



JĘZYK SQL

dla początkujących

Opracowano przez: Sebastian Stasiak

Spis treści

1.	Wstęp do relacyjnych baz danych.....	3
1.1.	Podstawowe pojęcia związane z bazami SQL	3
1.2.	Podział języka SQL	4
1.3.	Narzędzia baz danych	5
1.4.	Microsoft SQL Management Studio.....	6
2.	Język DQL (Data Query Language) – kwerendy wybierające.....	14
2.1.	Pobieranie rekordów z użyciem SELECT	15
2.2.	Zapisywanie kwerendy - widok	20
2.3.	Modyfikacja zapisanego widoku	21
2.4.	Usuwanie istniejącego widoku	21
2.5.	Generowanie kodu SQL	21
2.6.	Operacje na połączonych tabelach	22
2.7.	Łączenie wyników zapytania	23
2.8.	Kwerendy zagnieżdżone	24
3.	Funkcje	26
3.1.	Funkcje tekstowe - zastosowania.....	27
3.2.	Funkcje matematyczne - zastosowania	30
3.3.	Funkcje daty i czasu - zastosowania	33
3.4.	Funkcje konwersji - zastosowania.....	35
3.5.	Funkcje specjalne - zastosowania.....	37
3.6.	Funkcje agregujące - zastosowania	38
4.	Język DML (Data Manipulation Language)	39
4.1.	Tworzenie tabeli SELECT INTO	39
4.2.	Dodawanie rekordów INSERT INTO	39
4.3.	Aktualizacja rekordów UPDATE.....	39
4.4.	Usuwanie rekordów DELETE.....	39
5.	Język DDL (Data Definition Language).....	40
5.1.	Tworzenie bazy danych	40
5.2.	Usuwanie bazy danych	40
5.3.	Tworzenie tabeli	40
5.4.	Modyfikacja tabeli.....	40
5.5.	Usuwanie tabeli	40
5.6.	Relacje	41
5.7.	Indeksy	44
6.	Zaawansowane zapytania wybierające SQL	45
6.1.	Zagnieżdżanie zapytań.....	45
6.2.	Obsługa NULL	46
6.3.	Funkcje okien	46
6.4.	CTE (Common Table Expression).....	49
7.	Wprowadzenie do programowania w T-SQL	51
7.1.	Procedury składowane	51
7.2.	Procedury systemowe	53
7.3.	Tworzenie obiektów z użyciem SQL	55
7.5.	Tworzenie tabeli	56
7.6.	Pętla WHILE	57
7.7.	Funkcja warunkowe	59
7.8.	Zmienne.....	60
7.9.	Tabele tymczasowe.....	61
7.10.	Tworzenie funkcji	63

Materiały szkoleniowe

7.11.	TRIGGER - wyzwalacz	63
7.12.	Zdalne uruchamianie procedur	65
7.13.	Kursory SQL	65
8.	Zaawansowane procedury SQL	69
8.1.	Tworzenie bazy danych	69
8.2.	Nazwa tabeli ze zmiennej	73
8.3.	Seryjne tworzenie tabel	73
8.4.	Procedura dodająca wpis do tabeli log	74
8.5.	Obsługa i przechwytywanie błędów	75
8.6.	Sterowanie przebiegiem programu	76
8.7.	Transakcje	78
8.8.	Zabezpieczenia danych	81
9.	Hurtownie danych	83
9.1.	Linked server	83
9.2.	Openquery dla Linked Server	84
9.3.	Rozproszone źródła danych	86
10.	Podstawy administracji serwera	88
10.1.	Instalacja serwera SQL	88
10.2.	Instalacja Microsoft SQL Server Management Studio	90
10.3.	Kopia zapasowa	91
10.4.	Zarządzanie uprawnieniami	91
10.5.	Uprawnienia	92
10.6.	Śledzenie błędów serwera	93
11.	Komunikacja z serwerem SQL – import i eksport danych	94
11.1.	BULK Import	94
11.2.	Konfiguracja połączenia ODBC	95
11.3.	Połączenie SQL Server – Excel	98
11.4.	Połączenie SQL Server – Access	101
12.	Dodatki	103
12.1.	Baza danych MySQL	103
12.2.	Funkcje w języku SQL dla SQL Server	104
12.3.	Składnia SQL	106
12.4.	Operatory SQL	107
12.5.	Słowa kluczowe SQL	107
12.6.	Typy danych SQL Server	109

1. Wstęp do relacyjnych baz danych

1.1. Podstawowe pojęcia związane z bazami SQL

Baza danych – zbiór danych zapisanych zgodnie z określonymi regułami. W węższym znaczeniu obejmuje dane cyfrowe gromadzone zgodnie z zasadami przyjętymi dla danego programu komputerowego specjalizowanego do gromadzenia i przetwarzania tych danych. Program taki nazywany jest „systemem zarządzania bazą danych” (ang. database management system, DBMS).

Baza relacyjna to model, gdzie wiele tabel danych może współpracować ze sobą (są między sobą powiązane). Posiadają one zwykle wewnętrzne języki programowania, wykorzystujące język SQL do operowania na danych.

Tabela (relacja) – wydzielony logicznie zbiór danych, zorganizowanych w formie tabeli składającej się z wierszy dzielonych na kolumny. Pojedyncza tabela może być reprezentacją pewnej encji (np. książek, mieszkań, ludzi), relacji między nimi, albo może stanowić zawartość całej bazy danych.

Rekord bazy danych – to pojedynczy wiersz tabeli nazywany stanowiący najczęściej zbiór informacji o pojedynczym obiekcie.

Kolumna w relacyjnym modelu baz danych stanowi zwykle atrybut jakiegoś obiektu (np. wielkość, grubość, tytuł, nazwisko) i stąd dane zawarte w kolumnach mają najczęściej jeden określony typ. Dodatkowo w bazach obsługiwanych przez język SQL kolumnom nadawane są nazwy, które są unikatowe w obrębie jednej tabeli.

Klucz główny (ang. primary key) – wybrany minimalny zestaw atrybutów relacji, jednoznacznie identyfikujący każdą krotkę tej relacji. To oznacza, że taki klucz musi przyjmować wyłącznie wartości niepowtarzalne i nie może być wartością pustą (null). Ponadto każda relacja może mieć najwyżej jeden klucz główny.

Klucz obcy – kombinacja jednego lub wielu atrybutów tabeli, które wyrażają się w dwóch lub większej liczbie relacji. Wykorzystuje się go do tworzenia relacji pomiędzy parą tabel, gdzie w jednej tabeli ten zbiór atrybutów jest kluczem obcym, a w drugiej kluczem głównym.

SQL (ang. Structured Query Language) – strukturalny język zapytań używany do tworzenia, modyfikowania baz danych oraz do umieszczania i pobierania danych z baz danych. Język SQL jest językiem deklaratywnym. Decyzję o sposobie przechowywania i pobrania danych pozostawia się systemowi zarządzania bazą danych (DBMS).

SQL został opracowany w latach 70. w firmie IBM. Stał się standardem w komunikacji z serwerami relacyjnych baz danych. Wiele współczesnych systemów relacyjnych baz danych używa do komunikacji z użytkownikiem SQL, dlatego potocznie mówi się, że korzystanie z relacyjnych baz danych to korzystanie z SQL-a.

Pierwszą firmą, która włączyła SQL do swojego produktu komercyjnego, był Oracle. Dalsze wprowadzanie SQL-a, w produktach innych firm, wiązało się nierozłącznie z wprowadzaniem modyfikacji pierwotnego języka. Wkrótce utrzymanie dalszej jednolitości języka wymagało wprowadzenia standardu.

Standardy SQL

W 1986 SQL stał się oficjalnym standardem, wspieranym przez Międzynarodową Organizację Normalizacyjną (ISO) i jej członka, Amerykański Narodowy Instytut Normalizacji (ANSI).

- SQL86 i SQL89 określały płaszczyznę łączącą istniejące wówczas produkty.
- SQL92, obowiązujący w produktach komercyjnych do dziś.
- SQL:2003 – nowy standard języka SQL. Został on opublikowany w 2004 roku.

Silniki baz danych SQL

Istnieje wiele różnorodnych systemów bazodanowych używających języka SQL:

DB2, Firebird, First SQL, Greenplum, HSQL, Ingres, Informix, InterBase SQL, MariaDB, MaxDB, Microsoft Access, Microsoft Jet, Microsoft SQL Server, Mimer SQL, MySQL, mSQL, Netezza, Oracle Database, Oracle Rdb, PostgreSQL, Pervasive, SQL/DS., SQLite, Sybase, Teradata.



Microsoft SQL Server (MS SQL) – system zarządzania bazą danych, wspierany i rozpowszechniany przez korporację Microsoft.



MySQL – wolnodostępny system zarządzania relacyjnymi bazami danych. MySQL rozwijany jest przez firmę Oracle.



PostgreSQL nazywany także Postgres to jeden z najpopularniejszych wolnodostępnych systemów zarządzania bazami danych.



Oracle Database – system zarządzania relacyjnymi bazami danych stworzony przez firmę Oracle Corporation

1.2. Podział języka SQL

Użycie SQL, zgodnie z jego nazwą, polega na zadawaniu zapytań do bazy danych. Zapytania można zaliczyć do jednego z czterech głównych podzbiorów:

- **SQL DML** (ang. Data Manipulation Language – „język manipulacji danymi”),
 - **DQL** - Data Query Language – Pobieranie danych w ramach DML
- **SQL DDL** (ang. Data Definition Language – „język definicji danych”),
- **SQL DCL** (ang. Data Control Language – „język kontroli nad danymi”).
- **SQL TCL** (Transactional Control Language) – obsługa transakcji.

Instrukcje SQL w obrębie zapytań tradycyjnie zapisywane są wielkimi literami, jednak nie jest to wymóg. Dodatkowo, niektóre programy do łączenia się z silnikiem bazy danych (np. psql w przypadku PostgreSQL) używają swoich własnych instrukcji, spoza standardu SQL, które służą np. do połączenia się z bazą, wyświetlenia dokumentacji itp.

DML - Data Manipulation Language

Służy do wykonywania operacji na danych – do ich umieszczania w bazie, kasowania, przeglądania oraz dokonywania zmian. Najważniejsze polecenia z tego zbioru to:

- **INSERT** – umieszczenie danych w bazie
- **UPDATE** – zmiana danych
- **DELETE** – usunięcie danych z bazy

Dane tekstowe muszą być zawsze ujęte w znaki pojedynczego cudzysłowu (').

DML / DQL - Data Query Language

DQL to język formułowania zapytań do bazy danych. W zakres tego języka wchodzi jedno polecenie – SELECT. Często SELECT traktuje się jako część języka DML, ale to podejście nie wydaje się właściwe, ponieważ DML z definicji służy do manipulowania danymi – ich tworzenia, usuwania i uaktualniania. Na pograniczu obu języków znajduje się polecenie SELECT INTO, które dodatkowo modyfikuje (przepisuje, tworzy) dane.

DDL - Data Definition Language

Dzięki DDL można operować na strukturach, w których dane są przechowywane – czyli np. dodawać, zmieniać i kasować tabele lub bazy. Najważniejsze polecenia tej grupy to:

- **CREATE** (np. CREATE TABLE, CREATE DATABASE, ...) – utworzenie struktury (bazy, tabeli, indeksu itp.),
- **DROP** (np. DROP TABLE, DROP DATABASE, ...) – usunięcie struktury,
- **ALTER** (np. ALTER TABLE ADD COLUMN ...) – zmiana struktury (dodanie/zmian kolumny).

DCL - Data Control Language

DCL ma zastosowanie do nadawania uprawnień do obiektów bazodanowych. Najważniejsze polecenia w tej grupie to:

- **GRANT** – służące do nadawania uprawnień do pojedynczych obiektów lub globalnie użytkownikowi.
- **REVOKE** – służące do odbierania wskazanych uprawnień konkretnemu.

- **DENY** - służące do zabrania operacji.

TCL - Transactional Control Language

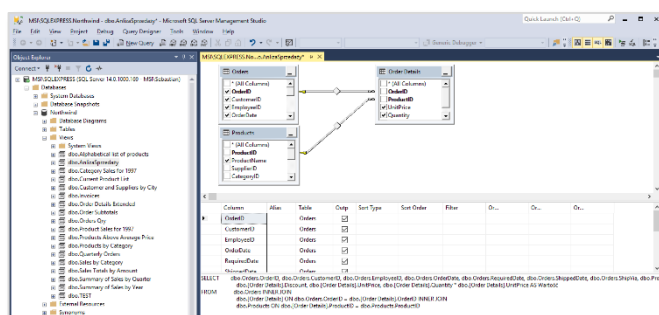
Polecenia języka TCL: COMMIT oraz ROLLBACK, wykorzystywane głównie podczas pracy z transakcjami. Polecenie COMMIT służy właśnie do zatwierdzenia transakcji, w przeciwieństwie do ROLLBACK, który odrzuca zmiany, jakie miała wprowadzić.

1.3. Narzędzia baz danych

Do pracy z każdym serwerem SQL niezbędne są odpowiednie narzędzia, umożliwiające tworzenie kodu i zarządzanie serwerem bazy danych SQL. Dla każdego z serwerów (SQL Server i MySQL) dostępne są niezależne narzędzia dostarczone przez producentów obu platform. Oba narzędzia pozwalają na tworzenie kodu w trybie tekstowym jak i graficznym. Dodatkowo pozwalają na tworzenie struktury bazy z użyciem diagramów bazy danych.

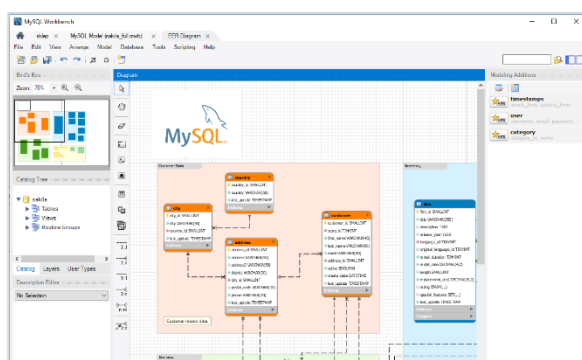
SQL Server

SQL Server Management Studio - zintegrowane środowisko do zarządzania wszystkimi komponentami (baza danych, usługi analityczne, usługi raportowe itd.), wchodzącymi w skład Microsoft SQL Server. Zawiera narzędzia do konfiguracji, monitorowania i administrowania SQL Server. Umożliwia budowę zapytań i skryptów, zawiera zarówno edytor skryptów jak i narzędzia graficzne. Główną cechą aplikacji jest Object Explorer, który pozwala na przeglądanie, wybieranie i wykonywanie działań na obiektach serwera.



MySQL

MySQL Workbench to ujednolicone wizualne narzędzie dla architektów baz danych, programistów i administratorów baz danych. Zapewnia modelowanie danych, tworzenie kodu SQL i wszechstronne narzędzia administracyjne do konfiguracji serwerów, administracji użytkownikami, tworzenia kopii zapasowych i wiele więcej



```

Wiersz polecenia - osql -S msi/sqlservr -U sa
2> go
EmployeeID  LastName      FirstName
-----
1 Davolio      Nancy
2 Fuller       Andrew
3 Leverling    Janet
4 Peacock      Margaret
5 Buchanan     Steven
6 Suyama       Michael
7 King         Robert
8 Callahan     Laura
9 Dodsworth    Anne

(9 rows affected)
1> SELECT EmployeeID, LastName, FirstName FROM Employees
2> go
EmployeeID  LastName      FirstName
-----
1 Davolio      Nancy
2 Fuller       Andrew
3 Leverling    Janet
4 Peacock      Margaret
5 Buchanan     Steven
6 Suyama       Michael
7 King         Robert
8 Callahan     Laura
9 Dodsworth    Anne

(9 rows affected)
1>
    
```

Alternatywną metodą zarządzania bazami danych jest zastosowanie narzędzi dostępnych w wersji dla wiersza poleceń. Dla SQL Server jest to OSQL.

1.4. Microsoft SQL Management Studio

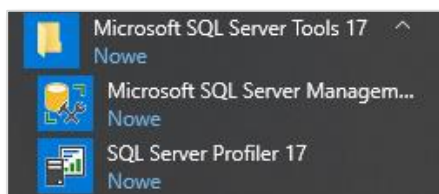
Do pracy z bazami SQL Server zalecane jest stosowanie środowiska Microsoft SQL Management Studio z uwagi na jego wszechstronność oraz rozbudowane możliwości. Niewątpliwym jego atutem, jest fakt, iż jest ono bezpłatne.

Logowanie do serwera SQL

Do poprawnego wykonania ćwiczeń znajdujących się w niniejszym podręczniku niezbędne jest posiadanie zainstalowanego oprogramowania Microsoft SQL Server w dowolnej wersji – zalecana jest jak najnowsza z uwagi na dostępne nowsze funkcjonalności, jak również na większą niezawodność. Przykładowe ćwiczenia wykonywane w podręczniku są realizowane w oparciu o wersję 2016 Developer Edition. Zalecaną wersją narzędzi są wersje Express (bezpłatne do użytku komercyjnego).

Niektóre elementy programu mogą się nieznacznie różnić w stosunku do innych wersji, ale większość przykładów jest zgodna z wersjami SQL Server od 2008 wzwyż.

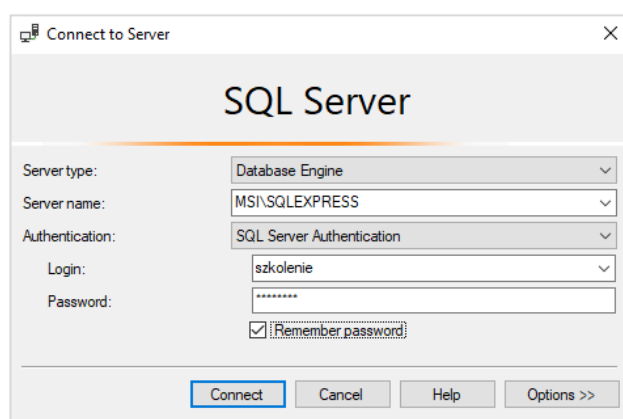
Informacja: Opis samodzielnej instalacji serwera SQL w bezpłatnej wersji Express znajduje się w dodatku na końcu podręcznika.



Pierwszą czynnością jaką wykonasz w swojej pracy z serwerem będzie logowanie do serwera SQL, które wykonasz korzystając z programu Microsoft SQL Management Studio. Odpowiednie parametry uzyskasz od swojego działu IT lub zdefiniujesz sam instalując serwer SQL.

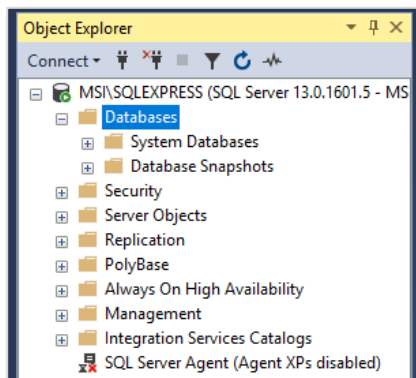
Informacje niezbędne przy logowaniu to:

Typ serwera:	Server type:	Database Engine
Adres serwera:	Server name:	serwer\SQLEXPRESS
Typ autoryzacji:	Authentication:	SQL Server Authentication
Login:	Login:	loginsql
Hasło:	Password:	hasłosql
Zapamiętaj hasło:	Remember password:	TAK



Język SQL dla początkujących

Materiały szkoleniowe

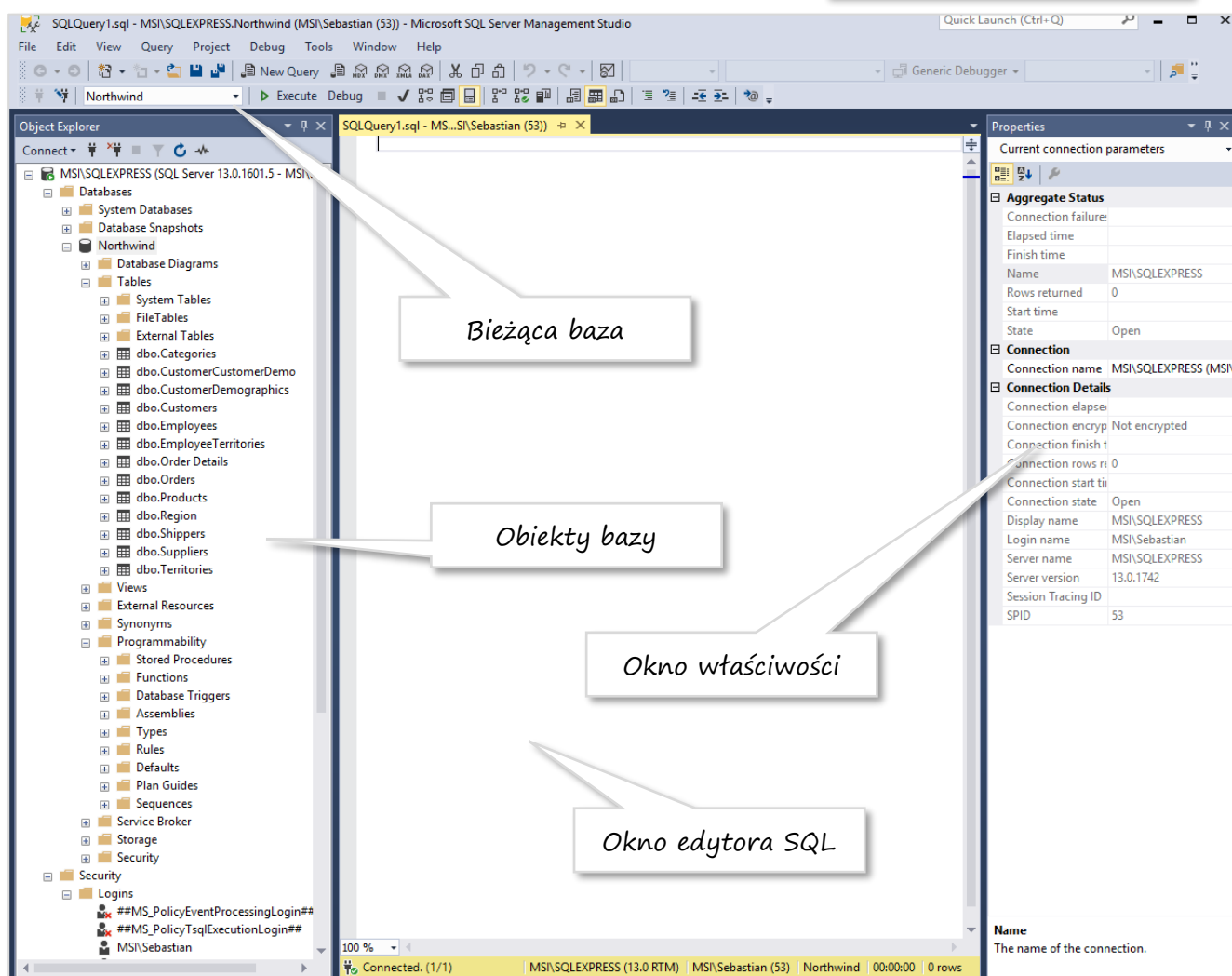


Poprawne logowanie pozwoli na przejście do programu, równocześnie łącząc się z serwerem. Podstawowym elementem programu jest Object Explorer, który służy do nawigacji po obiektach podłączonego SQL Server.

W przypadku, gdy instalacja serwera została wykonana przez dział IT po rozwinięciu gałęzi Databases powinna pokazać się szukana baza danych.

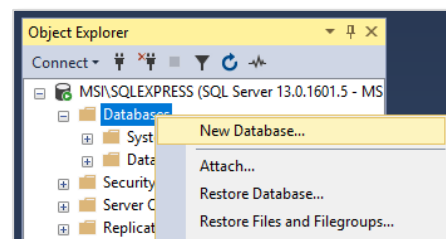
Przydatne: Widoczność obiektów bazy danych jest uzależniona od posiadanych uprawnień. W przypadku braku widoczności obiektów skonsultuj się z administratorem serwera.

Interfejs programu



Tworzenie bazy danych

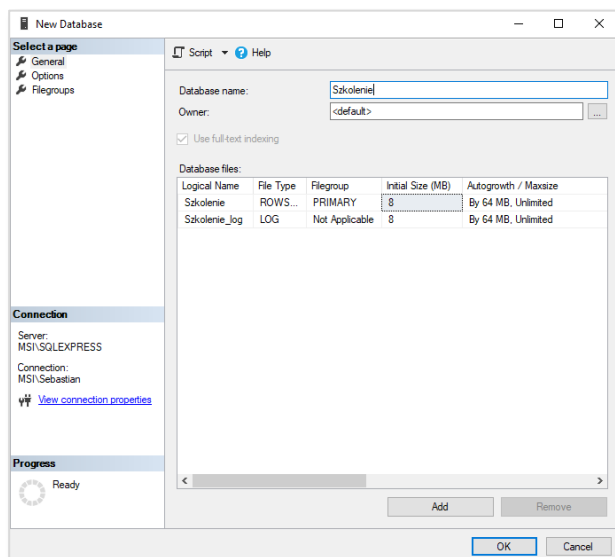
Całość szkolenia jest przeprowadzana na skonfigurowanej i przygotowanej bazie danych. Podczas pracy samodzielnej konieczne może być utworzenie bazy roboczej we własnym zakresie. Możesz tego dokonać klikając prawym klawiszem myszy na gałęzi **Databases** i wybierając **New Database**.



Przydatne: W dalszej części podręcznika zostały przedstawione metody tworzenia obiektów z użyciem skryptu języka SQL.

Nadaj bazie danych nazwę Szkolenie i zatwierdź ok. Opcjonalnie korzystając z tego okna możesz również przypisać administratora bazy danych oraz pozostałych użytkowników i nadać im uprawnienia.

Uprawnienia możesz nadawać w ramach całej bazy lub dla poszczególnych jej obiektów takich jak tabele, widoki, czy procedury.

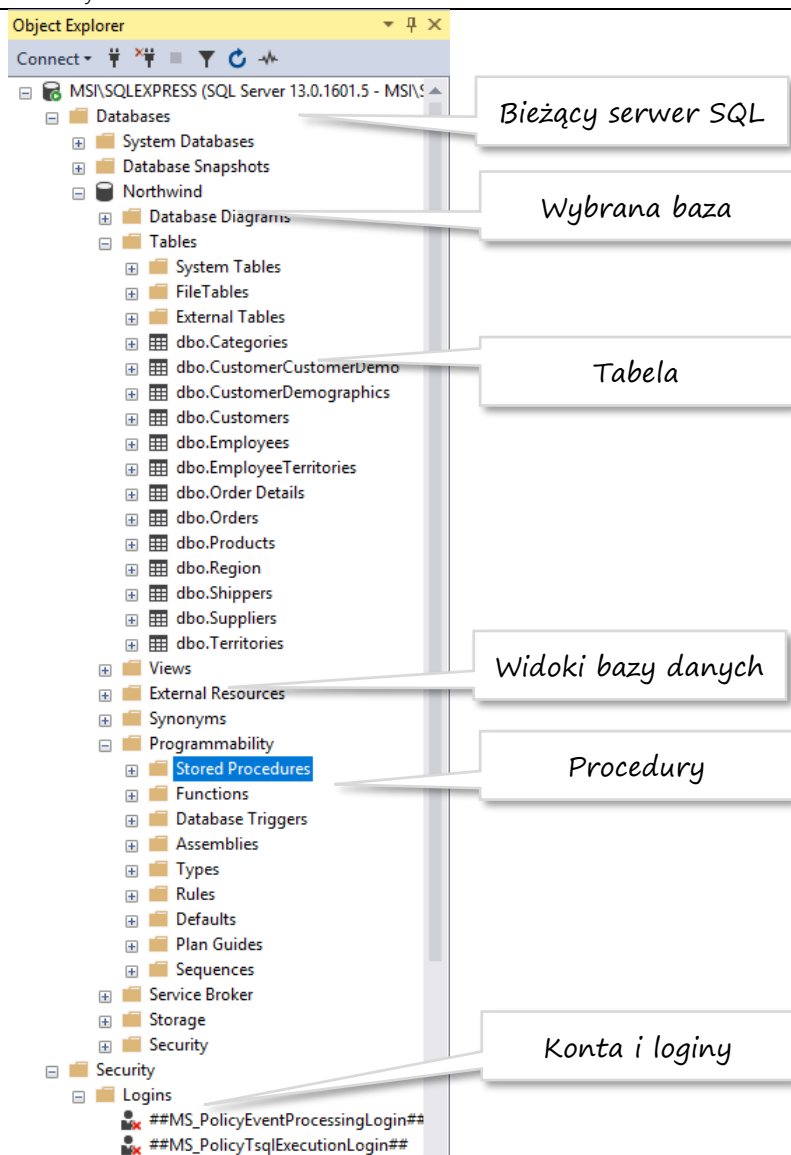


Podstawowe typy dostępu do bazy danych to:

db_datareader	tylko odczyt
db_datawriter	zapis
db_owner	właściciel
db_backupoperator	operator kopii zapasowej

Uprawnienia możesz nadać i zmodyfikować w dowolnym momencie.

Zakończ konfigurowanie bazy danych klikając **OK**.



Baza danych - zbiór danych zapisanych zgodnie z określonymi regułami. W węższym znaczeniu obejmuje dane cyfrowe gromadzone zgodnie z zasadami przyjętymi dla danego programu komputerowego specjalizowanego do gromadzenia i przetwarzania tych danych.

Tabela - wydzielony logicznie zbiór danych, zorganizowanych w formie tabeli składającej się z wierszy dzielonych na kolumny

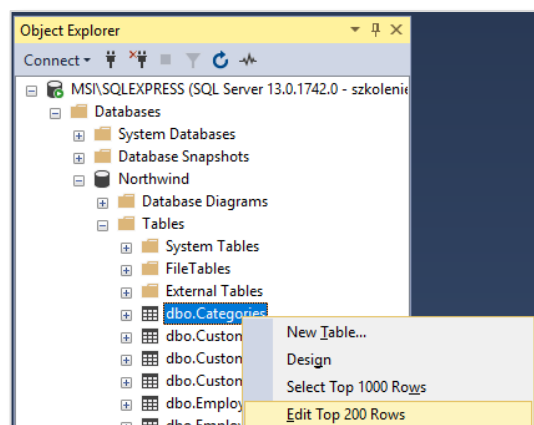
Widok (perspektywa) to logiczny byt (obiekt), osadzony na serwerze baz danych. Umożliwia dostęp do podzbioru kolumn i wierszy tabel lub tabeli na podstawie zapytania w języku SQL, które stanowi część definicji tego obiektu. Przy korzystaniu z widoku jako źródła danych należy odwoływać się identycznie jak do tabeli.

Procedura składowana to element bazy danych, który jest umiejscowiony bezpośrednio w systemie bazy danych, a nie po stronie klienta.

Edycja danych na serwerze SQL

Efektom podania poprawnych danych jest pojawienie się **Object explorer**a i wyświetlenie dostępnych baz danych dla wskazanego loginu. Jest to także dobry moment na sprawdzenie uprawnień i poprawności konfiguracji konta klienckiego.

Rozwiń gałąź **Databases**, następnie **Tables**. Znajdują się tutaj dostępne w bazie danych tabele. Możesz zaznaczyć jedną z nich prawym klawiszem myszy i wybrać **Edit Top 200 Rows**.



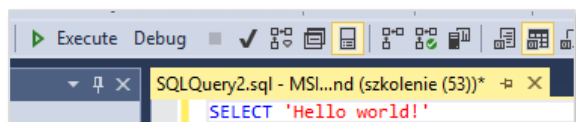
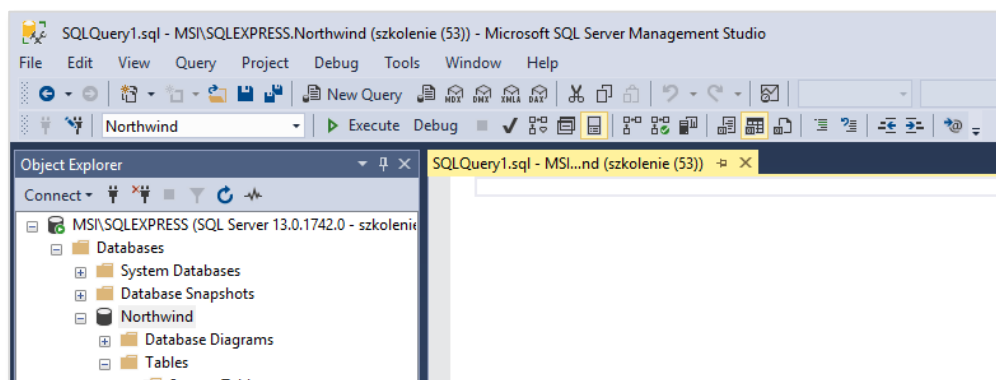
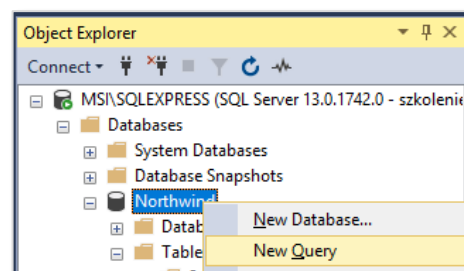
Materiały szkoleniowe

Tabela, która się wyświetli, będzie w trybie edycji. Spróbuj zmienić istniejące dane, jeśli konto dostępowe jest w trybie tylko do odczytu będzie to niemożliwe. W przypadku posiadania uprawnień odczyt zapis edycja będzie możliwa. Po zakończeniu testu koniecznie zamknij okno edytor tabeli.

CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coff...	<Binary data>
2	Condiments	Sweet and savo...	<Binary data>
3	Confections	Desserts, candi...	<Binary data>
4	Dairy Products	Breads, crackers...	<Binary data>
5	Grains/Cereals	Prepared meats	<Binary data>
6	Meat/Poultry	Dried fruit and ...	<Binary data>
7	Produce	Seaweed and fish	<Binary data>
8	Seafood		

Po wstępnej weryfikacji swoich uprawnień postanowiłeś przetestować działanie bazy przez stworzenie pierwszej kwerendy.

Kliknij na nazwie bazy w **Object Explorerze** i wybierz opcję **New Query**. Okno, które się w wyniku tego pojawi pozwoli na stworzenie kwerendy w trybie tekstowym.



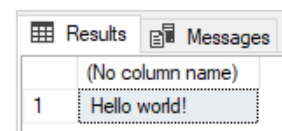
Najprostszym sposobem przetestowania systemu jest wykonanie najprostszej kwerendy. W oknie edytora wpisz:

```
SELECT 'Hello world!'
```

Kliknij przycisk **Execute** lub **F5** na klawiaturze. Efektem działania kwerendy będzie tekst wyświetlony w oknie **Results**. Efekt działania kwerendy jest zadowalający, co pozwala Ci przejść do dalszych działań.

Możesz także użyć polecenia **PRINT**, które przekazuje komunikat do okna **Messages**

```
PRINT 'Hello world!'
```



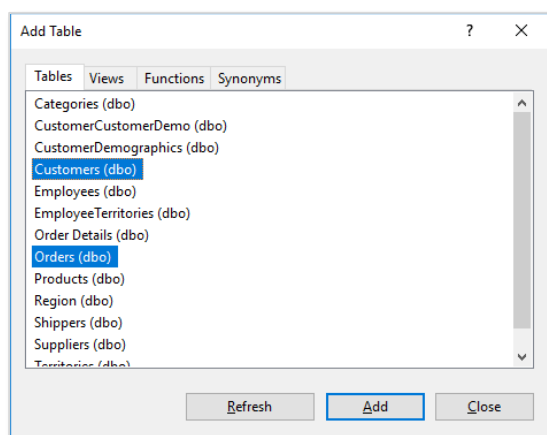
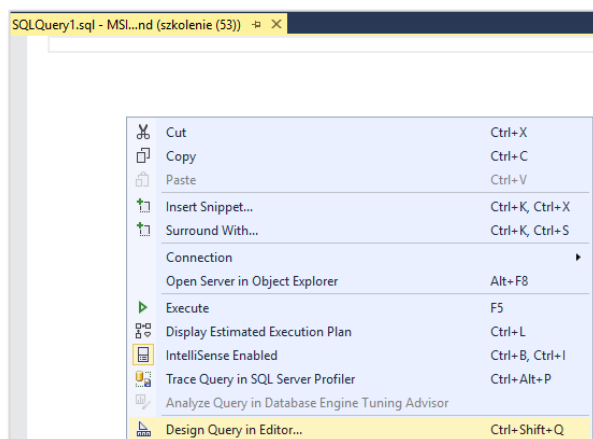
Kwerenda w trybie graficznym

Na początek postanowiłeś skorzystać z ułatwienia jakim jest niewątpliwie graficzny edytor zapytań.

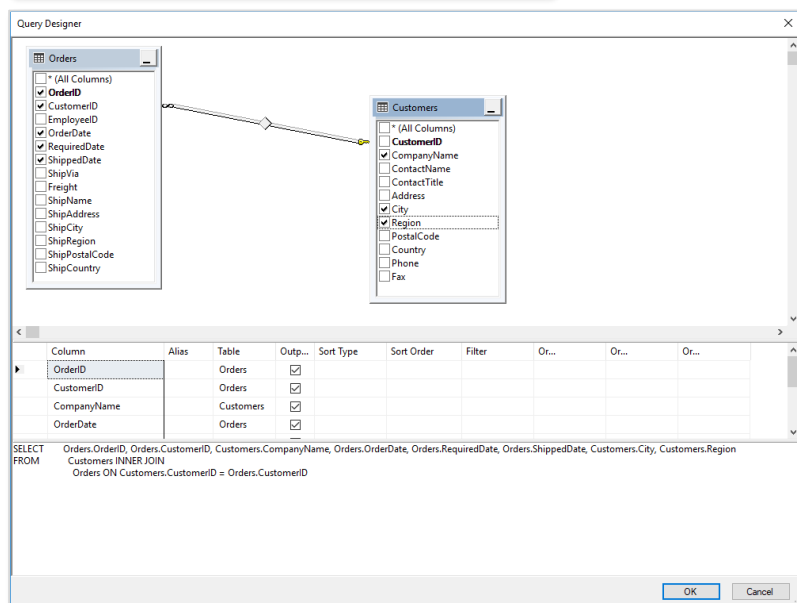
Utwórz nową pustą kwerendę. Kliknij prawym klawiszem myszy na niej i wybierz opcję **Design Query in Editor**. Możesz skorzystać ze skrótu klawiaturowego **Ctrl+Q**.

Okno **Add Table** pozwala na wskazanie tabel lub kwerend, które posłużą jako źródło danych dla nowej kwerendy. Wybierz table **Customers** i **Orders**. Możesz później dodać kolejne table w miarę potrzeb.

Niektóre table mogą być połączone za pomocą relacji. Możesz też na bieżąco niezależnie od istnienia relacji sprzęgać odpowiadające sobie pola w tabelach.



Korzystając z myszy z tabeli **Orders**: wybierz **OrderID** (klucz podstawowy), **CustomerID**, **OrderDate**, **RequiredDate**, **ShippedDate**, a z tabeli **Customers**: wybierz **CompanyName**, **City**, **Region**. Zwróć uwagę, że table są połączone za pomocą relacji opartych na kluczach głównych.



Przydatne: Relacja działa poprzez dopasowywanie danych w kolumnach klucza – zwykle kolumny o tej samej nazwie w obu tabelach. W większości wypadków relacje dopasowują klucz podstawowy z tabeli, która zawiera identyfikator z unikatowym dla każdego wiersza z wpisem w kluczu obcym w drugiej tabeli.

Efektom poniższej kwerendy jest lista zamówień połączona z nazwami kontrahentów.

```
SELECT      Orders.OrderID, Orders.CustomerID, Customers.CompanyName,
            Orders.OrderDate, Orders.RequiredDate, Orders.ShippedDate,
            Customers.City, Customers.Region
FROM        Customers INNER JOIN
            Orders ON Customers.CustomerID = Orders.CustomerID
```

Uruchom ją korzystając z przycisku F5.

OrderID	CustomerID	CompanyName	OrderDate	RequiredDate	ShippedDate	City	Region
10248	VINET	Vins et alcools Chevalier	2016-07-04 00:00:00.000	2016-08-01 00:00:00.000	2016-07-16 00:00:00.000	Reims	NULL
10249	TOMSP	Toms Spezialitäten	2016-07-05 00:00:00.000	2016-08-16 00:00:00.000	2016-07-10 00:00:00.000	Münster	NULL
10250	HANAR	Hanari Carnes	2016-07-08 00:00:00.000	2016-08-05 00:00:00.000	2016-07-12 00:00:00.000	Rio de Janeiro	RJ
10251	VICTE	Victuailles en stock	2016-07-08 00:00:00.000	2016-08-05 00:00:00.000	2016-07-15 00:00:00.000	Lyon	NULL

Obiekty serwera

Przeznaczenie obiektów serwera i ich zastosowanie

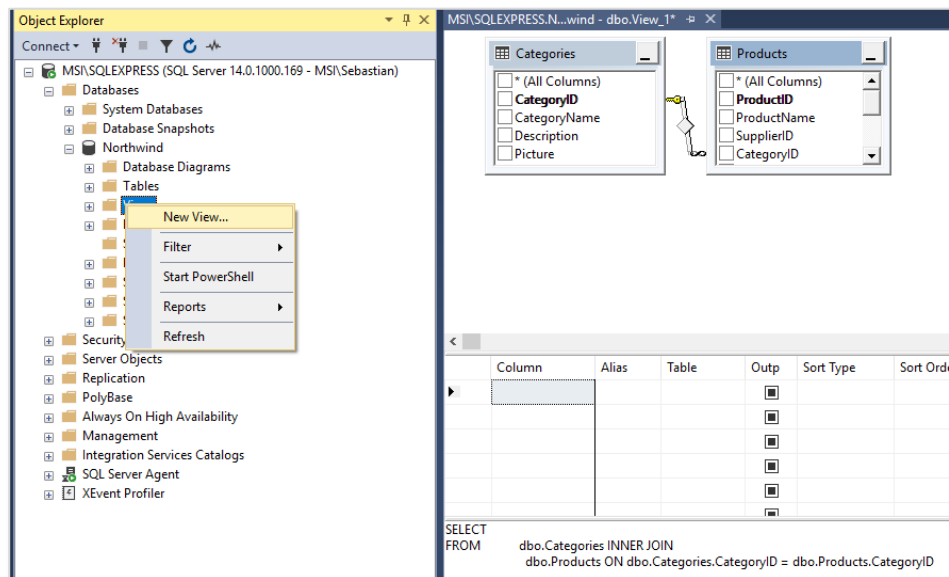
- **Tabela** – przechowuje dane w zorganizowany sposób
- **Widok** - (perspektywa) to logiczny byt (obiekt), osadzony na serwerze baz danych. Umożliwia dostęp do podzbioru kolumn i wierszy tabel lub tabeli na podstawie zapytania w języku SQL, które stanowi część definicji tego obiektu. Przy korzystaniu z widoku jako źródła danych należy odwoływać się identycznie jak do tabeli
- **Procedura** - Procedura składowana jest umiejscowiona bezpośrednio w systemie bazy danych, a nie po stronie klienta. Pozwala to na zmniejszenie liczby kroków wymiany danych pomiędzy klientem a systemem zarządzania bazą danych, co może przyczynić się do wzrostu wydajności systemu.
- **Wyzwalacz** - (ang. trigger) – procedura wykonywana automatycznie jako reakcja na pewne zdarzenia w tabeli bazy danych. Wyzwalacze mogą ograniczać dostęp do pewnych danych, rejestrować zmiany danych lub nadzorować modyfikacje danych.

Przydatne: Klucz podstawowy (ang. primary key) zwany też **kluczem głównym** to jedno lub więcej pól, których wartość jednoznacznie identyfikuje każdy rekord w tabeli. Taka cecha klucza nazywana jest **unikatowością**. Klucz podstawowy służy do wiązania rekordów w jednej tabeli z rekordami z innej tabeli.

Kwerendy wybierające w Query Designerze, zapisywanie widoków SQL

Alternatywną metodą tworzenia nowego widoku jest skorzystanie z przycisku **New View**, który pozwala na stworzenie widoku w nieco inny sposób.

Tak przygotowany widok można zapisać bezpośrednio na serwerze.



2. Język DQL (Data Query Language) – kwerendy wybierające

Praca z kodem niezależnie od używanego środowiska wymaga zastosowania pewnych reguł i praktyk związanych z czytelnością i organizacją tworzonego skryptu.

Komentarze

Podczas tworzenia kodu wskazane jest tworzenie opisów do poszczególnych jego bloków. Pomaga w tym system komentarzy. Znaki -- pozwalają na dodanie komentarza liniowego, co powoduje pominięcie tego fragmentu podczas wykonywania skryptu.

```
-- to jest komentarz liniowy  
PRINT 'Witaj'
```

Dłuższe bloki komentarzy możesz umieszczać pomiędzy znakami /* */ - jest to tzw. komentarz blokowy.

```
/* to jest komentarz blokowy  
PRINT 'Witaj'  
*/  
PRINT 'Żegnaj!'
```

Polecenie USE

Pracując ze skryptem bazy danych masz możliwość przełączania się pomiędzy różnymi bazami znajdującymi się na jednym serwerze. Jest to o tyle istotne, iż uruchomienie kodu w innej niż wymagana baza może spowodować niepożądane akcje serwera. Jednocześnie możesz się przemieszczać podczas wykonywania kodu pomiędzy różnymi bazami.

```
USE Northwind  
GO  
SELECT * FROM Customers
```

Polecenie GO

Praca z dłuższym kodem może sprawiać problemy. Część poleceń języka SQL nie może być wykonywana razem z innymi. Zastosowanie polecenia GO powoduje, że kolejne części kodu są traktowane jako niezależne od siebie skrypty.

Polecenie PRINT

Wykonując kolejne fragmenty kodu możesz mieć potrzebę przekazywania informacji diagnostycznych (messages). Możesz je wykonywać za pomocą polecenia PRINT.

```
PRINT 'Witaj'
```

Kwerendy

Kwerendy wybierające tworzone mogą być z użyciem interfejsu graficznego jak i trybu tekstowego. Podczas pracy z danymi traktowane są jak zwykłe tabele. W niektórych przypadkach nie ma możliwości zapisu za ich pośrednictwem danych w źródłowych tabelach.

2.1. Pobieranie rekordów z użyciem SELECT

Podstawowym zadaniem języka SQL jest pobieranie i prezentacja informacji pobranych z bazy danych. Dodatkowo język SQL pozwala na modyfikację i przetwarzanie danych. Otrzymałeś nową bazę danych i postanowiłeś przetestować na jej przykładzie działanie języka SQL. Otwórz nową kwerendę w widoku tekstowym. W tym oknie będziesz wyświetlał wyniki swoich prac.

Lista firm z tabeli Customers

Swoj cel możesz zrealizować za pomocą polecenia **SELECT**, które służy do wyświetlania danych pobranych z bazy. Kolejność wymienionych kolumn nie ma znaczenia. W zależności od wybranego serwera bazy danych i jego konfiguracji wielkość liter w kwerendzie może mieć znaczenie. Ta kwerenda pobiera i wyświetla wymienione w tabeli kolumny **Customers** i może mieć postać:

```
SELECT [CustomerID] , [CompanyName] , [ContactName] , [ContactTitle] , [Address]
, [City] , [Region] , [PostalCode] , [Country] , [Phone] , [Fax]
FROM [Northwind].[dbo].[Customers]
```

Istnieje możliwość uproszczenia zapisu kwerendy poprzez usunięcie nawiasów kwadratowych (o ile w nazwie kolumny nie ma spacji i nie jest zastrzeżoną instrukcją SQL). Przy okazji możesz zastosować krótszą nazwę tabeli, bez informacji o bazie danych, ponieważ kwerenda oparta jest na jednej tabeli, z jednej bazy danych, dlatego nie musisz dodawać informacji o bazie danych i prefiksu **[dbo]**.

```
SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address,
City, Region, PostalCode ,Country, Phone, Fax
FROM Customers
```

W związku z tym, że chcesz wyświetlić wszystkie pola tabeli, możesz ją uprościć do następującej formy (znak ***** oznacza wszystkie kolumny):

```
SELECT *
FROM Customers
```

Ze względu na obowiązujące standardy zapisu kwerend dla języka SQL istnieje możliwość oznaczania nazw obiektów z użyciem cudzysłowów zamiast nawiasów kwadratowych.

```
SELECT *
FROM "Customers"
```

Sortowanie wyniku zapytania

Zadanie: Przygotuj listę klientów posortowaną rosnąco wg kraju i malejąco wg miasta.

Podpowiedź: Sortowanie wyniku kwerendy SQL domyślnie jest rosnące, dlatego też **ASC** jest opcjonalne.

```
SELECT *
FROM Customers
ORDER BY Country ASC, City DESC
```

Lista klientów z Niemiec

Zadanie: Otrzymałeś zadanie pobrania listy klientów z Niemiec.

Podpowiedź: Ciągi tekstowe w kwerendach SQL muszą być otoczone za pomocą apostrofów.

```
SELECT *
FROM Customers
WHERE Country = 'Germany'
```


Klienci z Niemiec lub Francji

Zadanie: Twoja poprzednia lista jest niewystarczająca. Musisz dodać do wyniku kwerendy firmy z Francji.

Podpowiedź: Operator OR – logiczne Lub pozwala na podanie więcej niż jednego kryterium

```
SELECT *
FROM Customers
WHERE Country = 'Germany' OR Country = 'France'
```

Klienci z Niemiec i Monachium

Zadanie: Lista klientów z Niemiec, którą utworzyłeś zawiera zbyt dużo rekordów. Musisz ją zawęzić ograniczając się do miejscowości Monachium.

Podpowiedź: W tym przypadku miejscowość posiada oryginalną pisownię - 'München'

```
SELECT *
FROM Customers
WHERE Country = 'Germany' AND City = 'München'
```

Klienci równocześnie z Niemiec i z Monachium lub z Francji

Zadanie: Otrzymałeś zadanie połączenia wszystkich firm z Francji z firmami z Monachium.

Podpowiedź: Warunki można łączyć za pomocą nawiasów. Obowiązuje standardowa kolejność działań

```
SELECT *
FROM Customers
WHERE (Country = 'Germany' AND City = 'München') OR Country = 'France'
```

Klienci z Niemiec, Francji i Polski

Zadanie: Konieczne jest uzyskanie listy firm z Niemiec, Francji i Polski w pojedynczym zapytaniu

Podpowiedź: Do większej ilości kryteriów, gdzie nie wystarcza standardowe **AND** konieczne jest użycie operatora **IN**. Wewnątrz IN() może być umieszczona podkwerenda zwracająca dowolną liczbę kryteriów.

```
SELECT *
FROM Customers
WHERE Country IN('Germany', 'France', 'Poland')
```

Klienci z miasta Münster

Zadanie: Wyświetlenie listy firm z miejscowości Münster.

Podpowiedź: W sytuacji, gdy nie ma możliwości wpisania znaku z klawiatury możliwe jest zastosowanie znaków wieloznacznych. Tutaj "_" oznacza jeden dowolny znak. Dodatkowo niezbędne jest zastosowanie operatora **Like**.

```
SELECT *
FROM Customers
WHERE City like 'M_nster'
```

Firmy na literę K

Zadanie: Uzyskanie listy firm z nazwami rozpoczynającymi się na literę K.

Podpowiedź: W momencie, gdy ciąg znaków ma dowolną długość obok operatora **Like** należy użyć znaku **%**, który oznacza dowolny ciąg znaków.

```
SELECT *
FROM Customers
WHERE CompanyName like 'K%'
```

Firmy na litery od K do T

Zadanie: Pozyskanie listy firm posiadających nazwę na litery od K do T.

Podpowiedź: Połączenie operatora Like z listą [] pozwala także na inne kombinacje parametru np. [K,T,W-Z]

```
SELECT *
FROM Customers
WHERE CompanyName like '[K-T]%'
```

Wszyscy klienci spoza Niemiec

Zadanie: Wygeneruj listę firm, z której wykluczysz firmy z Niemiec.

Podpowiedź: Znak <> oznacza różny od, alternatywnie możesz użyć != (nierówny). Oba pozwalają na wyłączenie warunku z listy wyników

```
SELECT *
FROM Customers
WHERE Country <> 'Germany'
```

Wszyscy klienci z wyłączeniem Niemiec, Francji i Polski

Zadanie: Stwórz listę firm, która nie będzie zawierała znajdujących się na obszarze Niemiec, Francji i Polski.

Podpowiedź: Operator **NOT** pozwala wyłączyć z wyniku także listę warunków.

```
SELECT *
FROM Customers
WHERE Country NOT IN('Germany','France','Poland')
```

Firmy bez faksu

Zadanie: Pozyskaj wykaz firm, które nie posiadają w swoich danych kontaktowych numeru faksu.

Podpowiedź: Pusty ciąg znaków ' ', tak jak 0 nie są wartością **NULL**. Wartość nieokreślona **NULL**, w bazach danych także po połączeniu z dowolną inną wartością zwraca wartość **NULL**. Dodatkowo kryterium zawierające **NULL** musi posiadać zapis **IS NULL**, a nie **= NULL**.

```
SELECT *
FROM Customers
WHERE Fax IS NULL
```

Firmy z faksem

Zadanie: Podobnie jak poprzednim wypadku, w kryterium wystąpi także fax, tym razem fax musi istnieć.

Podpowiedź: Wartość niepustą uzyskujemy poprzez zaprzeczenie **IS NOT NULL**

```
SELECT *
FROM Customers
WHERE Fax IS NOT NULL
```

Zamówienia o wartości ponad 1000

Zadanie: Sprawdź, które zamówienia posiadają wartość ponad 1000

Podpowiedź: Operatory matematyczne <, >, <>, <=, >= działają zgodnie z regułami matematycznymi

```
SELECT *
FROM AnlizaSprzedaży
WHERE Wartość > 1000
```

Zamówienia o wartości pomiędzy 1000 a 2000

Zadanie: Pobierz szczegółowe informacje o zamówieniach z przedziału od 1000 do 2000.

Podpowiedź: Operator **BETWEEN AND** pozwala na podanie zakresu, dodatkowo wartości brzegowe są także brane pod uwagę. Dodatkowo AND powiązany z BETWEEN ma wyższy priorytet od „zwykłego” AND.

```
SELECT *
FROM AnlizaSprzedaży
WHERE Wartość Between 1000 AND 2000
```

10 największych zamówień

Zadanie: Pobierz listę 10 największych zamówień z Analizy sprzedaży.

Podpowiedź: Klauzula **TOP** pozwala na ograniczenie liczby wyników kwerendy. Sama kwerenda musi mieć zachowany porządek sortowania

```
SELECT top 10 *
FROM AnlizaSprzedaży
ORDER BY Wartość
```

10% najgorszych zamówień

Zadanie: Wskaż na liście Analizy sprzedaży końcowe 10% najgorszych wyników.

Podpowiedź: Klauzula **TOP PERCENT** odcina procentowo część listy

```
SELECT top 10 PERCENT *
FROM AnlizaSprzedaży
ORDER BY Wartość DESC
```

Wartość brutto sprzedaży

Zadanie: W oparciu o Analizę sprzedaży wylicz wartość brutto dla poszczególnych pozycji sprzedażowych.

Podpowiedź: Tworzone własne pola obliczeniowe, powinny mieć nazwane nagłówki kolumn tzw. **aliasy**. Dodatkowo zwróć uwagę na poprawny zapis wartości dziesiętnej w języku SQL – kropka jest separatorem dziesiętnym. Symbol * w tym wypadku oznacza mnożenie.

```
SELECT OrderID, Wartość * 1.23 AS WartośćBrutto
FROM AnlizaSprzedaży
```

Poprzez ALLIAS można tabeli albo kolumnie nadać inną nazwę, która będzie jej synonimem. Jest to szczególnie przydatne przy skracaniu nazw tabel lub kolumn. W ten sposób zapytanie może stać się dla nas bardziej czytelne.

Liczba miesięcy od ostatniego zamówienia

Zadanie: Policz ile miesięcy minęło od każdego ze złożonych zamówień.

Podpowiedź: Funkcja **DATEDIFF()** zwraca liczbę interwałów czasowych pomiędzy dwiema datami. Funkcja **GETDATE()** zwraca bieżący czas, **month** to nazwa interwału miesięcznego (występują też inne: day, year itd.)

```
SELECT OrderID, DATEDIFF(month, GETDATE(), OrderDate) AS OstatnieZamówienie
FROM AnlizaSprzedaży
```

Czas dostawy

Zadanie: Policz ile dni trwała realizacja poszczególnych zamówień.

Podpowiedź: Interwał czasowy **day** dla jest pomocny dla obliczenia różnicy w dniach

```
SELECT OrderID, DATEDIFF(Day, OrderDate, ShippedDate) AS CzasDostawy
FROM AnlizaSprzedaży
```

Ostatni dzień miesiąca

Zadanie: Określ dla każdego ze złożonych zamówień datę – ostatni dzień miesiąca, potrzebny do gwarancji.

Podpowiedź: Funkcja DATEADD() – dodaje wskazany interwał czasowy do daty. Funkcja DATEFROMPARTS służy do złożenia daty z elementów roku, miesiąca i dnia.

```
SELECT OrderID,
       DATEADD(DAY, -1, DATEFROMPARTS(year(OrderDate), month(OrderDate), 1)) AS
Gwarancja
FROM AnlizaSprzedaży
```

Lista krajów - unikat

Zadanie: Wyświetl listę krajów, które znajdują się w tabeli Customers

Podpowiedź: Do wyświetlenia listy unikatowych rekordów z bazy danych możesz wykorzystać **DISTINCT**.

```
SELECT DISTINCT Country
FROM Customers
```

Lista krajów – zliczanie

Zadanie: Podobnie jak w poprzednim zadaniu wyświetl listę krajów, ale użyj do tego funkcji agregujących.

Podpowiedź: Opis i szczegółowe zastosowanie funkcji agregujących znajduje się w dalszej części skryptu.

```
SELECT Country
FROM Customers
GROUP BY Country
```

Wielkość sprzedaży

Zadanie: Korzystając z funkcji warunkowych określ, czy zamówienie jest duże, czy małe.

Podpowiedź: Funkcja warunkowa IIF jest konstrukcją odpowiadającą funkcji **jeżeli()** z MS Excel.

```
SELECT OrderID, IIF(Wartość > 1000, 'Duża', 'Mała')
FROM AnlizaSprzedaży
```

Progi sprzedaży

Zadanie: Podobnie jak w poprzednim przykładzie, określ wielkość sprzedaży. Tym razem ustal trzy przedziały.

Podpowiedź: Alternatywną formą tworzenia warunku (dla starszych wersji SQL jedyną dostępną) jest skorzystanie z bloku **CASE WHEN ... THEN ... ELSE ... END**

```
SELECT OrderID,
       CASE
           WHEN Wartość > 1000 THEN 'Duża'
           WHEN Wartość > 500  THEN 'Średnia'
           ELSE 'Mała'
       END
```

2.2. Zapisywanie kwerendy - widok

Kwerendy, które były tworzone podczas ćwiczenia były uruchamiane i ich wynik był dostępny na ekranie w formie nietrwałej. Serwer SQL pozwala na stworzenie obiektu **View** – Widok/perspektywa, który później jest dostępny w formie generowanego na żądanie wyniku widocznego i działającego jak tabela.

W bieżącym przykładzie zostanie stworzony (przy założeniu posiadania stosownych uprawnień) Widok zawierający listę firm na literę A. Poleceniem tworzącym obiekt jest **CREATE**.

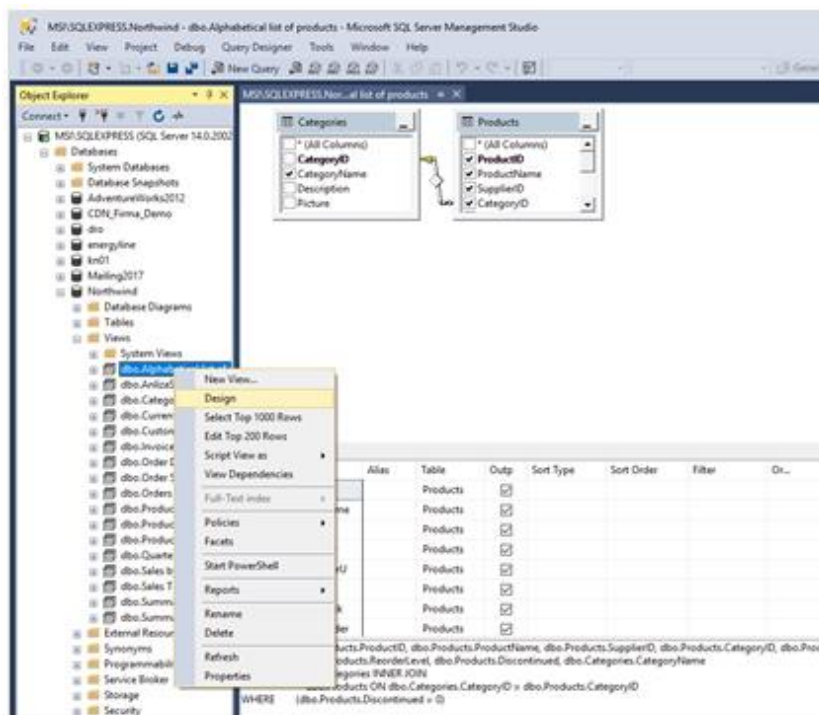
Uwaga: Tworzenie obiektów wymaga dodatkowych uprawnień dla zalogowanego użytkownika, bez których tworzenie i modyfikacja obiektów nie jest możliwa.

```
CREATE VIEW FirmyNaLiteręA
AS
SELECT *
FROM Customers
WHERE CompanyName like 'A%'
```

Alternatywnie dostępna jest opcja tworzenia obiektu po kliknięciu prawym klawiszem myszy na **Views** i **New view** w eksploratorze obiektów.

Niezależnie od metody tworzenia obiektu, zwykle istnieje możliwość utworzenia tak stworzonej kwerendy w trybie projektu graficznego.

Istnieje część kwerend, których nie da się w ten sposób edytować. Zwykle są to kwerendy zawierające komentarze lub używające UNION itp.



2.3. Modyfikacja zapisanego widoku

Stworzony uprzednio widok możemy edytować z zastosowaniem polecenia **ALTER**. Alternatywnie dostępna jest edycja obiektu po kliknięciu prawym klawiszem myszy na obiekcie i polecenie **Design**. Tryb konstruktora zapytań nie jest dostępny dla każdego typu kwerendy.

```
ALTER VIEW FirmyNaLiteręA
AS
SELECT *
FROM Customers
WHERE CompanyName like 'A%'
```

2.4. Usuwanie istniejącego widoku

Obiekt View jak każdy inny obiekt można usunąć z serwera za pomocą polecenia **DROP**. Alternatywnie dostępna jest edycja obiektu po kliknięciu prawym klawiszem myszy na obiekcie i polecenie **Delete**.

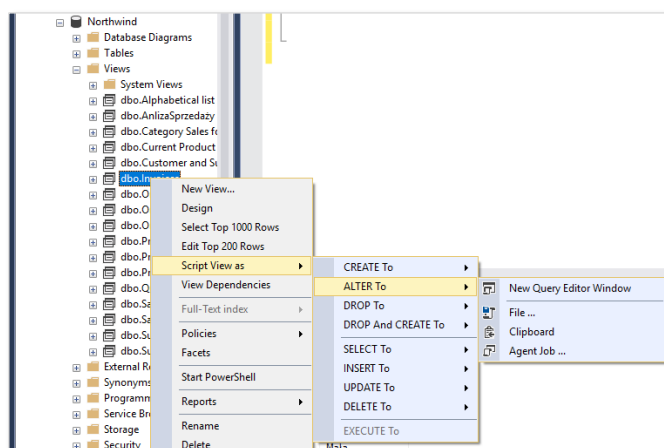
```
DROP VIEW FirmyNaLiteręA
```

2.5. Generowanie kodu SQL

Microsoft Management Studio wyposażony jest w narzędzia niezbędne do szybkiego generowania kodu SQL. W celu uzyskania gotowego do pracy kodu. W tym celu wystarczy zaznaczyć prawym klawiszem myszy dowolny obiekt w bazie danych i korzystając z opcji Script View/Table as wybrać odpowiedni szablon.

Dostępne są polecenia:

- CREATE – tworzenie obiektu
- ALTER – modyfikacja obiektu
- DROP – usuwanie obiektu
- DROP and CREATE – usuń i stwórz obiekt
- SELECT – skrypt wybierania rekordów
- INSERT – skrypt dodawania rekordów
- UPDATE – aktualizacja rekordów
- DELETE – usuwanie rekordów



Skrypt może być wyświetlony w nowym oknie, zapisany do pliku, schowka lub przekazany agentowi zadań SQL.

2.6. Operacje na połączonych tabelach

Kwerendy oparte o wiele tabel. Do tego momentu wszystkie kwerendy były budowane o jedną tabelę. Kolejne ćwiczenia pozwolą na połączeniu wielu tabel w jednym zapytaniu.

Typy złączeń		
INNER JOIN	naturalne	wyświetlone będą tylko te dane z obu tabel, w których wartości są równe
LEFT OUTER JOIN	lewe	wyświetlone będą wszystkie wiersze z lewej tabeli i odpowiadające im zgodne wiersze z prawej
RIGHT OUTER JOIN	prawe	wyświetlone będą wszystkie wiersze z prawej tabeli i odpowiadające im zgodne wiersze z lewej
FULL OUTER JOIN	pełne	wszystkie wiersze z obu tabel
CROSS JOIN	krzyżowe - iloczyn kartezyjański	dane z jednej tabeli pomnożone przez dane z drugiej tabeli

Złączenie INNER JOIN

Postanowiłeś do listy zamówień, która zawiera zamiast nazwy kontrahenta jedynie jego identyfikator liczbowy, dołączyć jego nazwę zawartą w tabeli Customers. Gdzie elementem wspólnym w obu tabelach jest pole CustomerID. Obie tabele połączone są za pomocą relacji.

```
SELECT Orders.OrderID, Orders.CustomerID, Customers.CompanyName, Orders.OrderDate
FROM
  Orders INNER JOIN Customers ON
    Orders.CustomerID = Customers.CustomerID
```

Funkcjonalność rozwiązania jest zbliżona do funkcji wyszukaj pionowo znanej z Excela.

Złączenie INNER JOIN

Po sprawdzeniu działania kwerendy używającej INNER JOIN postanowiłeś wykorzystać ją w praktyce. Posiadasz dwie tabele **Customers** i **CustomersOLD**, które zawierają dwie bazy klientów, różniące się od siebie. Zastosowanie poniższej kwerendy spowoduje wyświetlenie wspólnej części obu tabel.

```
SELECT Customers.CustomerID, Customers.CompanyName, CustomersOLD.CompanyName AS Old
FROM
  Customers INNER JOIN CustomersOLD
ON
  Customers.CustomerID = CustomersOLD.CustomerID
```

Złączenie RIGHT OUTER JOIN

Kolejna kwerenda pozwoli wyświetlić całą zawartość tabeli CustomersOLD i odpowiadających jej wpisów w tabeli Customers.

```
SELECT Customers.CustomerID, Customers.CompanyName, CustomersOLD.CompanyName AS Old
FROM
  Customers RIGHT OUTER JOIN CustomersOLD
ON
  Customers.CustomerID = CustomersOLD.CustomerID
```

Złączenie LEFT OUTER JOIN

Działaniem w podobny sposób wyświetlającym informacje, ale tym razem to tabela Customers będzie pokazana jako cała jest złączenie LEFT OUTER JOIN.

```
SELECT Customers.CustomerID, Customers.CompanyName, CustomersOLD.CompanyName AS Old
FROM
Customers LEFT OUTER JOIN CustomersOLD
ON
Customers.CustomerID = CustomersOLD.CustomerID
```

Złączenie FULL OUTER JOIN

Kolejnym krokiem jest zastosowanie kwerendy, która pokaże zawartość obu tabel razem.

```
SELECT Customers.CustomerID, Customers.CompanyName, CustomersOLD.CompanyName AS Old
FROM
Customers FULL OUTER JOIN CustomersOLD
ON
Customers.CustomerID = CustomersOLD.CustomerID
```

Złączenie CROSS JOIN

Ostatnim typem złączenia jest iloczyn kartezjański, czyli przemnożenie każdego rekordu w tabeli Customers przez rekord w tabeli CustomersOLD.

```
SELECT Customers.CustomerID, Customers.CompanyName, CustomersOLD.CompanyName AS Old
FROM
Customers CROSS JOIN CustomersOLD
```

2.7. Łączenie wyników zapytania

Poza łączeniem tabel możliwe jest także łączenie wyników z kilku tabel w jedną dłuższą tabelę. Sprawdzisz także możliwość wyświetlania wartości unikatowych – bez powtórzeń.

Unikatowe wyniki DISTINCT

Otrzymałeś zadanie zebrania listy krajów z jakimi prowadzisz wymianę handlową.

```
SELECT DISTINCT Country
FROM Customers
```

Łączenie bez powtórzeń UNION

Postanowiłeś scalić zawartość tabel Customers i CustomersOLD, aby otrzymać unikalną listę klientów zebraną z obu tabel. Zdublikowane rekordy zostaną pominięte.

```
SELECT *
FROM Customers
UNION
SELECT *
FROM CustomersOLD
```

Łączenie z powtórzeniami UNION ALL

Postanowiłeś sprawdzić jak wygląda lista firm nawet jeśli się powtarzają

```
SELECT *
FROM Customers
UNION ALL
SELECT *
FROM CustomersOLD
```


Część wspólna INTERSECT

Chciałbyś zobaczyć wspólną część obu tabel Customers

```
SELECT *  
    FROM Customers  
INTERSECT  
SELECT *  
    FROM CustomersOLD
```

Część niepowtarzalna w innej tabeli EXCEPT

Postanowiłeś zobaczyć te elementy, które są unikalne w obu tabelach

```
SELECT *  
    FROM Customers  
EXCEPT  
SELECT *  
    FROM CustomersOLD
```

W bazie danych Oracle SQL polecenie EXCEPT występuje pod nazwą MINUS.

2.8. Kwerendy zagnieżdżone

Podczas tworzenie kwerendy możesz jako kryterium zastosować wynik innej kwerendy umieszczony jako warunek dla WHERE. Jedynym warunkiem jest to, żeby podkwerenda zwracała pojedynczą kolumnę z wynikiem. Sam wynik nie musi zawierać klauzuli **DISTINCT**.

Operator IN

Sprawdź jacy klienci w tabeli **Customers** znajdują się także w tabeli **CustomersOLD**. Pomocny w tym będzie operator **IN** pobierający parametry z zagnieżdżonej kwerendy.

```
SELECT CustomerID, CompanyName FROM Customers  
    WHERE CustomerID  
        IN (SELECT CustomerID  
            FROM CustomersOLD)
```

Operator NOT IN

W sytuacji konieczności sprawdzenia, którzy klienci występują wyłącznie w tabeli **Customers** możesz użyć zaprzeczenia **IN**, czyli **NOT IN**.

```
SELECT CustomerID, CompanyName FROM Customers  
    WHERE CustomerID  
        NOT IN (SELECT CustomerID  
                FROM CustomersOLD)
```

Wyrażenia tabelaryczne CTE - WITH

Możesz także wykorzystać **CTE** i **WITH**, w celu optymalizacji zapytania.

```
WITH lista  
AS  
(SELECT CustomerID FROM CustomersOLD)  
SELECT CustomerID, CompanyName  
    FROM Customers  
    WHERE CustomerID  
        IN (SELECT CustomerID  
            FROM lista)
```

Podkwerenda z JOIN

Wynik podkwerendy możesz także wykorzystać podczas łączenia tabel z użyciem złączeń. Np. **INNER JOIN**.

```
SELECT Customers.CustomerID, X.CustomerID AS Y
  FROM   Customers INNER JOIN
        (SELECT CustomerID
         FROM   CustomersOLD) AS X ON Customers.CustomerID = X.CustomerID
```

Zaawansowane łączenie tabel

Łącząc tabele w wyniku zapytania możliwe jest także użycie innych znaków niż „=“.

```
SELECT Customers.CustomerID, CustomersOLD.CustomerID AS Y
  FROM Customers INNER JOIN CustomersOLD
        ON Customers.CustomerID <> CustomersOLD.CustomerID
```

```
SELECT Customers.CustomerID, CustomersOLD.CustomerID AS Y
  FROM Customers INNER JOIN CustomersOLD
        ON Customers.CustomerID <= CustomersOLD.CustomerID
```

```
SELECT Customers.CustomerID, CustomersOLD.CustomerID AS Y
  FROM Customers INNER JOIN CustomersOLD
        ON Customers.CustomerID >= CustomersOLD.CustomerID
```

3. Funkcje

Funkcje i wyrażenia w języku SQL są dostępne w zależności od typu i wersji silnika bazy danych.

Funkcje daty i czasu - przykłady

Chciałbyś ustalić termin gwarancji dla wysłanych produktów. Wynosi on 18 miesięcy od momentu wysyłki towaru.

```
SELECT OrderID,  
       DATEADD(month, 18, OrderDate) AS KolejnyKontakt  
FROM AnlizaSprzedaży
```

Z logistycznego punktu widzenia interesująca może się wydać informacja o czasie jaki został poświęcony na dostawę towaru. Informacja ta musi być podana w dniach.

```
SELECT OrderID,  
       DATEDIFF(day, OrderDate, ShippedDate) AS DługośćDostawy  
FROM AnlizaSprzedaży
```

Ostatnią informacją związaną z datami jest wyświetlenie roku zamówienia.

```
SELECT OrderID, OrderDate,  
       DATENAME(YEAR, OrderDate) AS DługośćDostawy  
FROM AnlizaSprzedaży
```

Funkcje matematyczne - przykłady

Operacje matematyczne jakie można wykonywać w kwerendzie są zbliżone do tych jakie oferuje MS Excel. Postanowiłeś policzyć wartość VAT na podstawie kolumny wartość wynosi on 7%.

```
SELECT *,  
       Round(Wartość * 0.07,2) AS WartośćVAT  
FROM AnlizaSprzedaży
```

Funkcje tekstowe - przykłady

Funkcje tekstowe umożliwiają łączenie, dzielenie i obsługę ciągów tekstowych. Postanowiłeś połączyć nazwę klienta z regionem, aby użyć tak połączonych ciągów w planowanej korespondencji seryjnej.

```
SELECT CompanyName + ', ' + Region AS Lokalizacja  
FROM Customers
```

Niestety takie działanie mimo, że dopuszczalne wyświetliło dużą ilość pól oznaczonych jako **NULL**. Dzieje się tak dlatego, że **NULL** jako wartość nieokreślona po połączeniu z każdą inną daje kolejną wartość nieokreśloną. Musisz skorygować ten zapis.

```
SELECT CompanyName + ', ' + ISNULL(Region,'') AS Lokalizacja  
FROM Customers
```

Inną metodą jest zastosowanie **CONCAT**, które zabezpiecza ciągi podczas łączenia.

```
SELECT CONCAT(CompanyName, ', ', Region) AS Lokalizacja  
FROM Customers
```

Chciałbyś uzyskać listę imion swoich kontaktów. Użyjesz do tego funkcji **LEFT** zwracającej określoną liczbę znaków od lewej strony oraz funkcję **CHARINDEX**, która podaje pozycję jednego ciągu znaków w drugim ciągu. Do tego dodasz **CustomerID** wpisany małymi literami i miasto wielkimi.

```
SELECT  
LOWER(CustomerID) AS CustomerID,  
LEFT(ContactName,CHARINDEX(' ',ContactName)-1) AS Imię,  
UPPER(City) AS City  
FROM Customers
```

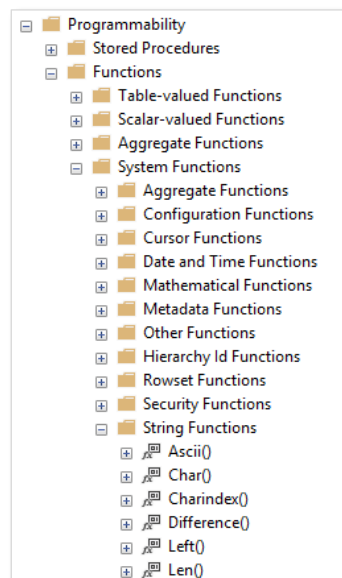
Dział callcenter potrzebuje wykazu numerów telefonów dla automatycznego systemu komunikacyjnego. Dane muszą być pozbawione myślników, które są w nich użyte

```
SELECT CustomerID,
       REPLACE(Phone, '-', '')
FROM   Customers
```

Pozostałe funkcje SQL

Podczas pracy z serwerem SQL dostępnych jest dużo więcej funkcji niż przedstawione w powyższych przykładach. Istnieje możliwość podejrzenia ich wykazu z zastosowaniem Object Explorera. Grupa Programmability/Functions/System Functions. Są one zgrupowane tematycznie, podobnie jak w MS Excel.

- Operacje na łańcuchach znaków LEN, LEFT, RIGHT, MID, REPLACE, TRIM, SUBSTRING, UPPER, LOWER
- Łączenie i dzielenie ciągów tekstowych CONCAT, TRIM
- Funkcje matematyczne ROUND, ABS, FLOOR, SQUARE
- Funkcje czasu i daty DATE, DATEADD, DATEDIFF, MONTH, YEAR, DAY
- Konwersja i rzutowanie typów: CAST(), CONVERT().



3.1. Funkcje tekstowe - zastosowania

Funkcje operujące na łańcuchach znaków.

ASCII

Zwraca kod ASCII dla wskazanego znaku

```
PRINT ASCII(11)
```

CHAR

Zwraca znak dla podanego kodu ASCII

```
PRINT CHAR(33)
```

CHARINDEX

Zwraca pozycję ciągu znaków w innym ciągu znaków

```
PRINT CHARINDEX('z', 'baza')
```

CONCAT

Łączy dwa lub więcej ciągów tekstowych

```
PRINT CONCAT('S', 'Q', 'L')
```

Concat z +

Łączy dwa lub więcej ciągów tekstowych operatorem +

```
PRINT 'S'+'Q'+'L'
```

CONCAT_WS

Łączy dwa lub więcej ciągów tekstowych z separatorem

```
PRINT CONCAT_WS('$', 'S', 'Q', 'L')
```

DATALENGTH

Zwraca liczbę bajtów użytych do przechowania wyrażenia

```
PRINT DATALENGTH('SQL')
```

DIFFERENCE

Zwraca różnicę pomiędzy dwoma ciągami znaków (liczbę znaków wspólnych)

```
PRINT DIFFERENCE('SQL', 'MSSQL')
```

FORMAT

Formatuje wynik do wskazanego formatu

```
PRINT FORMAT(11.99, '0.0000')
```

LEFT

Zwraca znaki z lewej strony ciągu

```
PRINT LEFT('MSSQL', 2)
```

LEN

Zwraca liczbę znaków w wyrażeniu

```
PRINT LEN('MSSQL')
```

LOWER

Przekształca ciąg znaków na litery małe

```
PRINT LOWER('MSSQL')
```

LTRIM

Usuwa spacje wiodące z lewej strony ciągu

```
PRINT LTRIM('    MSSQL')
```

NCHAR

Zwraca znak Unicode w oparciu o podany nr znaku

```
PRINT NCHAR(122)
```

PATINDEX

Zwraca lokalizację wzorca w łańcuchu znaków

```
PRINT PATINDEX('%Q%', 'MSSQL')
```

QUOTENAME

Zwraca łańcuch znaków unicode z dodanymi kwalifikatorami tekstu

```
PRINT QUOTENAME('MSSQL')
```

REPLACE

Zamienia jeden ciąg znaków na inny

```
PRINT REPLACE('MSSQL', 'L', 'F')
```

REPLICATE

Zwraca wskazany ciąg znaków określoną ilość razy

```
PRINT REPLICATE('MSSQL', 2)
```

REVERSE

Zwraca ciąg znaków w odwróconej kolejności

```
PRINT REVERSE('MSSQL')
```

RIGHT

Zwraca znaki od prawej strony ciągu tekstowego

```
PRINT RIGHT('MSSQL', 2)
```

RTRIM

Usuwa spacje od prawej strony ciągu tekstowego

```
PRINT RTRIM('MSSQL ')
```

SOUNDEX

Zwraca kod porównujący dla ciągu tekstowego

```
PRINT SOUNDEX('MSSQL')
```

SPACE

Zwraca określoną liczbę spacji

```
PRINT SPACE
```

STR

Zwraca liczbę jako łańcuch znaków

```
PRINT STR(123)
```

STUFF

Usuwa wskazaną liczbę znaków z ciągu tekstowego od wskazanej pozycji, następnie zamienia ją na inny ciąg

```
PRINT STUFF('MSSQL', 3, 2, 'X')
```

SUBSTRING

Zwraca określoną liczbę znaków z ciągu tekstowego od podanej liczby początkowej

```
PRINT SUBSTRING('MSSQL', 3, 2)
```

TRANSLATE

Zamienia znaki w ciągu tekstowym na wskazany

```
PRINT TRANSLATE('MSSQL', 'M', 'n')
```

TRIM

Usuwa spacje na końcu i początku ciągu tekstowego

```
PRINT TRIM(' MSSQL ')
```

UNICODE

Zwraca wartość dla pierwszego znaku w ciągu tekstowym

```
PRINT UNICODE('MSSQL')
```

UPPER

Konwertuje ciąg tekstowy na litery wielkie

```
PRINT UPPER('mssql')
```

3.2. Funkcje matematyczne - zastosowania

Standardowe funkcje matematyczne, przydatne podczas obliczeń.

ABS

Zwraca wartość bezwzględną liczby

```
PRINT ABS(-12)
```

ACOS

Funkcja Acos zwraca arcus cosinus argumentu, czyli odwrotność jego cosinusa. Arcus cosinus to kąt, którego cosinus jest argumentem. Zwrócony kąt jest określony w radianach i ma wartość z zakresu od 0 do π .

```
PRINT ACOS(-1)
```

ASIN

Funkcja Asin zwraca arcus sinus argumentu, czyli odwrotność jego sinusa. Arcus sinus to kąt, którego sinus jest argumentem. Zwrócony kąt jest określony w radianach i należy do zakresu od $-\pi/2$ do $\pi/2$.

```
PRINT ASIN(1)
```

ATAN

Funkcja Atan zwraca arcus tangens argumentu, czyli odwrotność jego tangensa. Arcus tangens to kąt, którego tangens jest argumentem. Zwrócony kąt jest określony w radianach i należy do zakresu od $-\pi/2$ do $\pi/2$.

```
PRINT ATAN(1)
```

ATN2

Funkcja Atan2 zwraca arcus tangens, czyli odwrotność tangensa, argumentu określonego w postaci współrzędnych x i y. Arcus tangens to kąt między osią x a prostą, która zawiera początek układu współrzędnych (0, 0) i punkt o współrzędnych (x, y). Kąt jest określony w radianach i należy do zakresu od $-\pi$ do π , z wyłączeniem wartości $-\pi$. Wynik dodatni oznacza kąt mierzony od osi x w kierunku przeciwnym do ruchu wskazówek zegara, a wynik ujemny oznacza kąt mierzony w kierunku zgodnym z ruchem wskazówek zegara. $\text{Atan2}(a, b)$ jest równe $\text{Atan}(b/a)$, z tą różnicą, że a może być równe 0 w przypadku funkcji Atan2.

```
PRINT ATN2(1, 2)
```

AVG

Zwraca wartość średnią (funkcja agregująca)

Materiały szkoleniowe

```
PRINT AVG()
```

CEILING

Zwraca najmniejszą liczbę całkowitą dla wartości gdzie liczba jest \geq od wartości

```
PRINT CEILING(88.222)
```

COUNT

Zwraca ilość elementów (funkcja agregująca)

```
PRINT COUNT(*)
```

COS

Funkcja Cos zwraca cosinus argumentu — kąt określony w radianach.

```
PRINT COS(23)
```

COT

Funkcja Cot zwraca cotangens argumentu — kąt określony w radianach.

```
PRINT COT(2)
```

DEGREES

Konwertuje radiany na stopnie

```
PRINT DEGREES(4)
```

EXP

zwraca wartość wykładniczą. Dla EXP(1) nam liczbę Eulera $\sim 2,71828182845905$

```
PRINT EXP(1)
```

FLOOR

Zwraca największą liczbę całkowitą dla wartości gdzie liczba \leq wartości

```
PRINT FLOOR(30.999)
```

LOG

Zwraca logarytm funkcja wykładnicza.

```
PRINT LOG
```

LOG10

Zwraca logarytm funkcja wykładnicza.

```
PRINT LOG10
```

MAX

Zwraca wartość maksymalną (funkcja agregująca)

```
PRINT MAX()
```

MIN

Zwraca wartość minimalną (funkcja agregująca)

Materiały szkoleniowe

```
PRINT MIN()
```

PI

Zwraca wartość liczby PI

```
PRINT PI()
```

POWER

Zwraca wartość liczby a do potęgi b

```
PRINT POWER(4,4)
```

RADIANS

Konwertuje kąt do radianów

```
PRINT RADIANS(200)
```

RAND

Zwraca a wartość losową

```
PRINT RAND()
```

ROUND

Zaokrągla liczbę do wskazanej liczby miejsc

```
PRINT ROUND(2.111,1)
```

SIGN

Zwraca znak liczby (dodatnia/ujemna)

```
PRINT SIGN(-222)
```

SIN

Zwraca sinus kąta

```
PRINT SIN(180)
```

SQRT

Zwraca pierwiastek kwadratowy z liczby

```
PRINT SQRT(100)
```

SQUARE

Zwraca liczbę podniesioną do kwadratu

```
PRINT SQUARE(3)
```

SUM

Zwraca wartość sumę (funkcja agregująca)

```
PRINT SUM()
```

TAN

Zwraca tangens kąta

```
PRINT TAN(30)
```

3.3. Funkcje daty i czasu - zastosowania

CURRENT_TIMESTAMP

Zwraca bieżącą datę i godzinę

```
PRINT CURRENT_TIMESTAMP
```

DATEADD

Dodaje wskazany interwał czasowy do daty

```
PRINT DATEADD(month, 4, '2018-05-06')
```

Dostępne interwały: day, month, year...

DATEDIFF

Zwraca ilość wskazanych interwałów czasowych między dwoma datami

```
PRINT DATEDIFF(month, '2018-05-06', '2019-08-06')
```

Dostępne interwały: day, month, year...

DATEFROMPARTS

Zwraca datę złożoną ze składników (rok, miesiąc, dzień)

```
PRINT DATEFROMPARTS(2018, 5, 5)
```

Funkcja zwracająca ostatni dzień miesiąca – w tym przypadku 30.04.2018

```
PRINT DATEFROMPARTS(2018, 5, 0)
```

DATENAME

Zwraca wskazaną część daty jako tekst

```
PRINT DATENAME(day, '2019-01-21')
```

DATEPART

Zwraca wskazaną część daty jako liczbę

```
PRINT DATEPART(day, '2019-01-21')
```

DAY

Zwraca dzień miesiąca z daty.

```
PRINT DAY('2018-05-06')
```

GETDATE

Zwraca bieżącą datę i czas systemowe

```
PRINT GETDATE()
```

GETUTCDATE

Zwraca bieżącą datę i czas wg UTC

```
PRINT GETUTCDATE()
```

ISDATE

Zwraca 1 jeśli wyrażenie jest datą, inaczej 0

```
PRINT ISDATE ('2018-05-06')
```

MONTH

Zwraca numer miesiąca z daty.

```
PRINT MONTH ('2018-05-06')
```

SYSDATETIME

Zwraca datę i godzinę wg serwera SQL.

```
PRINT SYSDATETIME()
```

YEAR

Zwraca rok z daty.

```
PRINT YEAR ('2018-05-06')
```

3.4. Funkcje konwersji - zastosowania

Pomagają przekształcić jeden format danych w inny – przegląd funkcji:

CAST

Rzuca jeden typ danych na inny, w przypadku błędu przerywa skrypt.

```
PRINT CAST('2018-03-01' AS smalldatetime)
PRINT CAST('2018-03-0A' AS smalldatetime)
```

TRY_CAST

Rzuca jeden typ danych na inny, w przypadku zwraca null.

```
PRINT CAST('2018-03-01' AS smalldatetime)
PRINT TRY_CAST('2018-03-0A' AS smalldatetime)
```

CONVERT

Konwertuje jeden typ danych na inny, w przypadku błędu przerywa skrypt.

```
PRINT CONVERT(smalldatetime, '2018-03-01')
PRINT CONVERT(smalldatetime, '2018-03-0A')
```

TRY_CONVERT

Konwertuje jeden typ danych na inny, w przypadku zwraca null.

```
PRINT TRY_CONVERT(smalldatetime, '2018-03-01')
PRINT TRY_CONVERT(smalldatetime, '2018-03-0A')
```

Podczas pracy z danymi umieszczonymi na serwerze konieczne jest modyfikowanie i konwersja danych. Dobrym przykładem będzie tu tabela Orders, gdzie czas jest podany z dokładnością od tysięcznych części sekundy. Taki format utrudnia wykorzystanie takich danych w Excelu. Dla poprawnego wyświetlenia daty użyj formatu DATE, który pozwoli na odcięcie części godzinowej od wartości daty.

```
SELECT OrderID, CONVERT(DATE, OrderDate) AS DataZamówienia
FROM Orders
```

W sytuacji, gdy pole czasu miałoby część godzinową możesz zastosować format docelowy SMALLDATETIME

```
SELECT OrderID, CONVERT(smalldatetime, OrderDate) AS DataZamówienia
FROM Orders
```

Alternatywnie zamiast funkcji konwersji możesz użyć funkcji rzutowania **CAST**.

```
SELECT OrderID, CAST(OrderDate AS Date) AS DataZamówienia
FROM Orders
```

Od wersji **SQL 2012** dodano funkcję **TRY_CAST** i **TRY_CONVERT**, które działają tak jak starsze ich wersje, ale w przypadku błędu konwersji zwracają **NULL**

```
SELECT OrderID, TRY_CONVERT(DATE, OrderDate) AS DataZamówienia
FROM Orders
```

Przeprowadź rzutowanie **TRY_CAST** daty na ciąg tekstowy

```
SELECT OrderID, TRY_CAST(OrderDate AS NVARCHAR(50)) AS DataZamówienia
FROM Orders
```

Błąd który się pojawił wynika z ustawień regionalnych, aby konwersja wyszła poprawnie użyj poniższego kodu.

```
SELECT OrderID, TRY_CAST(CONVERT(DATE, OrderDate) AS NVARCHAR(50)) AS
DataZamówienia
FROM Orders
```

Oprócz konwersji czasu może pojawić się potrzeba konwersji liczb, spróbuj przekonwertować wartość frachtu do formatu liczby całkowitej **INT**.

```
Select OrderID, convert(INT, Freight) AS Fracht
FROM Orders
```

Spróbuj przekształcić dane w kwerendzie [Orders Qry] w bardziej czytelną formę – przykład poniżej.

```
SELECT OrderID
      ,CustomerID
      ,EmployeeID
      ,CONVERT(date, OrderDate) AS OrderDate
      ,CONVERT(date, RequiredDate) AS RequiredDate
      ,CONVERT(date, ShippedDate) AS ShippedDate
      ,ShipVia
      ,Freight
      ,ShipName
      ,ShipAddress
      ,ShipCity
      ,ISNULL(ShipRegion, '') AS ShipRegion
      ,ShipPostalCode
      ,ShipCountry
      ,CompanyName
      ,Address
      ,City
      ,ISNULL(Region, '') AS Region
      ,PostalCode
      ,Country
FROM [Orders Qry]
```

Ustawienia regionalne w konwersji

Konwersja danych wg wybranych ustawień regionalnych

```
PRINT CONVERT(NVARCHAR(50), GETDATE(), 121) -- 12 112
```

Parsowanie PARSE / TRY_PARSE

Zamiana wartości „wyglądającej jak” liczba lub data w liczbę lub datę możliwa jest z użyciem zwykłej konwersji, ale zależna jest od bieżących ustawień regionalnych użytkownika.

```
PRINT CONVERT(date, '12-sep-18')
PRINT CONVERT(date, '12-sie-18')
```

Zamiana wartości „wyglądającej jak” liczba lub data w liczbę lub datę

```
PRINT PARSE('12-sie-18' AS date USING 'pl-pl')
PRINT TRY_PARSE('02-sie-18' as date USING 'PL-pl')
```

3.5. Funkcje specjalne - zastosowania

Funkcje pomocnicze dla bazy danych, zwracają zwykle informacje o systemie, zalogowanym użytkowniku, a także wszelkiego rodzaju funkcje sprawdzające. W poniższych przykładach przedstawione zostało ich działanie w oparciu o polecenie PRINT.

COALESCE

Zwraca pierwszą niepustą wartość z listy

```
PRINT COALESCE (1, 2)
```

CURRENT_USER

Zwraca nazwę bieżącego użytkownika bazy danych

```
PRINT CURRENT_USER
```

ISNULL

Podstawia wskazaną wartość za wartość pustą

```
PRINT ISNULL (NULL, '123')
```

ISNUMERIC

Sprawdza, czy wartość jest liczbą

```
PRINT ISNUMERIC (123)
```

NULLIF

Zwraca NULL jeśli oba wyrażenia są równe

```
PRINT NULLIF ('abc', 'abc')
```

SESSION_USER

Zwraca nazwę bieżącego użytkownika bazy danych

```
PRINT SESSION_USER
```

SYSTEM_USER

Zwraca login bieżącego użytkownika bazy danych

```
PRINT SYSTEM_USER
```

USER_NAME

Zwraca nazwę bieżącego użytkownika bazy danych

```
PRINT USER_NAME ()
```

3.6. Funkcje agregujące - zastosowania

Zestaw funkcji agregujących pozwala na grupowanie i zliczanie elementów tabel lub kwerend i pozwala przedstawić je w formie zwięzłego wyniku. Postanowiłeś określić na początek maksymalną, minimalną i średnią wartość sprzedaży oraz poznać łączną ilość transakcji.

```
SELECT
    max(Wartość) AS Maksimum,
    min(Wartość) AS Minimum,
    avg(Wartość) AS Średnia,
    count(Wartość) AS Ilość
FROM AnlizaSprzedaży
```

Korzystając z okazji chciałbyś także wyświetlić operacje o wartości wyższej od średniej. Ta operacja wymaga zastosowania podkwerendy wyliczającej średnią wartość sprzedaży.

```
SELECT *
FROM AnlizaSprzedaży
WHERE Wartość > (SELECT avg(Wartość) FROM AnlizaSprzedaży)
```

Istotną informacją okazała się wartość sprzedaży w roku 2016 z podziałem na miesiące. Do poprawnego jej policzenia wymagane może być zastosowanie dodatkowej podkwerendy.

```
SELECT SUM(Wartość) AS ŁącznaWartość, Miesiąc
FROM
    (SELECT Wartość, MONTH(OrderDate) AS Miesiąc
    FROM AnlizaSprzedaży
    WHERE YEAR(OrderDate) = 2016) AS X
GROUP BY Miesiąc
ORDER BY Miesiąc
```

W zoptymalizowanej formie funkcja będzie wyglądała następująco:

```
SELECT SUM(Wartość) AS ŁącznaWartość, MONTH(OrderDate) AS Miesiąc
FROM AnlizaSprzedaży
WHERE YEAR(OrderDate) = 2016
GROUP BY MONTH(OrderDate)
```

Na koniec testowania funkcji agregujących chciałbyś poznać ilość i wartość transakcji dla klienta 'VINET'.

```
SELECT COUNT(OrderID) AS Ilość, SUM(Wartość) AS ŁącznaWartość, CustomerID
FROM AnlizaSprzedaży
GROUP BY CustomerID
HAVING CustomerID = 'VINET'
```

HAVING a WHERE

Zarówno HAVING jak i WHERE służą do filtrowania wyników. Różnią się tym, że WHERE filtruje dane przed agregacją, a HAVING już po jej zakończeniu. Możliwe jest jednak ich równoczesne stosowanie.

```
SELECT ProductID, SUM(quantity) AS Quantity
FROM [Order Details]
WHERE Quantity > 10
GROUP BY ProductID
HAVING SUM(quantity) > 100
```

Odpowiednie stosowanie powyższych warunków pomaga znacząco zoptymalizować wykonanie kwerend.

4. Język DML (Data Manipulation Language)

Podczas pracy z danymi bazy danych może się pojawić konieczność modyfikacji danych oraz przenoszenia danych pomiędzy bazami SQL. Przed przystąpieniem do dalszych kroków sprawdź bieżącą bazę danych polecenie **USE**.

- Aktualizacja danych z zastosowaniem UPDATE
- Dodawanie rekordów do tabeli z użyciem polecenia INSERT
- Usuwanie danych z przy zastosowaniu polecenia DELETE
- Wyprowadzanie wyniku zapytania do nowej tabeli z użyciem SELECT INTO

4.1. Tworzenie tabeli SELECT INTO

Pierwszą taką operacją będzie przeniesienie danych **Kontrahentów** z bazy **Northwind** do bazy **MojaBaza**. Obie bazy znajdują się na tym samym serwerze, więc operacja będzie wymagała wyłącznie użycie kwerendy tworzącej tabelę.

```
SELECT *  
    INTO   Kontrahenci  
    FROM   Customers
```

4.2. Dodawanie rekordów INSERT INTO

Istnieje możliwość dodawania rekordów pojedynczo, jeden po drugim lub bezpośrednio jako wyniku innego zapytania. W tym przykładzie dodaj nowego kontrahenta do tabeli **Kontrahenci**.

```
INSERT INTO Kontrahenci  
    (CustomerID ,CompanyName,ContactName)  
VALUES  
    ('BBHHG', 'Firma X', 'Jan Nowak')
```

4.3. Aktualizacja rekordów UPDATE

Zaktualizuj dane uprzednio dodanej osoby kontaktowej kontrahenta do 'Marian Kowalski'

```
UPDATE Kontrahenci  
    SET ContactName = 'Marian Kowalski'  
    WHERE CustomerID = 'BBHHG'
```

Przed procedurą aktualizacji danych dobrym pomysłem jest przygotowanie najpierw kwerendy, która będzie wyświetlała dane, a dopiero po weryfikacji jej działania zostanie uruchomiona właściwa kwerenda aktualizująca. W

```
SELECT * FROM [Order Details]  
    WHERE Discount = 0 AND UnitPrice < 20
```

Właściwa kwerenda aktualizująca dane.

```
UPDATE [Order Details] SET Discount = 0.05  
    WHERE Discount = 0 AND UnitPrice < 20
```

Kwerenda aktualizująca resztę danych

```
UPDATE [Order Details] SET Discount = 0.15  
    WHERE Discount = 0 AND UnitPrice >= 20
```

4.4. Usuwanie rekordów DELETE

Na koniec usuń nowododanego kontrahenta z bazy danych.

```
DELETE  
    FROM Kontrahenci  
    WHERE CustomerID = 'BBHHG'
```


5. Język DDL (Data Definition Language)

Tworzenie i modyfikacja obiektów SQL. Tworzenie i zarządzanie bazą danych jest możliwe z poziomu narzędzi graficznych Microsoft Management Studio jak i za pomocą poleceń SQL. W tej części podręcznika zostaną przedstawione elementy. Do poprawnego działania tych funkcji niezbędne jest posiadanie uprawnień administracyjnych. Tworzenie obiektów w istniejącej bazie danych jest możliwe przy posiadaniu uprawnień administracyjnych dla wskazanej bazy.

- Usuwanie obiektów z użyciem polecenia DROP
- Tworzenie tabeli: definiowanie klucza głównego, wymuszenia unikatowych wartości dla kolumn, zastosowanie wartości domyślnych.
- Modyfikacja istniejącej tabeli z użyciem ALTER TABLE
- Tworzenie Indeksu dla tabeli.
- Tworzenie i modyfikacja relacji między tabelami – pojęcie klucza głównego i obcego.
- Tworzenie i modyfikacja widoku/perspektywy z poziomu kodu SQL.
- Kontrola poprawności danych dla danych w tabel.

5.1. Tworzenie bazy danych

Zaloguj się do bazy z uprawnieniami administracyjnymi. Za tworzenie obiektów odpowiada polecenie **CREATE**. Otwórz New Query i wpisz polecenie:

```
CREATE DATABASE MojaBaza
```

5.2. Usuwanie bazy danych

Do usuwania obiektów służy polecenie DROP. Usuń aktualną bazę poleceniem:

```
DROP DATABASE MojaBaza
```

Nie zapomnij utworzyć bazy ponownie i przejdź do nowej bazy poleceniem USE.

```
USE MojaBaza
```

5.3. Tworzenie tabeli

Utwórz tabelę Moja tabela korzystając z poleceń SQL

```
CREATE TABLE MojaTabela  
(ID INT NOT NULL,  
Imię NVARCHAR(50),  
Nazwisko NVARCHAR(50),  
Wynagrodzenie DECIMAL(18,2))
```

5.4. Modyfikacja tabeli

Modyfikacja obiektów możliwa jest przy zastosowaniu polecenia **ALTER**. Dodaj pole uwagi do istniejącej tabeli.

```
ALTER TABLE MojaTabela ADD Uwagi NVARCHAR(200)
```

5.5. Usuwanie tabeli

Każdy z obiektów bazy danych można usunąć poprzez zastosowanie polecenia DROP

```
DROP TABLE MojaTabela
```

Dodatkowe opcje tworzenia i zarządzania bazą danych są dostępne przy użyciu narzędzi Management Studio.

5.6. Relacje

Relacje bazy danych to powiązania między tabelami, które są tworzone za pomocą instrukcji łączenia w celu zapewnienia integralności bazy danych.

Każda poprawnie zaprojektowana baza danych, to poprawnie zaprojektowane tabele, w których poprawnie wyznaczono klucze oraz zdefiniowano typy relacji. Istnieją jednak sytuacje, gdy wprowadzenie relacji może utrudnić import i przetwarzanie danych, dlatego nie zawsze są one niezbędne do pracy.

Dla poprawnego działania relacji niezbędne jest umieszczanie w tabeli kolumny identyfikatorów, zawierającej kolejne liczby całkowite. Numery te są zwykle sztuczne i nie wynikają z danych zawartych w tabeli. Takie podejście upraszcza identyfikację poszczególnych rekordów, a sztucznie nadawany numer gwarantuje unikalność poszczególnych rekordów, w przypadku danych realnych, zawsze istnieje ryzyko powtórzenia wartości.

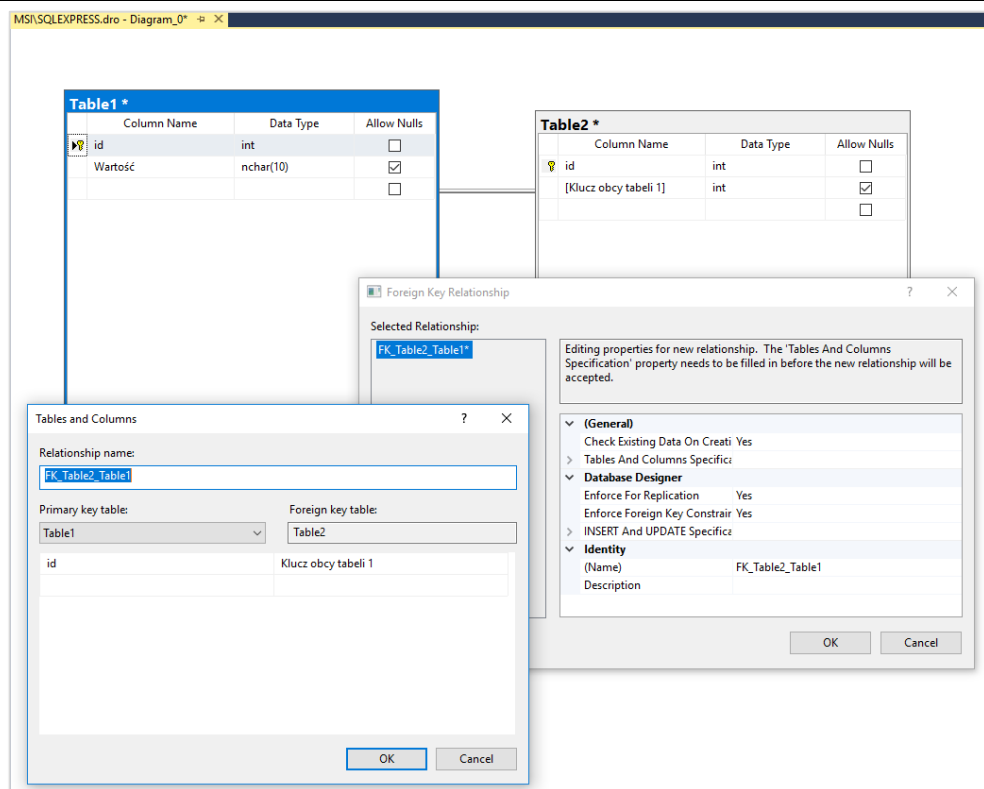
Pole tabeli, identyfikujące poszczególne rekordy nazywamy **kluczem podstawowym**. W dobrze zaprojektowanej bazie danych, dane występują tylko raz, nie powtarzają się w innych tabelach. Podstawą projektu bazy danych jest poprawne zdefiniowanie tabel a w nich kluczy. Umieszczając klucz główny jednej tabeli w innej tabeli klucz obcy, definiujemy tym samym relację między tymi tabelami.

Typy relacji

- **Jeden do jednego** - obydwie tabele zawierają tylko jeden rekord po każdej stronie relacji.
- **Jeden do wielu** - tabela klucza podstawowego zawiera tylko jeden rekord, który dotyczy żadnego, jednego lub wielu rekordów w powiązanej tabeli.
- **Wiele do wielu** – każdy z rekordów w obu tabelach może się odnosić do dowolnej liczby rekordów lub nie odnosić się do żadnego z rekordów w drugiej tabeli. W przypadku takich relacji wymagana jest trzecia tabela nazywana tabelą powiązań, ponieważ systemy relacyjne nie mogą bezpośrednio obsługiwać takiej relacji.

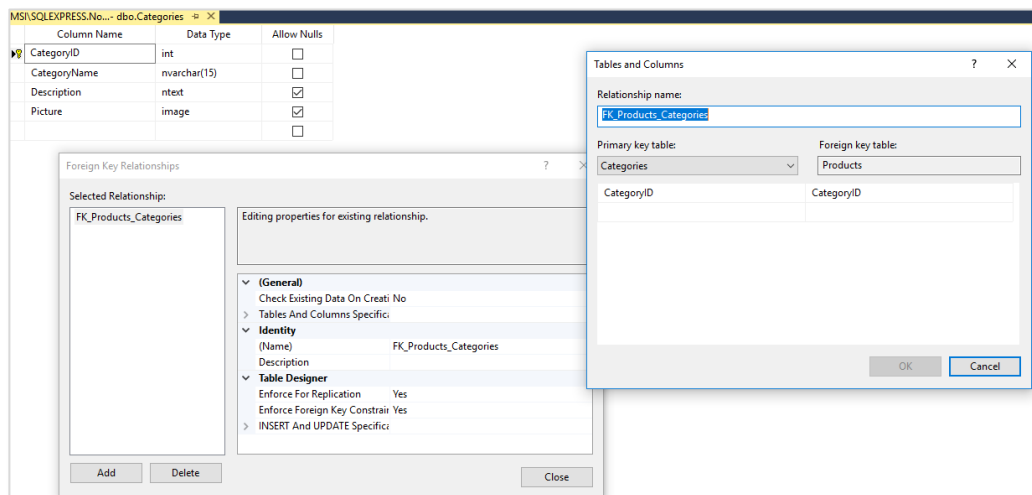
Relacje w trybie graficznym

Korzystając ze wsparcia narzędzi graficznych możliwe jest tworzenie relacji w widoku diagramu. Ten sposób jest dostępny z poziomu Microsoft Management Studio. W tym trybie cała praca sprowadza się do kilku ruchów myszą, polegających na połączeniu ze sobą pól w dwóch tabelach.



Relacje w trybie projektu tabeli

Drugą metodą jest dodawanie relacji w widoku projektu tabeli.



W tym trybie istnieje także możliwość (przed zapisaniem zmian) wygenerowania kodu SQL definiującego zmiany.

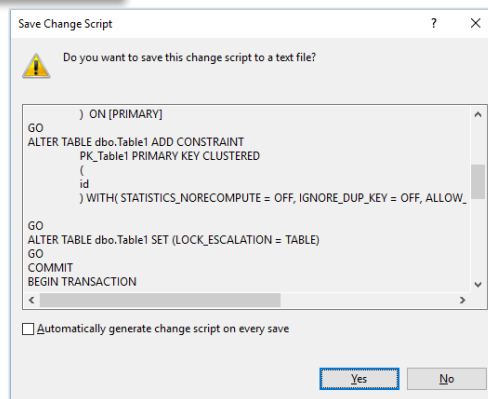


Dodatkowe parametry tworzone podczas generowania relacji definiują kaskadową aktualizację i usuwanie rekordów. Zapobiega to przypadkowej modyfikacji lub skasowaniu danych.

W trybie SQL mają one zapis:

ON UPDATE NO ACTION

ON DELETE NO ACTION



Relacje w trybie SQL

Najwydajniejszym i dającym największą kontrolę nad tworzonymi relacjami jest tryb SQL, gdzie cała procedura zostaje zapisana w formie zwięzłego kodu. Skrypt relacji dla bazy ZKM ma następującą postać:

```
ALTER TABLE Kursy ADD CONSTRAINT
    FK_Kursy_Pojazdy FOREIGN KEY (pojazd)
REFERENCES Pojazdy (id)
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
GO
ALTER TABLE Kursy ADD CONSTRAINT
    FK_Kursy_Kierowcy FOREIGN KEY (kierowca)
REFERENCES Kierowcy (id)
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
GO
```

Usuwanie relacji

Jest możliwe we wszystkich powyższych trybach. W trybie SQL usunięcie relacji będzie miało postać:

```
ALTER TABLE Kursy
    DROP CONSTRAINT FK_Kursy_Kierowcy
```

5.7. Indeksy

Indeks to specjalna struktura danych wprowadzona w celu zwiększenia prędkości wykonywania operacji na tabeli, zwłaszcza operacji wyszukiwania. Indeks w bazie danych jest odpowiednikiem spisu treści w książce. Może być stworzony na jednej lub więcej kolumnach tabeli lub widoku. Przyspiesza on znacznie wyszukiwanie, ale aktualizacja danych w takiej tabeli jest dużo wolniejsza – aktualizacji wymagane jest przebudowanie indeksu.

Przykładowy indeks stwórz dla tabeli apteki, zawierającej unikalne pole **Nazwa** i kontrolę poprawności dla **Rabat**:

```
CREATE TABLE Apteki
(id_apteki INT IDENTITY (1,1) PRIMARY KEY,
Nazwa NVARCHAR(255) UNIQUE,
Adres NVARCHAR(255),
Rabat tinyint DEFAULT 0 CHECK ([Rabat] BETWEEN 0 AND 25))
```

Aktualny wykaz istniejących dla bieżącej bazy danych kontroli poprawności kolumn uzyskasz stosując zapytanie:

```
SELECT * FROM sys.check_constraints
```

Nowy indeks dla kolumny nazwa możesz stworzyć za pomocą poniższego polecenia.

```
CREATE INDEX idx_apteki_nazwa
ON apteki (nazwa)
```

Usunięcie indeksu jest realizowane za pomocą polecenia **DROP**.

```
DROP INDEX apteki.idx_apteki_nazwa
```

Aktualny wykaz istniejących dla bieżącej bazy danych indeksów uzyskasz wydając polecenie:

```
SELECT * FROM sys.indexes
```

Argumenty polecenia CREATE INDEX	
Argument	Wyjaśnienie
UNIQUE	Nie dopuszcza się występowania dwóch takich samych wierszy. Jeśli mimo wszystko zajdzie taka sytuacja, to indeks nie zostanie utworzony.
CLUSTREAD / NONCLUSTREAD	Określenie sposobu tworzenia indeksu. Można utworzyć tylko 1 indeks klastrowy i 249 nieklastrowych. Domyślnie – indeksy nieklastrowe
Nazwa_indeksu	Określa nazwę tworzonego indeksu
tabela widok	Nazwa tabeli lub perspektywy (widoku) na podstawie, na której ma zostać utworzony indeks
Kolumna	Nazwa kolumny lub kolumn (gdy ma być to indeks złożony)
ASC \ DESC	Określenie sposobu sortowania indeksu (rosnąco –ASC, malejąco – DESC), domyślnie indeks jest posortowany rosnąco
ON grupa_plików	Określa grupę plików, w których indeks ma być tworzony
PAD_INDEX	Określa, że powinna być zostawiona przestrzeń dla przyszłych aktualizacji indeksów (przydaje się np. przy atrybutach typu <i>varchar</i>)
FILLFACTOR = współczynnik wypełnienia	Określa, jaki procent bloku ma zostać zapełniony
IGNORE_DUP_KEY	Używany z klauzulą UNIQUE. Jeśli nie będzie klauzuli IGNORE_DUP_KEY, i wystąpi próba dodania indeksu o istniejącej wartości – to indeks nie zostanie utworzony, jeśli natomiast stworzymy indeks z klauzulą IGONRE_DUP_KEY, to w tej samej sytuacji zdublowany klucz zostanie odrzucony, ale indeks zostanie utworzony.
DROP EXISTING	Używany do ponownego tworzenia istniejących indeksów, przy tworzeniu indeksów klastrowych można odczuć wzrost wydajności, ponieważ tworzą się na nowo indeksy nieklastrowe
STATISTICS_NO_RECOMPUTE	Oznacza, że statystyki indeksu nie mają być na bieżąco uaktualniane
SORT IN TEMPDE	Oznacza, że bezpośrednie wyniki sortowania mają być przechowywane w bazie tempdb, jeśli baza ta znajduje się na innym dysku, niż baza z danymi użytkownika, to znacznie zwiększa to wydajność

6. Zaawansowane zapytania wybierające SQL

6.1. Zagnieżdżanie zapytań

Podczas tworzenia zapytań do bazy danych można korzystać z dowolnej liczby tabel i widoków. Jednak w sytuacji, gdy zostanie utworzonych dużo widoków w bazie zaczyna pojawiać się chaos. Rozwiązaniem tego problemu jest zagnieżdżanie zapytań. Dodatkowym plusem takiego rozwiązania jest optymalizacja samego zapytania, co powoduje jego szybsze wykonanie.

Przykładowa kwerenda łącząca dane z tabeli i kwerendy, gdzie jedna z nich jest generowana dynamicznie jako zapytanie zagnieżdżone.

```
SELECT Customers.CustomerID, Customers.City,  
       Y.Country, Y.Region  
FROM   Customers INNER JOIN  
       (SELECT * FROM CustomersOLD WHERE Region IS NOT NULL) AS Y  
ON Customers.CustomerID = Y.CustomerID
```

Konstrukcja ta nie ogranicza się jedynie do tabel lokalnych. Przykład zapytania łączącego za pomocą **INNER JOIN** tabelę **Customers** z serwera lokalnego z pobraną za pomocą kwerendy tabelą **Customers** z Linked Server:

```
SELECT Customers.CustomerID, Customers.CompanyName,  
       Customers.City, x.address  
FROM Customers INNER JOIN  
       (SELECT *  
        FROM OPENQUERY  
        ([mój_mysql], 'SELECT * FROM northwind.Customers')) AS x  
ON  
Customers.CustomerID = x.CustomerID
```

Przykład wielokrotnie zagnieżdżanych kwerend, gdzie wynik jest podstawiany pod WHERE

```
SELECT *  
FROM ORDERS WHERE  
CustomerID IN (SELECT CustomerID FROM  
              (SELECT Customers.CustomerID, Customers.City,  
                     Y.Country, Y.Region  
FROM Customers  
INNER JOIN  
      (SELECT *  
       FROM CustomersOLD  
       WHERE Region IS NOT NULL) AS Y  
ON  
Customers.CustomerID = Y.CustomerID) AS X)
```

Dzięki poniższej konstrukcji nastąpiła optymalizacja działania kwerendy poprzez przeniesienie operacji obliczenia do ostatniego kroku zapytania. Szczególnie istotne jest stosowanie sortowania w ostatnim kroku, co znacząco przyspiesza wykonanie obliczeń.

```
SELECT TOP 100 PERCENT Orders.OrderID, Wartość, Wartość * 0.23 AS VAT  
FROM ORDERS  
INNER JOIN  
  (SELECT OrderID, UnitPrice, Quantity, UnitPrice*Quantity AS Wartość  
FROM [Order Details]) AS X  
ON Orders.OrderID = x.OrderID  
WHERE CustomerID IN  
  (SELECT CustomerID FROM Customers WHERE Region IS NULL)  
ORDER BY Wartość
```

6.2. Obsługa NULL

Null jest specjalnym znacznikiem w języku SQL, wskazujący, że dana nie istnieje w bazie danych. Znacznik Null z powodu powiązanej z nim logiki trójwartościowej i specjalnych wymagań wobec stosowania go w złączeniach wymagania specjalnej obsługi w funkcjach agregujących i operatorach grupujących.

Spróbuj połączyć dwa pola z tabeli Customers razem: City i Region – do ich połączenia możesz użyć znaku +.

```
SELECT City + Region FROM Customers
```

Zadanie zostało wykonane, ale zwróć uwagę, że jeśli jedno z pól zawierało NULL, to wynik połączenia dwóch tekstów dawał NULL. Jedną z metod na obsłużenie wartości null, jest zastosowanie funkcji ISNULL, która w przypadku natrafienia na wartość nieokreśloną NULL powoduje wyświetlenie drugiego argumentu funkcji.

```
SELECT City + ISNULL(Region, ' ?') FROM Customers
```

Działanie powyższej funkcji jest zadowalające, ale zwróć uwagę, że dla każdego pola, które może zostać użyte w zapytaniu konieczne jest użycie funkcji ISNULL. W przypadku łączenia tekstów, może to być problematyczne. Rozwiązaniem pozwalającym łączyć teksty jest funkcja CONCAT, która to w przypadku wystąpienia wartości pustej, traktuje ją jako pusty łańcuch znaków.

```
SELECT CONCAT(City , Region, 1) FROM Customers
```

Dodatkowa funkcjonalność jaką daje CONCAT to konwersja podstawionych danych do formatu tekstowego. Sprawdź jak się zachowa funkcja po podstawieniu daty lub liczby.

```
SELECT CONCAT(City , Region, GETDATE()) FROM Customers
```

Wartość NULL powoduje także pewne problemy w kwerendach zliczających. Sprawdź jaki będzie wynik poniższej kwerendy.

```
SELECT COUNT(REGION) FROM Customers
```

Następnie porównaj go z kolejną. Zwróć uwagę na różnice w ilości policzonych rekordów.

```
SELECT COUNT(*) FROM Customers
```

W końcowej wersji możesz to naprawić, zamieniając wartość pustą na 0. O ile zajdzie taka konieczność.

```
SELECT COUNT(ISNULL(REGION,0)) FROM Customers
```

6.3. Funkcje okien

Funkcje okien pozwalają na szeregowanie zbiorów i umożliwiają prowadzenie rankingu z numeracją rekordów włącznie. Wyświetl najpierw listę wszystkich zamówień.

```
SELECT * FROM [Order Details]
```

Agregacja

Możesz zastosować grupowanie danych z powyższej tabeli, w celu wyliczenia wartości poszczególnych pozycji zamówień i pogrupowania ich w ramach zamówień.

```
SELECT OrderID,  
       SUM(UnitPrice * Quantity) as Wartość  
FROM [Order Details]  
GROUP BY OrderID
```

Dla powyższych przykładów możesz użyć dowolnych innych funkcji agregujących. Problem zaczyna się jednak pojawiać, gdy chcielibyśmy uzyskać w wyniku zapytania kolumny niezgrupowane.

Funkcja OVER()

Funkcji okna OVER() używa się zazwyczaj w połączeniu z funkcjami szeregującymi. Jako ich nierozłączny element służy do określania zakresu i sposobu nadawania numeracji wierszy. Daje ona dostęp do wszystkich kolumn, nie tylko do atrybutów grupujących.

```
SELECT OrderID, ProductID ,
       SUM(UnitPrice * Quantity) OVER (PARTITION BY OrderID ) as Wartość
FROM [Order Details]
```

Funkcje LAST_VALUE() i FIRST_VALUE()

Stosując funkcje okien możemy używać dodatkowych funkcji w ramach agregacji – FIRST_VALUE() i LAST_VALUE(), oznaczają one odpowiednio pierwszy i ostatni element w zagregowanej grupie.

```
SELECT OrderID, ProductID , Quantity,
       FIRST_VALUE(Quantity) OVER (PARTITION BY OrderID ORDER BY OrderID) AS
       PierwszaWartość,
       LAST_VALUE(Quantity) OVER (PARTITION BY OrderID ORDER BY OrderID) AS
       OstatniaWartość
FROM [Order Details]
```

Funkcja ROW_NUMBER()

W ramach wybranej uprzednio grupy możesz uzyskać także numer wiersza. Dodaj w tym celu do wyniku kwerendy numer wiersza ROWNUMBER(). Używając numeracji wierszy funkcja OVER musi zawierać narzucone odgórnie sortowanie wg wybranego kryterium w ramach grupy.

```
SELECT OrderID, ProductID ,
       ROW_NUMBER() OVER (PARTITION BY OrderID ORDER BY OrderID ) AS Wiersz,
       SUM(UnitPrice * Quantity) OVER (PARTITION BY OrderID ) as Wartość
FROM [Order Details]
```

Numeracja wierszy może mieć podobne zastosowanie jak w arkuszu Excela. Dodaj w pierwszej kolumnie nr wiersza, pamiętając o sortowaniu.

```
Select ROW_NUMBER() OVER (ORDER BY Country desc) as Wiersz, *
FROM Customers
```

Numeracja może się odbywać wg mniejszych fragmentów. Podziel powyższy widok na partycje wg kraju i ponumeruj jego wystąpienia

```
Select ROW_NUMBER() OVER (PARTITION BY Country ORDER BY Country desc) as Wiersz,
       Country AS Kraj, *
FROM Customers
```

Ponumeruj firma w ramach poszczególnych krajów.

```
SELECT ROW_NUMBER() OVER (ORDER BY Country) AS LP, *
FROM Customers
```

Fragmentowane okna PARTITION BY

Zmodyfikuj widok okna dodając PARTITION BY

```
SELECT ROW_NUMBER() OVER (PARTITION BY Country ORDER BY Country) AS LP,
       * FROM Customers
```

Przetestuj działanie partycjonowania poprzez zmianę porządku sortowania

```
SELECT ROW_NUMBER() OVER (PARTITION BY Country ORDER BY City) AS LP,
       * FROM Customers
```


Podczas tworzenia partycji możliwe jest dalsze jej fragmentowanie – użyj Country i City jako partycji.

```
SELECT ROW_NUMBER() OVER (PARTITION BY Country, City ORDER BY City) AS LP, *
FROM Customers
```

Rozmiar ramki ROWS

Określenie rozmiaru ramki z dokładnością do pojedynczego wiersza jest dostępne poprzez użycie ROWS. Początek i koniec przedziału, definiujemy używając :

- UNBOUNDED PRECEDING – wszystkie rekordy od początku ramki (także poprzedzające wiersz ramki)
- <unsigned integer> PRECEDING – liczba wierszy poprzedzających
- CURRENT ROW - bieżący wiersz dla wyznaczonej ramki
- <unsigned integer> FOLLOWING – liczba wierszy po danym elemencie.
- UNBOUNDED FOLLOWING – wszystkie wiersze do końca okna

```
SELECT OrderID, ProductID , Quantity,
SUM(Quantity) OVER (PARTITION BY OrderID ORDER BY OrderID
ROWS BETWEEN 0 PRECEDING AND 2 FOLLOWING ) AS IlośćŁącznie
FROM [Order Details]
```

Zakres ramki RANGE

Różnica pomiędzy ROWS a RANGE to inna interpretacji przedziałów, których jest 3.

- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW – od początku przedziału do danego zakresu określonego przez wartość elementu sortującego
- RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING – jw. W odwrotnym kierunku.
- RANGE CURRENT ROW – wiersze, dla których atrybut sortowany jest równy bieżącemu rekordowi w ramach okna.

```
SELECT OrderID, ProductID , Quantity,
SUM(Quantity) OVER (PARTITION BY OrderID ORDER BY OrderID
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS IlośćŁącznie
FROM [Order Details]
```

Optymalizacja okien

Tworząc zaawansowane zapytania do bazy danych możesz zdecydować o kolejności wykonania się poszczególnych fragmentów zapytania. Możesz najpierw połączyć tabele, wykonać obliczenia, a na ich podstawie wykonać szeregowanie danych. Będzie to miało wpływ na czas wykonania całej kwerendy.

```
SELECT *,
SUM(Wartość) OVER (ORDER BY OrderID) AS WartośćZamówienia
FROM
(SELECT Orders.OrderID, OrderDate, CompanyName, (UnitPrice * Quantity) AS Wartość
FROM Orders INNER JOIN
[Order Details] ON Orders.OrderID = [Order Details].OrderID INNER JOIN
Customers ON Orders.CustomerID = Customers.CustomerID) AS X
```

Przetestuj pozostałe funkcje agregujące przy zastosowaniu okien.

```
SELECT *,
SUM(Wartość) OVER (ORDER BY OrderID) AS WartośćZamówienia,
MAX(Wartość) OVER (ORDER BY OrderID) AS MaxWartośćZamówienia,
MIN(Wartość) OVER (ORDER BY OrderID) AS MinWartośćZamówienia,
AVG(Wartość) OVER (ORDER BY OrderID) AS ŚredniaWartośćZamówienia,
COUNT(Wartość) OVER (ORDER BY OrderID) AS LiczbaPozycji
FROM
(SELECT Orders.OrderID, OrderDate, CompanyName, (UnitPrice * Quantity) AS Wartość
FROM Orders INNER JOIN
[Order Details] ON Orders.OrderID = [Order Details].OrderID INNER JOIN
Customers ON Orders.CustomerID = Customers.CustomerID) AS X
```

Wyświetl wartość całości zamówienia obok pojedynczej wartości zamówienia.

```
SELECT *,  
SUM(Wartość) OVER (PARTITION BY CompanyName ORDER BY OrderID) AS WartośćZamówienie  
FROM  
(SELECT Orders.OrderID, OrderDate, CompanyName, (UnitPrice * Quantity) AS Wartość  
FROM Orders INNER JOIN  
[Order Details] ON Orders.OrderID = [Order Details].OrderID INNER JOIN  
Customers ON Orders.CustomerID = Customers.CustomerID) AS X
```

Przelicz wartość procentową pozycji pojedynczego zamówienia do wszystkich zamówień

```
SELECT *,  
Wartość/SUM(Wartość) OVER (ORDER BY OrderID) AS ProcentZamówienia  
FROM  
(SELECT Orders.OrderID, OrderDate, CompanyName, (UnitPrice * Quantity) AS Wartość  
FROM Orders INNER JOIN  
[Order Details] ON Orders.OrderID = [Order Details].OrderID INNER JOIN  
Customers ON Orders.CustomerID = Customers.CustomerID) AS X
```

Funkcja RANK()

Nadaje kolejne wartości wierszom w ramach zbioru, w przypadku wyniku ex aequo – numery są pomijane.

```
SELECT RANK() OVER (PARTITION BY Country ORDER BY City) AS Pozycja, *  
FROM Customers
```

Funkcja DENSE_RANK()

Działa tak jak RANK() z różnicą – jest funkcją ciągłą. Inkrementuje wartości bez względu na faktyczną pozycję w rankingu, nadaje dokładnie kolejny numer pozycji.

```
SELECT DENSE_RANK() OVER (PARTITION BY Country ORDER BY City) AS LP, *  
FROM Customers
```

Funkcja NTILE(n)

Funkcja szeregująca NTILE(n), tak jak pozostałe rankingowe, działa w oparciu o funkcję OVER(). Działa dzieląc okno na kafelki n-tile. W poniższym przypadku zbiór jest podzielony na 3 kafelki.

```
SELECT NTILE(3) OVER (ORDER BY Country) AS Grupa, *  
FROM Customers
```

6.4. CTE (Common Table Expression)

CTE to nazwane zapytanie reprezentujące tymczasowy zestaw rekordów definiowany w zasięgu jednego polecenia SELECT, INSERT, UPDATE lub DELETE. Składnia CTE zbliżona jest do podkwerendy języka T-SQL lub elementu obliczanego w języku MDX.

Wykorzystywana składnia CTE jest rozpoznawalna, dzięki zastosowaniu słowa kluczowego WITH, które w T-SQL wykorzystywane jest podawania wskazówek dla optymalizatora lub do deklarowania przestrzeni nazw XML (klauzula WITH XMLNAMESPACES).

Tworząc zapytanie CTE należy pamiętać o stosowaniu średnika, po poprzedzającym poleceniu. Pozwoli to uniknąć błędu parsowania. Przykład prostego użycie CTE, gdzie definiowane jest nazwane podzapytanie, które następnie jest wykorzystywane w poleceniu SELECT jako źródło danych.

```
WITH Pracownicy  
AS  
(SELECT *  
FROM Northwind.dbo.Customers  
WHERE Country = 'UK')
```

```
SELECT * FROM Pracownicy
```

Tworząc zapytanie CTE można tworzyć aliasy dla kolumn oraz wykonywać wszelkie operacje związane z przetwarzaniem kwerendy. Dla pojedynczego polecenia DML można użyć więcej niż jednego CTE, co pozwala na dochodzenie do docelowego krok po kroku. Każde kolejne definiowane CTE może odwoływać się do CTE wcześniej zdefiniowanych.

Utworzone uprzednio CTE można łączyć ze zwykłymi tabelami i widokami, uzyskując coraz bardziej skomplikowane zapytania. Samo zapytanie nie musi się ograniczać wyłącznie do instrukcji SELECT.

```
-- tworzenie CTE

WITH Obroty
AS
    (SELECT OrderID, CustomerID AS Klient, SUM(Wartość) AS Wartość
    FROM
        (SELECT Orders.OrderID, CustomerID, (UnitPrice * Quantity) AS Wartość
        FROM    [Order Details] INNER JOIN
            Orders ON [Order Details].OrderID = Orders.OrderID) AS X
    GROUP BY OrderID, CustomerID)

-- łączenie CTE z tabelą i przekierowywanie wyniku do nowej tabeli

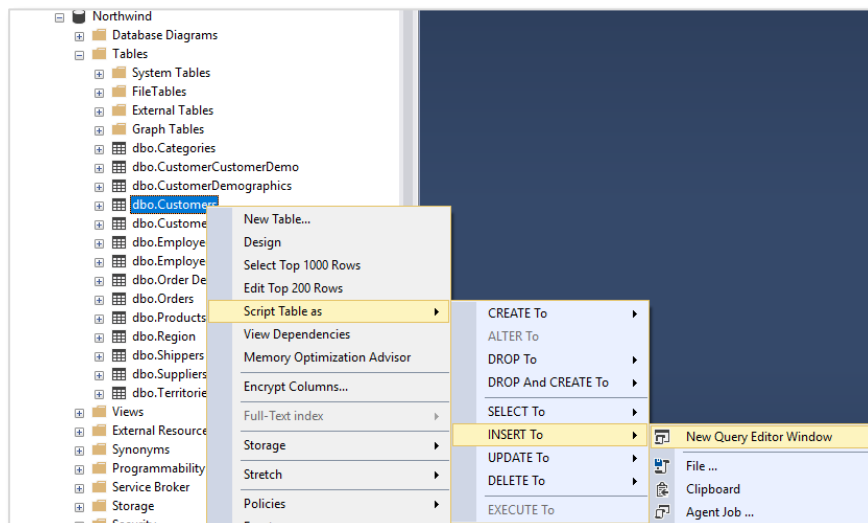
SELECT CustomerID, CompanyName, ContactName, OrderID,
CONVERT(Decimal (18,4), Wartość) AS Wartość,
CONCAT(Country, ' ', Region) AS Kraj,
ISNULL(Fax, '') AS Fax
INTO CustomerTMP                                -- utworzenie obiektu
FROM Customers INNER JOIN
Obroty ON Customers.CustomerID = Obroty.Klient
```

7. Wprowadzenie do programowania w T-SQL

7.1. Procedury składowane

Do tego momentu wszystkie polecenia akcji były wykonywane doraźnie. Nadszedł czas na procedurę, która zostanie umieszczona na serwerze i będzie mogła być uruchamiana na żądanie.

Procedura dodająca kontrahenta do bazy danych



Prostym sposobem na wygenerowanie gotowego skryptu dołączającego dane do tabeli jest skorzystanie z wbudowanego narzędzia. Po dokonaniu kilku niezbędnych modyfikacji, takich jak dodanie zmiennych można zapisać ją na serwerze.

Przydatne: Podczas tworzenie procedury możesz używać zmiennych. Oznacza się je znakiem @ muszą zawierać typ danych. Zmienne systemowe posiadają oznaczenie @@.

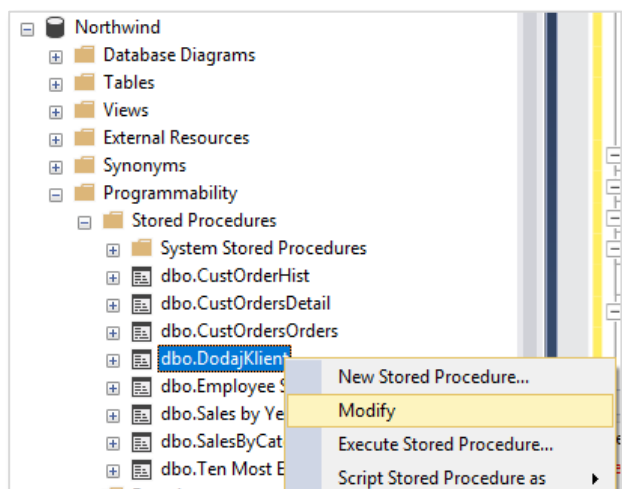
Do zapisu procedury służy polecenie CREATE PROCEDURE. Po zakończeniu wpisaniu odpowiedniego kodu SQL należy uruchomić go poleceniem Run/F5. Spowoduje to utworzenie się obiektu typu procedura składowana w bazie danych.

```
CREATE PROCEDURE DodajKlienta
(
    @CustomerID NCHAR(5),
    @CompanyName NVARCHAR(40),
    @ContactName NVARCHAR(30),
    @ContactTitle NVARCHAR(30),
    @Address NVARCHAR(60),
    @City NVARCHAR(15),
    @Region NVARCHAR(15),
    @PostalCode NVARCHAR(10),
    @Country NVARCHAR(15),
    @Phone NVARCHAR(24),
    @Fax NVARCHAR(24))
AS
INSERT INTO Customers
([CustomerID], [CompanyName], [ContactName], [ContactTitle], [Address],
[City], [Region], [PostalCode], [Country], [Phone], [Fax])
VALUES
(@CustomerID, @CompanyName, @ContactName, @ContactTitle, @Address,
@City, @Region, @PostalCode, @Country, @Phone, @Fax)
GO
```

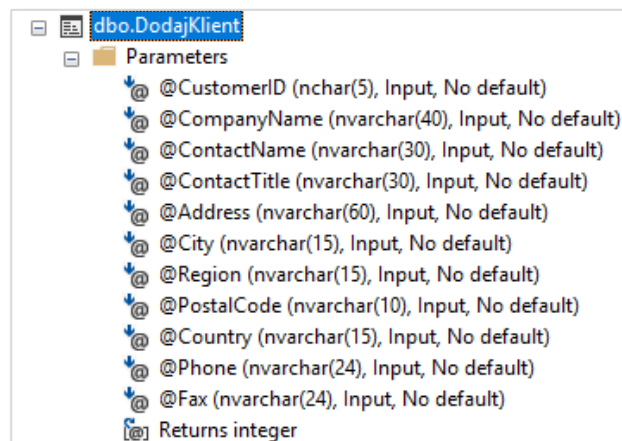
Zapisaną w taki sposób procedurę znajdziesz w Object Explorerze. Jeśli nie będzie widoczna od razu odśwież widok obiektów.

Materiały szkoleniowe

Z tego miejsca możesz także ją zmodyfikować (polecenie Modify) lub zamienić do postaci skryptu (polecenie Script Stored Procedure as).



Rozwijając procedurę masz możliwość podejrzenia parametrów jakie są wymagane do jej uruchomienia, które posiadają typ akceptowanych danych oraz informacje o wartościach domyślnych, jakie może przyjąć procedura.



Usunięcie procedury składowanej możliwe jest za pomocą Object Browser lub polecenia SQL. Podczas uruchamiania procedur warto pamiętać o wskazaniu właściwej bazy danych, której ma dotyczyć akcja. Zapobiegnie to przypadkowemu uruchomieniu kodu w niewłaściwej bazie danych.

```
USE Northwind
GO
DROP PROCEDURE DodajKlient
GO
```

Próba modyfikacji procedury wywoła poniższy kod SQL. Po wprowadzeniu niezbędnych poprawek konieczne jest użycie polecenia Run/F5. Polecenie Zapisz z menu aplikacji zapisze sam kod do pliku, a nie zmiany w obiekcie bazy danych.

```
USE Northwind
GO
ALTER PROCEDURE DodajKlient

(
    @CustomerID NCHAR(5),
    @CompanyName NVARCHAR(40),
    @ContactName NVARCHAR(30),
    @ContactTitle NVARCHAR(30),
    @Address NVARCHAR(60),
    @City NVARCHAR(15),
    @Region NVARCHAR(15),
    @PostalCode NVARCHAR(10),
    @Country NVARCHAR(15),
    @Phone NVARCHAR(24),
    @Fax NVARCHAR(24))

AS
INSERT INTO Customers
(CustomerID, CompanyName ,ContactName, ContactTitle, Address
, City, Region, PostalCode, Country, Phone, Fax)
VALUES
(@CustomerID, @CompanyName, @ContactName, @ContactTitle, @Address,
@City, @Region, @PostalCode, @Country, @Phone, @Fax)
```

Materiały szkoleniowe

Procedurę możesz uruchomić korzystając z polecenie EXEC lub EXECUTE – jeśli (tak jak w tym wypadku) procedura posiada dodatkowe parametry należy je podać.

```
EXECUTE DodajKlient
    @CustomerID = 'XXXXX'
    ,@CompanyName = 'Firma'
    ,@ContactName = 'Osoba kontaktowa'
    ,@ContactTitle = ''
    ,@Address = ''
    ,@City = ''
    ,@Region = ''
    ,@PostalCode = ''
    ,@Country = ''
    ,@Phone = ''
    ,@Fax = ''
GO
```

Przetestuj działanie procedury usuwającej wskazanego kontrahenta z tabeli Customers.

```
CREATE PROCEDURE UsuńKontrahenta
( @id NCHAR(5) )
AS
DELETE FROM Customers
    WHERE CustomerID = @id
```

Stworzoną procedurę możesz uruchomić za pomocą polecenia

```
EXEC UsuńKontrahenta 'ZZZZ'
```

7.2. Procedury systemowe

Oprócz procedur tworzonych przez użytkowników pracujących z bazą danych, serwer zawiera procedury systemowe. Są procedury pozwalające na pracę serwera, ale można z nich korzystać podczas zwykłej pracy. Przykładami takich procedur mogą być:

Wyświetlenie informacji o wskazanej tabeli.

```
exec sp_help customers
```

Wyświetlenie informacji o kolumnach we wskazanej tabeli.

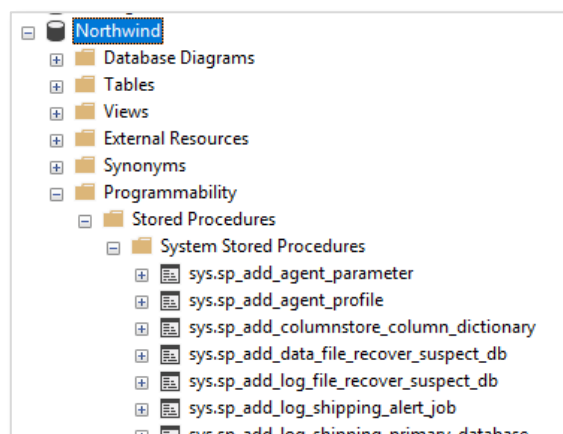
```
exec sp_columns customers
```

Wyświetlenie informacji o istniejących bazach na bieżącym serwerze.

```
exec sp_databases
```

Materiały szkoleniowe

Procedury systemowe dostępne dla danej bazy, widoczne są pod nazwą bazy danych Programmability\Stored Procedures\System Stored Procedures.



7.3. Tworzenie obiektów z użyciem SQL

Przygotuj skrypt, który utworzy nową bazę danych o nazwie Pracownicy. Podczas tworzenia bazy danych sprawdź, czy ona istnieje. Jeśli istnieje to najpierw usuń bazę starą, a później utwórz nową właściwą bazę. Baza ma zawierać pojedynczą tabelę z listą kierowców. Sama tabela ma zawierać informacje takie jak imię, nazwisko, pesel itp. Zdefiniuj dla pola identyfikator (z automatycznym numerowaniem) załóż dodatkowo na nim klucz główny, a na koniec wymuś dla pola PESEL niepowtarzalność oraz załóż na nim indeks, który znacząco poprawi wyszukiwanie informacji w nowej tabeli.

IF EXISTS – warunkowe wykonanie skryptu

W pierwszej kolejności utwórz nową bazę za pomocą skryptu, który sprawdzi istnienie bazy danych. Polecenie IF EXISTS zwraca wartość logiczną w postaci PRAWDA/FAŁSZ i wykonuje się w przypadku potwierdzenia warunku.

```
IF EXISTS
(
    SELECT name FROM master.dbo.sysdatabases
    WHERE name = N'Pracownicy'
)
BEGIN
    SELECT 'Baza danych już istnieje. Zostaje usunięta.' AS Message
    DROP Database Pracownicy
END
ELSE
BEGIN
    CREATE DATABASE [Pracownicy]
    SELECT 'Została utworzona nowa baza danych'
END
```

IF NOT EXISTS

Podobnie jak poprzedni przykład, ten sprawdza nieistnienie obiektu bazy danych i wykonuje się w zależności od spełnienia warunku.

```
IF NOT EXISTS
(
    SELECT name FROM master.dbo.sysdatabases
    WHERE name = N'Pracownicy'
)
BEGIN
    CREATE DATABASE [Pracownicy]
END
ELSE
BEGIN
    SELECT 'Istnieje już baza pracownicy'
END
```

Wyrażenie SOME / ANY

Sprawdza, czy jakikolwiek **EmployeeID** jest mniejszy niż 3, następnie wykonuje lub nie wskazaną akcję.

```
IF 3 < SOME (SELECT EmployeeID FROM Employees)
PRINT 'TRUE'
ELSE
PRINT 'FALSE' ;
```

Wyrażenie ALL

Sprawdza, czy wszystkie **EmployeeID** są mniejsze niż 3, następnie wykonuje lub nie wskazaną akcję.

```
IF 3 < ALL (SELECT EmployeeID FROM Employees)
PRINT 'TRUE'
ELSE
PRINT 'FALSE' ;
```


7.5. Tworzenie tabeli

W kolejnym kroku możesz utworzyć nową tabelę pracownicy. Skrypt może być wykonany jednorazowo lub rozbity na kilka kroków. Niektóre polecenia umieszczone w skrypcie mogą zwrócić komunikat, iż nie muszą być jedynym poleceniem w skrypcie. Problem ten rozwiązuje dodanie dyrektywy GO, która to powoduje, że baza danych widzi nie jeden długi skrypt, a sekwencję krótszych kroków wykonywanych po sobie.

IDENTITY określa automatyczne numerowanie rekordów od 1 co 1. Dodatkowo pole to jest kluczem głównym tabeli (PRIMARY KEY). CONSTRAINT wymusza unikatowe wartości dla wskazanego pola.

Kontrola poprawności danych CHECK wymusza zastosowanie we wprowadzanej wartości do pola tabeli, w tym wypadku PESEL wartości zgodnej z regułą. Tu regułą jest wartość liczbową z zakresu 0-9, a same liczby są ciągiem tekstowym (łańcuchem znaków).

```
USE Pracownicy
GO
CREATE TABLE Kierowca
    (id INT IDENTITY(1,1),
     Imię NVARCHAR(50) NOT NULL,
     Nazwisko NVARCHAR (50),
     PESEL CHAR(11) UNIQUE,
     Email NVARCHAR (100),
     CONSTRAINT id PRIMARY
     (id ASC))
GO
ALTER TABLE Kierowca WITH CHECK
    ADD CONSTRAINT [CK_Kierowca]
    CHECK
        (PESEL like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
GO
ALTER TABLE Kierowca
    CHECK CONSTRAINT [CK_Kierowca]
GO
```

Dodawanie i modyfikacja rekordów

Twoim zadaniem jest dodać do tabeli nową kolumnę o nazwie Skype (typ danych to NVARCHAR (32) z uwagi na możliwe różne długości loginu), następnie do tabeli pracownicy danych nowozatrudnionych osób. Musisz wygenerować mail na podstawie nazwy firmy i danych osoby, a także login w oparciu o dane pracownika i nazwę firmy. Tworząc nową kolumnę w tabeli skorzystasz z polecenia ALTER TABLE oraz ADD umożliwiającego dodanie nowej kolumny. Dodatkowo przydatne może okazać się zastosowanie zmiennej @firma. Do sklejenia tekstów możesz użyć znaków '+' lub funkcji CONCAT, która znacznie lepiej radzi sobie z zadaniem dla wartości NULL.

```
USE Pracownicy;
GO
    ALTER TABLE Kierowca
    ADD skype NVARCHAR (32) ;
GO
-- deklaracja zmiennych
    DECLARE @firma NVARCHAR (10);
-- przypisanie wartości zmiennym
    set @firma = 'firma.pl';
-- dodawanie rekordów
INSERT Kierowca (Imię, Nazwisko, PESEL)
    VALUES ('Marcin', 'Szczaw', '78091503615')
INSERT Kierowca (Imię, Nazwisko, PESEL)
    VALUES ('Janina', 'Majakowska', '79011503615')
INSERT Kierowca (Imię, Nazwisko, PESEL)
    VALUES ('Antonina', 'Hałas', '95031503615')
-- aktualizacja pola skype
UPDATE Kierowca
    SET Skype = CONCAT(REPLACE(@firma, '.pl', ''), '_', Imię, '_', Nazwisko);
-- aktualizacja pola email
UPDATE Kierowca
```

```
SET Email = LOWER(LEFT(Imię, 1)+'.' + Nazwisko + '@'+@firma)
```

Na koniec popraw jednemu z pracowników nazwisko i pozostałe dane Janina Majakowska na Majakowska-Manikowska.

```
DECLARE @firma char (10);
set @firma = 'firma.pl';
UPDATE Kierowca SET nazwisko = nazwisko + '-S Manikowska '
WHERE imię = 'Janina' AND nazwisko = ' Manikowska '
UPDATE Kierowca
SET email = LOWER(LEFT(Imię, 1) + '.' + Nazwisko + '@'+@firma)
skype = CONCAT(replace(@firma, '.pl', ''), '_ ', Imię, '_ ', Nazwisko)
WHERE imię = 'Janina' AND nazwisko = 'Majakowska-Manikowska '
```

Podczas tworzenia tabeli może zaistnieć konieczność sprawdzenia, czy tabela o wskazanej nazwie już nie istnieje w bazie danych. Można to zweryfikować poprzez funkcję IF EXISTS. Stwórz tabelę handlowcy sprawdzając, najpierw, czy taka istnieje. Jeżeli istnieje to ją usuń.

```
IF EXISTS (SELECT * FROM sys.tables WHERE name = 'handlowcy')
BEGIN
    DROP TABLE handlowcy
END

CREATE TABLE Handlowcy(
    lp INT,
    Imię NVARCHAR(50),
    Nazwisko NVARCHAR(50),
    Zatrudniony DATE DEFAULT GETDATE(),
    Pensja DECIMAL(18,4) DEFAULT 1000)
```

Korzystając ze skryptu dodaj do niej kilka pozycji, Zwróć uwagę, że część pól może być wypełniona poprzez użycie wartości domyślnych.

```
INSERT INTO Handlowcy
    (lp ,Imię ,Nazwisko ,PESEL ,Zatrudniony ,Pensja)
VALUES
    (1, 'Tadek', 'Styczeń','76120511111','2014-01-05',8400)
INSERT INTO [Handlowcy]
    (lp ,Imię ,Nazwisko ,PESEL ,Pensja)
VALUES
    (2, 'Marek', 'Luty','36120511111', 5400)
INSERT INTO [Handlowcy]
    (lp ,Imię ,Nazwisko ,PESEL ,Zatrudniony ,Pensja)
VALUES
    (3, 'Robert', 'Marzec','86120511111','2014-06-05',2400)
INSERT INTO [Handlowcy]
    (lp ,Imię ,Nazwisko ,PESEL ,Zatrudniony ,Pensja)
VALUES
    (4, 'Witek', 'Kwiecień','88120511111','2015-08-05',1900)
```

7.6. Pętla WHILE

Pętle w programowaniu służą do zapętlenia, czyli wykonywania danego bloku kodu wiele razy. W SQL Server mamy do dyspozycji pętlę WHILE. Pętla WHILE jest wykonywana do momentu spełnienia określonego warunku. Poniższy przykład pokazuje zastosowanie pętli wyświetlającej kolejne liczby od 1 do 100.

```
PRINT 'Początek pętli'
PRINT CONVERT(NVARCHAR(50), GETDATE(), 121) -- 12 112
DECLARE @zmienna INT
SET @zmienna = 1
WHILE @zmienna <= 100
BEGIN
    PRINT @zmienna
    SET @zmienna = @zmienna +1
END
```

Materiały szkoleniowe

Podczas tworzenia pętli należy pamiętać, aby zapewnić punkt wyjścia z pętli po spełnieniu określonego warunku. W przeciwnym wypadku pętla będzie działała bez końca.

Pętla WHILE wykonuje się do momentu spełnienia określonego warunku. Dobrym przykładem użycia pętli jest dodanie do tabeli kolejnych rekordów. W tym przypadku będą to progi dla premii dla wynagrodzeń, które będą się zwiększały od 1000 zł o 3% co 200 zł i będzie ich 30.

Zaczniesz od stworzenia tabeli premia. Do wykonania zadania wykorzystasz kod:

```
CREATE TABLE Stawki (  
    id_stawki INT IDENTITY(1,1),  
    Przedział DECIMAL(12, 2),  
    Stawka DECIMAL(12, 4))
```

W kolejnym kroku dodasz pierwszy wiersz ze stawką bazową, która będzie punktem wyjścia do kolejnych progów i stawek procentowych.

```
INSERT premia (przedział, stawka)  
VALUES (500,1);
```

Ostatnim etapem będzie wypełnienie tabeli danymi zgodnie z założeniami – wzrost o 1% w stosunku do poprzedniego rekordu, a także z uwzględnieniem samych przedziałów premii.

```
-- Deklaracja zmiennych  
  
DECLARE @Pmax INT  
DECLARE @lPrzedział decimal (12,2)  
DECLARE @lStawka decimal (12,4)  
DECLARE @Krok INT  
  
-- przypisanie zmiennym wartości  
  
SET @lPrzedział = 1000  
SET @lStawka = 200  
-- dodanie pierwszego rekordu do Stawek  
INSERT Stawki (przedział, stawka) VALUES (@lPrzedział, @lStawka)  
SET @krok = 1  
  
-- początek pętli  
  
WHILE @krok <= 30 BEGIN  
    SET @Pmax = (SELECT TOP 1 id_stawki  
                FROM Stawki ORDER BY id_stawki DESC)  
    SET @lPrzedział = (SELECT przedział  
                      FROM Stawki WHERE id_stawki = @Pmax) + 200  
    SET @lStawka = (SELECT stawka  
                   FROM Stawki WHERE id_stawki = @Pmax) * 1.03  
    INSERT Stawki (przedział, stawka)  
                VALUES (@lPrzedział, @lStawka)  
    SET @krok = @krok + 1  
END
```

7.7. Funkcja warunkowe

W Microsoft SQL Server do wersji 2012 nie występowała funkcja warunkowa „jeżeli” znana z Excela. Zamiast tego konieczne było użycie warunku opartego o CASE.

W tym kroku otrzymałeś zadanie utworzenia tabeli zawierającej pensje swoich kierowców. Wynik musi być umieszczony w nowej tabeli.

W tym celu pobierzesz informacje z tabeli Kierowcy i w oparciu o te rekordy utworzysz nową tabelę korzystając z polecenia SELECT INTO. Założenia zadania to: wysokość pensji 20 zł za godzinę. Panie pracują 150 h miesięcznie, a panowie 180.

Rozstrzygnięcie, czy jest to pan czy pani wykonasz analizując ostatnią literę imienia, gdzie a oznacza panią, a każda inna litera pana.

```
SELECT *, CASE RIGHT(imię,1)
            WHEN 'a'
            THEN 150*20
            ELSE 180*20
        END
AS Pensja
FROM Kierowca
```

Funkcja warunkowe IIF

W wersji SQL Server 2012 i nowszej wprowadzono funkcję IIF znaną z Microsoft Access, która działa identycznie jak funkcja „JEŻELI”. W języku SQL istnieje także funkcja IF, która ma inne zastosowanie (np. IF EXISTS – testowane w poprzednim ćwiczeniu). Poniższy przykład działa identycznie jak ten z poprzedniego przykładu.

```
SELECT *, IIF( RIGHT(imię,1) = 'a' , 150 * 20, 180 * 20)
AS Pensja
FROM Kierowca
```

CASE vs IIF

Różnica pomiędzy dwoma rozwiązaniami jest taka, że CASE może zawierać więcej niż dwa możliwe odpowiedzi. Poniższy przykład ilustruje użycie trzech warunków.

```
SELECT *,
        CASE RIGHT(imię,1)
            WHEN 'a'
            THEN 120 * 20
            WHEN 'n'
            THEN 110 * 30
            ELSE 160 * 20
        END
AS Pensja
FROM Kierowca;
```

WARUNKI w KWERENDZIE

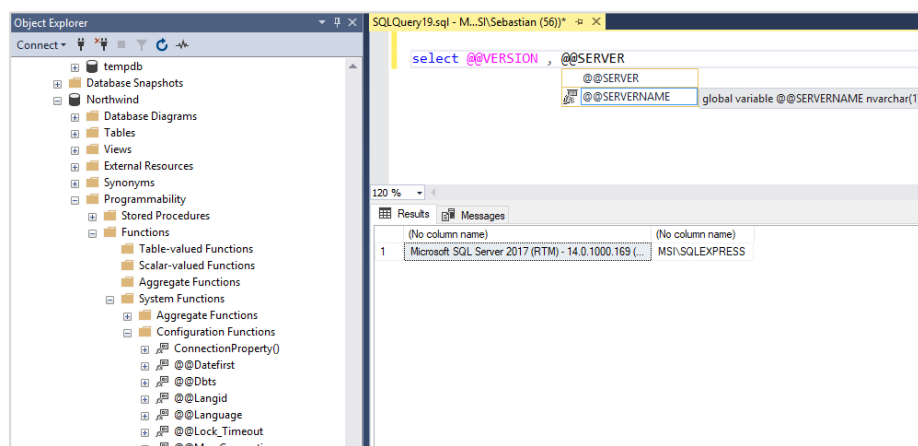
Przykład kwerendy, w której zastosowano funkcje warunkowe tworzące zawartość kolumny.

```
CREATE VIEW KierowcyZDodatkami
AS
SELECT id
      ,Imię
      ,Nazwisko
      ,[Data urodzenia]
      ,[Data zatrudnienia]
      ,Stawka
      ,Grupa,
      CASE Grupa
        WHEN 'GR I'
        THEN 0.05
        WHEN 'GR II'
        THEN 0.1
        ELSE 0.15
      END AS Dodatek,
      Stawka * (CASE Grupa
        WHEN 'GR I'
        THEN 0.05
        WHEN 'GR II'
        THEN 0.1
        ELSE 0.15
      END) + Stawka AS StawkaZDodatkiem -- Alias obliczonej kolumny nie może
      być użyty dalej
FROM [Kierowcy]
```

7.8. Zmienne

Podczas tworzenia kodu język SQL umożliwia używanie zmiennych, które mogą być używane w dalszych krokach programu. Zmienne mogą mieć zasięg globalny tj. obejmować cały serwer lub lokalny, gdzie ich zasięg ogranicza się do aktualnie przetwarzanego skryptu.

Zmienne lokalne oznaczane są @, a globalne @@.



Nazwy funkcji globalnych umieszczone są w funkcjach systemowych oraz podpowiadane podczas pisania kodu. Zmienne lokalne będą także widoczne podczas pisania kodu SQL – znak @.

Możesz przetestować dostęp do zmiennych globalnych za pomocą prostego polecenia

```
SELECT @@VERSION AS WersjaSerwera , @@SERVERNAME AS NazwaSerwera
```

Materiały szkoleniowe

Istnieje także możliwość wyświetlenia tych samych informacji podczas wykonywania skryptu za pomocą polecenia PRINT, które przekazuje informację do okna Messages, a nie jako wynik zapytania.

```
PRINT @@VERSION
PRINT @@SERVERNAME
```

Zmienne lokalne muszą być w pierwszej kolejności zadeklarowane za pomocą DECLARE. Równocześnie należy nadać im nazwę i typ przechowywanych danych. W kolejnych krokach nowej zmiennej powinna zostać przypisana wartość za pomocą polecenia SET.

Przykładem zastosowania zmiennej lokalnej może być:

```
DECLARE @zmienna INT
SET @zmienna = 1
SELECT @zmienna
```

W związku z tym, iż zmienne zwykle są wykorzystywane w procedurach koniecznym może się okazać zastosowanie bloku BEGIN/END. W takim przypadku powyższa instrukcja powinna zostać zmodyfikowana do postaci:

```
DECLARE @zmienna INT
SET @zmienna = 1
BEGIN
    SELECT @zmienna
END
```

7.9. Tabele tymczasowe

Tworząc skrypty SQL konieczne staje się tymczasowe przechowywanie informacji w bazie danych, ale niekoniecznie powinny być one widoczne w bazie roboczej. Służą do tego tabele tymczasowe, które umieszczane są w bazie tempdb jako Temporary Tables. Tabele tymczasowe mają nazwy rozpoczynające się od ##.

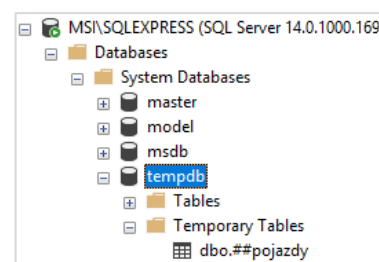


Tabela tymczasowa UŻYTKOWNIKA

Podczas przetwarzania danych możliwe jest tworzenie tabel tymczasowych, przeznaczonych na aktualnie obrabiane dane. Można je łatwo tworzyć, modyfikować, obrabiać w nich dane, a następnie usuwać wraz z zawartością.

```
CREATE TABLE #Apteki
(id INT,
Nazwa NVARCHAR(255),
Adres NVARCHAR(255))
```

Pobieranie danych z tabeli tymczasowej jest identyczne jak w przypadku zwykłej tabeli. Jedyną różnicą jest fizyczna lokalizacja tabeli. Nie znajduje się ona w bieżącej bazie danych użytkownika,

```
SELECT * FROM #Apteki
```

Usunięcie tabeli wraz z zawartością realizujemy przez polecenie DROP.

```
DROP TABLE #Apteki
```

Tabela tymczasowa DLA WSZYSTKICH

Oprócz tabel dostępnych dla bieżącego użytkownika, oznaczonych prefiksem # istnieją tabele tymczasowe, które są dostępne dla wszystkich użytkowników serwera. Takie tabele oznaczone są prefiksem ##.

Tworzenie tabel tymczasowych dla wszystkich użytkowników nie różni się od tabel zwykłych i realizowane jest za pomocą polecenia CREATE.

Poniższy przykład ilustruje tworzenie tabeli tymczasowej ##Apteki widocznej dla wszystkich użytkowników.

```
CREATE TABLE ##Apteki
(id INT,
Nazwa NVARCHAR(255),
Adres NVARCHAR(255))
```

Pobieranie danych z zastosowaniem SELECT.

```
SELECT * FROM ##Apteki
```

Usuwanie tabeli z użyciem polecenia DROP.

```
DROP TABLE ##Apteki
```

Przykład zastosowania tabel tymczasowej

Skrypt tworzący tabelę tymczasową, następnie dodający pojedynczy rekord, który zostaje odczytany poleceniem SELECT jest umieszczony poniżej. Ostatnim krokiem jest usunięcie tabeli wraz z zawartością.

```
-- tworzenie tabeli tymczasowej
CREATE TABLE ##pojazdy
(id INT, pojazd NVARCHAR(30))
-- dodawanie przykładowych rekordów
INSERT INTO ##pojazdy
(id, pojazd) SELECT 1, 'Rower'
-- pobieranie danych
SELECT *
FROM ##pojazdy
-- usuwanie tabeli
DROP TABLE ##pojazdy
```

Przykład skryptu pobierającego dane z tabeli Customers, na podstawie którego tworzona jest ogólnodostępna tabela ##Customers. Drugi krok aktualizuje dane tabeli, a trzeci dodaje brakującą kolumnę. Cała procedura jest identyczna jak w przypadku zwykłej tabeli.

```
SELECT *
INTO ##CustomersTMP
FROM Customers

UPDATE ##CustomersTMP
SET Region = ''
WHERE Region IS NULL
ALTER TABLE ##CustomersTMP
ADD Obroty DECIMAL(18,2)
```

Przykład skryptu tworzącego ogólnodostępna tabelę tymczasową w oparciu o istniejące dane. Podczas tej konkretnej operacji wykorzystywane są funkcje agregujące oraz łączenie tabel.

```
SELECT Klient, SUM(Wartość) AS Wartość
INTO ##ObrotyŁączne
FROM
(SELECT (UnitPrice * Quantity) AS Wartość, Orders.CustomerID AS Klient
FROM [Order Details] INNER JOIN
Orders ON [Order Details].OrderID = Orders.OrderID) AS X
GROUP BY Klient
```

W oparciu o utworzone uprzednio tabele wykonywana jest aktualizacja danych w tabeli ##Customers.

```
UPDATE ##CustomersTMP
SET Obroty = ##ObrotyŁączne.Wartość
FROM ##CustomersTMP INNER JOIN ##ObrotyŁączne
ON ##CustomersTMP.CustomerID = ##ObrotyŁączne.Klient
```

Ostatnim krokiem podczas przetwarzania danych jest wyświetlenie zawartości świeżo zmodyfikowanej tabeli. Równie dobrze po wykonaniu aktualizacji dane mogą być usunięte.

```
SELECT * FROM ##CustomersTMP
```

W związku z tym, iż table i zawarte w nich dane nie są potrzebne, dobrym pomysłem jest ich usunięcie.

```
DROP TABLE ##CustomersTMP
```

7.10. Tworzenie funkcji

SQL Server pozwala na tworzenie własnych funkcji. Odbyna się to za pomocą polecenia CREATE. Usuwanie i modyfikacja za pomocą odpowiednio DROP i ALTER.

Przykładem funkcji konwertującej datę do formatu tekstowego (możliwe jest użycie funkcji GETDATE() jako argumentu zwracającego bieżący czas.

Tworzona funkcja przyjmuje argument w postaci czasu – zmienna @czas. Następnie określony jest typ zwracany, w tym przypadku jest to NVARCHAR(10).

```
CREATE FUNCTION DataTekstem(@czas smalldatetime)
RETURNS NVARCHAR(10)
AS
BEGIN
DECLARE @RETURN as NVARCHAR(10)
SET @RETURN = convert(date, @czas)
RETURN @RETURN
END
```

Uruchomienie funkcji może wymagać podania schematu użytkownika (dbo) z uwagi na przypisanie funkcji jako funkcji użytkownika. Pierwszy test to zwykłe wyświetlenie przekonwertowanej daty za pomocą PRINT.

```
PRINT dbo.datatekstem('2018-02-02')
```

Drugim zastosowaniem funkcji własnej jest użycie jej w instrukcji SELECT.

```
SELECT dbo.DataTekstem(GETDATE())
```

Po utworzeniu funkcji w tym wypadku ją usunąć, korzystając z polecenia DROP.

```
DROP FUNCTION datatekstem
```

7.11. TRIGGER - wyzwalacz

Wyzwalacz (ang. trigger) – akcja wykonywana automatycznie jako reakcja na pewne zdarzenia w tabeli bazy danych. Mogą one ograniczać dostęp do pewnych danych, rejestrować zmiany danych lub nadzorować modyfikacje danych. Twoim celem jest stworzenie tabeli umożliwiającej rejestrację informacji o czasie dodania nowych elementów do tabeli pracownicy. W tym celu niezbędne jest utworzenie tabeli o nazwie **Rejestrator**, która będzie przechowywała zbierane informacje.

```
CREATE TABLE Rejestrator
(id_zdarzenia INT IDENTITY(1,1),
Czas datetime DEFAULT GETDATE(),
Akcja NVARCHAR (20) DEFAULT 'brak')
```

Drugim krokiem będzie utworzenie procedury ułatwiającej dopisywanie kolejnych rekordów do tabeli **Rejestrator**.

```
CREATE PROCEDURE rejestruj
@akcja NVARCHAR(20)
AS
BEGIN
INSERT INTO rejestrator (Akcja) VALUES (@akcja)
```


Materiały szkoleniowe

```
END
```

Możesz przetestować działanie procedury dodając trzy wpisy testowe.

```
EXEC rejestruj 'test1'
EXEC rejestruj 'test2'
EXEC rejestruj 'test3'
```

Ostatnim etapem zadania jest stworzenie samego wyzwalacza, który będzie w tym przypadku rejestrował w tabeli rejestrator informacje o dokonaniu kolejnego wpisu do tabeli **Kierowcy**. Będzie to **TRIGGER FOR INSERT**.

```
CREATE TRIGGER obserwuj
ON kierowca
FOR INSERT
AS
INSERT INTO rejestrator (Akcja)
VALUES ('nowy kierowca')
```

Chcąc przetestować działanie nowego wyzwalacza wykonaj polecenie INSERT INTO dla tabeli kierowca

```
INSERT INTO Kierowca (Imię, Nazwisko)
VALUES ('Marian', 'Kowalski')
```

Chcąc dokładniej przetestować działanie mechanizmu działania wyzwalaczy innego typu możesz użyć poniższego skryptu tworzącego dedykowaną bazę danych **WYZWALACZE**:

```
CREATE DATABASE Wyzwalacze
GO
USE Wyzwalacze
GO
```

Stworzenie tabel do pracy

```
CREATE TABLE Pracownicy
(id INT IDENTITY (1,1),
Imię NVARCHAR(50),
Nazwisko NVARCHAR(50))
GO
CREATE TABLE MyLog
(id INT IDENTITY (1,1),
Czas datetime DEFAULT Getdate(),
Opis NVARCHAR(50))
```

Dodanie procedury dodającej pracownika

```
CREATE PROCEDURE PracownikDodaj
@imie NVARCHAR(50),
@nazwisko NVARCHAR(50)
AS
INSERT INTO Pracownicy (Imię, Nazwisko)
VALUES (@imie, @nazwisko)
```

Procedura dodająca wpis do tabeli Log

```
CREATE PROCEDURE LogDodaj
@opis NVARCHAR(50)
AS
INSERT INTO MyLog (Opis)
VALUES (@opis)
```

Przetestuj dodawanie pracownika do bazy danych używając:

```
EXEC PracownikDodaj @imie = 'Jan', @nazwisko = 'Kowalski'
```

Dodaj wyzwalacz dla akcji po dodaniu rekordu **FOR INSERT**

```
CREATE TRIGGER ObserwatorIN
ON Pracownicy
FOR INSERT
AS
EXEC LogDodaj @opis = 'COŚ SIĘ DODAŁO'
```

Kolejny wyzwalacz, akcja po aktualizacji **FOR UPDATE**

```
CREATE TRIGGER ObserwatorUP
ON Pracownicy
FOR UPDATE
AS
EXEC LogDodaj @opis = 'COŚ SIĘ ZAKTUALIZOWAŁO'
```

Dodaj wyzwalacz dla akcji po kasowaniu **FOR DELETE**

```
CREATE TRIGGER ObserwatorDEL
ON Pracownicy
FOR DELETE
AS
EXEC LogDodaj @opis = 'COŚ SIĘ USUNĘŁO'
```

Wykonaj test dodawania nowego pracownika.

```
EXEC PracownikDodaj @imię = 'Tadeusz', @nazwisko = 'Nowak'
EXEC PracownikDodaj @imię = 'Maria', @nazwisko = 'Nowak'
```

Wykonaj test aktualizacji danych pracownika.

```
UPDATE Pracownicy SET Nazwisko = 'Kowalska'
WHERE Nazwisko = 'Nowak' AND Imię = 'Maria'
```

Dla ostatniego wyzwalacza wykonaj test kasowania rekordu.

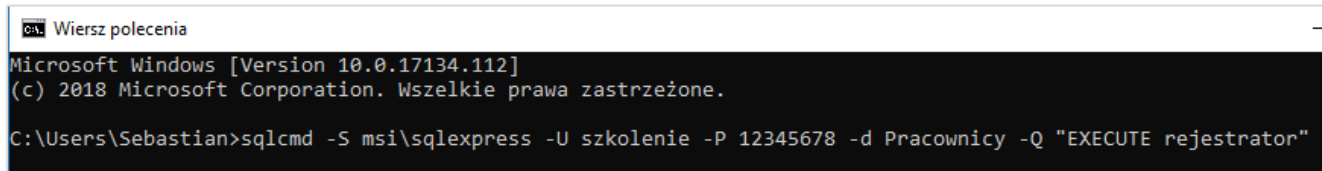
```
DELETE FROM Pracownicy WHERE Imię = 'Tadeusz'
```

Na koniec sprawdź co nowego się pojawiło w tabeli MyLog

```
SELECT * FROM MyLog
```

7.12. Zdalne uruchamianie procedur

Istnieje możliwość uruchamiania procedur składowanych za pomocą narzędzia **SQLCMD** z poziomu wiersza poleceń **CMD**. Rozwiązanie to pozwala na dodanie skryptów do harmonogramu programu Windows, co pozwala na ich cykliczne i automatyczne uruchamianie bez uruchamiania środowiska SQL Management Studio.



```
Wiersz polecenia
Microsoft Windows [Version 10.0.17134.112]
(c) 2018 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\Sebastian>sqlcmd -S msi\sqlexpress -U szkolenie -P 12345678 -d Pracownicy -Q "EXECUTE rejestrator"
```

7.13. Kursory SQL

Pracując z kodem Transact-SQL spotykamy zadania, których nie da się rozwiązać stosując pojedyncze instrukcje SQL. Jednym z takich problemów jest praca rekord po rekordzie, gdzie każda z operacji musi się odnosić do różnych tabel i wykonywać różne niekoniecznie identyczne i powtarzalne operacje. Dodatkowo może być konieczne sięganie do rekordu poprzedniego, porównywanie wartości między bieżącym a poprzednim itp.

Materiały szkoleniowe

Kursor to bufor, który pozwala na przechowanie poprzez zapisanie do niego kolejnych rekordów bazy danych. Można go stosować w procedurach, wyzwalaczach i wszędzie tam gdzie konieczna może być praca w trybie krokowym, rekord po rekordzie. Z uwagi na ich małą wydajność należy dokładnie przemyśleć ich stosowanie, a w sytuacji, gdy istnieje rozwiązanie alternatywne, raczej skorzystać z niego.

Składnia kursora

Chcąc użyć kursora należy wykonać ściśle określone kroki. Schemat tych kroków nie jest szczególnie skomplikowany, ale powtarzalny i czytelny.

Deklaracja kursora - pierwszym krokiem jest deklaracja kursora, która jest zawsze wykonywana dla instrukcji **SELECT**. Wynik tej instrukcji będzie wykonywany rekord po rekordzie.

```
DECLARE nazwa_kursora CURSOR
FOR instrukcja_SELECT
```

Krok drugi to otwarcie kursora

```
OPEN nazwa_kursora
```

Kolejny krok to pobranie danych kolejnego wiersza. Służy do tego instrukcja **FETCH**. Podczas pobierania danych z pojedynczego rekordu konieczne jest zadeklarowanie tylu zmiennych i o takich typach danych jakie zwraca instrukcja **SELECT**.

```
FETCH NEXT FROM nazwa_kursora INTO zmienne
```

Z kursora korzystamy zwykle w pętli, co pozwala na przejście po wszystkich kolejnych rekordach. Aby sprawdzić, czy ostatnie wywołanie **FETCH** zwróciło wiersz stosujemy zmienną systemową **@@FETCH_STATUS**, która zwracając wartość 0 oznacza, że pobranie wiersza odbyło się z powodzeniem. W przeciwnym przypadku oznacza wystąpienie błędu, lub osiągnięcie ostatniego rekordu.

```
WHILE @@FETCH_STATUS = 0
```

Kończąc pracę z kursorem należy go zamknąć, co go zamyka i zwalnia utworzone przez niego blokady rekordów na których działał. Jest on w tym momencie nadal dostępny i możliwy do użycia.

```
CLOSE nazwa_kursora
```

Instrukcja **DEALLOCATE** powoduje jego całkowite usunięcie z pamięci.

```
DEALLOCATE nazwa_kursora
```

Kursor i suma krocząca

Jednym z przykładów zastosowania kursora może być uzyskanie sumy kroczącej, która będzie zliczała kolejne wartości **Quantity**, narastająco, posuwając się rekord po rekordzie wg kolejnych numerów zamówienia.

```
DECLARE kursor CURSOR FOR
    SELECT OrderID, Quantity FROM [Order Details]
    ORDER BY OrderID, ProductID
DECLARE @Qk INT, @orderid INT, @quantity smallint
OPEN kursor
    FETCH FROM kursor INTO @orderid, @quantity -- pierwszy rekord
    SET @Qk = @quantity -- pierwsze quantity
    WHILE @@FETCH_STATUS = 0 -- pętla
    BEGIN
        FETCH FROM kursor INTO @orderid, @quantity -- kolejny rekord
        SET @Qk = @Qk + @quantity -- zliczam sumę quantity
        PRINT CONCAT(@orderid, ' ', @Qk) -- wyświetlam sumę
    END
    CLOSE kursor
DEALLOCATE kursor
```

Przykładowy kursor rozbudowany o dodatkowe pola kursor będzie miał następującą postać:

```
DECLARE kursor CURSOR FOR
    SELECT OrderID, Freight FROM Orders ORDER BY OrderID
    DECLARE @orderID INT, @freight decimal(18,4),
    @freightP decimal(18,4), @sumaFreight decimal(18,4)
    OPEN kursor
    FETCH NEXT FROM kursor INTO @orderID, @freight
    SET @sumaFreight = @freight
    WHILE @@FETCH_STATUS = 0
        BEGIN
            FETCH NEXT FROM kursor INTO @orderID, @freight
            SET @sumaFreight = @sumaFreight + @freight
            PRINT CONCAT('Zamówienie: ', @orderID, ' narastajaco: ', @sumaFreight)
        END
    CLOSE kursor
    DEALLOCATE kursor
```

Kursor i praca wiersz po wierszu

Jedną z charakterystycznych cech tabel bazy danych jest, to że dane w nich nie są układane w kolejności dodawania. Co oznacza, że użycie **SELECT** bez **ORDER BY** nie gwarantuje ich kolejności zgodnej z kolejnością dodawania. Dlatego też nie ma możliwości w zwykłych warunkach pobierania wartości z wiersza poprzedniego, czy następnego. Rozwiązaniem tego problemu może być kursor. Chcąc porównać identyfikator klienta bieżącego z identyfikatorem klienta poprzedniego możesz użyć konstrukcji:

```
DECLARE kursor CURSOR FOR
    SELECT CustomerID, CompanyName FROM Customers
    ORDER BY CustomerID
    -- deklaracja zmiennych
    DECLARE @customerID char(5), @companyName NVARCHAR(50), @oldID char(5)
    -- otwarcie kursora
    OPEN kursor
    -- pobranie pierwszego rekordu i załadowanie danych do zmiennych
    FETCH NEXT FROM kursor INTO @customerID, @companyName
    -- początek pętli poruszającej się po rekordach
    WHILE @@FETCH_STATUS = 0
        BEGIN
            -- PRINT @customerID
            SET @oldID = @customerID
            FETCH NEXT FROM kursor INTO @customerID, @companyName
            -- wyświetlenie porównania
            PRINT @customerID + ' ' + @oldID
        END
    -- zamknięcie i dealokacja kursora
    CLOSE kursor
    DEALLOCATE kursor
```

Kursor SCROLL

Poprzedni kursor służył do poruszania się wyłącznie do przodu, a co za tym idzie był najszybszy z dostępnych. Oprócz tego typu kursorów dostępne są także kursory **SCROLL** poruszające się i do góry i na dół w dowolnej konfiguracji. W poniższym przykładzie jest pobierany i wyświetlany wyłącznie id kontrahenta.

```
DECLARE firma SCROLL CURSOR FOR
    SELECT CustomerId FROM Customers
        ORDER BY CustomerId
DECLARE @cID char(5)
OPEN firma
-- ostatni
    FETCH LAST FROM firma INTO @cID
    PRINT @cID
-- poprzedni
    FETCH PRIOR FROM firma INTO @cID
    PRINT @cID
-- pozycja 2 bezwzględnie
    FETCH ABSOLUTE 2 FROM firma INTO @cID
    PRINT @cID
-- 3 w dół względnie
    FETCH RELATIVE 3 FROM firma INTO @cID
    PRINT @cID
-- 2 w górę względnie
    FETCH RELATIVE -2 FROM firma INTO @cID
    PRINT @cID
CLOSE firma
DEALLOCATE firma
```

8. Zaawansowane procedury SQL

Całość prac związanych z bazą danych może zostać zautomatyzowana za pomocą skryptu języka SQL. Poczynając od stworzenia bazy danych poprzez wygenerowanie jej obiektów, po pobranie i konwersję danych.

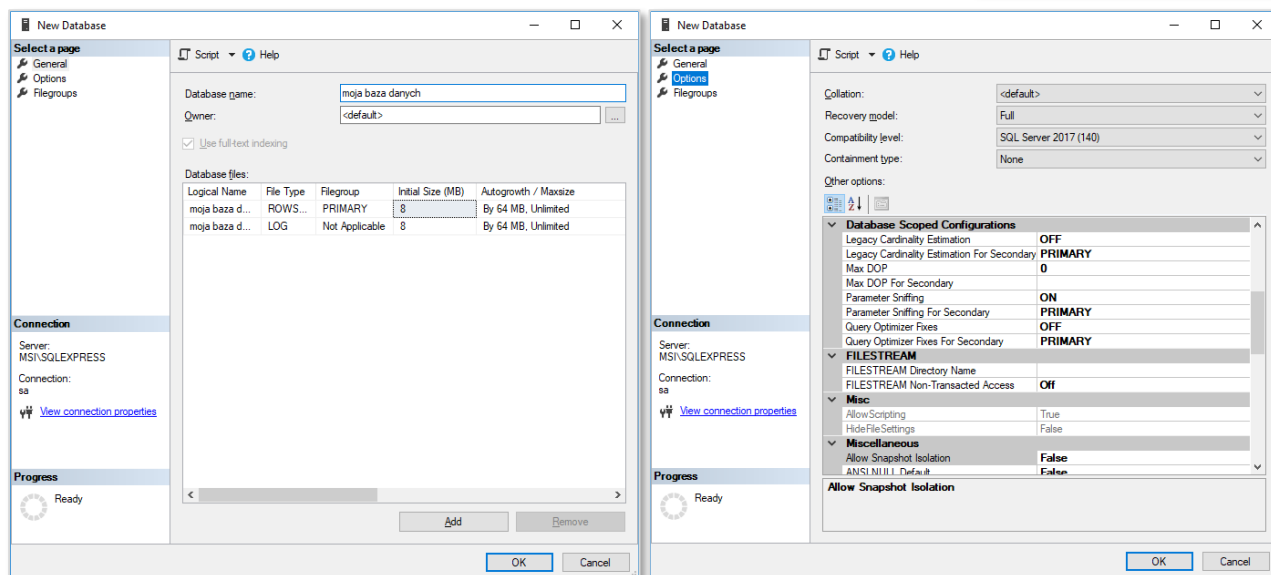
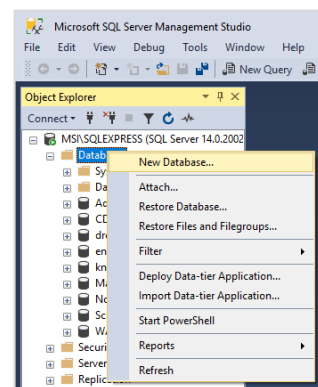
8.1. Tworzenie bazy danych

Baza danych SQL Server pozwala na tworzenie baz danych w dwóch trybach. Graficznym z zastosowaniem SQL Management Studio oraz tekstowym za pomocą języka SQL.

Tryb graficzny

Opcja pierwsza sprowadza się do kliknięcia prawym klawiszem myszy opcji New Database.

W kolejnym oknie możesz nadać jej unikalną w ramach serwera nazwę oraz wskazać właściciela, czyli jej administratora. Dodatkowo w kolejnych oknach możliwe jest ustawienie dodatkowych parametrów, takich jak: poziom kompatybilności silnika bazy danych, wersję językową – w tym zasady sortowania.



Tryb tekstowy

Podstawowym zadaniem trybu tekstowego jest automatyzacja tworzenia obiektów bazy danych i maksymalne odciążenie operatora bazy. W związku z tym twórca skryptu powinien zapewnić jego działanie w każdych warunkach, a co za tym idzie, wszelkie zadania powinny być realizowane automatycznie.

Poleceniem służącym do tworzenia bazy danych jest CREATE.

```
CREATE DATABASE MojaBazaDanych
```

Z kolei poleceniem usuwającym bazę danych będzie DROP.

```
DROP DATABASE MojaBazaDanych
```

Tworząc bazę danych w pierwszej kolejności należy zadbać o odpowiedni kontekst bieżącej bazy danych. W tym momencie konieczne jest przejście do bazy Master.

```
USE master
GO
```

Materiały szkoleniowe

Skrypt działający w trybie autonomicznym powinien wykonać się automatycznie. W przypadku błędu podczas jego wykonania musi on wykonać określoną akcję, bez ryzyka jego zatrzymania lub wywołania błędu, który spowoduje serię kolejnych błędów i nieodwracalne zniszczenia. Dobrym pomysłem jest na początek sprawdzenie, czy istnieje już baza o wskazanej nazwie. Jeśli tak możesz zakończyć skrypt lub usunąć starą bazę i stworzyć nową o tej samej nazwie. Równocześnie należy odłączyć od bazy ewentualnych zalogowanych użytkowników. Pamiętaj, że jednym z nich jesteś Ty. Stąd konieczność przejścia do bazy master. Sprawdzenie istnienia bazy danych można wykonać szukając jej nazwy w tabeli systemowej sys.databases. Warunkowe stworzenie bazy, wraz z odłączeniem użytkowników będzie miało postać:

```
IF EXISTS(SELECT * FROM sys.databases WHERE name = 'MojaBazaDanych')
BEGIN
    ALTER DATABASE Praca SET SINGLE_USER WITH ROLLBACK IMMEDIATE
    DROP DATABASE Praca
END
CREATE DATABASE Praca
GO
USE Praca
GO
```

W dalszym ciągu skryptu możesz (tu nie jest to wymagane, ponieważ baza jest świeżo utworzona i pusta) sprawdzić istnienie nowotworzonych tabel. Tworzysz tabelę Pracownik.

```
IF EXISTS(SELECT * FROM sys.tables WHERE name = 'Pracownik')
BEGIN
    DROP TABLE Pracownik
END
CREATE TABLE Pracownik
(
    id INT IDENTITY (1,1) NOT NULL,
    Aktywny bit DEFAULT 1 NOT NULL,
    Imię NVARCHAR(50) NOT NULL,
    Nazwisko NVARCHAR(50) NOT NULL,
    PESEL char(11) UNIQUE, -- UNIKALNOŚĆ pola w ramach tabeli
    Miejscowość INT DEFAULT 1,
    DataZatrudnienia date DEFAULT getdate(), -- wartość domyślna bieżący czas
    Stanowisko INT DEFAULT 1)
GO
```

Dyrektywa **GO** jest niezbędna, ponieważ polecenie **CREATE** może być jedynym w skrypcie, a **GO** rozdziela go na mniejsze samodzielne fragmenty. Kolejne tabele do stworzenia to tabela **Stanowiska** oraz **Miejscowości**.

```
IF EXISTS(SELECT * FROM sys.tables WHERE name = 'Stanowiska')
BEGIN
    DROP TABLE Stanowiska
END
CREATE TABLE Stanowiska
(
    id INT IDENTITY (1,1),
    Stanowisko NVARCHAR(16))
GO
IF EXISTS(SELECT * FROM sys.tables WHERE name = 'Miejscowości')
BEGIN
    DROP TABLE Miejscowości
END
CREATE TABLE Miejscowości -- słownik na serwerze
(
    id INT NOT NULL,
    Miejscowość NVARCHAR(50))
GO
```

Podczas tworzenia kolejnych obiektów bazy danych może się zdarzyć, że zapomnisz lub nie będziesz miał możliwości stworzenia pewnych elementów, czy to tabeli, czy procedury. Na szczęście istnieje możliwość zmodyfikowania dowolnego obiektu w bazie danych i dodanie lub modyfikacja poszczególnych elementów. Pomocne w tym celu będzie zastosowanie polecenia ALTER. Zwykle takie elementy tabeli jak sprawdzanie poprawności są dodawane po stworzeniu tabeli i niekiedy po zaimportowaniu danych. W tym wypadku chciałbyś, aby w tabeli Pracownik

Materiały szkoleniowe

DataZatrudnienia była mniejsza lub równa dacie bieżącej, a numer PESEL był nie tylko niepowtarzalny, co już zostało zdefiniowane podczas tworzenia obiektu, ale także jego składniki mogły być wyłącznie liczbami (0-9). Polecenie modyfikujące będzie miało postać:

```
ALTER TABLE Pracownik
ADD CONSTRAINT CK_DataZatrudnienia CHECK (DataZatrudnienia <= GETDATE())
GO
ALTER TABLE Pracownik
ADD CONSTRAINT CK_Pesel CHECK (PESEL like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
```

Dodatkowo na polu id, które ma zdefiniowaną autonumerację, brakuje klucza głównego. Sama autonumeracja nie gwarantuje unikatowości numeru kolejnego, ponieważ istnieje możliwość jej wyłączenia i dopisania numerów ręcznie. Tworzenie klucza głównego (**PRIMARY KEY**) odbywa się poprzez polecenie **ALTER**. Klucze główne musisz w tym przypadku utworzyć dla tabel **Pracownik**, **Miejscowości** i **Stanowiska**. Pole klucza głównego nie musi być autonumerowane, ani nie ma wymogu, by było liczbą.

```
ALTER TABLE Pracownik
ADD CONSTRAINT PK_Pracownik PRIMARY KEY (id)
GO
ALTER TABLE Miejscowości
ADD CONSTRAINT PK_Miejscowości PRIMARY KEY (id)
GO
ALTER TABLE Stanowiska
ADD CONSTRAINT PK_Stanowiska PRIMARY KEY (id)
GO
```

Mając stworzony zestaw tabel dobrym pomysłem jest połączenie ich za pomocą relacji, co pozwoli na zapewnienie składowanym w nich danym spójności i poprawności. Dodanie relacji możliwe jest dla pól o tych samych typach danych (liczba całkowita – liczba całkowita i tekst – tekst). W przypadku różnych typów danych w obu łączonych kolumnach, stworzenie relacji będzie niemożliwe. W tym przypadku należy połączyć tabele **Pracownik** i **Miejscowości** – pola **pracownik.Miejscowość = Miejscowości.id** oraz **Pracownik** i **Stanowiska** – pola **Pracownik.Stanowisko = Stanowiska.id**. Obie relacje powinny blokować kaskadowe usuwanie i aktualizację. Zapobiegnie to przypadkowemu usunięciu lub aktualizacji danych z tabel podrzędnych.

```
ALTER TABLE Pracownik
ADD CONSTRAINT FK_Pracownik_Miejscowości FOREIGN KEY
(Miejscowość) REFERENCES Miejscowości (id)
ON UPDATE NO ACTION
ON DELETE NO ACTION
GO
ALTER TABLE Pracownik
ADD CONSTRAINT FK_Pracownik_Stanowiska FOREIGN KEY
(Stanowisko) REFERENCES Stanowiska (id)
ON UPDATE NO ACTION
ON DELETE NO ACTION
GO
```

Utworzone tabele bazy danych oraz reguły sprawdzania poprawności możesz wyświetlić z bazy danych używając polecenia **SELECT** i tabel z prefiksem **sys**. Np.:

```
SELECT * FROM sys.check_constraints
SELECT * FROM sys.key_constraints
SELECT * FROM sys.tables
```

Przed dodaniem danych do nowych tabel dobrym pomysłem jest wykonanie kopii bazy danych, co w razie niepowodzenia w dalszych krokach, pozwoli na sprawne i szybkie odzyskanie danych sprzed wykonania poprzednich kroków. Polecenie **BACKUP DATABASE** – powoduje wykonanie kopii do domyślnej lokalizacji kopii bazy danych.

```
BACKUP DATABASE Praca TO DISK = 'kopia_praca.bak'
GO
```


Materiały szkoleniowe

Po przygotowaniu bazy danych, tabel, relacji i wykonaniu kopii możliwe staje się przystąpienie do dodawania danych. Pierwszą tabelą, która ma zostać uzupełniona będzie tabela Stanowiska. Pewnym utrudnieniem będzie fakt, iż numery id stanowisk muszą być narzucone z góry. Nie mogą one pochodzić z autonumeracji. Można to osiągnąć wyłączając na chwilę **IDENTITY_INSERT**, a później uaktywniając go ponownie.

```
SET IDENTITY_INSERT Stanowiska ON
INSERT INTO Stanowiska (id, Stanowisko) VALUES (1,'prezes') -- wymuszenie
wstawienia nr id
INSERT INTO Stanowiska (id, Stanowisko) VALUES (2,'dyrektor')
INSERT INTO Stanowiska (id, Stanowisko) VALUES (3,'kierownik')
INSERT INTO Stanowiska (id, Stanowisko) VALUES (4,'specjalista')
INSERT INTO Stanowiska (id, Stanowisko) VALUES (5,'konserwator')
INSERT INTO Stanowiska (id, Stanowisko) VALUES (6,'serwisant')
INSERT INTO Stanowiska (id, Stanowisko) VALUES (7,'brygadzysta')
SET IDENTITY_INSERT Stanowiska OFF
GO
```

Tabela Miejscowości nie ma zdefiniowanego autonumeru na polu z kluczem głównym, więc możesz bez problemu dopisywać do niej dane. W przypadku danych znajdujących się na serwerze zdalnym możesz je pobrać poprzez LinkedServer. W tym przypadku jego przykładowa nazwa to: [\[zdalny.serwer.pl,51433\].\[baza\].\[miejscowość\]](#). Cała operacja sprowadza się do zastosowania polecenia **INSERT INTO**.

```
INSERT INTO Miejscowości
(id, Miejscowość )
SELECT id, Miejscowość
FROM [zdalny.serwer.pl,51433].[baza].[Miejscowości]
```

Druga tabela posiada autonumerację, więc procedura wymaga zastosowania **IDENTITY_INSERT ON**. Dodatkowo w przypadku tej tabeli dane zostaną pobrane z serwera MySQL [\[zdalny_my.serwer.pl,51433\]](#) z zastosowaniem LinkedServer i połączone z danym z serwera [\[zdalny.serwer.pl,51433\]](#).

```
SET IDENTITY_INSERT Pracownik ON
INSERT INTO Pracownik
(id, Aktywny, Imię, Nazwisko, PESEL, Miejscowość, Stanowisko)
SELECT id, 1 AS Aktywny, Imię, Nazwisko, PESEL, Miejscowość, Stanowisko
FROM
(SELECT *
FROM [zdalny.serwer.pl,51433].[DanePracowników]
UNION
SELECT *
FROM OPENQUERY ( [mój mysql], 'SELECT * FROM DanePracowników')) AS X
SET IDENTITY_INSERT Pracownik OFF
```

8.2. Nazwa tabeli ze zmiennej

Korzystając z języka T-SQL możliwe stają się czynności, które w normalnych warunkach są niedostępne. Jednym z takich przykładów jest tworzenie obiektu z nazwy będącej zmienną. Poniższy przykład ilustruje procedurę tworzącą tabelą o nazwie generowanej na podstawie bieżącego miesiąca. Procedura jako taka nie tworzy tabeli, a jedynie generuje kod SQL niezbędny do realizacji tego zadania.

```
-- nazwa obiektu ze zmiennej
DECLARE @SQL NVARCHAR(MAX)
DECLARE @nazwa NVARCHAR(8)
-- przypisanie zmiennym wartości
SET @nazwa = CONVERT(NVARCHAR(23), getdate(),112)
SET @SQL = 'CREATE TABLE Sprzedaż' + @nazwa + '
(id INT,
pole NVARCHAR(50))'
EXEC (@SQL)
PRINT @SQL
-- kasowanie danych z poprzedniego miesiąca
SET @nazwa = CONVERT(NVARCHAR(23), DATEADD(DAY, -1, getdate()),112)
SET @SQL = 'DROP TABLE Sprzedaż'+@nazwa
PRINT @SQL
EXEC (@SQL)
```

8.3. Seryjne tworzenie tabel

Korzystając z możliwości T-SQL możliwe staje się także generowanie tabel seryjnie, w oparciu o pętlę WHILE. W tym zadaniu konieczne jest stworzenie skryptu tworzącego tabelę dla każdego miesiąca jaki istnieje w tabeli Orders.

```
-- deklarujemy zmienne
DECLARE      @SQL NVARCHAR(max),
             @StartDate char(10),
             @krok INT
-- przypisujemy wartości zmiennym
SET @StartDate = (SELECT MIN(Convert(char(10), orderdate, 112)) FROM ORDERS)
SET @Krok = DATEDIFF(month, (SELECT MIN(OrderDate) FROM ORDERS), (SELECT
MAX(OrderDate) FROM ORDERS))

-- pętla tworząca tabele
WHILE @krok >= 0
BEGIN

-- sprawdzamy czy tworzona tabela istnieje, jeśli tak to ją usuwamy
SET @SQL = 'IF EXISTS(SELECT * FROM sys.tables WHERE name like ''Orders' +
CONVERT(char(6), DATEADD(Month, @krok, @StartDate),112) + '')
DROP TABLE Orders' + CONVERT(char(6), DATEADD(Month, @krok, @StartDate),112)

-- najpierw PRINT(SQL) w celu zweryfikowania poprawności skryptu
-- następnie EXEC(SQL) w celu jego wykonania
EXEC (@SQL)
-- przetwarzamy ciąg dalszy skryptu

SET @SQL = 'SELECT * INTO Orders' +
CONVERT(char(6), DATEADD(Month, @krok, @StartDate),112) +
' FROM Orders WHERE CONVERT(char(6), OrderDate, 112) = ' +
'Convert(char(6), DateAdd(month, ' + Convert(char(2), @Krok) +
', '' + @StartDate + '''),112) '
-- najpierw PRINT, później EXEC
PRINT (@SQL)
-- dodajemy 1 to bieżącego kroku
SET @krok = @krok - 1
-- kończymy pętlę i wracamy do jej początku
```

8.4. Procedura dodająca wpis do tabeli log

Procedura dodająca wpis do tabeli ObrotyLog

Przetwarzając skrypt warto rejestrować skutki jego działania. Najlepszą metodą jest zapis poszczególnych kroków do tabeli. W poniższym przypadku stworzona jest uniwersalna procedura dodająca informacje o kolejnych etapach przetwarzania kodu.

```
CREATE PROCEDURE ObrotyLog
    @uzytkownik NVARCHAR(20),
    @operacja NVARCHAR(16),
    @uwagi NVARCHAR(255)
AS
INSERT INTO [MSSQL01.DCSWEB.PL,51433].[1449_isa].[1449_kurs].[my_log]
    ([Uzytkownik], [Operacja], [Uwagi])
VALUES (@uzytkownik, @operacja, @uwagi)
```

8.5. Obsługa i przechwytywanie błędów

Podczas wykonywania skryptu mogą nastąpić niespodziewane zdarzenia i dobrym pomysłem byłoby ich obsłużenie, co jest zdecydowanie rozsądniejsze niż awaryjne zakończenie skryptu z błędem. Najprostszym sposobem wywołania błędu może być dzielenie przez 0.

```
PRINT 5/0
```

Co spowoduje wyświetlenie błędu i zakończenie działania skryptu

```
Msg 8134, Level 16, State 1, Line 1  
Divide by zero error encountered.
```

Blok TRY/CATCH

Chcąc zapobiec takiemu zdarzeniu można zastosować blok **TRY/CATCH**, który pozwala po wykonaniu instrukcji wykonać akcję przewidzianą przez programistę. Poniższy przykład pozwala po wystąpieniu błędu na wyświetlenie jego komunikatu i podążenie dalej.

```
BEGIN TRY  
    PRINT 5/0  
END TRY  
BEGIN CATCH  
    PRINT ERROR_MESSAGE()  
END CATCH  
PRINT 'coś co występuje dalej'
```

Oprócz zwykłego komunikatu o błędzie możliwe jest pozyskanie dodatkowych informacji o nim.

```
BEGIN TRY  
    PRINT 5/0  
END TRY  
BEGIN CATCH  
    PRINT ERROR_MESSAGE() + ' komunikat błędu'  
    PRINT CONCAT(ERROR_NUMBER(), ' nr błędu')  
    PRINT CONCAT(ERROR_SEVERITY(), ' stan błędu')  
    PRINT ERROR_LINE()          -- nr linii kodu z błędem  
    PRINT ERROR_PROCEDURE()     -- procedura błędu  
END CATCH
```

RAISERROR

Istnieją różne metody przechwycenia błędu skryptu. W odpowiedniej obsłudze pomaga RAISERROR, który to obsługuje stan i poziom błędu tutaj 16 wyrzuca błąd w konsoli (czerwony nr 11-19)

```
BEGIN TRY  
    DECLARE @Int TINYINT;  
    SET @Int = 999  
END TRY  
BEGIN CATCH  
    DECLARE @ErrorMessage NVARCHAR(MAX);  
    SET @ErrorMessage = ERROR_MESSAGE();  
    RAISERROR (@ErrorMessage, 16, 1);  
END CATCH;
```

Tylko błędy na poziomie 11-19 są przechwytywane w bloku TRY ,a pozostałe kończą dane zapytanie.

```
BEGIN TRY
    DECLARE @Int TINYINT;
    SET @Int = 999
END TRY
BEGIN CATCH
    DECLARE @ErrorMessage NVARCHAR(MAX);
    SET @ErrorMessage = ERROR_MESSAGE();
    RAISERROR (@ErrorMessage, 1, 1);
END CATCH;
```

8.6. Sterowanie przebiegiem programu

SET NOCOUNT ON

Wyłączenie komunikatów zliczających wyświetlone/przetworzone rekordy. Możesz to sprawdzić wykonując na bazie Northwind polecenia

```
PRINT 'Pierwsze polecenie'
SET NOCOUNT OFF
    select * from orders
PRINT 'Drugie polecenie'
SET NOCOUNT ON
    select * from orders
```

Dla pierwszego polecenia otrzymasz w okienku **Messages** wynik dla drugiego już nie

```
Pierwsze polecenie
(830 rows affected)
Drugie polecenie
```

WAITFOR – wstrzymanie skryptu

Decydując się na uruchomienie skryptu możesz wstrzymać jego wykonanie. Istnieją dwa sposoby zatrzymania jego przetwarzania. Jest to bardzo przydatne jeśli chcemy, aby wykonane poprzednio polecenia miały czas się wykonać (np. kopia zapasowa) lub zależy nam na uruchomieniu poszczególnych części skryptu w okresie mniejszego obciążenia bazy danych.

Pierwszy sposób to wstrzymanie skryptu do wskazanej godziny:

```
WAITFOR TIME '22:30'
PRINT '!'
```

Drugi pozwala na wstrzymanie go na wskazany okres czasu.

```
WAITFOR DELAY '0:02'
PRINT '!'
```

Instrukcja GOTO

Podczas wykonywania skryptu, często istnieje konieczność warunkowego przejścia do innego jego fragmentu lub do przeskoczenia jego fragmentów. Jest to szczególnie przydatne podczas wdrażania procedur obsługi błędów. Do skoku wykorzystywana jest instrukcja **GOTO** wskazująca oznaczoną etykietę. Etykietą może być dowolna nazwa zakończona znakiem dwukropka. Poniższy przykład ilustruje skok do kolejnej instrukcji po wystąpieniu błędu, pomijane są niektóre instrukcje.

```
BEGIN TRY
    PRINT 5/0
END TRY
BEGIN CATCH
    PRINT ERROR_MESSAGE()
    -- skok do etykiety
    GOTO koniec
END CATCH
PRINT 'ten wpis się wyświetli jeśli nie będzie błędu'
-- etykieta
koniec:
```

8.7. Transakcje

Transakcja to sekwencja operacji wykonywanych jako pojedyncza jednostka pracy. Każda taka transakcja ma cztery kluczowe wartości, które są określone skrótem **ACID** (Atomic Consistent Isolated Durability).

- **ATOMIC** - cała operacja jest traktowana jako pojedyncza jednostka. Wykonywane jest wszystko albo nic.
- **CONSISTENT** - zakończona transakcja pozostawi bazę danych w spójnym stanie wewnętrznym.
- **ISOLATED** - widzi bazę danych w spójnym, dlatego jeśli dwie transakcje próbują zaktualizować tę samą tabelę najpierw zostanie wykonana pierwsza z nich a potem druga.
- **DURABILITY** – wynik transakcji jest stale przechowywany w systemie.

Transakcja AUTOCOMMIT

Najprostszą transakcją w SQL Server może być pojedyncza instrukcja modyfikacji danych.

```
UPDATE Customers
SET City = 'London'
WHERE City = 'Londyn'
```

SQL Server zapisuje do loga akcję. Następnie dokonuje aktualizacji danych oraz zapisuje do loga informację o przeprowadzonej operacji. Dane są bezpośrednio na dysku ale sama aktualizacja wykonuje się na danych znajdujących się w pamięci. Kolejnym krokiem jest zapis tych danych na dysku. Jeżeli pojawi się błąd po wykonaniu transakcji i zapisaniu informacji w dzienniku, SQL Server wykorzysta tę informację do przeprowadzenia operacji ROLL FORWARD na przeprowadzonej transakcji podczas jej kolejnego uruchomienia.

Transakcje Explicit Transactions

Polecenie **BEGIN TRANSACTION** rozpoczyna, a **COMMIT TRANSACTION** kończy transakcję. Każda z instrukcji zawarta bloku traktowana jest jako logiczna jednostka pracy. Jeżeli po pierwszej instrukcji pojawi się błąd nie dojdzie do wykonania polecenia **COMMIT TRANSACTION** co oznacza, że ani pierwsza ani druga instrukcja nie zostaną wykonane.

```
BEGIN TRANSACTION
UPDATE Customers
SET City = 'London'
WHERE City = 'Londyn'
UPDATE Customers
SET City = 'New York'
WHERE City = 'Nowy Jork'
COMMIT TRANSACTION
```

Cofanie transakcji ROLL BACK

Istnieje także możliwość samodzielnego cofnięcia transakcji, jeśli nie spełni ona oczekiwań. W poniższym przykładzie operacja zostanie zatwierdzona tylko, jeżeli liczba zaktualizowanych rekordów będzie równa dwóm.

```
BEGIN TRANSACTION
UPDATE Customers
SET City = 'London'
WHERE City = 'Londyn'
IF @@ROWCOUNT = 2
COMMIT TRANSACTION
ELSE
ROLLBACK TRANSACTION
```

Transakcje w procedurach składowanych

W większości przypadków transakcji używa się wewnątrz procedur składowanych. Poniższa procedura wykonuje się wyłącznie jeśli zadziała dodanie kontrahenta i aktualizacja jego danych. Przy kolejnej próbie transakcja nie zadziała, z uwagi na zduplikowany CustomerID, pojawi się błąd integralności danych.

```
CREATE PROCEDURE TransactionX
AS
BEGIN TRANSACTION
    INSERT INTO Customers (CustomerID, CompanyName, ContactName)
    VALUES ('ZBCVT', 'Firma X', 'Arnold S')
UPDATE Customers
    SET City='Berlin'
    WHERE CustomerID = 'ZBVCT'
COMMIT TRANSACTION
GO
```

Jeśli procedura ma działać, mimo błędu kod wymaga pewnych modyfikacji. W tym wypadku sprawdzamy poprawność wykonania każdej z dwóch instrukcji. Jeżeli instrukcja się nie powiedzie (@ERROR <> 0) cofamy zmiany oraz używamy instrukcji RETURN do zaprzestania wykonywania procedury składowanej. Ważne jest sprawdzenie błędu po każdym wykonaniu bloku instrukcji, ponieważ istnieje ryzyko niepoprawnego wykonania transakcji.

```
CREATE PROCEDURE TransactionY
AS
BEGIN TRANSACTION
INSERT INTO Customers (CustomerID, CompanyName, ContactName)
VALUES ('ZBCVT', 'Firma X', 'Arnold S')
IF @@ERROR <> 0
    BEGIN
        ROLLBACK TRANSACTION
        RETURN 10
    END
UPDATE Customers
SET City='Berlin'
WHERE CustomerID = 'ZBVCT'
IF @@ERROR <> 0
    BEGIN
        ROLLBACK TRANSACTION
        RETURN 11
    END
COMMIT TRANSACTION
GO
```

Status zwracanego błędu o stopniu zagrożenia wynoszącym 10 lub mniej oznacza ostrzeżenie i nie jest zgłaszany wyjątek. Błędy występujące w procedurach z poziomem 11 do 20 zwracają wyjątek języka SQL wyłapywany przez **SqlErrorCollection**. Niektóre błędy o poziomie 11 lub większym przerywają całą procedurę składowaną.

Transakcje nazwane

Oprócz zwykłego zarządzania i obsługą błędów w transakcjach istnieje możliwość tworzenia, wykonywania i cofania transakcji nazwanych, czyli takich o zadeklarowanej nazwie.

```
CREATE TABLE ValueTableS (wartość INT)
GO
DECLARE @NazwaTransakcji varchar(20) = 'Transakcja'
BEGIN TRAN @NazwaTransakcji
    INSERT INTO ValueTableS VALUES(1),(2);

    ROLLBACK TRAN @NazwaTransakcji
    INSERT INTO ValueTableS VALUES(3),(4);
SELECT wartość FROM ValueTableS

DROP TABLE ValueTableS
```



```
DECLARE @NazwaTransakcji varchar(20) = 'Transakcja'
BEGIN TRAN @NazwaTransakcji
    DELETE FROM CustomersTMP
    ROLLBACK TRAN @NazwaTransakcji
SELECT * FROM CustomersTMP
```

Obsługa błędów w transakcji

Podczas tworzenia dłuższych skryptów konieczne może się okazać sprawdzanie poprawności ich wykonania, a w razie potrzeby cofnięcia dokonanych zmian. W tym celu niezbędne jest zastosowanie bloku TRY / CATCH. Działanie powyższych instrukcji możesz prześledzić na poniższym przykładzie.

```
CREATE TABLE ##pojazdy
    (id INT, pojazd NVARCHAR(30))

BEGIN TRANSACTION

-- początek transakcji

BEGIN TRY

    -- Insert modyfikacja danych

    INSERT INTO ##pojazdy
        (id, pojazd) SELECT 1, 'Rower'

    -- Wywołanie błędu

    SELECT 2/0
END TRY

-- Przechwytywanie błędu

BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;
    IF @@TRANCOUNT > 0

    -- Cofanie zmian

        ROLLBACK TRANSACTION;
END CATCH;

-- Zatwierdzenie zmian

IF @@TRANCOUNT > 0
    COMMIT TRANSACTION;

SELECT * FROM ##pojazdy

DROP TABLE ##pojazdy
```

8.8. Zabezpieczenia danych

Dane przechowywane w tabelach bazy danych mogą być szyfrowane, ale osobną kwestią pozostaje ich czytelność. Niektóre z kolumn, na przykład zawierające hasła użytkowników powinny być przechowywane w formie nieczytelnej dla przypadkowego operatora bazy danych.

W pierwszej kolejności przygotuj procedurę zwracającą dane. Taka konfiguracja zapytania pozwoli określić jakie informacje będą zwracane do użytkownika pobierającego informacje z bazy danych. Użytkownik nie będzie miał fizycznie dostępu do danych, ale dane zwraca mu procedura, która odpowiada także, za kontrolę wyświetlania informacji zwrotnych.

```
CREATE PROCEDURE CustomerShow
AS
SELECT *
FROM Customers
```

Uruchom procedurę, aby przetestować jej działanie.

```
EXEC CustomerShow
```

W pierwszym kroku dodaj kolumnę hasło – możesz ustawić typ danych jako NVARCHAR(50)

```
ALTER TABLE Customers
ADD Hasło NVARCHAR(50)
```

Jako, że zapisanie hasła otwartym tekstem nie do końca ma sens usuń kolumnę Hasło

```
ALTER TABLE Customers
DROP COLUMN Hasło
```

Dane w kolumnach można zabezpieczyć za pomocą funkcji hashujących. Zaktualizuj hasło na trzy różne sposoby: bez konwersji, stosując hashowanie MD5 i SHA1.

```
UPDATE Customers
SET Hasło = '123'
WHERE Country = 'USA'

UPDATE Customers
SET Hasło = HASHBYTES('MD5','abc123')
WHERE Country = 'UK'

UPDATE Customers
SET Hasło = HASHBYTES('SHA1','abc123')
WHERE Country = 'France'
```

Sprawdź jak wyglądają zwracane dane z serwera za pomocą procedury CustomerShow. Zwróć uwagę na wygląd zwracanych danych. Najlepszą metodą na przechowanie hasła jest zapis danych w formacie binarnym. Co pozwoli ukryć w łatwy sposób treść haseł. Sprawdź działanie funkcji hashującej MD5 w poleceniu PRINT.

```
PRINT HASHBYTES('MD5','abc123')
```

Usuń poprzednią kolumnę hasło, a następnie dodaj wspomnianą kolumnę w formie binarnej.

```
ALTER TABLE Customers
ADD Hasło VARBINARY(255)
```

Materiały szkoleniowe

Dodając dane do kolumny binarnej należy je przekonwertować z formatu tekstowego do binarnego. Aktualizację haseł wykonaj także na 3 sposoby.

```
UPDATE Customers
  SET Hasło = CONVERT(VARBINARY(255), '123')
  WHERE Country = 'USA'

UPDATE Customers
  SET Hasło = CONVERT(VARBINARY(255), HASHBYTES('MD5','abc123'))
  WHERE Country = 'UK'

UPDATE Customers
  SET Hasło = CONVERT(VARBINARY(255), HASHBYTES('SHA1','abc123'))
  WHERE Country = 'France'
```

Dodaj do procedury parametr hasło i zmodyfikuj ją ośpownie

```
ALTER PROCEDURE CustomerShow
@hasło NVARCHAR(50)
AS
SELECT *
  FROM Customers
  WHERE Hasło = @hasło
```

Przetestuj działanie procedury

```
EXEC CustomerShow @hasło = '123'
```

Spróbuj także odczytać dane bezpośrednio z tabeli, stosując jako warunek dane w formie binarnej. Dane takie zapisujemy poprzedzając ciąg prefiksem 0x

```
SELECT * FROM Customers
  WHERE Hasło = 0xE99A18C428CB38D5F260853678922E03
```

Zmodyfikuj procedurę tak, aby pobierała hasło w formie tekstowej, a warunek był sprawdzany już w wersji przekonwertowanej. Uwaga typ danych tekstowych to VARCHAR, a nie NVARCHAR – z uwagi na dwubajtowe kodowanie tekstu.

```
ALTER PROCEDURE CustomerShow
-- TYP DANYCH NIE NVARCHAR() !
  @hasło varchar(255)
AS
SET @hasło = HASHBYTES('MD5',@hasło)
SELECT *
  FROM Customers
  WHERE Hasło = @hasło
```

Poprawnie wykonane zapytanie zwróci tylko te dane, gdzie kolumna posiada hasło w formie zaszyfrowanej.

```
EXEC CustomerShow @hasło = 'abc123'
```

9. Hurtownie danych

Baza danych SQL Server może poza swoimi własnymi bazami i tabelami odwoływać się do obiektów znajdujących się na innych serwerach. Dzięki czemu może stanowić doskonały punkt wyjścia dla budowy hurtowni danych, która integruje w sobie wiele ich różnorodnych źródeł.

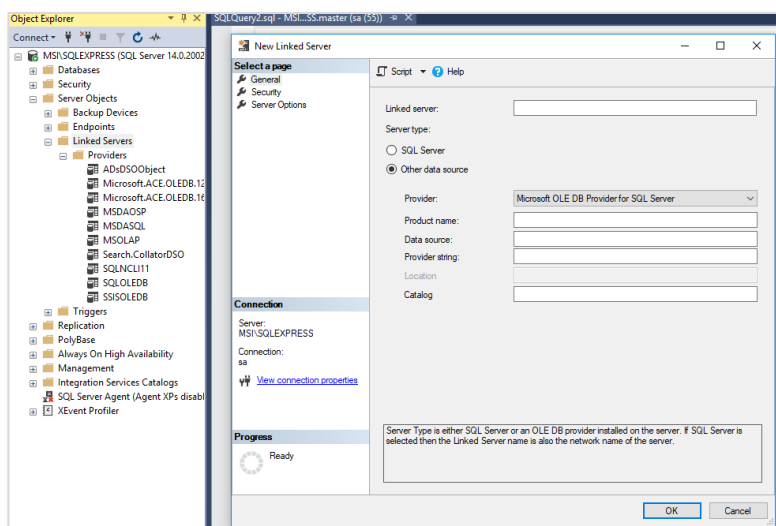
9.1. Linked server

Podstawową metodą dodawania zewnętrznych źródeł danych jest Linked Server, który dostępny jest z użyciem Management Studio.

Aktywne Linked Servers znajdują się w Server Objects.

W trybie graficznym stworzenie połączenia sprowadza się do kliknięcia prawym klawiszem myszy i wybrania New Linked Server.

W przypadku innego SQL Servera, wystarczy znać jego adres, login i hasło, aby go podłączyć.



Dla innych typów danych można skorzystać z połączeń opartych o systemowe ODBC. Niekiedy konieczne będzie doinstalowanie dodatkowego sterownika ze strony producenta konkretnej bazy danych (MySQL, Oracle itp.)

Tworzenie Linked Server

Z użyciem skryptu dodanie kolejnego serwera wygląda następująco. Tworzenie Linked Server dla MySQL. Inne bazy działają podobnie. Dodatkowe informacje na:

<https://docs.microsoft.com/en-us/sql/t-sql/FUNCTIONS/openquery-transact-sql?view=sql-server-2017>

```
EXEC master.dbo.sp_addlinkedserver
    @server      = N'MÓJ MYSQL',          -- nazwa serwera
    @srvproduct  = N'MySQL',              -- nazwa produktu
    @provider     = N'MSDASQL',           -- sterownik
    @datasrc     = N'Mój MySQL'           -- nazwa źródła ODBC
```

Tworzenie loginu dla Linked Server

Podczas tworzenia serwera konieczne jest jeszcze utworzenie loginu z uprawnieniami

```
EXEC master.dbo.sp_addlinkedsrvlogin
    @rmtsrvname  = N'MÓJ MYSQL',
    @useself     = N'False',
    @locallogin   = NULL,
    @rmtuser     = N'login',
    @rmtpassword  = 'hasło'
```

Parametry Linked Server

Dodatkowe parametry połączenia. Można je wyklikać podczas tworzenia serwera, i są one opcjonalne.

```
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'collation compatible', @optvalue=N'false'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'data access', @optvalue=N'true'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'dist', @optvalue=N'false'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'pub', @optvalue=N'false'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'rpc', @optvalue=N'false'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'rpc out', @optvalue=N'false'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'sub', @optvalue=N'false'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'connect timeout', @optvalue=N'0'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'collation name', @optvalue=null
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'lazy schema validation', @optvalue=N'false'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'query timeout', @optvalue=N'0'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'use remote collation', @optvalue=N'true'
EXEC master.dbo.sp_serveroption
    @server=N'MÓJ_MYSQL', @optname=N'remote proc transaction promotion',
    @optvalue=N'true'
```

9.2. Openquery dla Linked Server

Podłączone źródła danych w postaci Linked Server wymagają specjalnego traktowania, jeśli chodzi o sposób pobierania z nich danych jak również ich aktualizacji, czy usuwania. Służy do tego OPENQUERY, które samo w sobie zawiera zapytanie do serwera. Co istotne zapis kwerendy wewnątrz musi być zgodny z zapisem SQL serwera źródłowego (Np. MySQL, Oracle)

Pobieranie danych ze zdalnego źródła

Standardowe polecenie pobierające dane ma postać:

```
SELECT *
FROM OPENQUERY
    ( [MÓJ MYSQL], 'SELECT * FROM northwind.Customers' )
```

OPENQUERY w formie widoku

Tworząc zapytania do dalszego użycia dobrze pamiętać o możliwości stworzenia widoku w oparciu o te dane. Zwróć uwagę, jakiej składni wymaga zdalny serwer i czy wielkość liter ma znaczenie.

```
CREATE VIEW ZdalnyCustomers
AS
SELECT *
FROM OPENQUERY
    ( [MÓJ MYSQL], 'SELECT * FROM northwind.Customers' )
```

Optimalizacja zapytań w OPENQUERY

Większość obróbki danych może zostać wykonana po stronie serwera źródłowego, co znaczy, że można pobierać z niego znacznie mniejsze porcje danych do dalszej obróbki. Należy pamiętać, aby nie sortować pobieranych danych, ponieważ znacząco przedłuża to czas przesyłu wyniku zapytania. Dlatego też użycie WHERE na poziomie podkwerendy znacząco zmniejsza ilość przesłanych danych.

```
SELECT *
  FROM OPENQUERY
    ([MÓJ MYSQL], 'SELECT * FROM northwind.Customers WHERE Customers.CustomerID =
    ''CVFRT'' ')
```

Późniejsze pobranie danych sprowadza się do prostej kwerendy:

```
SELECT *
  FROM ZdalnyCustomers
```

Tworzenie tabeli z wyniku OPENQUERY

Dobrym pomysłem na buforowanie przesyłanych danych jest tworzenie w bieżącej bazie tabel pobranych z Linked Serwera.

```
INSERT INTO
  OPENQUERY ([MÓJ MYSQL], 'SELECT * FROM northwind.Customers')
  SELECT * FROM CUSTOMERS
```

Dodawanie rekordu do zdalnej tabeli

Dodawanie rekordów do zdalnej tabeli wymaga, aby kolumny były wypisane, nie może to być *.

```
INSERT INTO
  OPENQUERY ([MÓJ MYSQL], 'SELECT CustomerID, CompanyName
  FROM northwind.Customers')
  VALUES ('YXXXX', 'Firma Y')
```

Dodawanie większych ilości rekordów

Tak jak w przypadku innych kwerend możliwe jest dodanie większych ilości rekordów na raz za pomocą instrukcji SELECT.

```
INSERT INTO
  OPENQUERY ([MÓJ MYSQL], 'SELECT CustomerID, CompanyName
  FROM northwind.Customers')
  SELECT CustomerID, CompanyName
    FROM CustomersOLD
   WHERE CustomerID = 'CVFRT'
```

Kasowanie danych przez OPENQUERY

Kasowanie CustomerID może być podkreślone, jako pole nieistniejące, błędne. Mimo to działa

```
DELETE FROM
  OPENQUERY ([MÓJ MYSQL], 'SELECT CustomerID, CompanyName
  FROM northwind.Customers')
  WHERE CustomerID = 'CVFRT'
```

Błędy OPENQUERY

Zwracane przez OPENQUERY błędy nie zawsze są poprawnie opisane, dlatego mogą one być po prostu widoczne jako błąd, bez dodatkowych informacji i szczegółów np. informacja zwrotna jest niesprecyzowana w przypadku powtórzonego **Customer ID**.

```
Msg 7399, Level 16, State 1, Line 22
The OLE DB provider "MSDASQL" for linked server "MÓJ MYSQL" reported an error. The
provider did not give any information about the error.
Msg 7343, Level 16, State 2, Line 22
The OLE DB provider "MSDASQL" for linked server "MÓJ MYSQL" could not INSERT INTO
table "[MSDASQL]". Unknown provider error.
```

9.3. Rozproszone źródła danych

Oprócz metod pobierania danych z zastosowaniem Linked Server istnieją także metody bezpośredniego odwoływania się do danych za pomocą **OPENROWSET**, czyli rozproszonych źródeł danych. Aby ten typ pytań zadziałał konieczne jest włączenie dostępu do kwerend **Ad Hoc**, co można uczynić włączając najpierw opcje zaawansowane.

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE WITH OVERRIDE;
```

Następnie dopiero możliwa jest ich aktywacja, która pozwala nie mniej standardowe metody pobierania danych.

```
EXEC sp_configure 'Ad Hoc Distributed Queries', 1;
RECONFIGURE WITH OVERRIDE;
```

Listę dostępnych sterowników dla źródeł rozproszonych możesz uzyskać wydając polecenie:

```
EXEC xp_enum_oledb_providers
```

Efektem wykonania polecenia jest wykaz dostępnych źródeł danych w systemie. Uwaga sterowniki muszą być dostępne dla samego serwera, a nie dla użytkownika.

Uwaga: w przypadku chęci pobierania danych z plików tekstowych, arkuszy Excel oraz baz danych Access konieczne może okazać się doinstalowanie Microsoft Access Database Engine w wersji zgodnej z posiadaną wersją SQL Servera (32 lub 64 bity).

Provider Name	Parse Name	Provider Description
1 SQLOLEDB	{0C7FF16C-38E3-11d0-97AB-00C04FC2AD98}	Microsoft OLE DB Provider for SQL Server
2 SQLNCLI11	{397C2819-8272-4532-AD3A-FB5E43BEAA39}	SQL Server Native Client 11.0
3 Microsoft.ACE.OLEDB.12.0	{3BE786A0-0366-4F5C-9434-25CF162E475E}	Microsoft Office 12.0 Access Database Engine OLE ...
4 Microsoft.ACE.OLEDB.16.0	{3BE786A2-0366-4F5C-9434-25CF162E475E}	Microsoft Office 16.0 Access Database Engine OLE ...
5 AdaDSOObject	{549365d0-ec26-11cf-8310-00aa00b505db}	OLE DB Provider for Microsoft Directory Services
6 SSISOLED	{688037C5-0B57-464B-A953-90A806CC34C2}	OLE DB Provider for SQL Server Integration Services
7 Search CollatorDSO	{9E175B8B-F52A-11D8-B9A5-505054503030}	Microsoft OLE DB Provider for Search
8 MSDASQL	{c8b522cb-5cf3-11ce-ade5-00aa0044773d}	Microsoft OLE DB Provider for ODBC Drivers
9 MSOLAP	{DBC724B0-DD86-4772-BB5A-FCC6CAB2FC1A}	Microsoft OLE DB Provider for Analysis Services 14.0
10 MSDASDP	{dfc8bdc0-e378-11d0-9b30-0080c7e9fe95}	Microsoft OLE DB Simple Provider

Dodatkowo konieczne będzie włączenie dodatkowych parametrów przetwarzania dla OLEDB:

```
EXEC sp_MSset_oledb_prop N'Microsoft.ACE.OLEDB.12.0', N'AllowInProcess', 1
GO
EXEC sp_MSset_oledb_prop N'Microsoft.ACE.OLEDB.12.0', N'DynamicParameters', 1
GO
```

OPENROWSET dla SQL Server

Pobranie danych z zewnętrznego serwera SQL Server polega na użyciu poniższego polecenia.

```
SELECT *
FROM OPENROWSET
( 'SQLNCLI11',
  'SERVER=serwersql.pl,51433;
  DataBase=1449_northwind;
  uid=kurs;pwd=123;',
  'SELECT * FROM Customers') AS tabela
```

OPENROWSET dla pliku CSV

Dzięki sterownikowi Access Database Engine możliwy jest także import danych bezpośrednio z plików tekstowych.

```
SELECT *
FROM OPENROWSET
('Microsoft.ACE.OLEDB.12.0',
'Text;Database=C:\dane\;HDR=yes',
'SELECT * FROM klienci.csv')
```

OPENROWSET dla arkusza Excel

Ten sam sterownik pozwala także na pobieranie danych bezpośrednio z pliku XLSX, czyli Microsoft Excel.

```
SELECT * FROM
OPENROWSET(
'Microsoft.ACE.OLEDB.12.0',           -- provider_name
'Excel 12.0;Database=C:\dane\plik.xlsx', -- provider_string
'SELECT * from [Arkusz1$]')           -- query lub sam [Arkusz1$], wtedy bez '
```

OPENROWSET dla pliku XML

Bez dodatkowych ustawień możliwe jest zaciągnięcie dowolnego pliku w całości do bazy danych.

```
SELECT CONVERT(xml, BulkColumn, 2) as XmlDocs
FROM
OPENROWSET
(BULK 'C:\Dane\A001z020102.xml',SINGLE_BLOB) as T1
```

Dzięki małej modyfikacji możliwe jest także bezpośrednie wydobycie danych z tabel umieszczonych w pliku XML, dzięki wykorzystaniu standardowych poleceń dostępnych w języku SQL. Poniższy przykład demonstruje metodę pobrania

```
DECLARE @xmldata XML
SELECT @xmldata=CONVERT(xml, BulkColumn, 2)
FROM
OPENROWSET (BULK 'C:\Dane\A001z020102.xml',SINGLE_BLOB) as T1

SELECT
    X.element.value('kod_waluty[1]','char(3)') as kod_waluty,
    X.element.value('..data_publicacji[1]','varchar(10)') as data_publicacji,
    X.element.value('kurs_sredni[1]','varchar(10)') as kurs_sredni,
    -- Cast( Replace(X.element.value('kurs_sredni[1]','varchar(10)'),',','.') as
smallmoney) as kurs_sredni,
    X.element.value('przelicznik[1]','INT') as przelicznik
FROM
    @xmldata.nodes ('/tabela_kursow/pozycja') as X(element)
```


10. Podstawy administracji serwera

10.1. Instalacja serwera SQL

Microsoft SQL Server (MS SQL) – system zarządzania bazą danych, wspierany i rozpowszechniany przez korporację Microsoft. Jest to główny produkt bazodanowy tej firmy, który charakteryzuje się tym, iż jako język zapytań używany jest przede wszystkim Transact-SQL, który stanowi rozwinięcie standardu ANSI/ISO.

Wersje programu SQL Server



Bezpłatna wersja próbna

180-dniowa pełna bezpłatna wersja próbna.



Edycja Developer

Kompletna, bezpłatna do tworzenia i testowania rozwiązań.



Edycja Express

Bezpłatna edycja programu do dowolnych zastosowań, w tym komercyjnych

Link do pobrania oprogramowania:

Zalecanie jest pobieranie wersji instalacyjnych SQL Server wyłącznie ze strony producenta.

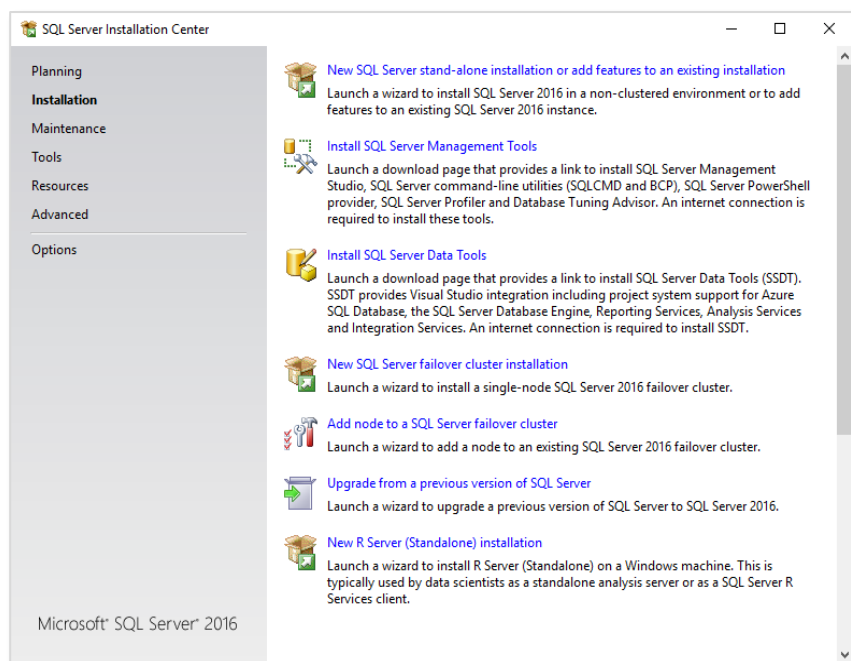
<https://www.microsoft.com/pl-pl/sql-server/sql-server-downloads>

Główne wydania SQL Server

- Wersja 2008 R2 – ostatnia wersja działająca z Windows XP i Windows Server 2003
- Wersja 2014 – ostatnia wydana wersja silnika w wersji 32 bitowej, ostatnia obsługiwana przez Windows 7.
- Wersja 2017 – kompatybilna wyłącznie z Windows 10 i Windows Server.

Instalacja silnika baz danych

Po pobraniu instalatora bazy danych oraz uruchomieniu programu instalacyjnego, wybierz opcję **Installation**, a następnie New SQL Server stand-alone installation.



Przydatne: Do instalacji zalecane jest pobranie najnowszej wersji silnika bazy danych z uwagi na większą ilość dostępnych funkcji. Podczas tworzenia samej bazy istnieje możliwość uruchomienia jej w trybie zgodności z wersjami wcześniejszymi.

Materiały szkoleniowe

Pierwszy krok instalacji umożliwia wybór instalowanej wersji silnika bazy danych: od wersji testowej (evaluation) poprzez Developer po wersję Express.

Product Key
 License Terms
 Global Rules
 Product Updates
 Install Setup Files
 Install Rules
 Feature Selection
 Feature Rules
 Feature Configuration Rules
 Ready to Install
 Installation Progress
 Complete

Validate this instance of SQL Server 2016 by entering the 25-character key from the Microsoft certificate of authenticity or product packaging. You can also specify a free edition of SQL Server: Developer, Evaluation, or Express. Evaluation has the largest set of SQL Server features, as documented in SQL Server Books Online, and is activated with a 180-day expiration. Developer edition does not have an expiration, has the same set of features found in Evaluation, but is licensed for non-production database application development only. To upgrade from one installed edition to another, run the Edition Upgrade Wizard.

☒ Specify a free edition:
 Developer
 Evaluation
 Developer
 Express

Uwaga: Ograniczenia wersji express: limit wielkości bazy danych 10 GB, ograniczenie pamięci RAM dla bazy do 1 GB, limit użycia do 4 rdzeni procesora.

Rule	Status
✓ Fusion Active Template Library (ATL)	Passed
✓ Consistency validation for SQL Server registry keys	Passed
✓ Computer domain controller	Passed
✓ Microsoft .NET Application Security	Passed
⚠ Windows Firewall	Warning

Po wybraniu edycji, zostaniesz poproszony o zatwierdzenie umowy użytkownika, a po tej procedurze nastąpi sprawdzenie zgodności systemu.

Jeśli nie wystąpią żadne problemy z weryfikacją wymagań zostaną wyświetlone kolejne okna dialogowe.

Podczas tych kroków możliwe jest skorzystanie z usługi **Microsoft Update**, która pobierze ewentualne aktualizacje oprogramowania dla serwera.

Na tym etapie możesz zdecydować o składnikach jakie chcesz zainstalować dla swojej bazy. Lista dostępnych opcji będzie się różniła w zależności od wybranej wersji serwera (najmniej możliwości będzie miała wersja express. Do poprawnej pracy niezbędna jest usługa **Database Engine Services**. Pozostałe usługi są opcjonalne i zależne od wybranej edycji.

Features:

Instance Features

- ☒ Database Engine Services
 - ☐ SQL Server Replication
 - ☐ R Services (In-Database)
 - ☐ Full-Text and Semantic Extractions for Sea
 - ☐ Data Quality Services
 - ☐ PolyBase Query Service for External Data
- ☐ Analysis Services
- ☐ Reporting Services - Native

Shared Features

- ☐ R Server (Standalone)
- ☐ Reporting Services - SharePoint
- ☐ Reporting Services Add-in for SharePoint Proc

☐ Default instance
☒ Named instance:

Instance ID:

Po zatwierdzeniu listy składników możesz nadać nazwę dla instancji swojego nowego serwera. Domyślnie jest to **SQLEXPRESS**.

Szczegóły konfiguracji serwera pozwalają na zdefiniowanie konfiguracji logowania do serwera. **Windows authentication mode** (tryb oparty na kontach systemu Windows) lub **Mixed mode** (dodatkowo możliwość logowania za pomocą kont serwera SQL).

W tym przypadku korzystamy z opcji **Mixed**, która umożliwia samodzielne zarządzania kontami dostępowymi. Dodatkowo bieżący użytkownik komputera jest dodawany jako administrator serwera baz danych.

Server Configuration | Data Directories | TempDB | FILESTREAM

Specify the authentication mode and administrators for the Database Engine.

Authentication Mode

- ☐ Windows authentication mode
- ☒ Mixed Mode (SQL Server authentication and Windows authentication)

Specify the password for the SQL Server system administrator (sa) account.

Enter password:
 Confirm password:

Specify SQL Server administrators

MS\Sebastian (Sebastian)	SQL Server administrators have unrestricted access to the Database Engine.
--------------------------	--

Add Current User | Add... | Remove

Uwaga: Standardowy login dla konta administratora to sa (server admin). To konto najwyższego rzędu, które pozwala na pełny dostęp do funkcji bazy danych.

Z tego samego miejsca możliwe jest zdefiniowanie katalogu, w którym będą przechowywane pliki bazy danych (Data directories).

W kolejnych krokach istnieje możliwość zdecydowania jakie usługi serwera i w jaki sposób będą uruchamiane (Automatic, Manual, Disabled).

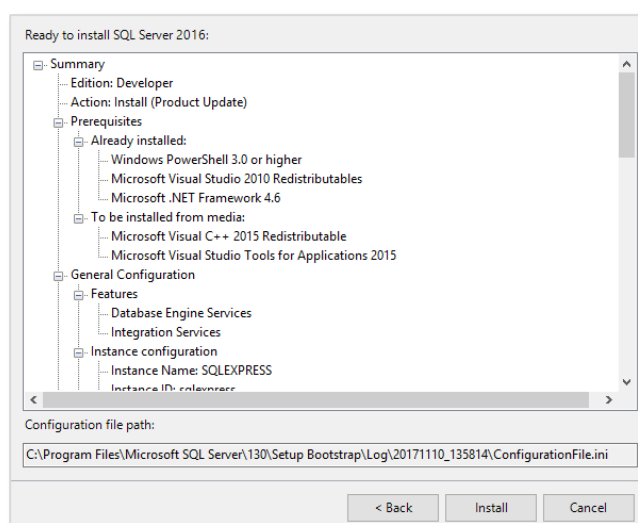
Service Accounts

Collation

Microsoft recommends that you use a separate account for each SQL Server service.

Service	Account Name	Password	Startup Type
SQL Server Agent	NT Service\SQLAgent\$S...		Manual
SQL Server Database Engine	NT Service\MSSQL\$SQL...		Automatic
SQL Server Integration Services 13.0	NT Service\MsDtsServer...		Automatic
SQL Server Browser	NT AUTHORITY\LOCAL ...		Automatic

Z punktu widzenia lokalnego użytkownika istotna jest usługa **SQL Database Engine**, włączająca sam silnik bazy danych, a dla klienta sieciowego **SQL Server Browser**.



Ostatnim krokiem przed rozpoczęciem instalacji jest lista kontrolna, jeśli wszystkie opcje są zaznaczone poprawnie, możesz przejść do kolejnego kroku, aby rozpocząć samą instalację

Po zakończeniu instalacji i wyświetleniu podsumowania, może być konieczne zresetowanie komputera.

Uwaga: Do zarządzania usługami serwera i ich dostępnością służy **SQL Server 2017 Configuration Manager** instalowane razem z bazą danych.

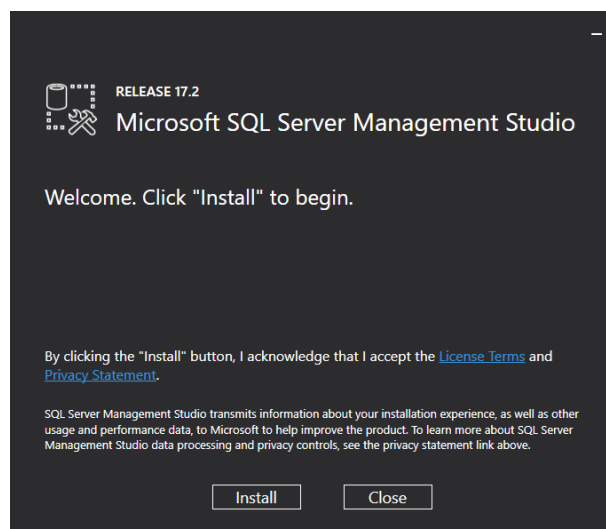
10.2. Instalacja Microsoft SQL Server Management Studio

SQL Server Management Studio to zintegrowane środowisko do zarządzania wszystkimi komponentami (baza danych, usługi analityczne, usługi raportowe itd.), wchodzącymi w skład Microsoft SQL Server. Zawiera narzędzia do konfiguracji, monitorowania i administrowania instancjami SQL Server. Umożliwia budowę zapytań i skryptów, zawiera zarówno edytor skryptów jak i narzędzia graficzne. Aplikacja po raz pierwszy pojawiła się wraz z Microsoft SQL Server 2005. Główną cechą aplikacji jest Object Explorer, który pozwala na przeglądanie, wybieranie i wykonywanie różnych działań na obiektach serwera.

Oprogramowanie jest dostępne do pobrania ze strony producenta. Sam proces instalacji dla najnowszej wersji polega na uruchomieniu instalatora i kliknięciu Install. Instalacja przebiega automatycznie.

Link do pobrania oprogramowania

<https://docs.microsoft.com/pl-pl/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017>



10.3. Kopia zapasowa

Kopia zapasowa może być wykonana za pomocą interfejsu graficznego, jak i z zastosowaniem języka SQL. Należy pamiętać jednak, że automatyzacja kopii możliwa jest przede wszystkim z zastosowaniem, drugiej opcji.

Kopia w domyślnej lokalizacji

Tworzenie kopii zapasowej z użyciem skryptu SQL wymaga polecenia podania **BACKUP**, nazwy archiwizowanej bazy, typu nośnika (np. DISK) oraz nazwy pliku kopii. Domyślnie jest ona zapisywana w domyślnym katalogu kopii, utworzonym podczas instalacji SQL Servera.

```
BACKUP DATABASE northwind TO DISK ='kopia_nw.bak'
```

Kopia we wskazanym katalogu

Podczas tworzenia kopii do wskazanego katalogu należy pamiętać, aby katalog ten był dostępny dla serwera SQL. Dodatkowym utrudnieniem może być fakt, że kopia nie może być wykonana na dysku sieciowym. Nowsze wersje pozwalają jednak na wykonanie kopii w chmurze.

```
BACKUP DATABASE zkm TO DISK ='c:\dane\zkm.bak'  
WITH NAME = 'baza_auto_backup', FORMAT
```

Przywracanie kopii

Przywrócenie danych z kopii możliwe jest przy użyciu polecenia RESTORE. Warto pamiętać, że podczas przywracania baza jest niedostępna dla pozostałych użytkowników.

```
RESTORE DATABASE Northwind  
FROM DISK = 'C:\Backup\northwind.bak'
```

Automatyzacja kopii zapasowej

Powyższe kroki można umieścić w stosownej procedurze, lub w skrypcie zewnętrznym i uruchomić za pomocą harmonogramu systemu Windows i polecenia SQLCMD.

10.4. Zarządzanie uprawnieniami

SQL Server posiada rozbudowane możliwości w zakresie zabezpieczania danych przed odczytem. W celu przetestowania możliwości zabezpieczania danych stwórz nową tabelę kontrahenci w oparciu o tabelę Customers.

```
SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address, City  
INTO Kontrahenci  
FROM Customers
```

Stwórz kolumnę w formie tekstowej, która będzie przechowywała nazwę użytkownika (przetestujesz działanie funkcji na swoim loginie)

```
ALTER TABLE Kontrahenci ADD Hasło NVARCHAR(50)
```

Usunąć powyższą kolumnę możesz poprzez polecenie DROP

```
ALTER TABLE Kontrahenci DROP Column Hasło
```

Zaktualizuj kolumnę hasło do nazwy użytkownika dla kraju USA, następnie stwórz widok odczytujący tabelę z danymi zalogowanego użytkownika.

```
UPDATE Kontrahenci SET Hasło = Suser_Sname()  
WHERE Country = 'USA'  
GO  
CREATE VIEW KontrahenciUżytkownika  
AS  
SELECT * FROM Kontrahenci
```

```
WHERE Hasło = Suser_Sname()
```

W kolejnym kroku stwórz procedurę, zwracającą takie same dane jak uprzednio stworzony widok.

```
CREATE PROCEDURE KontrahenciUztkownikaP
AS
SELECT * FROM Kontrahenci
WHERE Hasło = Suser_Sname()
```

W przypadku większej ilości operatorów bazy danych możesz stworzyć rozwiązanie oparte o grupy użytkowników. W celu przetestowania stwórz tabelę grupy i dodaj do niej kilka rekordów.

```
CREATE TABLE Grupy
(Grupa NVARCHAR(50))
GO
INSERT INTO Grupy (Grupa) VALUES ('sa')
INSERT INTO Grupy (Grupa) VALUES ('prezes')
```

Zmodyfikuj tak procedurę, aby warunkiem otrzymania danych zwrotnych było istnienie użytkownika w tabeli grupy.

```
ALTER PROCEDURE KontrahenciUztkownikaP
AS
IF EXISTS(Select * from Grupy WHERE grupa = SUSER_SNAME())
BEGIN
    SELECT * FROM Kontrahenci
END
```

Dodatkowe informacje o bieżącym loginie uzyskasz uruchamiając poniższe polecenia.

```
PRINT Suser_Sname()
PRINT User_ID()
PRINT User
PRINT suser_sid()
PRINT user_name()
```

10.5. Uprawnienia

Większość ustawień dotyczących uprawnień użytkownika możliwa jest do zdefiniowania za pomocą management studio i myszy. Czasami jednak konieczne jest stworzenie użytkownika/loginu i nadanie uprawnień z poziomu skryptu.

Uprawnienia obiektowe

Dostępne akcja dla obiektu

tabela i widok	kolumna	procedura
SELECT / INSERT / DELETE / UPDATE	SELECT / UPDATE / REFERENCES	EXECUTE

Typy uprawnień

- **GRANT** - przyznane - może przeprowadzić akcję
- **DENY** - zabronione - nie może wykonać operacji - opcja jest nadrzędna
- **REVOKE** - odebrane - nie może wykonać zadanej operacji - może to być nadpisane przynależnością do roli

Przykłady użycia

```
GRANT SELECT on ORDERS to szkolenie
DENY SELECT on ORDERS to szkolenie
GRANT CREATE TABLE to szkolenie
```

Tworzenie loginu

```
USE MASTER
```

Materiały szkoleniowe

```
GO
CREATE LOGIN użytkownik WITH PASSWORD = '123'
```

W wersji rozbudowanej użytkownik jest tworzony w podobny sposób.

```
CREATE LOGIN użytkownik WITH PASSWORD = '123',
    DEFAULT_DATABASE=[Northwind],
    DEFAULT_LANGUAGE=[polski],
    CHECK_EXPIRATION=OFF,
    CHECK_POLICY=ON
```

Usuwanie loginu

```
DROP LOGIN użytkownik
```

Tworzenie użytkownika

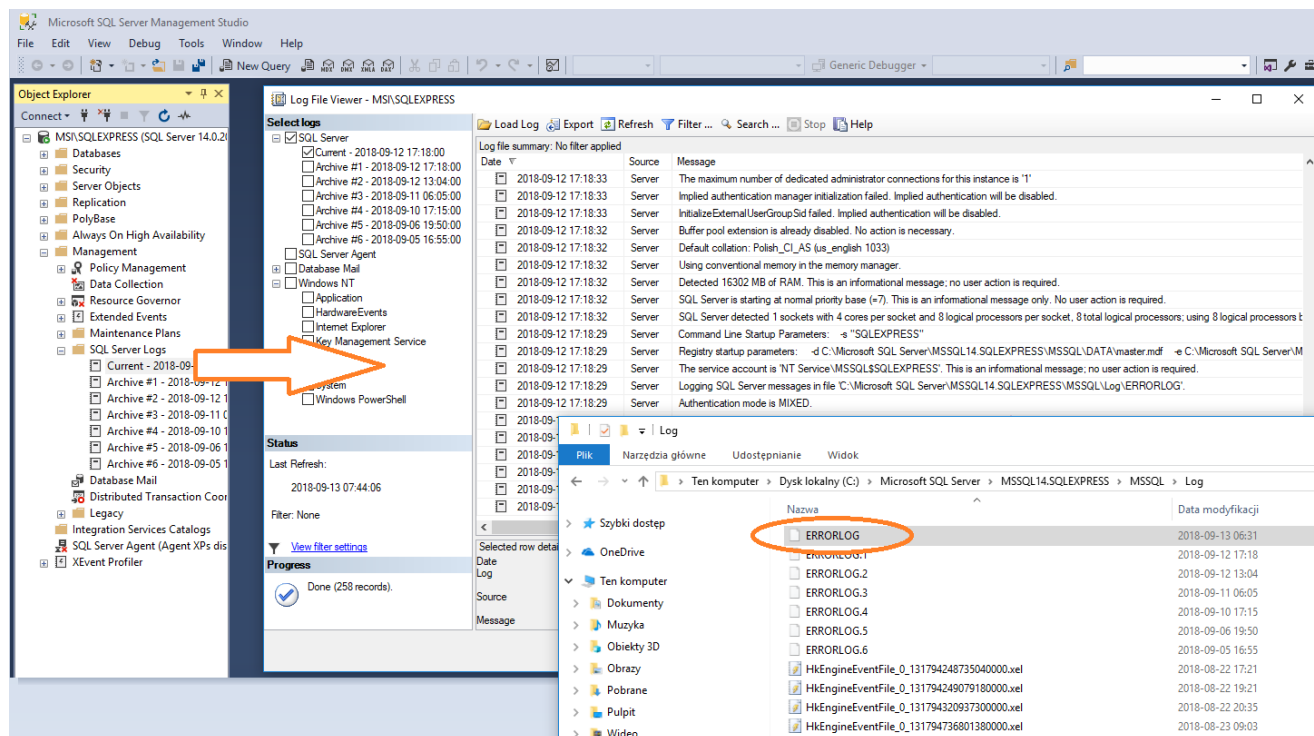
```
CREATE USER użytkownik FOR LOGIN użytkownik
GO
EXEC sp_addrolemember 'db_owner', użytkownik
```

Nadawanie uprawnień

```
GRANT ALL PRIVILEGES ON ORDERS TO użytkownik
GRANT SELECT, INSERT, UPDATE ON ORDERS TO użytkownik
```

10.6. Śledzenie błędów serwera

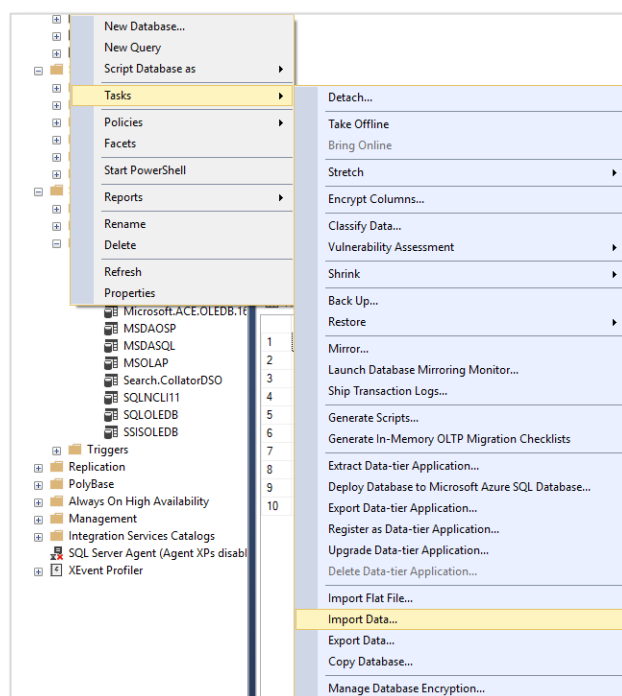
Pracując z serwerem SQL konieczne może okazać się śledzenie jego działania i przeglądanie logu z błędami. Można się do niego dostać używając Management Studio lub za pomocą eksploratora windows, poruszając się bezpośrednio po strukturze plików serwera.



11. Komunikacja z serwerem SQL – import i eksport danych

Baza danych SQL Server posiada duży wybór narzędzi pomocnych w imporcie i eksporcie danych. Część z nich to narzędzia wbudowane nie wymagające szczególnego traktowania. Dzięki zintegrowanym kreatorom możliwa jest szybka i sprawna operacja import/eksportu danych pomiędzy dowolnymi lokalizacjami.

Dodatkowymi narzędziami jest **Import Flat file**, który służy do zaawansowanego przetwarzania plików CSV/TXT.



11.1. BULK Import

Najbardziej zaawansowaną formą importu danych z plików tekstowych jest BULK Import, czyli bezpośrednia metoda pobierania danych z pliku. Jedyne o czym należy pamiętać to uprawnienia serwera do katalogu zawierającego dane do pobrania.

Metoda ta najlepiej się sprawdza z zastosowaniem tabel tymczasowych, które znacząco przyspieszają procedurę przetwarzania danych.

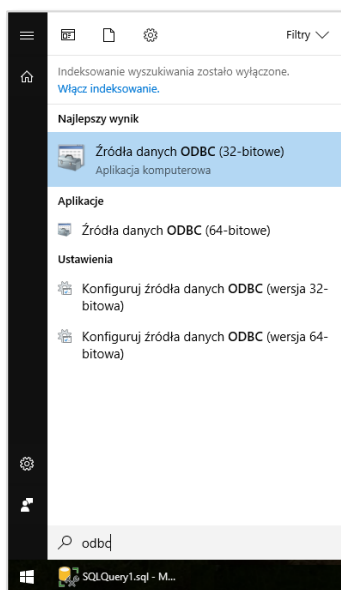
```
-- deklaracja zmiennych
DECLARE @path NVARCHAR(255)
DECLARE @sql NVARCHAR(2000)
-- tworzenie tabeli tymczasowej
CREATE TABLE #Sprzedaż
(
    id INT,
    Klient NVARCHAR(255),
    Kwota decimal(18,2),
    Pracownik INT,
    Czas datetime)
-- przypisanie wartości do zmiennych
SET @path = 'C:\dane\sprzedaż.csv'
SET @sql = 'BULK INSERT #Sprzedaż
            FROM ''' + @path + '''
            WITH (FIELDTERMINATOR = ';'',
            MAXERRORS = 0,
            ROWTERMINATOR = ''\\n'',
            FIRSTROW = 2)'
-- uruchomienie polecenia SQL
EXEC (@sql)
-- przetwarzamy dane i robimy "coś"
SELECT * FROM #Sprzedaż
-- usuwamy tabelę tymczasową
DROP TABLE #Sprzedaż
```

Uwaga: Polecenie EXEC (@sql) musi zawierać zmienną w nawiasach, ponieważ inaczej się ona nie uruchomi.

11.2. Konfiguracja połączenia ODBC

Pobieranie danych z istniejącej bazy jest możliwe poprzez połączenie bezpośrednie aplikacja klienta z serwerem lub z użyciem oprogramowania pośredniczącego – sterownika ODBC.

W przypadku zamiaru częstego korzystania z tej samej bazy danych lub łączenia się z nią z różnych aplikacji (Excel, Access, Power BI) najwygodniejsze jest skonfigurowanie połączenia ODBC, które pozwoli korzystać z danych niezależnie od wybranej aplikacji.



Podczas tego kursu wykorzystywany jest **Microsoft SQL Server**, a co za tym idzie zaprezentowana jest konfiguracja sterownika dla tej bazy danych.

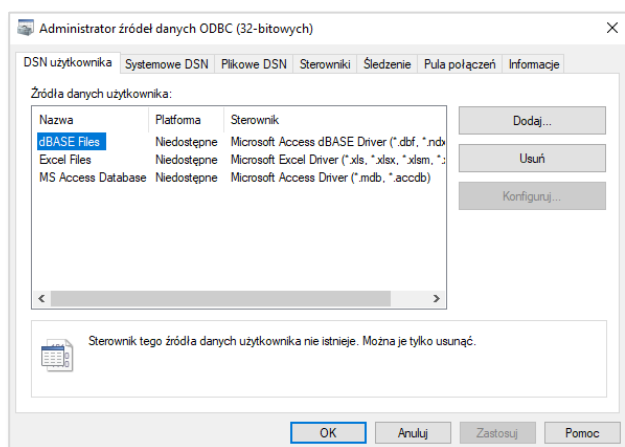
Konfigurację połączenia ODBC z bazą danych rozpocznesz od przejścia do **Menu Start** i wpisania „odbc”. W wyniku wyszukiwania pokaże się skrót **Źródła danych ODBC (32-bitowe)**.

Okno administratora baz danych ODBC umożliwia zestawienie i konfigurację wybranego połączenia.

DSN użytkownika to połączenia dostępne dla aktualnie zalogowanego użytkownika, które nie będą widoczne dla innych użytkowników komputera.

Systemowe DSN to te połączenia, które będą widoczne dla każdej osoby, jaka będzie miała dostęp do bieżącego komputera. Dla skonfigurowania tego typu połączenia niezbędne są uprawnienia administracyjne.

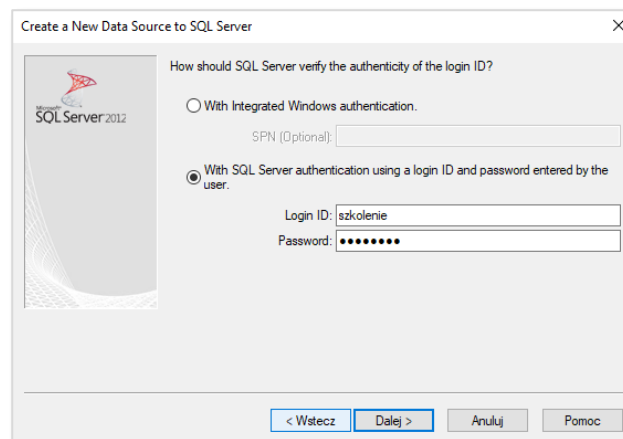
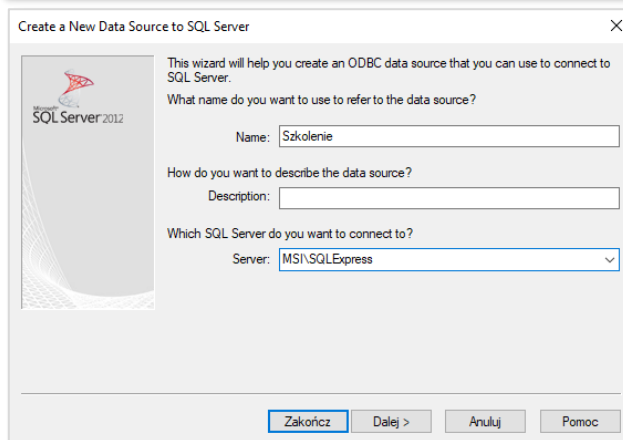
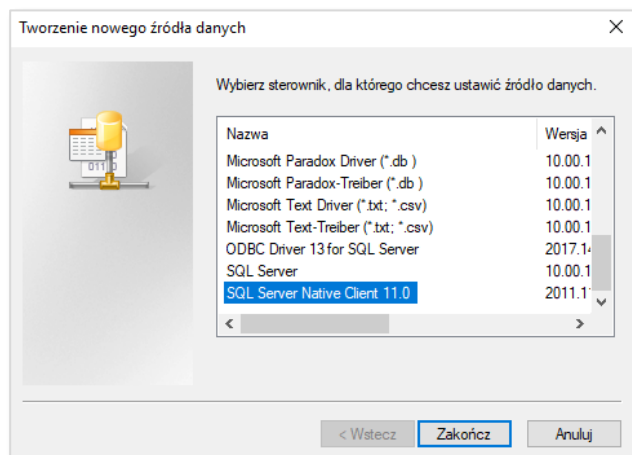
Sterowniki pozwalają na zweryfikowanie jakie są w systemie zainstalowane sterowniki baz danych. W przypadku braku ich dla konkretnej bazy danych możliwe jest ich doinstalowanie, przy czym należy zwrócić uwagę na wersję 32/64 danego sterownika.



Informacja: ODBC (ang. *Open DataBase Connectivity* – otwarte łącze baz danych) – interfejs pozwalający programom łączyć się z systemami zarządzającymi bazami danych. Jest to API niezależne od języka programowania, systemu operacyjnego i bazy danych.

Przydatne: Do połączenia się z bazą (SQL Server, Oracle, MySQL...) może być wymagane zainstalowanie dedykowanego jej sterownika ODBC. Konfiguracja dla większości baz danych jest zbliżona.

Uwaga: Wersja 32/64 bit dla sterownika ODBC powinna być zgodna z posiadaną wersją 32/64 Office. W przeciwnym wypadku połączenie nie będzie możliwe.



Chcąc dodać nowe połączenie (po upewnieniu się, że sterowniki są dostępne w systemie) kliknij **Dodaj**, aby uruchomić procedurę dodawania nowego połączenia.

Pierwsze okno pozwala na wybranie sterownika bazy danych, który zostanie użyty w nowym połączeniu. W tym konkretnym przypadku będzie to **SQL Server Native Client 11.0**. Po jego wskazaniu i kliknięciu **Zakończ** rozpocznie się procedura konfiguracji połączenia.

Użyty w przykładzie sterownik jest przeznaczony dla SQL w wersji 2012.

Dla pola **Name** (Nazwa) należy wpisać nazwę identyfikującą dane połączenie, tutaj: **Szkolenie**.

Description (opis) może zawierać opcjonalne informacje o połączeniu.

Server to adres i instancja serwera SQL, z którym połączenie zostanie zrealizowane.

Tutaj: **serwer\instancja**.

Okno danych logowania umożliwia podanie sposobu logowania (zintegrowane konto Windows) **With Integrated Windows authentication**, lub (login i hasło serwera SQL) **With SQL Server authentication**.

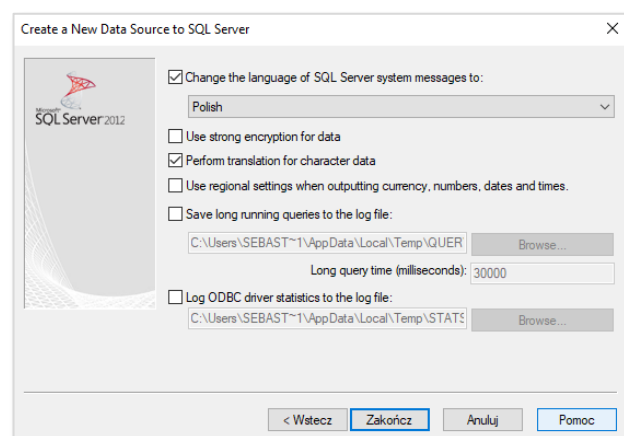
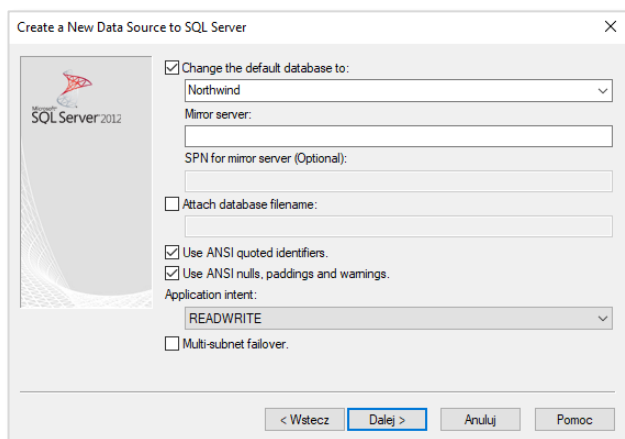
Dla bieżącego przykładu:

Login ID: **LoginSQL**

Password: **HasłoSQL**

Poprawnie podane hasło i login spowodują, iż w następnym oknie pojawią się dostępne bazy danych. Dla tego ćwiczenia to baza **Northwind**.

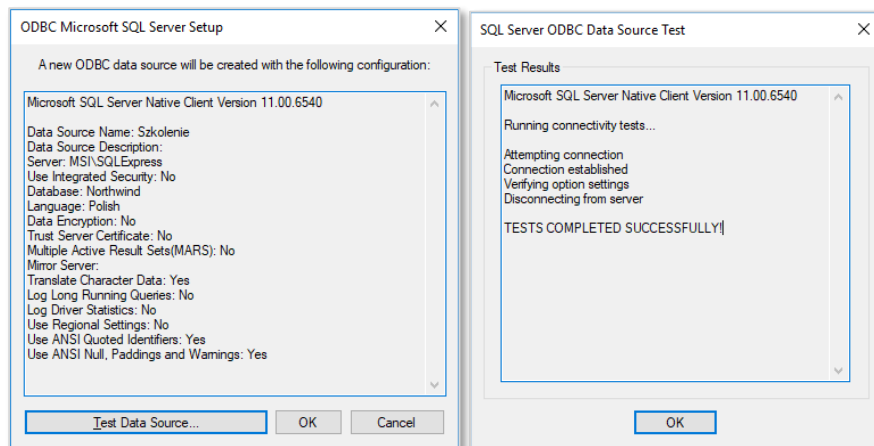
Dodatkowo okno to pozwala na ustawienie dodatkowych parametrów połączenia. Jednym z najistotniejszych jest tryb dostępu **Application intent**, który może przyjąć wartości **READONLY** (tylko odczyt) lub **READWRITE** (odczyt/zapis).



Materiały szkoleniowe

Kliknięcie **Zakończ** spowoduje wyświetlenie okna potwierdzającego zakończenie tworzenia połączenia i umożliwiające dokonanie jego testu.

Przycisk **Test Data Source** pozwala na przetestowanie nowego połączenia.



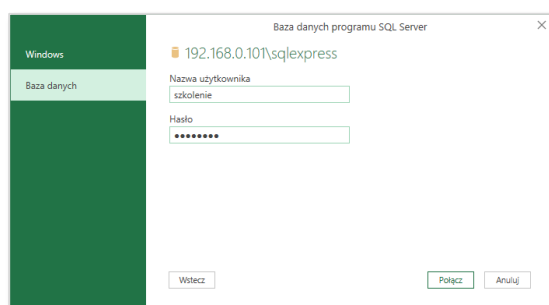
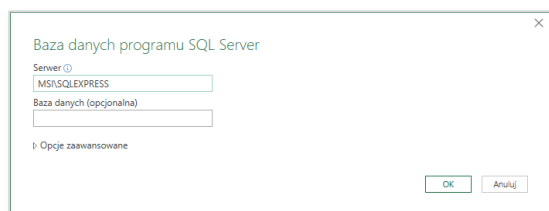
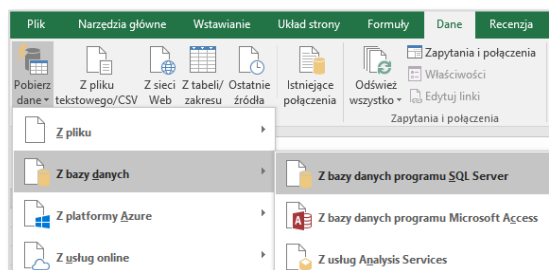
*Uwaga: Ze względów bezpieczeństwa o ile nie ma potrzeby zapisu w bazie danych, zalecane jest ustawienie **READONLY**, co zapobiegnie przypadkowemu uszkodzeniu danych.*

11.3. Połączenie SQL Server – Excel

Dysponując połączeniem poprawnie skonfigurowanym połączeniem możliwe jest bezpośrednie podłączenie danych znajdujących się na serwerze SQL. Z poziomu arkusz kalkulacyjny istnieją dwa sposoby podłączenia danych: bezpośrednie i przez interfejs ODBC.

Połączenie bezpośrednie

Zaktualizowana wersja Microsoft Office w wersji 2016/365 została wyposażona w nowy moduł przyłączania danych. Zmiana dotyczyła w szczególności Microsoft Excel.

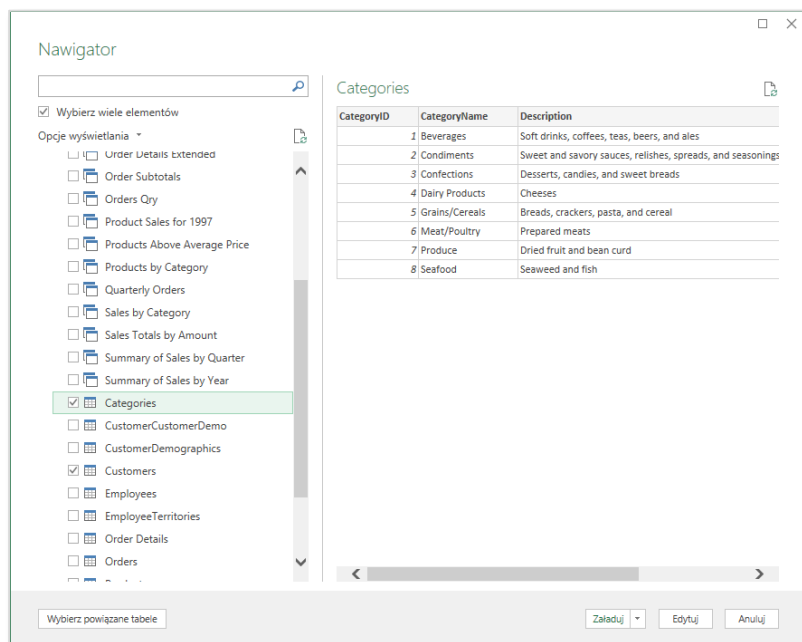


Otwórz nowy plik Excela. Przejdź na wstążkę **Dane** i rozwijając przycisk **Pobierz dane** wybierz **Z bazy danych** **Z bazy danych programu SQL Server**. Spowoduje to otwarcie okna kreatora połączeń.

W pierwszym oknie zostaniesz poproszony o podanie adresu serwera z jakim chcesz się połączyć i (opcjonalnie) bazy danych. W prezentowanym przykładzie adres serwera to **serwer\SQLEXPRESS**, a nazwa bazy pozostanie pusta. Zatwierdź swój wybór klikając OK.

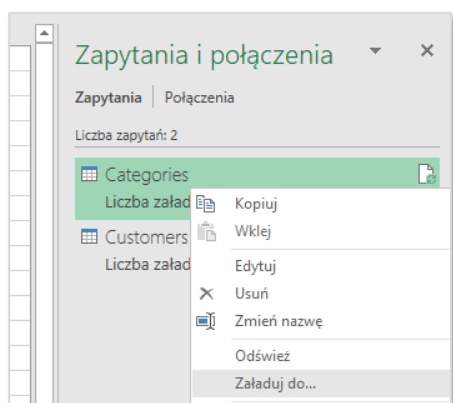
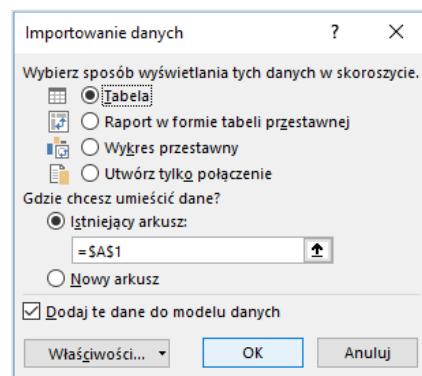
Krok drugi pozwala na wybór metody logowania **Windows** (zintegrowane) lub **Baza danych** (logowanie do serwera). Podłączając się do serwera testowego wybierz **Baza danych**, a jako login i hasło wpisz odpowiednio: **loginsql** i **haslosql**. Klikając **Połącz**, przy założeniu, że wszystkie dane zostały wpisane poprawnie powinno ukazać się okno wyboru dostępnych baz danych i obiektów w nich zawartych.

Uwaga: Wygląd okien i szczegóły połączenia danych mogą się różnić w zależności od wersji posiadanego pakietu MS Office.



Masz zamiar podłączyć naraz dwie tabele, dlatego zaznacz opcję **Wybierz wiele elementów**.

Zaznacz tabele **Categories** i **Customers**, następnie kliknij **Łaźaduj**. Spowoduje to utworzenie si¿ dwu¿ch nowych tabel połączonych w modelu danych Excela.

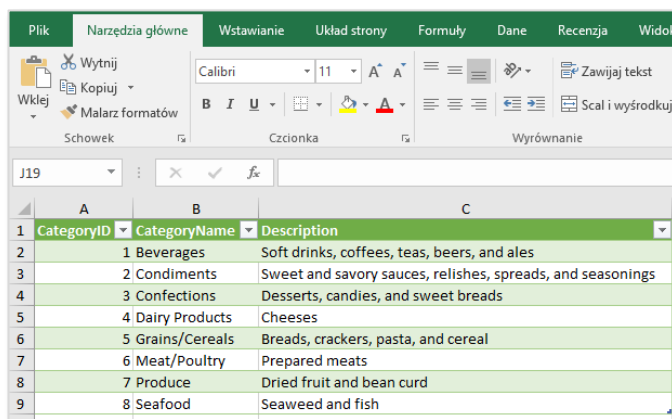


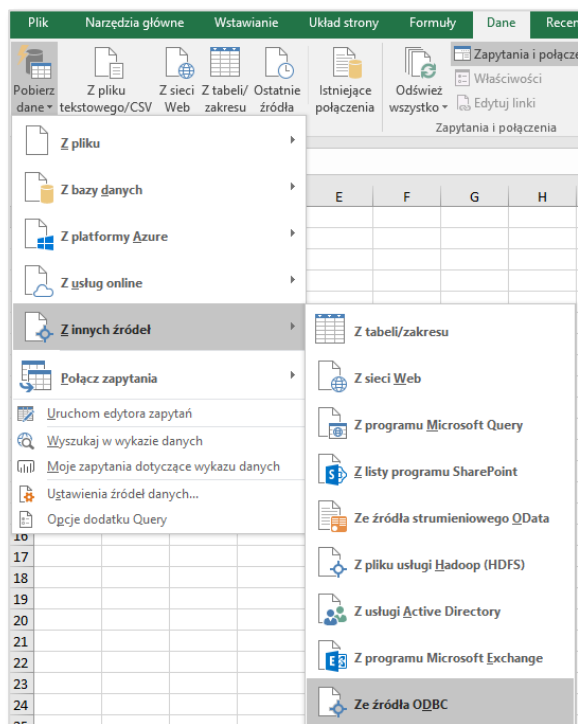
Kliknij prawym klawiszem myszy wybraną tabelę np. **Categories** i wybierz opcję **Łaładuj do...**

W oknie dialogowym Importowanie danych wskaż miejsce, gdzie mają one zostać załadowane np. **Tabela** i zatwierdź OK.

Efektem dokonanych czynności jest pobrana z serwera tabela z danymi. Jej zawartość możesz odświeżać, klikając prawym klawiszem myszy na obszarze danych i wybierając **odśwież**.

Dodatkowo możesz skorzystać modelu danych w celu dokładniejszego ich analizowania i przetwarzania.



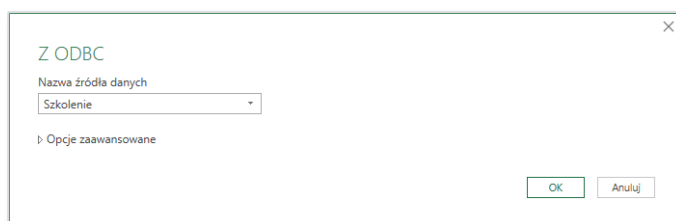


Domyślnym sposobem logowania dla tej opcji jest logowanie do **Bazy danych**. Użytkownik **loginsql**, a hasło to **hasłosql**.

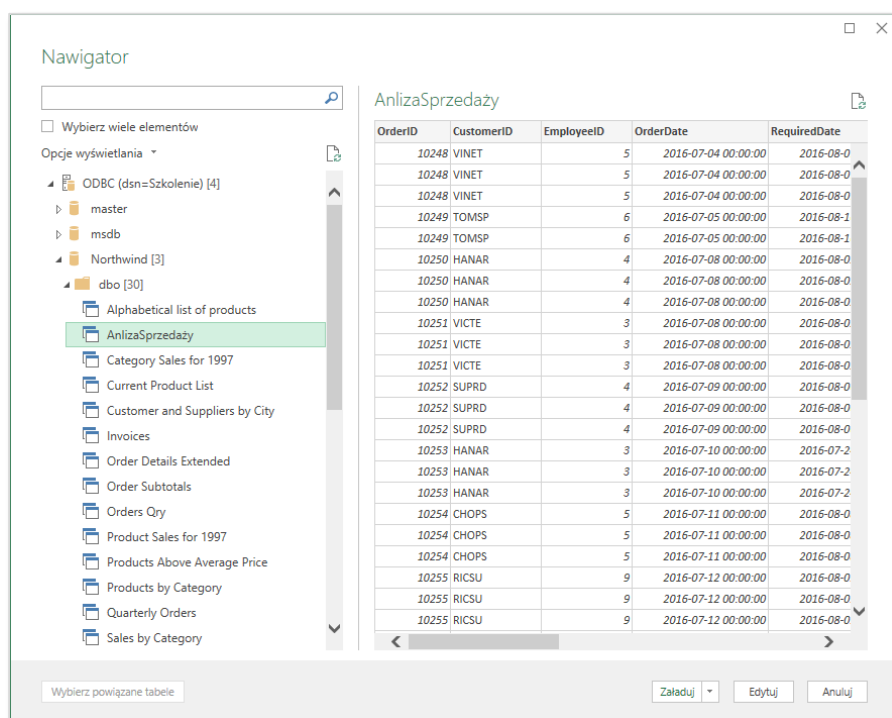
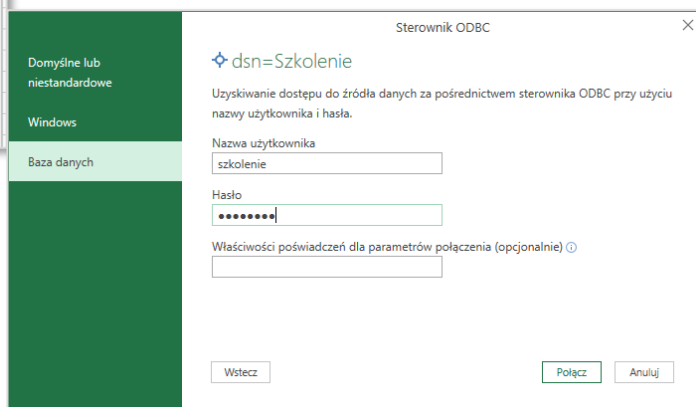
Po poprawnym zalogowaniu się powinna pojawić się lista dostępnych dla danego połączenia baz danych i zawartych w nich tabel i widoków.

Bardziej uniwersalną metodą pobrania danych do arkusza programu Excel jest skorzystanie ze skonfigurowanego uprzednio połączenia **ODBC**.

Tak jak w przypadku połączenia bezpośredniego skorzystaj ze wstążki **Dane** i **Pobierz dane** \ Z innych źródeł \ Ze źródła ODBC.



Wybierz interesujące Cię połączenie. W tym przypadku **Szkolenie** i zatwierdź **OK**.



Dodatkowo widoczne będą inne obiekty serwera baz danych SQL. Widoczność tych obiektów jest zależna od szczegółowych uprawnień danego użytkownika.

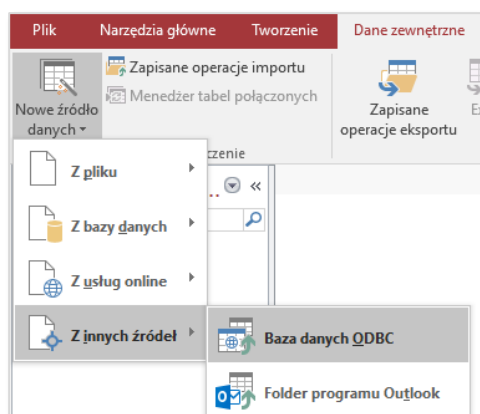
Możesz zaznaczyć jedną lub więcej tabel (wybierz wiele elementów). Może to być kwerenda **AnalizaSprzedaży**.

Przycisk **załaduj** przeniesie wskazane dane bezpośrednio do arkusza.

Metody odświeżania danych są identyczne jak w przypadku połączenia bezpośredniego.

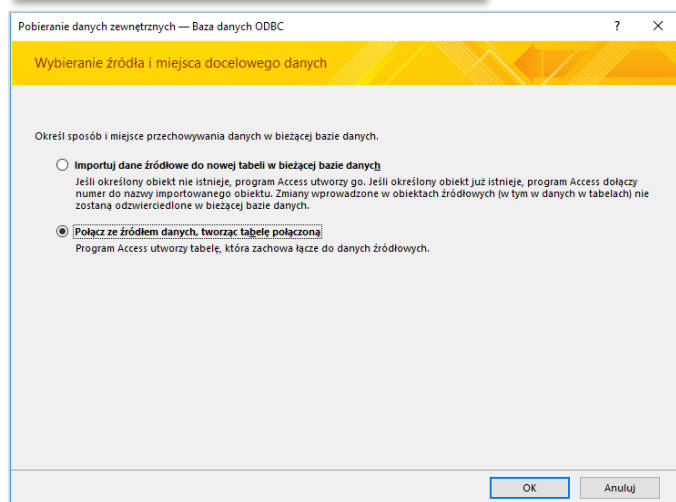
OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShipDate	ShipVia	ProductName	Quantity	Discount	UnitPrice	Wartość
10248	VINET	5	2016-07-04 00:00	2016-08-01 00:00	2016-07-16 00:00	3	Queso Cabrales	12	0	14	168
10248	VINET	5	2016-07-04 00:00	2016-08-01 00:00	2016-07-16 00:00	3	Singaporean Hokkien Fried Mee	10	0	9,8	98
10248	VINET	5	2016-07-04 00:00	2016-08-01 00:00	2016-07-16 00:00	3	Mozzarella di Giovanni	5	0	34,8	174
10249	TOMSP	6	2016-07-05 00:00	2016-08-16 00:00	2016-07-10 00:00	1	Tofu	9	0	18,6	167,4
10249	TOMSP	6	2016-07-05 00:00	2016-08-16 00:00	2016-07-10 00:00	1	Manjimup Dried Apples	40	0	42,4	1696
10250	HANAR	4	2016-07-08 00:00	2016-08-05 00:00	2016-07-12 00:00	2	Jack's New England Clam Chowder	10	0	7,7	77
10250	HANAR	4	2016-07-08 00:00	2016-08-05 00:00	2016-07-12 00:00	2	Manjimup Dried Apples	35	0,150000006	42,4	1484
10250	HANAR	4	2016-07-08 00:00	2016-08-05 00:00	2016-07-12 00:00	2	Louisiana Fiery Hot Pepper Sauce	15	0,150000006	16,8	252
10251	VICTE	3	2016-07-08 00:00	2016-08-05 00:00	2016-07-15 00:00	1	Gusta's Knäckebröd	6	0,050000001	16,8	100,8
10251	VICTE	3	2016-07-08 00:00	2016-08-05 00:00	2016-07-15 00:00	1	Ravioli Angelo	15	0,050000001	15,6	234
10251	VICTE	3	2016-07-08 00:00	2016-08-05 00:00	2016-07-15 00:00	1	Louisiana Fiery Hot Pepper Sauce	20	0	16,8	336
10252	SUPRD	4	2016-07-09 00:00	2016-08-06 00:00	2016-07-11 00:00	2	Sir Rodney's Marmalade	40	0,050000001	64,8	2592
10252	SUPRD	4	2016-07-09 00:00	2016-08-06 00:00	2016-07-11 00:00	2	Geitost	25	0,050000001	2	50
10252	SUPRD	4	2016-07-09 00:00	2016-08-06 00:00	2016-07-11 00:00	2	Camembert Pierrot	40	0	27,2	1088

11.4. Połączenie SQL Server – Access

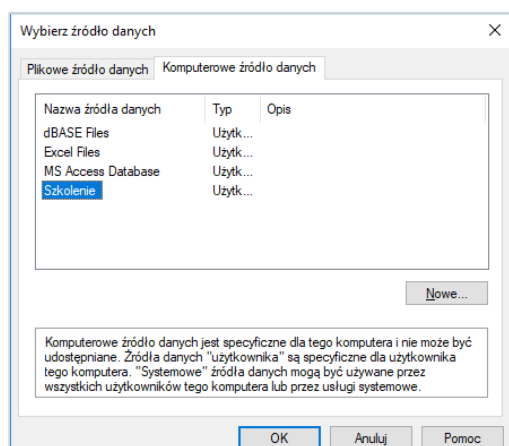


Podłączenie danych umieszczonych na serwerze SQL do bazy danych Access jest możliwe z użyciem interfejsu ODBC. Chcąc przetestować działanie połączenia utwórz nową pustą bazę danych. Przejdź na wstążkę Dane zewnętrzne, Nowe źródło danych, Z innych źródeł, Baza danych ODBC

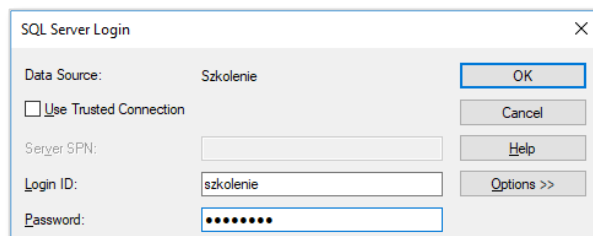
W oknie dialogowym pobierania danych zewnętrznych wybierz opcję **Połącz ze źródłem danych, tworząc tabelę połączoną**.



Uwaga: Program Access posiada możliwość importu danych do samej bazy, jak i ich przyłączenia w formie tabel dotłączonych. Są one zlokalizowane fizycznie na serwerze, a sam Access tworzy jedynie **łącze** do takiej tabeli.

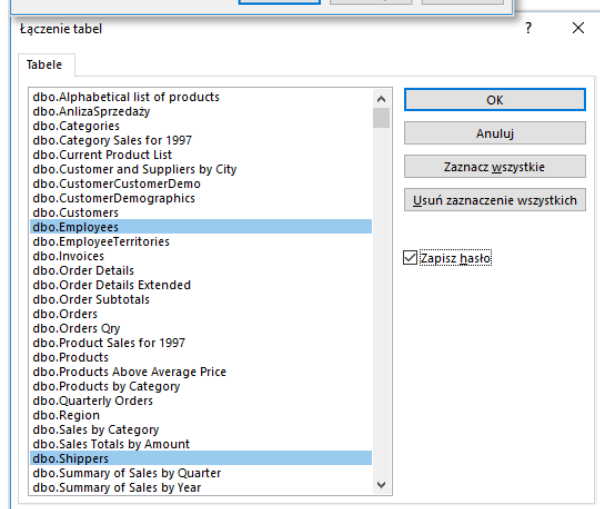


Podobnie jak w przypadku Excela, wskaż **komputerowe źródło danych** o nazwie **Szkolenie** i zatwierdź OK. W oknie logowania do serwera podaj login i hasło do serwera (**szkolenie / 12345678**). Z tego miejsca możliwe jest także tworzenie nowych połączeń ODBC.

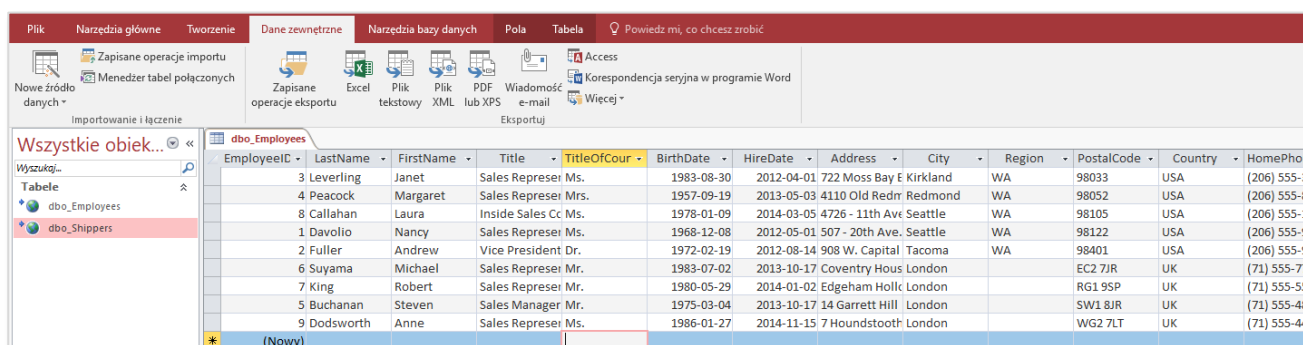
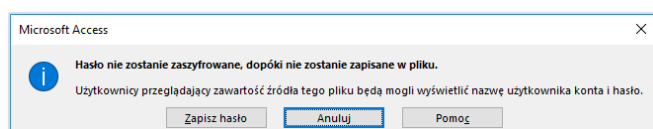


Po poprawnym zalogowaniu się do serwera możliwy staje się wybór tabel, które mają zostać dołączone do bazy Access.

Zaznacz tabele **dbo.employess** i **dbo.shippers**. Zaznacz opcję **Zapisz hasło**, co umożliwi zapamiętanie go przez program Access do ponownego użycia.



Uwaga: Tabele połączone w MS Access mają możliwość pracy w trybie Odczyt/Zapis. Co pozwala edytować dane na serwerze bezpośrednio.



12. Dodatki

12.1. Baza danych MySQL

MySQL – wolnodostępny system zarządzania relacyjnymi bazami danych. MySQL rozwijany jest przez firmę Oracle. Wcześniej przez większość czasu jego tworzeniem zajmowała się szwedzka firma MySQL AB. MySQL AB została kupiona 16 stycznia 2008 roku przez Sun Microsystems, a ten 27 stycznia 2010 roku przez Oracle. W międzyczasie Monty Widenius (współtwórca MySQL) stworzył MariaDB – forka (alternatywną wersję) opartego na licencji GPL. MariaDB jest oparta na tym samym kodzie bazowym co MySQL i dąży do utrzymania kompatybilności z jej poprzednimi wersjami.

MySQL Community Server

MySQL Community Edition to swobodnie dostępna wersja najpopularniejszej na świecie bazy danych o otwartym kodzie źródłowym, która jest obsługiwana przez aktywną społeczność programistów i entuzjastów open source.

Link do pobrania oprogramowania:

<https://dev.mysql.com/downloads/mysql/>

MySQL Workbench

MySQL Workbench zapewnia administratorom i programistom zintegrowane środowisko narzędziowe dla:

- Projektowanie i modelowanie baz danych
- SQL Development
- Administracja baz danych
- Migracja baz danych

Wersja Community (OSS) jest dostępna na tej stronie na licencji GPL.

Link do pobrania oprogramowania:

<https://dev.mysql.com/downloads/workbench/>

MySQL Connectors ODBC

MySQL oferuje standardową łączność sterowników baz danych do korzystania z MySQL z aplikacjami i narzędziami, które są zgodne ze standardami branżowymi ODBC i JDBC. Każdy system współpracujący z ODBC lub JDBC może korzystać z MySQL. Connector / ODBC to standardowy sterownik bazy danych dla platform Windows, Linux, Mac OS X i Unix

Link do pobrania oprogramowania:

<https://dev.mysql.com/downloads/connector/odbc/>

12.2. Funkcje w języku SQL dla SQL Server

Funkcje tekstowe		
Funkcje operujące na łańcuchach znaków		
ASCII	Zwraca kod ASCII dla wskazanego znaku	PRINT ASCII(11)
CHAR	Zwraca znak dla podanego kodu ASCII	PRINT CHAR(33)
CHARINDEX	Zwraca pozycję ciągu znaków w innym ciągu znaków	PRINT CHARINDEX('z','baza')
CONCAT	Łączy dwa lub więcej ciągów tekstowych	PRINT CONCAT('S','Q','L')
Concat z +	Łączy dwa lub więcej ciągów tekstowych operatorem +	PRINT 'S'+ 'Q'+ 'L'
CONCAT_WS	Łączy dwa lub więcej ciągów tekstowych z separatorem	PRINT CONCAT_WS('\$','S','Q','L')
DATALENGTH	Zwraca liczbę bajtów użytych do przechowania wyrażenia	PRINT DATALENGTH('SQL')
DIFFERENCE	Zwraca różnicę pomiędzy dwoma ciągami znaków (liczbę znaków wspólnych)	PRINT DIFFERENCE('SQL','MSSQL')
FORMAT	Formatuje wynik do wskazanego formatu	PRINT FORMAT(11.99,'0.0000')
LEFT	Zwraca znaki z lewej strony ciągu	PRINT LEFT('MSSQL',2)
LEN	Zwraca liczbę znaków w wyrażeniu	PRINT LEN('MSSQL')
LOWER	Przekształca ciąg znaków na litery małe	PRINT LOWER('MSSQL')
LTRIM	Usuwa spacje wiodące z lewej strony ciągu	PRINT LTRIM(' MSSQL')
NCHAR	Zwraca znak Unicode w oparciu o podany nr znaku	PRINT NCHAR(122)
PATINDEX	Zwraca lokalizację wzorca w łańcuchu znaków	PRINT PATINDEX('%Q%', 'MSSQL')
QUOTENAME	Zwraca łańcuch znaków unicode z dodanymi kwalifikatorami tekstu	PRINT QUOTENAME('MSSQL')
REPLACE	Zamienia jeden ciąg znaków na inny	PRINT REPLACE('MSSQL','L','F')
REPLICATE	Zwraca wskazany ciąg znaków określoną ilość razy	PRINT REPLICATE('MSSQL',2)
REVERSE	Zwraca ciąg znaków w odwróconej kolejności	PRINT REVERSE('MSSQL')
RIGHT	Zwraca znaki od prawej strony ciągu tekstowego	PRINT RIGHT('MSSQL',2)
RTRIM	Usuwa spacje od prawej strony ciągu tekstowego	PRINT RTRIM('MSSQL')
SOUNDEX	Zwraca kod porównujący dla ciągu tekstowego	PRINT SOUNDEX('MSSQL')
SPACE	Zwraca określoną liczbę spacji	PRINT SPACE
STR	Zwraca liczbę jako łańcuch znaków	PRINT STR(123)
STUFF	Usuwa wskazaną liczbę znaków z ciągu tekstowego od wskazanej pozycji, następnie zamienia ją na inny ciąg	PRINT STUFF('MSSQL',3,2,'X')
SUBSTRING	Zwraca określoną liczbę znaków z ciągu tekstowego od podanej liczby początkowej	PRINT SUBSTRING('MSSQL',3,2)
TRANSLATE	Zamienia znaki w ciągu tekstowym na wskazany	PRINT TRANSLATE('MSSQL','M','n')
TRIM	Usuwa spacje na końcu i początku ciągu tekstowego	PRINT TRIM(' MSSQL')
UNICODE	Zwraca wartość dla pierwszego znaku w ciągu tekstowym	PRINT UNICODE('MSSQL')
UPPER	Konwertuje ciąg tekstowy na litery wielkie	PRINT UPPER('mssql')
Funkcje matematyczne - zastosowania		
Standardowe funkcje matematyczne, przydatne podczas obliczeń.		
ABS	Zwraca wartość bezwzględną liczby	PRINT ABS(-12)
ACOS	Funkcja Acos zwraca arcus cosinus argumentu, czyli odwrotność jego cosinusa. Arcus cosinus to kąt, którego cosinus jest argumentem. Zwrócony kąt jest określony w radianach i ma wartość z zakresu od 0 do π .	PRINT ACOS(-1)
ASIN	Funkcja Asin zwraca arcus sinus argumentu, czyli odwrotność jego sinusa. Arcus sinus to kąt, którego sinus jest argumentem. Zwrócony kąt jest określony w radianach i należy do zakresu od $-\pi/2$ do $\pi/2$.	PRINT ASIN(1)
ATAN	Funkcja Atan zwraca arcus tangens argumentu, czyli odwrotność jego tangensa. Arcus tangens to kąt,	PRINT ATAN(1)

	którego tangens jest argumentem. Zwrócony kąt jest określony w radianach i należy do zakresu od $-\pi/2$ do $\pi/2$.	
ATN2	Funkcja Atan2 zwraca arcus tangens, czyli odwrotność tangensa, argumentu określonego w postaci współrzędnych x i y. Arcus tangens to kąt między osią x a prostą, która zawiera początek układu współrzędnych (0, 0) i punkt o współrzędnych (x, y). Kąt jest określony w radianach i należy do zakresu od $-\pi$ do π , z wyłączeniem wartości $-\pi$.	PRINT ATN2(1,2)
AVG	Zwraca wartość średnią (funkcja agregująca)	PRINT AVG()
CEILING	Zwraca najmniejszą liczbę całkowitą dla wartości gdzie liczba jest \geq od wartości	PRINT CEILING(88.222)
COUNT	Zwraca ilość elementów (funkcja agregująca)	PRINT COUNT(*)
COS	Funkcja Cos zwraca cosinus argumentu — kąt określony w radianach.	PRINT COS(23)
COT	Funkcja Cot zwraca cotangens argumentu — kąt określony w radianach.	PRINT COT(2)
DEGREES	Konwertuje radiany na stopnie	PRINT DEGREES(4)
EXP	zwraca wartość wykładniczą. Dla EXP(1) liczbę Eulera $\sim 2,71828182845905$	PRINT EXP(1)
FLOOR	Zwraca największą liczbę całkowitą dla wartości gdzie liczba \leq wartości	PRINT FLOOR(30.999)
LOG	Zwraca logarytm funkcja wykładnicza.	PRINT LOG(10)
LOG10	Zwraca logarytm funkcja wykładnicza.	PRINT LOG10
MAX	Zwraca wartość maksymalną (funkcja agregująca)	PRINT MAX()
MIN	Zwraca wartość minimalną (funkcja agregująca)	PRINT MIN()
PI	Zwraca wartość liczby Pi	PRINT PI()
POWER	Zwraca wartość liczby a do potęgi b	PRINT POWER(4,4)
RADIANS	Konwertuje kąt do radianów	PRINT RADIANS(200)
RAND	Zwraca a wartość losową	PRINT RAND()
ROUND	Zaokrągla liczbę do wskazanej liczby miejsc	PRINT ROUND(2.111,1)
SIGN	Zwraca znak liczby (dodatnia/ujemna)	PRINT SIGN(-222)
SIN	Zwraca sinus kąta	PRINT SIN(180)
SQRT	Zwraca pierwiastek kwadratowy z liczby	PRINT SQRT(100)
SQUARE	Zwraca liczbę podniesioną do kwadratu	PRINT SQUARE(3)
SUM	Zwraca wartość sumę (funkcja agregująca)	PRINT SUM()
TAN	Zwraca tangens kąta	PRINT TAN(30)
Funkcje daty i czasu		
CURRENT_TIMESTAMP	Zwraca bieżącą datę i godzinę	PRINT CURRENT_TIMESTAMP
DATEADD	Dodaje wskazany interwał czasowy do daty	PRINT DATEADD(month,4,'2018-05-06')
DATEDIFF	Zwraca ilość wskazanych interwałów czasowych między dwoma datami	PRINT DATEDIFF(month, '2018-05-06', '2019-08-06')
DATEFROMPARTS	Zwraca datę złożone ze składników (rok, miesiąc, dzień)	PRINT DATEFROMPARTS(2018,5,5)
DATENAME	Zwraca wskazaną część daty jako tekst	PRINT DATENAME(day, '2019-01-21')
DATEPART	Zwraca wskazaną część daty jako liczbę	PRINT DATEPART(day, '2019-01-21')
DAY	Zwraca dzień miesiąca z daty.	PRINT DAY('2018-05-06')
GETDATE	Zwraca bieżącą datę i czas systemowe	PRINT GETDATE()
GETUTCDATE	Zwraca bieżącą datę i czas wg UTC	PRINT GETUTCDATE()
ISDATE	Zwraca 1 jeśli wyrażenie jest datą, inaczej 0	PRINT ISDATE('2018-05-06')
MONTH	Zwraca numer miesiąca z daty.	PRINT MONTH('2018-05-06')
SYSDATETIME	Zwraca datę i godzinę wg serwera SQL.	PRINT SYSDATETIME()
YEAR	Zwraca rok z daty.	PRINT YEAR('2018-05-06')

Funkcje konwersji			
Pomagają przekształcić jeden format danych w inny			
CAST	Rzutowuje jeden typ danych na inny, w przypadku błędu przerywa skrypt.	PRINT CAST('2018-03-01' AS smalldatetime)	AS
TRY_CAST	Rzutowuje jeden typ danych na inny, w przypadku zwraca null.	PRINT CAST('2018-03-01' AS smalldatetime)	AS
CONVERT	Konwertuje jeden typ danych na inny, w przypadku błędu przerywa skrypt.	PRINT CONVERT(smalldatetime, '2018-03-01')	
TRY_CONVERT	Konwertuje jeden typ danych na inny, w przypadku zwraca null.	PRINT TRY_CONVERT(smalldatetime, '2018-03-01')	
CONVERT	Zamiana wartości „wyglądającej jak” liczba lub data w liczbę lub datę	PRINT CONVERT(date, '12-sep-18')	
PARSE	Zamiana wartości „wyglądającej jak” liczba lub data w liczbę lub datę	PRINT PARSE('12-sie-18' AS date USING 'pl-pl')	
TRY_PARSE	Zamiana wartości „wyglądającej jak” liczba lub data w liczbę lub datę	PRINT TRY_PARSE('02-sie-18' as date USING 'PL-pl')	
Funkcje specjalne			
Funkcje pomocnicze dla bazy danych, zwracają zwykle informacje o systemie			
COALESCE	Zwraca pierwszą niepustą wartość z listy	PRINT COALESCE(1,2)	
CURRENT_USER	Zwraca nazwę bieżącego użytkownika bazy danych	PRINT CURRENT_USER	
ISNULL	Podstawia wskazaną wartość za wartość pustą	PRINT ISNULL(NULL, '123')	
ISNUMERIC	Sprawdza, czy wartość jest liczbą	PRINT ISNUMERIC(123)	
NULLIF	Zwraca NULL jeśli oba wyrażenia są równe	PRINT NULLIF('abc', 'abc')	
SESSION_USER	Zwraca nazwę bieżącego użytkownika bazy danych	PRINT SESSION_USER	
SYSTEM_USER	Zwraca login bieżącego użytkownika bazy danych	PRINT SYSTEM_USER	
USER_NAME	Zwraca nazwę bieżącego użytkownika bazy danych	PRINT USER_NAME()	

12.3. Składnia SQL

Polecenie	Składnia	Polecenie	Składnia
AND / OR	SELECT Kolumna FROM tabela WHERE warunek AND OR warunek	ALTER TABLE	ALTER TABLE tabela ADD Kolumna datatype lub ALTER TABLE tabela DROP COLUMN Kolumna
AS (alias)	SELECT Kolumna AS column_alias FROM tabela lub SELECT Kolumna FROM tabela AS tabela_alias	BETWEEN	SELECT Kolumna FROM tabela WHERE Kolumna BETWEEN wartość1 AND wartość2
CREATE DATABASE	CREATE DATABASE database_name	CREATE TABLE	CREATE TABLE tabela (Kolumna1 typ_danych, Kolumna2 typ_danych)
CREATE INDEX	CREATE INDEX UNIQUE INDEX index_name ON tabela (Kolumna)	CREATE VIEW	CREATE VIEW widok AS SELECT Kolumna FROM tabela WHERE warunek
DELETE	DELETE FROM tabela WHERE warunek	DROP DATABASE	DROP DATABASE database_name
DROP INDEX	DROP INDEX tabela.index_name	DROP TABLE	DROP TABLE tabela
EXISTS	IF EXISTS (SELECT * FROM tabela WHERE warunek) BEGIN -- zadania do wykonania jeśli tak END ELSE BEGIN -- zadania do wykonania jeśli nie END	GROUP BY	SELECT Kolumna, funkcja_agregująca(Kolumna) FROM tabela WHERE Warunek GROUP BY Kolumna
HAVING	SELECT Kolumna, funkcja_agregująca (Kolumna) FROM tabela	IN	SELECT Kolumna FROM tabela

INSERT INTO	WHERE Warunek GROUP BY Kolumna HAVING funkcja_agregująca (Kolumna) = wartość INSERT INTO tabela VALUES (wartość1, wartość2...) lub INSERT INTO tabela (kolumna1, kolumna2...) VALUES (wartość1, wartość2...)	INNER JOIN	WHERE Kolumna IN (wartość1,wartość2,..) SELECT Kolumna FROM tabela1 INNER JOIN tabela2 ON tabela1.Kolumna=tabela2.Kolumna
LEFT JOIN	SELECT Kolumna FROM tabela1 LEFT JOIN tabela2 ON tabela1.Kolumna=tabela2.Kolumna	RIGHT JOIN	SELECT Kolumna FROM tabela1 RIGHT JOIN tabela2 ON tabela1.Kolumna=tabela2.Kolumna
FULL JOIN	SELECT Kolumna FROM tabela1 FULL JOIN tabela2 ON tabela1.Kolumna=tabela2.Kolumna	LIKE	SELECT Kolumna FROM tabela WHERE Kolumna LIKE wzorzec
ORDER BY	SELECT Kolumna FROM tabela ORDER BY Kolumna	SELECT	SELECT Kolumna FROM tabela
SELECT *	SELECT * FROM tabela	SELECT DISTINCT	SELECT DISTINCT Kolumna FROM tabela
SELECT INTO	SELECT * INTO tabela1 FROM tabela2	SELECT TOP	SELECT TOP ilość/procent Kolumna FROM tabela
TRUNCATE TABLE	TRUNCATE TABLE tabela	UNION	SELECT Kolumna FROM tabela1 UNION SELECT Kolumna FROM tabela2
UNION ALL	SELECT Kolumna FROM tabela1 UNION ALL SELECT Kolumna FROM tabela2	UPDATE	UPDATE tabela SET kolumna1= wartość, kolumna2= wartość WHERE Warunek
WHERE	SELECT Kolumna FROM tabela WHERE Warunek		

12.4. Operatory SQL

Operatory arytmetyczne		Operatory bitowe	
+	Dodawanie	&	bitowe AND
-	Odejmowanie		bitowe OR
*	Mnożenie	^	bitowe exclusive OR
/	Dzielenie	Operatory porównania	
%	Reszta z dzielenia	=	Równe
Operatory logiczne		>	Większe od
ALL	PRAWDA jeśli wszystkie warunki są spełnione	<	Mniejsze od
AND	PRAWDA jeśli wszystkie warunki oddzielone AND są spełnione	>=	Większe lub równe od
ANY	PRAWDA jeśli dowolny warunek jest spełniony	<=	Mniejsze lub równe od
BETWEEN	PRAWDA jeśli operator spełnia warunki w zakresie	<>	Różne od
EXISTS	PRAWDA jeśli podzapytanie zwraca rekordy	Operatory złożone	
IN	PRAWDA jeśli operator zwraca przynajmniej jeden warunek z listy	+=	Dodaj do
LIKE	PRAWDA jeśli operator jest zgodny ze wzorcem	-=	Odejmij od
NOT	Zwraca rekordy jeśli warunek nie jest prawdziwy	*=	Pomnóż przez
OR	PRAWDA jeśli dowolny warunek rozdzielony OR jest PRAWDA	/=	Podziel przez
SOME	PRAWDA jeśli dowolne podzapytanie spełnia warunek	%=	Reszta z dzielenia przez

12.5. Słowa kluczowe SQL

Słowo kluczowe	Znaczenie
----------------	-----------

--	Komentarz liniowy
/*	Komentarz blokowy
ADD	Używane z ALTER TABLE do dodania kolumny
ADD CONSTRAINT	Używane z ALTER TABLE do dodania kontroli poprawności
ALTER COLUMN	Używane z ALTER TABLE do zmiany typu danych w tabeli
ALTER TABLE	Używane do dodawania, usuwania lub modyfikacji kolumny w tabeli
ALL	Używane z WHERE lub HAVING zwraca PRAWDĘ jeśli warunki są spełnione
AND	Używane z WHERE zwraca rekordy spełniające warunki
ANY	Używane z WHERE lub HAVING zwraca PRAWDĘ jeśli podzapytania spełni warunek
AS	Używane jako alias dla nazwy kolumny lub tabeli
ASC	Porządek sortowania - rosnący
BETWEEN	Używane z WHERE do wybrania danych z podanego zakresu
CASE	Używane do tworzenia różnych wyników w oparciu o warunki
CHECK	Kontrola poprawności danych dla kolumny
CREATE DATABASE	Używane dla stworzenia nowej bazy danych
CREATE INDEX	Używane do tworzenia indeksu (dopuszczającego zduplikowane wartości)
CREATE OR REPLACE VIEW	Używane do aktualizacji widoku
CREATE TABLE	Używane podczas tworzenia tabeli
CREATE PROCEDURE	Używane podczas tworzenia procedury
CREATE UNIQUE INDEX	Używane do tworzenia indeksu (bez powtórzeń)
CREATE VIEW	Używane podczas tworzenia widoku będącego wynikiem instrukcji SELECT
DEFAULT	Warunek definiujący wartość domyślną dla kolumny
DELETE	Używane przy usuwaniu rekordów
DESC	Malejący porządek sortowania
DISTINCT	Używane z SELECT zwraca wartości bez powtórzeń
DROP COLUMN	Używane do usuwania kolumn w tabeli
DROP CONSTRAINT	Używane do usuwania reguł sprawdzania poprawności
DROP DATABASE	Używane do usuwania bazy danych
DROP INDEX	Używane przy usuwaniu indeksu
DROP TABLE	Używane do usuwania tabeli
DROP VIEW	Używane do usuwania widoku
EXEC	Uruchamia procedurę składową
EXISTS	Sprawdza istnienie rekordu w podzapytaniu
FOREIGN KEY	Klucz obcy w tabeli
FROM	Używane do określenia skąd mają zostać pobrane dane
GROUP BY	Grupowanie danych po agregacji (SUM, AVG, MIN, MAX, COUNT)
HAVING	Stosowane zamiast WHERE po wykonaniu agregacji
IN	Używane jako wielokrotny warunek w kwerendzie
INDEX	Lista rekordów w tabeli stosowana dla szybszego wyszukiwania
INNER JOIN	Zwraca rekordy posiadające wspólne wartości w obu tabelach
INSERT INTO	Używane do dodawania nowych rekordów do tabeli
INSERT INTO SELECT	Używane do kopiowania danych z jednej tabeli do drugiej
IS NULL	Używane z WHERE zwraca wartości puste
IS NOT NULL	Używane z WHERE zwraca wartości niepuste
LEFT JOIN	Zwraca wszystkie rekordy z lewej tabeli i pasujące do niej z prawej tabeli
LIKE	Używane z WHERE zwraca rekordy zgodne ze wzorcem
LIMIT	Ogranicza liczbę zwracanych rekordów
NOT	Zwraca tylko rekordy gdzie warunek nie jest spełniony
NOT NULL	Zwraca rekordy gdzie wartości są niepuste
OR	Używane z WHERE zwraca rekordy gdzie jest spełniony jeden z warunków
ORDER BY	Określa porządek sortowania
OUTER JOIN	Złączenie tabel zwracające rekordy z jednej tabeli przemnożone przez drugą tabelę
PRIMARY KEY	Klucz główny
RIGHT JOIN	Złączenie prawe pokazuje wszystkie rekordy z prawej tabeli i ich odpowiedniki z lewej
SELECT	Używane do pobierania danych z bazy danych
SELECT DISTINCT	Używane do pobrania wartości bez powtórzeń
SELECT INTO	Używane do kopiowania wyniku zapytania do nowej tabeli

SELECT TOP	Używane przy pobieraniu wartości szczytowej z wyniku kwerendy
SET	Używane z UPDATE do aktualizacji danych
TABLE	Używane przy tworzeniu, modyfikacji i kasowaniu tabel
TRUNCATE TABLE	Używane do kasowania zawartości tabeli, ale nie samej tabeli
UNION	Łączy wynik dwóch kwerend pomijając wartości zduplikowane
UNION ALL	Łączy wynik dwóch kwerend NIE pomijając wartości zduplikowanych
UNIQUE	Warunek unikalności wartości kolumny w tabeli
UPDATE	Używane do aktualizacji danych w tabeli
VALUES	Wskazuje wartości dla polecenia INSERT INTO
VIEW	Używane do tworzenia, aktualizacji o usuwania widoków
WHERE	Filtruje wyniki rezultatu zapytania

12.6. Typy danych SQL Server

Typy tekstowe – łańcuchy znaków		
Typ danych	Opis	Rozmiar max
char(n)	Znakowy o stałej długości n	8 000 znaków
varchar(n)	Znakowy o zmiennej długości n	8 000 znaków
varchar(max)	Znakowy o zmiennej długości	8 000 znaków
text	Znakowy o zmiennej długości	1 073 741 824 znaki
NCHAR	Znakowy o stałej długości n - Kodowanie Unicode	4 000 znaków
NVARCHAR	Znakowy o zmiennej długości n - Kodowanie Unicode	4 000 znaków
NVARCHAR(max)	Znakowy o zmiennej długości - Kodowanie Unicode	4 000 znaków
ntext	Znakowy o zmiennej długości - Kodowanie Unicode	536 870 912 znaków
binary(n)	Dane binarne	8 000 bajtów
varbinary	Dane binarne	8 000 bajtów
varbinary(max)	Dane binarne	2GB
image	Dane binarne	2GB
Typy całkowitoliczbowe		
bit	Liczba 0, 1 lub NULL	1 bit
tinyint	Liczba od 0 do 255	1 bajt
smallint	Liczba od -32 768 do 32 767	2 bajty
INT	Liczba od -2 147 483 648 do 2 147 483 647	4 bajty
bigint	Liczba od -9 223 372 036 854 775 808 do 9 223 372 036 854 775 807	8 bajtów
Typy zmiennoprzecinkowe (p – ilość liczb przed przecinkiem od 1 do 18, s – po przecinku, łącznie do 38 znaków)		
decimal(p,s)	Liczba dziesiętna od -10 ³⁸ +1 do 10 ³⁸ -1	5-17 bajtów
numeric(p,s)	Liczba dziesiętna od -10 ³⁸ +1 do 10 ³⁸ -1	5-17 bajtów
smallmoney	Waluta od -214 748,3648 do 214 748,3647	4 bajty
money	Waluta od -922 337 203 685 477,5808 do 922 337 203 685 477,5807	8 bajtów
float(n)	Liczba zmiennoprzecinkowa -1,79E + 308 do 1,79E + 308	4 – 8 bajtów
real	Liczba zmiennoprzecinkowa -3,40E + 38 to 3,40E + 38	4 bajty
Data i czas		
datetime	Od 1753-01-01 do 9999-12-31 z precyzją 3,33 milisekundy	8 bajtów
datetime2	Od 0001-01-01 do 9999-12-31 z precyzją 100 nanosekund	6 - 8 bajtów
smalldatetime	Od 1900-01-01 do 2076-06-06 z precyzją 1 minuty	4 bajty
date	Od 1900-01-01 do 9999-12-31 tylko data	3 bajty
time	Tylko czas z precyzją 100 nanosekund	3 - 5 bajtów
datetimeoffset	Identyczny z datetime2 ze strefą czasową	8 - 10 bajtów
timestamp	Unikalny znacznik czasu definiowany podczas każdej modyfikacji rekordu	
Pozostałe formaty		
sql_variant	Dane różne poza text, ntext i timestamp	8000 bajtów
uniqueidentifier	Identyfikator GUID	
xml	Dane XML	Max 2GB
cursor	Referencja do kursora	
table	Zestaw danych do dalszego przetwarzania	