

Bazy Danych 2 – Lab 4

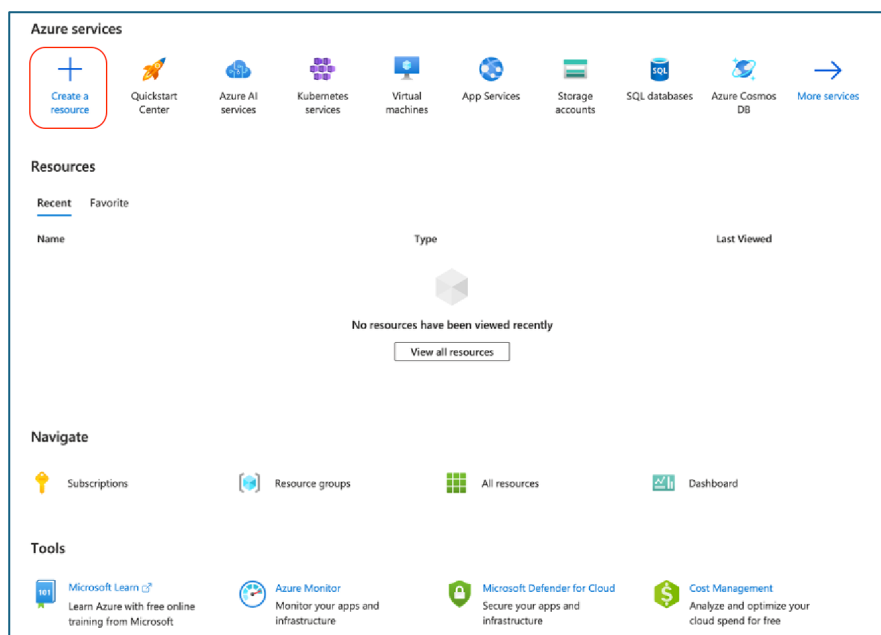
Na dzisiejszych zajęciach rozpoczniemy pracę z platformą **Microsoft Azure**, która pozwala na tworzenie i zarządzanie usługami chmurowymi, w tym relacyjnymi bazami danych SQL. Dzięki specjalnemu programowi **Azure for Students**, każdy uczestnik może uzyskać **darmowy kredyt w wysokości 100 USD**, bez potrzeby podpinania karty płatniczej.

Celem dzisiejszych ćwiczeń będzie:

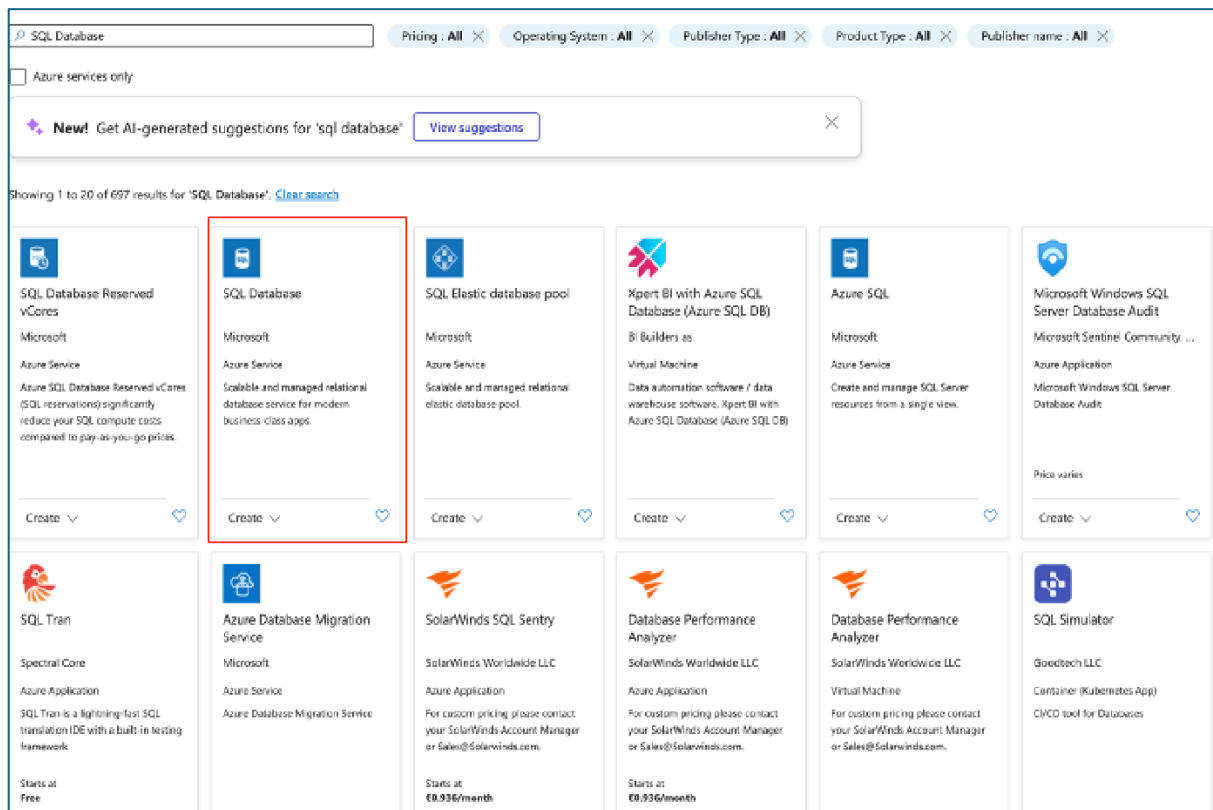
- **Założenie konta studenckiego Azure**
- **Utworzenie środowiska z przykładową bazą danych – AdventureWorks**
- **Zalogowanie się i przygotowanie do wykonywania zapytań SQL w chmurze**
- **Widoki**
- **Procedure**

Baza danych AdventureWorks to przykładowy zestaw danych udostępniony przez firmę Microsoft, który umożliwia ćwiczenie zapytań SQL w realistycznym środowisku biznesowym. Po założeniu konta i skonfigurowaniu środowiska, będziemy mogli korzystać z edytora zapytań online, bez potrzeby instalowania oprogramowania lokalnie.

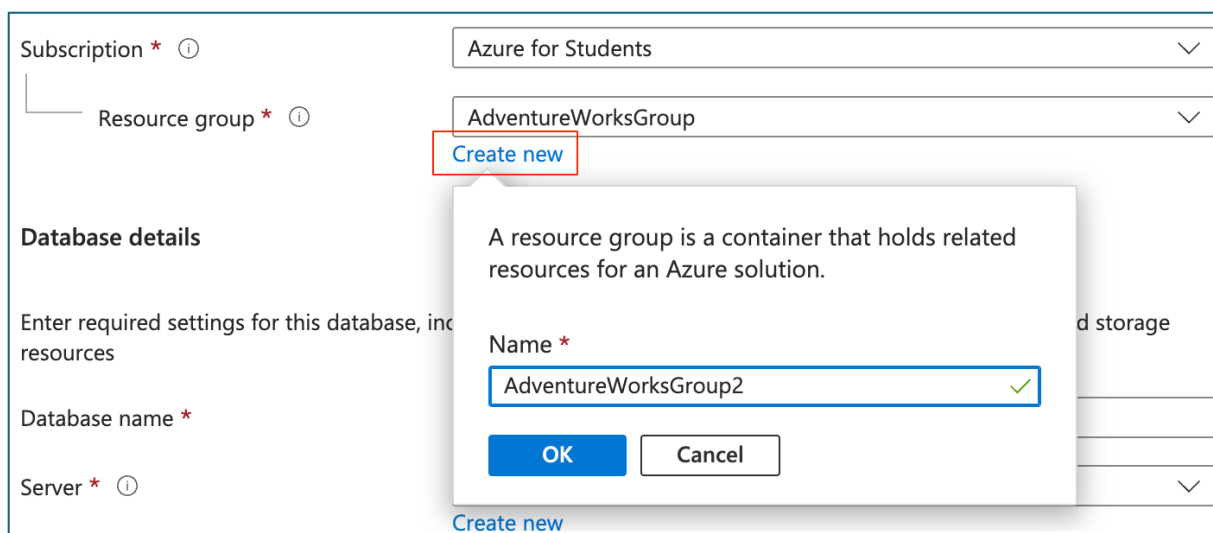
1. Tworzenie bazy danych AdventureWorks w Azure - Wejdź na stronę <https://azure.microsoft.com/en-us/free/students/> i zarejestruj się, korzystając z adresu e-mail przypisanego do uczelni. Po pomyślnej weryfikacji otrzymasz 100 USD w darmowym kredycie.
2. Zaloguj się do portalu azure <https://portal.azure.com/#home>
3. Utwórz nowy zasób - W lewym górnym rogu kliknij przycisk **“Create a resource”**, aby rozpocząć konfigurację nowego środowiska.



4. Wybierz typ zasobu – SQL - W wyszukiwarce wpisz **“SQL Database”** i kliknij wynik, który umożliwia utworzenie relacyjnej bazy danych SQL w chmurze.



5. Utwórz nową grupę zasobów (Resource Group)- Kliknij **“Create new”**, a następnie nadaj swojej grupie nazwę, np. StudentLabGroup.



6. Nazwij bazę danych

W polu **Database name** wpisz:

AdventureWorks (lub inną unikalną nazwę, jeśli baza już istnieje).

7. Utwórz nowy serwer SQL

Kliknij **“Create new server”**, a następnie:

- Wprowadź unikalną nazwę serwera (np. student-sql-lab)
- Wybierz lokalizację **Poland Central** lub najbliższą
- Utwórz login administratora i silne hasło

Server * ⓘ

bazydanychuken (Poland Central) ▼

Create new

Create SQL Database Server

Microsoft

Server details

Enter required settings for this server, including providing a name and location. This server will be created in the same subscription and resource group as your database.

Server name *

bazydanychuken ✓
database.windows.net

Location *

(Europe) Poland Central ▼

Authentication

ⓘ Azure Active Directory (Azure AD) is now Microsoft Entra ID. [Learn more](#) ⓘ

Select your preferred authentication methods for accessing this server. Create a server admin login and password to access your server with SQL authentication, select only Microsoft Entra authentication [Learn more](#) ⓘ using an existing Microsoft Entra user, group, or application as Microsoft Entra admin [Learn more](#) ⓘ , or select both SQL and Microsoft Entra authentication.

Authentication method

☐ Use Microsoft Entra-only authentication

☐ Use both SQL and Microsoft Entra authentication

☒ Use SQL authentication

Server admin login *

sqladmin ✓

Password *

..... ✓

Confirm password *

..... ✓

8. Wybierz parametry serwera - Kliknij **“Configure database”** i wybierz poziom **Basic** (wystarczający do testów i pracy laboratoryjnej).

Compute + storage * ⓘ

Basic
2 GB storage
Backup storage redundancy: **Locally-redundant backup storage**
[Configure database](#)

Configure

[Feedback](#)

Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. [Learn more](#)

SQL Database Hyperscale: Low price, high scalability, and best feature set. [Learn more](#)

Service tier: Basic (For less demanding workloads) [Compare service tiers](#)

DTUs [Compare DTU options](#)

5 (Basic)

Data max size (GB)


Backup storage redundancy

Choose how your PITR and LTR backups are replicated. Geo-restore or ability to recover from regional outage is only available when geo-redundant storage options are selected.

Backup storage redundancy ⓘ

☒ Locally-redundant backup storage

☐ Zone-redundant backup storage



Cost summary

Basic (Basic)	
Cost per DTU (in USD)	0.98
DTUs selected	x 5
ESTIMATED COST / MONTH	4.90 USD

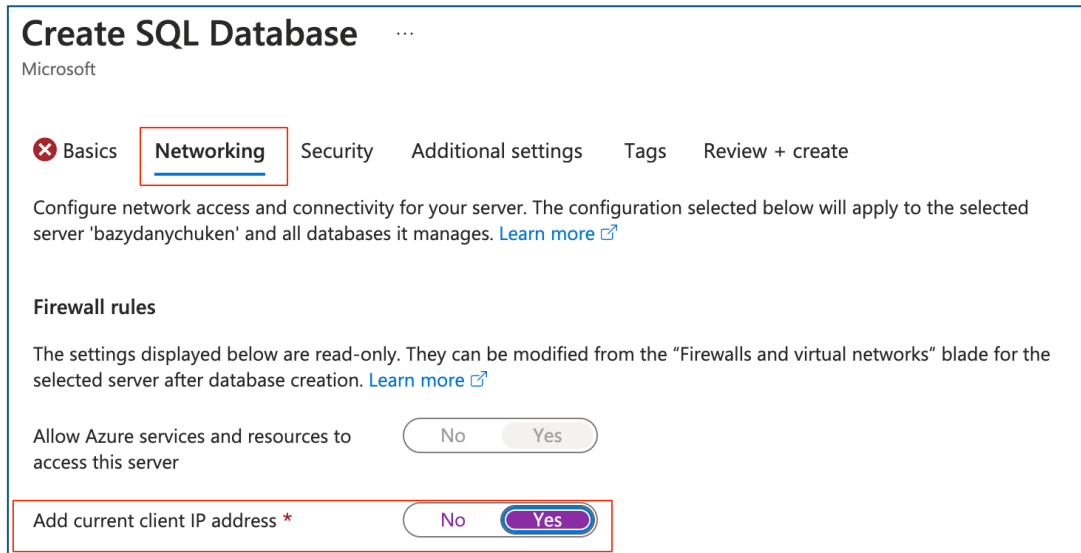
9. Skonfiguruj dostęp sieciowy (Networking)

W zakładce **Networking** zaznacz:

“Allow Azure services to access this server” – Yes

Dodaj swój aktualny adres IP do listy reguł dostępu

Pamiętaj: jeśli zmienisz komputer lub sieć, będziesz musiał dodać nowy adres IP!



Create SQL Database ...

Microsoft

✕ Basics **Networking** Security Additional settings Tags Review + create

Configure network access and connectivity for your server. The configuration selected below will apply to the selected server 'bazydanychuken' and all databases it manages. [Learn more](#)

Firewall rules

The settings displayed below are read-only. They can be modified from the "Firewalls and virtual networks" blade for the selected server after database creation. [Learn more](#)

Allow Azure services and resources to access this server ☐ No ☒ Yes

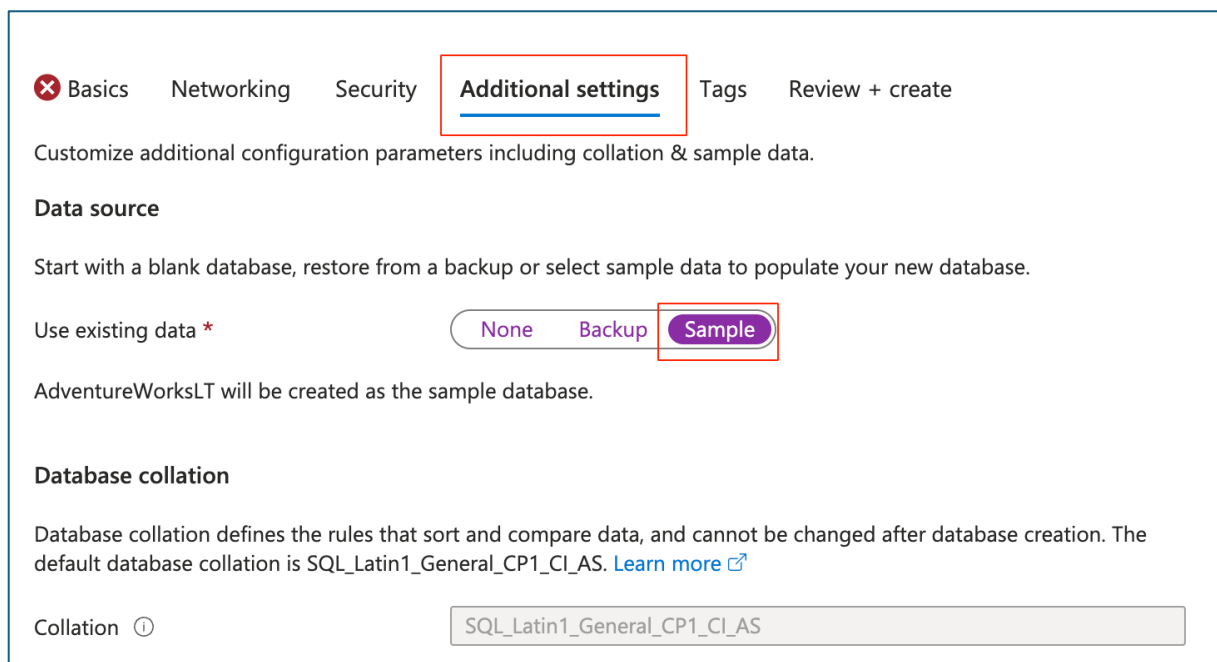
Add current client IP address * ☐ No ☒ Yes

10. Dodaj przykładowe dane

W zakładce **Additional Settings** wybierz:

Use existing data → Sample

Dzięki temu baza zostanie zainicjowana przykładowymi danymi (AdventureWorksLT).



✕ Basics Networking Security **Additional settings** Tags Review + create

Customize additional configuration parameters including collation & sample data.

Data source

Start with a blank database, restore from a backup or select sample data to populate your new database.

Use existing data * ☐ None ☐ Backup ☒ Sample

AdventureWorksLT will be created as the sample database.

Database collation

Database collation defines the rules that sort and compare data, and cannot be changed after database creation. The default database collation is SQL_Latin1_General_CP1_CI_AS. [Learn more](#)

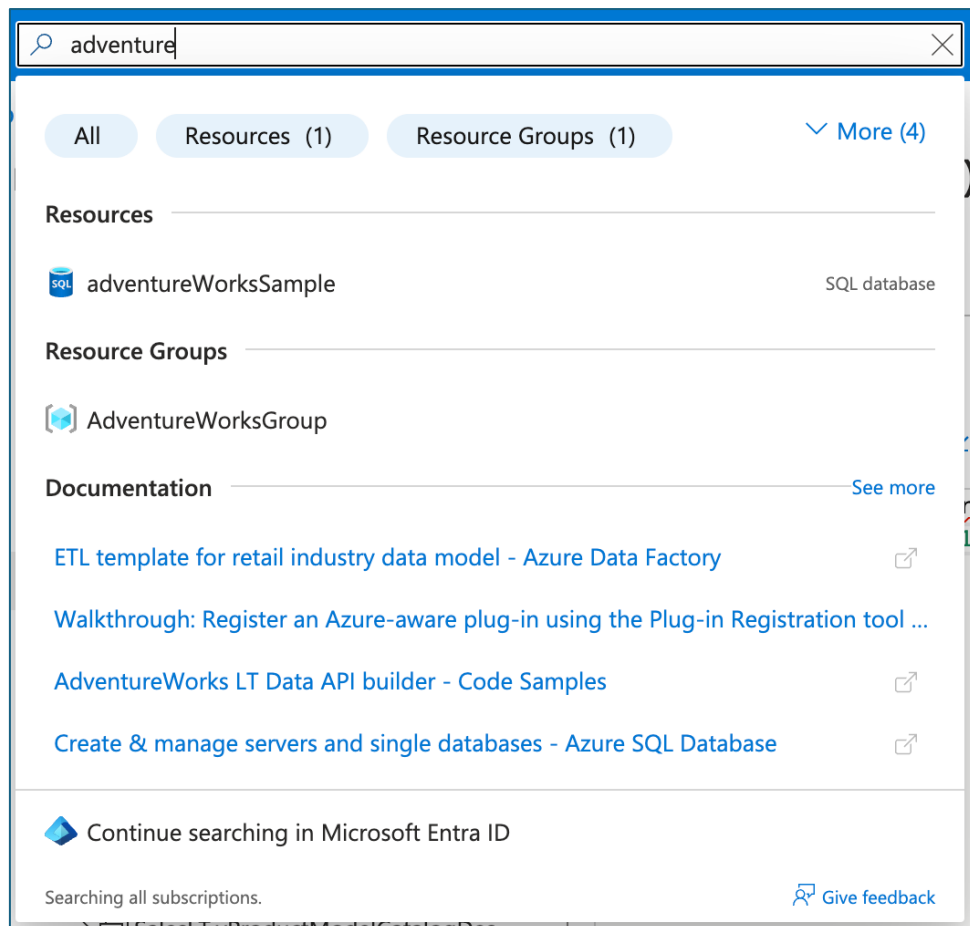
Collation ⓘ SQL_Latin1_General_CP1_CI_AS

11. Utwórz bazę danych

Kliknij **“Review + Create”**, a następnie **“Create”**, aby rozpocząć wdrażanie zasobu. Proces ten może potrwać kilka minut.

12. Odszukaj swoją bazę danych

Po zakończeniu wdrożenia przejdź do sekcji **“SQL databases”** i kliknij na nazwę swojej bazy (np. AdventureWorks).



13. Zaloguj się do edytora zapytań

Przejdź do zakładki **“Query editor (preview)”**

Zaloguj się za pomocą danych administratora serwera SQL

Teraz możesz pisać i wykonywać zapytania SQL bezpośrednio w przeglądarce!

adventureWorksSample (bazydanychuken/adventureWorksSample) | Query editor (preview) ☆ ...

SQL database

Search 🔍 ◊ << 👤 Login + New Query ↗ Open query 🗨 Feedback 📖 Getting started

Overview
Activity log
Tags
Diagnose and solve problems
Query editor (preview)
Mirror database in Fabric (preview)
Resource visualizer
> Settings
> Data management
> Integrations
> Power Platform
> Security
> Intelligent performance
> Monitoring
> Automation
> Help

Query editor (preview) is a tool to run SQL queries against Azure SQL Database in the Azure portal. It is designed for lightweight querying and object exploration in your database. For more information and troubleshooting, [Learn more](#)

SQL

Welcome to SQL Database Query Editor

SQL server authentication

Login *

Password *

OK

Microsoft Entra authentication

[Continue as 01186035@pw.edu.pl](#)

OR

Widoki

<https://www.tutorialspoint.com/sql/sql-using-views.htm>

Widoki to nic innego jak zapytania SQL, które zostały zapisane i są przechowywane w bazie danych pod pewną nazwą. W rzeczywistości jest to tabela, która została wygenerowana predefiniowanym zapytaniem SQL.

Widok może zawierać wszystkie wiersze danej tabeli lub interesującą nas część. Może zostać utworzony na bazie jednej lub wielu tabel. Wszystko zależy od samego zapytania SQL i efektu, który chcemy uzyskać.

Widoki są typem wirtualnych tabel na których możemy wykonywać szereg operacji takich jak:

- strukturyzowanie danych w przystępny sposób;
- ograniczenie (widocznych) danych do jedynie niezbędnych;
- łączenie wyników z różnych tabel, które mogą stanowić podstawę do generowania raportów.

Tworzenie widoków

Bazowym poleceniem pozwalającym na tworzenie widoków jest CREATE VIEW.

Podstawowe zapytanie przy wykorzystaniu powyższego polecenia przybiera poniższą postać:

```
CREATE VIEW nazwa_widoku AS  
SELECT kolumna1, kolumna2, ... kolumnaN  
FROM nazwa_tabeli  
WHERE (warunki)
```

Możemy tworzyć skomplikowane zapytania, łączyć wiele tabel czy używać podzapytań tak jak w przypadku normalnego SELECT'A.

Dysponując bazą danych **AdventureWorksLT**, przygotowujemy zapytanie SQL prezentujące informacje o klientach, ich adresach oraz historii zamówień. Dzięki temu możemy analizować, kto złożył zamówienie, kiedy je złożono, ile kosztowało oraz jaki adres został przypisany do klienta. Zapytanie korzysta z danych z kilku tabel, połączonych ze sobą na podstawie relacji. To pozwala na stworzenie spójnego widoku łączącego informacje z różnych obszarów. Dobrym podejściem jest najpierw przygotować zapytanie zwracające oczekiwane dane, a następnie – na jego podstawie – stworzyć widok (ang. *view*), który ułatwi przyszły dostęp do tych informacji i ich analizę.

```
SELECT
  c.CustomerID,
  c.FirstName + ' ' + c.LastName AS CustomerName,
  soh.SalesOrderID,
  soh.OrderDate,
  soh.TotalDue,
  a.AddressLine1,
  a.AddressLine2,
  a.City,
  a.StateProvince,
  a.PostalCode,
  a.CountryRegion
FROM SalesLT.Customer AS c
JOIN SalesLT.SalesOrderHeader AS soh
  ON c.CustomerID = soh.CustomerID
JOIN SalesLT.CustomerAddress AS ca
  ON c.CustomerID = ca.CustomerID
JOIN SalesLT.Address AS a
  ON ca.AddressID = a.AddressID;
```

W drugim kroku dokonamy utworzenia widoku na podstawie powyższego polecenia:

```
CREATE VIEW vw_CustomerOrderInfo AS
SELECT
    c.CustomerID,
    c.FirstName + ' ' + c.LastName AS CustomerName,
    soh.SalesOrderID,
    soh.OrderDate,
    soh.TotalDue,
    a.AddressLine1,
    a.AddressLine2,
    a.City,
    a.StateProvince,
    a.PostalCode,
    a.CountryRegion
FROM SalesLT.Customer AS c
JOIN SalesLT.SalesOrderHeader AS soh
    ON c.CustomerID = soh.CustomerID
JOIN SalesLT.CustomerAddress AS ca
    ON c.CustomerID = ca.CustomerID
JOIN SalesLT.Address AS a
    ON ca.AddressID = a.AddressID;
```

Jak wygląda odczyt danych z widoków? W identyczny sposób jak przy zwykłym zapytaniu. Zamiast nazwy tabeli podajemy nazwę zdefiniowanego widoku:

```
SELECT * FROM vw_CustomerOrderInfo;
```

Dodawanie/Aktualizowanie/Kasowanie rekordów

Temat ten wymaga nieco głębszej analizy. Jest jednocześnie mało skomplikowany w przypadku utworzenia widoku na bazie jednej tabeli. W takim scenariuszu wszelkie operacje DML (dodawanie, kasowanie, aktualizacja) dokonane na widoku zostaną automatycznie przeniesione to tabeli bazowej.

Utwórz widok:

```
CREATE VIEW CustomerWithAddress AS

SELECT

    c.CustomerID,

    c.FirstName,

    c.LastName,

    a.City

FROM SalesLT.Customer c

JOIN SalesLT.CustomerAddress ca ON c.CustomerID = ca.CustomerID

JOIN SalesLT.Address a ON ca.AddressID = a.AddressID;
```

Jeżeli spróbujesz dokonać usunięcia rekordu z widoku przy pomocy przykładowego polecenia:

```
DELETE FROM CustomerWithAddress

WHERE CustomerID = 1;
```

Zobaczysz poniższy komunikat: Failed to execute query. Error: View or function 'CustomerWithAddress' is not updatable because the modification affects multiple base tables.

W tym miejscu trzeba jeszcze wspomnieć o klauzuli WITH CHECK OPTION. Jej użycie pozwala upewnić się, że polecenia INSERT oraz UPDATE spełniają warunek/warunki określone w definicji widoku – w przeciwnym wypadku zostanie zwrócony błąd.

Dodaj nowy widok utworzony na bazie tabeli HumanResources.Department który będzie zwracał tylko nazwy departamentów zaczynające się od litery 'P':

```
CREATE VIEW AddressesFromL AS  
  
SELECT * FROM SalesLT.Address  
  
WHERE City LIKE 'L%'  
  
WITH CHECK OPTION;
```

Dodając WITH CHECK OPTION nakładamy ograniczenie na instrukcje dodawania i aktualizowania rekordów. Jeżeli ten warunek będzie spełniony nowy rekord zostanie dodany/zaktualizowany zarówno w widoku jak i tabeli bazowej:

```
-- pamiętajcie również o ograniczeniach nałożonych na tabelę bazową  
  
INSERT INTO AddressesFromL (AddressLine1, City, StateProvince, CountryRegion,  
PostalCode)  
  
VALUES ('Testowa 1', Krakow, 'Merseyside', 'UK', 'L1 2AB');
```

Wykonanie polecenia niespełniającego warunku zwróci błąd:

```
Failed to execute query. Error: The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that specifies WITH CHECK OPTION and one or more rows resulting from the operation did not qualify under the CHECK OPTION constraint. The statement has been terminated.
```

Kasowanie widoków

Po zakończonych próbach z widokami warto zrobić porządki w bazie danych i je skasować – jeżeli nie są nam potrzebne. W tym celu posłużymy się bardzo prostym poleceniem:

```
DROP VIEW nazwa_widoku
```

Dla naszego widoku, na którym eksperymentowaliśmy polecenie to przyjmie postać:

```
DROP VIEW AddressesFromL
```

Procedury

https://www.w3schools.com/sql/sql_stored_procedures.asp

<https://www.tutorialspoint.com/sql/sql-stored-procedures.htm>

Procedury składowane to nic innego jak przygotowany kod **SQL**, który może być wielokrotnie używany. Jeżeli więc mamy zapytanie, które używane jest wielokrotnie warto zapisać je jako procedurę składowaną, którą będzie gotowa do wielokrotnego użytku za każdym razem, kiedy będzie potrzebna.

Oprócz uruchamiania wielokrotnie tego samego kodu **SQL** będzie istniała możliwość przekazania parametrów do zapisanej procedury – w zależności od przekazanych parametrów wynik działania takiej procedury będzie inny.

Tworzenie prostej procedury

Zanim utworzysz procedurę składowaną musisz wiedzieć jaki jest wynik końcowy, niezależnie czy wybierasz dane, czy chcesz je wstawić itd.

W tym prostym przykładzie pobierzemy wszystkie dane z tabeli **Person.Address**

```
SELECT * FROM SalesLT.Address;
```

Możemy teraz przejść do utworzenia pierwszej procedury:

```
CREATE PROCEDURE dbo.uspGetAddress  
  
AS  
  
BEGIN  
  
    SELECT * FROM SalesLT.Address;  
  
END;
```

Wynikiem działania powyższej procedury jest zwrócenie wszystkich danych ze wspomnianej wyżej tabeli. Wywołanie wygląda w następujący sposób:

```
EXEC dbo.uspGetAddress;  
  
-- lub:  
  
EXEC uspGetAddress;
```

Podczas tworzenia procedury można użyć polecenia **CREATE PROCEDURE** lub **CREATE PROC**. Po nazwie procedury należy użyć słowa kluczowego **AS** a reszta to zwykły kod **SQL**.

Warto mieć na uwadze, że w procedurze składowanej nie można używać słowa kluczowego **GO**. Kiedy kompilator **SQL** zauważy to słowo kluczowe uzna, że jest to koniec procedury.

Używanie zmiany kontekstu bazy danych również nie jest dopuszczalne: **USE dbName**. Powodem jest to, że część procedury byłaby traktowana jako osobny pakiet a z definicji jest to zbiór jednej serii poleceń.

Tworzenie procedury z parametrami

Prawdziwą potęgą procedur składowanych jest możliwość przekazywania parametrów i oczekiwania różnych wyników – związane jest to z odpowiednim przygotowaniem zapytań.

Pojedynczy parametr

W tym przykładzie przygotujemy zapytanie do tabeli **Person.Address**, które będzie nam zwracało wyniki ograniczone jedynie do konkretnego miasta. Wymaga to zatem przekazania do procedury jednego, dodatkowego parametru:

```
CREATE PROCEDURE dbo.uspGetAddressByCity  
  
    @City NVARCHAR(30)  
  
AS  
  
BEGIN  
  
    SELECT *  
  
    FROM SalesLT.Address  
  
    WHERE City = @City;  
  
END;
```

W tym przypadku wywołanie przedstawia się w następujący sposób:

```
EXEC dbo.uspGetAddressByCity @City = 'Dallas';
```

Możemy oczywiście dowolnie przygotować naszą instrukcję. Na poniższym przykładzie można zobaczyć modyfikację, która daje użytkownikom punkt wyjścia do wyszukiwanych danych:

```
CREATE PROCEDURE dbo.uspGetAddressByCity2  
  
    @City NVARCHAR(30)  
  
AS  
  
BEGIN  
  
    SELECT *  
  
    FROM SalesLT.Address  
  
    WHERE City LIKE @City + '%';  
  
END;
```

Wywołanie procedury wygląda dokładnie tak jak w powyższym przypadku, przy czym wyniki będą się znacznie różniły.

Spróbuj wykonać polecenie:

```
EXEC dbo.uspGetAddressByCity2 @City = 'L'; -- zwróci np. 'London', 'Leeds'
```

W obu przypadkach silnik bazy danych zakłada, że parametr zostanie przekazany. Jeżeli nie, można spodziewać się poniższego komunikatu błędu:

```
EXEC dbo.uspGetAddressByCity2;  
Failed to execute query. Error: Procedure or function 'uspGetAddressByCity2' expects  
parameter '@City', which was not supplied.
```


Domyślna wartość parametru

W większości przypadków dobrą praktyką jest przekazywanie wartości **NULL**, nie zawsze jednak jest to możliwe. W poniższym można zobaczyć jak przekazać domyślną wartość tak aby nie było konieczności określania wartości dla parametru oczekiwanego przez procedurę. Jeżeli przygotujemy i uruchomimy taką procedurę nie zostaną zwrócone żadne dane, ponieważ zapytanie będzie poszukiwało wartości **NULL** dla kolumny **City**:

```
CREATE PROCEDURE dbo.uspGetAddressNullParam
    @City NVARCHAR(30) = NULL
AS
BEGIN
    SELECT *
    FROM SalesLT.Address
    WHERE City = @City;
END;
```

Możemy również wprowadzić pewną modyfikację polegającą na użyciu sprawdzenia **ISNULL**. W takim wypadku, jeżeli zostanie przekazana jakaś wartość pozwoli ona na zwrócenie określonych wyników. Jeżeli parametr nie zostanie przekazany dojdzie do zwrócenia wszystkich rekordów. Warto mieć na uwadze fakt, że w przypadku, kiedy kolumna **City** posiada wartość **NULL** to taka wartość nie zostanie włączona w wynik operacji. Będzie należało dodać sprawdzenie **ISNULL** na tej kolumnie:

```
CREATE PROCEDURE dbo.uspGetAddressNullParam
    @City NVARCHAR(30) = NULL
AS
BEGIN
    SELECT *
    FROM SalesLT.Address
    WHERE City = ISNULL(@City, City);
END;
```

Wiele parametrów

Przekazywanie wielu parametrów nie jest skomplikowane. Wystarczy podać każdy parametr oraz jego typ oddzielone od siebie przecinkiem tak jak zostało to pokazane poniżej:

```
CREATE PROCEDURE dbo.uspGetAddressMultipleParams
    @City NVARCHAR(30) = NULL,
    @AddressLine1 NVARCHAR(60) = NULL
AS
BEGIN
    SELECT *
    FROM SalesLT.Address
    WHERE City = ISNULL(@City, City)
        AND AddressLine1 LIKE '%' + ISNULL(@AddressLine1, AddressLine1) + '%';
END;
```

Następnie:

```
EXEC dbo.uspGetAddressMultipleParams @City = 'London', @AddressLine1 =
'Street';

EXEC dbo.uspGetAddressMultipleParams @City = 'Seattle'; -- tylko po mieście

EXEC dbo.uspGetAddressMultipleParams; -- bez parametrów, zwróci wszystko
```

Zwracanie parametrów z procedur

W poprzednich podrozdziałach skupiliśmy się na przekazywaniu parametrów do procedur. Odwrotną sytuacją jest przekazywanie wartości parametrów z procedury składowanej. Jedynym z przykładów może być wywołanie kolejnej procedury, która nie zwraca żadnych danych, ale zwraca wartości parametrów, które mogą być użyte przez inną procedurę.

Definiowanie takich parametrów jest bardzo podobne do parametrów wejściowych z taką różnicą, że musimy użyć słowa kluczowego **OUTPUT**. Zamiennie można używać słowa kluczowego **OUT**:

```
CREATE PROCEDURE dbo.uspGetAddressCount
    @City NVARCHAR(30) = NULL,
    @AddressCount INT OUTPUT
AS
BEGIN
    SELECT @AddressCount = COUNT(*)
    FROM SalesLT.Address
    WHERE City = ISNULL(@City, City);
END;
```

Wywołanie takiej procedury wygląda w następujący sposób:

```
DECLARE @AddressCount INT;

EXEC dbo.uspGetAddressCount

    @City = 'London',

    @AddressCount = @AddressCount OUTPUT;

SELECT @AddressCount AS AddressCount;
```

Powyższa procedura może być również wywołana bez podawania nazw parametrów:

```
DECLARE @AddressCount INT;
EXEC dbo.uspGetAddressCount 'London', @AddressCount OUTPUT;
SELECT @AddressCount AS AddressCount;
```

Zadania widoki

Zadanie 1: Widok z podstawowego zapytania

Stwórz widok wyświetlający tylko podstawowe dane klientów. Utwórz widok o nazwie vw_CustomersBasic pokazujący kolumny: CustomerID, FirstName, LastName, CompanyName z tabeli SalesLT.Customer.

Zadanie 2: Widok z filtrowaniem

Pokazać klientów, którzy mają firmowy adres e-mail (czyli zawierający @company.com). Utwórz widok vw_CompanyEmailCustomers, który pokazuje klientów, których EmailAddress zawiera ciąg @company.com.

Zadanie 3: Widok z JOIN – klienci i ich adresy

Połączyć dane klientów i adresów w jeden widok. Utwórz widok vw_CustomersWithAddress, który zawiera CustomerID, FirstName, LastName, City, CountryRegion Skorzystaj z table SalesLT.Customer, SalesLT.CustomerAddress, SalesLT.Address

Zadanie 4: Widok z aliasami i sortowaniem

Pokazać produkty z opisem i ceną, posortowane malejąco po cenie. Utwórz widok vw_ProductsSorted, zawierający ProductID, Name, ListPrice, StandardCost Nadaj kolumnom aliasy (AS) i posortuj wynik malejąco po ListPrice. W widokach ORDER BY działa tylko przy TOP, ale do ćwiczeń można to zostawić lub usunąć sortowanie.

Zadanie 5: Widok z WITH CHECK OPTION

Ograniczyć wstawianie danych tylko do określonych warunków. Utwórz widok vw_AddressesInLondon, który pokazuje tylko adresy z miasta „London”. Użyj WITH CHECK OPTION, aby uniemożliwić dodanie rekordu z innym miastem.

Zadanie 6: Widok klientów z brakującym numerem telefonu

Pokazać, jak filtrować rekordy z wartościami NULL. Utwórz widok vw_CustomersWithoutPhone, który pokazuje klientów bez podanego numeru telefonu (Phone IS NULL).

Zadanie 7: Widok z obliczoną kolumną zysku

Dodanie kolumny obliczeniowej ($ListPrice - StandardCost$).bUtwórz widok vw_ProductsWithMargin, który zawiera: ProductID, Name, ListPrice, StandardCost oraz nową kolumnę ProfitMargin.

Zadanie 8: Widok z warunkiem logicznym OR

Pokazać produkty drogie lub mające wartość 0 (np. darmowe lub błędnie wycenione). Utwórz widok vw_ProductsPriceCheck, który pokazuje produkty, gdzie $ListPrice > 1000$ **lub** $ListPrice = 0$.

Zadanie 9: Widok z funkcją TOP

Przećwiczenie ograniczania wyników. Utwórz widok vw_Top10Products, który pokazuje 10 najdroższych produktów.

Zadanie 10: Widok klientów z zamówieniami (JOIN)

Pokazać połączenie danych klientów i zamówień. Utwórz widok vw_CustomersWithOrders, który łączy SalesLT.Customer i SalesLT.SalesOrderHeader. Widok powinien zawierać: CustomerID, FirstName, LastName, SalesOrderID, OrderDate, TotalDue.

Zadania procedury

Zadanie 1: Prosta procedura bez parametrów

Utwórz procedurę usp_GetAllCustomers, która zwraca wszystkie dane z tabeli SalesLT.Customer.

Zadanie 2: Procedura z 1 parametrem wejściowym

Utwórz procedurę usp_GetAddressesByCity, która przyjmuje nazwę miasta jako parametr i zwraca adresy z tego miasta.

Zadanie 3: Procedura z domyślnym parametrem i ISNULL

Utwórz procedurę usp_GetAddressesOptionalCity, która przy braku podanego miasta zwróci wszystkie rekordy.

Zadanie 4: Procedura z parametrem OUTPUT

Utwórz procedurę usp_GetProductCountByColor, która przyjmuje kolor produktu i zwraca liczbę produktów o tym kolorze w zmiennej OUTPUT.

Zadanie 5: Procedura z więcej niż jednym parametrem

Utwórz procedurę usp_GetProductsByColorAndPrice, która przyjmuje dwa parametry:

@Color – kolor produktu

@MaxPrice – maksymalna cena

I zwraca produkty spełniające oba warunki.

Zadanie 6: Wyszukiwanie z LIKE (częściowe dopasowanie)

Utwórz procedurę usp_SearchProductByName, która przyjmuje fragment nazwy produktu jako parametr i zwraca wszystkie pasujące produkty.

Zadanie 7: Lista zamówień klienta po ID

Utwórz procedurę usp_GetOrdersByCustomer, która przyjmuje @CustomerID i zwraca wszystkie zamówienia tego klienta z tabeli SalesLT.SalesOrderHeader.

Zadanie 8: Top N najdroższych produktów

Utwórz procedurę usp_GetTopProducts, która przyjmuje @TopCount i zwraca tyle najdroższych produktów (ListPrice).

Zadanie 9: Agregacja z warunkiem – SUMA dla klienta

Utwórz procedurę usp_GetTotalSalesByCustomer, która przyjmuje @CustomerID i zwraca sumę wszystkich jego zamówień (TotalDue).

Zadanie 10: Produkty w podanym przedziale cenowym

Utwórz procedurę usp_GetProductsByPriceRange, która przyjmuje dwa parametry @MinPrice i @MaxPrice i zwraca produkty w tym zakresie.