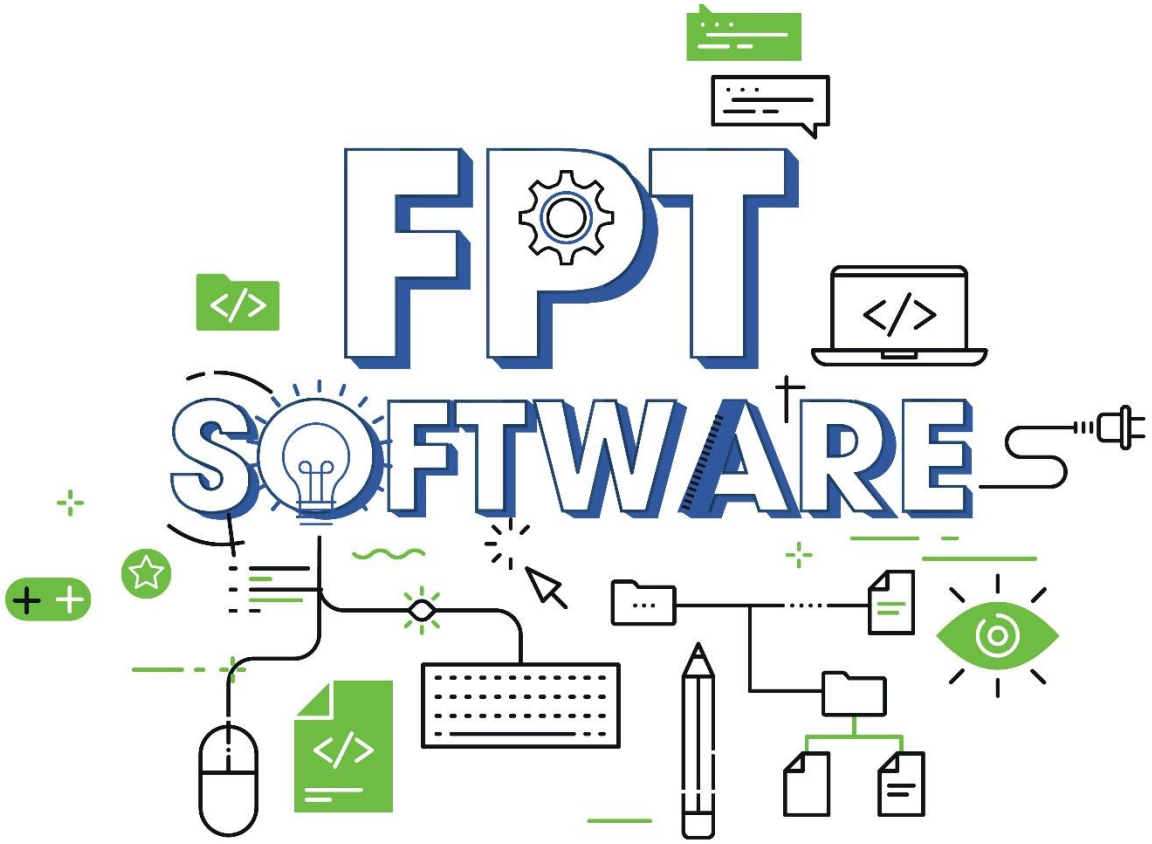
# Transactions and SQL Injection

*Fsoft Academy*



# Lesson Objectives

- Able to create Transactions in SQL Server
- Understand what is SQL Injection and avoid the injection error in queries
- Able to use SQL Server Profiler

# Agenda

1. SQL Transactions
2. SQL Injection
3. SQL Server Profiler

Section1

# SQL TRANSACTIONS

# WHAT'S SQL TRANSACTIONS?

- SQL transaction is a **single unit of work** applied to a database.
- It is a **sequence of ordered operations performed** on the database.
- SQL statements are used to execute tasks such as update data or get data from a database.

TRANSACTION là một giao dịch gồm nhiều câu lệnh SQL được thực hiện như một đơn vị duy nhất.

# PROPERTIES OF TRANSACTIONS

ACID

- **Atomicity:**
  - ✓ Ensures that all operations within the work unit are completed successfully.
  - ✓ Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.
- **Consistency**:
  - ✓ Ensures that the database properly changes states upon a successfully committed transaction. dữ liệu trong database luôn hợp lệ trước và sau khi transaction được thực thi
- **Isolation**:
  - ✓ Enables transactions to operate independently of and transparent to each other.
- **Durability**: đảm bảo rằng kết quả của một transaction đã COMMIT sẽ được lưu vĩnh viễn, ngay cả khi hệ thống gặp sự cố (mất điện, crash, v.v.).
  - ✓ Ensures that the result or effect of a committed transaction persists in case of a system failure. SQL Server lưu mọi thay đổi vào Transaction Log trước khi ghi vào database.Nếu hệ thống sập, log sẽ giúp khôi phục trạng thái chính xác.

# SQL Transactions (Transact-SQL)

- **SQL Server operates in the following transaction modes:**

4 loại transactions

| Modes | Detail |
|---|---|
| Autocommit transactions | Each individual statement is a transaction. Mỗi câu lệnh DML (INSERT, UPDATE, DELETE, ...) là một transaction riêng lẻ. Nếu lỗi , tự động ROLLBACK. Không thể ROLLBACK nhiều câu lệnh một lúc. |
| Explicit transactions | Each transaction is explicitly started with the BEGIN TRANSACTION statement and explicitly ended with a COMMIT or ROLLBACK statement. Nếu không COMMIT hoặc ROLLBACK, transaction có thể bị khóa. |
| Implicit transactions | A new transaction is implicitly started when the previous transaction completes, but each transaction is explicitly completed with a COMMIT or ROLLBACK statement. |
| Batch-scoped transactions | This mode is applicable only to SQL Server Multiple Active Result Sets (MARS) feature, designed to permits multiple batches to be executed using a single SQL connection. |

# BEGIN TRANSACTION (Transact-SQL)

- Marks the starting point of an explicit, local transaction.
- Explicit transactions start with the BEGIN TRANSACTION statement and end with the COMMIT or ROLLBACK statement.
- **Syntax**

```
--Applies to SQL Server and Azure SQL Database

BEGIN { TRAN | TRANSACTION }
    [ { transaction_name | @tran_name_variable }
      [ WITH MARK [ 'description' ] ]
    ]
[ ; ]
```

# COMMIT COMMAND

- The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.
- The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.
- The syntax for the COMMIT command:

```
-- Applies to SQL Server (starting with 2008) and Azure SQL Database

COMMIT [ { TRAN | TRANSACTION }  [ transaction_name | @tran_name_variable ] ] [
WITH ( DELAYED_DURABILITY = { OFF | ON } ) ]
[ ; ]
```

# ROLLBACK COMMAND

- The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.
- **The syntax for a ROLLBACK command:**

```
--Applies to SQL Server and Azure SQL Database

ROLLBACK { TRAN | TRANSACTION }
     [ transaction_name | @tran_name_variable
     | savepoint_name | @savepoint_variable ]
[ ; ]
```

# SQL Transactions (Transact-SQL)

- **Example:**

```
GO

INSERT INTO dbo.Account
VALUES     ('A678SA', '2018-01-01', '2018-01-01', '2028-01-01', 1000),
           ('A678SB', '2018-01-01', '2018-01-01', '2028-01-01', 800)
```

```
DECLARE @status NVARCHAR(100), @result INT
BEGIN TRY
          BEGIN TRAN TRAN1
            UPDATE dbo.Account SET balance = balance + 900
            WHERE account_number LIKE 'A678SB';

            UPDATE dbo.Account SET balance = balance - 900
            WHERE  account_number LIKE 'A678SB';

          COMMIT TRAN TRAN1
END TRY
BEGIN CATCH
          ROLLBACK TRAN TRAN1
END CATCH
```

**Account** table:

| Column Name | Data Type |
|---|---|
| account_id | int |
| account_number | varchar(50) |
| open_date | datetime |
| date_valid_from | datetime |
| date_valid_in | datetime |
| balance | decimal(12, 2) |

**Example** *with column constraint:*

```
…
balance decimal(12,2) CHECK(balance > 0),
…
```

**Result:** *This transaction will be rolled back*

# SAVEPOINT COMMAND

giúp rollback 1 phần thay vì toàn bộ

- A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
- The syntax for a SAVEPOINT command is as shown below:

```
SAVEPOINT SAVEPOINT_NAME;
```

trong sql server ko dùng mà trong mysql , postgres, oracle

- This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.
- The syntax for rolling back to a SAVEPOINT is as shown below:

```
ROLLBACK TO SAVEPOINT_NAME;
```

# SAVEPOINT COMMAND

- **Example 1**:

```
BEGIN TRANSACTION

   INSERT INTO TestTable (id, val_test)
   VALUES  ( 1, 10)
   -- this will create a savepoint after the first INSERT
   SAVE TRANSACTION FirstInsert

   INSERT INTO TestTable (id, val_test)
   VALUES (2, 20)

   -- this will rollback to the savepoint right after the first INSERT was done
   ROLLBACK TRANSACTION FirstInsert

-- this will commit the transaction leaving just the first INSERT
COMMIT

SELECT * FROM TestTable
```

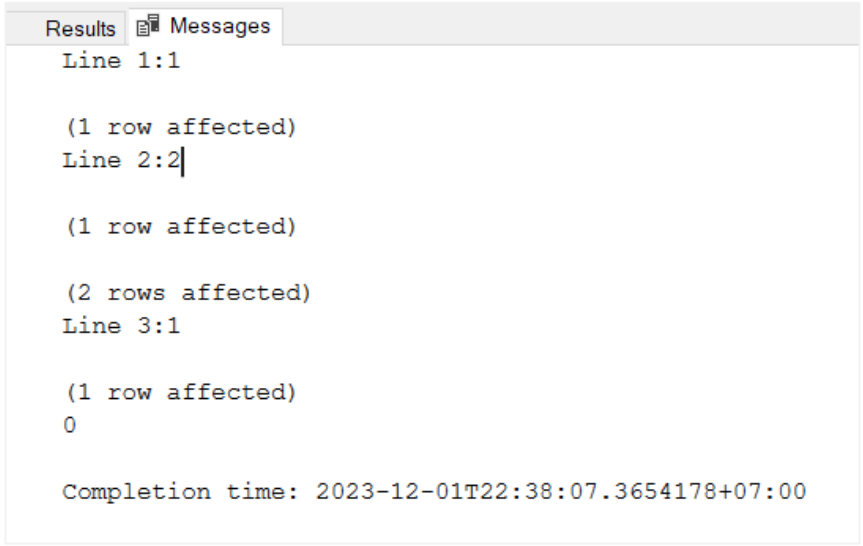
# SAVEPOINT COMMAND

- **Example 2**:
  - ✓Using SQL Server transaction savepoints with the same savepoint name.
  - ✓Duplicate savepoint names are used and the transaction rolled back to the **second savepoint**.

```
BEGIN TRANSACTION


   INSERT INTO TestTable (id, val_test)
   VALUES  ( 1, 10)

   -- this will create a savepoint after the first INSERT
   SAVE TRANSACTION DataInsert
   INSERT INTO TestTable (id, val_test)
   VALUES (2, 20)

   -- this will create a savepoint with same name
   SAVE TRANSACTION DataInsert
   INSERT INTO TestTable (id, val_test)
   VALUES (3, 30)

   -- this will rollback to the savepoint right after the first INSERT was done
   ROLLBACK TRANSACTION DataInsert

-- this will commit the transaction leaving just the first INSERT
COMMIT
```

# SQL Transactions (Transact-SQL)

- **Rolling back a nested transaction**
  - ✓ This demonstrates committing an "**inner**" transaction then rolling back the "**outer**".
  - ✓ Even though the **"inner" transaction is committed** when the **"outer" one is rolled back** the whole thing gets rolled back and **no rows are in the table**.
- **Output:**




- **Example 1:**

```
-- Create a table to use during the tests
CREATE TABLE TransactionTest (val_test int)
GO

-- Test using 2 transactions and a rollback on the
-- outer transaction
BEGIN TRANSACTION -- outer transaction
    PRINT 'Line 1:' + CAST(@@TRANCOUNT AS VARCHAR)
    INSERT INTO TransactionTest VALUES (1)

BEGIN TRANSACTION -- inner transaction
    PRINT 'Line 2:' + CAST(@@TRANCOUNT AS VARCHAR)
    INSERT INTO TransactionTest VALUES (2)
    SELECT * FROM TransactionTest

    COMMIT -- commit the inner transaction

PRINT 'Line 3:' + CAST(@@TRANCOUNT AS VARCHAR)
    INSERT INTO TransactionTest VALUES (3)
ROLLBACK -- roll back the outer transaction

PRINT @@TRANCOUNT
```

# SQL Transactions (Transact-SQL)

- **Example 2**:
  - ✓ After rolling back the inner transaction we actually get an error when trying to commit the outer transaction.
  - ✓ Once the ROLLBACK command is executed the whole transaction is rolled back all the way to the beginning.
  - ✓ The last **insert statement (value 3)** is in *an implicit transaction* and *will commit* even though the COMMIT statement returns an error. So we end up with one row with a 3 in it.

```
-- Test using 2 transactions and a rollback on the
-- inner transaction
BEGIN TRANSACTION -- outer transaction
    PRINT 'LINE1: ' + CAST(@@TRANCOUNT AS VARCHAR);
    INSERT INTO TransactionTest VALUES (1)
    BEGIN TRANSACTION -- inner transaction
        PRINT 'LINE2: ' + CAST(@@TRANCOUNT AS VARCHAR);
        INSERT INTO TransactionTest VALUES (2)
    ROLLBACK -- roll back the inner transaction
    PRINT 'LINE3: ' + CAST(@@TRANCOUNT AS VARCHAR);
    INSERT INTO TransactionTest VALUES (3)
-- We get an error here because there is no transaction
-- to commit.
COMMIT -- commit the outer transaction
PRINT 'LINE4: ' + CAST(@@TRANCOUNT AS VARCHAR);
SELECT * FROM TransactionTest
```

**Output**:

Results

| | val_test |
|---|---|
| 1 | 3 |

Results | Messages

```
LINE1: 1

(1 row affected)
LINE2: 2

(1 row affected)
LINE3: 0

(1 row affected)
Msg 3902, Level 16, State 1, Line 38
The COMMIT TRANSACTION request has no corresponding BEGIN TRANSACTION.
LINE4: 0

(3 rows affected)

Completion time: 2023-12-01T22:48:26.0096288+07:00
```

# SQL Transactions (Transact-SQL)

- **Solution for Example 2**:

✓ One way to avoid the error is to take advantage of the @@TRANCOUNT system variable.

```
TRUNCATE TABLE TransactionTest
GO

BEGIN TRANSACTION -- outer transaction
    PRINT 'LINE1: ' + CAST(@@TRANCOUNT AS VARCHAR);
    INSERT INTO TransactionTest VALUES (1)
    BEGIN TRANSACTION -- inner transaction
        PRINT 'LINE2: ' + CAST(@@TRANCOUNT AS VARCHAR);
        INSERT INTO TransactionTest VALUES (2)
    ROLLBACK --roll back the inner transaction
    PRINT 'LINE3: ' + CAST(@@TRANCOUNT AS VARCHAR);
    INSERT INTO TransactionTest VALUES (3)
    IF @@TRANCOUNT > 0
        -- No error this time
        COMMIT -- commit the outer transaction
PRINT 'LINE4: ' + CAST(@@TRANCOUNT AS VARCHAR);

SELECT * FROM TransactionTest
```

# RELEASE SAVEPOINT COMMAND

- The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.
- The syntax for a SAVEPOINT command is as shown below:

```
RELEASE SAVEPOINT SAVEPOINT_NAME;
```

```
-- Không có lệnh "RELEASE SAVEPOINT" trong SQL Server
-- Chỉ có thể rollback hoặc commit toàn bộ giao dịch
```

# SET TRANSACTION COMMAND

- The SET TRANSACTION command can be used to initiate a database transaction. This command is used to specify characteristics for the transaction that follows. For example, you can specify a transaction to be read only or read write.
- The syntax for a SET TRANSACTION command is as follows:

```
SET TRANSACTION [ READ WRITE | READ ONLY ];
```

SQL Server chỉ hỗ trợ SET TRANSACTION ISOLATION LEVEL., ko có read wirte, read only và đặt tên cho transaction như Oracle

SET TRANSACTION ISOLATION LEVEL READ COMMITTED; : Chỉ đọc dữ liệu đã được commit từ các transaction đang mở.

Section 2

# SQL Injection

# WHAT'S SQL INJECTION?

- **SQL injection** is a code injection technique that might destroy your database.
- SQL injection is one of the most common web hacking techniques.
- SQL injection is the placement of malicious code in SQL statements, via web page input.

# SQL INJECTION EXAMPLE

## SQL IN WEB PAGE

- SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.
- **Example**:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

# SQL INJECTION EXAMPLE

## SQL INJECTION BASED ON 1=1 IS ALWAYS TRUE

- Conditional sentences are always true. A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

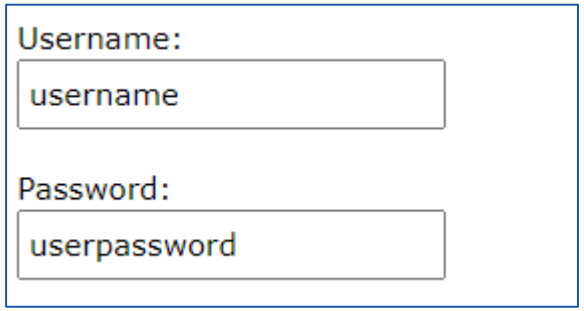
- Example:

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

# SQL INJECTION EXAMPLE

## SQL INJECTION BASED ON ""="" IS ALWAYS TRUE

- Example of a user login on a web site:




```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'
```

# SQL INJECTION EXAMPLE

## SQL INJECTION BASED ON BATCHED SQL STATEMENTS

- Most databases support batched SQL statement.
- A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.
- Example:

```
SELECT * FROM Users; DROP TABLE Suppliers
```

# SQL INJECTION EXAMPLE

## USE SQL PARAMETERS FOR PROTECTION

- To protect a web site from SQL injection, you can use SQL parameters.
- SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.
- Example:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = @0";
db.Execute(txtSQL,txtUserId);
```

Section 3

# SQL Server Profiler

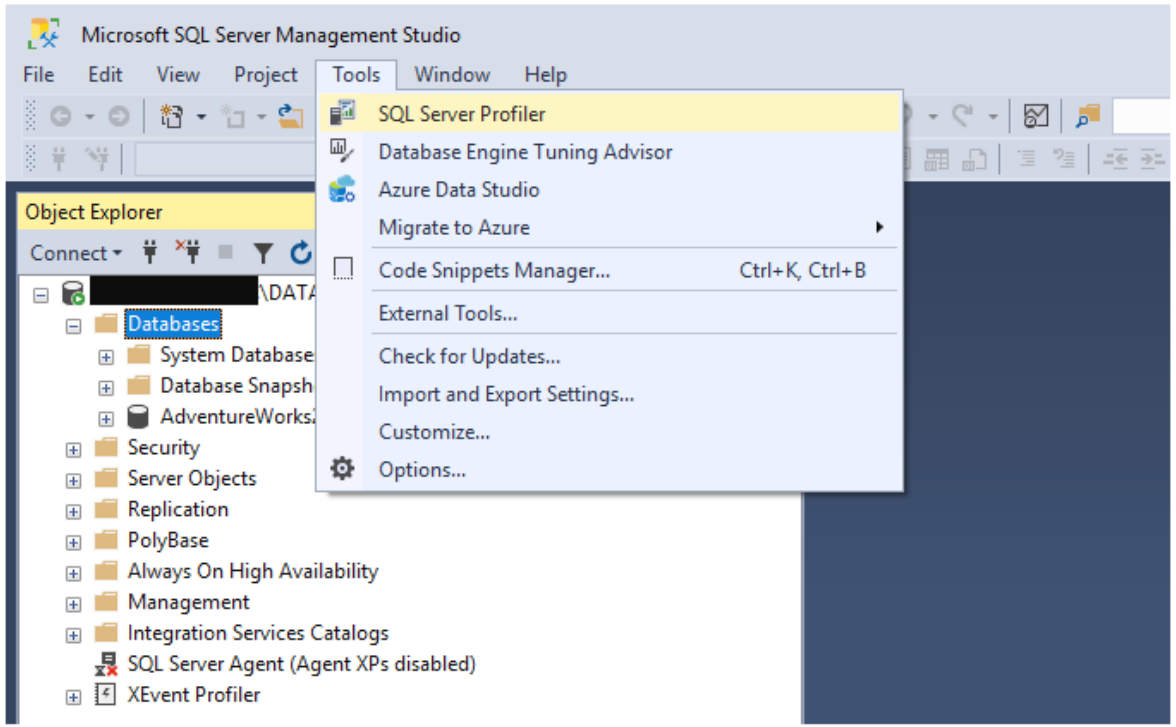
# What's SQL Server Profiler?

- SQL Server Profiler is an interface to create and manage traces and analyze and replay trace results. Events are saved in a trace file that can later be analyzed or used to replay a specific series of steps when diagnosing a problem.
- SQL Server is responsible for two main operations:
  - **Tracing**: It can monitor all operations executed over an instance
  - **Replay**: It can rerun all operations logged in a trace later

SQL Server Profiler là một công cụ trong SQL Server dùng để theo dõi (tracing) và phát lại (replay) các sự kiện xảy ra trên một instance.

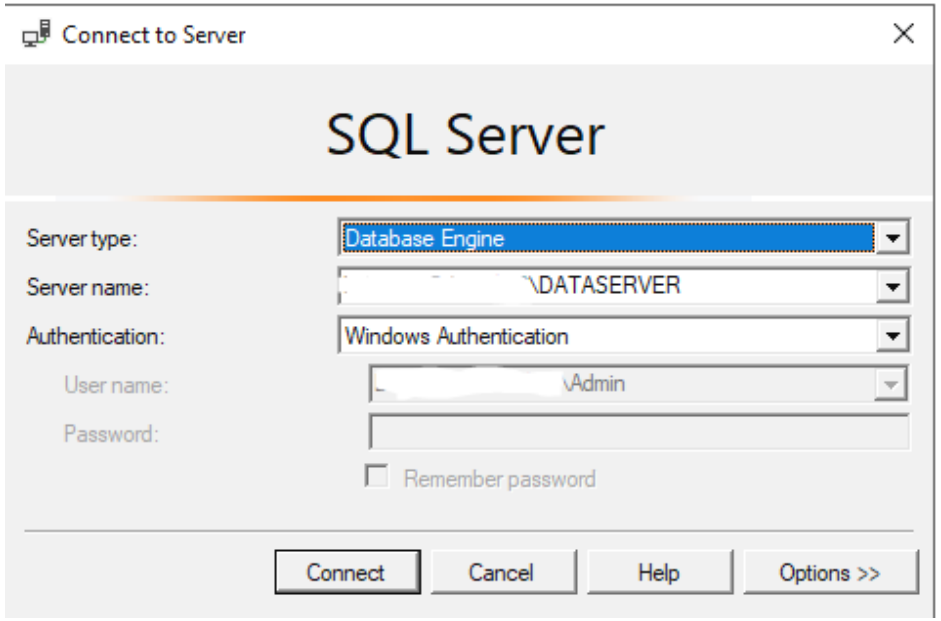
# How to use Profiler?

- You can simply find a shortcut of this tool under the **Tools** menu inside the SQL Server Management Studio as shown in the image below:



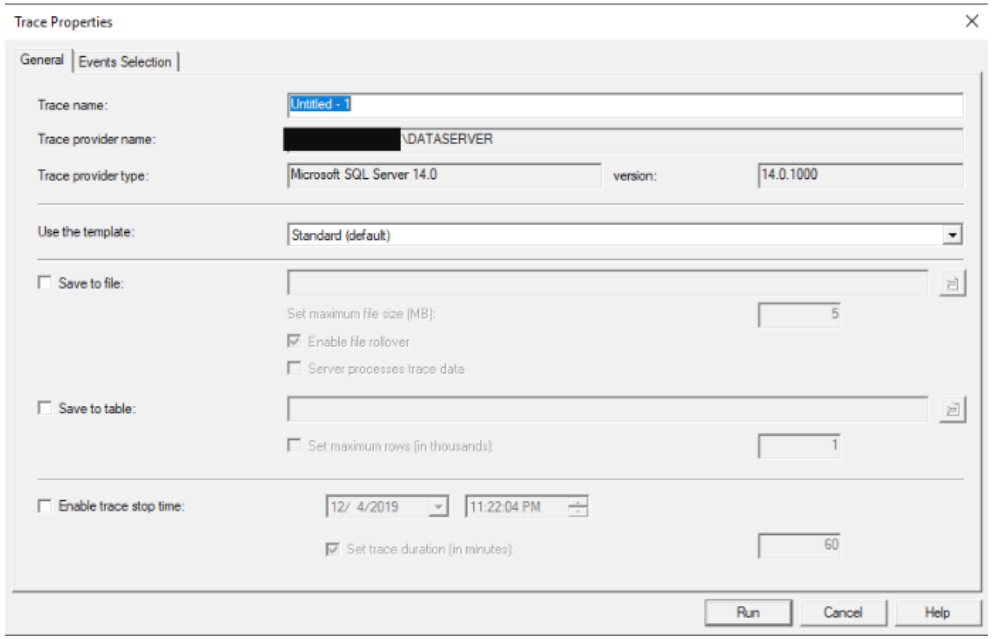
# How to use Profiler?

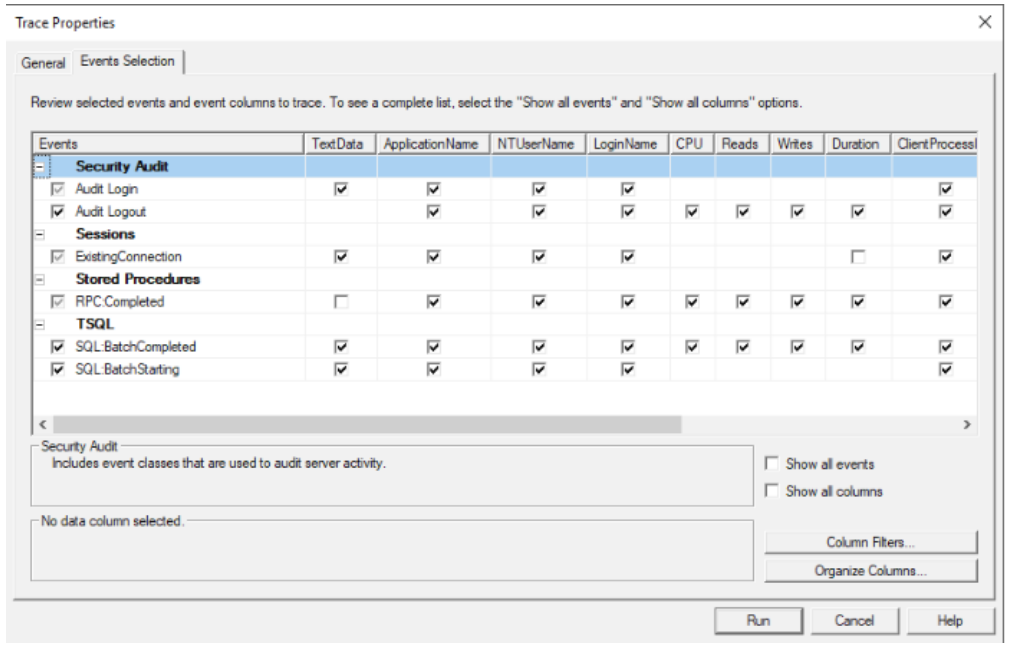
- When you open the Profiler, the authentication form is shown. You have to select whether you need to connect to an Analysis Service instance or a Database Engine.
- Then you should enter the instance name, the authentication type, and the credentials:



# How to use Profiler?

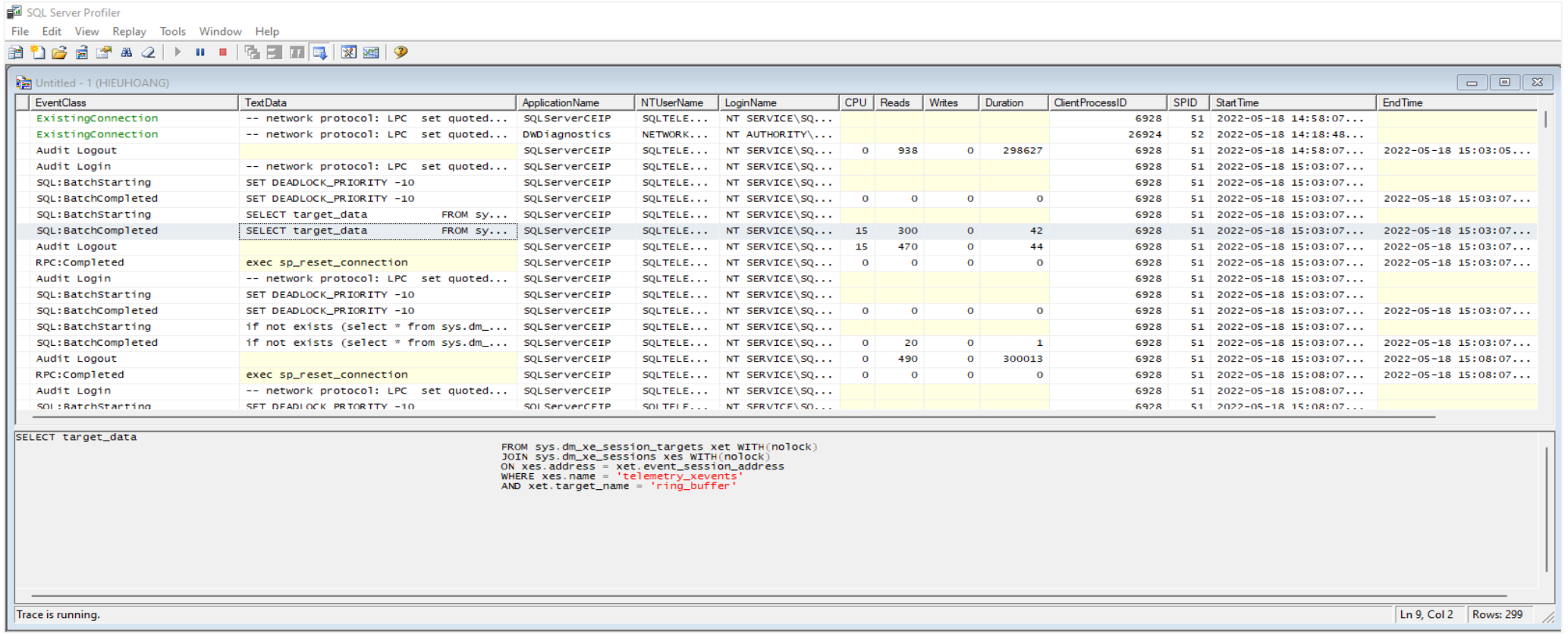
- When the connection is established, a new trace form is shown. In this form, there are two tabs: (1) **General** tab and (2) **Events Selection**.



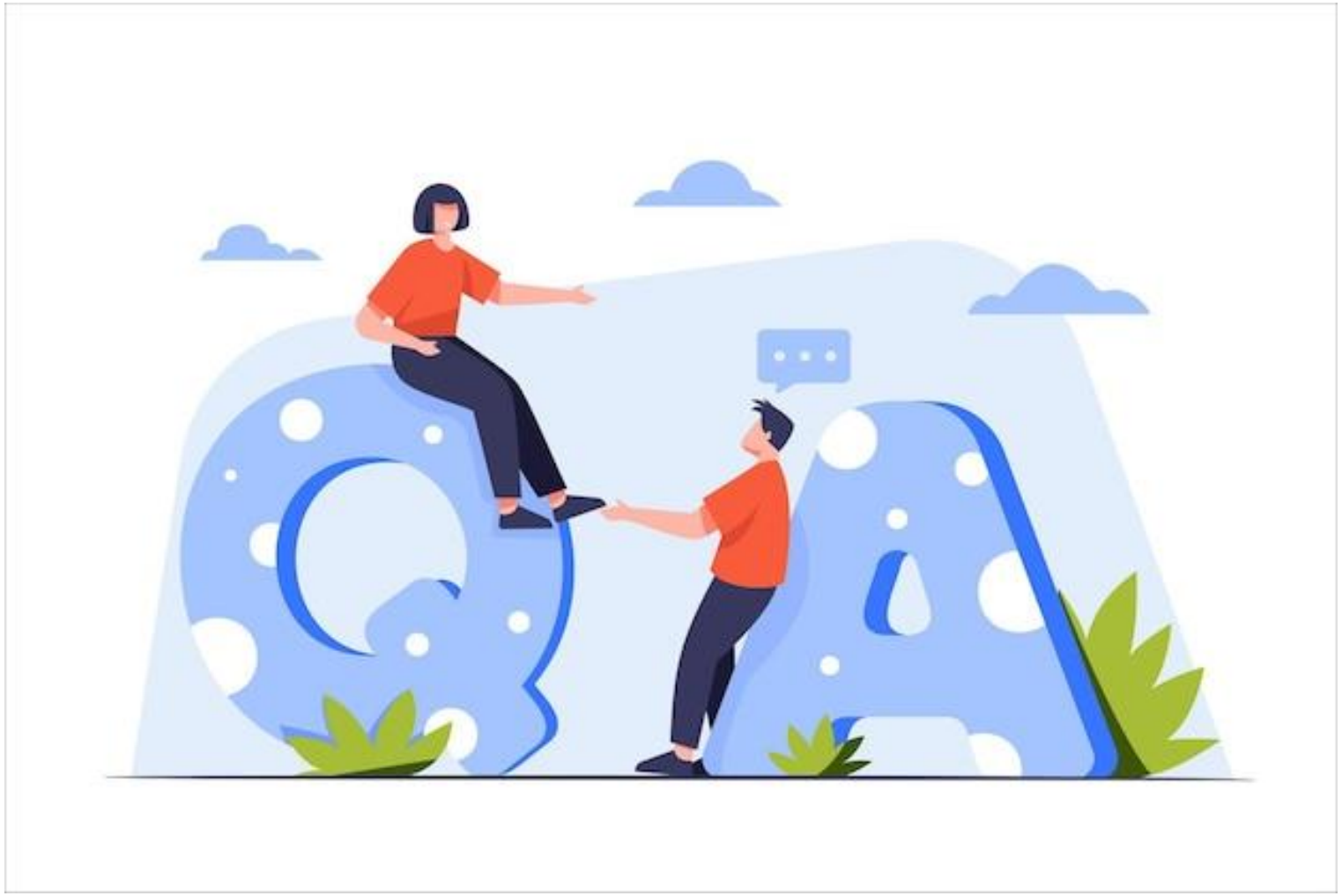



# How to use Profiler?

- Monitor the running process



# Q&A



# THANK YOU!