświetl te rekordy z tabeli **products**, które w kolumnie **category_id** mają inną wartość niż 1, 2 lub 3
- Do poprzedniego zapytania dodaj kolejny warunek: i w kolumnie **quantity_per_unit** pojawia się jedna z wartości (dla Twojej wygody zapis gotowy do przeklejenia do zapytania): '25 - 825 g cans', '4 - 450 g glasses', '12 - 12 oz cans', '10 - 500 g pkgs.'

Sprawdź się!

- Ilu operatorów OR trzeba by użyć do zapisania wyrażenia równoważnego:

```
plan IN ('basic', 'basic plus', 'standard', 'standard plus')
```

- Jak można by inaczej zapisać warunek:

```
plan <> 'basic' AND plan <> 'basic plus' AND plan <> 'standard'  
AND plan <> 'standard plus'
```

Propozycja rozwiązania

```
SELECT * FROM customers WHERE city IN ('Berlin', 'London', 'Madrid');  
SELECT * FROM customers WHERE country NOT IN ('Brazil', 'USA', 'Venezuela');  
SELECT * FROM products WHERE category_id NOT IN (1,2,3);  
SELECT * FROM products WHERE category_id NOT IN (1,2,3) AND  
quantity_per_unit IN ('25 - 825 g cans', '4 - 450 g glasses', '12 - 12 oz cans', '10 -  
500 g pkgs.')
```

Korzystanie z funkcji i obliczeń

Notatka:

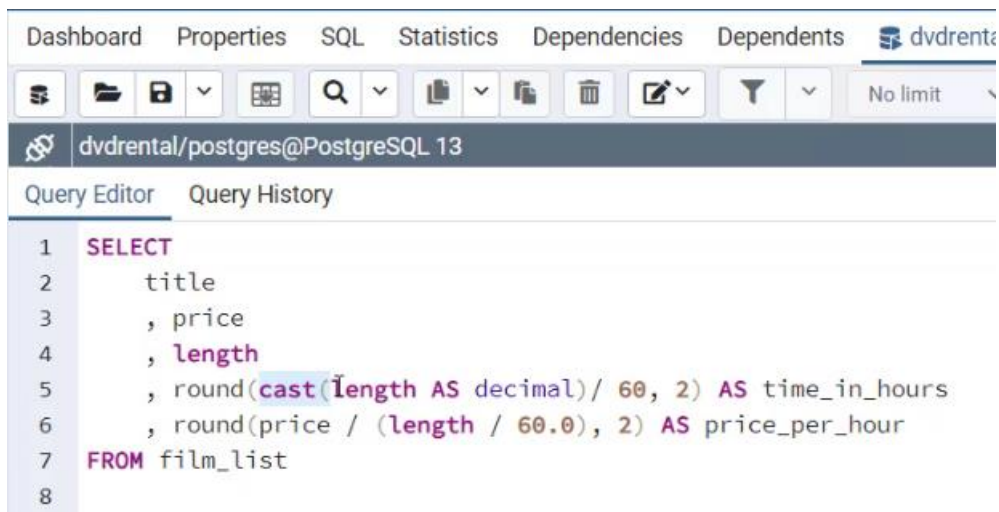
- W klauzuli SELECT można umieścić nie tylko same nazwy kolumn tabeli, ale również wyrażenia
- Wyrażenia wyliczane mogą też być umieszczane w innych częściach zapytania, np. WHERE lub ORDER BY
- Domyślnie kolumny wyliczane nie mają nazwy, więc należy je aliasować
- Jeśli wynik wyrażenia jest niepoprawnie konwertowany do określonego typu, to można użyć funkcji CAST albo w obliczeniach użyć wartości innego typu dzięki czemu automatycznie wykonywana konwersja zostanie zmieniona, np.:

```
cast(length/60 AS decimal)
Length/60.0
```

- Do zaokrąglenia wartości można użyć funkcji round:

```
round(length/60.0, 2)
```

- Aliasy kolumn wyliczanych mogą być wykorzystywane w ORDER BY
- Aliasy kolumn wyliczanych NIE mogą być wykorzystywane w WHERE ani w innych wyrażeniach w SELECT
- Powyższe uwagi o wykorzystaniu aliasów w WHERE i ORDER BY biorą się z kolejności wykonywania operacji: FROM, WHERE, SELECT, GROUP BY HAVING, ORDER BY



The screenshot shows a database query editor interface. At the top, there are tabs for 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', and 'Dependents'. Below these is a toolbar with various icons for file operations and search. The main area displays a SQL query in a 'Query Editor' tab. The query is as follows:

```
1 SELECT
2     title
3     , price
4     , length
5     , round(cast(length AS decimal) / 60, 2) AS time_in_hours
6     , round(price / (length / 60.0), 2) AS price_per_hour
7 FROM film_list
8
```

Laboratorium

W tym zadaniu będziemy pracować na tabeli **order_details** zawierającej szczegóły realizowanego zamówienia. Kolumna **unit_price** zawiera cenę produktu, **quantity** ilość kupowanych produktów, a **discount** procentowy rabat. Podejrzewamy, że pracownicy udzielali zbyt wysokich rabatów. Gdzie się da korzystaj z aliasów, a gdzie się nie da... no cóż... welcome in hell!

1. Wyświetl wszystkie rekordy z tabeli **order_details**
2. Dodaj wyrażenie wyznaczające wartość pozycji, jako cena razy ilość. Wykonaj konwersję zwracanego wyniku na decimal. Zaokrąglij wynik do 2 miejsc po przecinku. Zaaliasuj wyrażenie jako **total**
3. Dodaj wyrażenie wyznaczające wartość pozycji po rabacie. W tym celu pomnóż cenę przez ilość i przez **(1 – discount)**. Podobnie jak poprzednio dokonaj konwersji na decimal, zaokrąglij z dokładnością do 2 miejsc po przecinku i zaaliasuj jako **total_after_discount**.
4. Dodaj kolumnę, w której wyliczysz wartość udzielonego rabatu. W tym celu od wyrażenia z pkt. 3 odejmij wartość z wyrażenia pkt 2. Zaaliasuj ją jako **discount_value**
5. Wyświetl tylko te rekordy, gdzie wartość udzielonego rabatu wynosi co najmniej 100
6. Uporządkuj dane wg wielkości udzielonego rabatu malejąco
7. I jak Ci się podoba ten potworek, którego napisałeś/aś 😊?

Sprawdź się!

1. Jaka funkcja służy do zmiany typu zmiennej, np. z int na decimal?
2. Jaką znasz sztuczkę dotyczącą „niejawnej” konwersji typu wyniku uzyskiwanego dla funkcji algebraicznych?
3. Jaka funkcja pozwoli zaokrąglić wynik?
4. Czy aliasy zdefiniowane w SELECT mogą być wykorzystywane w ORDER BY?
5. Czy aliasy zdefiniowane w SELECT mogą być wykorzystywane w WHERE?

Propozycja rozwiązania

```
SELECT *
  , round(CAST(unit_price * quantity AS decimal), 2) as total
  , round(CAST(unit_price * quantity * (1- discount) AS decimal ), 2) as
    total_after_discount
  , round(CAST(unit_price * quantity AS decimal), 2) -
    round(CAST(unit_price * quantity * (1- discount) AS decimal ), 2) AS
    discount_value
FROM order_details
WHERE
round(CAST(unit_price * quantity AS decimal), 2) -
round(CAST(unit_price * quantity * (1- discount) AS decimal ), 2) >= 100
ORDER BY discount_value DESC;
```

NULL czyli brak wartości

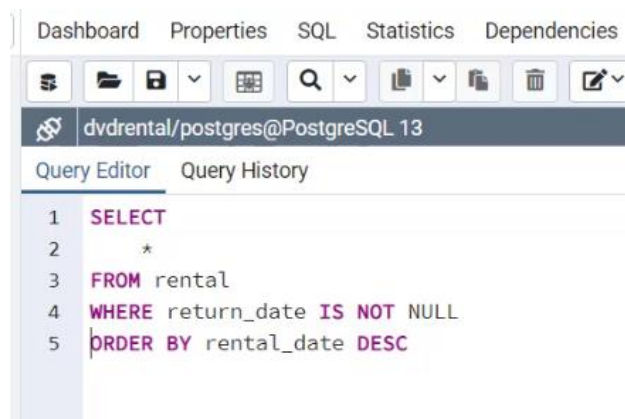
Notatka:

- Brak wartości w SQL jest reprezentowany przez NULL
- Projekt tabeli musi przewidywać fakt, że w określonej kolumnie jest dopuszczalna wartość NULL. Istnieją różne szkoły dopuszczania wartości NULL – jedni stosują ją tylko tam, gdzie to niezbędne, inni pozwalają jej używać gdy się tylko da
- Porównania do wartości NULL wykonuje się poprzez wyrażenie IS NULL (zwykły operator „=” zwróciłby wartość UNKNOWN). Operator może być łączony z NOT:

```
return_date IS NULL
```

```
return_date IS NOT NULL
```

- Agregacja COUNT(*) zwraca liczbę rekordów, ale...
 - ...COUNT przyjmujące jako argument nazwę kolumny np. COUNT(return_date), liczy tylko ile jest wierszy, w których return_date nie jest NULL
 - Podobnie funkcje SUM, AVG, MIN, MAX – opuszczają wiersze, w których przekazany argument ma wartość NULL
 - Jeśli z wykorzystaniem funkcji COUNT chcesz poznać liczbę wartości unikalnych w danej kolumnie to skorzystaj z COUNT(DISTINCT customer_id)
- Na marginesie – kiedy chcesz utworzyć tabelę tymczasową, to służy do tego polecenie CREATE TEMPORARY TABLE, a do usuwania tabeli służy polecenie DROP – więcej o tabelach tymczasowych w dalszej części kursu



Laboratorium

1. Czy są jakieś niewysłane zamówienia? Wyświetl wszystkie rekordy tabeli **orders**, gdzie **shipped_date** jest puste
2. Czy są jakieś niepoprawne rekordy w tabeli **orders**, w których data zamówienia (**order_date**) jest pusta? Wyświetl je (jeśli takie są),
3. Czy wszyscy klienci mają fax? Wyświetl rekordy z tabeli **customers**, w których **fax** jest pusty.
4. Ile rekordów w tabeli **customers** nie ma wypełnionego pola **region**?
5. Ile jest unikalnych wartości w tabeli **customers** w kolumnie **country**, wśród klientów, którzy nie mają wartości w kolumnie **region**?

Sprawdź się!

1. Twoim zdaniem (i nie ma niepoprawnych odpowiedzi), następujące kolumny powinny przyjmować wartość NULL, czy nie:
 2. - dla lotu samolotu – data i godzina odlotu
 3. - dla lotu samolotu – data i godzina przylotu
 4. - dla obywatela – data urodzenia
 5. - dla nieruchomości – powierzchnia użytkowa lokalu
 6. - dla nieruchomości – powierzchnia przynależącego parkingu
 7. - dla obywatela – płeć
 8. - dla pacjenta – czy pacjent przechodził COVID
 9. - dla wyposażenia – kolor farby
10. Jak sprawdzić czy w kolumnie **is_finished** jest wartość NULL?
11. Jak sprawdzić czy w kolumnie **is_finished** jest wartość nie NULL?
12. W kolumnie **is_finished** 300 wierszy ma wartość True, 200 wierszy wartość False, a 50 nie ma wprowadzonej wartości. Co zwróci funkcja **COUNT(*)**, a co **COUNT(is_finished)**, a co **COUNT(DISTINCT is_finished)** ?

Propozycja rozwiązania

```
SELECT * FROM orders WHERE shipped_date IS NULL;
SELECT * FROM orders WHERE order_date IS NULL;
SELECT * FROM customers WHERE fax IS NULL;
SELECT COUNT(*) FROM customers WHERE region IS NULL;
SELECT COUNT(DISTINCT country) FROM customers WHERE region IS NULL;
```

Eliminacja NULL - funkcja COALESCE

Notatka:

- Null powoduje problemy z powodu specyficznego sposobu pracy z tymi wartościami przez funkcje języka SQL
- Gdy chcesz podmienić istniejące w tabeli wartości na inne, możesz korzystać z wyrażenia CASE

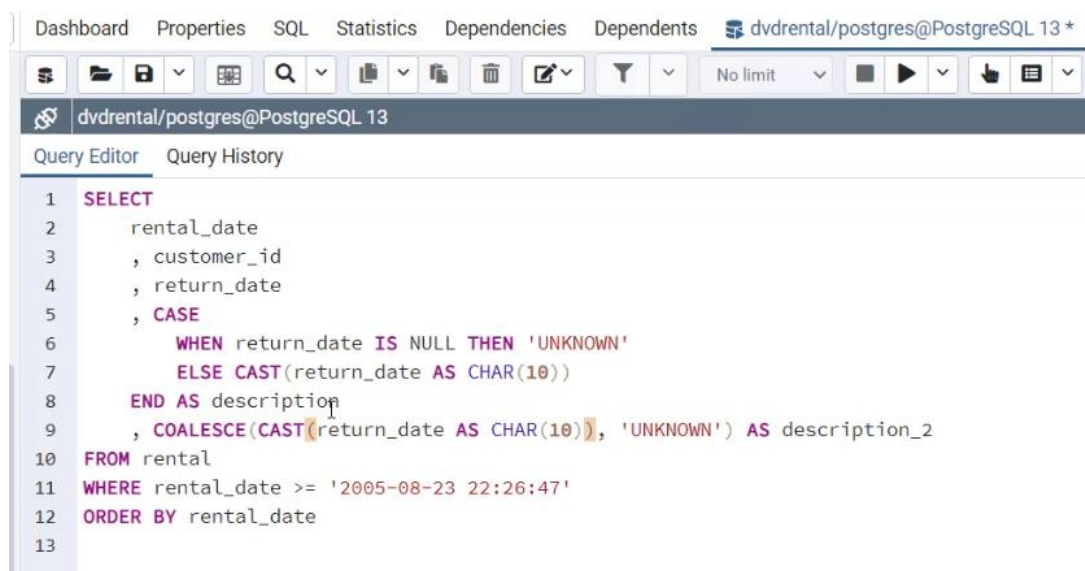
```
CASE
WHEN day IN (1,2,3,4,5) IS NULL THEN 'workday'
WHEN day IN (6,7) THEN 'weekend'
ELSE 'unknown'
END
```

- Wartości zwracane przez CASE muszą być tego samego typu
- Wyszczególniona funkcja do eliminacji NULL to COALESCE
- Funkcja COALESCE przyjmuje wiele argumentów i zwraca pierwszy nie-nullowy
- Wartości zwracane przez COALESCE muszą być tego samego typu
- Polecenie:

```
COALESCE(city, region, country, 'unknown')
```

zwróci city o ile city nie jest null, a jeśli jest,

- to region o ile nie jest null, a jeśli jest,
 - to country o ile nie jest null, a jeśli jest,
 - to napis 'unknown'



```
Dashboard Properties SQL Statistics Dependencies Dependents dvdrental/postgres@PostgreSQL 13 *
dvdrental/postgres@PostgreSQL 13
Query Editor Query History
1 SELECT
2     rental_date
3     , customer_id
4     , return_date
5     , CASE
6         WHEN return_date IS NULL THEN 'UNKNOWN'
7         ELSE CAST(return_date AS CHAR(10))
8     END AS description
9     , COALESCE(CAST(return_date AS CHAR(10)), 'UNKNOWN') AS description_2
10 FROM rental
11 WHERE rental_date >= '2005-08-23 22:26:47'
12 ORDER BY rental_date
13
```

Laboratorium

1. Z tabeli **suppliers** wyświetl:
 - a. **company_name**
 - b. **region**, a jeśli **region** jest **NULL**, to **country** – zaaliasuj kolumnę **region_country**
 - c. **fax**, a jeśli **fax** jest **NULL**, to **phone** – zaaliasuj kolumnę **fax_phone**
 - d. **homepage**, a jeśli **homepage** jest **NULL**, to napis 'no website'
2. Z tabeli **orders** wyświetl:
 - a. **Order_id**
 - b. **shipping_date**, a jeśli **shipping_date** jest **NULL**, to napis "not shipped yet". Tu będzie potrzebna konwersja daty (**shipping_date**) na typ **VARCHAR(10)**. Zaaliasuj kolumnę **shipping_info** i posortuj po niej malejąco
3. Z tabeli **products** wyświetl nazwę produktu (**product_name**), kategorię (**category_id**) oraz w zależności od wartości **category_id**: jeśli to jest **1** to napis "Beverages", jeśli to **2**, to "Condiments", jeśli to **3** to "Confections", w przeciwnym wypadku napis 'other'

Sprawdź się!

1. W kolumnie **type** znajduje się wartość liczbową, która ma specjalne znaczenie określające typ kontraktu: 1 – BASIC, 2 – STANDARD, 3 – PREMIUM, 4 – ENTERPRISE. Jak w zapytaniu w locie zamieniać wartość 1, 2, 3, 4 na odpowiedni napis?
2. Bazując na danych z poprzedniego przykładu, jak można by obsłużyć przypadek pojawienia się w kolumnie **type** jeszcze jakiejś innej wartości np. 5, 6 itd.? Chcielibyśmy, aby w takim przypadku pojawił się napis „OTHER”.
3. Obsługujesz bazę z ocenami i recenzjami klientów sklepu. W niektórych rekordach jest zapisana tylko ocena, a pole kolumna **comments** jest puste **NULL**. Twoim zadaniem jest wymyślenie sposobu na zamianę pustego komentarza na tekst „no additional comments”

Propozycja rozwiązania

```
SELECT
    company_name
    , COALESCE(region, country) AS region_country
    , COALESCE(fax, phone) as fax_or_phone
    , COALESCE(homepage, 'no website') as website
FROM suppliers;

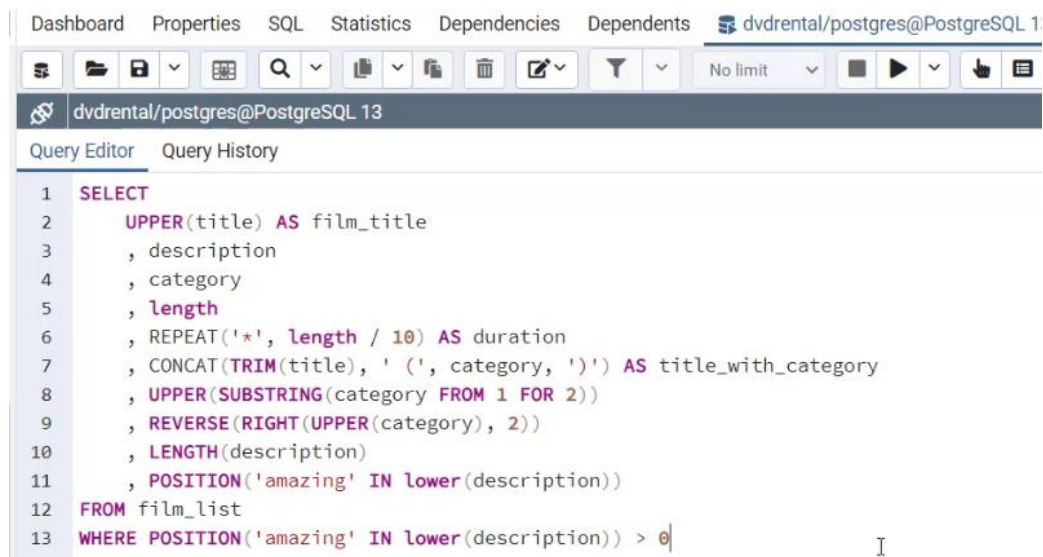
SELECT
    order_id
    , COALESCE(CAST(shipped_date AS VARCHAR(10)), 'not-shipped-yet') as
                                                shipping_info
FROM orders ORDER BY shipping_info DESC;

SELECT product_name, category_id,
CASE
    WHEN category_id = 1 THEN 'Beverages'
    WHEN category_id = 2 THEN 'Condiments'
    WHEN category_id = 3 THEN 'Confections'
    ELSE 'Other'
END
FROM products;
```


Funkcje tekstowe

Notatka:

- Postgres oferuje dużo funkcji pracujących z tekstami, a te najpopularniejsze to:
 - **LOWER(title)** – zmienia wielkość liter na małe
 - **UPPER(title)** – zmienia wielkość liter na wielkie
 - **REPEAT('*', length/10)** – powtarza znak określoną ilość razy
 - **CONCAT(title, '/', category)** – skleja ze sobą napisy
 - **TRIM(title)** – usuwa „białe” znaki
 - **SUBSTRING(category FROM 1 FOR 2)** – wybiera z napisu category podnapis zaczynający się na 1-szym znaku i o długości 2-óch znaków
 - **LEFT(category, 2)** – zwraca 2 pierwsze znaki z napisu category
 - **RIGHT(category, 2)** – zwraca 2 ostatnie znaki z napisu category
 - **REVERSE(category)** – odwraca napis (od końca do początku)
 - **LENGTH(category)** – zwraca długość napisu
 - **POSITION('amazing' IN LOWER(description))** – zwraca pozycję podnapisu 'amazing' w kolumnie tekstowej description
- Pamiętaj o aliasowaniu kolumn wyliczanych
- Wynik działań i obliczeń na wartościach, spośród których niektóre są NULL, daje wynik NULL
- Funkcje można ze sobą składać
- Funkcje mogą występować w klauzuli SELECT, ale też np. w WHERE



```
1 SELECT
2     UPPER(title) AS film_title
3     , description
4     , category
5     , length
6     , REPEAT('*', length / 10) AS duration
7     , CONCAT(TRIM(title), ' (' , category, ')') AS title_with_category
8     , UPPER(SUBSTRING(category FROM 1 FOR 2))
9     , REVERSE(RIGHT(UPPER(category), 2))
10    , LENGTH(description)
11    , POSITION('amazing' IN lower(description))
12 FROM film_list
13 WHERE POSITION('amazing' IN lower(description)) > 0
```

Laboratorium

1. Z tabeli **customers** wyświetl:
 - a. **company_name** – wielkimi literami
 - b. **title** małymi literami
 - c. 3 pierwsze litery z **company_name** połączone z 3 ostatnimi z **city**
2. Teraz będzie trochę trudno, ale właśnie takie problemy czasami trzeba rozwiązać... chcemy „rozbić” kolumnę **address** ze względu na przecinek. To co jest przed przecinkiem ma być zaaliasowane jako **address_1**, a to co jest po przecinku ma mieć alias **address_2**. Jeśli w adresie nie ma przecinka, to **address_1** ma mieć wartość **address**, a **address_2** ma być pustym napisem.
 - a. Dla **address_1**: zbuduj **CASE/WHEN/ELSE/END**, który zbada czy w **address** znajduje się przecinek. Jeśli TAK, to wytnij z **address** tekst od pierwszego znaku do pierwszego przecinka w napisie (**POSITION** -1). Jeśli NIE to zwróć **address**
 - b. Dla **address_2**: zbuduj **CASE/WHEN/ELSE/END**, który zbada czy w **address** znajduje się przecinek. Jeśli TAK, to wytnij tekst z **address** od pierwszego przecinka (**POSITION** + 1) do końca napisu (długość napisu - pozycja znaku pierwszego przecinka w napisie (**LENGTH** - **POSITION**)). Jeśli NIE to zwróć pusty napis.

Sprawdź się!

Jak nazywają się funkcje, które:

- Zamieniają tekst na małe litery
- Zamieniają tekst na duże litery
- Powtarza określony znak kilka razy
- Łączy napisy
- Usuwa „białe znaki” na początku i końcu
- Wycina z napisu pewien „podnapis”
- Wycina z napisu pewną liczbę znaków z lewej lub prawej strony
- Odwraca napis od końca do początku
- Zwraca długość napisu
- Znajduje pozycję, na której w jednym napisie znajduje się drugi

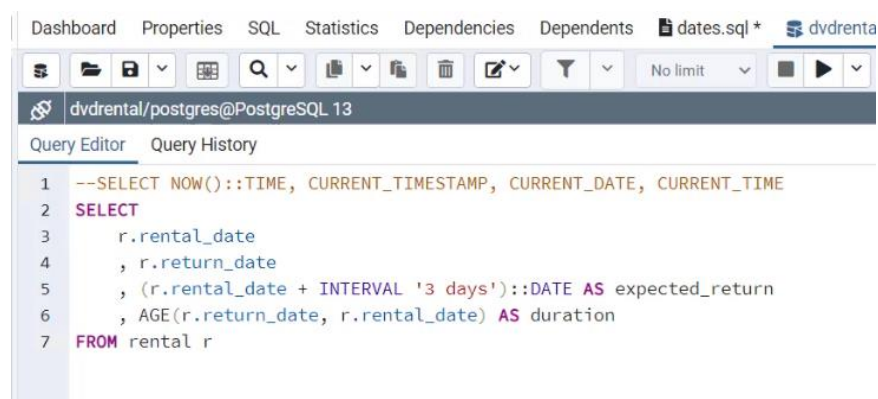
Propozycja rozwiązania

```
SELECT
  UPPER(company_name) AS company
  , LOWER(contact_title) AS title
  , CONCAT(LEFT(company_name,3), RIGHT(city, 3)) AS code
  , CASE
      WHEN POSITION(',') IN address > 0
      THEN SUBSTRING(address FROM 1 FOR POSITION(',')-1)
      ELSE address
    END AS address_1
  , CASE
      WHEN POSITION(',') IN address > 0
      THEN SUBSTRING(address FROM POSITION(',')+1
        FOR LENGTH(address) - POSITION(',') IN address))
      ELSE ''
    END AS address_2
FROM customers;
```

Funkcje daty i czasu

Notatka:

- W Postgres mamy do dyspozycji kilku typów czasu: **date** (tylko data), **time** (tylko czas), **timestamp** (data i czas), **timestamp with time zone** (data i czas ze strefą czasową), **interval** (różnica czasu)
- Funkcje wspierające pracę z datą i czasem to między innymi:
 - **NOW()** – zwraca czas bieżący (timestamp with timezone)
 - **NOW()::DATE** – tylko bieżąca data
 - **NOW()::TIME** – tylko bieżący czas
 - **CURRENT_TIMESTAMP** – zwraca bieżącą datę i czas (typ timestamp)
 - **CURRENT_DATE** – tylko bieżąca data
 - **CURRENT_TIME** – tylko bieżący czas
 - **DATE_TRUNC('day', rental_date)** – zwraca datę "ściętą" do dnia
 - **DATE_TRUNC('month', rental_date)** – zwraca datę "ściętą" do miesiąca
 - **DATE_TRUNC('year', rental_date)** – zwraca datę "ściętą" do roku
 - **DATE_PART('day', rental_date)** – wycina z daty numer dnia, można użyć też 'month', 'year'
 - **DATE_PART('dow', rental_date)** – numer dnia tygodnia (0=niedziela)
 - **EXTRACT('dow' FROM rental_date)** – numer dnia tygodnia (0=niedziela) – można też użyć 'year', 'month', 'day'
 - **rental_date + INTERVAL '3 days'** – dodawanie dni do daty, można użyć też '3 day', '3 months', słowo INTERVAL można opuścić
 - **AGE(return_date, rental_date)** – odejmowanie dat – wynik to interwał
 - **DATE '2030-05-01'** – konwersja napisu do daty, podobnie konwersja do time lub timestamp
- Funkcje daty i czasu można zagnieżdżać, co działa zresztą dla wszystkich funkcji



```
1  --SELECT NOW()::TIME, CURRENT_TIMESTAMP, CURRENT_DATE, CURRENT_TIME
2  SELECT
3      r.rental_date
4      , r.return_date
5      , (r.rental_date + INTERVAL '3 days')::DATE AS expected_return
6      , AGE(r.return_date, r.rental_date) AS duration
7  FROM rental r
```

Laboratorium

Poniższe zadania dotyczą tabeli **orders**.

1. Ile czasu trwa przetwarzanie zamówienia? Wyświetl datę zamówienia (**order_date**), datę wysyłki (**shipped_date**) oraz różnicę między tymi dwiema datami.
2. A średnio, ile czasu trwa realizacja zamówienia? Wyświetl średnią różnicę czasu między **shipped_date**, a **order_date**.
3. Wyświetl te zamówienia, które nie mają wypełnionej **shipped_date**. Chcemy przeprowadzić symulację „jak by to było, gdyby dzisiaj wysłać do tej pory niewysłane zamówienia”. Wyświetl datę dzisiejszą, **order_date** i różnicę między nimi.
4. Oj. Niewysłane zamówienia mają już po kilkanaście lat. Datę dzisiejszą z symulacji z pkt. 3 zamień na datę 2000-01-01
5. Po rozmowie z kierownikiem działu zamówień, okazało się, że ze 100% pewnością można przyjąć, że jeśli **shipped_date** jest **NULL**, to jest to błąd w danych i można przyjąć, że zamówienie było wysłane w ciągu miesiąca od zamówienia. Wyświetl **order_date** i wyliczoną kolumnę **default_ship_date** wyznaczoną jako: **ship_date** (jeśli **ship_date** jest znane) lub **order_date** powiększone o 1 miesiąc (gdy **ship_date** jest **NULL**)
6. Analityk na potrzeby raportu potrzebuje zestawienia zamówień zawierających rok, miesiąc i dzień zamówienia w osobnych kolumnach zapisanych jako liczby całkowite. Napisz zapytanie, które oprócz daty zamówienia (**order_date**) zwróci w osobnych kolumnach rok, miesiąc i dzień.

Sprawdź się!

1. Jakie mamy typy danych związane z datą lub czasem?
2. Jaka funkcja zwraca czas bieżący?
3. Jak z timestamp wyłuskać tylko samą datę? A jak tylko sam czas?
4. Jak z daty „wyłuskać” wyłącznie numer dnia, miesiąca i roku?
5. Co jest wynikiem dodawania interwału do daty?
6. Jak skonwertować datę zapisaną w postaci napisu 2030-05-01 na typ DATE?
7. Jak wyznaczyć różnicę między dwiema datami?

Propozycja rozwiązania

```
SELECT order_date, shipped_date, AGE(shipped_date, order_date) AS process_time
FROM orders;

SELECT AVG(AGE(shipped_date, order_date)) AS avg_process_time FROM orders;

SELECT NOW()::DATE AS today, order_date, AGE(NOW()::DATE, order_date) AS time_to_today
FROM orders WHERE shipped_date IS NULL;

SELECT DATE '2000-01-01' AS close_day, order_date,
       AGE(DATE '2000-01-01', order_date) AS time_to_close_day
FROM orders WHERE shipped_date IS NULL;

SELECT order_date
       , COALESCE(shipped_date, order_date + INTERVAL '1 month') AS default_ship_day
FROM orders;

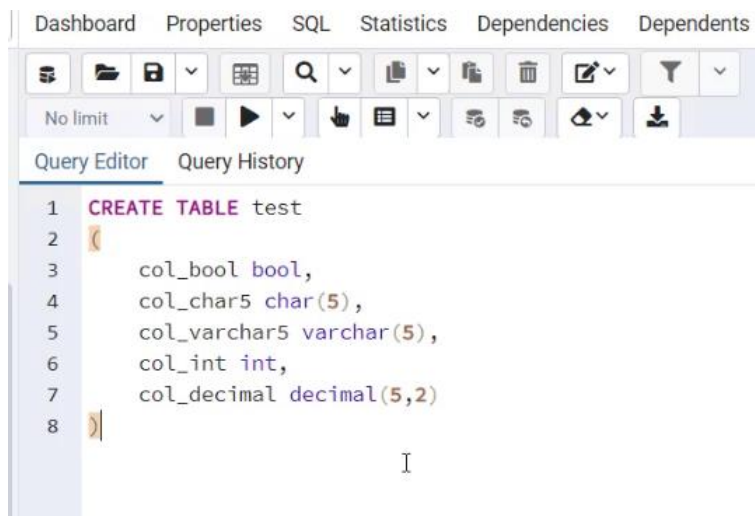
SELECT order_date, EXTRACT('year' FROM order_date) AS year,
       EXTRACT('month' FROM order_date) AS month,
       EXTRACT('day' FROM order_date) AS day
FROM orders;
```

Przegląd typów danych

Notatka:

- Typ definiuje rodzaj danych, jaki jest przyjmowany dla kolumny danych, pozwala korzystać z określonych funkcji, dokonywać złączeń danych z różnych tabel, poprawia wydajność i wykorzystanie zasobów
- Postgres słynie z dostępności wielu typów danych. Te najważniejsze to:
 - **boolean, bool** - typ logiczny. Przyjmuje wartości True/False, 'yes'/'no', 'y'/'n'
 - **character, char, character varying, varchar** – typ znakowy
 - **char(5)** – tekst do 5 liter, zawsze ma rozmiar 5 (stały rozmiar)
 - **varchar(5)** – tekst do 5 liter, rozmiar będzie <=5 (rozmiar dynamiczny)
 - **timestamp, time, date, interval** – typy związane z czasem
 - **integer, int** – liczba całkowita
 - **decimal(5,2)** – liczba dziesiętna (5 cyfr w tym 2 po przecinku)
- Przy okazji – polecenie tworzące tabelę to

```
CREATE TABLE table_name
(
  id INT,
  name VARCHAR(50),
  is_active BOOL,
  score_percent DECIMAL(3,2)
)
```



Laboratorium

1. Utwórz nową tabelę **books**, a w niej kolumny:
2. Identyfikator książki - **book_id** – liczba całkowita
3. Tytuł książki – **title** – napis o zmiennej długości, maksymalnie 100 znaków
4. Autor – **author** – napis o zmiennej długości, maksymalnie 50 znaków
5. Liczba stron – **pages** – liczba całkowita
6. Data wydania książki – **year_printed** – liczba całkowita
7. Procent ukończenia czytania książki – **percent_finished** – liczba dziesiętna, 5 cyfr w tym 2 po przecinku
8. Czy książka jest „tylko dla dorosłych” – **adults_only** – typ logiczny tak/nie
9. Wprowadź do tabeli kilka rekordów, np.:
10. 'Frankenstein', 'Mary Shelley', 289 stron, wydana w 2007, przeczytana w 100%, bez ograniczeń wiekowych
11. 'Tales of Terror and Madness', 'Edgar Allan Poe', 188 stron, wydana w 2001, przeczytana tylko w 45.33%, tylko dla dorosłych
12. Popętniaj błędy, prowokuj komunikaty o błędach. Na koniec usuń tabelę korzystając z polecenia

```
DROP table books;
```

Sprawdź się!

1. Jaki typ danych zaproponujesz dla:
 - a. liczba określająca stężenie alkoholu w napoju alkoholowym
 - b. wiek samochodu
 - c. nawa produktu w Twoim ulubionym supermarkecie
 - d. kod województwa
 - e. numer ISBN książki (masz chyba jakąś pod ręką?)
 - f. procentowa wartość zysku/straty gracza giełdowego

Propozycja rozwiązania

```
CREATE TABLE books
(
  book_id INT,
  title VARCHAR(100),
  author VARCHAR(50),
  pages INT,
  year_printed INT,
  percent_finished DECIMAL(5,2),
  adults_only BOOL
);

INSERT INTO books
VALUES (1, 'Frankenstein', 'Mary Shelley', 289, 2007, 100, FALSE);

INSERT INTO books
VALUES (2, 'Tales of Terror and Madness', 'Edgar Allan Poe', 188, 2001, 45.33, TRUE);

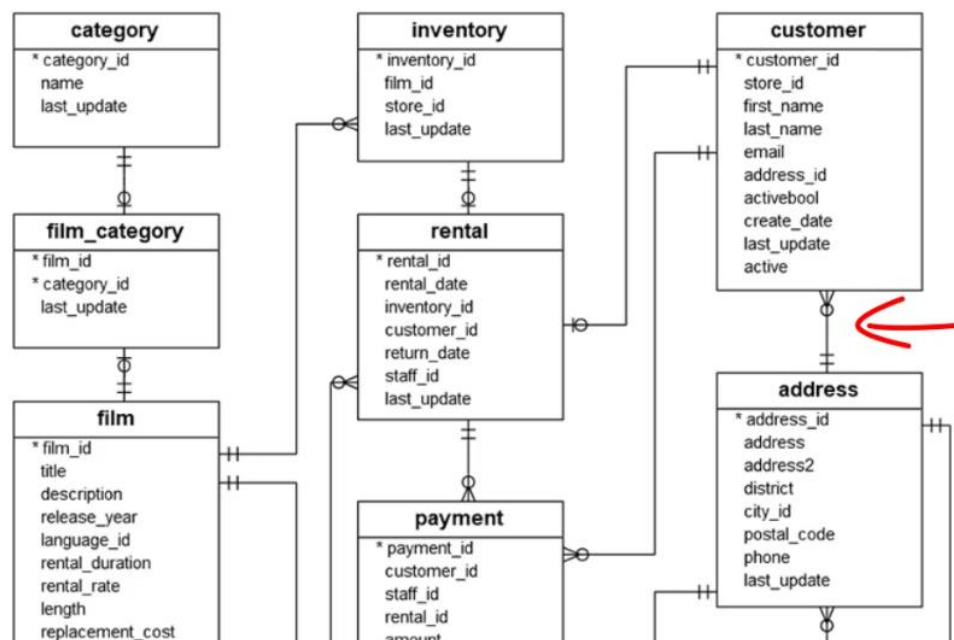
SELECT * FROM books;

DROP TABLE books;
```

Relacje w relacyjnej bazie danych

Notatka:

- Relacyjność bazy danych polega na tym, że dane znajdujące się w poszczególnych tabelach pozostają ze sobą w relacjach
- Zazwyczaj w tabelach wybiera się kolumnę lub kolumny, które jednoznacznie identyfikują rekordy tej tabeli. To tzw. klucz podstawowy (PRIMARY KEY). Często jest tworem sztucznym (np. customer_id)
- Aby odczytać cały zestaw informacji do danych znajdujących się w jednej tabeli trzeba dobrać pasujące rekordy z innych tabel. Zazwyczaj w pierwszej tabeli zapisuje się identyfikator rekordu znajdującego się w innej tabeli. Innymi słowy w danej tabeli zapisuje się wartości z klucza podstawowego innej tabeli. Takie wykorzystanie klucza podstawowego innej tabeli w danej tabeli nazywa się kluczem obcym (FOREIGN KEY).
- Tabele zazwyczaj przechowują dane dotyczące jednego tematu/obiektu/entity np. filmy, klienci, aktorzy itp.
- Relacyjność optymalizuje pracę zapytań i wykorzystanie zasobów.
- Mamy relacje jeden do wielu, jeden do jednego i wiele do wielu. Ta ostatnia jest implementowana poprzez dodatkową tabelę



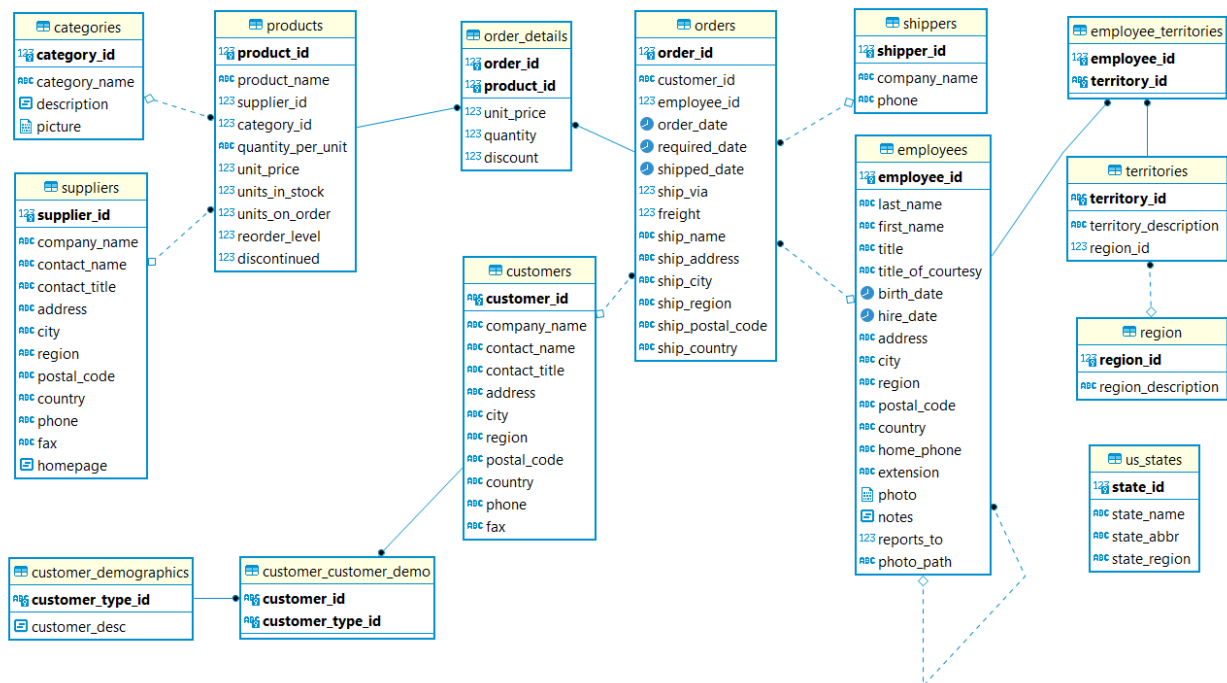
Laboratorium

1. Zainstaluj program **DBeaver** i podłącz się do bazy **northwind**
2. Wygeneruj diagram bazy danych, jeśli zechcesz wydrukuj go sobie (większy jest też dołączony na ostatnich stronach tego podręcznika)
3. Spróbuj zrozumieć relacje, jakie łączą poszczególne tabele. Bardzo to pomoże w kolejnych lekcjach

Sprawdź się!

1. Co oznacza, że tabela właściciel jest powiązana relacją jeden do wielu z tabelą samochody?
2. Jak implementuje się relację jeden do wielu?
3. Co to jest klucz podstawowy?
4. Jak łączy się dwie tabele w relacji wielu do wielu – np. rozważ tabelę nauczyciele i klasy. Ile klas jest uczonych przez jednego nauczyciela? A ilu nauczycieli ma jedna klasa?

Propozycja rozwiązania

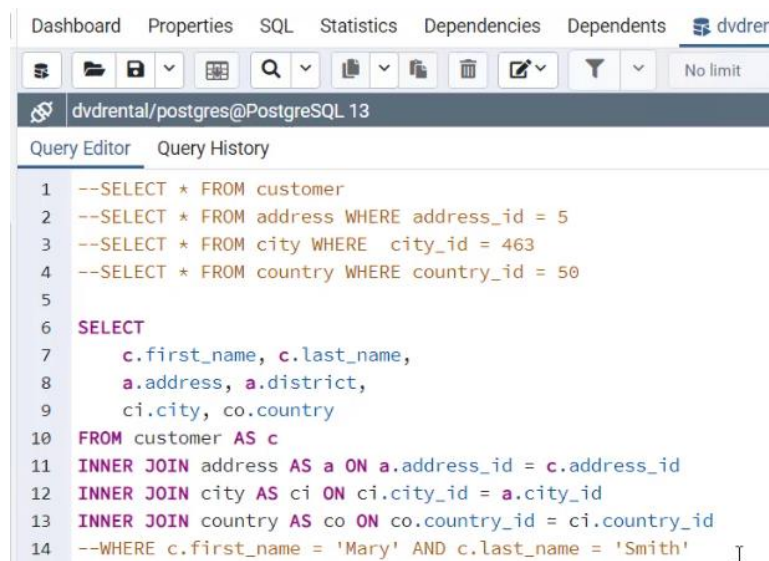


Złączenie tabel przez INNER JOIN

Notatka:

- Do łączenia danych z wielu tabel służy polecenie JOIN
- Po słowie JOIN wpisuje się nazwę dołączanej tabeli i można jej nadawać alias (to jedna z dobrych praktyk)
- Po nazwie tabeli w klauzuli ON określa się sposób dopasowywania rekordów z jednej tabeli z rekordami w drugiej tabeli. Zazwyczaj jest to relacja równości wartości w dwóch kolumnach, z których jedna jest kluczem obcym w pierwszej tabeli, a druga jest kluczem podstawowym w drugiej
- Dane zostaną wyświetlone tylko jeśli dopasowanie rekordów się uda. Jeśli pasujący rekord w drugiej tabeli nie istnieje, to taki wiersz nie będzie wyświetlony (to jest specyficzne dla polecenia INNER JOIN, które skrótowo można zapisywać jako JOIN)
- Jeśli łączone tabele zawierają kolumny o takich samych nazwach, to podczas korzystania z tych kolumn dojdzie do błędu niejednoznaczności. Korzystanie z aliasów tabel i odwoływanie się do kolumn poprzez wskazanie aliasem na tabelę z której ta kolumna pochodzi pozwoli wyeliminować ten błąd.
- W jednym zapytaniu można łączyć ze sobą wiele tabel (nie tylko dwie)
- Składnia:

```
SELECT * FROM customer c
JOIN address a ON a.address_id = c.address_id
```



The screenshot shows a PostgreSQL query editor interface. The top bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, and Dependents, along with a user profile icon labeled 'dvdrrer'. Below the tabs is a toolbar with various icons for file operations and query execution. The main area is titled 'Query Editor' and shows a SQL query with line numbers 1 through 14. The query is a complex SELECT statement with multiple INNER JOINs. It starts with a comment line 1: '--SELECT * FROM customer'. Line 2: '--SELECT * FROM address WHERE address_id = 5'. Line 3: '--SELECT * FROM city WHERE city_id = 463'. Line 4: '--SELECT * FROM country WHERE country_id = 50'. Line 5 is empty. Line 6: 'SELECT'. Line 7: 'c.first_name, c.last_name,'. Line 8: 'a.address, a.district,'. Line 9: 'ci.city, co.country'. Line 10: 'FROM customer AS c'. Line 11: 'INNER JOIN address AS a ON a.address_id = c.address_id'. Line 12: 'INNER JOIN city AS ci ON ci.city_id = a.city_id'. Line 13: 'INNER JOIN country AS co ON co.country_id = ci.country_id'. Line 14: '--WHERE c.first_name = 'Mary' AND c.last_name = 'Smith' I

Laboratorium

1. Napisz zapytanie łączące products i categories zwracające category_name i product_name
2. Napisz zapytanie łączące products i suppliers i zwracające product_name i company_name
3. Napisz zapytanie łączące 3 powyższe tabele i zwracające nazwy wymienione powyżej
4. Napisz zapytanie do order_details. Wybierz unit_price i quantity. Dołącz orders I wybierz kolumnę order_date. Dołącz products I wybierz kolumnę product_name. Dołącz customers i dołącz company_name
5. To jest duże zadanie, które wymaga użycia wielu prezentowanych wcześniej technik.
Napisz zapytanie, które wyświetli nazwę klienta (**company_name**) z tabeli **customers** oraz sumę wartości zrealizowanych zamówień w roku **1997** przez każdego z klienta. Sam(a) odgadnij jakie tabele trzeba ze sobą połączyć. Wartość zamówień wyznaczysz mnożąc **unit_price** przez **quantity**. Do wyznaczenia całkowitej kwoty zamówień skorzystaj z funkcji agregującej **SUM** i nie zapomnij o klauzuli **GROUP BY**.
Jeśli chcesz to posortuj dane wg. wielkości zamówień malejąco. Możesz zadbać o wyświetlenie sumarycznej wartości jako zaokrąglonej liczby dziesiętnej z dwoma miejscami po przecinku. W zależności, jak zrealizujesz to zadania może być konieczne skorzystanie z funkcji konwertującej typ danych na decimal.

Sprawdź się!

1. Co stanie się jeśli w zapytaniu wykorzystującym INNER JOIN łączącym tabelę właściciel z tabelą auto okaże się, że właściciel nie ma żadnego auta? Czy taki właściciel zostanie wyświetlony?
2. Co stanie się jeśli „leniwy programista” usunie z zapytania z INNER JOIN słowo INNER?
3. Czy JOIN może łączyć ze sobą tylko 2 tabele, czy możliwe jest łączenie wielu tabel naraz?
4. W dwóch tabelach łączonych ze sobą poleceniem JOIN występuje kolumna name. Uruchomienie zapytania kończy się błędem „column reference ‘name’ is amiguous”. Jak to naprawić?

Propozycja rozwiązania

```
SELECT c.category_name, p.product_name
FROM products p JOIN categories c ON p.category_id = c.category_id;

SELECT p.product_name, s.company_name
FROM products p JOIN suppliers s ON s.supplier_id = p.supplier_id;

SELECT c.category_name, p.product_name, s.company_name
FROM products p JOIN categories c ON p.category_id = c.category_id
JOIN suppliers s ON s.supplier_id = p.supplier_id;

SELECT od.unit_price, od.quantity, o.order_date, p.product_name, c.company_name
FROM order_details od
JOIN orders o ON o.order_id = od.order_id
JOIN products p ON p.product_id = od.product_id
JOIN customers c on c.customer_id = o.customer_id;

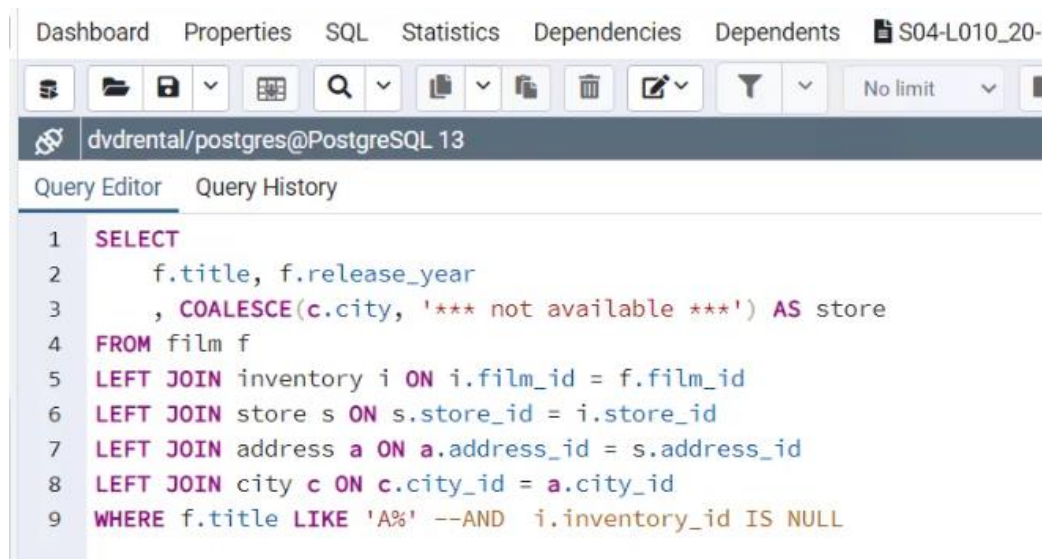
SELECT c.company_name, ROUND(CAST(SUM(od.unit_price * od.quantity)AS DECIMAL(10,2)),2)
AS total
FROM order_details od
JOIN orders o ON o.order_id = od.order_id
JOIN customers c on c.customer_id = o.customer_id
WHERE o.order_date BETWEEN DATE '1997-01-01' AND DATE '1997-12-31'
GROUP BY c.company_name
ORDER BY total DESC
```

Złączenie tabel przez OUTER JOIN

Notatka:

- Operator JOIN lub INNER JOIN (bo to to samo), prezentował w wyniku wiersz tylko jeśli dopasowanie rekordów w jednej i drugiej tabeli się udało
- Jeśli wierszy nie uda się dopasować, to takie wiersze są omijane
- Jeśli takie wiersze też miały by być zwrócone, to należy użyć LEFT OUTER JOIN lub LEFT JOIN
- Jeśli dla rekordu w tabeli po lewej stronie operatora LEFT JOIN nie udało się dopasować z żadnym rekordem w tabeli po prawej stronie, to wiersz zostanie mimo wszystko zwrócony, a kolumny z tej drugiej tabeli w takim wierszu będą zawierały NULL
- Przykład zastosowania tego operatora to np. wyszukiwanie filmów, których nie ma w wypożyczalni lub zarejestrowanych klientów, którzy jeszcze nie wypożyczyli żadnego filmu
- Jeśli chcesz żeby wyszukiwanie wierszy zaczynało się w tabeli po prawej stronie operatora JOIN to należy użyć RIGHT OUTER JOIN lub RIGHT JOIN
- Na co dzień nie stosuje się RIGHT JOIN, ta składnia przydaje się, gdy podczas pisania zapytania chcemy sprawdzić wynik polecenia po odwrócenia kolejności przetwarzanych tabel

```
SELECT * FROM film f
LEFT JOIN inventory i ON i.film_id = f.film_id
```



The screenshot shows a PostgreSQL Query Editor window. The title bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, and Dependents, along with a file icon and the name S04-L010_20-. Below the title bar is a toolbar with various icons for file operations, search, and execution. The main area displays a SQL query in a monospaced font, with line numbers 1 through 9 on the left. The query is a SELECT statement that joins the film, inventory, store, address, and city tables. It uses LEFT JOIN for all joins. The WHERE clause filters for films starting with 'A' and excludes records where the inventory_id is NULL.

```
1 SELECT
2     f.title, f.release_year
3     , COALESCE(c.city, '*** not available ***') AS store
4 FROM film f
5 LEFT JOIN inventory i ON i.film_id = f.film_id
6 LEFT JOIN store s ON s.store_id = i.store_id
7 LEFT JOIN address a ON a.address_id = s.address_id
8 LEFT JOIN city c ON c.city_id = a.city_id
9 WHERE f.title LIKE 'A%' --AND i.inventory_id IS NULL
```

Laboratorium

1. Czy są produkty, które jeszcze nigdy nie zostały zamówione? Napisz zapytanie łączące **products** i **order_details**. Wyświetl tylko te nazwy produktów (**product_name**), które nie mają pasujących zamówień (kolumna **order_id** jest NULL)
2. Czy mamy takich klientów, którzy nie złożyli jeszcze żadnego zamówienia? Napisz zapytanie łączące **customers** z **orders**. Wyświetl tylko te nazwy firm (**company_name**), które nie mają pasujących zamówień (kolumna **order_id** jest NULL)
3. Ile jest rekordów, jeśli łącząc tabelę **customers** z tabelą **orders** rozpocząć od **customers** (przy zastosowaniu OUTER JOIN)?
4. Ile jest rekordów, jeśli łącząc tabelę **customers** z tabelą **orders** rozpocząć od **orders** (przy zastosowaniu OUTER JOIN)?
5. Zapytania 3 i 4 powinny zwrócić różne wyniki. Jaki z tego wniosek?

Sprawdź się!

1. Kiedy użyjesz INNER JOIN a kiedy OUTER JOIN?
 - a. - Robisz zestawienie klientów, którzy składali zamówienia w ubiegłym miesiącu
 - b. - Robisz książkę telefoniczną wszystkich klientów. Dane są czasami niekompletne i brakuje wpisów w innych tabelach, ale ty chcesz żeby nawet takie wybrakowane dane były wyświetlane
 - c. - Dla dyrektora szkoły robisz zestawienie „klasy i wychowawcy”
 - d. - Przygotowujesz inwentaryzację w magazynie. Chcesz pokazać nazwę produktu z tabeli produkty i ile takich produktów jest w magazynie zapisaną w tabeli inventory. Chcesz ująć też te produkty, które zostały już wyprzedane.
 - e. - Przygotowujesz inwentaryzację w magazynie. Chcesz pokazać nazwę produktu z tabeli produkty i ile takich produktów jest w magazynie zapisaną w tabeli inventory. Chcesz opuścić też te produkty, które zostały już wyprzedane.

Propozycja rozwiązania

```
SELECT
  p.product_name
FROM products p
LEFT JOIN order_details od ON od.product_id = p.product_id
WHERE od.order_id IS NULL;
```

```
SELECT
  c.company_name
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.customer_id
WHERE o.order_id IS NULL;
```

```
SELECT COUNT(*) FROM customers c
LEFT JOIN orders o ON o.customer_id = c.customer_id;
```

```
SELECT COUNT(*) FROM customers c
RIGHT JOIN orders o ON o.customer_id = c.customer_id;
```

Trzeba być świadomym różnic między INNER/OUTER oraz LEFT/RIGHT. Różnice w wynikach świadczą o istnieniu rekordów bez odpowiedników w innych tabelach, co zwykle ma jakieś uzasadnienie biznesowe: klient nic nie kupił, produktu się nie sprzedał, policjant miał urlop i nie wystawił żadnego mandatu.

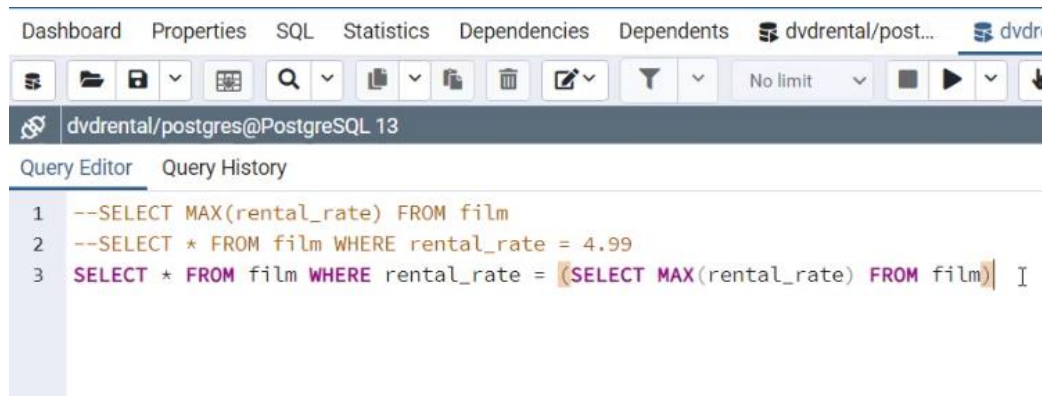
Podzapytania skalarne

Notatka:

- Podzapytanie to zapytanie umieszczone w innym zapytaniu. Podzapytanie nazywa się często zapytaniem wewnętrznym, w odróżnieniu od nadrzędnego zapytania, które nazywamy zewnętrznym
- Podzapytanie wpisuje się w nawiasie okrągłym:

```
SELECT
*
FROM film
WHERE
rental_rate = (SELECT MAX(rental_rate) FROM film)
```

- Podzapytanie skalarne, to takie podzapytanie, które zwraca wartość skalarną, np. liczbę, datę, napis itd.
- Podzapytanie skalarne można umieścić w tym miejscu zapytania zewnętrznego, gdzie można umieścić wartość skalarną, więc np. w SELECT lub WHERE
- Podzapytanie skalarne można też umieszczać w wyrażeniach
- Podzapytanie nieskorelowane jest niezależne od zapytania zewnętrznego, tzn. można je uruchamiać niezależnie od zapytania zewnętrznego



Laboratorium

1. Korzystając z podzapytań, wyświetl najdroższy produkt z tabeli **products**. W tym celu
 - a. Napisz zapytanie, które wyznaczy najwyższą cenę produktu (**unit_price**)
 - b. Napisz zapytanie, które zwróci rekordy, w których cena jest równa temu, co zwraca zapytanie z pkt.(a)
2. Dla każdego produktu z tabeli **products** wyświetl jego nazwę (**product_name**), cenę (**unit_price**) oraz różnicę między ceną i średnią ceną wszystkich produktów (którą wyznaczysz podzapytaniem). Wynik uporządkuj wg tej różnicy
3. Wyświetl listę pracowników z tabeli **employees**, którzy zostali zatrudnieni w tym samym roku, co pierwsi pracownicy. Skorzystaj z podzapytań. W tym celu:
 - a. Napisz zapytanie, które wyznaczy minimalny rok daty zatrudnienia pracownika (**hire_date**) – skorzystaj z funkcji wyodrębniającej rok z daty
 - b. Napisz zapytanie, które wyświetli wszystkie pola z tabeli **employees**, gdzie rok z daty zatrudnienia (**hire_date**) jest taki sam, jak rok zwrócony z zapytania z pkt (a)

Sprawdź się!

1. Kiedy skorzystasz z podzapytania, a kiedy z JOIN? (Niektóre zadania da się zrealizować i tak i tak)
2. W bibliotece znajdują się książki. Książki są pisane przez autorów. Chcesz zobaczyć jednocześnie informacje o autorze i o książce.
3. W bibliotece znajdują się książki. Książki są pisane przez autorów. Chcesz zobaczyć książki tego autora, którego książek jest w bibliotece najwięcej (książki najpopularniejszego autora)
4. Pływalnia ogłasza promocję. Kto kupi najwięcej biletów i karnetów w miesiącu lipcu i sierpniu dostanie nagrodę. Trzeba znaleźć najlepszego klienta.
5. Pływalnia potrzebuje raportu, który zaprezentuje dane adresowe klienta (tabela klienci) i szczegółowe informacje o kupionych przez niego biletach i karnetach (tabela sprzedaż)
6. W jakich miejscach w zapytaniu można używać podzapytań skalarnych?

Propozycja rozwiązania

```
SELECT MAX(unit_price) FROM products;

SELECT * FROM products
WHERE unit_price = (SELECT MAX(unit_price) FROM products);

SELECT
    product_name
    , unit_price
    , unit_price - (SELECT AVG(unit_price) FROM products) AS deviation
FROM products
ORDER BY deviation

SELECT * FROM employees
WHERE
    EXTRACT('year' FROM hire_date) =
        (SELECT MIN(EXTRACT('year' FROM hire_date)) FROM employees)
```

Podzapytanie a join

Notatka:

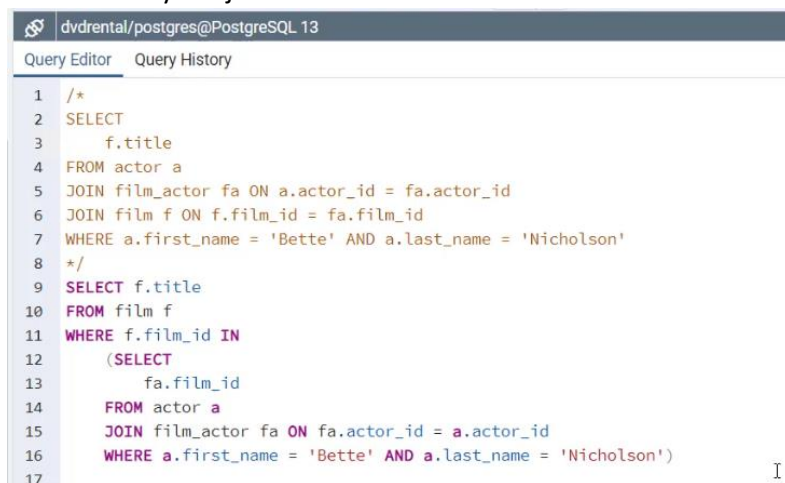
- Jeśli podzapytanie zwraca listę wartości, a chcesz wykorzystać te wartości do filtrowania danych w klauzuli WHERE, to:
 - Podzapytanie powinno zwracać tylko jedną kolumnę (zazwyczaj będzie to klucz podstawowy tabeli z zapytania zewnętrznego)
 - W zapytaniu zewnętrznym w WHERE skorzystaj z operatora IN, wskazując, że wartość nie ma znajdować się na liście zwróconej przez podzapytanie

```
SELECT
    product_name
FROM products
WHERE color IN (SELECT color FROM product WHERE quality = 1)
```

- Do zdefiniowania kryterium z wykorzystaniem operatora „=”, podzapytanie musi zwrócić tylko jedną wartość. Np. jeśli tylko w tabeli product znajduje się więcej produktów z quality równym 1, to poniższe zapytanie zakończyłoby się błędem:

```
SELECT
    product_name
FROM products
WHERE color = (SELECT color FROM product WHERE quality = 1)
```

- Jeden problem można często rozwiązać korzystając z zapytań z joinami lub z podzapytaniami. Z punktu widzenia silnika bazy danych dwie z pozoru różne formy zapytania mogą być wykonywane dokładnie w ten sam sposób. JOIN często ma lepszą wydajność niż podzapytanie, ale podzapytanie pozwala często łatwiej zrozumieć trudne sytuacje biznesowe



```
1  /*
2  SELECT
3      f.title
4  FROM actor a
5  JOIN film_actor fa ON a.actor_id = fa.actor_id
6  JOIN film f ON f.film_id = fa.film_id
7  WHERE a.first_name = 'Bette' AND a.last_name = 'Nicholson'
8  */
9  SELECT f.title
10 FROM film f
11 WHERE f.film_id IN
12     (SELECT
13         fa.film_id
14     FROM actor a
15     JOIN film_actor fa ON fa.actor_id = a.actor_id
16     WHERE a.first_name = 'Bette' AND a.last_name = 'Nicholson')
```


Laboratorium

1. Szukamy zamówień jakie były realizowane przez pracownika o nazwisku Buchanan. Zadanie zrealizujemy na 2 sposoby
2. Napisz podzapytanie zwracające numer pracownika (**employee_id**) z tabeli **employees**, dla pracownika o nazwisku (**last_name**) to 'Buchanan'. Następnie napisz zapytanie zewnętrzne skierowane do tabeli zamówień (**orders**), które będzie wyświetlać tylko zamówienia, które w **employee_id** mają wartość zwróconą przez podzapytanie
3. Napisz zapytanie do tabeli **orders**, dołącz pasujące dane z tabeli **employees** łącząc je w oparciu o numer pracownika (**employee_id**). W wyrażeniu filtrującym wskaż, że wyświetlane ma być tylko nazwisko (**last_name**) 'Buchanan'
4. Chcemy wyświetlić zamówienia złożone przez klientów z USA. Zadanie realizujemy na 2 sposoby:
5. Napisz podzapytanie, które wyświetla numer klienta (**customer_id**) z tabeli **customers** dla klientów z kraju (**country**) 'USA'. Napisz zapytanie zewnętrzne wyświetlające informacje o zamówieniach (**orders**), gdzie w kolumnie **customer_id** znajduje się wartość zwracana przez podzapytanie
6. Napisz zapytanie łączące tabelę zamówień (**orders**) i klientów (**customers**) w oparciu o równość w kolumnie z numerem klienta (**customer_id**). W wyrażeniu filtrującym określ, że zwracane mają być tylko rekordy dla klientów z kraju (**country**) 'USA'

Sprawdź się!

1. Prawda czy fałsz: większość zapytań JOIN da się przepisać na podzapytania
2. Zdecydowanie podzapytania mają znacznie lepszą wydajność niż JOIN-y
3. Kiedy JOIN zamieniasz na podzapytanie, to podzapytanie musi zwracać wartość skalarną

Propozycja rozwiązania

```
SELECT * FROM orders
WHERE employee_id IN
  (SELECT
    employee_id
    FROM employees e
    WHERE last_name = 'Buchanan');

SELECT * FROM orders o
JOIN employees e ON o.employee_id = e.employee_id
WHERE e.last_name = 'Buchanan'

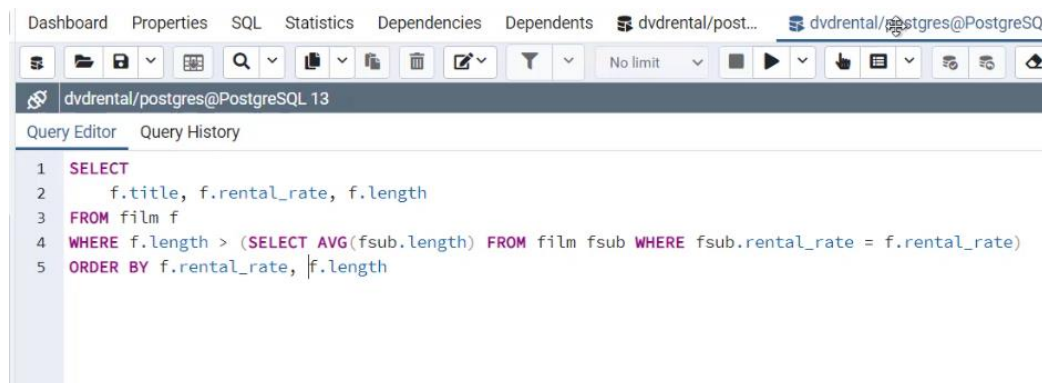
SELECT *
FROM orders WHERE customer_id IN
  (SELECT
    customer_id
    FROM customers
    WHERE country = 'USA')

SELECT *
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE c.country = 'USA'
```


Podzapytania skorelowane

Notatka:

- Zapytanie skorelowane to takie zapytanie, w którym
 - zapytanie wewnętrzne korzysta z danych zapytania zewnętrznego, a
 - zapytanie zewnętrzne korzysta z zapytania wewnętrznego
- W zapytaniu skorelowanym zapytania wewnętrzne nie można uruchomić niezależnie, ponieważ korzysta ono z danych zapytania zewnętrznego, a więc zależy od niego, jest z nim skorelowane
- Jeśli zarówno zapytanie zewnętrzne i wewnętrzne pracują na tych samych tabelach, to koniecznie należy zastosować różne aliasy dla tych tabel. Jest tak dlatego, że trzeba jakoś rozróżniać skąd mają być brane kolumny – z zapytania zewnętrznego czy wewnętrznego
- Analizując jak działa zapytanie skorelowane, możesz:
 - wyobrazić sobie, że zapytanie zewnętrzne w danej chwili pracuje na jednym rekordzie.
 - Dla tego rekordu osobno jest uruchamiane zapytanie wewnętrzne, które pewne dane z zapytania zewnętrznego traktuje jak parametry.
 - To zapytanie wewnętrzne zwraca pewną wartość, która z kolei pozwala zapytaniu zewnętrznemu sprawdzić spełnienie pewnych kryteriów względnie dokonać pewnych obliczeń



The screenshot shows a PostgreSQL query editor interface. The top bar includes tabs for Dashboard, Properties, SQL, Statistics, Dependencies, and Dependents. The main area displays a SQL query in the Query Editor. The query is a SELECT statement that retrieves film titles, rental rates, and lengths, filtered by a correlated subquery that calculates the average length of films with the same rental rate. The query is as follows:

```
1 SELECT
2     f.title, f.rental_rate, f.length
3 FROM film f
4 WHERE f.length > (SELECT AVG(fsub.length) FROM film fsub WHERE fsub.rental_rate = f.rental_rate)
5 ORDER BY f.rental_rate, f.length
```

Laboratoria

1. Raport „Produkty droższe od średniej z podziałem na kategorie”. Napisz zapytanie, które z tabeli **products**, które wyświetli tylko te produkty, które mają cenę (**unit_price**) większą od średniej ceny produktów należących do tej samej kategorii (**category_id**).
2. Do zapytania z pkt. 1 dodaj kolumnę prezentującą średnią cenę produktów z tej kategorii
3. Manager chciałby zobaczyć raport prezentujący, jaki produkt w zamówieniu jest „najważniejszy” – stanowi największy procent wartości danego zamówienia. Dlatego:
 - a. Napisz zapytanie do tabeli zamówień (**orders**), wyświetl numer zamówienia (**order_id**) oraz wartość danej pozycji na fakturze (**unit_price * quantity**)
 - b. Dołącz tabelę **products** korzystając z kolumny **product_id**. Wyświetl **product_name**
 - c. W SELECT dodaj podzapytanie, które zsumuje całkowitą wartość danego zamówienia, tzn. zwróci sumę **unit_price * quantity** dla tego samego **order_id**, co w zapytaniu zewnętrznym. Pamiętaj: **unit_price** i **quantity** należy tu brać z zapytania wewnętrznego.
 - d. W SELECT dodaj kolejne wyrażenie, które wartość pozycji na fakturze (**unit_price * quantity**) podzieli przez wartość z pkt 4. Wartość będzie ładniej prezentować się jako % wartości zamówienia po pomnożeniu przez 100.

Sprawdź się!

1. Opisz swoimi słowami na czym polega korelacja zapytań
2. W przypadku zapytań skorelowanych – co jest używane w zapytaniu wewnętrznym?
3. A jaki wpływ może mieć zapytanie wewnętrzne na zapytanie zewnętrzne?
4. Czy w przypadku zapytań skorelowanych można „wyjąć” z tego zapytania tylko zapytanie wewnętrzne i uruchomić je niezależnie?

Propozycja rozwiązania

```
SELECT
    product_name, unit_price
FROM products p
WHERE
    p.unit_price > (SELECT AVG(p1.unit_price) FROM products p1
                   WHERE p1.category_id = p.category_id)

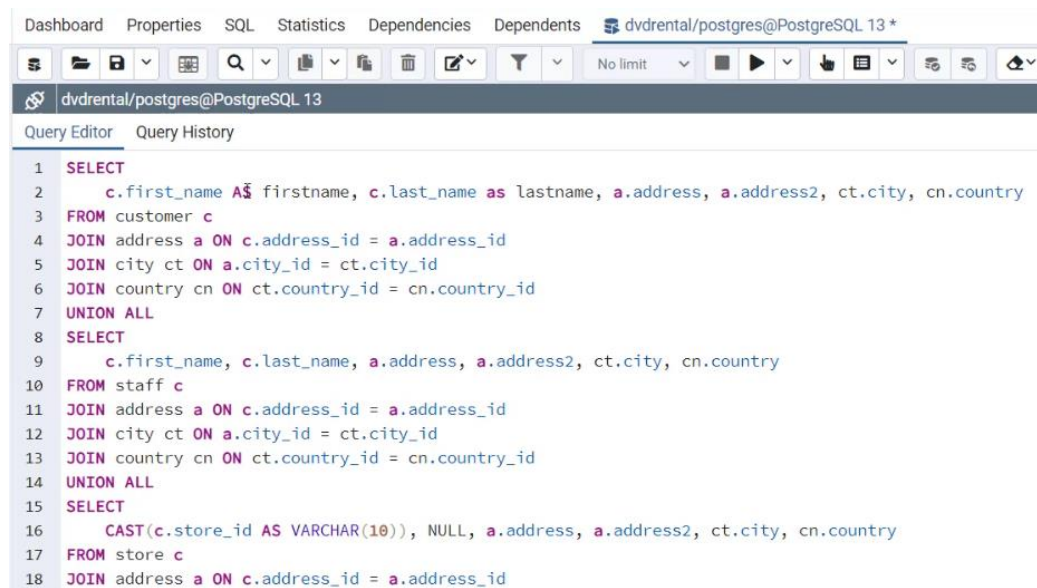
SELECT
    product_name, unit_price
    , (SELECT AVG(p1.unit_price) FROM products p1
       WHERE p1.category_id = p.category_id) AS category_avg
FROM products p
WHERE
    p.unit_price > (SELECT AVG(p1.unit_price) FROM products p1
                   WHERE p1.category_id = p.category_id)

SELECT
    od.order_id
    , p.product_name
    , od.unit_price * od.quantity AS order_value
    , (SELECT SUM(od1.unit_price * od1.quantity) FROM order_details od1
       WHERE od1.order_id = od.order_id) AS order_sum
    , 100 * od.unit_price * od.quantity /
      (SELECT SUM(od1.unit_price * od1.quantity) FROM order_details od1
       WHERE od1.order_id = od.order_id) AS order_percent
FROM order_details od
JOIN products p ON od.product_id = p.product_id
```

Złączenie wyników zapytań przez UNION

Notatka:

- Aby połączyć ze sobą wyniki dwóch podobnych zapytań należy skorzystać z polecenia UNION
- Oba łączone ze sobą zapytania muszą być do siebie podobne. Podobieństwo polega na tym, że:
 - Zwracana jest taka sama liczba kolumn
 - Typy poszczególnych kolumn muszą być kompatybilne ze sobą:
 - Liczby są kompatybilne z liczbami napisy z napisami itp.
 - NULL jest kompatybilne z każdym typem
 - Jeśli jakieś kolumny zapytań są ze sobą niekompatybilne, to należy je konwertować korzystając z CAST
- Polecenie UNION zwraca wiersze z pierwszego zapytania i z drugiego zapytania w postaci jednego result-set. Jeśli oba zapytania zwracałyby taki sam wiersz, to UNION zaprezentuje go tylko raz
- Jeśli takie powtarzające wartości miałyby się pojawić w wyniku wielokrotnie, to należy użyć UNION ALL
- Ze względu na eliminację duplikatów polecenie UNION jest bardziej obciążające obliczeniowo aniżeli UNION ALL
- Poprzez UNION można łączyć ze sobą również więcej wyników zapytań
- Nazwy kolumn w wynikowym result-set są definiowane przez pierwsze zapytanie



```
1 SELECT
2     c.first_name AS firstname, c.last_name as lastname, a.address, a.address2, ct.city, cn.country
3 FROM customer c
4 JOIN address a ON c.address_id = a.address_id
5 JOIN city ct ON a.city_id = ct.city_id
6 JOIN country cn ON ct.country_id = cn.country_id
7 UNION ALL
8 SELECT
9     c.first_name, c.last_name, a.address, a.address2, ct.city, cn.country
10 FROM staff c
11 JOIN address a ON c.address_id = a.address_id
12 JOIN city ct ON a.city_id = ct.city_id
13 JOIN country cn ON ct.country_id = cn.country_id
14 UNION ALL
15 SELECT
16     CAST(c.store_id AS VARCHAR(10)), NULL, a.address, a.address2, ct.city, cn.country
17 FROM store c
18 JOIN address a ON c.address_id = a.address_id
```

Laboratorium

Firma zamierza zintegrować klientów, dostawców i pracowników i planuje wspólny event. Masz za zadanie przygotować listę obecności, ale po kolei:

1. Napisz zapytanie do tabeli **employees**, które zwróci **last_name**, **first_name** i stały napis 'EMPLOYEE' zaaliasowany jako **type**
2. Napisz zapytanie do tabeli **customers**, które zwróci fragment napisu z kolumny **contact_name** od pierwszej spacji do końca napisu (będzie to nazwisko), fragment napisu z kolumny **contact_name** od początku do pierwszej spacji (będzie to imię) i stały napis 'CUSTOMER'
3. Napisz zapytanie do tabeli **suppliers**, które wyodrębni nazwisko i imię z **contact_name**, tak samo jak w poprzednim punkcie. Trzecia kolumna ma zawierać stały napis 'SUPPLIER'
4. Napisz zapytanie do tabeli **shippers**, które zwróci **company_name**, napis pusty, a jako trzecią kolumnę 'SHIPPER'
5. Połącz zapytania tak, aby został zwrócony jeden wynik. Wartości powtarzające się mają być zwrócone wielokrotnie

Sprawdź się!

1. Przygotowujesz zestawienie serwerów wykorzystywanych w firmie. Tabela **servers_app** zawiera informacje o serwerach aplikacyjnych, a tabela **servers_db** informacje o serwerach bazodanowych, a tabela **servers_utils** o serwerach dla antywirusa, automatyzacji itp. Czasami na jednym serwerze jest zainstalowana aplikacja i jej baza danych lub program narzędziowy i baza danych. Chcesz stworzyć zapytanie, w którym zostaną wyświetlone informacje o wszystkich serwerach, ale każdy serwer powinien się pojawić tylko jeden raz. Z jakich poleceń skorzystasz?
2. A jakiego polecenia użyjesz jeśli serwery miałyby się powtarzać?
3. Na czym polega zasada kompatybilności typów w łączonych zapytaniach?
4. Jak można rozwiązać problem braku kompatybilności typów w łączonych podzapytaniach?
5. Null jest kompatybilne czy niekompatybilne z innymi typami?

Propozycja rozwiązania

```
SELECT
    e.last_name, e.first_name, 'EMPLOYEE' AS type
FROM employees e
UNION ALL
SELECT
    SUBSTRING(c.contact_name, POSITION(' ' IN c.contact_name)+1,
        LENGTH(c.contact_name)-POSITION(' ' IN c.contact_name))
    , LEFT(c.contact_name, POSITION(' ' IN c.contact_name))
    , 'CUSTOMER'
FROM customers c
UNION ALL
SELECT
    SUBSTRING(s.contact_name, POSITION(' ' IN s.contact_name)+1,
        LENGTH(s.contact_name)-POSITION(' ' IN s.contact_name))
    , LEFT(s.contact_name, POSITION(' ' IN s.contact_name))
    , 'SUPPLIER'
FROM suppliers s
UNION ALL
SELECT
    s.company_name, '', 'SHIPPER'
FROM shippers s
```

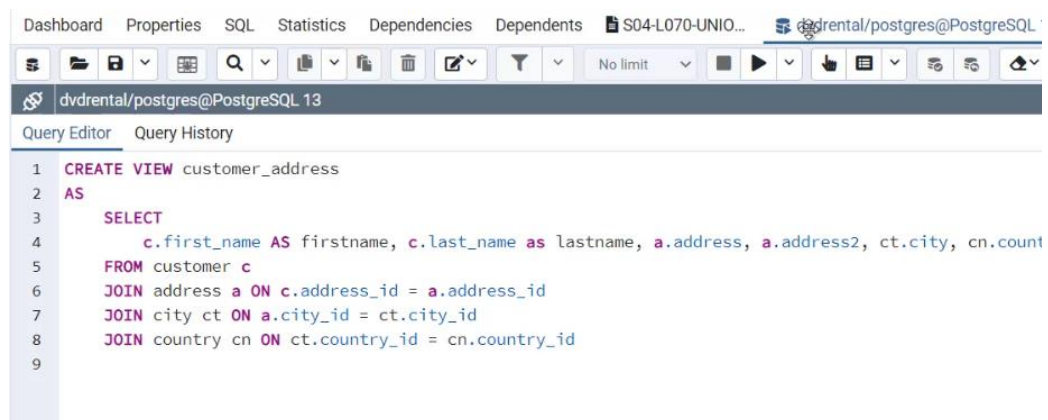
Wprowadzenie do widoków - CREATE VIEW

Notatka:

- Widok to nazwa przypisana do zapytania
- Zalety widoków to:
 - Łatwe i wielokrotne korzystanie z raz utworzonego zapytania
 - Udostępnienie użytkownikowi tylko wybranych kolumn lub wierszy zbioru danych
 - Ukrycie przed użytkownikiem skomplikowanej struktury logicznej relacyjnej bazy danych
 - Definiowanie granic bezpieczeństwa danych
 - Ułatwienie modyfikacji kodu aplikacji korzystającej z widoku
- Definicje widoków można podejrzeć w pgAdmin na zakładce SQL
- Korzystanie z widoków z punktu widzenia użytkownika przypomina po prostu korzystanie z tabeli
- Tworząc widok korzystasz ze składni

```
CREATE VIEW view_name
AS
SELECT first_name, last_name FROM customer
```

- Jeśli w widoku trzeba dokonać niewielkich zmian, to można korzystać z polecenia CREATE OR REPLACE, w przypadku większych zmian można korzystać z nieco bardziej złożonych poleceń, lub po prostu usunąć widok korzystając z DROP VIEW i utworzyć nowy



```
1 CREATE VIEW customer_address
2 AS
3     SELECT
4         c.first_name AS firstname, c.last_name as lastname, a.address, a.address2, ct.city, cn.count
5     FROM customer c
6     JOIN address a ON c.address_id = a.address_id
7     JOIN city ct ON a.city_id = ct.city_id
8     JOIN country cn ON ct.country_id = cn.country_id
9
```

Laboratorium

1. Księgowość planuje często korzystać z listy nazwisk reprezentantów klientów. Każdorazowe pisanie zapytania wycinającego z tabeli **customers** z kolumny **contact_name** nazwiska i imienia jest uciążliwe, dlatego trzeba utworzyć widok **customer_names**, który wycina nazwisko i imię z **customer_names**. Pamiętaj o aliasowaniu wyliczanych kolumn
2. Sprawdź działanie widoku
3. Księgowość poprosiła o dodanie tytułu osoby jako trzeciej kolumny. Usuń widok i utwórz go na nowo dodając kolumnę **contact_title** jako trzecią kolumnę
4. Sprawdź działanie widoku

Sprawdź się!

1. Jak korzysta się z widoków?
2. Jakie polecenie definiuje widok?
3. Jak usunąć widok?
4. Kiedy warto korzystać z widoków?

Propozycja rozwiązania

```
CREATE VIEW customer_names
AS
SELECT
    SUBSTRING(c.contact_name, POSITION(' ' IN c.contact_name)+1,
    LENGTH(c.contact_name)-POSITION(' ' IN c.contact_name)) AS last_name
    , LEFT(c.contact_name, POSITION(' ' IN c.contact_name)) AS first_name
FROM customers c

SELECT * FROM customer_names

DROP VIEW customer_names

CREATE VIEW customer_names
AS
SELECT
    SUBSTRING(c.contact_name, POSITION(' ' IN c.contact_name)+1,
    LENGTH(c.contact_name)-POSITION(' ' IN c.contact_name)) AS last_name
    , LEFT(c.contact_name, POSITION(' ' IN c.contact_name)) AS first_name
    , contact_title
FROM customers c

SELECT * FROM customer_names
```

Dodawanie rekordów - INSERT

Notatka:

- Przed wstawieniem rekordów do tabeli należy zapoznać się ze strukturą tabeli: informacjami przechowywanymi w kolumnach, typami danych, dodatkowymi właściwościami, jak unikalność, automatyczne numerowanie wierszy, dopuszczanie wartości NULL, wartości domyślne itp.
- Do dodawania rekordów służy polecenie INSERT o składni:

```
INSERT INTO tabela VALUES(201, 'Brad', 'Pitt', NOW())
```

- Wstawiając rekordy można opuścić niektóre kolumny. Wtedy wartości będą generowane przez zdefiniowane wartości domyślne lub pozostaną NULL, o ile NULL jest dopuszczalny. Składnia z wybranymi kolumnami wygląda tak:

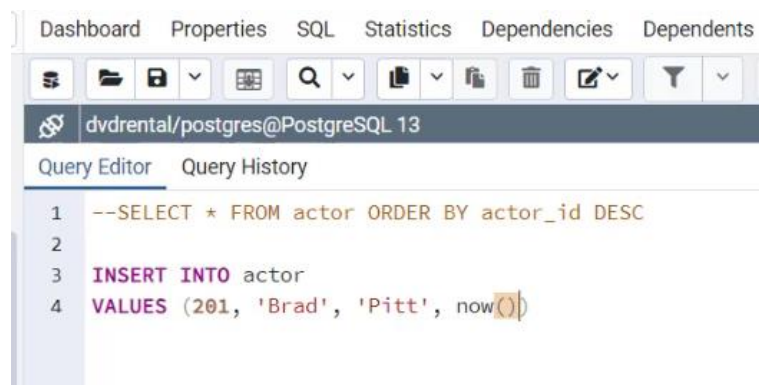
```
INSERT INTO actor(first_name, last_name) VALUES('Amy', 'Adams')
```

- Jeśli NULL nie jest dopuszczalny i brakuje wartości domyślnej, a polecenie INSERT nie specyfikuje kolumny, to dojdzie do błędu.
- Polecenie INSERT pozwala wstawić więcej rekordów za jednym razem:

```
INSERT INTO actor(first_name, last_name)  
VALUES('Leonardo', 'Di Caprio'), ('Julia', 'Roberts')
```

- Jeśli po wstawieniu nowego rekordu chcesz zobaczyć identyfikator nowo wstawianego rekordu, wystarczy do polecenia INSERT dodać na końcu:

```
RETURNING actor_id
```



Laboratorium

1. Nasza hurtownia będzie handlować pieczywem. Wypełnij minimalną liczbę kolumn w tabeli **categories** wprowadzając kategorię **'Bread'**. Numer kategorii musisz określić samodzielnie – wybierz pierwszą wolną wartość. Po wstawieniu rekordu sprawdź zawartość tabeli.
2. Dodajemy nowy product do kategorii z pkt. 1. Wstaw do tabeli **products** nowy produkt: `id=78`, `product_name = Baguette`, `supplier_id=3`, `category_id=9`, `quantity_per_unit=piece`, `unit_price=2`, `units_in_stock=99`, `units_in_order=0`, `reorder_level=0`, `discontinued=0`. Po wstawieniu rekordu sprawdź zawartość tabeli.
3. Klient składa zamówienie. Wstaw do tabeli **orders** nowy rekord z następującymi wartościami: `order_id=11078`, `customer_id=FRANK`, `employee_id=7`, `order_date= data dzisiejsza`, `required_date = data za 3 dni`, `shipped_date= NULL`, `ship_via=2`, `freight=1`, `ship_name = Ben Benson`, `ship_address = Padre Rico 8`, `ship_city = Pueblo Piernico`, `ship_region = NULL`, `ship_postal_code = 12-345`, `ship_country = San Escobar`. Po wstawieniu rekordu sprawdź zawartość tabeli.
4. W ramach zamówienia, klient kupuje bagietkę. Wstaw rekord do tabeli **order_details**: `order_id = 11078`, `product_id = 78`, `unit_price = 2`, `quantity = 1`, `discount = 0`. Po wstawieniu rekordu sprawdź zawartość tabeli.

Sprawdź się!

1. Gdzie można uzyskać informacje o tym, jak wygląda struktura tabeli, jakie kolumny akceptują wartość NULL itp.?
2. Jaka jest składnia polecenia INSERT jeśli chcesz wstawić wartości dla wszystkich kolumn?
3. Jaka jest składnia polecenia INSERT jeśli chcesz wstawić wartości tylko dla niektórych kolumn?
4. Jaka jest składnia polecenia INSERT jeśli chcesz wstawić do tabeli od razu większą liczbę rekordów?
5. Jaka jest składnia polecenia INSERT jeśli chcesz, aby to polecenie zwróciło pewne informacje o właśnie wstawionych wierszach (np. identyfikator klucza wygenerowany przez bazę)?
6. Co stanie się, jeśli wstawisz 2 rekordy z takim samym PRIMARY KEY?

Propozycja rozwiązania

```
INSERT INTO categories(category_id, category_name) VALUES (9,'Bread') RETURNING category_id;
SELECT * FROM categories WHERE category_id = 9;
INSERT INTO products
VALUES(78, 'Baguette', 3, 9, 'piece', 2, 99, 0, 0, 0);
SELECT * FROM products WHERE product_id = 78;
INSERT INTO orders VALUES(11078, 'FRANK',7, NOW(), NOW()+ '3 days',NULL,2,1,
'Ben Benson', 'Padre Rico 8', 'Pueblo Piernico', NULL, '12-345','San Escobar')
SELECT* FROM orders WHERE order_id = 11078
INSERT INTO order_details VALUES(11078, 78, 2, 1, 0)
SELECT * FROM order_details WHERE order_id = 11078
```

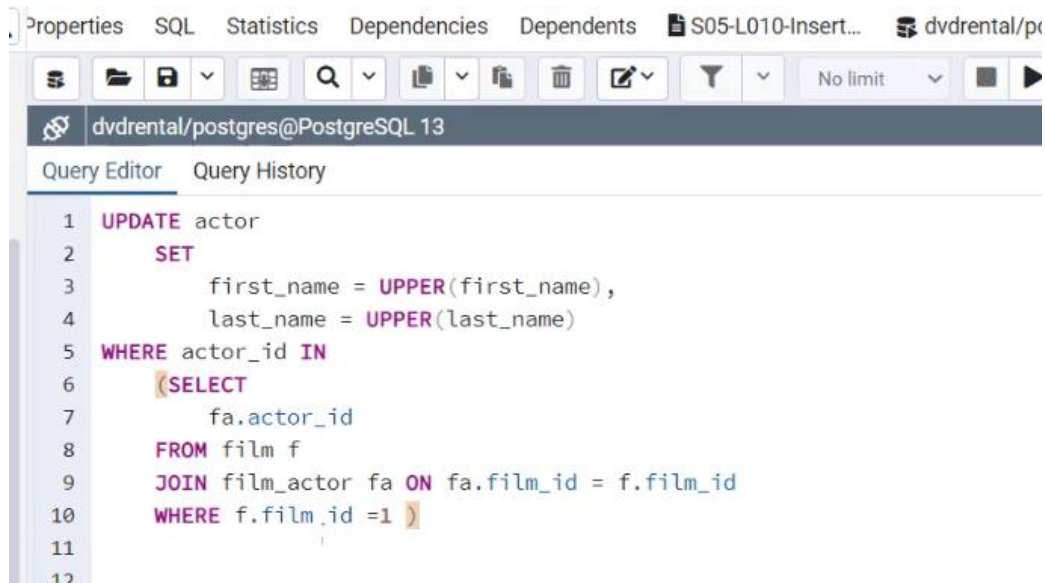

Modyfikacja rekordów - UPDATE

Notatka:

- Do modyfikacji rekordu służy UPDATE. Składnia wygląda następująco:

```
UPDATE actors
SET
    first_name = 'Jolie',
    last_name = 'Angelina',
WHERE actor_id = 208
```

- Uruchomienie polecenia UPDATE bez klauzuli WHERE spowoduje modyfikację wszystkich rekordów
- Aby uniknąć pomyłek przy aktualizacji rekordu najlepiej zaczynać od napisania polecenia SELECT z odpowiednią klauzulą WHERE
- Jeśli aktualizacja rekordu ma się odbyć w oparciu o dane z innej tabeli, to w klauzuli WHERE można skorzystać z podzapytania i operatora IN:



The screenshot shows a PostgreSQL query editor window titled 'dvdrental/postgres@PostgreSQL 13'. The 'Query Editor' tab is active, displaying the following SQL code:

```
1 UPDATE actor
2     SET
3         first_name = UPPER(first_name),
4         last_name = UPPER(last_name)
5 WHERE actor_id IN
6     (SELECT
7         fa.actor_id
8     FROM film f
9     JOIN film_actor fa ON fa.film_id = f.film_id
10    WHERE f.film_id = 1 )
```

Laboratorium

1. W tabeli **categories** zmień pole Description na „Bakery products” dla wiersza z Category_id = 9
2. W tabeli **products** w rekordzie z product_id = 78 zmień: quantity_per_unit = 'pcs', unit_price = 2.49, units_in_stock = 98
3. Napisz zapytanie, które z tabeli **order_details** wyświetli **product_id**, dla tych produktów, które mają w tabeli odnotowaną sumę ilości sprzedanych produktów (**quantity**) < 100
4. W oparciu o zapytanie z pkt. 3 dla tych produktów, które mają odnotowaną sprzedaż w ilości mniejszej niż 100 zmień w tabeli **products** kolumnę **discontinued** na 1

Sprawdź się!

1. Jak wygląda składnia polecenia UPDATE
2. Co stanie się po uruchomieniu polecenia UPDATE bez klauzuli WHERE?
3. Czy wyliczenie nowej wartości w rekordzie może się odbywać w oparciu o wartości już występujące w tym rekordzie (np. zwiększ cenę o 10%)?
4. Jaka dobra praktyka pozwala aktualizować rzeczywiście tylko te rekordy, które mają być zmodyfikowane?

Propozycja rozwiązania

```
UPDATE categories
  SET
    description = 'Bakery products'
WHERE category_id = 9;
```

```
UPDATE products
  SET
    quantity_per_unit = 'pcs',
    unit_price = 2.49,
    units_in_stock = 98
WHERE product_id = 78;
```

```
SELECT
  product_id --, SUM(quantity)
FROM order_details
GROUP BY product_id
HAVING SUM(quantity) < 100
```

```
UPDATE products
  SET discontinued = 1
WHERE product_id IN
(
  SELECT
    product_id --, SUM(quantity)
  FROM order_details
  GROUP BY product_id
  HAVING SUM(quantity) < 100
)
```

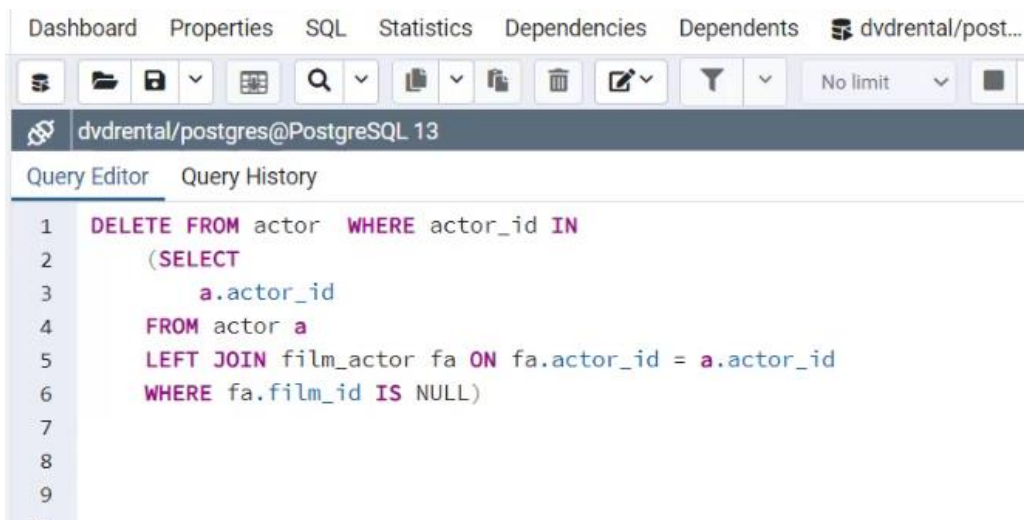
Usuwanie rekordów - DELETE

Notatka:

- Do usuwania rekordu służy DELETE. Składnia wygląda następująco:

```
DELETE FROM actors  
WHERE actor_id = 208
```

- Uruchomienie polecenia UPDATE bez klauzuli WHERE spowoduje usunięcie wszystkich rekordów
- Aby uniknąć pomyłek przy usuwaniu rekordu najlepiej zaczynać od napisania polecenia SELECT z odpowiednią klauzulą WHERE
- Jeśli tabele są połączone kluczami obcymi, to usunięcie rekordu uda się tylko wtedy, gdy do danego rekordu nie ma w innych tabelach rekordów, które odwołują się do tego rekordu. Jeśli takie rekordy by istniały, to polecenie kończy się błędem
- Jeśli usuwanie rekordu ma się odbyć w oparciu o dane z innej tabeli, to w klauzuli WHERE można skorzystać z podzapytania i operatora IN:



The screenshot shows a web-based PostgreSQL interface. At the top, there are tabs for 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', and 'Dependents'. Below these is a toolbar with various icons for database operations. The main area is titled 'Query Editor' and contains the following SQL code:

```
1 DELETE FROM actor WHERE actor_id IN  
2 (SELECT  
3     a.actor_id  
4     FROM actor a  
5     LEFT JOIN film_actor fa ON fa.actor_id = a.actor_id  
6     WHERE fa.film_id IS NULL)  
7  
8  
9
```

Laboratorium

W tym laboratorium usuniemy rekordy pracowicie wstawiane przez nas kilka lekcji wcześniej. Poniżej w poleceniach są używane identyfikatory zgodne z treścią poprzednich lekcji, jeśli jednak w Twoim przypadku identyfikatory się różnią, to odpowiednio zmodyfikuj polecenia

1. Spróbuj usunąć rekord z tabeli **categories**, w którym **Category_id = 9**. Polecenie powinno się zakończyć błędem o tym, że do rekordu odnosi się inny rekord z tabeli **products**. To dobrze!
2. Spróbuj usunąć rekord z tabeli **products**, w którym **product_id = 78**. Polecenie powinno się zakończyć błędem o tym, że do rekordu odnosi się inny rekord z tabeli **orders**. To dobrze!
3. Spróbuj usunąć rekord z tabeli **orders**, w którym **order_id = 11078**. Polecenie powinno się zakończyć błędem o tym, że do rekordu odnosi się inny rekord z tabeli **order_details**. To też dobrze!
4. Spróbuj usunąć rekord z tabeli **order_details**, w którym **order_id = 11078**. Polecenie powinno się zakończyć sukcesem. To jeszcze lepiej!
5. Teraz wykonaj ponownie czynności z pkt. 3, 2, 1. Jeśli zrobisz to w tej kolejności, to każde z poleceń powinno się udać, bo do usuwanych rekordów żaden rekord z zależnych tabel już się nie odnosi.

Sprawdź się!

1. Jaka jest składnia polecenia DELETE?
2. Masz tabelę products, o której wiadomo tylko tyle, że product_id jest PRIMARY KEY. Chcesz usunąć jeden konkretny rekord. Do czego najlepiej będzie się odwołać w klauzuli WHERE?
3. Co stanie się, kiedy uruchomisz polecenie DELETE bez klauzuli WHERE?
4. Jaka dobra praktyka pozwala usuwać rzeczywiście tylko te rekordy, które mają być usunięte?

Propozycja rozwiązania

```
--error will be displayed
DELETE FROM categories WHERE category_id = 9;

--error will be displayed
DELETE FROM products WHERE product_id = 78;

--error will be displayed
DELETE FROM orders WHERE order_id = 11078;

--success
DELETE FROM order_details WHERE order_id = 11078;

--success
DELETE FROM orders WHERE order_id = 11078;
DELETE FROM products WHERE product_id = 78;
DELETE FROM categories WHERE category_id = 9;
```

Tworzenie tabel poleceniem SELECT. Tabele tymczasowe

Notatka:

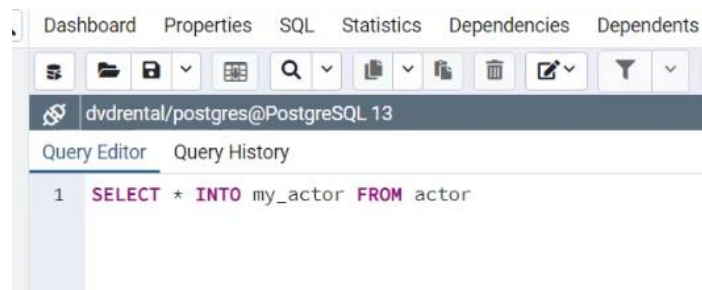
- W poleceniu SELECT za listą kolumn można dopisać słowo INTO i podać nazwę tabeli. W efekcie powstanie nowa tabela, której zawartością są wiersze zwracane przez zapytanie
- Nowo tworzona tabela ma strukturę zbliżoną do struktury oryginalnych tabel, z których pochodzą wiersze
- Ponowne uruchomienie polecenia SELECT ... INTO zwróci błąd, bo tabela już istnieje, więc nie można jej utworzyć
- Polecenie SELECT ... INTO przydaje się do tworzenia kopii danych, np. w celu przeprowadzenia symulacji
- Do usuwania tabeli służy polecenie

DROP TABLE

- Aby uniknąć tworzenia tabel dostępnych dla wszystkich użytkowników, można takie tymczasowe zbiory danych zapisywać w tabelach tymczasowych
- Kiedy tworzysz tabelę tymczasową, to jej nazwę poprzedzasz słowem TEMPORARY lub w skrócie TEMP
- Domyślnie tabele tymczasowe nie są prezentowane w pgAdmin
- Tabele tymczasowe są automatycznie kasowane po zakończeniu sesji
- Przykładowe wywołanie polecenia SELECT może wyglądać tak::

SELECT * INTO TEMPORARY temp_actor FROM actors

- SELECT ... INTO można wykorzystać do tworzenia nowej pustej tabeli o takiej samej strukturze jak oryginalna tabela. Wystarczy w WHERE wpisać kryterium, które nigdy nie jest spełnione, np. 1=0



Laboratorium

Firma chce przeprowadzić analizę zmian w ofercie i cenniku, dlatego:

1. Przepisz do tymczasowej tabeli **temp_order_details** wszystkie rekordy z **order_details**
2. Wyznacz sumę wartości sprzedaży w oparciu o **temp_order_details** (ma to być suma iloczynu **quantity * unit_price**)
3. Usuń z **temp_order_detail** rekordy produktów należące do kategorii nr 1. Możesz zacząć od napisania polecenia wyświetlającego **product_id** z tabeli **products**, które mają **category_id=1**. Potem w oparciu o to podzapytanie napisz polecenie usuwające odpowiednie rekordy z **temp_order_details**
4. Powtórz zapytanie z pkt. 2, żeby sprawdzić jaki byłby efekt wycofania produktów z kategorii 1 na całkowitą sprzedaż
5. W tabeli **temp_order_details** zasymuluj podwyżkę ceny produktów z kategorii 2 o 50%. Możesz zacząć od napisania polecenia wyświetlającego **product_id** z tabeli **products**, które mają **category_id=2**. Potem w oparciu o to podzapytanie napisz polecenie zmieniające **unit_price** w tabeli **temp_order_details** na **unit_price * 1.5**
6. Powtórz zapytanie z pkt. 2, żeby sprawdzić jaki byłby efekt podwyżki ceny produktów z kategorii 2 na całkowitą sprzedaż
7. Usuń tabelę **temp_order_details**

Sprawdź się!

1. Czy SELECT INTO tworzy nowe tabele, czy dopisuje dane do istniejącej tabeli?
2. Jak usuwa się tabelę?
3. Jak zapisać wynik zapytania w tabeli tymczasowej?
4. Kiedy przestaje istnieć tabela tymczasowa?
5. Jak na szybko utworzyć tabelę o takiej samej strukturze, jak oryginalna, ale pustą?

Propozycja rozwiązania

```
SELECT * INTO TEMP temp_order_details FROM order_details;

SELECT SUM(quantity * unit_price) FROM temp_order_details
--1354458

DELETE FROM temp_order_details
WHERE product_id IN (SELECT product_id FROM products WHERE category_id = 1)

SELECT SUM(quantity * unit_price) FROM temp_order_details
--1067931

UPDATE temp_order_details
SET unit_price = 1.5 * unit_price
WHERE product_id IN (SELECT product_id FROM products WHERE category_id = 2)

SELECT SUM(quantity * unit_price) FROM temp_order_details
--1124779

DROP TABLE temp_order_details
```

Przepisywanie rekordów za pomocą INSERT INTO ... SELECT

Notatka:

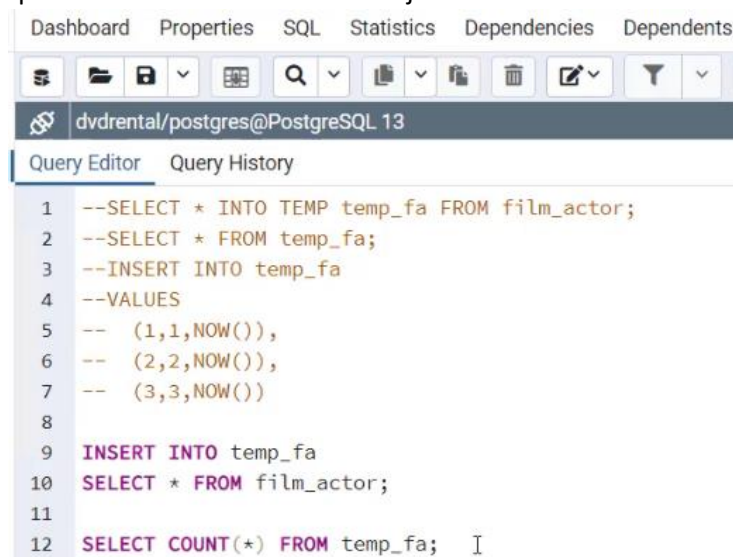
- Polecenie SELECT ... INTO tworzy nową tabelę i wypełnia ją rekordami generowanymi przez polecenie SELECT
- Powtórne uruchomienie polecenia SELECT ... INTO kończy się błędem, bo tabela już istnieje
- Jeśli chcesz dopisać masowo rekordy generowane przez polecenie SELECT do istniejącej tabeli, to należy się posłużyć poleceniem INSERT INTO ... SELECT. INSERT odpowiada za wstawianie rekordów, a SELECT za wygenerowanie wartości
- Jeśli chcesz utworzyć nową tabelę posłuż się SELECT ... INTO

```
SELECT * INTO TEMP temp_fa FROM film_actor
```

- A kiedy chcesz dopisać dane do istniejącej tabeli wykorzystaj INSERT INTO ... SELECT:

```
INSERT INTO temp_fa  
SELECT * FROM film_actor
```

- Wykorzystywane tu polecenie SELECT może być dowolnie skomplikowane. Może łączyć dane z wielu tabel, korzystać z podzapytań, wyliczać w locie wartości nowych kolumn itd.
- Najważniejsza różnica między SELECT ... INTO a INSERT INTO ... SELECT jest taka, że w pierwszym przypadku docelowa tabela nie może jeszcze istnieć, a w przypadku drugiego polecenia tabela docelowa musi już istnieć



```
1 --SELECT * INTO TEMP temp_fa FROM film_actor;  
2 --SELECT * FROM temp_fa;  
3 --INSERT INTO temp_fa  
4 --VALUES  
5 -- (1,1,NOW()),  
6 -- (2,2,NOW()),  
7 -- (3,3,NOW())  
8  
9 INSERT INTO temp_fa  
10 SELECT * FROM film_actor;  
11  
12 SELECT COUNT(*) FROM temp_fa; I
```

Laboratorium

W celu wykonania kolejnych analiz sprzedażowych poproszono Cię o przygotowanie danych testowych.

1. Utwórz tabelę tymczasową **temp_order_details**, w której zostaną zapisane **product_id**, **unit_price** i **quantity** z tabeli **order_details**.
2. Wyznacz wartość sumy **quantity * unit_price**, dla wszystkich rekordów z **temp_order_details**
3. Chcemy symulować zmiany ceny i wielkości sprzedaży dla produktu nr 1, dlatego najpierw z tabeli **temp_order_details** usuń rekordy z **product_id=1**
4. Do tabeli **temp_order_details** dopisz jeszcze raz rekordy z tabeli **order_details**, ale tylko te, które dotyczą sprzedaży produktu numer 1 (**product_id=1**). Podczas konstruowania polecenia **SELECT** wybierającego te rekordy przyjmujemy dodatkowo następujące założenia;
 - a. Cena produktów zostanie obniżona o 10%, nowa cena to więc **unit_price * 0.9**
 - b. Dzięki promocji liczba sprzedanych produktów będzie rosła o 20%, więc **quantity** to teraz **quantity * 1.2**
5. Uruchom ponownie polecenie z pkt. 2, żeby sprawdzić całkowitą wartość sprzedaży

Sprawdź się!

1. Czy tabela wykorzystywana w **INSERT INTO ... SELECT** musi już wcześniej istnieć?
2. Co można powiedzieć o strukturze tabeli do której trafiają nowe dane, w odniesieniu do zapytania, które zasila tą tabelę?

Propozycja rozwiązania

```
SELECT product_id, unit_price, quantity INTO TEMP temp_order_details FROM
order_details;
```

```
SELECT SUM(quantity * unit_price) FROM temp_order_details;
--1354458
```

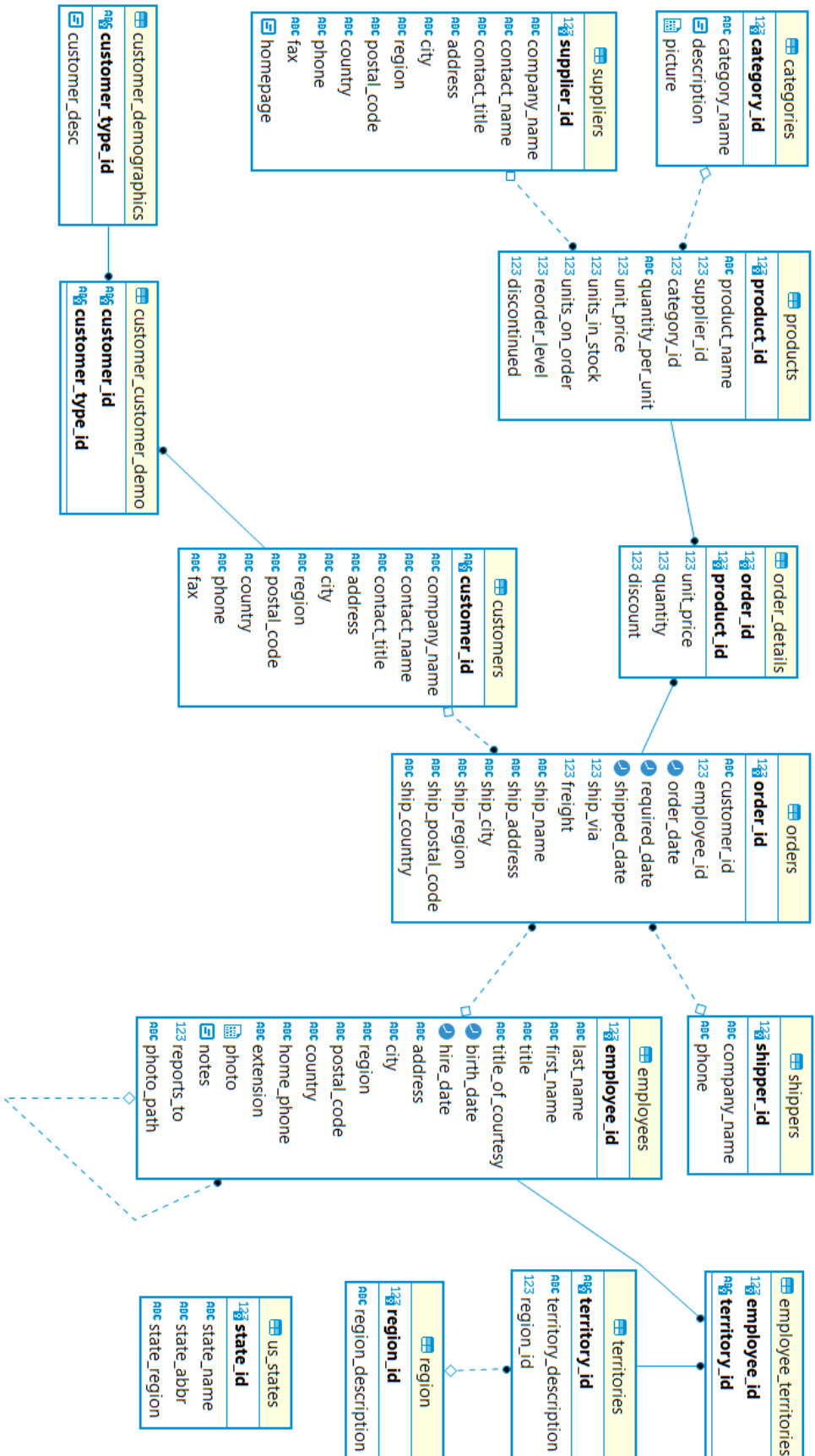
```
DELETE FROM temp_order_details WHERE product_id = 1
```

```
INSERT INTO temp_order_details
SELECT product_id, unit_price * 0.9, quantity*1.2
FROM order_details
WHERE product_id = 1
```

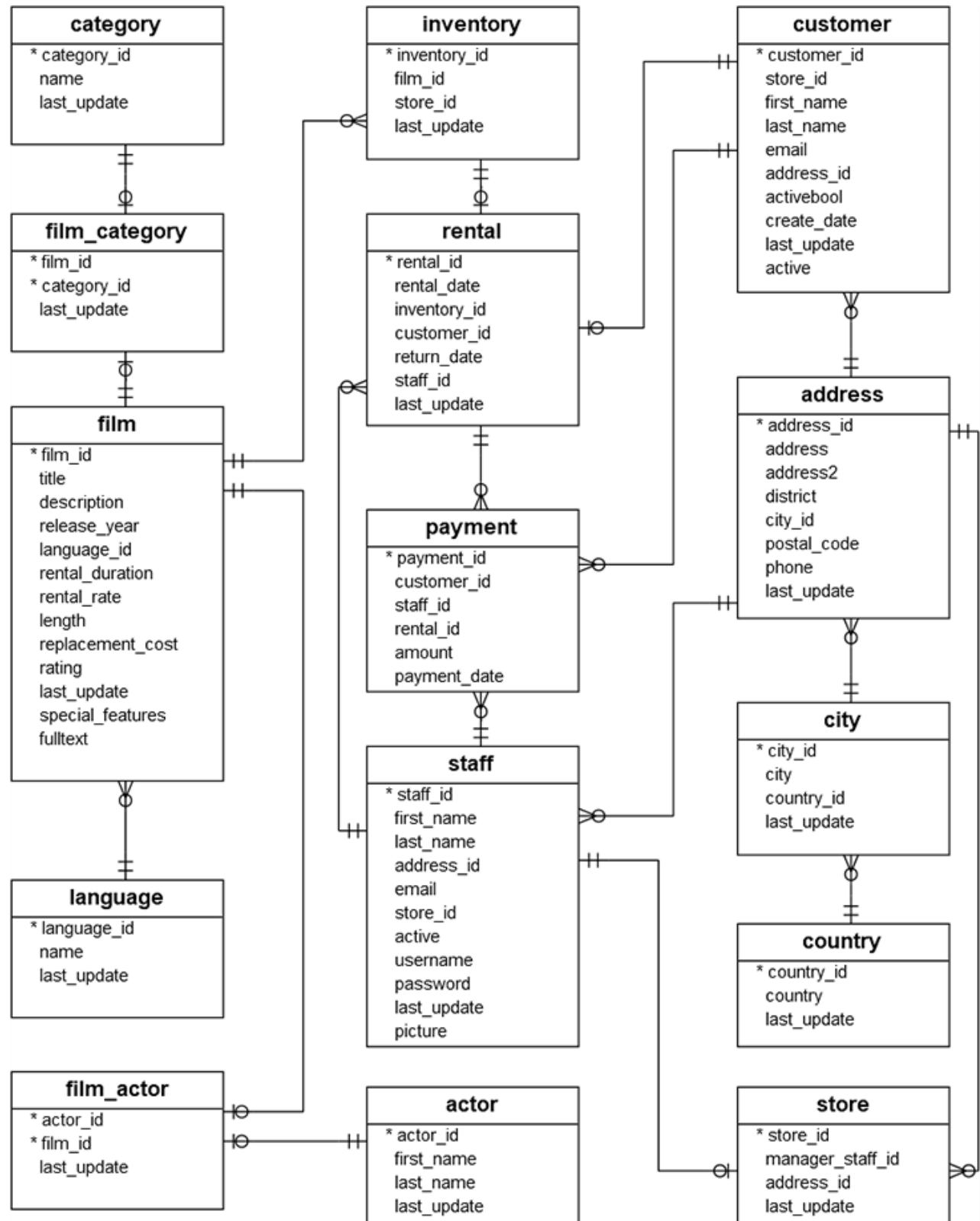
```
SELECT SUM(quantity * unit_price) FROM temp_order_details;
--1355622
```

```
DROP TABLE temp_order_details;
```


Dodatek - Diagram bazy danych Northwind



Dodatek - Diagram bazy danych DVD Rental



Notatka – kroki instalacji z komentarzem:

```
# aktualizacja repozytoriów / sprawdzenie konfiguracji repozytoriów
sudo yum -y update
sudo dnf install
https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-
redhat-repo-latest.noarch.rpm
sudo dnf -qy module disable postgresql
sudo dnf repolist

# sprawdzenie dostępnych pakietów i instalacja postgresql
sudo yum search postgresql13
sudo dnf install postgresql13 postgresql13-server

# konfiguracja postgresql – inicjowanie instalacji
sudo /usr/pgsql-13/bin/postgresql-13-setup initdb

# wyświetlenie plików konfiguracyjnych postgresql
ls /var/lib/pgsql/13/data/

# konfiguracja automatycznego uruchamiania postgresql ze sprawdzeniem
sudo systemctl enable --now postgresql-13
sudo systemctl status postgresql-13

# definicja hasła dla użytkownika bazy danych postgres
$ sudo su - postgres
$ psql -c "alter user postgres with password 'StrongDBPassword'"

# sprawdzenie adresu IP – przyda się do konfiguracji sieci
ip address

# definicja interfejsów sieciowych, na których ma nasłuchiwać postgresql
sudo vim /var/lib/pgsql/13/data/postgresql.conf
listen_addresses = '*'

# definicja wymogów bezpieczeństwa przy połączeniu
sudo vim /var/lib/pgsql/13/data/pg_hba.conf
# Accept from trusted subnet (Recommended setting)
host all all 192.168.171.0/24 md5
# Accept from anywhere (not recommended)
host all all 0.0.0.0/0 md5

# restart postgresql po konfiguracji
sudo systemctl restart postgresql-13

# testowanie lokalnego połączenia
psql -U postgres -h localhost -p 5432 postgres
SELECT version();
quit;

# otwarcie portów, na poziomie systemu operacyjnego
sudo firewall-cmd --add-service=postgresql --permanent
sudo firewall-cmd --reload
sudo firewall-cmd --list-all
```

Notatka:

- Strona domowa pgAdmin <https://www.pgadmin.org/>
- Etapy instalacji pgAdmin

```
# czyszczenie starych repozytoriów
sudo rpm -e pgadmin4-fedora-repo
sudo rpm -e pgadmin4-redhat-repo

# instalacja nowego repozytorium
sudo rpm -i https://ftp.postgresql.org/pub/pgadmin/pgadmin4/yum/pgadmin4-redhat-repo-1-1.noarch.rpm

# instalacja pgAdmin (wersja desktop i web)
sudo yum install pgadmin4

# konfiguracja aplikacji pgAdmin 4
sudo /usr/pgadmin4/bin/setup-web.sh
```

- Podczas połączenia do pgAdmin domyślnie należy przejść podwójną weryfikację:
 - Czy użytkownik może korzystać z pgAdmin
 - Czy użytkownik ma dostęp do zarządzanego serwera PostgreSQL
- Aby skonfigurować http do połączeń z zewnątrz należy otworzyć porty:

```
# listowanie otwartych wyjątków
sudo firewall-cmd --list-all

# definiowanie nowego wyjątku
sudo firewall-cmd --add-service=http --permanent

# przeładowanie konfiguracji firewall-a
sudo firewall-cmd --reload
```

```
[student@localhost ~]$ sudo /usr/pgadmin4/bin/setup-web.sh
[sudo] password for student:
Setting up pgAdmin 4 in web mode on a Redhat platform...
Creating configuration database...
NOTE: Configuring authentication for SERVER mode.

Enter the email address and password to use for the initial pgAdmin user account:

Email address: student@example.local
Password:
Retype password:
pgAdmin 4 - Application Initialisation
=====

Creating storage and log directories...
Configuring SELinux...
The Apache web server is not running. We can enable and start the web server for you to fix
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd
Apache successfully enabled.
Apache successfully started.
You can now start using pgAdmin 4 in web mode at http://127.0.0.1/pgadmin4
```


Spróbuj też!

