IT資格道場

DP-800 学習用テキスト

予備知識ゼロから合格ラインへ ―― 読んで理解する1冊目

無料サンプル(第1章まで)

DP-800 出題範囲(Skills measured)2026-03-12版 準拠
2026年11月版

SAMPLE

© IT資格道場｜無料サンプル｜転載・再配布・AI学習利用禁止

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

SAMPLE

目次

この教材の使い方

- 試験の基本情報
- 試験の設計図：3 分野と配点
- 構成
- 4 ステップ学習法

第1章 Design and develop database solutions [1.1.1/1.1.2/1.1.3/1.1.4/1.1.5/1.1.6/1.2.1/1.2.2/1.2.3/1.2.4/1.2.5/1.3.1/1.3.2/1.3.3/1.3.4/1.3.5/1.3.6/1.3.7/1.3.8/1.4.1/1.4.2/1.4.3/1.4.4/1.4.5]

- 表の設計：データ型・列・インデックス・列ストア [1.1.1]
- 特殊な表：メモリ最適化・テンポラル・外部・台帳・グラフ [1.1.2]
- JSON 列と JSON インデックス [1.1.3]
- 制約：PRIMARY KEY・FOREIGN KEY・UNIQUE・CHECK・DEFAULT [1.1.4]
- SEQUENCE [1.1.5]
- パーティション分割 [1.1.6]
- ビュー [1.2.1]
- スカラー関数 [1.2.2]
- テーブル値関数 [1.2.3]
- ストアド プロシージャ [1.2.4]
- トリガー [1.2.5]
- 共通テーブル式（CTE） [1.3.1]
- ウィンドウ関数 [1.3.2]
- JSON 関数 [1.3.3]
- 正規表現 [1.3.4]
- あいまい一致の関数 [1.3.5]
- グラフ クエリと MATCH [1.3.6]

SAMPLE

相関クエリ [1.3.7]
エラー処理 [1.3.8]
AI 支援ツールを使うときのセキュリティ [1.4.1]
GitHub Copilot と Copilot in Fabric を使えるようにする [1.4.2]
チャットのモデルと MCP のツールの選択 [1.4.3]
Copilot の指示ファイル [1.4.4]
MCP サーバーへの接続 [1.4.5]

サンプルはここまでです

SAMPLE

3 冊の使い方は、ダウンロード欄の案内を参照してください。

SAMPLE

この教材の使い方

このテキストの目的は、Microsoft の Associate 資格への合格です。

Microsoft Certified: SQL AI Developer Associate(試験コード DP-800、試験名 Developing AI-Enabled Database Solutions)

DP-800 は、**データベースの開発者の視点で、AI を組み込んだデータベースソリューションをつくる**資格。対象は SQL Server・Azure SQL・Fabric の SQL データベースです。製品名の暗記は問われない。問われるのは、「この要件なら、どの句・設定・操作か」です。

対象の製品は 3 つ。

- SQL Server 2025
- Azure SQL(SQL Database と SQL Managed Instance)
- Fabric の SQL データベース

同じ T-SQL でも、製品によって使える機能が違うことがあります。このテキストは、その違いを軸に、公式の出題範囲(3 分野)に沿って進める構成。

> 公式の想定読者は、SQL のデータベースで表・処理・セキュリティ・デプロイ・AI 機能を実装する開発者です。

深い AI の知識は要りません。押さえるのは 2 点です。

- 各機能の「何をする道具か」
- 「選ぶ条件・設定の勘どころ」

SAMPLE

この 2 点で合格ラインに届きます。T-SQL や設定ファイルのコード例は出ますが、**丸暗記は不要。**「どの句が何をするか」が読めれば十分です。

試験の基本情報

項目	内容
試験名	Microsoft Certified: SQL AI Developer Associate(DP-800)
出題数	50 問
出題形式	択一・複数選択のほか、ドラッグ アンド ドロップ、並べ替え、ホットエリア、アクティブ スクリーン、ケーススタディ、「問題と解決策」が含まれることがあります。試験ごとの形式は事前に公表されません
試験時間	120 分
合格ライン	1,000 点満点中 **700 点**(スケールスコア)
受験方法	ピアソン VUE。テストセンターまたはオンライン監督試験
言語	日本語で受験可(英語ほか多言語提供)

分からない問題も、必ずどれかを選んで埋めること。受験料や実施形式は変わることがあります。申し込み時に、公式ページの最新情報を必ず確認してください。

試験の設計図：3 分野と配点

このテキストが準拠する Skills measured は **2026 年 3 月 12 日版**。

SAMPLE

分野	配点	一言でいうと
① データベース ソリューションの設計と開発	**35〜40%**	設計して書く —— 表・オブジェクト・T-SQL・AI 支援ツール
② データベース ソリューションのセキュリティ保護、最適化、デプロイ	**35〜40%**	守って出す —— 暗号化・アクセス制御・性能・CI/CD・Azure 連携
③ データベース ソリューションに AI 機能を実装する	**25〜30%**	AI をつなぐ —— 外部モデル・埋め込み・ベクトル検索・RAG

①と②が 35〜40% で最大です。この 2 分野で全体の大半を占め、③が 25〜30% で続く形。

配点の偏りは小さく、**どれか 1 分野を捨てる作戦は取れない**。時間が限られるなら、①②を先に、次に③の順で仕上げると効率的。

構成

このテキストは 3 分野を 3 つの章に分けた構成。同じ機能が複数の分野にまたがることもあります(Data API builder は②の配置にも③の RAG にも出ます)。並べ方は**公式の分野順そのまま**。

- 第 1 章 … 分野①(データベース ソリューションの設計と開発)
- 第 2 章 … 分野②(セキュリティ保護、最適化、デプロイ)
- 第 3 章 … 分野③(AI 機能の実装)

見出しの末尾の [1.1.1] などの数字は、公式の出題範囲の番号です。

SAMPLE

4 ステップ学習法

購入者限定テキストは 3 冊で 1 セット。次の順番で使うと最短です。

- **STEP 1(理解する)**:「学習用テキスト」を通読し、SQL と AI 機能の地図と「選ぶ条件・設定の勘どころ」を頭に入れる
- **STEP 2(解き方を掴む)**:「頻出度別ノート」で頻出度 A の論点を解いて、解き方を掴む
- **STEP 3(腕試し)**: 本サイトの問題演習で出題形式に慣れる
- **STEP 4(再確認)**:「頻出度別ノート」に戻って頻出度 A の論点を再確認する(試験直前もここ)

「暗記用ノート」は STEP には数えず、空き時間や用語の再確認に使ってください。

DP-800 は操作と書き方を問う試験。可能なら Azure の無料アカウントで SQL Database を作り、実際に触ると定着が段違いです。時間がなければ、先に「頻出度別ノート」で頻出度 A の論点を確認。それを含む章から読み始めても構いません。

SAMPLE

第1章 Design and develop database solutions [1.1.1/1.1.2/1.1.3/1.1.4/1.1.5/1.1.6/1.2.1/1.2.2/1.2.3/1.2.4/1.2.5/1.3.1/1.3.2/1.3.3/1.3.4/1.3.5/1.3.6/1.3.7/1.3.8/1.4.1/1.4.2/1.4.3/1.4.4/1.4.5]

分野①｜配点 35〜40%｜出題範囲 1.1.1〜1.4.5

この章は、SQL Server・Azure SQL・Fabric の SQL データベースで、表と処理を設計して書く力を扱います。試験全体の 35〜40% を占める最大の分野です。出題の多くは、要件を読んで「この句・この型・この関数」を選ぶ形をとります。構文を暗記するより、各機能の前提と制限を押さえておくと、場面問題で迷いません。

対象の製品は、Azure SQL を 2 つに分けて数えると、次の 4 つです。

- SQL Server 2025
- Azure SQL Database
- Azure SQL Managed Instance
- Fabric の SQL データベース

同じ T-SQL でも、製品によって使える範囲が違う機能があり、差は該当する節で示します。

SAMPLE

表の設計：データ型・列・インデックス・列ストア [1.1.1]

データ型と列の設計

表の設計は「どの型で、どの大きさで持つか」から始まります。ディスクベースの表では、1 行の最大は 8,060 バイトです。可変長の列は row-overflow storage で行の外に押し出せるので、列の合計がそれを超えても、表が作れなくなるわけではありません。1 表の列数は通常 1,024 までで、sparse column set を含む表だけが最大 30,000 列になります。[1]

列の型は、用途ごとに候補が決まっています。

用途	主な型
文字列	varchar（非 Unicode）・nvarchar（Unicode）。大きな値は varchar(max)・nvarchar(max)
数値	int・bigint・decimal(p,s)・float
日時	date・time・datetime2・datetimeoffset
半構造化	json
AI 向け	vector

行と列の大きさの上限は、次の「1 行 8,060 バイト」と「1 表 1,024 列」です。[2]

AI 向けの型が vector です。`VECTOR(dimensions[, base_type])` の形で宣言し、基本型を省略すると float32 になり、float16 を明示すると半精度で持てます。次元数の上限は、float32 が 1,998、float16 が 3,996 です。0 次元の宣言はできません。内部は最適化されたバイナリ形式で、表示は JSON 配列です。各要素は単精度（4 バイト）の浮動小数点値として数えます。[3]

SAMPLE

float16 には製品差があります。SQL Server 2025 ではプレビューで、PREVIEW_FEATURES データベース スコープ構成を ON にしないと使えません。Azure SQL Database・Managed Instance・Fabric の SQL データベースでは GA です。Fabric の SQL データベースでは、vector や json の列を持つ表は、今は OneLake にミラーされません。

```
CREATE TABLE dbo.Docs (
    DocId   int           NOT NULL PRIMARY KEY,
    Title   nvarchar(200) NOT NULL,
    Emb     vector(1536)  NULL        -- 基本型は float32
);
```

間違えやすいのは上限の取り違えです。float32 の 1,998 と float16 の 3,996 は、半精度のぶん倍の次元が入る関係にあります。

インデックスの設計

インデックスの型は 2 つに分かれます。クラスター化は表の行そのものを並べ、1 表に 1 つしか作れません。非クラスター化は別の構造で、1 表に 999 まで作れます。キーの大きさの上限は、インデックスの種類で違います。

- クラスター化インデックス：キー 900 バイトまで
- ディスクベース表の非クラスター化インデックス：キー 32 列・1,700 バイトまで
- メモリ最適化テーブルの非クラスター化インデックス：2,500 バイトまで [1]

キー列に入れられない型や、キーの上限を使いたくない列は、付加列（INCLUDE 列）に置きます。付加列はキーの列数とキー サイズの計算に入りません。text・ntext・image 以外の型なら置けます。付加列は非クラスター化

SAMPLE

インデックスにだけ付けられ、検索に使う列がキー、結果に返すだけの列が付加列という分担になります。クエリの必要な列がすべてインデックスにそろうと、表本体を読まずに済み、この状態を「クエリをカバーする」と呼びます。[4]

```
CREATE NONCLUSTERED INDEX IX_Address_PostalCode
ON Person.Address (PostalCode)
INCLUDE (AddressLine1, AddressLine2, City, StateProvinceID);
```

フィルター選択インデックス（フィルター付きインデックス）は、WHERE 句で行の一部だけを索引にする非クラスター化インデックスです。NULL がほとんどの列で非 NULL だけを引く場合、処理済みの印が付いた行が大半で未処理だけを引く場合、種類の混ざった行の一種類だけを引く場合に向きます。サイズと維持費が小さくなり、統計も絞った行で取れます。[5]

```
CREATE NONCLUSTERED INDEX FIBillOfMaterialsWithEndDate
ON Production.BillOfMaterials (ComponentID, StartDate)
WHERE EndDate IS NOT NULL;
```

制限は 4 つ覚えます。ビューには作れません。フィルターに使えるのは単純な比較演算子だけで、LIKE は使えません。NULL との比較は `= NULL` と書けず、`IS NULL`・`IS NOT NULL` で書きます。計算列はフィルター定義に書けません。主キー制約と一意制約にフィルターは付けられませんが、UNIQUE のインデックスには付けられます。クエリの WHERE がインデックスの WHERE の部分集合になっていないと、インデックスは使われません。インデックス付きビューとの違いは、フィルター付きは非一意でもよく、オンラインで再構築できる点です。

SAMPLE

列ストア インデックス

列ストア インデックスは、列ごとに圧縮して持つ構造で、分析クエリに向きます。種類は 2 つで、クラスター化列ストアは表全体の主ストレージ、非クラスター化列ストアは行ストアの表に付ける二次インデックスです。後者を OLTP の表に付けると、同じ表で更新と分析を混在させられます。更新の多い OLTP の表で分析もしたいときは、非クラスター化列ストアを足せばよく、行ストアの主ストレージを置き換える必要はありません。[6]

行は行グループ（rowgroup）単位で扱います。1 つの行グループが持てるのは最大 1,048,576 行までで、それを超えて詰めることはできません。一括ロードで列ストアへ直接入る最小の行数は 102,400 行で、それ未満の少量ロードは deltastore に入ります。deltastore は delta rowgroup の集まりで、バックグラウンドの tuple-mover が、閉じた delta rowgroup を COMPRESSED の行グループへ圧縮します。delta rowgroup も 1,048,576 行の閾値に達すれば、列ストアへ移されます。

項目	値	読み方
行グループの最大行数	1,048,576	満杯の行グループ
直接圧縮に入る最小行数	102,400	未満は deltastore
小さな書き込みの置き場	delta rowgroup（deltastore）	tuple-mover が圧縮

間違えやすいのは、102,400 行未満のロードが圧縮済みにならない点で、少量の挿入を繰り返しても、一括ロードのようには圧縮されません。

SAMPLE

特殊な表：メモリ最適化・テンポラル・外部・台帳・グラフ [1.1.2]

メモリ最適化テーブル

メモリ最適化テーブルは、`MEMORY_OPTIMIZED = ON` で作る、メモリに載せる表で、永続性は DURABILITY で選びます。既定の SCHEMA_AND_DATA は、再起動やフェイルオーバーの後も、データは消えません。SCHEMA_ONLY はスキーマだけを保持し、再起動でデータが消えるので、セッション データや一時的な作業表に向く選択肢です。SCHEMA_ONLY は MEMORY_OPTIMIZED = ON のときにしか指定できません。[7]

ハッシュ インデックスの BUCKET_COUNT の最大は 1,073,741,824 です。この表を速く呼ぶ道が、ネイティブ コンパイルのストアド プロシージャです。`NATIVE_COMPILATION` と `SCHEMABINDING` を付け、本体を 1 つの `BEGIN ATOMIC WITH (...)` で囲み、トランザクション分離レベルと言語を指定します。`EXECUTE AS` は任意で、下の例では OWNER を付けています。

```
CREATE PROCEDURE dbo.usp_Add @id int
WITH NATIVE_COMPILATION, SCHEMABINDING, EXECUTE AS OWNER
AS BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE
= N'us_english')
    DELETE FROM dbo.HotTable WHERE Id = @id;
END;
```

Fabric の SQL データベースでは、in-memory・ledger・ledger history・Always Encrypted の表は作れません。[8]

SAMPLE

テンポラル テーブル

テンポラル テーブル（システム バージョン管理テーブル）は、行の変更履歴を自動で別表に残す表です。必須の要素は 2 つあります。主キーと、datetime2 型の 2 列で定義する `PERIOD FOR SYSTEM_TIME` です。2 列は `GENERATED ALWAYS AS ROW START` と `ROW END` で宣言し、`SYSTEM_VERSIONING = ON` で有効にします。[9]

```
CREATE TABLE dbo.Emp (
    EmpId     int PRIMARY KEY,
    Salary    money NOT NULL,
    ValidFrom datetime2 GENERATED ALWAYS AS ROW START NOT NULL,
    ValidTo   datetime2 GENERATED ALWAYS AS ROW END   NOT NULL,
    PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
) WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.EmpHistory,
        HISTORY_RETENTION_PERIOD = 6 MONTHS));
```

履歴の保持期間は `HISTORY_RETENTION_PERIOD` で DAYS・WEEKS・MONTHS・YEARS のいずれかの単位で指定します。省略すると既定は INFINITE（無期限）で、履歴は消えません。

過去の状態は `FOR SYSTEM_TIME` 句で読みます。サブ句は 5 つです。

- `AS OF`：その時点の状態
- `FROM ... TO`：期間と重なって有効だった行。終了の時刻ちょうどに始まる行は含まない
- `BETWEEN ... AND`：FROM ... TO と同じ範囲に、終了の時刻ちょうどに始まる行を足したもの

SAMPLE

- `CONTAINED IN`：期間の内側で開始から終了まですべて完結した行
- `ALL`：現行表と履歴表の全行

間違えやすいのは、FROM..TO と BETWEEN..AND の端の扱いです。FROM..TO は終了の境界を含まず、BETWEEN..AND は含みます。

外部テーブル

外部テーブルは、データを移さずに外のデータを表として読む仕組みで、データの仮想化と呼びます。先に `CREATE EXTERNAL DATA SOURCE` で接続先を作り、必要なら `CREATE EXTERNAL FILE FORMAT` でファイルの形を作り、最後に `CREATE EXTERNAL TABLE` で `LOCATION` と `DATA_SOURCE` を指定します。認証が要る接続先なら、外部データ ソースの `CREDENTIAL` にデータベース スコープ資格情報を指定するので、資格情報を先に作っておかないと、外部データ ソースを作れません。FILE_FORMAT と reject のオプションは任意で、DEFAULT 制約は使えません。[10]

Azure SQL Database で使えるデータ ソースの TYPE は 3 つです。BLOB_STORAGE、RDBMS、SHARD_MAP_MANAGER です。TYPE は常に要るわけではなく、指定するのは一部のデータ ソースだけです。RDBMS は、別の Azure SQL Database へ弾力的クエリ（elastic query・プレビュー）で問い合わせるときに使います。BLOB_STORAGE は BULK INSERT などの一括操作用で、これを指す外部テーブルは作れません。

台帳テーブル

台帳（ledger）テーブルは、改ざんを検出できる表で、次の 2 種類があります。

SAMPLE

種類	作り方	できる操作	履歴テーブル
更新可能な台帳テーブル	`LEDGER = ON`	INSERT・UPDATE・DELETE	要る（旧バージョンを残す）
追加のみの台帳テーブル	`LEDGER = ON (APPEND_ONLY = ON)`	INSERT だけ	要らない

更新可能な台帳テーブルには、台帳テーブルと履歴テーブルを結合した台帳ビューをシステムが自動で作り、行の変更の履歴はこのビューで読みます。データベースのダイジェストは、最新ブロックのハッシュで、その時点の台帳テーブル全体の状態を表します。改ざんできない保管先（不変ポリシー付きの Azure Blob Storage や Azure Confidential Ledger）へ出しておき、ダイジェストの検証では、保管したダイジェストと現在の表から再計算したハッシュを突き合わせます。一致しなければ検証は失敗し、改ざんは見逃されません。台帳データベースとして作ると、入れられるのは台帳テーブルだけで、表は既定で更新可能な台帳テーブルとして作られます。一度台帳テーブルにした表は、通常の表へ戻せません。攻撃者が台帳を一時的に外して改ざんし、再び有効にする手口を防ぐためです。[11]

提供範囲は、SQL Server 2022 以降、Azure SQL Database、Azure SQL Managed Instance です。Fabric の SQL データベースでは作れません。

グラフ テーブル

グラフ テーブルは、ノードとエッジで関係を持つ表です。`CREATE TABLE ... AS NODE` と `AS EDGE` で作ります。1 つのデータベースにグラフは 1 つです。Fabric の SQL データベースでも使えますが、ノード表とエッジ表は OneLake にミラーされません。暗黙の列が自動で付きます。

SAMPLE

- node テーブル：`$node_id`
- edge テーブル：`$edge_id`・`$from_id`・`$to_id`

エッジは常に有向で、2 つのノードを結びます。`CONNECTION (node_table TO node_table)` を付けると、エッジが結べるノードの組を制限できます。これがエッジ制約です。エッジ テーブルの ON DELETE は NO ACTION か CASCADE です。[12]

```
CREATE TABLE Person (ID int PRIMARY KEY, Name varchar(50)) AS NODE;
CREATE TABLE Restaurant (ID int PRIMARY KEY, Name varchar(50)) AS NODE;
CREATE TABLE likes (Rating int, CONSTRAINT ec CONNECTION (Person TO
Restaurant)) AS EDGE;
```

クエリ側の `MATCH` と `SHORTEST_PATH` は 1.3.6 で扱います。

JSON 列と JSON インデックス [1.1.3]

JSON 列

ネイティブの `json` 型は、JSON ドキュメントをバイナリ形式で持ちます。文字列は UTF-8（Latin1_General_100_BIN2_UTF8）で内部保持され、上限は、ドキュメントが 2 GB、ユニーク キーが最大 32,000、キー文字列が 7,998 バイト、入れ子が 128 レベルです。入れられるのは JSON オブジェクトか配列だけで、スカラー値・boolean・JSON の null は入りません。SQL の NULL は入ります。[13]

`modify` メソッドで列内のドキュメントを部分更新でき、可能なら in-place で更新されます。提供範囲は SQL Server 2025、Azure SQL Database、

SAMPLE

Managed Instance（SQL Server 2025 または Always-up-to-date ポリシー）、Fabric の SQL データベースで、いずれも GA です。

`json` 型が使えない環境や旧来の形では、nvarchar(max) などの文字列列に JSON を入れ、`ISJSON` の CHECK 制約で形を守ります。

```
CREATE TABLE dbo.Orders (
    Id   int PRIMARY KEY,
    Info nvarchar(max) NOT NULL CONSTRAINT ck_info CHECK (ISJSON(Info)
= 1)
);
```

ISJSON は第 2 引数に VALUE・ARRAY・OBJECT・SCALAR を取れます（SQL Server 2022 で追加）。省略時は、オブジェクトか配列なら 1 を返します。キーの一意性は検査しません。

間違えやすいのは、json 列が通常のインデックスのキー列にならない点です。json 列は INCLUDE 列か、フィルター インデックスの WHERE にだけ使えます。

JSON インデックス

`CREATE JSON INDEX` は、json 列の中のパスを索引にする専用のインデックスです。

```
CREATE JSON INDEX json_content_index
    ON docs (content)
    FOR ('$.a', '$.b') WITH (FILLFACTOR = 80);
```

SAMPLE

FOR 句を省略すると、既定のパスは `$` で、そこから下をすべて再帰的に索引にします。パスどうしは重ねられません。`$.a` と `$.a.b` は、前者が後者を含むので両方指定するとエラーです。[14]

前提と制限は次のとおりです。

- 表にクラスター化主キー（クラスタリング キー）が要る。キーは 31 列・128 バイト未満
- 1 つの json 列に JSON インデックスは 1 つだけ。1 表では最大 249 個
- インデックス付きビュー・計算列・メモリ最適化テーブルの上には作れない
- 作成・変更はオフラインだけ。`ONLINE = ON` はエラー。DATA_COMPRESSION は構文にあるが、制限事項で未対応
- `OPTIMIZE_FOR_ARRAY_SEARCH` は配列検索を最適化する選択肢で、既定は OFF

効く述語は `JSON_PATH_EXISTS`、`JSON_VALUE`（`RETURNING` 付きを含む）、`JSON_CONTAINS` です。JSON_VALUE の LIKE と IS NULL には、まだ使えません。

計算列にパスを取り出して通常のインデックスを張る方法とは、使い分けがあります。JSON インデックスは、パスが多い、決まっていない、配列の中を探す、といった場面に向きます。

制約：PRIMARY KEY・FOREIGN KEY・UNIQUE・CHECK・DEFAULT [1.1.4]

制約の種類と付いてくるインデックス

制約は 5 種類です。PRIMARY KEY は 1 表に 1 つで、構成列はすべて NOT NULL になります（NULL 可を指定しないと NOT NULL で作られます）。32

SAMPLE

列・合計キー長 900 バイトを超えられません。既定でユニーク クラスター化インデックスが自動で作られます。非クラスター化として指定することもできます。[15]

UNIQUE は重複を禁じますが、NULL は許します。1 列あたり NULL は 1 つだけです。PRIMARY KEY は NULL を許しません。UNIQUE は既定で一意の非クラスター化インデックスを伴います。

CHECK は、FALSE になる値を拒否します。NULL は UNKNOWN に評価されるので、制約には拒否されません。「NULL を拒否したい」なら NOT NULL を併用します。DEFAULT は、値を省略した行に入れる既定値で、1 列に 1 つしか持てません。既存の表には `ALTER TABLE ... ADD CONSTRAINT ... DEFAULT ... FOR 列` で足せます。列の追加と同時に `WITH VALUES` を付ければ、既存行にも既定値が入ります。

FOREIGN KEY は、別表の主キーか一意キーを参照します。1 表が参照できる表・列は、最大 253 を超えられません。被参照側の上限は、SQL Server 2016 以降の互換性レベル 130 以上で、最大 10,000 になります。

外部キーの参照動作と制約の有効・無効

親の行を更新・削除したとき、子の行をどうするかは、`ON DELETE` と `ON UPDATE` で決まります。

指定	親の変更時の子
NO ACTION（既定）	エラーにして変更を取り消す
CASCADE	子も更新・削除する
SET NULL	子の外部キー列を NULL にする（列が NULL 可であること）

SAMPLE

指定	親の変更時の子
SET DEFAULT	子の外部キー列を既定値にする（全外部キー列に DEFAULT 定義が要る）

CASCADE が使えない条件は、トリガーとの組み合わせです。INSTEAD OF DELETE トリガーを持つ表には ON DELETE CASCADE を指定できません。UPDATE も、INSTEAD OF UPDATE トリガーがあれば同様に不可です。[16]

外部キーや CHECK を後から足すと、既存データを検証します。`WITH NOCHECK` を付けると、この検証を省略できます。省略して足した制約は、以後の変更だけを検査し、クエリ オプティマイザーからは考慮されません。一時的に止めたいときは `ALTER TABLE ... NOCHECK CONSTRAINT` で無効にし、`CHECK CONSTRAINT` で戻します。戻すときに `WITH CHECK` を付けないと、既存行は検証されません。

SEQUENCE [1.1.5]

SEQUENCE の作成と NEXT VALUE FOR

SEQUENCE は、特定の表に結び付かない番号の発生源です。

```
CREATE SEQUENCE dbo.OrderSeq AS int
    START WITH 1000 INCREMENT BY 1
    MINVALUE 1000 MAXVALUE 999999
    NO CYCLE CACHE 50;

SELECT NEXT VALUE FOR dbo.OrderSeq;
```

既定値を表にします。[17]

SAMPLE

句	既定
データ型	bigint（整数系と、scale 0 の decimal・numeric が使える）
START WITH	昇順なら最小値、降順なら最大値
INCREMENT BY	1（0 は不可。負なら降順）
CYCLE	NO CYCLE
CACHE	CACHE（サイズはエンジンが決める）

CYCLE が有効で上限に達すると、最小値（降順では最大値）から再開します。START WITH の値へは戻りません。ここは取り違えやすい所です。CACHE は、現在値と残りの数だけをメモリに持ちます。停電などの異常停止で、キャッシュ内の未使用の番号は失われ、欠番が出ます。シーケンスの番号は現在のトランザクションの外で生成されるので、ロールバックしても戻りません。

`NEXT VALUE FOR` は DEFAULT 制約の中でも使えます。`OVER (ORDER BY ...)` を付けると、複数行に順序を指定して番号を振れます。

SEQUENCE と IDENTITY の違い

IDENTITY は、列のプロパティで、その表に行を挿入したときに値が生成されます。`IDENTITY(seed, increment)` の seed と increment は、両方指定するか両方省略するかのどちらかで、省略時は (1,1) です。[18]

SEQUENCE を選ぶ場面は 3 つです。

- 挿入より前に番号が要る
- 複数の表や列で同じ番号の流れを共有する
- 範囲を一度にまとめて確保する

SAMPLE

最後の場面には `sys.sp_sequence_get_range` を使います。引数は `@sequence_name` と `@range_size` で、確保した範囲の先頭値などが OUTPUT で返ります。IDENTITY も SEQUENCE も欠番は起こり得るので、「連番の保証」を求める要件には、どちらも向きません。

パーティション分割 [1.1.6]

パーティション関数とスキーム

大きな表を、列の値の範囲で物理的に分ける仕組みです。作る順序は決まっています。

1. `CREATE PARTITION FUNCTION`：境界値で範囲を決める
2. `CREATE PARTITION SCHEME`：関数の各範囲をファイル グループへ対応付ける
3. `CREATE TABLE`（または `CREATE INDEX`）の `ON scheme(列)`：表に適用する

```
CREATE PARTITION FUNCTION pf_year (date)
AS RANGE RIGHT FOR VALUES ('2025-01-01', '2026-01-01');

CREATE PARTITION SCHEME ps_year
AS PARTITION pf_year ALL TO ([PRIMARY]);

CREATE TABLE dbo.Sales (SaleDate date NOT NULL, Amount money)
ON ps_year (SaleDate);
```

境界値が n 個なら、できるパーティションは n + 1 個です。境界値は最大 14,999 個、パーティションは最大 15,000 です。パーティション列は 1 つしか指定できません。RANGE は LEFT か RIGHT で、省略すると LEFT です。境界値

SAMPLE

がどちらの区間に属するかを決めます。LEFT なら境界値は左の区間の最大値になり、RIGHT なら右の区間の最小値になります。上の例は RIGHT なので、`'2025-01-01'` の行は 2 つ目の区間に入ります。[19]

スキームでは、ファイル グループが関数の作るパーティション数より多いとき、最初の未割り当てが NEXT USED になります。`ALL TO` を付けると、全パーティションが同じファイル グループに載ります。スキームの作成には ALTER ANY DATASPACE 権限が要ります。

Azure SQL Database と Fabric の SQL データベースには PRIMARY ファイル グループしかありません。全パーティションは、PRIMARY に置くことになります。

整列したインデックスとパーティションの切り替え

表と同じパーティション スキームの上に作ったインデックスを、整列したインデックス（aligned index）と呼びます。表と非クラスター化インデックスが整列していると、パーティションを構造ごと高速に切り替えられます。`ALTER TABLE ... SWITCH PARTITION` は、行を動かさずメタデータだけを付け替えるので、大量の行を出し入れしても一瞬で済み、行のコピーは起きません。典型は、古い月を履歴表へ切り出す「switch out」と、ロード済みの表を取り込む「switch in」の 2 つです。[20]

境界の増減には `SPLIT RANGE` と `MERGE RANGE` を使います。パーティション単位の削除には `TRUNCATE TABLE ... WITH (PARTITIONS (n))` が使えます。

Fabric の SQL データベースでは、パーティションの切り替えと、パーティション単位の圧縮の DDL は許可されません。[21]

SAMPLE

ビュー [1.2.1]

ビューの作成

ビューは、保存した SELECT に名前を付けた仮想の表です。選択リストは最大 1,024 列です。ビュー定義の ORDER BY は、TOP か OFFSET と併用するときだけ書けます。併用しても、ビューから読む結果の並びは保証されません。作成には、データベースの CREATE VIEW 権限と、作成先スキーマの ALTER 権限が要ります。[22]

オプションは 3 つです。

- `SCHEMABINDING`：定義に影響する形で、基の表を変更できなくする。基の表の列を落とす ALTER は失敗する
- `WITH CHECK OPTION`：ビュー経由の INSERT・UPDATE が、ビューの WHERE 条件に合う行だけを作るよう強制する。変更後の行は、必ずビューで見える
- `VIEW_METADATA`：ブラウズ モードのメタデータが、基の表ではなくビューの名前を返す

```
CREATE VIEW dbo.vJpCustomer WITH SCHEMABINDING AS
SELECT CustomerId, Name, Country FROM dbo.Customer WHERE Country =
N'JP'
WITH CHECK OPTION;
```

CHECK OPTION が効くのは、ビュー経由の更新だけです。基の表を直接更新する文は止められません。また、WITH CHECK OPTION を使う更新可能なビューには、INSTEAD OF トリガーを定義できません。ビュー経由で更新できるのは、変更する列が 1 つの基の表の列だけを参照し、集計関数・計算・UNION

SAMPLE

などの集合演算で作った列でなく元の列を直接参照し、GROUP BY・HAVING・DISTINCT の影響を受けない場合です。

インデックス付きビュー

インデックス付きビューは、結果を物理的に保存したビューです。最初に作るインデックスは、ユニーク クラスター化インデックスでなければなりません。その後に非クラスター化を足せます。SCHEMABINDING も必須です。GROUP BY を含む定義は `COUNT_BIG(*)` を含み、HAVING を含みません。[23]

集計を何度も同じ形で引くレポートに向きます。更新の多い表の上では、更新のたびにビューも維持されるため、書き込みの負荷は減りません。

SQL Server Standard では、`NOEXPAND` ヒントでビューを直接参照します。Azure SQL Database と Managed Instance は、ヒントなしでも自動で使います。

スカラー関数 [1.2.2]

スカラー関数と決定性・インライン化

スカラー関数は 1 つの値を返し、`RETURNS` で型を宣言します。関数のオプションは次のとおりです。[24]

オプション	働き
ENCRYPTION	定義を隠す
SCHEMABINDING	参照先の変更を止める
RETURNS NULL ON NULL INPUT / CALLED ON NULL INPUT	NULL 入力時に本体を実行するか
EXECUTE AS	実行コンテキスト

SAMPLE

オプション	働き
INLINE	インライン化の許可（ON・OFF）

関数にはできないことがあります。データベースの状態を変える操作は不可です。TRY...CATCH、@@ERROR、RAISERROR も使えません。非決定的な NEWID・NEWSEQUENTIALID・RAND・TEXTPTR は関数内で使えません（Msg 443）。ビューで包めば回避できます。ネストは最大 32 レベルです。

スカラー UDF インライン化は、関数の本体を呼び出し元のクエリに展開して、行ごとの関数呼び出しの重さをなくす機能です。データベース互換性レベルを 150 にするだけで対象になり、クエリや関数の書き換えは要りません。SQL Server 2019 以降、Azure SQL Database、Managed Instance が対象です。

操作	書き方
1 つの関数だけ止める	`CREATE/ALTER FUNCTION ... WITH INLINE = OFF`
1 つのクエリだけ止める	`OPTION (USE HINT('DISABLE_TSQL_SCALAR_UDF_INLINING'))`
構造上できるか調べる	`sys.sql_modules` の `is_inlineable` 列

`is_inlineable` は、関数が構造上インライン化できるかを示します。コンパイル時に実際にインライン化されるかまでは保証しません。データベース全体で止めるときは `ALTER DATABASE SCOPED CONFIGURATION SET TSQL_SCALAR_UDF_INLINING = OFF` を使います。

インライン化を阻む要素の例を挙げます。

SAMPLE

- `GETDATE()` のような時間依存の関数や、`NEWSEQUENTIALID()` のように副作用のある関数を呼ぶ
- テーブル変数やテーブル値パラメーターを参照する
- `EXECUTE AS CALLER` 以外の実行コンテキストを使う
- CTE、`STRING_AGG`、XML のメソッド、`@@ROWCOUNT` を参照する
- 呼び出し側が、関数を ORDER BY・GROUP BY の中、計算列、CHECK 制約の中で使う

`SELECT @val += col FROM t` のような変数の累積代入も、インライン化できません。ネイティブ コンパイルのスカラー UDF（インライン化の対象外）では、`SCHEMABINDING` と `EXECUTE AS` が必須で、本体は `BEGIN ATOMIC WITH` を使います。

Fabric Data Warehouse のスカラー UDF はプレビューです。Fabric の SQL データベースの話ではありません。

テーブル値関数 [1.2.3]

インライン TVF と複数ステートメント TVF

テーブル値関数は、表を返す関数です。2 種類あります。[25]

項目	インライン TVF	複数ステートメント TVF（MSTVF）
本体	単一の SELECT。`RETURNS TABLE`、`RETURN (SELECT ...)`	`@変数 TABLE (...)` を宣言し、BEGIN…END で行を詰める
戻り変数	なし	`@return_variable`（TABLE 型）

SAMPLE

項目	インライン TVF	複数ステートメント TVF（MSTVF）
性能	呼び出し側のクエリに展開され、最適化されやすい	展開されず、行数の見積もりが外れやすい
BEGIN…END	不要	必要

```
CREATE FUNCTION dbo.fnOrders (@cid int)
RETURNS TABLE AS
RETURN (SELECT OrderId, Total FROM dbo.Orders WHERE CustomerId = @cid);

SELECT c.Name, o.OrderId
FROM dbo.Customer AS c
CROSS APPLY dbo.fnOrders(c.CustomerId) AS o;
```

TVF は、SELECT・INSERT・UPDATE・DELETE の FROM 句で、テーブル式が書ける場所なら呼べます。外側の行ごとに引数を渡すときは `CROSS APPLY` か `OUTER APPLY` を使います。パラメーターは `@name [AS] type [NULL] [= default] [READONLY]` の形で、最大 2,100 個です。既定値があれば、省略して実行できます。

関数は複数の結果セットを返せません。必要ならストアド プロシージャを使います。関数内では動的 SQL と一時テーブルが使えず、テーブル変数は使えます。FOR XML 句も使えません。作成には、データベースの CREATE FUNCTION 権限とスキーマの ALTER 権限が要ります。

選ぶ基準は単純で、1 つの SELECT で書けるならインライン TVF、途中の手順が要るときだけ MSTVF です。

SAMPLE

ストアド プロシージャ [1.2.4]

パラメーターと実行コンテキスト

`CREATE [ OR ALTER ] PROCEDURE` で作ります。OR ALTER が使えるのは、SQL Server 2016 SP1 以降、Azure SQL Database、Fabric の SQL データベースです。CREATE PROCEDURE は、バッチの最初の文でなければなりません。パラメーターは最大 2,100 個です。[26]

```
CREATE OR ALTER PROCEDURE dbo.usp_GetTotal
    @cid   int,
    @since date = NULL,          -- 既定値があれば省略できる
    @total money OUTPUT
WITH EXECUTE AS OWNER
AS
BEGIN
    SELECT @total = SUM(Total) FROM dbo.Orders
    WHERE CustomerId = @cid AND (@since IS NULL OR OrderDate >=
@since);
    RETURN 0;                     -- 戻り値は整数の状態コード
END;
```

`OUTPUT`（または `OUT`）は、値を呼び出し元へ返すパラメーターです。呼び出し側も `EXEC ... @total OUTPUT` と書きます。`READONLY` は、本体でパラメーターを変更できなくする指定です。テーブル値パラメーターでは READONLY が必須で、入力専用です。`RETURN` が返すのは整数の状態コードで、表や文字列は返せません。

`EXECUTE AS` の指定は 4 つあります。

SAMPLE

- `CALLER`：呼び出し元のコンテキストで実行する
- `SELF`：作成または変更した人のユーザーで実行する
- `OWNER`：モジュールの現在の所有者のコンテキストで実行する
- `'user_name'`：指定したユーザーで実行する

呼び出し元に表の権限を渡さず、プロシージャの EXECUTE だけ渡す設計に使います。ネストは最大 32 レベルで、超えると呼び出しチェーン全体が失敗します。

作成には、データベースの CREATE PROCEDURE 権限とスキーマの ALTER 権限が要ります。db_ddladmin のメンバーなら、権限を足さなくても作れます。[27]

再コンパイルとパラメーター スニッフィング

パラメーター スニッフィングは、プロシージャが最初にコンパイルされるときの引数の値で、プランが決まり、キャッシュされて以後も使われる動作です。値の偏りが大きい列では、最初の値に合ったプランが別の値で遅くなることがあります。[28]

対処は 3 つあります。

手段	書き方	効果
プロシージャ全体を毎回コンパイル	`CREATE PROCEDURE ... WITH RECOMPILE`	プランをキャッシュしない
文だけを毎回コンパイル	文の末尾に `OPTION (RECOMPILE)`	その文だけ、実行時の値で最適化

SAMPLE

手段	書き方	効果
特定の値や平均で最適化	`OPTION (OPTIMIZE FOR (@p = 値))` または `OPTIMIZE FOR UNKNOWN`	固定した前提でプランを作る

`OPTIMIZE FOR UNKNOWN` は、実行時の引数の値を使わず、述語の全列値にわたる平均的な選択性でプランを作ります。

`WITH RECOMPILE` は実行のたびにコンパイルするので、CPU を使います。全体を毎回コンパイルする必要はありません。呼び出し頻度の高いプロシージャでは、問題の文だけに `OPTION (RECOMPILE)` を付ける方が軽く済みます。

トリガー [1.2.5]

DML トリガー

DML トリガーは、表やビューの INSERT・UPDATE・DELETE（MERGE を含む）で自動実行される処理です。`FOR`（AFTER と同じ）・`AFTER`・`INSTEAD OF` のいずれかと、対象の操作を指定します。AFTER は操作の実行後に動き、INSTEAD OF は操作の代わりに動きます。INSTEAD OF は、1 表（またはビュー）につき、操作ごとに最大 1 つしか作れません。WITH CHECK OPTION を使う更新可能なビューには、INSTEAD OF トリガーを定義できません。[29]

トリガーの中では 2 つの論理表を使います。

論理表	中身
inserted	INSERT・UPDATE 後の新しい行（変更後）
deleted	DELETE・UPDATE で変更される前の行（変更前）

SAMPLE

UPDATE は「削除してから挿入する」動きに似ていて、変更前が deleted、変更後が inserted に入ります。どちらも直接は変更できません。1 回の文で複数行が変わるので、トリガーは複数行を前提に書きます。`SELECT @x = col FROM inserted` のように 1 行だけを取る書き方は、複数行で誤動作します。

トリガー内の `ROLLBACK TRANSACTION` は、ネストの深さにかかわらず、以降のトリガーを実行させません。作成には、表やビューの ALTER 権限が要ります。

DDL トリガーとログオン トリガー

DDL トリガーは CREATE・ALTER・DROP などに反応します。範囲は `ON DATABASE`（現在のデータベース）か `ON ALL SERVER`（サーバー全体）です。ログオン トリガーに `ON DATABASE` はありません。`ON ALL SERVER` で作り、ログイン時に動きます。サーバー スコープの DDL トリガーとログオン トリガーの作成には、CONTROL SERVER 権限が要ります。[30]

再帰は 2 つの設定で決まります。直接の再帰は、`ALTER DATABASE` の `RECURSIVE_TRIGGERS` 設定で有効になり、これを無効にしても止まるのは直接の再帰だけです。間接の再帰まで止めるには、サーバー オプション `nested triggers` を 0 にします。`nested triggers` を 0 にすると、`RECURSIVE_TRIGGERS` の設定にかかわらず、再帰のトリガーも動きません。

共通テーブル式（CTE） [1.3.1]

共通テーブル式と再帰 CTE

CTE は、1 つの文の実行範囲の中で定義する、名前付きの一時的な結果セットです。`WITH 名前 AS (SELECT ...)` に続けて、SELECT・INSERT・UPDATE・MERGE・DELETE を書きます。CTE の結果は実体化されません。外側から参照するたびに、定義のクエリが再実行されます。何度も参照するなら、一時テ

SAMPLE

ーブルなどを検討してください。複数の CTE はカンマで続けて定義します。バッチ内の文なので、直前の文はセミコロンで終えます。[31]

定義の中では、INTO、クエリ ヒント付きの OPTION 句、FOR BROWSE が使えません。ORDER BY は TOP か OFFSET/FETCH があるときだけ使えます。

再帰 CTE は、自分自身を参照します。構成は 2 部です。

```
WITH Org AS (
    SELECT EmpId, MgrId, 0 AS Lvl FROM dbo.Emp WHERE MgrId IS NULL   --
アンカー
    UNION ALL
    SELECT e.EmpId, e.MgrId, o.Lvl + 1
-- 再帰メンバー
    FROM dbo.Emp AS e JOIN Org AS o ON e.MgrId = o.EmpId
)
SELECT * FROM Org OPTION (MAXRECURSION 10);
```

アンカー メンバーが初期の結果を作ります。再帰メンバーは前回の出力を入力に繰り返し、行が返らなくなるまで止まりません。最後に UNION ALL で全結果を結合します。アンカーはすべて、最初の再帰メンバーより前に置きます。アンカーと再帰メンバーの間、および再帰メンバー同士をつなぐ演算子は、UNION ALL しか使えません。

`MAXRECURSION` は、無限ループを防ぐために再帰のレベルを制限します。値は 0〜32767 で、文ごとに 1 つ指定します。既定は 100 で、0 を指定すると制限がなくなります。再帰メンバーの中では、DISTINCT、GROUP BY、PIVOT、HAVING、スカラー集計、TOP、サブクエリ、LEFT/RIGHT/OUTER JOIN が使えません。INNER JOIN だけは使えます。再帰 CTE が返す列は、元の列が NOT

SAMPLE

NULL でも、NOT NULL のままにはなりません。常に NULL 許容です。再帰部分の ROW_NUMBER などは、CTE 全体ではなく、現在の再帰レベルの行集合にしか働きません。

ウィンドウ関数 [1.3.2]

順位付けの関数と OVER

OVER 句は、ウィンドウ関数を適用する前に、行集合の分割と並べ替えを決めます。`PARTITION BY` で区切り、`ORDER BY` で並べます。移動平均・累計・グループ内の上位 N の計算に使います。[32]

関数	同順位の扱い	例（得点が 90・80・80・70・60 の降順）
ROW_NUMBER	同順位でも別の連番	1,2,3,4,5
RANK	同じ値を付け、後に欠番が出る	1,2,2,4,5
DENSE_RANK	同じ値を付け、欠番なし	1,2,2,3,4
NTILE(n)	行を n 個のグループに配り、所属番号（1 始まり）を返す	n = 2 なら 1,1,1,2,2

NTILE で行数がグループ数で割り切れないとき、グループの大きさは 1 だけ違う 2 種類になり、大きいグループが先に来ます。5 行を 2 つに分けるなら、3 行と 2 行です。ランキング関数は ORDER BY が必須で、OVER 句の ROWS/RANGE は指定できません。

集計・前後の値・フレーム・WINDOW 句

`LAG` は現在行より前、`LEAD` は後の行を、自己結合なしで取ります。offset の既定は 1 で、default を省略すると、範囲の外では NULL が返ります。`IGNORE`

SAMPLE

`NULLS` / `RESPECT NULLS` は SQL Server 2022 以降で使え、指定しないと RESPECT NULLS です。`FIRST_VALUE` と `LAST_VALUE` は、フレームの先頭と末尾の値を返します。[33]

フレームは、集計の対象にする行の範囲です。`ROWS` は現在行の前後の固定の行数（物理）で、`RANGE` は現在行の値に対する値の範囲（論理）で限定します。ORDER BY を指定して ROWS/RANGE を省略すると、既定のフレームは `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` です。この既定のせいで、LAST_VALUE は、パーティションの末尾の値を返してくれません。パーティション末尾まで見たいときは、`ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` を明示します。

```
SELECT OrderDate, Amount,
       SUM(Amount) OVER (ORDER BY OrderDate
                         ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
Last3,
       LAG(Amount, 1, 0) OVER (ORDER BY OrderDate) AS Prev
FROM dbo.Sales;
```

`WINDOW` 句は、名前付きのウィンドウ定義を 1 回書いて複数の関数で使い回す書き方です。SQL Server 2022 より前では使えません。互換性レベルも、160 以上が必要です。

JSON 関数 [1.3.3]

JSON を組み立てる関数

`JSON_OBJECT` はキーと値の組から JSON オブジェクトを、`JSON_ARRAY` は値の並びから配列を作ります。NULL の扱いを `NULL ON NULL`（SQL の NULL を

SAMPLE

JSON の null にする）か `ABSENT ON NULL`（その要素を省く）で選びます。既定は関数で逆です。[34]

関数	既定の NULL 句
JSON_OBJECT	NULL ON NULL
JSON_ARRAY	ABSENT ON NULL
JSON_OBJECTAGG	NULL ON NULL
JSON_ARRAYAGG	ABSENT ON NULL

```
SELECT JSON_OBJECT('name': 'Ann', 'age': NULL);              --
{"name":"Ann","age":null}
SELECT JSON_OBJECT('name': 'Ann', 'age': NULL ABSENT ON NULL); --
{"name":"Ann"}
SELECT JSON_ARRAY(1, NULL, 3);                                -- [1,3]
```

`JSON_ARRAYAGG` と `JSON_OBJECTAGG` は集計関数です。複数行を 1 つの配列、またはオブジェクトにまとめます。どちらも SQL Server 2025 で追加されました。行を JSON にする従来の方法は `FOR JSON`（`PATH` か `AUTO`）で、SELECT の結果全体を JSON 文字列にします。

間違えやすいのは、既定の取り違えです。JSON_OBJECT は null を残し、JSON_ARRAY は NULL 要素を落とします。

SAMPLE

JSON を読む関数

関数	取り出すもの	補足
JSON_VALUE	スカラー値	`RETURNING` なしは nvarchar(4000)
JSON_QUERY	オブジェクトまたは配列	スカラーは取れない
OPENJSON	行と列の表	互換性レベル 130 以上
JSON_CONTAINS	値が含まれるか（1/0/NULL）	SQL Server 2025 以降
JSON_MODIFY	値を更新した JSON を返す	1 回で 1 プロパティ

JSON_VALUE は 4,000 文字を超える値を、lax モードでは NULL にし、strict モードではエラーにします。長い値は JSON_VALUE では取れません。OPENJSON を使います。`RETURNING data_type` は SQL Server 2025 以降で、json 型の入力のときだけ使えます。[35]

OPENJSON は、WITH 句なしだと key・value・type の 3 列を返します。WITH 句で列名・型・パスを明示でき、キーの照合は大文字小文字を区別します。内側のオブジェクトや配列を JSON のまま受けるには、列に `AS JSON` を付け、型は nvarchar(MAX) 以外にできません。

```
SELECT o.OrderId, i.Sku, i.Qty
FROM dbo.Orders AS o
CROSS APPLY OPENJSON(o.Info, '$.items')
     WITH (Sku nvarchar(20) '$.sku', Qty int '$.qty') AS i;
```

`JSON_CONTAINS(target, search_value [, path [, search_mode]])` は、検索値が含まれるかを返します。search_mode の既定 0 は等価比較で、1 は LIKE の

SAMPLE

意味論です。search_mode は、文字列以外の検索値には効きません。path の既定は `$`（文書全体）です。引数が NULL か、パスが見つからないときは NULL を返します。Fabric Data Warehouse では使えません。

`JSON_MODIFY` のパスは、モード省略時は lax です。strict ではプロパティがなければエラーになります。`append` は配列の末尾へ追加します。1 回の呼び出しで、複数のプロパティは更新できません。複数のプロパティを変えるときは、JSON_MODIFY を入れ子にし、内側の結果を外側の第 1 引数に渡します。

正規表現 [1.3.4]

正規表現の関数の働き

T-SQL の正規表現は RE2 ライブラリに基づきます。スカラー関数は 5 つです。[36]

関数	返すもの
REGEXP_LIKE	一致するか（述語として WHERE で使う）
REGEXP_REPLACE	一致部分を置換した文字列
REGEXP_SUBSTR	一致した部分文字列
REGEXP_INSTR	一致の位置
REGEXP_COUNT	一致の回数

REGEXP_REPLACE の `occurrence` の既定 0 は、全一致を置換します。`string_replacement` を省略すると空文字列で、一致部分が消えます。置換文字列の中では `\1` 〜 `\9` でキャプチャ グループを、`&` で一致全体を参照できます。REGEXP_INSTR の `return_option` は、0 が開始位置、1 が終了位置で、

SAMPLE

既定は 0 です。開始位置 start が文字列長を超えても、エラーにはなりません。戻り値は 0 です。

```
SELECT Phone FROM dbo.Customer
WHERE REGEXP_LIKE(Phone, '^\d{3}-\d{4}-\d{4}$');

SELECT REGEXP_REPLACE('2026-11-17', '(\d+)-(\d+)-(\d+)', '\3/\2/\1');
-- 17/11/2026
```

フラグ（flags）は `i`（大文字小文字を無視）、`m`（複数行モード）、`s`（`.` が `\n` にも一致）、`c`（大文字小文字を区別、既定）です。ほかの文字はエラーで、矛盾する指定は最後の文字が有効です。上限は 3 つあります。

- パターン：8,000 バイト
- flags：30 文字
- string_expression の LOB（varchar(max)/nvarchar(max)）：2 MB

表を返す正規表現の関数と前提

表を返す関数は 2 つです。

- `REGEXP_MATCHES`：match_id（bigint）・start_position・end_position・match_value・substring_matches（json）の列を返す。一致がなければ行を返さない。
- `REGEXP_SPLIT_TO_TABLE`：value（分割後の文字列）と ordinal（bigint、1 始まり）の 2 列を返す。パターンに一致しなければ、元の文字列を返す。

`REGEXP_LIKE` は、データベースの互換性レベル 170 以上が必要です。ほかの 4 つのスカラー関数は、全互換性レベルで使えます。2 つのテーブル値関数は、

SAMPLE

互換性レベル 170 が必要です。ただし `ALLOW_BUILTIN_TVF_IN_ALL_COMPAT_LEVELS` データベース スコープ構成を有効にすれば、170 未満でも使えるようになります。[37]

提供範囲は、SQL Server 2025、Azure SQL Database、Fabric の SQL データベースです。SQL Server 2022 以前では使えません。Managed Instance では、SQL Server 2025 または Always-up-to-date ポリシーのときだけ使えます。Fabric Data Warehouse では、スカラー関数がプレビューで、テーブル値関数は未対応です。

あいまい一致の関数 [1.3.5]

EDIT_DISTANCE・JARO_WINKLER の選び方と値の向き

文字列の「似ている度合い」を数える関数が 4 つあります。いずれもプレビュー機能で、文字式を取ります。varchar(max) と nvarchar(max) は使えません。EDIT_DISTANCE と EDIT_DISTANCE_SIMILARITY は、入力のどれかが NULL なら NULL を返します。[38]

関数	戻り値	範囲と向き
EDIT_DISTANCE	int	挿入・削除・置換・転置の最小回数
EDIT_DISTANCE_SIMILARITY	int	0（類似なし）〜100（完全一致）。大きいほど似ている
JARO_WINKLER_DISTANCE	float	先頭の一致を優遇した距離
JARO_WINKLER_SIMILARITY	int	0〜100。大きいほど似ている

SAMPLE

EDIT_DISTANCE は Damerau-Levenshtein（Optimal String Alignment）で、隣り合う文字の入れ替え（転置）も 1 回と数えます。任意の第 3 引数 `maximum_distance` は、実際の距離がそれを超えるとき `maximum_distance + 1` を返します。省略か負なら実際の距離を返し、NULL なら NULL を返します。上限を付けなければ、遠い組の計算を早く打ち切れません。

EDIT_DISTANCE_SIMILARITY の式は `(1 - (edit_distance / greatest(len(string1), len(string2)))) * 100` です。`Colour` と `Color` の結果は、公式の例で次のとおりです。

関数	結果
EDIT_DISTANCE	1
EDIT_DISTANCE_SIMILARITY	83
JARO_WINKLER_DISTANCE	0.0333…（約 0.033）
JARO_WINKLER_SIMILARITY	97

距離と類似度を取り違えてはいけません。距離は 0 が完全一致、類似度は 100 が完全一致です。重複する顧客名の検出なら、`WHERE EDIT_DISTANCE_SIMILARITY(a.Name, b.Name) >= 80` のように、類似度でしきい値を置きます。人名の綴り揺れのように、語頭が合っていることが多い対象は、Jaro-Winkler が向きます。提供範囲は、次の 5 つです。

- SQL Server 2025
- Azure SQL Database
- Managed Instance（SQL Server 2025 または Always-up-to-date ポリシー）
- Fabric の SQL データベース

SAMPLE

- Fabric の SQL analytics endpoint と Warehouse

グラフ クエリと MATCH [1.3.6]

MATCH と SHORTEST_PATH

`MATCH` は、グラフのノード表とエッジ表にだけ使え、SELECT の WHERE 句の一部として書きます。ASCII アートのような構文で、ノードの別名と、エッジの別名を矢印でつないで経路を表します。[39]

```
SELECT p.Name, r.Name
FROM Person AS p, likes AS l, Restaurant AS r
WHERE MATCH(p-(l)->r);        -- Person から Restaurant への有向の
likes
```

矢印の向きは 2 通りの書き方があります。`-( edge )->` と `<-( edge )-` です。エッジは方向が必須で、`MATCH(p-(l)-r)` のように向きを省くことはできません。ノード名は繰り返し使えますが、エッジ名は繰り返せません。MATCH と組み合わせられる論理演算は AND だけで、OR と NOT は使えません。

`SHORTEST_PATH` は、SQL Server 2019 以降で、MATCH の中でのみ使えます。任意の 2 ノード間の最短経路を 1 本だけ返し、全経路は返しません。同じ長さの最短経路が複数あるときは、探索で最初に見つかった 1 本です。任意長のパターンには量指定子を使います。`+` は 1 回以上の繰り返し、`{1,n}` は 1〜n 回の繰り返しで、n は正の整数に限り、どちらも最短経路が見つかった時点で探索を終えます。

SAMPLE

```
SELECT p1.Name,
       STRING_AGG(p2.Name, '->') WITHIN GROUP (GRAPH PATH) AS Path,
       LAST_VALUE(p2.Name) WITHIN GROUP (GRAPH PATH) AS Last
FROM Person AS p1, friendOf FOR PATH AS f, Person FOR PATH AS p2
WHERE MATCH(SHORTEST_PATH(p1(-(f)->p2)+))
  AND p1.Name = 'Alice';
```

任意長パターンに参加するノード表とエッジ表には、FROM 句で `FOR PATH` を付けます。そのテーブルの属性は SELECT で直接射影できません。`STRING_AGG`・`LAST_VALUE`・`SUM`・`AVG`・`MIN`・`MAX`・`COUNT` を `WITHIN GROUP (GRAPH PATH)` 付きで使って取ります。`LAST_NODE()` は、2 つの任意長パターンをつなぐ（前のパターンの最後のノードから次を始める）ために使います。

ノード表の暗黙列 `$node_id` を選択すると、列名は `$node_id_<一意の接尾辞>` で表示されます。クエリで `$node_id_<接尾辞>` の長い名前を書く必要はありません。疑似列名 `$node_id` で参照するのが、唯一の推奨方法です。エッジ表へ INSERT するときは、`$node_id` で両端のノードを指定します。

相関クエリ [1.3.7]

相関サブクエリと APPLY

相関サブクエリ（繰り返しサブクエリ）は、外側のクエリの値に依存し、外側で選ばれうる行ごとに繰り返し実行されます。サブクエリは次の 3 種類に分けられます。

- IN・ANY・ALL で使うリスト型
- 修飾のない比較演算子で単一値を返す型

SAMPLE

- EXISTS による存在テスト型 EXISTS は存在テストなので、データそのものは返しません。選択リストは慣例として `*` で、結果は TRUE/FALSE です。入れ子にできるのは、WHERE・HAVING 句の中や別のサブクエリの中に、最大 32 レベルまでです。[40]

```
SELECT c.CustomerId
FROM dbo.Customer AS c
WHERE EXISTS (SELECT 1 FROM dbo.Orders AS o WHERE o.CustomerId =
c.CustomerId);
```

サブクエリには制限があります。ORDER BY は TOP と併用するときだけです。COMPUTE 句と INTO 句は使えません。単一値を返す比較演算子のサブクエリは、GROUP BY と HAVING を含められません。

`APPLY` は、FROM 句で、左の表の各行に対して右の表式を評価します。右側に、左の列を引数にとるテーブル値関数を置けます。`CROSS` か `OUTER` の指定を省くことはできません。

演算子	右側が空の左の行
CROSS APPLY	結果から除く（INNER JOIN に近い）
OUTER APPLY	1 行残し、右の列は NULL（LEFT JOIN に近い）

APPLY は、各左行の右側の結果を、左側との UNION ALL で結合して返します。参照の向きは一方向です。右側は左の列を引数にできますが、左側のテーブル値関数に右側の列を渡すことはできません。「各顧客の直近 3 件の注文」のような、行ごとに上位 N を取る問いが典型で、`CROSS APPLY (SELECT TOP (3) ... WHERE o.CustomerId = c.CustomerId ORDER BY ...)` と書きます。

SAMPLE

エラー処理 [1.3.8]

TRY...CATCH・THROW・RAISERROR

`TRY...CATCH` は、重大度が 10 より大きく、接続を閉じない実行時エラーをすべて捕捉します。TRY ブロックの直後に CATCH ブロックが必要で、複数のバッチや BEGIN...END をまたげず、IF...ELSE にも分割できません。ユーザー定義関数の中では使えません。[41]

捕捉されないものがあります。重大度 10 以下の警告、セッションを止める重大度 20 以上のエラー、アテンション（接続切断など）、KILL によるセッション終了です。

CATCH の中では、ERROR_ 系の関数（`ERROR_NUMBER`・`ERROR_SEVERITY`・`ERROR_STATE`・`ERROR_PROCEDURE`・`ERROR_LINE`・`ERROR_MESSAGE`）が使えます。CATCH の外では、どの関数も NULL しか返しません。

```
BEGIN TRY
    BEGIN TRAN;
    UPDATE dbo.Acct SET Bal = Bal - 100 WHERE Id = 1;
    UPDATE dbo.Acct SET Bal = Bal + 100 WHERE Id = 2;
    COMMIT;
END TRY
BEGIN CATCH
    IF XACT_STATE() <> 0 ROLLBACK;
    THROW;            -- 元のエラーを再スロー
END CATCH;
```

SAMPLE

項目	THROW	RAISERROR
エラー番号	50,000 以上 2,147,483,647 以下の int を指定（事前定義は不要）	msg_id は sys.messages に定義が必要。msg_str を使うと 50000
メッセージ	nvarchar(2048)、printf の書式は不可	msg_str は最大 2,047 文字、printf 書式を使える
重大度	常に 16（再スローは元の重大度）	指定する
SET XACT_ABORT	尊重する	尊重しない

引数なしの THROW は、CATCH ブロックの中でだけ使え、捕捉した例外を再発生させます。THROW の前の文はセミコロンで終えます。state は 0〜255 の tinyint で、範囲外は指定できません。新しいアプリケーションでは、RAISERROR より THROW を使います。

トランザクションとエラー

`SET XACT_ABORT` の既定は、T-SQL の文では OFF で、トリガーの中では ON です。ON にすると、実行時エラーでトランザクション全体が終了してロールバックされます。コンパイル エラー（構文エラーなど）には影響しません。[42]

`XACT_STATE()` は smallint を返します。

値	意味
1	コミットできるアクティブなトランザクションがある
0	トランザクションがない
-1	コミット不能（uncommittable）。完全なロールバックだけができ、書き込みはできない

SAMPLE

`@@TRANCOUNT` は、開いているトランザクションの入れ子の数です。コミット不能かどうかは `@@TRANCOUNT` では分かりません。反対に、内側のトランザクションがあるかどうかは `XACT_STATE()` では分かりません。CATCH の中では、まず `XACT_STATE()` を見ます。-1 なら `ROLLBACK` するしかありません。1 ならコミットも、ロールバックも選べます。0 なら、何もしません。TRY の外ならトランザクションを終わらせるエラーも、TRY の中で起きると、トランザクションをコミット不能（-1）にして CATCH へ移ります。XACT_ABORT を ON にしたトランザクションの実行時エラーもこれに当たり、エラー後に -1 になる場面が増えるので、CATCH の先頭でこの判定を入れる形が定石です。

AI 支援ツールを使うときのセキュリティ [1.4.1]

モデルへ渡るもの・権限・生成した SQL の確かめ

AI 支援ツールの安全は、3 つの問いに分けて考えます。何がモデルへ送られるか、どの権限で動くか、生成された SQL を誰が確かめるかです。

送られるものは、ツールごとに違い、一律には言えません。SSMS の GitHub Copilot は、プロンプトに加えて、接続先の SQL のバージョンやデータベースの情報も、モデルへ送ることがあります。プロンプト・応答・システム メタデータは保持せず、モデルの学習や再学習には使いません。VS Code の MSSQL 拡張の Copilot は、開いているファイルとアクティブな DB 接続にだけアクセスし、チャット内容を保存せず、プロンプトと補完をモデルの学習に使いません。[43]

Fabric の Copilot は、現在のユーザーが権限を持つデータにだけアクセスし、データは Azure OpenAI Service で処理されます。学習には使われず、ほかの顧客も利用できません。Fabric の SQL データベースの Copilot が参照できるのは、データベースのスキーマ、同じセッションの過去のメッセージ、実

SAMPLE

行した SQL の内容とエラー メッセージです。テーブルのデータそのものには触れません。[44]

権限の原則は 1 つです。セキュリティ境界は、Copilot の承認の画面ではなく、SQL Server の権限です。SSMS の Agent mode はクエリの実行前に承認を求めますが、境界として頼れるのは、ログインに与えた権限の強制です。権限のないテーブルへの SELECT は、Copilot からも実行できません。VS Code の Agent mode も、MSSQL 拡張と同じ接続コンテキストと資格情報を使い、新しい認証や権限の変更を持ち込みません。各ツール呼び出しにはユーザーの承認が要ります。だから対策は、Copilot 専用の最小権限のアカウントを作って接続することです。[45]

管理者向けの統制が 3 つあります。いずれも SSMS 22.7 以降です。

- データベースの `CONSTITUTION.md` で、データベース ユーザーか SQL ログインを指定し、Copilot のクエリを専用の最小権限アカウントで実行させる
- GitHub の AI Controls の MCP 設定に許可リストを置き、リスト外の MCP サーバーへの接続を SSMS にブロックさせる
- Visual Studio 用の管理用テンプレート（ADMX/ADML）の Copilot Settings > Disable Agent Mode で、Agent mode を完全に無効にする

Fabric のデータ エージェントを、Foundry・Copilot Studio・Microsoft 365 Copilot・MCP サーバーなど Fabric 以外のサービスで使うと、応答が Fabric のコンプライアンス境界やリージョンの外へ送られる可能性があります。規制のあるデータでは、この点を先に確かめます。

生成された SQL を、読まずに実行してはいけません。とくに UPDATE・DELETE・DDL は、テスト環境で試します。

SAMPLE

GitHub Copilot と Copilot in Fabric を使えるようにする [1.4.2]

SSMS と VS Code で使えるようにする

SSMS で GitHub Copilot を使うには SSMS 22 以降が要ります。GitHub Copilot のバッジから「Open Chat Window to Sign In」を選びます。未導入なら Visual Studio Installer で AI Assistance ワークロードを追加します。GitHub のアカウントでサインインします（Copilot Free でも使えます）。機能ごとの必要バージョンは次のとおりです。[46]

機能	必要なもの
オートコンプリート	SSMS 22.2 以降
データベース指示	SSMS 22.3 以降と AI Assistance ワークロード
Agent mode、管理用の統制	SSMS 22.7 以降

VS Code では、MSSQL 拡張をインストールし、有効な GitHub Copilot のサブスクリプションを用意します。SQL Server 2019/2022/2025、Azure SQL、Fabric の SQL データベース、Fabric Data Warehouse、Lakehouse（SQL analytics endpoint）に対応します。

Copilot in Fabric を有効にする

Fabric の Copilot は、有料の Fabric 容量（F2 以上）を持つテナントでは、既定で有効です。前提は 3 つです。[47]

1. 有料の Fabric 容量（F2 以上）か、Power BI Premium 容量（P1 以上）がある
2. 容量が対応リージョンにある

SAMPLE

3. ユーザー範囲を絞る必要があれば、セキュリティ グループを使う

トライアル SKU は使えません。Pro と PPU のワークスペースは直接は対応せず、Copilot 対応の容量に割り当てます。有効化は、Fabric の管理ポータルのテナント設定で、「Users can use Copilot and other features powered by Azure OpenAI」を有効にします。必要に応じて、リージョン外での処理の設定と、OpenAI サブプロセッサーの設定も有効にします。

テナント設定	既定
Users can use Copilot and other features powered by Azure OpenAI	有効
Capacities can be designated as Copilot in Fabric capacities	有効
OpenAI as a Microsoft Subprocessor に関する設定	無効
容量の地域の外での処理と保存の設定	無効

テナントまたは容量が、米国とフランスの外にある場合は、管理者が「Data sent to Azure OpenAI can be processed outside your capacity's geographic region…」を有効にするまで、Copilot は使えません。テナント設定が容量管理者に委任されているときは、各容量管理者が、その容量の Copilot 設定も有効にします。DAX クエリ ビューなど、特定の体験だけを個別に有効にはできません。ワークロード単位の制御です。主権クラウドは未対応です。

Fabric Data Warehouse の Copilot はプレビューです。テーブル名・ビュー名・列名・主キー/外部キーのメタデータから T-SQL を生成し、テーブルのデータは使いません。

SAMPLE

チャットのモデルと MCP のツールの選択 [1.4.3]

モデルの選択とモード

SSMS 22 のチャットは、プロンプト欄のモデル ピッカーでモデルを選びます。既定のモデルは、サブスクリプションで違います。Copilot Enterprise/Business では、モデルが SSMS で使えるようになる前に、管理者が Copilot 設定で Preview ポリシーを有効にする必要があります。[48]

独自のモデル（BYOM）は、ピッカーの Manage Models から、OpenAI・Anthropic・Google・xAI・Azure・Foundry Local のプロバイダーを選んで、API キーを入れて足します。Azure のモデルを足すときの入力は 3 つです。

- Display Name：ピッカーに出る名前
- Model ID：デプロイメント名
- Resource Endpoint：エンドポイントの URL

カスタム モデルは Copilot Chat のみに対応します。ツール呼び出し（tool calling）をサポートしないモデルは、使うべきではありません。埋め込みや意図の検出などは、Copilot API を使い続けることがあります。

Fabric の Copilot は、Azure OpenAI Service を既定で使います。管理者が OpenAI as a Microsoft Subprocessor の設定を有効にすると、指定した容量で OpenAI のモデルを使えます。Fabric のチャットに、個別のモデル選択の画面は記載されていません。[49]

VS Code の MSSQL 拡張では、次のように使い分けます。

モード	用途
Ask	単発の質問

SAMPLE

モード	用途
Edit	複数のファイルにまたがるクエリの書き換え
Agent	複数ステップの作業を任せる。MSSQL 拡張のツールが `@mssql` の指定なしで自動で使われる
Plan	変更の前に計画を書く

Fabric の SQL データベースの Copilot は、SSMS か VS Code から接続した場合、Ask mode が既定で読み取り専用です。Agent mode は、書き込みを含む複数ステップの作業ができますが、データの変更には明示的な承認が要ります。

MCP ツールの選択と承認

SSMS では、チャット ウィンドウ下部のモードのドロップダウンでエージェント モード（Agent）を選び、Tools アイコンでツールの一覧を確認します。SSMS 組み込みはローカル MCP サーバー `sql-tools` で、ツールを無効にすることはできません。MCP サーバーを足した直後は、ツールが無効です。Tools の一覧で、各ツールを手動で有効にします。MCP は Agent mode だけで使え、Ask mode では使えません。[50]

ツール承認の範囲は、ツールごとに違います。

- SSMS：Allow once / Allow for this session / Allow always から選ぶ。リセットは Tools > Options > GitHub > Copilot > Tools
- VS Code：Allow in this session / workspace / Always allow

Fabric MCP Server（ローカル）では、ツールの範囲を絞れます。VS Code では `fabricMcp.enabledServices` で限定し、読み取り専用にするには

SAMPLE

`fabricMcp.readOnly` を true にします。ほかのクライアントでは、開始引数に `--namespace` と `--read-only` を付けます。変更したら、サーバーを再起動するまで反映されません。

Copilot の指示ファイル [1.4.4]

リポジトリとユーザーの指示ファイル

指示ファイルは、Copilot へ毎回渡す前提条件を、Markdown で置く仕組みです。置き場所と名前は、ツールで違います。[51]

ツール	ファイル	補足
SSMS（リポジトリ）	リポジトリ ルートの `.github/copilot-instructions.md`	Tools > Options > GitHub > Copilot > Copilot Chat で読み込みを有効にする
SSMS（ユーザー）	`%USERPROFILE%/copilot-instructions.md`	リポジトリの指示と併用される
VS Code	`.github/instructions/<name>.instructions.md`	複数のファイルを、異なるファイル種別へ割り当てられる

SSMS では、指示はチャット画面に出ません。使われると、応答の References の一覧にファイル名が出ます。ファイルは `.md` の Markdown で、メインの見出しと、各セクションの小見出しで構成します。

VS Code のフロントマターの `applyTo` キーに glob パターン（例 `**/*.sql`）を書くと、その指示の適用範囲を絞れます。glob に一致する指示は、すべてのリクエストに自動で注入され、スラッシュ コマンドは要りません。ask・edit・agent・インライン補完のすべてに効きます。

SAMPLE

作成の手順は、Copilot Chat の歯車アイコンから Instructions & Rules > New instruction file を選び、場所に `.github/instructions`、ファイル名を入力します。フロントマター付きの `.instructions.md` が作られ、保存後に再読み込みは要りません。

```
---
applyTo: "**/*.sql"
---
# SQL の書き方
- スキーマ名で修飾する（dbo.Customer）
- 列は表の別名で修飾する
```

ほかのツールの同等のファイルは、Cursor が `.cursorrules`、OpenAI Codex が `AGENTS.md`、Claude Code が `CLAUDE.md` です。

データベース指示（拡張プロパティ）

データベースの中に指示を持たせる方法もあります。拡張プロパティとして保存し、名前は必ず `AGENTS.md` にします。1 つのオブジェクトに、`AGENTS.md` を 2 つ置くことはできません。更新は `sp_updateextendedproperty`、削除は `sp_dropextendedproperty` です。拡張プロパティ名 `CONSTITUTION.md` は、データベースに 1 つだけ設定でき、最も優先度の高い指示になります。そのデータベースを使う全ユーザーの Copilot 会話に適用されます。

前提は、SSMS 22.3 以降と AI Assistance ワークロード、Copilot 付きの GitHub アカウントです。Ask mode の Copilot は、変更クエリを実行できません。T-SQL を生成して保存します。[52]

SAMPLE

使い分けは、持ち主で決めます。個人の好みはユーザー レベル、チームのコード規約はリポジトリ、データベースの利用者全員に必ず守らせたいルールは `CONSTITUTION.md` です。

MCP サーバーへの接続 [1.4.5]

SQL MCP Server（Data API builder）

SQL MCP Server は、Data API builder（DAB）のバージョン 1.7 以降に含まれる、オープンソースで無料の機能です。2.0 より前の版では、最新の機能を使えません。MCP プロトコルのバージョンは 2025-06-18 が既定です。トランスポートは 2 つで、streamable HTTP は標準のホスティング用で、stdio はローカルや CLI 用です。[53]

設計方針は 3 つです。

- NL2SQL は意図的に非対応で、DAB のエンティティの抽象化とクエリ ビルダーで、決定的な T-SQL を生成する（NL2DAB）
- データ操作（DML）専用で、スキーマ変更（DDL）は対象外
- MCP は既定で有効で、特定のツール（例 delete-record）を絞りたいときだけ設定する

DML ツールは 7 つです。`describe_entities` ・ `create_record` ・ `read_records` ・ `update_record` ・ `delete_record` ・ `execute_entity` ・ `aggregate_records`（aggregate_records は 2.0 以降）です。操作の前に、`describe_entities` で対象を調べます。フィールドの説明（description）がなければ、エージェントは正しい列名を知りません。

stdio は `dab start --mcp-stdio role:<role-name>` で起動し、ロールを省略すると anonymous です。認証はシミュレーター プロバイダーを使い、リクエス

SAMPLE

トは 1 MB に制限されます。runtime の `mcp.enabled` が false のままでは、stdio は動きません。

mcp.json の書き方

VS Code は、ワークスペースの `.vscode/mcp.json` に `servers` を定義します。[54]

```
{
  "servers": {
    "sql-mcp-server": {
      "type": "http",
      "url": "http://localhost:5000/mcp"
    }
  }
}
```

stdio の場合は、VS Code が DAB を子プロセスとして起動します。`dab start` を別に動かす必要はありません。

SAMPLE

```
{
  "servers": {
    "sql-mcp-server": {
      "type": "stdio",
      "command": "dab",
      "args": ["start", "--mcp-stdio", "role:anonymous",
               "--loglevel", "error",
               "--config", "${workspaceFolder}/dab-config.json"]
    }
  }
}
```

HTTP は `type` と `url`、stdio は `type`・`command`・`args` を書きます。`role:anonymous` は、エンティティの権限で定義したロールに替えれば、アクセスの範囲を絞れます。SSMS では、Agent mode の Tools パネルから「Add custom MCP server」で Server ID と Type（HTTP は URL、stdio はコマンド）を入力するか、MCP registry から足します。手動で書くなら `%USERPROFILE%\.mcp.json` に `servers` を書きます。[55]

Fabric の MCP と権限

Fabric には 3 つの MCP があります。

SAMPLE

サーバー	接続と特徴
Fabric Core MCP Server（リモート）	エンドポイント `https://api.fabric.microsoft.com/v1/mcp/core`、Streamable HTTP、認証は Microsoft Entra ID の OAuth 2.0 ベアラートークン。VS Code では `.vscode/mcp.json` に `type: http` で足し、サーバー名は `fabric`
Fabric Data Warehouse MCP server	プレビュー。SQL analytics endpoint と Warehouse が対象で、ツールは `executeSQL` のみ。グローバルのエンドポイント `https://api.fabric.microsoft.com/v1/mcp/dataPlane/sqlEndpoint` と、アイテムを指定するエンドポイントがある
Fabric MCP Server（ローカル）	OSS でローカル実行。ツールセットは `docs`（サインイン不要）・`onelake`・`core`・`datafactory`

リモートかローカルかは、データの処理場所を決めません。[56]

権限は、読み取りと変更のツールの違いと、接続先の認証の 2 面で考えます。Fabric Core MCP Server は、REST API と同じ認証と認可で動きます。

トークンのスコープと、Fabric のワークスペースやアイテムの権限の両方が許可しなければ、成功しません。サービス プリンシパルを使うときは、REST API の呼び出しを許可するテナント設定が要ります。SQL MCP Server では、読み取りのツール（`read_records`）と、変更するツール（`create_record`・`update_record`・`delete_record`）は別なので、エンティティの権限（例 `anonymous:read`）で、許可する操作を絞ります。

SAMPLE

サンプルはここまでです

続き(第2章〜巻末)は、DP-800 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

Footnotes

1. https://learn.microsoft.com/en-us/sql/sql-server/maximum-capacity-specifications-for-sql-server（Bytes per row・Columns per table・Nonclustered/Clustered index key）。↩ ↩2

2. https://learn.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql 、 https://learn.microsoft.com/en-us/sql/sql-server/maximum-capacity-specifications-for-sql-server 。↩

3. https://learn.microsoft.com/en-us/sql/t-sql/data-types/vector-data-type（Syntax・Remarks・float16 のプレビューの注記）。↩

4. https://learn.microsoft.com/en-us/sql/relational-databases/indexes/create-indexes-with-included-columns（Design Recommendations・Limitations）。↩

5. https://learn.microsoft.com/en-us/sql/relational-databases/indexes/create-filtered-indexes（Design considerations・Limitations）。↩

6. https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview （ rowgroup ・ deltastore・tuple-mover）。↩

SAMPLE

7. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql （ MEMORY_OPTIMIZED ・ DURABILITY ）、https://learn.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/a-guide-to-query-processing-for-memory-optimized-tables（ネイティブ コンパイル プロシージャ）。↩

8. https://learn.microsoft.com/en-us/fabric/database/sql/limitations（Fabric の SQL データベースで作れないテーブル）。↩

9. https://learn.microsoft.com/en-us/sql/relational-databases/tables/creating-a-system-versioned-temporal-table 、https://learn.microsoft.com/en-us/sql/relational-databases/tables/manage-retention-of-historical-data-in-system-versioned-temporal-tables 、https://learn.microsoft.com/en-us/sql/relational-databases/tables/querying-data-in-a-system-versioned-temporal-table（FOR SYSTEM_TIME）。↩

10. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-external-table-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/statements/create-external-data-source-transact-sql（TYPE の一覧）。↩

11. https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-overview 、https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-updatable-ledger-tables 、https://learn.microsoft.com/en-us/sql/relational-databases/security/ledger/ledger-append-only-ledger-tables 。↩

SAMPLE

12. https://learn.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-architecture 、https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-sql-graph（CONNECTION・ON DELETE）。↩

13. https://learn.microsoft.com/en-us/sql/t-sql/data-types/json-data-type （保管形式・上限・modify・提供範囲）、https://learn.microsoft.com/en-us/sql/t-sql/functions/isjson-transact-sql 。↩

14. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-json-index-transact-sql （Arguments・Predicates supported・Requirement・Limitations）。↩

15. https://learn.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints 、https://learn.microsoft.com/en-us/sql/relational-databases/tables/create-primary-keys 、https://learn.microsoft.com/en-us/sql/relational-databases/tables/unique-constraints-and-check-constraints 。↩

16. https://learn.microsoft.com/en-us/sql/t-sql/statements/alter-table-column-constraint-transact-sql （参照アクション・WITH NOCHECK）、https://learn.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints （CASCADE の制限）。↩

17. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-sequence-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/next-value-for-transact-sql 、

SAMPLE

https://learn.microsoft.com/en-us/sql/relational-databases/sequence-numbers/sequence-numbers （Remarks・Cache management）。↩

18. https://learn.microsoft.com/en-us/sql/relational-databases/sequence-numbers/sequence-numbers（IDENTITY との違い）、https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql-identity-property 、https://learn.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-sequence-get-range-transact-sql 。↩

19. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-partition-function-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/statements/create-partition-scheme-transact-sql 、https://learn.microsoft.com/en-us/sql/relational-databases/partitions/partitioned-tables-and-indexes 。↩

20. https://learn.microsoft.com/en-us/sql/relational-databases/partitions/partitioned-tables-and-indexes （aligned index・SWITCH）。↩

21. https://learn.microsoft.com/en-us/fabric/database/sql/limitations（パーティション DDL の制限）。↩

22. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-view-transact-sql（Arguments・Remarks・Permissions）。↩

23. https://learn.microsoft.com/en-us/sql/relational-databases/views/create-indexed-views （Required SET options・

SAMPLE

GROUP BY の要件・NOEXPAND）。↩

24. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql 、https://learn.microsoft.com/en-us/sql/relational-databases/user-defined-functions/create-user-defined-functions-database-engine 、https://learn.microsoft.com/en-us/sql/relational-databases/user-defined-functions/scalar-udf-inlining （Inlineable scalar UDF requirements・Enable/Disable）。↩

25. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql 、https://learn.microsoft.com/en-us/sql/relational-databases/user-defined-functions/create-user-defined-functions-database-engine 。↩

26. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql（Syntax・Arguments・Remarks）。↩

27. https://learn.microsoft.com/en-us/sql/t-sql/statements/execute-as-clause-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql（Permissions）。↩

28. https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide （Parameter sensitivity）、https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql （WITH RECOMPILE）、https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query（OPTIMIZE FOR）。↩

SAMPLE

29. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql 、https://learn.microsoft.com/en-us/sql/relational-databases/triggers/dml-triggers 、https://learn.microsoft.com/en-us/sql/relational-databases/triggers/use-the-inserted-and-deleted-tables 。↩

30. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql （DDL・logon・再帰・nested triggers・Permissions）。↩

31. https://learn.microsoft.com/en-us/sql/t-sql/queries/with-common-table-expression-transact-sql （Guidelines・Recursive CTE・MAXRECURSION）。↩

32. https://learn.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/row-number-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/rank-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/dense-rank-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/ntile-transact-sql 。↩

33. https://learn.microsoft.com/en-us/sql/t-sql/functions/lag-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/lead-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql（ROWS/RANGE・既定のフレーム）、https://learn.microsoft.com/en-us/sql/t-sql/queries/select-window-transact-sql 。↩

SAMPLE

34. https://learn.microsoft.com/en-us/sql/t-sql/functions/json-object-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/json-array-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/json-objectagg-transact-sql 。↩

35. https://learn.microsoft.com/en-us/sql/t-sql/functions/json-value-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/json-query-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/openjson-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/json-contains-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/json-modify-transact-sql 。↩

36. https://learn.microsoft.com/en-us/sql/relational-databases/regular-expressions/overview 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/regexp-like-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/regexp-replace-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/regexp-instr-transact-sql 。↩

37. https://learn.microsoft.com/en-us/sql/t-sql/functions/regexp-matches-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-sql/functions/regexp-split-to-table-transact-sql 。↩

38. https://learn.microsoft.com/en-us/sql/t-sql/functions/edit-distance-transact-sql 、 https://learn.microsoft.com/en-us/sql/t-

SAMPLE

sql/functions/edit-distance-similarity-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/jaro-winkler-distance-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/jaro-winkler-similarity-transact-sql 。↩

39. https://learn.microsoft.com/en-us/sql/t-sql/queries/match-sql-graph 、https://learn.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-shortest-path 、https://learn.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-architecture 。↩

40. https://learn.microsoft.com/en-us/sql/relational-databases/performance/subqueries 、https://learn.microsoft.com/en-us/sql/t-sql/queries/from-transact-sql（APPLY）。↩

41. https://learn.microsoft.com/en-us/sql/t-sql/language-elements/try-catch-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/language-elements/throw-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/language-elements/raiserror-transact-sql 。↩

42. https://learn.microsoft.com/en-us/sql/t-sql/statements/set-xact-abort-transact-sql 、https://learn.microsoft.com/en-us/sql/t-sql/functions/xact-state-transact-sql 。↩

43. https://learn.microsoft.com/en-us/ssms/github-copilot/overview 、https://learn.microsoft.com/en-us/sql/tools/visual-studio-code-extensions/github-copilot/limitations-and-known-issues 。↩

SAMPLE

44. https://learn.microsoft.com/en-us/fabric/fundamentals/copilot-privacy-security 、 https://learn.microsoft.com/en-us/fabric/database/sql/copilot-sql-database 、 https://learn.microsoft.com/en-us/fabric/admin/service-admin-portal-copilot 。↩

45. https://learn.microsoft.com/en-us/ssms/github-copilot/agent-mode 、 https://learn.microsoft.com/en-us/ssms/github-copilot/admin-controls 、 https://learn.microsoft.com/en-us/sql/tools/visual-studio-code-extensions/github-copilot/agent-mode 。↩

46. https://learn.microsoft.com/en-us/ssms/github-copilot/get-started 、https://learn.microsoft.com/en-us/ssms/github-copilot/overview 、 https://learn.microsoft.com/en-us/sql/tools/visual-studio-code-extensions/github-copilot/custom-instructions 。↩

47. https://learn.microsoft.com/en-us/fabric/fundamentals/copilot-enable-fabric 、 https://learn.microsoft.com/en-us/fabric/admin/service-admin-portal-copilot 、 https://learn.microsoft.com/en-us/fabric/database/sql/copilot-sql-database 、 https://learn.microsoft.com/en-us/fabric/data-warehouse/copilot 。↩

48. https://learn.microsoft.com/en-us/ssms/github-copilot/ai-models 。↩

49. https://learn.microsoft.com/en-us/fabric/fundamentals/copilot-privacy-security 、 https://learn.microsoft.com/en-us/sql/tools/visual-studio-code-extensions/github-copilot/agent-

SAMPLE

mode 、 https://learn.microsoft.com/en-us/fabric/database/sql/copilot-sql-database 。↩

50. https://learn.microsoft.com/en-us/ssms/github-copilot/agent-mode 、 https://learn.microsoft.com/en-us/ssms/github-copilot/mcp-servers 、 https://learn.microsoft.com/en-us/rest/api/fabric/articles/mcp-servers/pro-dev-local/overview-local-mcp-server 。↩

51. https://learn.microsoft.com/en-us/ssms/github-copilot/custom-instructions 、https://learn.microsoft.com/en-us/sql/tools/visual-studio-code-extensions/github-copilot/custom-instructions 。↩

52. https://learn.microsoft.com/en-us/ssms/github-copilot/database-instructions 。↩

53. https://learn.microsoft.com/en-us/azure/data-api-builder/mcp/overview 、 https://learn.microsoft.com/en-us/azure/data-api-builder/mcp/data-manipulation-language-tools 。↩

54. https://learn.microsoft.com/en-us/azure/data-api-builder/mcp/quickstart-visual-studio-code 。↩

55. https://learn.microsoft.com/en-us/ssms/github-copilot/mcp-servers 。↩

56. https://learn.microsoft.com/en-us/rest/api/fabric/articles/mcp-servers/core-remote/get-started-core 、 https://learn.microsoft.com/en-us/rest/api/fabric/articles/mcp-servers/core-remote/permissions-core-mcp-server 、

SAMPLE

https://learn.microsoft.com/en-us/fabric/data-warehouse/data-warehouse-mcp-server 、 https://learn.microsoft.com/en-us/rest/api/fabric/articles/mcp-servers/pro-dev-local/overview-local-mcp-server 。↩

SAMPLE

DP-800 学習用テキスト(無料サンプル)

版
2026年11月版(DP-800 出題範囲(Skills measured)2026-03-12版 準拠)

発行
IT資格道場(https://www.shikaku-doujou.com)

お問い合わせ
info@shikaku-doujou.com

Microsoft、Azure、SQL Server、Microsoft Fabric および Microsoft の製品名・サービス名は、Microsoft Corporation の米国およびその他の国における商標または登録商標です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式シラバスの内容を基に作成したもので、Microsoft Corporation との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。

SAMPLE