

Implement SQL solutions by using AI-assisted tools

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/1-introduction>

Introduction

Completed

Imagine you're a database developer working on a complex enterprise application that uses SQL Server, Azure SQL, and Microsoft Fabric. You need to write dozens of stored procedures, optimize query performance, and ensure your database code follows team standards. Traditionally, this means hours spent writing boilerplate code, searching documentation, and reviewing query plans—but AI-assisted development tools can change that workflow entirely.

GitHub Copilot and Fabric Copilot are AI assistants that understand your database context and help you write T-SQL faster and more accurately. These tools can suggest complete queries based on natural language descriptions, explain existing code, help diagnose performance issues, and even learn your team's coding standards through custom instruction files.

In this module, you'll learn how to:

- Describe AI-assisted development tools available for Microsoft SQL platforms
- Interpret the security impact of using AI-assisted tools in database development
- Enable GitHub Copilot and Fabric Copilot in your development environment
- Configure model and Model Context Protocol (MCP) tool options in chat sessions
- Create and configure GitHub Copilot instruction files for consistent AI behavior
- Connect to MCP server endpoints for SQL Server and Fabric Lakehouse

Prerequisites

- Experience writing T-SQL code for SQL Server or Azure SQL
- Familiarity with SQL Server Management Studio (SSMS) or Visual Studio Code
- Basic understanding of database development concepts (tables, stored procedures, views)

- A GitHub account (Copilot subscription recommended but not required to understand the concepts)

2. Describe AI-assisted development tools available for Microsoft SQL platforms

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/2-describe-ai-assisted-development-tools>

Describe AI-assisted development tools available for Microsoft SQL platforms

Completed

AI-assisted development tools are transforming how database professionals work with Microsoft SQL platforms. Instead of manually writing every query, debugging complex T-SQL code, or searching through documentation, you can now leverage AI assistants that understand your database context and provide intelligent suggestions in real time.

What are AI-assisted development tools?

AI-assisted development tools use large language models (LLMs) to help you write, understand, and optimize code. AI-assisted tools analyze your code context, database schema, and natural language prompts to generate relevant suggestions. For database development on SQL platforms, two primary AI assistants are available:

- **GitHub Copilot** works directly in your development environment—whether that's Visual Studio Code, Visual Studio, or SQL Server Management Studio (SSMS). It provides code completions, answers questions about your database, and helps you write T-SQL queries by understanding your schema and the patterns you use.
- **Fabric Copilot** is integrated into Microsoft Fabric workspaces and provides AI assistance for data engineering, data science, and analytics workloads. When working with SQL databases in Fabric, Copilot can help you explore data, generate queries, and understand your lakehouse structures.

How GitHub Copilot and Fabric Copilot enhance database development

Consider a scenario where you need to write a complex query joining multiple tables with specific filtering conditions. Traditionally, you might spend time reviewing table schemas, checking column names and data types, and testing different query approaches. With AI-assisted tools, you can describe what you need in natural language, and the assistant generates a starting query based on your actual database schema.



GitHub Copilot and Fabric Copilot provide several key capabilities:

- **Code completion:** As you type T-SQL, the assistant suggests complete statements, column names, and table references based on your connected database
- **Natural language to SQL:** Describe what you want to retrieve or modify, and the assistant generates the corresponding T-SQL code
- **Code explanation:** Highlight existing code and ask the assistant to explain what it does—useful when working with unfamiliar stored procedures or complex queries
- **Query optimization:** Get suggestions for improving query performance based on best practices and your specific schema
- **Error assistance:** When queries fail, the assistant can help diagnose issues and suggest fixes

AI assistants across Microsoft SQL platforms

Each Microsoft SQL platform has specific AI integration capabilities:

SQL Server and Azure SQL Database are supported through GitHub Copilot in SSMS and Visual Studio Code. You can install the [GitHub Copilot extension in SSMS](#) to get AI assistance while managing databases, writing queries, and troubleshooting performance issues.

Azure SQL Managed Instance also supports GitHub Copilot through the same development tools, allowing you to maintain consistency in your workflow whether you're working with on-premises SQL Server or managed cloud instances.

SQL databases in Microsoft Fabric benefit from both Fabric Copilot within the Fabric portal and GitHub Copilot when connecting from external tools. This dual approach means you can use the tool that best fits your current task—Fabric Copilot for quick data exploration in the portal, or GitHub Copilot for more intensive development work in your preferred IDE.

The role of Model Context Protocol (MCP)

Model Context Protocol (MCP) extends AI assistant capabilities by allowing them to connect directly to your data sources. When you configure an MCP server for your SQL database, the AI assistant can query your actual schema, sample data, and metadata to provide more accurate and contextual suggestions.

With MCP, your AI assistant isn't just working with generic SQL knowledge—it understands your specific tables, columns, relationships, and data types. This contextual awareness significantly improves the relevance and accuracy of generated code. You'll learn how to [configure MCP tool options](#) and connect to MCP server endpoints in later units.

Responsible AI considerations

When using AI-assisted tools for database development, you're working with systems that apply [responsible AI principles](#). Both GitHub Copilot and Fabric Copilot are designed with safeguards to help ensure generated code is appropriate and doesn't include harmful content.

Important

AI-generated code suggestions should always be reviewed before execution, especially for queries that modify data or affect database security. The assistant provides helpful starting points, but you remain responsible for validating that generated code meets your requirements and follows your organization's standards.

Understanding GitHub Copilot and Fabric Copilot prepares you to make informed decisions about enabling and configuring them for your database development workflows. The following units guide you through the security considerations, setup process, and advanced configuration options that help you get the most value from AI-assisted development.

3. Interpret security impact of using AI-assisted tools

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/3-interpret-security-impact>

Interpret security impact of using AI-assisted tools

Completed

Before enabling AI-assisted tools in your database development environment, you need to understand the security implications. GitHub Copilot and Fabric Copilot process code, database schemas, and potentially sensitive data patterns, making security awareness essential for responsible adoption.

How AI assistants process your data

When you use GitHub Copilot or Fabric Copilot, your prompts and code context are sent to cloud-based AI models for processing. Understanding what data flows to these services helps you make informed decisions about where and how to use AI assistance.

GitHub Copilot transmits the following data to generate suggestions:

- The code you're currently writing or editing
- Content from open files in your editor that provide context
- Database schema information when connected to a database
- Your natural language prompts and questions

Fabric Copilot processes similar data within the context of your Fabric workspace, including metadata about your lakehouses, warehouses, and semantic models.

Note

Both GitHub Copilot and Fabric Copilot are designed to protect your data. Prompts and responses are not used to train the underlying AI models, and data is encrypted in transit and at rest.

Organizational policy considerations

Your organization might have specific policies about AI tool usage, especially when working with sensitive data. Before enabling AI-assisted tools, consider these questions:

Data classification: Does your database contain personally identifiable information (PII), financial records, healthcare data, or other regulated information? Some organizations restrict AI tool usage for databases containing sensitive data classifications.

Compliance requirements: Are you subject to any compliance frameworks? Review whether using cloud-based AI services aligns with your compliance obligations.

Intellectual property: Does your organization have policies about code or schema information being processed by external AI services? Some proprietary database designs might be considered trade secrets.

Access controls: Who in your organization should have access to AI-assisted tools? You might need to align AI tool licensing with existing database access permissions.

GitHub Copilot security features

GitHub Copilot includes several security features that help protect your organization:

Content filtering: Copilot filters suggestions to avoid generating potentially harmful code patterns, including SQL injection vulnerabilities and other security anti-patterns.

Organization policies: Enterprise administrators can configure policies that control how Copilot is used across the organization, including which repositories can use Copilot and whether suggestions can include code that matches public repositories.

Audit logging: Organizations using GitHub Enterprise Cloud can track Copilot usage through audit logs, providing visibility into how the tool is being used.

For detailed information about GitHub Copilot's security practices, review the official [GitHub Copilot in SSMS documentation](#).

Protecting credentials and connection strings

One critical security practice is ensuring that AI tools never see your database credentials directly. Follow these guidelines:

- **Never paste credentials into prompts:** Don't ask the AI assistant to help with connection strings that contain actual passwords or keys
- **Use environment variables:** Store connection information in environment variables rather than hardcoding in files the AI can see
- **Review before committing:** Check AI-generated code for any accidentally included sensitive information before committing to source control

```
-- AVOID: Hardcoded credentials in AI-visible files
-- This pattern should never appear in your code
-- CREATE LOGIN username WITH PASSWORD = 'actual_password';

-- RECOMMENDED: Use parameterized approaches
-- CREATE LOGIN username WITH PASSWORD = $(password);
```

Evaluating AI suggestions critically

AI-generated code requires the same security review as any code entering your database environment. Consider these practices:

Review permissions: When Copilot suggests `GRANT` or `REVOKE` statements, verify the permissions align with your least-privilege security model.

Validate dynamic SQL: AI-generated dynamic SQL might be vulnerable to injection if not properly parameterized. Always check for appropriate use of `sp_executesql` with parameters.

Check data exposure: Review `SELECT` statements to ensure they don't inadvertently expose more data than intended, especially in views or stored procedures that might be used by applications.

Tip

Treat AI-generated code as a first draft that requires human review. The assistant optimizes for functionality and common patterns, but you know your specific security requirements.

MCP server security considerations

When configuring Model Context Protocol (MCP) servers, you establish direct connections between AI assistants and your databases. This requires careful attention to security:

Authentication: MCP connections should use the principle of least privilege. Create dedicated service accounts with read-only access when the AI only needs to query schema information.

Network security: Consider whether MCP traffic should traverse public networks. For sensitive environments, you might require private endpoints or VPN connections.

Data sampling: Some MCP configurations allow sampling data to improve suggestions. Understand what data might be transmitted and whether this aligns with your data handling policies.

Understanding these security considerations prepares you to implement AI-assisted tools in a way that aligns with your organization's security posture. The next unit guides you through the actual process of enabling GitHub Copilot and Fabric Copilot with security best practices in mind.

4. Enable GitHub Copilot and Fabric Copilot

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/4-enable-github-copilot-fabric-copilot>

Enable GitHub Copilot and Fabric Copilot

Completed

With a clear understanding of security considerations, you're ready to enable GitHub Copilot and Fabric Copilot for your database development workflows. This unit walks you through the setup process for each tool across different development environments.

Prerequisites for AI-assisted tools

Before enabling AI-assisted tools, ensure you have the required prerequisites:

For GitHub Copilot:

- A GitHub account with Copilot access (Individual, Business, or Enterprise subscription)
- A supported development environment (Visual Studio Code, Visual Studio, or SQL Server Management Studio 22+)
- An active internet connection for AI model communication

For Fabric Copilot:

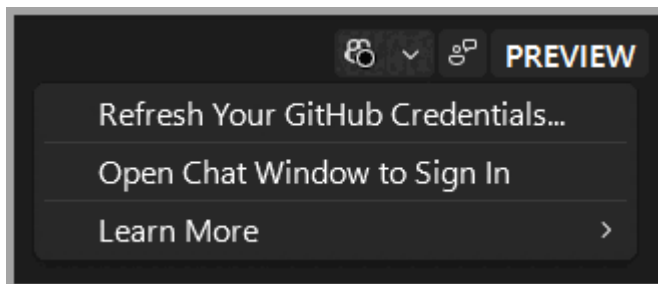
- A Microsoft Fabric capacity (F2 or higher) or Power BI Premium capacity (P1 or higher)
- Appropriate permissions in your Fabric workspace
- Copilot features enabled at the tenant level by your administrator

Enable GitHub Copilot in SQL Server Management Studio

SQL Server Management Studio (SSMS) 22 and later versions include native support for GitHub Copilot. To [install GitHub Copilot in SSMS](#), follow these steps:

1. Launch the **Visual Studio Installer** on your machine
2. Locate your SSMS installation and select **Modify**
3. On the Workloads tab, select **AI Assistance**
4. Select **Modify** to install the GitHub Copilot components

After installation, you'll see a GitHub Copilot status icon in the upper-right corner of SSMS. The icon indicates whether Copilot is active, inactive, unavailable, or not installed.



To sign in and activate Copilot:

1. Select the GitHub Copilot badge in SSMS
2. Choose **Open Chat Window to Sign In**
3. Sign in with your GitHub account that has Copilot access
4. If you don't have a Copilot subscription, select **Sign up for Copilot Free** to get started

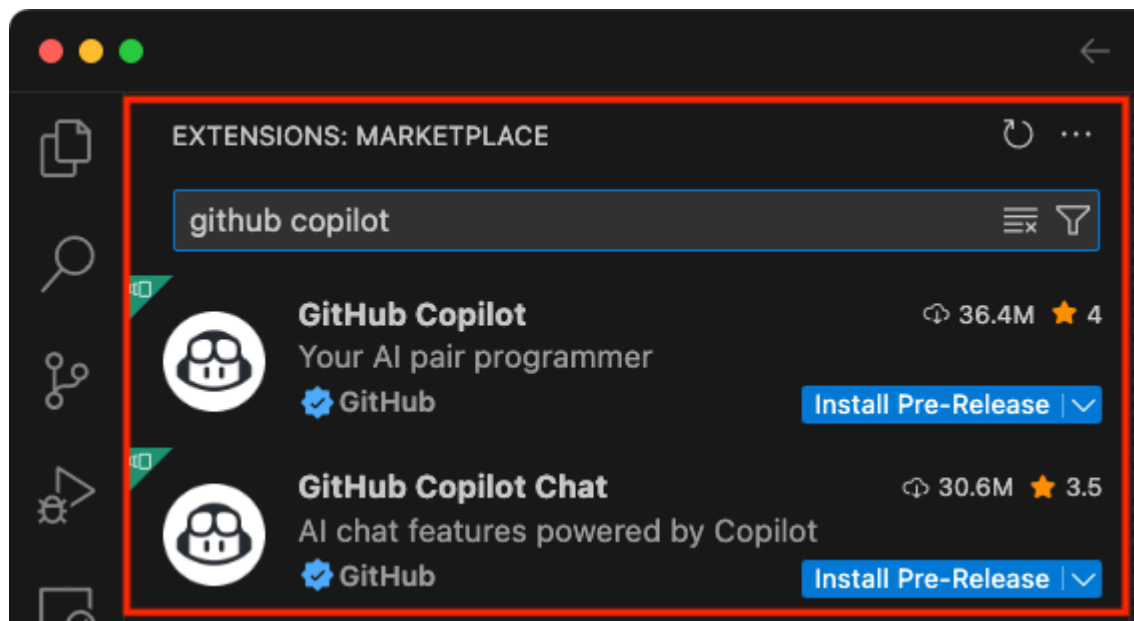
Tip

If you have multiple GitHub accounts, ensure you sign in with the account that has an active Copilot subscription. You can check your subscription status at GitHub's Copilot settings page.

Enable GitHub Copilot in Visual Studio Code

Visual Studio Code provides a flexible environment for database development with the MSSQL extension combined with GitHub Copilot. To [set up GitHub Copilot in VS Code](#):

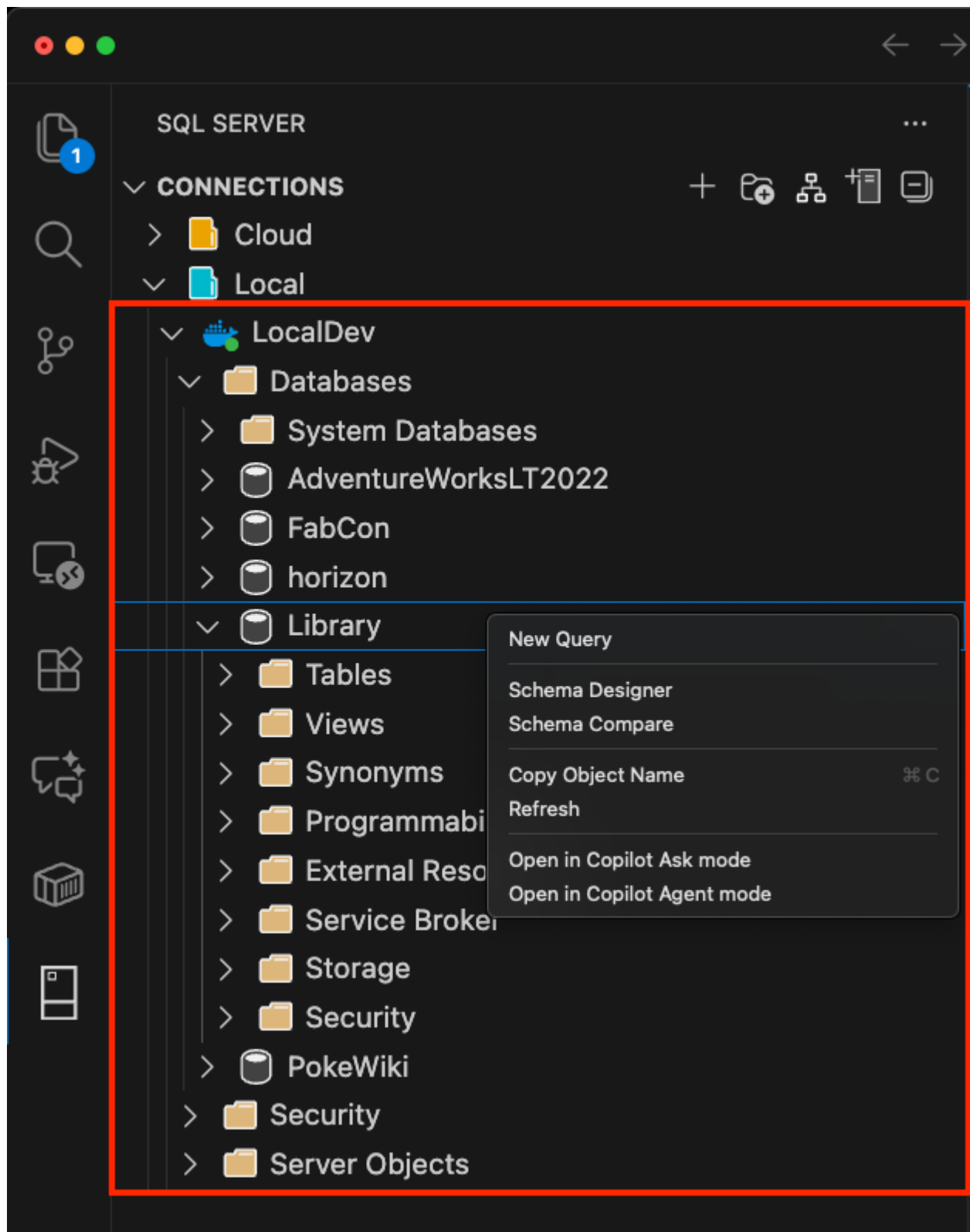
1. Open Visual Studio Code
2. Access the Extensions view (**Ctrl+Shift+X** on Windows/Linux, **Cmd+Shift+X** on macOS)
3. Search for and install the **GitHub Copilot** extension
4. Search for and install the **GitHub Copilot Chat** extension
5. Sign in to your GitHub account when prompted



For SQL database development, you'll also want the MSSQL extension:

1. In the Extensions view, search for `mssql`
2. Install the **SQL Server (mssql)** extension by Microsoft
3. Connect to your database using the Connections view

With both Copilot and MSSQL extensions installed, you can start conversations about your database directly in the Copilot Chat panel. Right-click on a connected database and select **Chat with this database** to begin a context-aware conversation.



Enable Fabric Copilot

Fabric Copilot is available within Microsoft Fabric workspaces when your organization meets the licensing requirements. Enabling Copilot involves both tenant-level and workspace-level settings.

Tenant-level configuration (requires Fabric Administrator):

1. Navigate to the Fabric Admin portal
2. Go to **Tenant settings**
3. Under the Copilot and Azure OpenAI Service section, enable **Users can use Copilot and other features powered by Azure OpenAI**
4. Configure which security groups have access to Copilot features

Workspace-level usage:

Once enabled at the tenant level, users with appropriate permissions can access Copilot within:

- Data Engineering experiences (notebooks, data pipelines)
- Data Warehouse experiences (SQL queries, semantic models)
- Data Science experiences (notebooks, experiments)

To use Copilot in a SQL database in Fabric:

1. Open your SQL database in the Fabric portal
2. Navigate to the query editor
3. Look for the Copilot button or icon in the toolbar
4. Start typing natural language queries or use the chat interface

Configure Copilot for your team

For enterprise deployments, you'll want consistent configuration across your team. Consider these setup practices:

Standardize subscriptions: Ensure all team members have the same level of Copilot access to avoid confusion about available features.

Document approved uses: Create guidelines about when and how to use AI assistance, especially for production database changes.

Set up shared instruction files: You can create custom instruction files that guide Copilot's behavior for your team's coding standards (covered in a later unit).

Verify your setup

After enabling AI-assisted tools, verify everything is working correctly:

In SSMS:

1. Connect to a database in the query editor
2. Open the Copilot Chat window (**Ctrl+\\, Ctrl+C**)
3. Ask a simple question like "What tables are in this database?"
4. Verify you receive a contextual response

In VS Code:

1. Connect to a database using the MSSQL extension
2. Open a new `.sql` file
3. Open the Copilot Chat panel (**Ctrl+Alt+I**)
4. Type a comment describing a query: `-- Get all customers from Seattle`
5. Press Enter and observe if Copilot suggests relevant SQL

In Fabric:

1. Open a SQL database in your workspace
2. Access the Copilot interface
3. Ask "Describe the tables in this database"
4. Verify the response reflects your actual schema

Note

If Copilot isn't providing contextual suggestions, verify your database connection is active and that your account has the necessary permissions to query schema information.

With GitHub Copilot and Fabric Copilot enabled, you're ready to explore advanced configuration options that customize how the AI assistants behave for your specific workflows.

5. Configure model and Model Context Protocol (MCP) tool options in a GitHub Copilot or Fabric Copilot chat session

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/5-configure-model-mcp-tool-options>

Configure model and Model Context Protocol (MCP) tool options in a GitHub Copilot or Fabric Copilot chat session

Completed

Now that you have GitHub Copilot and Fabric Copilot enabled, you can customize how they behave during chat sessions. Selecting the right model and configuring Model Context Protocol (MCP) tools significantly improves the relevance and accuracy of AI suggestions for your database work.

Understanding model selection

GitHub Copilot supports multiple AI models, each with different capabilities and performance characteristics. You can select which model to use for different scenarios based on your needs. Available models typically include:

- **GPT**: Advanced reasoning capabilities, excellent for complex T-SQL generation and database design questions
- **Claude models**: Strong at explaining code and providing detailed documentation
- **Gemini models**: Available in some configurations with different strengths

To change models in a Copilot chat session in **SSMS**:

1. Open the Copilot Chat window
2. Look for the model selector dropdown at the top of the chat
3. Select the model you want to use for your current session

To change models in a Copilot chat session in **VS Code**:

1. Open the Copilot Chat panel (**Ctrl+Alt+I**)
2. Use the model picker in the chat interface
3. Switch models as needed for different tasks

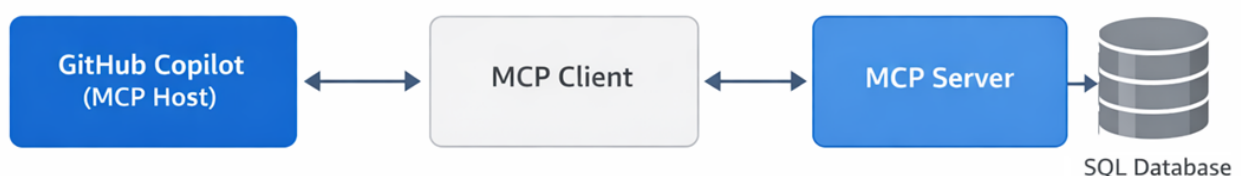
Tip

For complex query optimization or database design questions, try more advanced models. For quick code completions, faster models often provide sufficient results with lower latency.

What is Model Context Protocol (MCP)?

Model Context Protocol (MCP) is an open standard that allows AI assistants to connect directly to external data sources and tools. Instead of the AI assistant only seeing the code in your editor, MCP enables it to query your actual database schema, sample data, and metadata in real time.

With MCP configured, when you ask "What columns are in the Customers table?", the assistant can query your database directly rather than relying on whatever code context is visible in your editor. This produces more accurate and reliable suggestions.



MCP follows a client-server architecture:

- **MCP Host:** Your AI environment (GitHub Copilot, Claude, etc.)
- **MCP Client:** The protocol client that connects to MCP servers
- **MCP Server:** A service that exposes specific data sources or tools

Configure MCP tools in GitHub Copilot

GitHub Copilot supports MCP integration through agent mode in VS Code. To [use MCP servers in VS Code](#):

Enable agent mode

Agent mode allows GitHub Copilot to use external tools, including MCP servers, to gather context and perform actions on your behalf.

1. Open the Copilot Chat panel in VS Code
2. Look for the mode selector (Ask, Edit, or Agent)
3. Switch to **Agent** mode to access MCP tools

Add an MCP Server

You can add MCP servers to your VS Code workspace using the Command Palette or by manually editing the configuration file.

1. Open the Command Palette (**Ctrl+Shift+P**)
2. Type **MCP: Add Server** and select it
3. Choose the server type (HTTP or Stdio)
4. Enter the server URL or configuration

Manage MCP Tools

Once MCP servers are added, you can control which tools are active for each chat session.

1. In Agent mode, select the tools icon in the chat panel
2. View available MCP servers and their tools
3. Enable or disable specific tools as needed

When MCP tools are configured, you'll see them listed when you click the tools icon in Agent mode. The assistant can then use these tools to query your databases and provide contextual suggestions.

MCP server options for SQL databases

Several MCP server options are available for connecting AI assistants to SQL databases:

- **SQL MCP Server:** Connects to SQL Server and Azure SQL databases using Microsoft's open-source solution built on Data API builder. This option provides a secure, managed way to expose database metadata to AI assistants.

- **Microsoft Fabric MCP Server:** Connects to [Fabric data agents as MCP servers](#), enabling AI assistants to query lakehouses, warehouses, and SQL databases within Fabric workspaces.
- **Azure MCP Server:** Provides broader Azure resource integration, including database services.

To configure a SQL MCP server in VS Code:

1. Create a `.vscode` folder in your workspace (if it doesn't exist)
2. Create a file named `mcp.json` in the `.vscode` folder
3. Add your server configuration:

```
{
  "servers": {
    "sql-server": {
      "type": "http",
      "url": "https://your-mcp-endpoint/mcp"
    }
  }
}
```

4. VS Code detects the configuration and prompts you to start the server
5. Authenticate when prompted

Configure MCP in Fabric Copilot

Fabric Copilot automatically has access to your workspace resources, but you can publish data agents as MCP servers for use in external tools:

1. Create and configure a data agent in your Fabric workspace
2. Publish the data agent
3. Go to the agent's **Settings** and open the **Model Context Protocol** tab
4. Copy the **MCP server URL**
5. Use this URL to configure MCP clients in VS Code or other tools

This approach allows you to use the same data agent from both the Fabric portal and external development environments.

Best practices for model and MCP configuration

- **Start with defaults:** Begin with default model and MCP settings, then adjust based on your experience. Not every scenario requires the most advanced configuration.
- **Match models to tasks:** Use more capable models for complex design decisions, simpler models for routine code completion.
- **Limit MCP scope:** Configure MCP connections with least-privilege access. If your AI workflow only needs schema information, don't grant data read permissions.

- **Test configurations:** After changing model or MCP settings, verify the assistant still provides accurate suggestions. Different models might interpret prompts differently.

Important

MCP servers establish direct connections to your databases. Ensure your network security policies allow these connections and that authentication follows the security best practices covered earlier in this module.

With model selection and MCP tools configured, your AI assistant has the context it needs to provide accurate, database-specific suggestions. The next unit covers how to create custom instruction files that further guide the assistant's behavior.

6. Create and configure GitHub Copilot instruction files

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/6-create-configure-copilot-instruction-files>

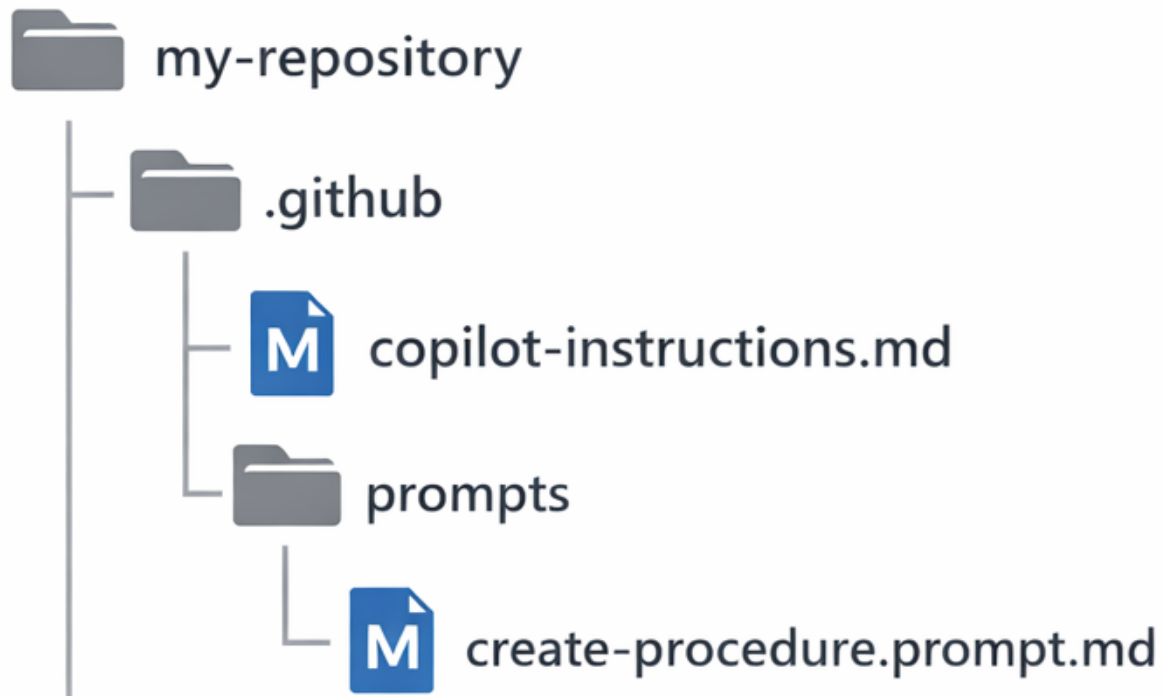
Create and configure GitHub Copilot instruction files

Completed

Custom instruction files provide a way to guide GitHub Copilot's behavior consistently across your projects and team. Instead of repeating the same context in every prompt, you define your preferences, coding standards, and project-specific guidance in files that Copilot reads automatically.

What are custom instruction files?

Custom instruction files are markdown documents that contain guidance for GitHub Copilot. When you place these files in specific locations, Copilot includes their contents as context when generating suggestions. This means you can influence how Copilot responds without modifying each prompt.



There are two main types of instruction files:

- **Repository instructions** (`copilot-instructions.md`): Apply to everyone working in a specific repository. Stored in the `.github` folder at the repository root.
- **Prompt files** (`.prompt.md`): Reusable prompts for specific tasks or scenarios. Stored in the `.github/prompts` folder and can be referenced in chat conversations.

Create a repository instruction file

To [create custom instructions for your repository](#), follow these steps:

1. Navigate to your repository root folder
2. Create a `.github` folder if it doesn't exist
3. Create a file named `copilot-instructions.md`
4. Add your instructions in markdown format

Here's an example instruction file for a database project:

```
# Project Guidelines for Copilot

## Database Development Standards

This project uses SQL Server 2022 with the following conventions:

### Naming Conventions
```

- Tables: PascalCase, singular (Customer, OrderDetail)
- Columns: PascalCase (FirstName, OrderDate)
- Stored procedures: usp_ActionEntity (usp_GetCustomerOrders)
- Views: vw_EntityName (vw_ActiveCustomers)
- Indexes: IX_TableName_ColumnName

T-SQL Style

- Use explicit column lists in SELECT statements (avoid SELECT *)
- Always include schema prefix (dbo.TableName)
- Use ANSI JOIN syntax, not comma-separated tables
- Include error handling in all stored procedures
- Use TRY...CATCH blocks for data modification operations

Security Requirements

- Never generate GRANT statements to public
- Use parameterized queries, never concatenate user input
- Avoid dynamic SQL when possible

Performance Guidelines

- Suggest appropriate indexes when creating tables
- Prefer SET NOCOUNT ON in stored procedures
- Use EXISTS instead of COUNT for existence checks

When Copilot generates T-SQL code in this repository, it considers these guidelines and produces suggestions that align with your standards.

Configure instruction file settings

In Visual Studio and VS Code, you can configure how Copilot uses instruction files:

In VS Code:

1. Open Settings (**Ctrl+,**)
2. Search for "GitHub Copilot: Instructions"
3. Configure which instruction files to include automatically

In Visual Studio:

1. Go to **Tools > Options**
2. Navigate to **GitHub > Copilot**
3. Configure custom instruction preferences

You can also reference instruction files explicitly in chat using the Command Palette (**Ctrl+Shift+P**) and selecting **Chat: Configure Instructions**.

Create reusable prompt files

Prompt files let you define templates for common tasks. For database development, you might create prompts for:

- Generating stored procedure templates
- Creating data migration scripts
- Reviewing query performance
- Documenting existing code

To create a prompt file:

1. Create a `.github/prompts` folder in your repository
2. Create a file with `.prompt.md` extension
3. Define the prompt template with any required parameters

Example prompt file for creating a stored procedure (`create-procedure.prompt.md`):

```
# Create Stored Procedure

Generate a stored procedure with the following specifications:

- Procedure name: {{procedureName}}
- Purpose: {{description}}
- Parameters: {{parameters}}

Follow these requirements:
- Include TRY...CATCH error handling
- Add SET NOCOUNT ON at the beginning
- Include appropriate comments
- Follow our naming conventions (usp_ prefix)
- Use explicit transactions for data modifications
```

To use this prompt in VS Code, reference it in the chat panel with the `#` symbol or through the prompt picker.

Team-wide instruction strategies

For enterprise teams, consider these approaches to instruction file management:

- **Centralized standards:** Maintain a master instruction file in a shared repository that gets copied or linked to project repositories.
- **Layered instructions:** Use organization-level instructions for general standards, with repository-specific additions for project requirements.
- **Version control:** Track instruction file changes in git so you can review how your AI guidance evolves over time.

Tip

Review your instruction files periodically. As your team's practices evolve or as Copilot improves, you might need to adjust your guidance.

Best practices for instruction files

To get the most out of custom instruction files, follow these best practices:

Be specific: Vague instructions produce vague results. Instead of "use good naming," specify your exact naming pattern.

Provide examples: Show Copilot what you want rather than just describing it. Include code snippets that demonstrate your preferred patterns.

Prioritize: Put your most important guidelines first. Copilot considers the full context, but prominent placement helps ensure critical rules are followed.

Test iteratively: After creating instruction files, test various prompts to see if Copilot follows your guidance. Adjust wording if results aren't as expected.

Keep current: Update instructions when your standards change. Outdated guidance leads to inconsistent suggestions.

Example of a specific, example-driven instruction:

```
## Error handling pattern

Always use this error handling pattern in stored procedures:

```sql
CREATE PROCEDURE dbo.usp_ExampleProcedure
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION;

 -- Procedure logic here

 COMMIT TRANSACTION;
 END TRY
 BEGIN CATCH
 IF @@TRANCOUNT > 0
 ROLLBACK TRANSACTION;

 -- Log error details
 DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
 DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
 DECLARE @ErrorState INT = ERROR_STATE();

 RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
 END CATCH;
END;
```

This pattern includes explicit transaction handling and proper error propagation.

With well-crafted instruction files, your team gets consistent AI assistance that :

## 7. Connect to MCP server endpoints, including Microsoft SQL Server and Fabric Lakehouse

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/7-connect-mcp-server-endpoints>

# Connect to MCP server endpoints, including Microsoft SQL Server and Fabric Lakehouse

Completed

Model Context Protocol (MCP) servers provide AI assistants with direct access to your data sources, enabling more accurate and contextual suggestions. Building on the MCP concepts from the previous units, this unit covers how to connect to MCP server endpoints for Microsoft SQL Server, Azure SQL databases, and Fabric Lakehouse environments.

## Connect to SQL Server MCP endpoints

For SQL Server and Azure SQL databases, you can use the [SQL MCP Server](#) or configure direct connections through supported tools.

### Using SQL MCP Server:

SQL MCP Server is Microsoft's open-source solution built on Data API builder that enables AI agents to interact with SQL databases. To set up [SQL MCP servers in VS Code](#):

1. Install and configure SQL MCP Server for your database
2. Enable the MCP endpoint in your SQL MCP Server configuration
3. Note the MCP endpoint URL (typically ending in `/mcp`)
4. Add the server to VS Code using the MCP: Add Server command

### Direct SQL Server connection:

As covered in the earlier unit on enabling GitHub Copilot, you can use the MSSQL extension's built-in integration in VS Code. Right-click your connected database and select **Chat with this database** to start schema-aware conversations without configuring a separate MCP server.

## Connect to Azure SQL MCP endpoints

Azure SQL databases support the same connection methods as SQL Server. For cloud deployments, consider these additional options:

### Using Azure MCP Server:

The Azure MCP Server provides integration across Azure services, including SQL databases:

1. Install the Azure MCP Server extension in VS Code
2. Authenticate with your Azure account
3. Configure access to your Azure SQL resources
4. Use Agent mode to query your databases

### Configuring network access:

Azure SQL databases require network configuration for MCP connections:

- Ensure your client IP is allowed through the Azure SQL firewall
- For private endpoints, configure your network to allow MCP traffic
- Consider using service principals for automated or shared MCP access

```
{
 "servers": {
 "azure-sql-mcp": {
 "type": "http",
 "url": "https://your-api-endpoint.azurewebsites.net/mcp",
 "headers": {
 "Authorization": "Bearer ${input:azure_token}"
 }
 }
 },
 "inputs": [
 {
 "id": "azure_token",
 "type": "promptString",
 "description": "Azure access token",
 "password": true
 }
]
}
```

## Connect to Fabric Lakehouse MCP endpoints

Microsoft Fabric provides MCP server capabilities through data agents, allowing AI assistants to query lakehouses, warehouses, and semantic models.

### Publishing a Fabric data agent as MCP server:

1. Create a data agent in your Fabric workspace

2. Configure the data agent with access to your lakehouse or warehouse
3. Publish the data agent
4. Navigate to the agent's **Settings > Model Context Protocol** tab
5. Copy the **MCP server URL** and **tool description**

### Configuring VS Code to connect:

Create or update your `.vscode/mcp.json` file:

```
{
 "servers": {
 "fabric-lakehouse": {
 "type": "http",
 "url": "https://your-fabric-mcp-endpoint-url"
 }
 }
}
```

When VS Code detects this configuration:

1. Select **Add Server** when prompted
2. Authenticate with your Microsoft account
3. The Fabric MCP server appears in your available tools

### Using the [Microsoft Fabric MCP Server extension](#):

For a streamlined experience, install the Fabric MCP Server extension from the VS Code marketplace. This extension simplifies configuration and provides better integration with Fabric workspaces.

## Configure authentication for MCP endpoints

MCP servers require authentication to access your data sources. Common authentication methods include:

**Interactive authentication:** You sign in through a browser prompt when connecting. Best for development environments.

**Service principal:** Configure a Microsoft Entra ID application with appropriate permissions. Best for automated scenarios and shared environments.

**API keys:** Some MCP servers support API key authentication. Store keys securely using environment variables or VS Code's input prompts.

### Important

Never store credentials directly in configuration files that might be committed to source control. Use environment variables, input prompts, or secure credential stores. For more guidance on protecting credentials, see the security considerations covered earlier in this module.

## Test your MCP connections

After configuring MCP endpoints, verify the connection is working:

### In VS Code Agent mode:

1. Switch to Agent mode in the Copilot Chat panel
2. Click the tools icon to see available MCP servers
3. Verify your configured servers appear with a green status
4. Ask a question like "What tables are available in the database?"
5. Confirm the response reflects your actual database schema

### Troubleshooting common issues:

Issue	Likely Cause	Solution
Server not listed	Configuration file syntax error	Validate JSON in mcp.json
Authentication failed	Expired credentials	Re-authenticate or refresh tokens
Connection timeout	Network/firewall blocking	Check firewall rules and network access
Empty schema results	Insufficient permissions	Verify database permissions for the authenticated user

## Best practices for MCP endpoint management

**Use least-privilege access:** Create dedicated accounts for MCP connections with only the permissions needed (typically read-only schema access).

**Separate environments:** Configure different MCP endpoints for development, test, and production databases. Avoid connecting AI assistants to production data during routine development.

**Monitor connections:** Track MCP server usage through logging and monitoring. Understand how often your AI assistant queries your databases.

**Keep configurations consistent:** For team environments, share MCP configurations through version control so everyone connects to the same endpoints.

### Tip

Start with a development or test database when learning MCP configuration. Once you're comfortable with the setup, extend to additional environments as needed.

With MCP endpoints configured, your AI assistant now has real-time access to your database schemas and metadata. Combined with custom instruction files from the previous unit, you have a fully configured, contextual AI development environment for your Microsoft SQL platforms.

## 8. Exercise - Configure AI-assisted tools for database development

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/8-exercise-design-implement-sql-solutions-ai-assisted-tools>

# Exercise - Configure AI-assisted tools for database development

---

Completed

In this exercise, you practice using AI-assisted development tools to design and implement SQL solutions. You work with GitHub Copilot in Visual Studio Code to generate T-SQL code, configure custom instruction files, and connect to MCP server endpoints for contextual database assistance.

### Note

To complete this exercise, you need access to SQL Server 2022 or later, Visual Studio Code with the GitHub Copilot and MSSQL extensions installed, and an active GitHub Copilot subscription.

Launch the exercise and follow the instructions.

[Launch Exercise](#)

## 9. Module assessment

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/9-knowledge-check>

## Module assessment

---

Completed

### 10. Summary

<https://learn.microsoft.com/en-us/training/modules/design-implement-sql-solutions-ai-assisted-tools/10-summary>

## Summary

---

Completed

In this module, you learned how to:

- Describe AI-assisted development tools (GitHub Copilot and Fabric Copilot) available for SQL Server, Azure SQL, and SQL databases in Microsoft Fabric
- Interpret the security impact of using AI tools, including data processing, organizational policies, and credential protection best practices
- Enable GitHub Copilot in SSMS and VS Code, and configure Fabric Copilot for your workspaces
- Configure model selection and Model Context Protocol (MCP) tools to enhance AI suggestions with real-time database context
- Create custom instruction files that guide Copilot's behavior for your team's coding standards
- Connect to MCP server endpoints for SQL Server, Azure SQL, and Fabric Lakehouse environments

### Key takeaways

- AI-assisted tools can significantly accelerate T-SQL development by providing contextual code suggestions, explanations, and optimization recommendations
- Security awareness is essential—never expose credentials to AI tools and always review generated code before execution
- MCP enables AI assistants to access your actual database schema, producing more accurate suggestions than generic AI assistance

- Custom instruction files provide consistent AI behavior across your team by codifying coding standards and project-specific guidelines

## Learn more

- [Install GitHub Copilot in SQL Server Management Studio](#)
- [Set up GitHub Copilot in VS Code](#)
- [Use SQL MCP servers in VS Code](#)
- [Microsoft Fabric MCP Server documentation](#)
- [Responsible AI principles and practices](#)