Dynamic SQL and SQL Injection Security Guide

This guide is based on the practical examples demonstrated by Jeff Honsowetz during the SQL Fundamentals session at Data Hikers Business Technologist Online Meetup. For hands-on practice, visit https://dbfiddle.uk/ and create a safe testing environment.

What is Dynamic SQL?

Dynamic SQL is **SQL code that's constructed and executed on the fly during runtime, rather than being hardcoded.** It's like writing a query where the structure—tables, columns, conditions—can change based on user input, application logic, or external data. For example, instead of a fixed query like SELECT * FROM users WHERE age = 30, you might build a query where the column, table, or filter value is determined at runtime, like SELECT * FROM $table WHERE $column = $value.

It's powerful for scenarios requiring flexibility, such as generating reports with varying filters or handling dynamic schemas. However, it's prone to SQL injection if parameters aren't sanitized properly, so always use parameterized queries or prepared statements to stay secure. It also can be harder to debug and optimize compared to static SQL.

What is SQL Injection Attack?

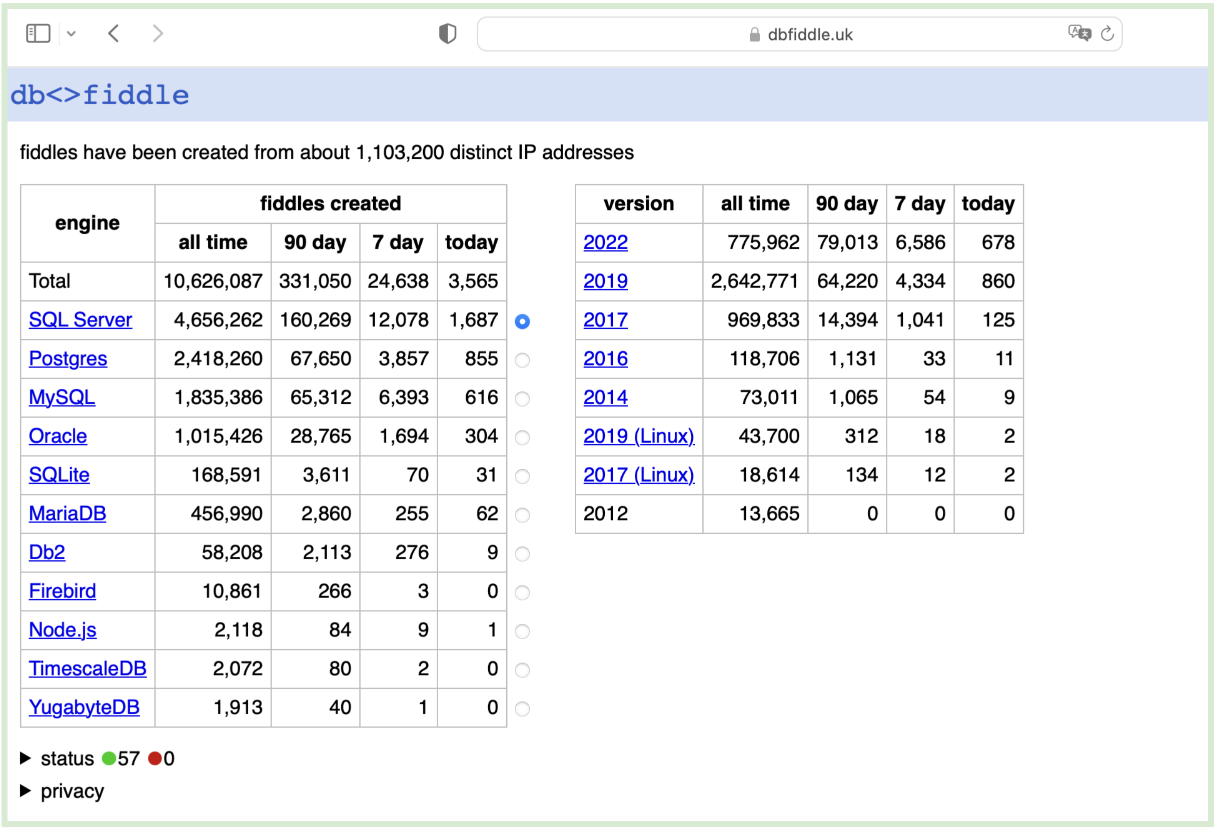
SQL Injection (SQLi) is a **cyberattack where malicious SQL statements are inserted into an entry field for execution by a database.** It's a code injection technique that targets data-driven applications, most commonly web applications.

The core vulnerability arises when an application constructs database queries using unvalidated or improperly sanitized user input. Instead of treating the user input as data, the application mistakenly interprets parts of it as executable SQL code.

Testing Your Knowledge

Safe Practice Environment

Step 1. You can safely test these concepts using dbfiddle.uk:



dbfiddle.uk

db<>fiddle

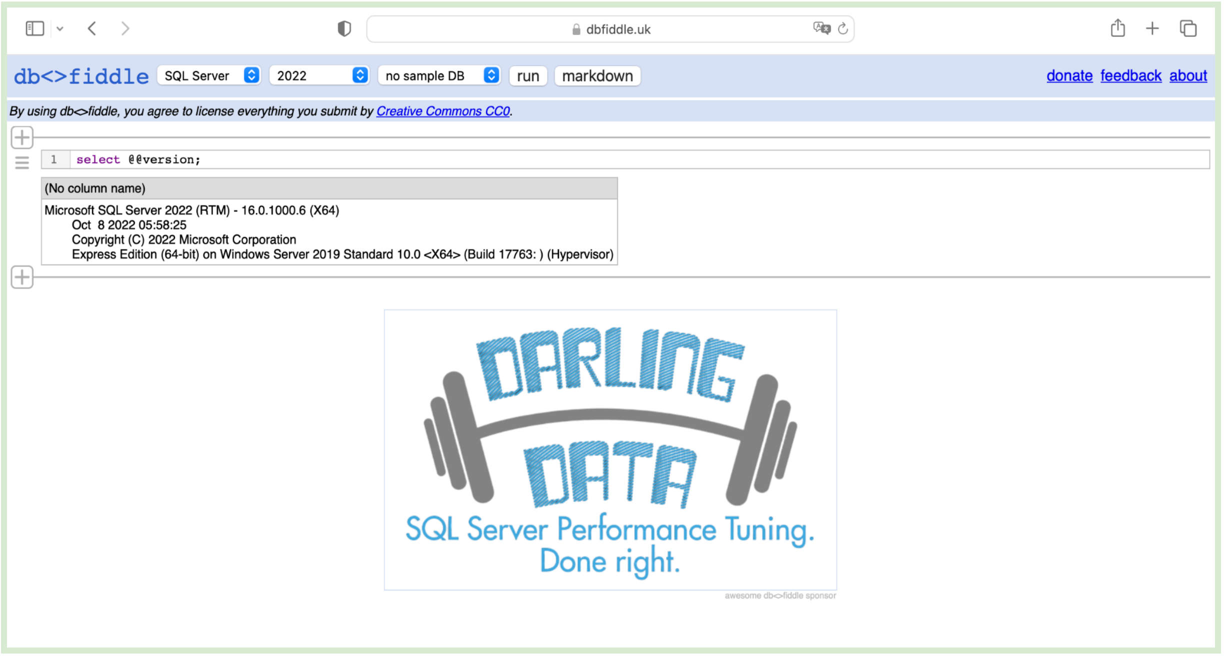
fiddles have been created from about 1,103,200 distinct IP addresses

engine	fiddles created				
	all time	90 day	7 day	today	
Total	10,626,087	331,050	24,638	3,565	
SQL Server	4,656,262	160,269	12,078	1,687	●
Postgres	2,418,260	67,650	3,857	855	○
MySQL	1,835,386	65,312	6,393	616	○
Oracle	1,015,426	28,765	1,694	304	○
SQLite	168,591	3,611	70	31	○
MariaDB	456,990	2,860	255	62	○
Db2	58,208	2,113	276	9	○
Firebird	10,861	266	3	0	○
Node.js	2,118	84	9	1	○
TimescaleDB	2,072	80	2	0	○
YugabyteDB	1,913	40	1	0	○

version	all time	90 day	7 day	today
2022	775,962	79,013	6,586	678
2019	2,642,771	64,220	4,334	860
2017	969,833	14,394	1,041	125
2016	118,706	1,131	33	11
2014	73,011	1,065	54	9
2019 (Linux)	43,700	312	18	2
2017 (Linux)	18,614	134	12	2
2012	13,665	0	0	0

▶ status 🟢57 🔴0
▶ privacy

You may use any engine and any version, but for the practice purposes choose SQL Server.



dbfiddle.uk

db<>fiddle SQL Server 2022 no sample DB run markdown donate feedback about

By using db<>fiddle, you agree to license everything you submit by Creative Commons CC0.

```
select @@version;
```

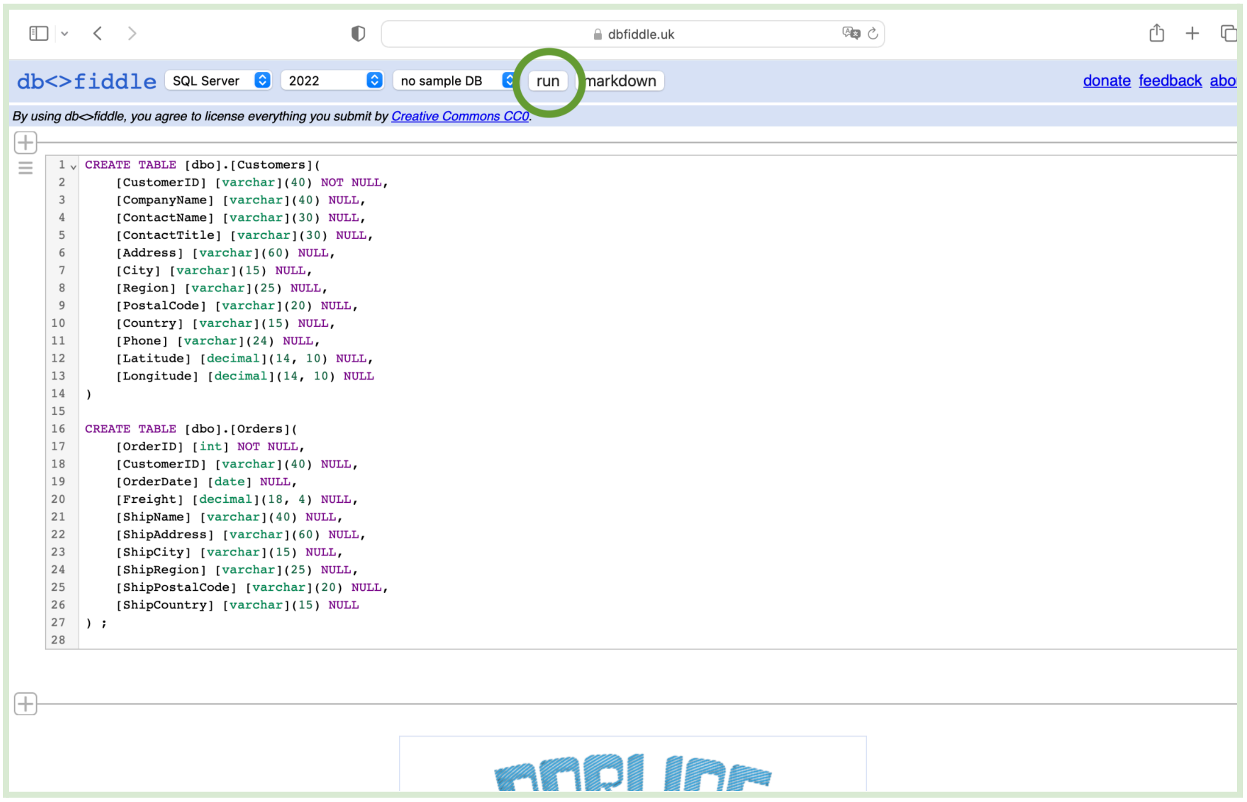
```
(No column name)
Microsoft SQL Server 2022 (RTM) - 16.0.1000.6 (X64)
    Oct  8 2022 05:58:25
    Copyright (C) 2022 Microsoft Corporation
    Express Edition (64-bit) on Windows Server 2019 Standard 10.0 <X64> (Build 17763: ) (Hypervisor)
```

DARLING DATA
SQL Server Performance Tuning. Done right.
awesome db<>fiddle sponsor

Step 2. Create a Sample Database:

```
CREATE TABLE [dbo].[Customers](

		[CustomerID] [varchar](40) NOT NULL,
		[CompanyName] [varchar](40) NULL,
		[ContactName] [varchar](30) NULL,
		[ContactTitle] [varchar](30) NULL,
		[Address] [varchar](60) NULL,
		[City] [varchar](15) NULL,
		[Region] [varchar](25) NULL,
		[PostalCode] [varchar](20) NULL,
		[Country] [varchar](15) NULL,
		[Phone] [varchar](24) NULL,
		[Latitude] [decimal](14, 10) NULL,
		[Longitude] [decimal](14, 10) NULL
)

CREATE TABLE [dbo].[Orders](
		[OrderID] [int] NOT NULL,
		[CustomerID] [varchar](40) NULL,
		[OrderDate] [date] NULL,
		[Freight] [decimal](18, 4) NULL,
		[ShipName] [varchar](40) NULL,
		[ShipAddress] [varchar](60) NULL,
		[ShipCity] [varchar](15) NULL,
		[ShipRegion] [varchar](25) NULL,
		[ShipPostalCode] [varchar](20) NULL,
		[ShipCountry] [varchar](15) NULL
) ;
```

Step 3. Insert the SQL code into the batch (field) and click "run"



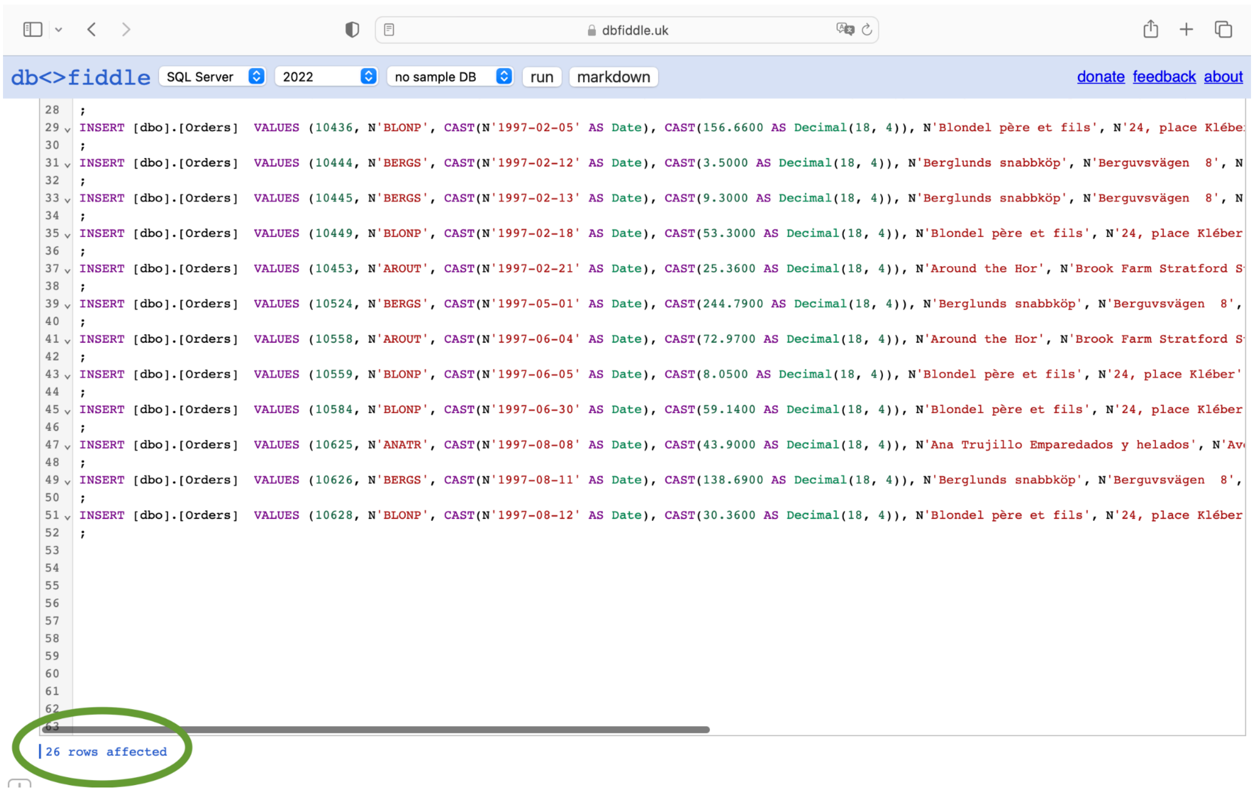
By using db<>fiddle, you agree to license everything you submit by Creative Commons CC0.

```
CREATE TABLE [dbo].[Customers](
    [CustomerID] [varchar](40) NOT NULL,
    [CompanyName] [varchar](40) NULL,
    [ContactName] [varchar](30) NULL,
    [ContactTitle] [varchar](30) NULL,
    [Address] [varchar](60) NULL,
    [City] [varchar](15) NULL,
    [Region] [varchar](25) NULL,
    [PostalCode] [varchar](20) NULL,
    [Country] [varchar](15) NULL,
    [Phone] [varchar](24) NULL,
    [Latitude] [decimal](14, 10) NULL,
    [Longitude] [decimal](14, 10) NULL
)

CREATE TABLE [dbo].[Orders](
    [OrderID] [int] NOT NULL,
    [CustomerID] [varchar](40) NULL,
    [OrderDate] [date] NULL,
    [Freight] [decimal](18, 4) NULL,
    [ShipName] [varchar](40) NULL,
    [ShipAddress] [varchar](60) NULL,
    [ShipCity] [varchar](15) NULL,
    [ShipRegion] [varchar](25) NULL,
    [ShipPostalCode] [varchar](20) NULL,
    [ShipCountry] [varchar](15) NULL
) ;

```

Step 4. Insert Sample Data

```
INSERT [dbo].[Customers] VALUES (N'ALFKI', N'Alfreds Futterkiste', N'Alfred Futter', N'Sales Representative', N'Obere Str. 57', N'Berlin', NULL, N'12209', N'Germany', N'030-0074321', CAST(52.5316770000 AS Decimal(14, 10)), CAST(13.3817770000 AS Decimal(14, 10)))
;
INSERT [dbo].[Customers] VALUES (N'ANATR', N'Ana Trujillo Emparedados y helados', N'ana tijo', N'Owner', N'Avda. de la Constitución 2222', N'México D.F.', NULL, N'05021', N'Mexico', N'(5) 555-4729', CAST(19.4326080000 AS Decimal(14, 10)), CAST(-99.1332090000 AS Decimal(14, 10)))
;
INSERT [dbo].[Customers] VALUES (N'AROUT', N'Around the Hor', N'Thomas Hardy', N'Sales Representative', N'120 Hanover Sq.', N'London', NULL, N'WA1 1DP', N'UK', N'(171) 555-7788', CAST(51.5098650000 AS Decimal(14, 10)), CAST(-0.1180920000 AS Decimal(14, 10)))
;
INSERT [dbo].[Customers] VALUES (N'BERGS', N'Berglunds snabbköp', N'Christina Berglund', N'Order Administrator', N'Berguvsvägen  8', N'Luleå', NULL, N'S-958 22', N'Sweden', N'0921-12 34 65', CAST(65.5848160000 AS Decimal(14, 10)), CAST(22.1567040000 AS Decimal(14, 10)))
;
INSERT [dbo].[Customers] VALUES (N'BLONP', N'Blondesddsl père et fils', N'Frédérique Citeaux', N'Marketing Manager', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France', N'88.60.15.31', CAST(48.5800020000 AS Decimal(14, 10)), CAST(7.7500000000 AS Decimal(14, 10)))
;
INSERT [dbo].[Orders]  VALUES (10265, N'BLONP', CAST(N'1996-07-25' AS Date), CAST(55.2800 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France')
;
INSERT [dbo].[Orders]  VALUES (10278, N'BERGS', CAST(N'1996-08-12' AS Date), CAST(92.6900 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N'Luleå', NULL, N'S-958 22', N'Sweden')
;
INSERT [dbo].[Orders]  VALUES (10280, N'BERGS', CAST(N'1996-08-14' AS Date), CAST(8.9800 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N'Luleå', NULL, N'S-958 22', N'Sweden')
;
INSERT [dbo].[Orders]  VALUES (10297, N'BLONP', CAST(N'1996-09-04' AS Date), CAST(5.7400 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France')
;
INSERT [dbo].[Orders]  VALUES (10308, N'ANATR', CAST(N'1996-09-18' AS Date), CAST(1.6100 AS Decimal(18, 4)), N'Ana Trujillo Emparedados y helados', N'Avda. de la Constitución 2222', N'México D.F.', NULL, N'05021', N'Mexico')
;
INSERT [dbo].[Orders]  VALUES (10355, N'AROUT', CAST(N'1996-11-15' AS Date), CAST(41.9500 AS Decimal(18, 4)), N'Around the Hor', N'Brook Farm Stratford St. Mary', N'Colchester', N'Essex', N'CO7 6JX', N'UK')
```

```
;
INSERT [dbo].[Orders]  VALUES (10360, N'BLONP', CAST(N'1996-11-22' AS Date), CAST(131.7000 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France')
;
INSERT [dbo].[Orders]  VALUES (10383, N'AROUT', CAST(N'1996-12-16' AS Date), CAST(34.2400 AS Decimal(18, 4)), N'Around the Hor', N'Brook Farm Stratford St. Mary', N'Colchester', N'Essex', N'CO7 6JX', N'UK')
;
INSERT [dbo].[Orders]  VALUES (10384, N'BERGS', CAST(N'1996-12-16' AS Date), CAST(168.6400 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N'Luleå', NULL, N'S-958 22', N'Sweden')
;
INSERT [dbo].[Orders]  VALUES (10436, N'BLONP', CAST(N'1997-02-05' AS Date), CAST(156.6600 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France')
;
INSERT [dbo].[Orders]  VALUES (10444, N'BERGS', CAST(N'1997-02-12' AS Date), CAST(3.5000 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N'Luleå', NULL, N'S-958 22', N'Sweden')
;
INSERT [dbo].[Orders]  VALUES (10445, N'BERGS', CAST(N'1997-02-13' AS Date), CAST(9.3000 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N'Luleå', NULL, N'S-958 22', N'Sweden')
;
INSERT [dbo].[Orders]  VALUES (10449, N'BLONP', CAST(N'1997-02-18' AS Date), CAST(53.3000 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France')
;
INSERT [dbo].[Orders]  VALUES (10453, N'AROUT', CAST(N'1997-02-21' AS Date), CAST(25.3600 AS Decimal(18, 4)), N'Around the Hor', N'Brook Farm Stratford St. Mary', N'Colchester', N'Essex', N'CO7 6JX', N'UK')
;
INSERT [dbo].[Orders]  VALUES (10524, N'BERGS', CAST(N'1997-05-01' AS Date), CAST(244.7900 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N'Luleå', NULL, N'S-958 22', N'Sweden')
;
INSERT [dbo].[Orders]  VALUES (10558, N'AROUT', CAST(N'1997-06-04' AS Date), CAST(72.9700 AS Decimal(18, 4)), N'Around the Hor', N'Brook Farm Stratford St. Mary', N'Colchester', N'Essex', N'CO7 6JX', N'UK')
;
INSERT [dbo].[Orders]  VALUES (10559, N'BLONP', CAST(N'1997-06-05' AS Date), CAST(8.0500 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France')
;
INSERT [dbo].[Orders]  VALUES (10584, N'BLONP', CAST(N'1997-06-30' AS Date), CAST(59.1400 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France')
;
INSERT [dbo].[Orders]  VALUES (10625, N'ANATR', CAST(N'1997-08-08' AS Date), CAST(43.9000 AS Decimal(18, 4)), N'Ana Trujillo Emparedados y helados', N'Avda. de la Constitución 2222', N'México D.F.', NULL,
N'05021', N'Mexico')
;
INSERT [dbo].[Orders]  VALUES (10626, N'BERGS', CAST(N'1997-08-11' AS Date), CAST(138.6900 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N'Luleå', NULL, N'S-958 22', N'Sweden')
;
INSERT [dbo].[Orders]  VALUES (10628, N'BLONP', CAST(N'1997-08-12' AS Date), CAST(30.3600 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber', N'Strasbourg', NULL, N'67000', N'France')
;
```



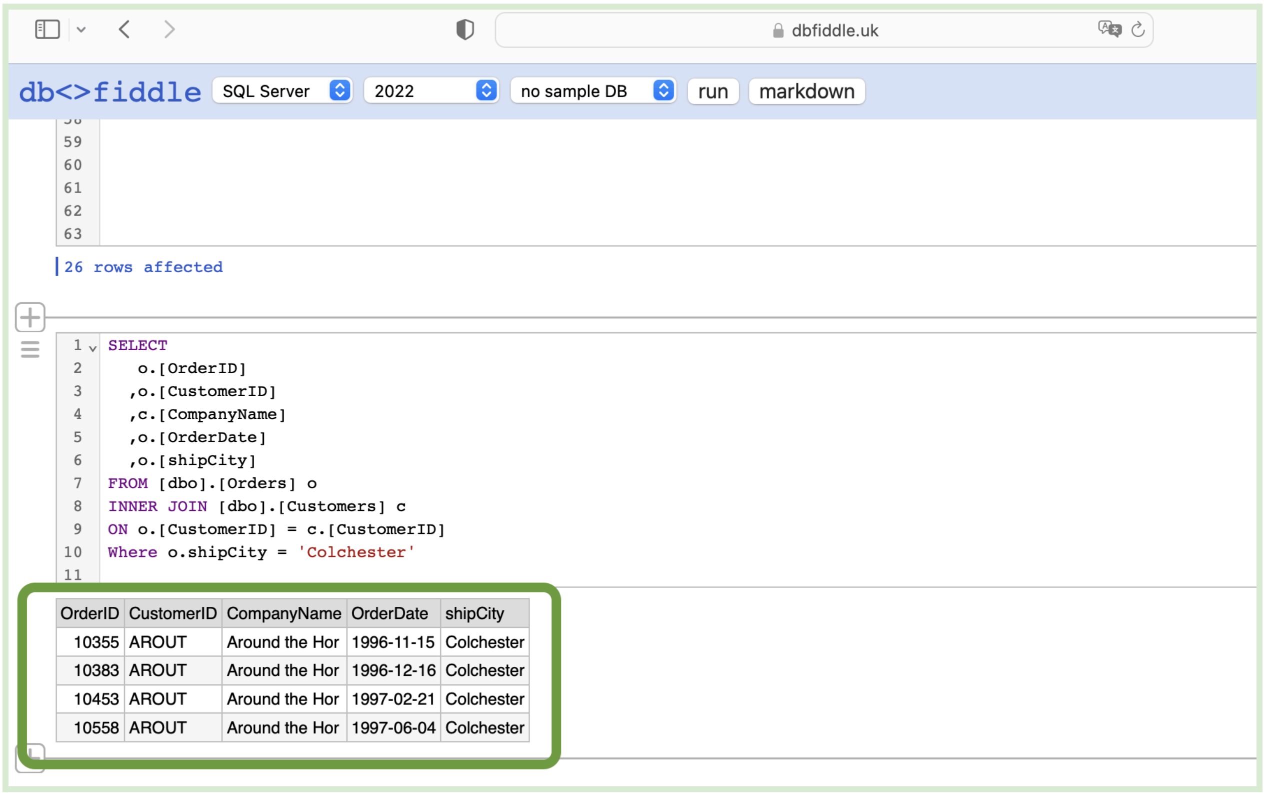
```
;
INSERT [dbo].[Orders]  VALUES (10436, N'BLONP', CAST(N'1997-02-05' AS Date), CAST(156.6600 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Klébe
;
INSERT [dbo].[Orders]  VALUES (10444, N'BERGS', CAST(N'1997-02-12' AS Date), CAST(3.5000 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N
;
INSERT [dbo].[Orders]  VALUES (10445, N'BERGS', CAST(N'1997-02-13' AS Date), CAST(9.3000 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8', N
;
INSERT [dbo].[Orders]  VALUES (10449, N'BLONP', CAST(N'1997-02-18' AS Date), CAST(53.3000 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber
;
INSERT [dbo].[Orders]  VALUES (10453, N'AROUT', CAST(N'1997-02-21' AS Date), CAST(25.3600 AS Decimal(18, 4)), N'Around the Hor', N'Brook Farm Stratford S
;
INSERT [dbo].[Orders]  VALUES (10524, N'BERGS', CAST(N'1997-05-01' AS Date), CAST(244.7900 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8',
;
INSERT [dbo].[Orders]  VALUES (10558, N'AROUT', CAST(N'1997-06-04' AS Date), CAST(72.9700 AS Decimal(18, 4)), N'Around the Hor', N'Brook Farm Stratford S
;
INSERT [dbo].[Orders]  VALUES (10559, N'BLONP', CAST(N'1997-06-05' AS Date), CAST(8.0500 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber'
;
INSERT [dbo].[Orders]  VALUES (10584, N'BLONP', CAST(N'1997-06-30' AS Date), CAST(59.1400 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber
;
INSERT [dbo].[Orders]  VALUES (10625, N'ANATR', CAST(N'1997-08-08' AS Date), CAST(43.9000 AS Decimal(18, 4)), N'Ana Trujillo Emparedados y helados', N'Av
;
INSERT [dbo].[Orders]  VALUES (10626, N'BERGS', CAST(N'1997-08-11' AS Date), CAST(138.6900 AS Decimal(18, 4)), N'Berglunds snabbköp', N'Berguvsvägen  8',
;
INSERT [dbo].[Orders]  VALUES (10628, N'BLONP', CAST(N'1997-08-12' AS Date), CAST(30.3600 AS Decimal(18, 4)), N'Blondel père et fils', N'24, place Kléber
;
```

```
26 rows affected
```

Step 4. Create a new Batch and Start with Safe Examples: A, B, C, D, E

Example A: Static SQL (Baseline)

```
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
Where o.shipCity = 'Colchester'
```



```
26 rows affected
```

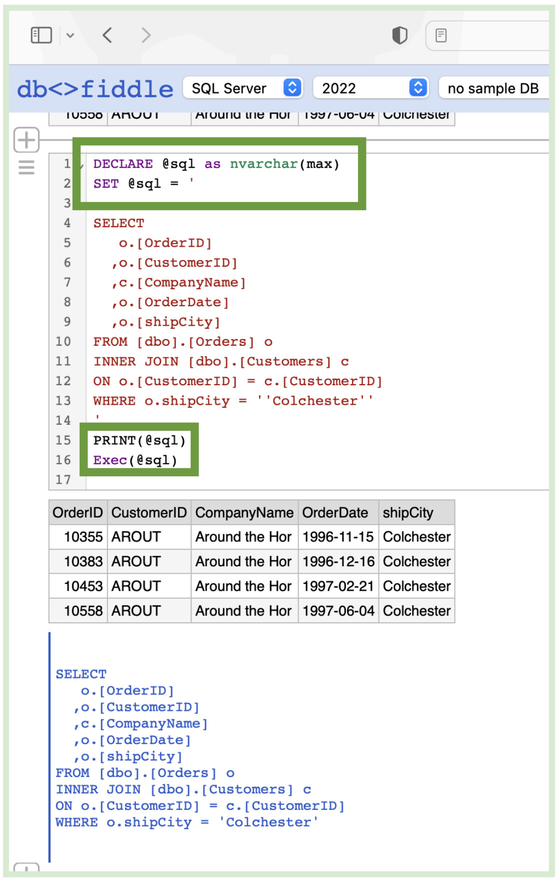
```
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
Where o.shipCity = 'Colchester'
```

OrderID	CustomerID	CompanyName	OrderDate	shipCity
10355	AROUT	Around the Hor	1996-11-15	Colchester
10383	AROUT	Around the Hor	1996-12-16	Colchester
10453	AROUT	Around the Hor	1997-02-21	Colchester
10558	AROUT	Around the Hor	1997-06-04	Colchester

Security Level: ✅ **SAFE** - This is a static query with hardcoded values. No user input is involved.

Example B: Basic Dynamic SQL

```
DECLARE @sql as nvarchar(max)
SET @sql = '

SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = ”Colchester”
'
PRINT(@sql)
Exec(@sql)
```


db<>fiddle SQL Server 2022 no sample DB

```
DECLARE @sql as nvarchar(max)
SET @sql = '

SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = ''Colchester''
'
PRINT(@sql)
Exec(@sql)
```

OrderID	CustomerID	CompanyName	OrderDate	shipCity
10355	AROUT	Around the Hor	1996-11-15	Colchester
10383	AROUT	Around the Hor	1996-12-16	Colchester
10453	AROUT	Around the Hor	1997-02-21	Colchester
10558	AROUT	Around the Hor	1997-06-04	Colchester

```
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = 'Colchester'
```

- ➔ Dynamic SQL Variable: @sql stores the entire SQL query as a string
- ➔ String Literal: The query is hardcoded with ’Colchester’ (notice the double quotes ”Colchester” to escape the single quotes)
- ➔ PRINT Statement: Shows the actual SQL that will be executed
- ➔ EXEC Command: Executes the dynamically built SQL string

Security Level: ⚠️ **NEUTRAL** - Dynamic SQL with hardcoded values. Still safe but introduces the dynamic execution pattern.

Why do we need to use Dynamic SQL?

Instead of writing:

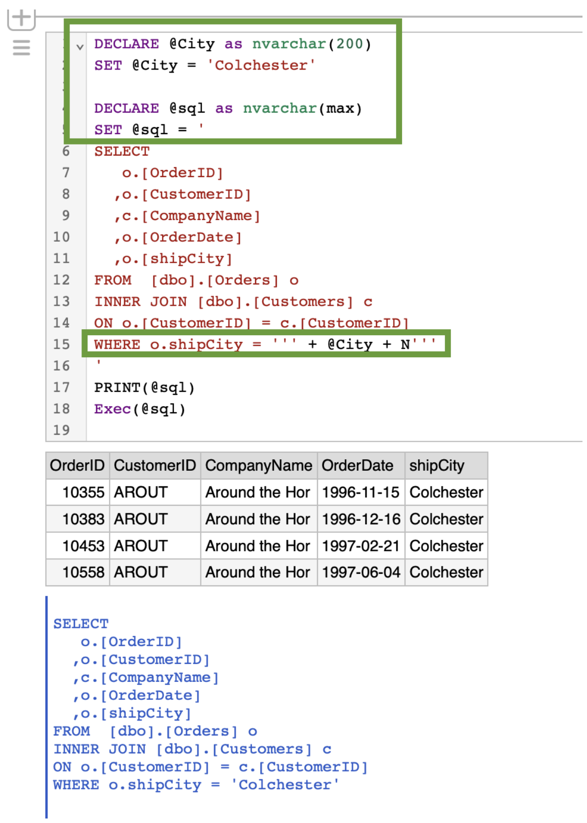
```
-- 100+ different static queries
SELECT ... WHERE city = 'London'
SELECT ... WHERE city = 'Paris'
SELECT ... WHERE city = 'Berlin'
SELECT ... WHERE city = 'Tokyo'
```

You write:

```
-- ONE flexible query

SELECT ... WHERE city = @CityFrom
```

Example C: String Concatenation

```
DECLARE @City as nvarchar(200)
SET @City = 'Colchester'

DECLARE @sql as nvarchar(max)
SET @sql = '
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = ''' + @City + N'''
'
PRINT(@sql)
Exec(@sql)
```



```
DECLARE @City as nvarchar(200)
SET @City = 'Colchester'

DECLARE @sql as nvarchar(max)
SET @sql = '
SELECT
    o.[OrderID]
   ,o.[CustomerID]
   ,c.[CompanyName]
   ,o.[OrderDate]
   ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = ''' + @City + N'''
'

PRINT(@sql)
Exec(@sql)
```

OrderID	CustomerID	CompanyName	OrderDate	shipCity
10355	AROUT	Around the Hor	1996-11-15	Colchester
10383	AROUT	Around the Hor	1996-12-16	Colchester
10453	AROUT	Around the Hor	1997-02-21	Colchester
10558	AROUT	Around the Hor	1997-06-04	Colchester

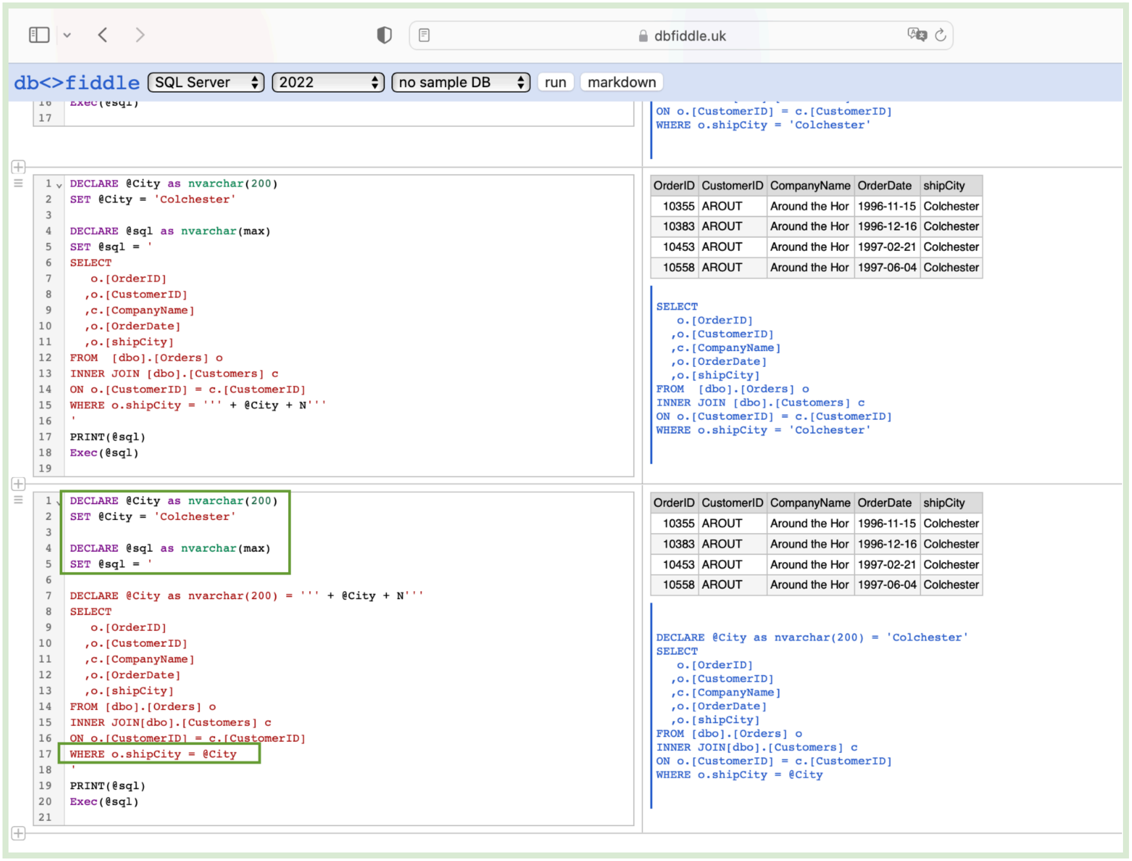
```
SELECT
    o.[OrderID]
   ,o.[CustomerID]
   ,c.[CompanyName]
   ,o.[OrderDate]
   ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = 'Colchester'
```

➔ WHERE o.shipCity = ''' + @City + N'''. This is where the value of the @City variable ('Colchester') is embedded directly into the WHERE clause string. The triple single quotes (''') are used to correctly enclose the string value 'Colchester' within the larger @sql string.

Security Level: 🚨 **VULNERABLE** - Direct string concatenation opens the door to SQL injection attacks.

Example D: Variable Declaration in Dynamic SQL

```
DECLARE @City as nvarchar(200)
SET @City = 'Colchester'

DECLARE @sql as nvarchar(max)
SET @sql = '

DECLARE @City as nvarchar(200) = ''' + @City + N'''
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM [dbo].[Orders] o
INNER JOIN[dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = @City
'
PRINT(@sql)
Exec(@sql)
```



```sql
Exec(@sql)

```

```
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = 'Colchester'
```

```sql
DECLARE @City as nvarchar(200)
SET @City = 'Colchester'

DECLARE @sql as nvarchar(max)
SET @sql = '
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = ''' + @City + N'''
'
PRINT(@sql)
Exec(@sql)

```

OrderID	CustomerID	CompanyName	OrderDate	shipCity
10355	AROUT	Around the Hor	1996-11-15	Colchester
10383	AROUT	Around the Hor	1996-12-16	Colchester
10453	AROUT	Around the Hor	1997-02-21	Colchester
10558	AROUT	Around the Hor	1997-06-04	Colchester

```
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = 'Colchester'
```

```sql
DECLARE @City as nvarchar(200)
SET @City = 'Colchester'

DECLARE @sql as nvarchar(max)
SET @sql = '

DECLARE @City as nvarchar(200) = ''' + @City + N'''
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM [dbo].[Orders] o
INNER JOIN[dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = @City
'
PRINT(@sql)
Exec(@sql)

```

OrderID	CustomerID	CompanyName	OrderDate	shipCity
10355	AROUT	Around the Hor	1996-11-15	Colchester
10383	AROUT	Around the Hor	1996-12-16	Colchester
10453	AROUT	Around the Hor	1997-02-21	Colchester
10558	AROUT	Around the Hor	1997-06-04	Colchester

```
DECLARE @City as nvarchar(200) = 'Colchester'
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM [dbo].[Orders] o
INNER JOIN[dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = @City
```

{DECLARE @City as nvarchar(200)

SET @City = 'Colchester'}

➔ This declares a variable named @City in the *outer scope* of the script and initializes it with the value 'Colchester'. This is the city name that will be passed into the dynamic query.

{ DECLARE @sql as nvarchar(max)

SET @sql = '}

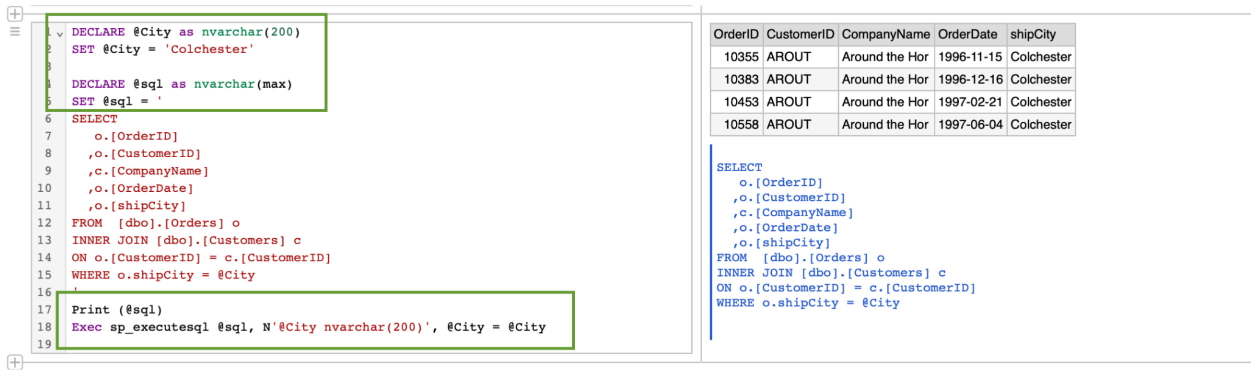
➔ This declares a nvarchar(max) variable named @sql to hold the dynamic SQL string. The SET statement constructs the actual SQL query as a string. DECLARE @City as nvarchar(200) = ''' + @City + N''': This is the key part of this specific example. It re-declares a new @City variable inside the dynamic SQL string and assigns it the value from the outer @City variable ('Colchester'). The triple single quotes (''') are used for escaping, ensuring 'Colchester' is correctly embedded as a string literal.

➔ The SELECT statement then uses this inner @City variable in its WHERE clause: WHERE o.shipCity = @City.

Security Level: 🚨 **VULNERABLE** - Even though a variable is used inside the dynamic SQL, the initial concatenation still creates vulnerability.

Example E: Parameterized Query (SECURE)

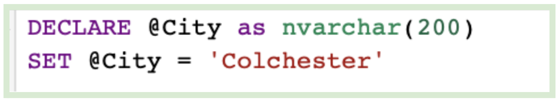
```
DECLARE @City as nvarchar(200)
SET @City = 'Colchester'

DECLARE @sql as nvarchar(max)
SET @sql = '
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = @City
'
Print (@sql)
Exec sp_executesql @sql, N'@City nvarchar(200)', @City = @City
```



```
DECLARE @City as nvarchar(200)
SET @City = 'Colchester'

DECLARE @sql as nvarchar(max)
SET @sql = '
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = @City
'
Print (@sql)
Exec sp_executesql @sql, N'@City nvarchar(200)', @City = @City
```

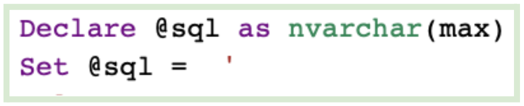
OrderID	CustomerID	CompanyName	OrderDate	shipCity
10355	AROUT	Around the Hor	1996-11-15	Colchester
10383	AROUT	Around the Hor	1996-12-16	Colchester
10453	AROUT	Around the Hor	1997-02-21	Colchester
10558	AROUT	Around the Hor	1997-06-04	Colchester

```
SELECT
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = @City
```



```
DECLARE @City as nvarchar(200)
SET @City = 'Colchester'
```

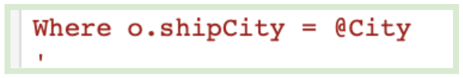
➔ This declares an nvarchar variable named @City and assigns it the value 'Colchester'. This variable holds the value that will be used to filter the orders.



```
Declare @sql as nvarchar(max)
Set @sql =  '
```

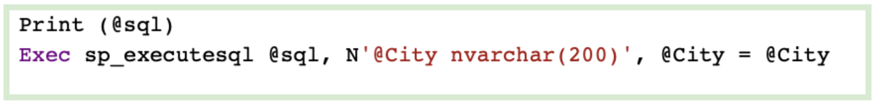
➔ This declares an nvarchar(max) variable named @sql to store the dynamic SQL query string.

➔ The SELECT statement within @sql is designed to retrieve order and customer details, joining dbo.Orders and dbo.Customers.



```
Where o.shipCity = @City
'
```

➔ Crucially, in the WHERE clause, it uses @City as a placeholder: WHERE o.shipCity = @City. The actual value ('Colchester') is *not* directly concatenated into this string.



```
Print (@sql)
Exec sp_executesql @sql, N'@City nvarchar(200)', @City = @City
```

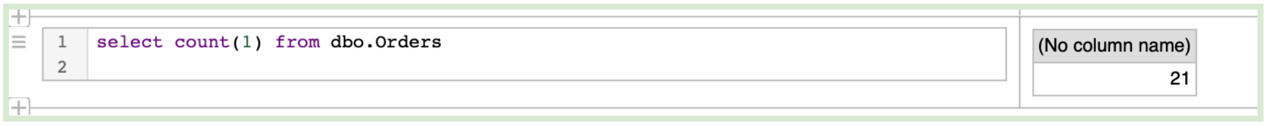
- ➔ @sql: The first argument is the dynamic SQL string to be executed.
- ➔ N'@City nvarchar(200)': The second argument is a string that defines the parameters used in the dynamic SQL. It tells sp_executesql that there's a parameter named @City and its data type is nvarchar(200).
- ➔ @City = @City: The third argument (and subsequent arguments for more parameters) provides the actual values for the defined parameters. Here, it maps the *outer* @City variable's value to the *inner* @City parameter in the dynamic SQL.

Security Level: ✅ **SECURE** - Uses sp_executesql with proper parameter binding.

Step 5. Practice Running SQL Injections For Educational Purposes Only: E, F, G

Example H: Before Running SQL Injections Check how many observations you have before the injection.

```
select count(1)  from dbo.Orders;
```



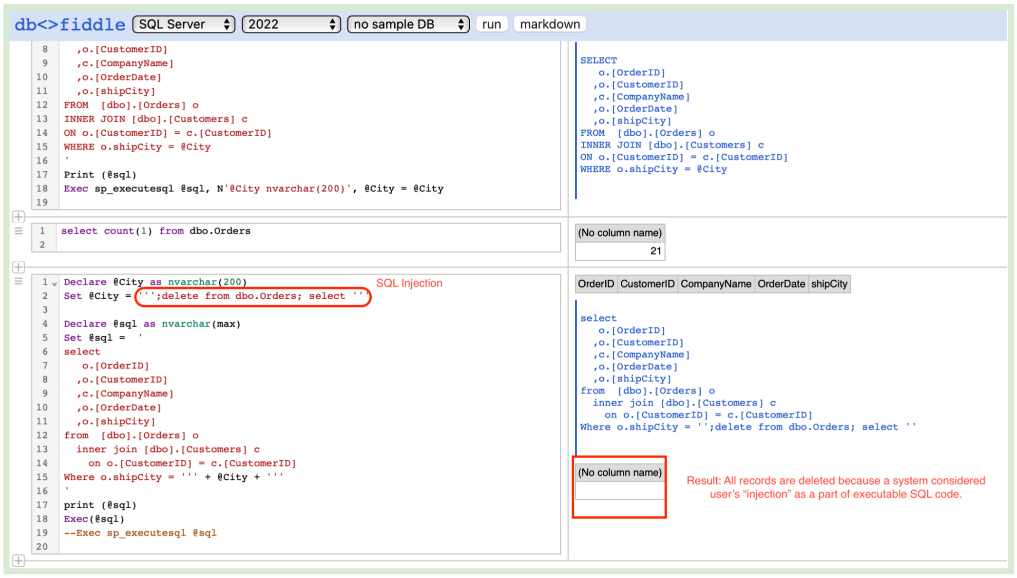
```
select count(1) from dbo.Orders
```

```
(No column name)
              21
```

* You have 21 observations(records) before the SQL Injection

Example G: Malicious DELETE Attack

```
Declare @City as nvarchar(200)
Set @City = '';delete from dbo.Orders; select ''

Declare @sql as nvarchar(max)
Set @sql =  '
select
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
from  [dbo].[Orders] o
  inner join [dbo].[Customers] c
    on o.[CustomerID] = c.[CustomerID]
Where o.shipCity = ''' + @City + '''
'
print (@sql)
Exec(@sql)
--Exec sp_executesql @sql
```



```
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = @City
'
Print (@sql)
Exec sp_executesql @sql, N'@City nvarchar(200)', @City = @City
```

```
SELECT
    o.[OrderID]
   ,o.[CustomerID]
   ,c.[CompanyName]
   ,o.[OrderDate]
   ,o.[shipCity]
FROM  [dbo].[Orders] o
INNER JOIN [dbo].[Customers] c
ON o.[CustomerID] = c.[CustomerID]
WHERE o.shipCity = @City
```

```
select count(1) from dbo.Orders
```

(No column name)
21

```
Declare @City as nvarchar(200)
Set @City = ''';delete from dbo.Orders; select '''

Declare @sql as nvarchar(max)
Set @sql =  '
select
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
from  [dbo].[Orders] o
  inner join [dbo].[Customers] c
    on o.[CustomerID] = c.[CustomerID]
Where o.shipCity = ''' + @City + '''
'
print (@sql)
Exec(@sql)
--Exec sp_executesql @sql
```

SQL Injection

OrderID	CustomerID	CompanyName	OrderDate	shipCity

```
select
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
from  [dbo].[Orders] o
  inner join [dbo].[Customers] c
    on o.[CustomerID] = c.[CustomerID]
Where o.shipCity = '';delete from dbo.Orders; select ''
```

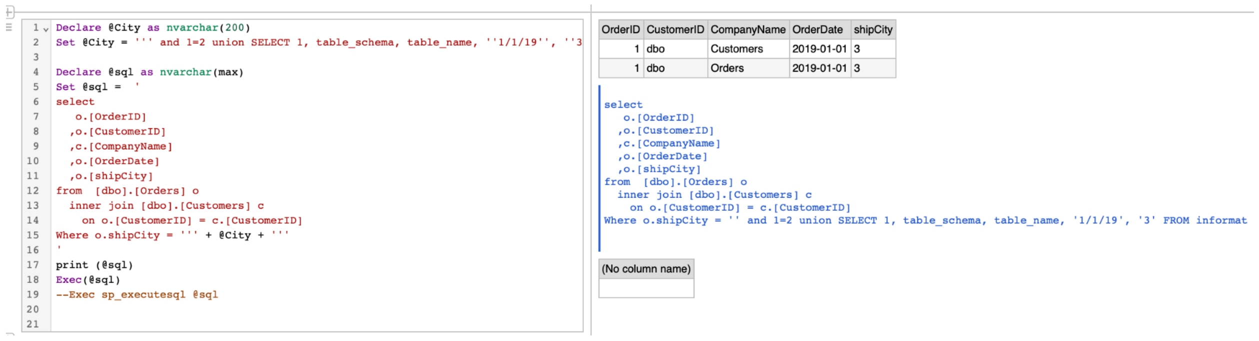
(No column name)

Result: All records are deleted because a system considered user's "injection" as a part of executable SQL code.

Result: 🚨 **CATASTROPHIC** - This would delete all records from the Orders table!

Example H: Data Extraction Attack (Before running this code, Delete SQL Code from Example G and Run the Code from the Beginning)

```
Declare @City as nvarchar(200)
Set @City = ''' and 1=2 union SELECT 1, table_schema, table_name, ''1/1/19'', ''3'' FROM information_schema.tables; select '''

Declare @sql as nvarchar(max)
Set @sql =  '
select
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
from  [dbo].[Orders] o
  inner join [dbo].[Customers] c
    on o.[CustomerID] = c.[CustomerID]
Where o.shipCity = ''' + @City + '''
'
print (@sql)
Exec(@sql)
--Exec sp_executesql @sql
```



```sql
Declare @City as nvarchar(200)
Set @City = ''' and 1=2 union SELECT 1, table_schema, table_name, ''1/1/19'', ''3

Declare @sql as nvarchar(max)
Set @sql =  '
select
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
from  [dbo].[Orders] o
  inner join [dbo].[Customers] c
    on o.[CustomerID] = c.[CustomerID]
Where o.shipCity = ''' + @City + '''
'
print (@sql)
Exec(@sql)
--Exec sp_executesql @sql
```

OrderID	CustomerID	CompanyName	OrderDate	shipCity
1	dbo	Customers	2019-01-01	3
1	dbo	Orders	2019-01-01	3

```
select
   o.[OrderID]
  ,o.[CustomerID]
  ,c.[CompanyName]
  ,o.[OrderDate]
  ,o.[shipCity]
from  [dbo].[Orders] o
  inner join [dbo].[Customers] c
    on o.[CustomerID] = c.[CustomerID]
Where o.shipCity = '' and 1=2 union SELECT 1, table_schema, table_name, '1/1/19', '3' FROM informat
```

(No column name)

Result: 🚨 **DATA BREACH** - This attack could expose database schema information to attackers.

Conclusion

Dynamic SQL is a powerful tool that can enhance application flexibility, but it requires careful implementation to avoid security vulnerabilities. The key is understanding that string concatenation is the enemy of security, while parameterized queries are your best defense against SQL injection attacks.

Remember: A single poorly implemented dynamic SQL query can compromise your entire database. Always prioritize security over convenience.

Key Security Insights:

1. Never Trust User Input

All user input should be treated as potentially malicious. Never concatenate user input directly into SQL strings.

2. Use Parameterized Queries

Always use sp_executesql with proper parameter binding:

```
Exec sp_executesql @sql, N'@param datatype', @param = @value
```

3. Validate Input

- Check data types
- Validate length constraints
- Use whitelist validation where possible
- Sanitize special characters

4. Principle of Least Privilege

- Use database accounts with minimal necessary permissions
- Avoid using sa or administrative accounts for application connections
- Grant only SELECT, INSERT, UPDATE, DELETE permissions as needed