IT資格道場

SnowPro Core Certification（COF-C03）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

SAMPLE

無料サンプル（第1章まで）

Snowflake SnowPro Core Certification（COF-C03）学習ガイド（英語版 2026年7月8日更新・日本語版 2026年2月20日更新）準拠
2026年11月版

© 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

この試験の範囲 [overview]

第1章 Snowflake AI Data Cloud Features and Architecture [1.1/1.2/1.3/1.4/1.5/1.6]

- Snowflake アーキテクチャ [1.1]
- Snowflake のインターフェースとツール [1.2]
- オブジェクト階層と種類 [1.3]
- 仮想ウェアハウスの構成 [1.4]
- ストレージの概念 [1.5]
- AI/ML とアプリケーション開発の機能 [1.6]

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

この試験の範囲 [overview]

SnowPro Core Certification（COF-C03）は、Snowflake AI Data Cloud を使う人が、Snowflake の基本の知識を持っているかを問う試験です。出題は 100 問の選択式（択一と複数選択）で、試験時間は 115 分、合格は 1000 点満点の 750 点です。受験料は 175 米ドルで、前提となる資格はありません。日本語でも受けられます（COF-C03-JPN）。

出題範囲は 5 つの分野に分かれます。この教材も、同じ順に章を立てます（下の表）。配点は、学習ガイドの割合です。

分野	配点
1 Snowflake AI Data Cloud の機能とアーキテクチャ（Snowflake AI Data Cloud Features and Architecture）	31%
2 アカウント管理とデータガバナンス（Account Management and Data Governance）	20%
3 データのロード、アンロード、接続（Data Loading, Unloading, and Connectivity）	18%
4 パフォーマンスの最適化、クエリ、変換（Performance Optimization, Querying, and Transformation）	21%
5 データコラボレーション（Data Collaboration）	10%

- 学習ガイドの細目: 19 個
- 章と節の見出しの後ろにある [2.1] のような印: その細目の番号

SAMPLE

第1章 Snowflake AI Data Cloud Features and Architecture [1.1/1.2/1.3/1.4/1.5/1.6]

この章は試験の31%を占め、5分野の中で最も大きい分野です。Snowflakeの作り、使う道具、オブジェクトの入れ子、ウェアハウス、テーブルとビュー、AIとMLの機能を扱います。クラウドの基礎とSQLの構文の基礎は前提の知識なので、説明しません。

Snowflake アーキテクチャ [1.1]

Snowflakeは、データを置く層、計算をする層、全体を調整する層の3つに分かれています。3つは別々に大きくでき、料金の数え方も別です。どの層が何の費用を生むかが、この節の中心になります。

3つの層と共有ディスク・非共有の組み合わせ

3つの層は、データベースストレージ層、コンピュート層、クラウドサービス層です。[1]

- **データベースストレージ層**: テーブルのデータを置く場所。クラウドのオブジェクトストレージに保存される
- **コンピュート層**: 仮想ウェアハウスが計算をする場所。クエリの実行、データの読み込み、Snowparkのコードの実行を担う
- **クラウドサービス層**: 認証、アクセス制御、メタデータ、クエリの解析と最適化など、全体を調整するサービスの集まり

作りの特徴は、共有ディスク型と非共有型の混成であることです。共有ディスク型と同じく、永続化したデータは中央の1か所にあり、どの計算ノードからも読めます。非共有型と同じく、クエリはMPP（超並列処理）のコンピュートクラ

SAMPLE

スターで処理され、各ノードがデータの一部を手元に持ちます。共有ディスクの扱いやすさと、非共有のスケールアウトの強さを、両方取る設計です。[1]

覚え方は「データは共有、計算は各ウェアハウスで独立」です。ウェアハウスは互いに計算資源を共有しないので、あるウェアハウスの負荷が別のウェアハウスの性能に影響しません。夜間のバッチ用ウェアハウスが全力で動いていても、営業部の BI 用ウェアハウスは遅くなりません。同じデータを読んでいても、計算が別だからです。

Snowflake は、手元のパソコンにも、オンプレミスにも、自前のプライベートクラウドにも入れられません。公開クラウドの上で、Snowflake が運用するサービスとして使います。ハードウェアの選定、ソフトウェアの更新、チューニングは Snowflake の側で行われます。[1]

クラウドサービス層の仕事

クラウドサービス層が受け持つ仕事は次のとおりです。[1]

- セキュリティ、認証、アクセス制御
- Snowflake Horizon Catalog
- クラウド基盤とのインフラ管理
- メタデータの管理（SNOWFLAKE データベースと Information Schema を含む）
- クエリの解析と最適化
- 法令や規格への準拠

この層も、Snowflake がクラウド事業者から借りた計算インスタンスの上で動きます。状態を持たない計算資源が複数のアベイラビリティゾーンにまたがり、可用性の高い分散メタデータストアで全体の状態を管理します。ログイン

SAMPLE

からクエリの割り振りまで、ユーザーの要求を最初に受ける層と考えると整理できます。リクエストされたクエリのキャッシュの扱いも、この層の仕事です。[2]

ウェアハウスが要る操作と要らない操作の区別は、場面として出題されます。クエリと、データのロードを含むすべての DML には、ウェアハウスが必要です。[3] 一方、SHOW TABLES のようにメタデータだけで答える操作は、ウェアハウスを必要とせず、料金もかかりません。[4] ウェアハウスが停止していても、テーブルの一覧は見られます。データの行を読む SELECT は、ウェアハウスが無ければ実行できません。

セッションには、初期状態でウェアハウスが結びついていません。ユーザーの既定のウェアハウスか、USE WAREHOUSE で指定したウェアハウスが必要になります。

クラウドサービスのクレジットの請求

クラウドサービス層の使用量にもクレジットがかかります。請求されるのは、1日のクラウドサービスの使用量が、同じ日の仮想ウェアハウスの使用量の10%を超えた分だけです。計算は1日ごとに、UTC の暦日で行います。[2]

たとえば、ある日、ウェアハウスが50クレジット、クラウドサービスが8クレジットだったとします。ウェアハウスの10%は5クレジットなので、クラウドサービスのうち5クレジットは調整で差し引かれ、請求されるのは残りの3クレジットです。合計は50+8−5で53クレジットになります。クラウドサービスが3クレジットだけだった日は、差し引きも3クレジットで、クラウドサービスの請求は0です。1日の調整は、実際の使用量を超えません。[2]

注意する点が3つあります。サーバーレス機能（自動クラスタリングなど）の使用量は、10%の計算に入りません。月の合計をならして10%を引くのではな

SAMPLE

く、毎日この計算をして合算します。月の合計が10%未満に見えても、特定の日に超えていれば、その日の超過分は請求されます。[2]

ストレージ層の性質と保存の費用

データをテーブルにロードすると、Snowflake は内部で最適化した、圧縮された列指向の形式に並べ替え、クラウドのストレージに保存します。ファイルの大きさ、構造、圧縮、メタデータ、統計はすべて Snowflake が管理するので、ユーザーはこれらを指定しません。扱えるデータは、構造化、半構造化（JSON や XML）、非構造化（文書、画像、音声）の3種類です。暗号化は自動で、全データに適用されます。[5][6]

保存の費用は、1か月あたりのテラバイト単位の定額で決まります。対象は4つあります。

- ロードやアンロード用にステージへ置いたファイル
- テーブルのデータ（Time Travel の履歴を含む）
- テーブルの Fail-safe
- 元のテーブルで消えたデータを参照しているクローン

テーブルのデータは、圧縮後の大きさで計算します。ステージに置いたファイルは、置いたファイルの大きさ（圧縮していてもいなくても）で計算します。[7]

エディション

エディションは4つあり、上位は下位の機能をすべて含みます。エディションは、クレジットとストレージの単価を決める要素の1つです。ほかに、アカウントのあるリージョンと、従量課金（On Demand）か前払いの契約（Capacity）かも、単価に影響します。[8]

SAMPLE

エディション	位置づけ
Standard	入門の版。標準の機能を制限なく使える
Enterprise	Standard の全機能に、大規模な組織向けの機能を足す
Business Critical	旧名 Enterprise for Sensitive Data。より高いデータ保護。HIPAA や HITRUST CSF の対象になる PHI を扱う
Virtual Private Snowflake（VPS）	Business Critical の全機能を、他のアカウントとハードウェアを共有しない独立した環境で提供

エディションごとの機能の違いは、最もよく問われます。次の表は、境目になる機能だけを抜き出したものです。[6]

機能	Standard	Enterprise	Business Critical	VPS
Time Travel 1日まで	あり	あり	あり	あり
Time Travel 90日まで（拡張）	なし	あり	あり	あり
Fail-safe（Time Travelの後の7日）	あり	あり	あり	あり
マルチクラスターウェアハウス	なし	あり	あり	あり
マテリアライズドビュー	なし	あり	あり	あり

SAMPLE

機能	Standard	Enterprise	Business Critical	VPS
列レベルのセキュリティ（マスキングポリシー）	なし	あり	あり	あり
行レベルのセキュリティ（行アクセスポリシー）	なし	あり	あり	あり
クエリアクセラレーション、検索最適化	なし	あり	あり	あり
暗号化データの定期的な鍵の更新（rekey）	なし	あり	あり	あり
週次リリースへの24時間前の早期アクセス	なし	あり	あり	あり
Tri-Secret Secure（顧客管理の暗号鍵）	なし	なし	あり	あり
クラウド事業者のプライベート接続（PrivateLinkなど）	なし	なし	あり	あり

SAMPLE

機能	Standard	Enterprise	Business Critical	VPS
アカウントのフェイルオーバーとフェイルバック	なし	なし	あり	あり
PHI（HIPAA、HITRUST CSF）、PCI DSSへの対応	なし	なし	あり	あり
専用のメタデータストアと計算資源	なし	なし	なし	あり
Snowflake Marketplaceの公開リスティング	あり	あり	あり	なし

Standard でも使える機能は多くあります。仮想ウェアハウス、リソースモニター、SSO、ネットワークポリシー、多要素認証、オブジェクトレベルのアクセス制御、ダイナミックテーブル、外部テーブル、Snowpipe、ストリームとタスク、Snowflake CLI などです。間違えやすいのは、Time Travel の最大期間と、マルチクラスターウェアハウスとマテリアライズドビューが Enterprise 以上であることです。Business Critical の特徴は、規制への対応、顧客管理の鍵、プライベート接続、フェイルオーバーにあります。[6]

クラウドとリージョン

Snowflake は、AWS、Microsoft Azure、Google Cloud の3つのクラウド上で動きます。アカウントはクラウドとリージョンを選んで作り、1つのアカウントは

SAMPLE

1つのリージョンに置かれます。リージョンは、データを地理的にどこに保存するかと、計算資源がどこに用意されるかを決めます。複数のリージョンでSnowflake を使うなら、必要なリージョンごとにアカウントを持ちます。[9]

規制への対応と、リージョンの種類

Business Critical 以上のエディションには、規制のある業界の要件に合わせた機能があります。PHI を HIPAA と HITRUST CSF に沿って扱うには、データを Snowflake に置く前に、利用する組織と Snowflake Inc. の間で、署名済みの事業提携契約（BAA）を結ぶ必要があります。米国の連邦・州政府の要件（FedRAMP や ITAR）と、アジア太平洋の指定リージョンでの IRAP の Protected（P）データへの対応も、Business Critical と VPS の機能で、Standard と Enterprise にはありません。[6]

VPS は、金融機関のように、最も厳しい要件を持つ組織向けです。機微なデータを集めて、分析し、共有する大企業を想定しています。[8]

米国政府の規制を受ける組織のために、リージョンが2種類あります。1つは、米国の商用リージョンのうち、政府の標準に一部対応するものです。もう1つは SnowGov リージョンで、AWS GovCloud（US）と Microsoft Azure Government の上にあり、米国政府と政府の請負業者だけが使えます。[9]

SnowGov リージョンは、FedRAMP、DoW の Impact Levels、GovRAMP、TX-RAMP、FIPS などの標準に対応します。クラウド事業者の政府リージョンは、ほかの商用リージョンとの間で、イベント通知を送ることも受けることもできません。[9]

日本で使うリージョンは、AWS の東京（ap-northeast-1）と大阪（ap-northeast-3）、Azure の Japan East（東京）です。この3つは、日本政府の情報

SAMPLE

システムのセキュリティ評価制度である ISMAP の評価を完了し、準拠に対応しています。[9]

Snowflake のインターフェースとツール [1.2]

Snowflake を操作する窓口は、Web の画面、コマンドラインのクライアント、開発環境の拡張、プログラムから使うドライバーとコネクタに分かれます。この作業にはどの道具が向くか、という判断が出題されます。

Snowsight

Snowsight は Snowflake の Web 画面で、SQL か Python でデータを扱います。主にできることは次のとおりです。[10]

- ワークシートで SQL を書いて実行し、共有とフォルダーで整理する。
- 結果をグラフにし、組み込みの可視化でダッシュボードを作る。
- クエリの性能と履歴を見る（Monitoring の Query History）。
- データベースやオブジェクトを閲覧する。
- Python ワークシートで Snowpark Python を書く。
- Snowflake Notebooks と Streamlit のアプリを作って動かす。
- ユーザーとロールの管理、コスト管理、Trust Center、予算、データロードの監視など、管理の操作を行う。

クエリ履歴には、画面が裏で実行したクライアント生成のクエリも、フィルターで出せます。ページによってはウェアハウスが必要です。Task Run History やデータのプレビューのように、メタデータ以外を表示する画面がそうで、X-Small で足りることが多いと案内されています。[4]

SAMPLE

Snowflake CLI

Snowflake CLI は、コマンド名が `snow` のオープンソースのコマンドラインツールです。新しく入れる時も、新しい作業でも、勧められるクライアントはこちらで、SnowSQL は旧来のクライアントという位置づけです。SnowSQL には新機能が追加されません。サポートは 1.5.x が2028年4月16日までです。[11]

SQL の実行に加えて、開発者向けの作業を扱います。Snowpark、Streamlit in Snowflake、Native App、Snowpark Container Services、Snowflake Notebooks で動くアプリを扱い、作成、管理、更新、確認ができます。コマンド群は `snow sql`、`snow stage`、`snow object`、`snow streamlit`、`snow app`、`snow snowpark`、`snow notebook`、`snow git` などです。requirements.txt に書かれ、Anaconda に無いパッケージも、アプリのパッケージに含めて Snowflake へ配れます。ネイティブライブラリに依存するパッケージには、この機能は効きません。[11][12]

接続は `config.toml` の `[connections]` に定義します。先頭の `default_connection_name` で、既定の接続を決められます。VS Code の拡張や Cortex Code など、複数の開発ツールで接続を共有したいときは、接続だけを `connections.toml` に書きます。macOS と Linux では、config.toml の権限を所有者の読み書きだけに限る必要があります（`chmod 0600`）。`--temporary-connection` を付ければ、ファイルに定義せずに接続できます。[13]

IDE との連携

Visual Studio Code には Snowflake の拡張があります。Visual Studio Marketplace から入れるか、`.vsix` のファイルを入れます。サインインすると、サイドバーに、アカウントの情報、既定のロール、データベースの一覧（Object Explorer）、クエリの履歴が出ます。VS Code の中で SQL を書いて実行し、結果を見られます。[14]

SAMPLE

Snowpark Python のコードの中の SQL には、デバッグ、構文の色分け、補完が効きます。Native App の作成と配備、ステージの中身の管理も、拡張からできます。接続の設定は、Snowflake CLI と同じ `connections.toml` を使います。[14]

拡張の Native App の機能が対応する Snowflake CLI は、バージョン 2.2.x と 3.x です。作業は NATIVE APP ペインから行い、Create new from template で雛形を選びます。basic は最小限のコードと手引き、spcs-basic は Snowpark Container Services の拡張コードと Streamlit のコード例を含みます。[14]

配備の選び方は2通りです。Run（deploy and re-install）は、大きな変更をして、アプリケーションオブジェクトが要る時に使います。Deploy は、アプリケーションパッケージとステージのファイルだけを配備し、アプリケーションオブジェクトが要らない時に使います。[14]

Teardown は、プロジェクト定義が指すアプリケーションオブジェクトとパッケージを削除します。App Debug Mode をオンにすると、コンシューマーには見えないオブジェクトも見られます。[14]

Object Explorer では、テーブルやビューのデータプレビューが最大100行まで出ます。ステージの中身も一覧にでき、ローカルのファイルを上げる Upload は内部ステージだけが対象です。それ以外の操作は、内部ステージと外部ステージの両方で使えます。[14]

クライアントの使い分け

道具	向く用途
Snowsight	ブラウザで SQL、ダッシュボード、管理、監視

SAMPLE

道具	向く用途
Snowflake CLI（snow）	開発者向けの作業、アプリの配備、CI/CD、スクリプトでの SQL 実行
SnowSQL	旧来のコマンドラインクライアント。既存の資産の維持
VS Code の拡張	開発環境の中で SQL と Snowpark を書く
ドライバーとコネクタ（JDBC、ODBC、Python、Node.js、.NET、Go、PHP）	自作のアプリや BI ツールから接続する
Snowflake SQL API	REST で SQL を実行する
Snowflake Terraform プロバイダー	オブジェクトをコードで管理する（Infrastructure as code）

コマンドラインやドライバーが使う既定のウェアハウスは、3段階で決まります。ユーザーの既定のウェアハウス、クライアントの設定ファイル、コマンドラインやドライバーの接続パラメーターの順に、後のものが前のものを上書きします。セッション中に USE WAREHOUSE を実行すれば、さらにいつでも変えられます。[3]

オブジェクト階層と種類 [1.3]

Snowflake のオブジェクトは、入れ物の階層に入っています。一番上が組織で、その下にアカウント、データベース、スキーマと続きます。テーブル、ビュー、関数、ステージなどは、スキーマの中に置かれます。[15]

入れ子と完全修飾名

オブジェクトの場所は、`データベース名.スキーマ名.オブジェクト名` の完全修飾名で指します。セッションで USE DATABASE と USE SCHEMA を実行して

SAMPLE

あれば、オブジェクト名だけで指せます。同じ名前のテーブルは、スキーマが違えば別々に作れます。

組織とアカウントのオブジェクト

組織は、事業体が持つアカウントをつなげる、Snowflake のオブジェクトです。ユーザーが組織を直接作ることはありません。自分で申し込んでアカウントを作ると、システムが決めた名前で組織が自動で作られ、以降は同じ組織の下にアカウントを足せます。組織の管理者は、リージョンやクラウドをまたいで、組織の全アカウントを見て、作って、管理できます。組織のレベルの仕事は、アカウントの管理、組織全体の使用量の確認、レプリケーションとフェイルオーバーの設定、アカウントをまたぐデータ共有などです。[16]

組織の管理者になる経路は2つです。複数アカウントの組織では、組織アカウントの GLOBALORGADMIN ロールを使います。ORGADMIN ロールを使う経路もありますが、複数アカウントの組織では使わなくなる方向です。[16]

組織には、3種類のアカウントが属せます。組織アカウントは、組織の管理者が複数アカウントの組織を管理し、ORGANIZATION_USAGE スキーマのプレミアムのビューを使うための特別なアカウントです。ふつうの Snowflake アカウントには、トライアルのアカウントも含まれます。[16]

Snowflake Open Catalog アカウントは、Open Catalog で定義したカタログを管理する特別なアカウントです。[16]

組織の名前とアカウントは、組織の管理者が SHOW ACCOUNTS で見られます。Snowsight では Admin の Accounts に出て、組織の名前がアカウント名の上に表示されます。どのロールでも、CURRENT_ORGANIZATION_NAME 関数で、いまのアカウントが属する組織の名前を取れます。[16]

SAMPLE

システムが付けた組織の名前を変えたい時は、Snowflake サポートに依頼します。その際、元のアカウントの URL を残すかを選べ、残した URL は90日後に自動で削除されます。[16]

アカウントのレベルに置くオブジェクトは、ユーザー、ロール、ウェアハウス、データベース、統合（integration）、リソースモニター、ネットワークポリシーなどです。データベースロールのように、データベースの中にあってスキーマには入らないオブジェクトもあります。ウェアハウスとユーザーは、データベースにもスキーマにも入りません。

スキーマに置くオブジェクト

スキーマの中に置くのは、テーブル、ビュー、ステージ、ファイル形式、パイプ、シーケンス、ストリーム、タスク、UDF、ストアドプロシージャ、ML モデルなどです。共有（share）は、アカウントのオブジェクトです。作るには、アカウントに対する CREATE SHARE の権限が要ります。

オブジェクト	役割
ステージ	ファイルの置き場所
ファイル形式	CSV や JSON などの読み方を、名前つきで持つ。CREATE FILE FORMAT で作り、ロードとアンロードの両方で使う
パイプ	Snowpipe がファイルを自動で取り込むための定義
ストリーム	テーブルの変更の追跡
タスク	SQL の定期実行

ML モデルは、モデルレジストリがスキーマレベルのオブジェクトとして保存し、学習したモデルの登録、版の管理、推論に使います。[17]

SAMPLE

アプリケーション（Native App）は、プロバイダーが作ったアプリを、コンシューマーのアカウントに入れたオブジェクトです。

UDF とストアドプロシージャ

UDF（ユーザー定義関数）は、組み込み関数に無い処理を足します。ハンドラーは Java、JavaScript、Python、Scala、SQL で書けます。種類は次のとおりです。[18]

種類	返すもの
UDF（スカラー関数）	入力の1行につき1つの値
UDAF（集計関数）	複数の行にまたがる集計の値
UDTF（表関数）	入力の1行につき表の形の値
ベクトル化 UDF、UDTF	pandas の DataFrame で、行の束を一度に処理する

外部関数（external function）は UDF の一種で、Snowflake の外で動くコードを呼び出して処理を足します。Snowflake の外で実行されるコードを、リモートサービスと呼びます。[19]

UDAF は、複数の行にまたがって、合計、平均、件数、最小と最大、標準偏差、推定などを計算します。ベクトル化 UDF は、入力の行の束を pandas の DataFrame で受け取り、結果を pandas の配列か Series で返します。ベクトル化 UDTF は、同じく行の束を DataFrame で受け取り、表形式の結果を返します。[18]

ストアドプロシージャは、複数のデータベース操作を何度も行う作業を自動にします。CALL で呼びます。UDF との違いは次のとおりです。[20]

SAMPLE

	ストアドプロシージャ	UDF
呼び方	CALL ストアドプロシージャ名	SELECT の式の中で呼ぶ
値	返してもよいが、返さなくてもよい	必ず式で値を返す
DDL と DML	実行できる（一時テーブルの削除、古いデータの削除、ユーザーの追加など）	向かない。単純な SELECT 程度
向く用途	管理作業、DDL と DML	行や集計ごとに値を計算して SQL の中で使う

SELECT の列に関数として置きたい、行ごとに値が要るという場面は UDF です。テーブルを掃除する、ユーザーを作るなど、DDL や DML を実行したい場面はストアドプロシージャです。[20]

所有者の権限と呼び出し側の権限

ストアドプロシージャは、所有者の権限か呼び出し側の権限の、どちらか一方で動きます。作る時に決め、既定は所有者の権限（EXECUTE AS OWNER）で、呼び出し側の権限にするなら EXECUTE AS CALLER を指定します。作った後でも、所有者が ALTER PROCEDURE で切り替えられます。[21]

所有者の権限のプロシージャは、呼び出した人にその権限が無くても、所有者が持つ権限で動きます。Doctor ロールが medical_records の古い行を削除するプロシージャを作り、Nurse ロールに実行権限を渡したとします。Nurse に削除権限が無くても、そのプロシージャ経由で削除できます。広い権限を渡さずに、特定の管理作業だけを別のロールへ任せたい場面に向きます。所有者の権限のプロシージャは、セッションの状態を変えられません。呼び出した

SAMPLE

人の現在のウェアハウスは引き継ぎ、データベースとスキーマはプロシージャを作った場所のものを使います。[21]

呼び出し側の権限のプロシージャは、呼び出した人の権限で動きます。呼び出した人が外で実行できない文は、中でも実行できません。呼び出した人の現在のウェアハウス、データベース、スキーマを引き継ぎます。セッション変数とセッションパラメーターを見たり、設定したりもでき、変更はプロシージャの終了後も残ります。呼び出した人のセッションの情報を読む必要があるなら、呼び出し側の権限を選びます。[21]

所有者の権限のプロシージャの中で、呼び出した人が権限を持つ表、ビュー、関数を使いたい時は、呼び出した人にその参照（reference）を渡してもらいます。プロシージャは所有者の権限で動いたまま、渡された対象だけを呼び出し側の権限で扱えます。呼び出し側の権限のプロシージャを、権限の一部だけで動かしたい時は、管理者が caller grants で、使える権限を絞ります。[21]

ストアドプロシージャが返せるのは、1つの値か、ハンドラーの言語が対応していれば表形式のデータです。ハンドラーのコードを CREATE PROCEDURE の文の中に書く（インライン）か、パッケージしてステージに置くかは、言語で決まります。Java、Python、Scala はインラインでもステージでも書けます。[22]

JavaScript と Snowflake Scripting はインラインだけです。CREATE PROCEDURE に TEMP か TEMPORARY を付けると、現在のセッションの間だけ続き、その後に削除される一時のプロシージャを作れます。[22]

シーケンス

シーケンスは、セッションや文をまたいで、同時に走る文の間でも重ならない、一意の番号を作るオブジェクトです。主キーのような、一意の値が要る列に使います。[23]

SAMPLE

```
CREATE SEQUENCE seq_order START = 1000 INCREMENT = 5;
SELECT seq_order.NEXTVAL;
```

NEXTVAL で次の値を得ます。START と INCREMENT の既定値は、どちらも1です。間隔を負の値にすると、値は減っていきます。列に IDENTITY や AUTOINCREMENT を指定しても、自動で採番できます。[23]

押さえる点は、番号に抜け（欠番）があり得ることです。Snowflake は、抜けのない連番を保証しません。生成した値が行に使われなくても、そのまま飛びます。同時に複数の INSERT が走ると、値が増加する順に並ぶ保証もなく、NOORDER を指定すると、増加の順序さえ保証されません。連続した番号が必須の業務には、シーケンスをそのまま使えません。INCREMENT を正から負へ（あるいは逆に）変えると、すでに出た値が再び出ることがあります。[23]

シーケンスの値は、64ビットの符号つき整数（-2^63 から 2^63-1）の範囲で表されます。次の値の内部の表現がこの範囲を超えると、クエリが失敗し、その値は失われます。そのときは、間隔を小さくするか、開始値の小さい新しいシーケンスを作ります。[23]

開始値1、間隔1のシーケンスが範囲を使い切ることは、まず起きません。[23]

シーケンスは、出した値ごとに、間隔の分だけ後ろの値を予約します。値100を出した時、間隔が2なら100と101、間隔が10なら100から109、間隔が-5なら96から100が予約されます。間隔を変えない限り、予約された値は出ません。[23]

CURRVAL（直前に出した値を参照する形）は、ほかのデータベースにはありますが、Snowflake にはありません。最初の表に1行入れ、そのキーを次の表の行で CURRVAL を使って参照する作り方は、1行ずつの小さな文を重ねるた

SAMPLE

め、Snowflake の勧める形に反します。同じ作業は、複数表への INSERT（multi-table INSERT）と、入れ子のサブクエリの中でシーケンスを参照する形で、まとめて行えます。[23]

パラメーターの種類と階層

パラメーターは、アカウント、ユーザーのセッション、オブジェクトの動作を制御する設定です。すべてに既定値があり、種類によって設定できる階層が違います。[24]

種類	設定できる階層	例
アカウントパラメーター	アカウントだけ（ALTER ACCOUNT）	ネットワークの設定、ユーザーへの直接の権限付与の禁止など
セッションパラメーター	アカウント → ユーザー → セッション	TIMEZONE、AUTOCOMMIT、STATEMENT_TIMEOUT_IN_SECONDS など
オブジェクトパラメーター	アカウント → 各オブジェクト（ウェアハウス、データベース、スキーマ、テーブルなど）	DATA_RETENTION_TIME_IN_DAYS など

セッションパラメーターは、アカウントに設定した値が全ユーザーの既定になります。ALTER USER で個別のユーザーが上書きし、ALTER SESSION でそのセッションだけを上書きします。オブジェクトパラメーターは、アカウントに設定した値がアカウント内のオブジェクトの既定になり、CREATE や ALTER で個別のオブジェクトが上書きします。データベース、スキーマ、テーブルは入れ子なので、データベースに設定した値は中のスキーマとテーブルの既定になり、下位で別の値を決めれば、そちらが優先されます。[24]

SAMPLE

パラメーターの優先順位

下の階層で設定した値が、上の階層の値を上書きします。アカウント、ユーザー、セッションのすべてに値がある場合、効くのはセッションの値です。[24]

例外は、STATEMENT_TIMEOUT_IN_SECONDS と STATEMENT_QUEUED_TIMEOUT_IN_SECONDS の2つです。この2つは、セッションの階層に加えて、ウェアハウスにも個別に設定できます。両方に設定がある場合は、2つの値のうち、0を除いた小さい方が効きます。ユーザーが10、セッションが20、ウェアハウスが15なら、セッションの階層ではセッションの20が勝ち、その20とウェアハウスの15のうち小さい15が、実際の上限です。ユーザーの10は、セッションに上書きされているので考慮されません。[25]

STATEMENT_TIMEOUT_IN_SECONDS は、文の待ち時間、実行時間、コンパイル時間などを全部合わせた時間に効きます。値は0から604800（7日）で、既定は172800（2日）です。0 を指定すると、最大のタイムアウト値が適用されます。[25]

パラメーターの確認は SHOW PARAMETERS で行います。何も付けないと、セッションパラメーターだけが出ます。アカウントのパラメーターを含めて全部見るには IN ACCOUNT を付け、個別のオブジェクトは IN WAREHOUSE mywh や IN DATABASE mydb のように指します。名前の絞り込みの LIKE は、IN より前に書きます。[24]

```
SHOW PARAMETERS;
SHOW PARAMETERS IN WAREHOUSE mywh;
SHOW PARAMETERS LIKE 'time%' IN ACCOUNT;
```

SAMPLE

アカウントだけに設定できる真偽値のパラメーターには、次のものがあります。[24]

パラメーター	働き	既定
ALLOW_BIND_VALUES_ACCESS	BIND_VALUES 表関数、QUERY_HISTORY の ACCOUNT_USAGE ビューと ORGANIZATION_USAGE ビュー、QUERY_HISTORY 関数で、バインド変数の値を取れるようにする	TRUE
ALLOW_CLIENT_MFA_CACHING	MFA のトークンをクライアント側の OS のキーストアに保存し、接続のたびの MFA の確認を省く	FALSE
ALLOW_ID_TOKEN	接続のトークンを OS のキーストアに保存し、ブラウザーの SSO で、接続のたびの認証を省く	FALSE
ENABLE_DUAL_STACK_LOAD_BALANCER	IPv4 と IPv6 の両方で接続できるエンドポイントにする。AWS のデプロイだけで使える	FALSE

ALLOW_ID_TOKEN を FALSE にしても、SSO で Snowflake に入れなくなるわけではなく、接続のたびに認証を求められるだけです。IPv6 のアドレスでアクセスを絞るには、ネットワークルールを TYPE = IPV6 で作り、ネットワークポリシーに加えます。IPv4 と IPv6 の両方でつなぐネットワークには、アドレスの種類ごとに別のネットワークルールを作ります。[24]

SAMPLE

アカウントとユーザーの階層に設定できるパラメーターもあります。ENABLE_UNREDACTED_QUERY_SYNTAX_ERROR と ENABLE_UNREDACTED_SECURE_OBJECT_ERROR で、どちらも既定は FALSE（隠す）です。設定できるのは、AUDIT 権限を持つか継承するロールだけです。ALTER USER で TRUE にする時は、エラーを起こしたユーザーではなく、内容を見たいユーザーを変えます。[24]

セッション変数とコンテキスト関数

セッション変数は、セッションが続く間だけ有効な、一時的な値です。SET で定義し、使う時は名前の前に `$` を付けます。UNSET で消し、SHOW VARIABLES で一覧を見ます。変数名は大文字小文字を区別しません。文字列とバイナリの変数は、16KBまでです。[26]

```
SET (lo, hi) = (40, 70);
SELECT AVG(salary) FROM emp WHERE age BETWEEN $lo AND $hi;
SET tbl = 'orders';
SELECT * FROM IDENTIFIER($tbl);
```

テーブル名のような識別子として変数を使う時は、IDENTIFIER() で包みます。JDBC ドライバーでは、接続のプロパティに `$変数名` を渡して、接続時に変数を定義することもできます。[26]

セッション変数は、リテラルの定数が置ける場所なら、どこでも使えます（文書に例外の記載がある場所を除きます）。FROM 句では、`TABLE($my_table_name)` のように TABLE() で包む書き方もできます。変数はほかのセッションからは見えず、セッションを閉じると消えます。[26]

SAMPLE

セッションをまたいで残り、複数のセッションで共有する値が要る時は、スキーマ変数（schema variables）を使います。[26]

`$` の構文を使えないツールのために、SYS_CONTEXT、SESSION_CONTEXT、GETVARIABLE と、それぞれに対応する設定側の関数があります。どれも、変数の値を文字列で受け取り、返します。変数は大文字で保存されるため、`SET var_artist_name = 'Jackson Browne';` の後に `GETVARIABLE('var_artist_name')` と小文字で呼ぶと、結果は NULL です。大文字の `'VAR_ARTIST_NAME'` で呼べば、値が返ります。[26]

セッションの文脈を読む関数が、コンテキスト関数です。CURRENT_ROLE、CURRENT_USER、CURRENT_WAREHOUSE、CURRENT_DATABASE、CURRENT_SCHEMA などが、いまのセッションで有効な値を返します。文脈そのものを切り替えるのが USE コマンドで、USE ROLE、USE WAREHOUSE、USE DATABASE、USE SCHEMA があります。ロールを指定せず、ユーザーに既定のロールも無い場合は、PUBLIC ロールが使われます。[26]

仮想ウェアハウスの構成 [1.4]

仮想ウェアハウスは、クエリと DML（データのロードを含む）を実行する計算資源のクラスターです。種類（Standard か Snowpark 最適化）、サイズ、自動停止などの設定で、性格が決まります。サイズを上げるか、クラスターを足すか、ウェアハウスを分けるかを、場面から選ぶ問題が多く出ます。[27]

サイズとクレジット

サイズは、ウェアハウスの1クラスターあたりの計算資源の量を決めます。X-Small が1時間1クレジットで、1段上がるごとに倍になります。次の数字は、第1世代（Gen1）の標準ウェアハウスのものです。[27]

SAMPLE

サイズ	1時間あたりのクレジット
X-Small	1
Small	2
Medium	4
Large	8
X-Large	16
2X-Large	32
3X-Large	64
4X-Large	128
5X-Large	256
6X-Large	512

CREATE WAREHOUSE で作る時の既定のサイズは X-Small です（Snowsight で作る時も同じです）。SQL では、XSMALL、SMALL、MEDIUM、LARGE、XLARGE、XXLARGE、XXXLARGE、X4LARGE、X5LARGE、X6LARGE を使います。ハイフンの入った '2X-LARGE' のような形は、単一引用符で囲みます。5X-Large と 6X-Large は、AWS と Azure のリージョンで使えます。[27][28]

マルチクラスターウェアハウスでは、1時間のクレジットの上限が、サイズのクレジット数に最大クラスター数を掛けた値です。Medium（4クレジット）で最大3クラスターなら、1時間で最大12クレジットになります。[29]

課金の単位とサイズ変更の動き

クレジットは秒単位で請求されます。ただし、ウェアハウスを起動または再開するたびに、最低1分（60秒）分が請求されます。1分を過ぎた後は、動き続けて

SAMPLE

いる限り秒単位です。止まっているウェアハウスは、クレジットを使いません。[30]

X-Small が30秒で止まると60秒分の0.017クレジット、61秒動いたなら61秒分が請求されます。61秒動いて止まり、すぐ再開して59秒で止めると、60+1+60の121秒分になります。起動のたびに最低料金がかかるため、停止と再開を細かく繰り返すと割高です。Large（1時間8クレジット）が10分動いたなら、8×(10/60)で約1.33クレジットです。[30]

動いているウェアハウスでも、サイズはいつでも変えられます。サイズを上げても、すでに実行中のクエリには効きません。新しく用意された計算資源は、待っているクエリと、これから来るクエリだけが使います。大きいサイズに上げた時は、増えた分の計算資源について、1分分が請求されます。Small（2クレジット）から Medium（4クレジット）なら、追加の2クレジットの1分分が加わります。[30]

サイズを下げると、取り除いた計算資源に結びついたキャッシュも消えます。停止した時と同じく、直後のクエリが遅くなることがあります。5X-Large か 6X-Large から 4X-Large 以下へ変える時は、古い資源が片づくまでの短い間、新旧の両方に課金されます。マルチクラスターウェアハウスのサイズを変えると、新しいサイズは、動いているクラスターにも、あとから起動するクラスターにも適用されます。[31]

Standard の Gen 1 と Gen 2

標準ウェアハウスには世代があります。Gen2 は、より速いハードウェアと、削除、更新、マージ、テーブルスキャンなどの最適化を備えた次世代版で、分析とデータエンジニアリングの高速化を狙っています。世代は CREATE WAREHOUSE か ALTER WAREHOUSE で指定し、方法は2通りです。[32]

SAMPLE

指定	Gen1	Gen2
GENERATION 句（推奨）	`GENERATION = '1'`	`GENERATION = '2'`
RESOURCE_CONSTRAINT 句	STANDARD_GEN_1	STANDARD_GEN_2

```
CREATE WAREHOUSE etl_wh WAREHOUSE_SIZE = XLARGE GENERATION = '2';
ALTER WAREHOUSE etl_wh SET GENERATION = '1';
```

GENERATION の値は、引用符で囲む必要があります。GENERATION 句と STANDARD_GEN_1、STANDARD_GEN_2 は、Snowsight では選べず、SQL で指定します。GENERATION と RESOURCE_CONSTRAINT の両方を書いて食い違うと、エラーになります。[32]

制約を押さえます。5X-Large と 6X-Large には、Gen2 がありません。Gen2 は標準ウェアハウスだけの機能で、Snowpark 最適化ウェアハウスには使えません。GENERATION を省いた時にどちらの世代になるかは、公式文書の頁によって書き方が違います（Gen2 の頁は「リージョンが対応していれば Gen2」、ウェアハウスの概要の頁は「今は既定ではない」）。世代を決めたい時は、GENERATION を明示します。AWS の EU（チューリッヒ）とアフリカ（ケープタウン）、Google Cloud の Middle East Central2（ダンマーム）、Azure の US Gov Virginia などは、Gen2 に対応していません。アカウントのレプリケーションを使うなら、複製先のリージョンも Gen2 に対応している必要があります。[32]

世代の切り替えは、動いている間も停止中もできます。動いている Gen1 を Gen2 に変えると、実行中のクエリは Gen1 の資源で終わるまで走り、新しいクエリは Gen2 の資源で走ります。その間は両方の資源に課金され、ウェアハ

SAMPLE

ウスは自動で停止しません。可用性を優先するなら動いたまま変え、費用を抑えたいなら、停止してクエリが無い時に変えます。設定は SHOW WAREHOUSES の resource_constraint 列に出ますが、INFORMATION_SCHEMA のウェアハウスのビューには出ません。[32]

Snowpark 最適化ウェアハウス

Snowpark 最適化ウェアハウスは、1ノードあたりのメモリと CPU アーキテクチャを選べるウェアハウスです。既定の構成では、ノードあたりのメモリが標準ウェアハウスの16倍になります。大きなメモリが要る Snowpark のコードや、特定の CPU アーキテクチャに依存するコードに勧められます。単一のウェアハウスノード上のストアドプロシージャで、機械学習の学習を行う場面が典型です。Snowpark を使わない処理は、ほとんど恩恵がありません。作成や再開は、標準より時間がかかることがあります。[33]

```
CREATE WAREHOUSE so_wh WITH
  WAREHOUSE_SIZE = 'MEDIUM'
  WAREHOUSE_TYPE = 'SNOWPARK-OPTIMIZED';
```

ハイフンが入るので、'SNOWPARK-OPTIMIZED' は単一引用符で囲みます。既定のサイズは MEDIUM、既定のメモリ構成は RESOURCE_CONSTRAINT = MEMORY_16X です。メモリを変えるには、RESOURCE_CONSTRAINT を指定します。

メモリ（最大）	RESOURCE_CONSTRAINT の値	最小のサイズ
16GB	MEMORY_1X、MEMORY_1X_x86	XSMALL

SAMPLE

メモリ（最大）	RESOURCE_CONSTRAINT の値	最小のサイズ
256GB	MEMORY_16X、MEMORY_16X_x86	M
1TB	MEMORY_64X、MEMORY_64X_x86	L

1TB の構成は AWS だけで、プレビュー機能です。Snowpark 最適化ウェアハウスは、Gen2 の形では用意されていません。標準と Snowpark 最適化の切り替えは、動いている間も停止中もでき、Gen2 の切り替えと同じく、実行中のクエリは元の資源で終わります。[33]

マルチクラスターウェアハウス

マルチクラスターウェアハウスは、複数のクラスターで同時実行を捌くウェアハウスで、Enterprise Edition 以上の機能です。最大クラスター数は1より大きく、最小は最大以下にします。CREATE WAREHOUSE の MAX_CLUSTER_COUNT と MIN_CLUSTER_COUNT で決め、どちらの既定値も1（単一クラスター）です。最大のクラスター数は、サイズが大きいほど低くなります。[29]

サイズ	最大クラスター数
XSMALL、SMALL、MEDIUM	300
LARGE	160
XLARGE	80
2XLARGE	40
3XLARGE	20

SAMPLE

サイズ	最大クラスター数
4XLARGE、5XLARGE、6XLARGE	10

Snowsight で選べる最大クラスター数は10までで、上の表の値は SQL の CREATE と ALTER だけに適用されます。

動かし方は2つのモードに分かれます。

- **最大化（Maximized）モード**: 最小と最大に、同じ値（1より大きい）を指定する。起動すると全クラスターが動き、動いている間は常に最大の資源が使える。同時接続数があまり変わらない負荷に向く。
- **自動スケール（Auto-scale）モード**: 最小と最大に、違う値を指定する。クエリが待たされ始めるとクラスターを足し、負荷が下がるとクラスターを止める。

自動スケールモードでは、最大を2か3、最小を1にして始め、負荷の変動を見ながら広げていくのが勧められています。[29]

Medium（1クラスター4クレジット）で最大3クラスターの自動スケールが、2時間動いたとします。1時間目はクラスター1だけで4クレジット、2時間目はクラスター1と2がずっと動き、クラスター3が30分動いて4+4+2の10クレジットになり、合計は14クレジットです。最大化モードで3クラスターが2時間動けば、12×2の24クレジットです。[29]

マルチクラスターは、ユーザー数とクエリ数の増減に合わせて、同時実行を増やすためのものです。1本ごとの遅いクエリや、データのロードの性能を上げる用途には、効果が薄く、そうした場合はサイズを上げます。自動停止は、ウェアハウス全体にだけ効き、個々のクラスターには効きません。最小のクラスター数が動いていて、指定の時間アクティビティが無い時に限り、停止します。自動

SAMPLE

再開は、ウェアハウス全体が停止している（動いているクラスターが無い）時だけ働きます。[29]

スケーリングポリシー

スケーリングポリシーは、自動スケールモードでだけ効きます。最大化モードは全クラスターが同時に動いているので、起動と停止の判断が要りません。SCALING_POLICY の値は STANDARD（既定）と ECONOMY です。かつてあった Legacy は廃止され、使っていたウェアハウスは Standard になっています。[29]

	Standard	Economy
優先するもの	待ちを防ぐ	費用を抑える
クラスターを足す条件	クエリが待たされた時、または動いているクラスターでは余力が足りないと見積もった時	追加のクラスターが、少なくとも6分間は忙しいと見積もった時だけ
クラスターを止める条件	負荷の低い状態が続いた後、最も負荷の低いクラスターのクエリが終わってから	最も負荷の低いクラスターの残りの仕事が6分未満と見積もった時
起こりやすいこと	待ちが少ない。クラスターが多く動き、クレジットが増える	動いているクラスターを満杯で使う。クエリが待たされて遅くなることがある

最大クラスター数が10以下なら、Standard は追加のクラスターを1つずつ起動します。10を超えるなら、急な増加に対応するため、複数を一度に起動します。

SAMPLE

```
CREATE WAREHOUSE mywh WITH MAX_CLUSTER_COUNT = 2, SCALING_POLICY =
'STANDARD';
ALTER WAREHOUSE mywh SET SCALING_POLICY = 'ECONOMY';
```

スケールアップとスケールアウトの使い分け

ウェアハウスを大きくする方法は2つあります。サイズを変えるのがスケールアップ、クラスターを足すのがスケールアウトです。[31]

症状	選ぶ手	理由
大きく複雑な1本のクエリが遅い	サイズを上げる（スケールアップ）	クエリの性能は、ウェアハウスのサイズにほぼ比例して伸びる
同時に使う人が増えて、クエリが待たされる	クラスターを足す（マルチクラスター）	同じサイズのまま、同時に処理できる数が増える
Standard Edition で同時実行が詰まる	ウェアハウスを追加し、クエリを振り分ける	マルチクラスターは Enterprise 以上
小さく単純なクエリが遅い	サイズを上げても変わらないことが多い	大きいほど速いとは限らない
データのロードが遅い	ファイルの数と大きさを見直す	ロードの性能は、ウェアハウスの大きさよりファイルの数と大きさで決まる

サイズを変える方法は、同時実行の問題にも限定的に効きますが、主な目的は1本ごとの性能です。自動の同時実行対応が欲しいなら、マルチクラスターウェアハウスが勧められています。課金は秒単位なので、大きいサイズで一気に処理し、使い終えたら止める運用でも、費用が増えるとは限りません。クエ

SAMPLE

リの時間がサイズにほぼ比例して縮むなら、総クレジットはほぼ同じで済み、待ち時間だけが短くなります。[31]

自動一時停止と自動再開

AUTO_SUSPEND は、何もしない時間が続いた後でウェアハウスを止める設定で、AUTO_RESUME は、そのウェアハウスに文が送られると自動で起動する設定です。どちらも既定で有効です。SQL で AUTO_SUSPEND を指定する時の既定は600秒（10分）で、0 か NULL にすると止まらなくなります。停止を判断する処理は約30秒ごとに動くので、秒単位の細かい制御には向きません。AUTO_RESUME が FALSE のウェアハウスは、ALTER WAREHOUSE か画面で明示的に再開するまで動きません。[34][28]

停止すると、ウェアハウスが持っていたテーブルデータのキャッシュは消えます。ウェアハウスは動いている間、読んだデータをキャッシュし、同じデータを読む後続のクエリを速くします。キャッシュの大きさは、計算資源に比例します。停止して節約するクレジットと、キャッシュを持ち続けて得る速度は、引き換えです。[35]

自動停止は、ワークロードの空き時間に合わせて、5分か10分以下の低い値にするのが勧められています。2〜3分おきにクエリが来るのに、自動停止を1〜2分にすると、停止と再開を繰り返し、再開のたびに最低60秒分が請求されるので、かえって割高です。重く安定した負荷が続く場合や、遅れなく使えることが必須の場合は、自動停止を無効にする選択もあります（起動は通常1〜2秒ですが、サイズと資源の空き具合で伸びます）。自動再開を無効にして手動で再開する運用は、費用とアクセスを管理者が絞りたい場合に向きます。[35]

SAMPLE

用途別の構成とチームごとの分離

用途	構成の目安
アドホックなクエリ	自動停止は短く、自動再開は有効。サイズは試して合わせ、小さく単純なクエリには大きなサイズを使わない
データロード	ファイルの数と大きさに合わせる。何百何千ものファイルを同時に取り込むのでなければ、Small から Large で足りる。大きくしても速くならず、クレジットだけ増えることがある
BI とレポート	同時接続が多い。マルチクラスター（Enterprise 以上）で同時実行を捌く。キャッシュが効くので、自動停止は短くしすぎない
ML の学習、大きなメモリが要る処理	Snowpark 最適化ウェアハウス

チームやワークロードごとにウェアハウスを分けると、互いの負荷が影響せず、使用量もウェアハウスごとに分けて見られます。複雑さの違うクエリを同じウェアハウスで混ぜると、負荷を分析しにくく、適切なサイズも選びにくくなります。似た複雑さで、似たデータのクエリを、同じウェアハウスで処理するのが勧められています。[31]

SAMPLE

ウェアハウスの操作

```
CREATE WAREHOUSE bi_wh
  WAREHOUSE_SIZE = 'MEDIUM'
  MIN_CLUSTER_COUNT = 1
  MAX_CLUSTER_COUNT = 4
  SCALING_POLICY = 'STANDARD'
  AUTO_SUSPEND = 300
  AUTO_RESUME = TRUE
  INITIALLY_SUSPENDED = TRUE
  STATEMENT_TIMEOUT_IN_SECONDS = 3600;

ALTER WAREHOUSE bi_wh SET WAREHOUSE_SIZE = 'LARGE';
ALTER WAREHOUSE bi_wh SUSPEND;
ALTER WAREHOUSE bi_wh RESUME;
SHOW WAREHOUSES;
```

INITIALLY_SUSPENDED を TRUE にすると作成直後は停止状態、FALSE（既定）なら作成後すぐに動き出してクレジットを使い始めます。RESOURCE_MONITOR には、ウェアハウスに結びつけるリソースモニターの名前を指定します。STATEMENT_TIMEOUT_IN_SECONDS は、実行中の文をシステムが取り消すまでの秒数です。STATEMENT_QUEUED_TIMEOUT_IN_SECONDS は、文がウェアハウスで待てる秒数で、どちらもウェアハウスに個別に設定できます。[25]

SHOW WAREHOUSES の出力には、min_cluster_count、max_cluster_count、started_clusters の列があります。マルチクラスターの設定と、いま起動しているクラスターの数が、これで分かります。[29]

SAMPLE

ストレージの概念 [1.5]

Snowflake のテーブルは、物理的にはマイクロパーティションに分かれ、論理的にはテーブルの種類で性格が決まります。

マイクロパーティションと刈り込み

Snowflake のテーブルのデータは、自動でマイクロパーティションに分けられます。1つのマイクロパーティションには、圧縮する前の大きさで50MBから500MBのデータが入ります（実際の保存は圧縮されているので、もっと小さくなります）。行のまとまりが1つのマイクロパーティションに対応し、その中は列ごとに独立して並び、列ごとに圧縮されます。静的なパーティションと違い、ユーザーは定義も保守もしません。挿入やロードの順に従って、自動で区切られていきます。[36]

マイクロパーティションごとに、Snowflake はメタデータを持ちます。列ごとの値の範囲、異なる値の数、最適化に使う追加の情報です。このメタデータを使い、クエリの実行時に、必要のないマイクロパーティションを読まずに飛ばします。これが刈り込み（プルーニング）で、まず不要なものを除き、残ったものの中でも、クエリが参照する列だけを読みます。[36]

1年分の日付と時間の列を持つ大きなテーブルで、特定の1時間を絞り込めば、理想的には全体の1/8760のマイクロパーティションだけを読みます。選ばれるデータの割合と、実際に読むマイクロパーティションの割合が近いほど、刈り込みが効いています。サブクエリを含む条件では、サブクエリの結果が定数でも、刈り込みは行われません。[36]

DML もメタデータを使います。テーブルの全行の削除のような操作は、メタデータだけの操作で済みます。行を更新、削除、追加すると、新しいマイクロパー

SAMPLE

ティションが作られます。列を削除しても、その列を含むマイクロパーティションはすぐには書き直されず、データはストレージに残ります。[36]

データのクラスタリングと深さ

テーブルのデータは、日付や地域のような自然な軸の順に並んでいると、クエリが速くなります。この並び具合がクラスタリングです。Snowflake は、データを挿入するたびに、各マイクロパーティションのクラスタリングのメタデータを記録します。そこから分かる状態は、テーブルを構成するマイクロパーティションの総数、指定した列で値の範囲が重なり合うマイクロパーティションの数、重なりの深さの3つです。マイクロパーティションどうしで、値の範囲が重ならないとは限りません。[36]

クラスタリングの深さは、指定した列について、重なったマイクロパーティションの平均の深さです。1以上の値で、小さいほど、その列についてよくクラスタリングされています。空のテーブルは深さが0です。深さは絶対的な物差しではありません。クエリの性能が期待どおりならよくクラスタリングされており、時間とともに遅くなっているなら、クラスタリングが崩れている可能性があります。[36]

確認には、2つのシステム関数を使います。

```
SELECT SYSTEM$CLUSTERING_INFORMATION('orders', '(order_date)');
SELECT SYSTEM$CLUSTERING_DEPTH('orders', '(order_date)');
```

SYSTEM$CLUSTERING_INFORMATION は、深さを含む詳しい情報を返し、SYSTEM$CLUSTERING_DEPTH は深さを返します。クラスタリングキーを定義しているテーブルなら、列の引数を省けます。キーの無いテーブルや、キー以

SAMPLE

外の列を調べたい時は、列を渡します。列が 1 つでも、'(order_date)' のように括弧で囲む必要があり、括弧は省けません。[3637]

深さを改善する手段が、大きなテーブルへの明示的なクラスタリングキーの設定です。

```
CREATE TABLE sales (sale_date DATE, region STRING, amount NUMBER)
  CLUSTER BY (sale_date, region);
ALTER TABLE sales CLUSTER BY (TO_DATE(sale_ts));
ALTER TABLE sales DROP CLUSTERING KEY;
```

クラスタリングが向くのは、次の3つにすべて当てはまるテーブルです。マイクロパーティションが非常に多く、通常は数テラバイト以上あること。クエリが選択的（行のごく一部だけを読む）か、並べ替えを行うこと。多くのクエリが、同じ少数の列で絞り込むか並べ替えることです。更新が少なく、問い合わせが多いテーブルほど、費用の釣り合いが取りやすくなります。[38]

キーに入れる列は、選択的な絞り込みに最もよく使う列を優先します。日付ベースのクエリが多い事実テーブルなら、日付の列が定番です。次に、結合の条件によく使う列を考えます。1つのキーには、多くのテーブルで3、4列までが勧められ、それ以上は、効果より費用が増えやすくなります。[38]

列の異なる値の数（カーディナリティ）も大切です。極端に少ない列（真偽値のフラグなど）は、刈り込みの効果が小さくなります。極端に多い列（秒まで入ったタイムスタンプなど）は、そのままではキーに向きません。後者は、日付に変換する式などで、カーディナリティを下げます。CLUSTER BY に複数の列を書く順番は、カーディナリティの低い列から高い列へ並べるのが原則です。[38]

SAMPLE

再クラスタリングは自動（Automatic Clustering）で行われ、保守は要りません。ただし、再クラスタリングはクレジットを消費します。新しいマイクロパーティションが作られ、元のものは、削除済みになっても Time Travel と Fail-safe のために残るので、ストレージの費用も増えます。クラスタリングキーを付ける前に、この2つの費用を考えます。[38]

テーブルの種類: 永続・一時・Transient

CREATE TABLE で作るテーブルの既定は永続（permanent）です。保持する必要のないデータ向けに、一時（temporary）と Transient があります。Time Travel と Fail-safe の日数の違いが、最も問われます。[39]

種類	存続する期間	Time Travel の日数	Fail-safe の日数	複製（クローン）できる先
一時（Temporary）	作ったセッションの間だけ	0 または 1（既定 1）	0	一時、Transient
Transient	明示的に消すまで	0 または 1（既定 1）	0	一時、Transient
永続（Standard Edition）	明示的に消すまで	0 または 1（既定 1）	7	一時、Transient、永続
永続（Enterprise 以上）	明示的に消すまで	0 から 90（既定 1）	7	一時、Transient、永続

要点は2つです。1つ目は、一時と Transient には Fail-safe が無いこと、2つ目は、Transient と一時の Time Travel が、Enterprise 以上でも最大1日であることです。90日まで延ばせるのは、永続のテーブル、スキーマ、データベースだ

SAMPLE

けです。Fail-safe は7日で、設定を変えられず、Time Travel が終わった直後から始まります。[40]

永続から一時や Transient へのクローンは作れますが、Transient や一時から永続へのクローンは作れません。作った後に、テーブルの種類を別の種類へ変えることもできません。保護が不要で再作成できる中間データには、Transient が向きます。Fail-safe の費用がかからないので、ストレージ料金を抑えられます。永続テーブルを複製して Transient を作った場合、作成の時点では元のマイクロパーティションを共有するので、追加のストレージは使いません。[39]

一時テーブルの性質

一時テーブルは、作ったセッションの間だけ存在し、セッションが終わるとデータは消えて、Snowflake も復旧できません。他のユーザーや他のセッションからは見えません。作る時に、スキーマの CREATE TABLE 権限は要りません。

```
CREATE TEMPORARY TABLE mytemptable (id NUMBER, creation_date DATE);
```

同じスキーマに同じ名前の永続テーブルがあっても、一時テーブルは作れます。そのセッションの中では、一時テーブルが優先され、同名の永続テーブルが見えなくなります。先に一時テーブルがあるところへ同名のテーブルを作っても、新しいテーブルは一時テーブルに隠され、そのセッションでの操作は、すべて一時テーブルに向かいます。CREATE OR REPLACE は、実質的にテーブルを削除して作り直すので、この隠れ方に特に注意します。[39]

一時テーブルのデータも、存在する間はストレージ料金の対象なので、24時間を超えて続くセッションで大きな一時テーブルを作ったら、不要になった時点で DROP します。大量の一時テーブルを作ると、Information Schema の

SAMPLE

COLUMNS ビューと TABLES ビューへの問い合わせが遅くなるので、セッション終了前の DROP が勧められています。ステージなど他のオブジェクトも一時として作れ、同じようにセッションの間だけ続きます。ストアドプロシージャの中では、1回の実行の間だけ存在する、プロシージャ単位の一時テーブルも作れます。[39]

Transient のデータベース、スキーマ、テーブル

Transient テーブルは、明示的に消すまで残り、必要な権限のある全員が使えます。一時テーブルと違ってセッションをまたぎ、永続テーブルと違って Fail-safe がありません。TRANSIENT を付けて作ります。

```
CREATE TRANSIENT TABLE mytranstable (id NUMBER, creation_date DATE);
CREATE TRANSIENT SCHEMA staging;
CREATE TRANSIENT DATABASE scratch_db;
```

Transient のスキーマの中で作ったテーブルは、すべて Transient になります。Transient のデータベースの中で作ったスキーマも、すべて Transient です。[39]

Apache Iceberg テーブル

Iceberg テーブルは、オープンな表形式 Apache Iceberg に従うテーブルです。Snowflake が持つ性能とクエリの意味を、ユーザーが管理するクラウドストレージのデータに適用します。既存のデータレイクを Snowflake の外に置いたままにしたい場合に向きます。データは Parquet 形式で、ACID トランザクション、スキーマ進化、隠れたパーティション、スナップショットを備えます。[41]

データの置き場には、2つの選択肢があります。

SAMPLE

置き場	データを置く場所	費用と保護
外部ボリューム	ユーザーが管理する S3、Google Cloud Storage、Azure Storage	ストレージ代はクラウド事業者が請求する。Fail-safe は無く、保護と復旧はユーザーの責任
Snowflake のストレージ（EXTERNAL_VOLUME = SNOWFLAKE_MANAGED）	Snowflake が管理する	Snowflake が請求する。永続テーブルは Fail-safe で保護される

外部ボリュームは、外部のクラウドストレージへつなぐ、アカウントレベルの名前つきオブジェクトです。クラウドの IAM の実体を持ち、1つの外部ボリュームを複数の Iceberg テーブルで使えます。[41]

カタログは、テーブルの現在のメタデータの場所を保持し、アトミックな更新を行う層です。選択肢は2つで、Snowflake をカタログにする（Snowflake 管理）か、AWS Glue などの外部のカタログを使うかです。外部のカタログを使う場合は、カタログ統合（catalog integration）という、名前つきのアカウントレベルのオブジェクトを作り、Snowflake Open Catalog との連携にも使います。Snowflake 管理のテーブルは、Snowflake Open Catalog に同期して、他の計算エンジンから問い合わせられます。[41]

Snowflake は、Iceberg テーブルのウェアハウスの計算とクラウドサービスの費用を請求します。外部ボリュームを使っている場合、ストレージ代は請求しません。外部ボリュームが Snowflake のアカウントと別のリージョンにあると、問い合わせの際に、クラウド側で転送の費用（egress）が発生します。[41]

外部テーブル

外部テーブルは、外部ステージにあるファイルを、Snowflake の中のテーブルのように問い合わせる機能です。外部ステージは Snowflake の外にあり、

SAMPLE

Snowflake は保存も管理もしません。COPY INTO が扱える形式のうち、XML 以外を読めます。[42]

- 読み取り専用で、DML はできません。問い合わせと結合、外部テーブルに対するビューの作成はできます。
- すべての外部テーブルが VARIANT 型の VALUE 列を持ち、外部ファイルの1行を表します。
- METADATA$FILENAME（ステージ内のパスを含むファイル名）と、METADATA$FILE_ROW_NUMBER（ファイル内の行番号）の疑似列があります。
- スキーマを知らなくても作れます（スキーマオンリード）。
- パーティション列を使って分割し、読む量を絞る

スキーマが分かっているなら、VALUE 列から式で取り出した仮想列を足し、型の検査を効かせられます。問い合わせは、Snowflake に保存したテーブルより遅くなることがあります。速くするには、外部テーブルの上にマテリアライズドビューを作ります。Parquet のデータなら、Iceberg テーブルが勧められています。[42]

ファイルのメタデータの更新が、運用の要です。作成した時点で、自動で1回更新されます。その後は、ステージに新しいファイルが届いた時に、クラウドのイベント通知で自動更新するか、ALTER EXTERNAL TABLE ... REFRESH で手動更新します。自動更新には、Snowpipe として請求される管理の費用がかかります。ユーザーが定義したパーティションの外部テーブルは、自動更新に対応していません。この場合は、所有者が ALTER EXTERNAL TABLE ... ADD PARTITION でパーティションを手動で追加します。手動の REFRESH もエラーになります。[42]

SAMPLE

ダイナミックテーブル

ダイナミックテーブルは、SELECT クエリの結果を実体として持ち、最新に保ちます。クエリと目標の遅れ（TARGET_LAG）を指定すると、Snowflake が依存関係を追跡し、予定に従って更新します。[43]

```
CREATE OR ALTER DYNAMIC TABLE dt_orders
  TARGET_LAG = '10 minutes'
  WAREHOUSE = transform_wh
  REFRESH_MODE = INCREMENTAL
AS
  SELECT order_id, customer_id, order_date, quantity * unit_price AS
line_total
  FROM raw_orders
  WHERE order_status != 'returned';
```

- **TARGET_LAG**: データが元のテーブルより、最大でどれだけ遅れてよいかの目標。10分なら、その遅れ以内に保とうとする。更新に時間がかかると、実際の遅れは目標を超えることがある。中間のテーブルには TARGET_LAG = DOWNSTREAM を指定でき、下流のテーブルが新しいデータを必要とした時だけ更新される。
- **WAREHOUSE**: 更新のクエリを実行するウェアハウス。
- **REFRESH_MODE**: INCREMENTAL は、前回の更新から変わった行だけを計算する。FULL は結果全体を更新する。AUTO は、作成時に増分にできる定義かどうかを見て、Snowflake が選ぶ。ほかに ADAPTIVE と CUSTOM_INCREMENTAL がある。

SAMPLE

結果は不可分に適用されるので、読み手が更新途中の状態を見ることはありません。複数のダイナミックテーブルが互いを参照すると、Snowflake は依存関係のグラフを読み取り、依存の順に、整合したスナップショットで更新します。依存関係の宣言も、実行順の記述も要りません。SELECT で書ける変換なら、ダイナミックテーブルの候補になります。[43]

費用は3種類です。更新のたびにウェアハウスが使うクレジット、依存関係の追跡などにかかるクラウドサービス（遅れの目標が短いほど増える）、更新で増減するマイクロパーティションのストレージです。[43]

ビューの種類

	標準ビュー	マテリアライズドビュー	セキュアビュー
結果の保存	しない（毎回クエリを実行）	する（事前に計算して保存）	標準でもマテリアライズドでも作れる
エディション	すべて	Enterprise 以上	すべて
主な目的	クエリの簡略化、行と列の公開範囲の制御	繰り返し使う集計の高速化	定義と内部の隠蔽、データ共有
費用と注意	なし	あり（背景のサービスが更新）	最適化の一部を使わないので、遅くなることがある

標準ビューはデータをキャッシュしないので、ビュー自体では速くなりません。ただし、行や列の公開範囲を絞る用途には使えます。[44]

SAMPLE

マテリアライズドビューの制約

マテリアライズドビューは、Enterprise Edition 以上で使えます。元のテーブルが変わると、背景のサービスが自動で更新します。この保守にはクレジットがかかります（サーバーレスの費用で、`MATERIALIZED_VIEW_MAINTENANCE` として記録されます）。更新は、元のテーブルへの挿入、更新、削除を反映します。更新が遅れていても、結果は常に最新です。古い部分は読み飛ばし、元のテーブルからデータを取るからです。遅れが大きいと、問い合わせは遅くなります。[44]

結果を保存するので、保守の費用のほかに、ストレージの費用もかかります。[44]

作る時の制約は次のとおりです。

- 問い合わせられるテーブルは1つだけで、結合（自己結合を含む）はできない。
- 他のマテリアライズドビュー、標準ビュー、ハイブリッドテーブル、ダイナミックテーブル、UDTF は、元にできない。
- UDF（外部関数を含む）、ウィンドウ関数、HAVING、ORDER BY、LIMIT は使えない。
- GROUP BY のキーは SELECT リストに入れる。GROUPING SETS、ROLLUP、CUBE、サブクエリの入れ子、MINUS、EXCEPT、INTERSECT も使えない。
- 使える集計関数は、AVG、COUNT、COUNT_IF、MAX、MIN、SUM、STDDEV 系、VARIANCE 系、APPROX_COUNT_DISTINCT、BITAND_AGG などの BIT 系。使えないものが多い。
- 集計関数の入れ子と、DISTINCT との組み合わせはできない。

SAMPLE

使う時の制約もあります。ビューそのものへの INSERT などの DML はできず、TRUNCATE もできません。CREATE MATERIALIZED VIEW ... CLONE でクローンすることも、Time Travel で過去の時点を問い合わせることもできません。ただし、マテリアライズドビューを含むスキーマやデータベースを複製すると、ビューも複製されます。保守を一時停止（ALTER MATERIALIZED VIEW ... SUSPEND）すると、再開するまでそのビューを問い合わせられません。元のテーブルの列を変更または削除してビューが無効になった場合は、RESUME できず、作り直します。保守の費用は、リソースモニターで制御できません。[44]

最新の更新の状況は、SHOW MATERIALIZED VIEWS の REFRESHED_ON、COMPACTED_ON、BEHIND_BY の列で分かります。使い所は、結果が元より大幅に小さく、元のテーブルの更新が少なく、問い合わせが多い場合です。複数のテーブルを結合する変換や、制約に当てはまらない処理には、ダイナミックテーブルを使います。[44]

セキュアビュー

セキュアビューは、CREATE VIEW や CREATE MATERIALIZED VIEW に SECURE を付けて作り、ALTER VIEW で SECURE を付けたり外したりもできます。

```
CREATE SECURE VIEW v_public_orders AS
  SELECT order_id, region FROM orders;
ALTER VIEW v_public_orders UNSET SECURE;
ALTER VIEW v_public_orders SET SECURE;
SHOW VIEWS LIKE 'v_public_orders';
```

セキュアビューが必要になる理由の1つ目は、標準ビューの内部の最適化が、見えないはずのデータをにじませる可能性があることです。最適化は元のテ

SAMPLE

ーブルのデータに触れるので、ユーザー定義関数のようなコードを通して、見せたくないデータが漏れ得ます。赤い商品だけ見せるビューに `WHERE 1/IFF(color = 'Purple', 0, 1) = 1` を付けて問い合わせると、紫の商品があればゼロ除算のエラーが出るので、存在が分かります。セキュアビューは、この種の最適化を使いません。[45]

2つ目の理由は、標準ビューの定義が、他のユーザーから見えることです。セキュアビューの定義と詳細は、ビューを所有するロールを持つ、権限のあるユーザーにしか見えません。SHOW VIEWS、GET_DDL、Information Schema の VIEWS では、権限の無いユーザーに定義が出ません。[45]

引き換えに、セキュアビューは標準ビューより遅くなることがあります。データの秘匿が目的のビューに使い、問い合わせを簡単にするだけの目的には使いません。データ共有で、共有するデータを絞るビューも、セキュアビューにします。セキュアかどうかは、標準のビューなら SHOW VIEWS の IS_SECURE 列で分かります（Information Schema と Account Usage の VIEWS にも同じ列があります）。マテリアライズドビューなら、SHOW MATERIALIZED VIEWS の IS_SECURE 列で確かめます。[45]

データ型と制約

主なデータ型は次のとおりです。[46]

分類	型	覚える点
数値	NUMBER、DECIMAL、NUMERIC	既定の精度と位取りは (38,0)。DECIMAL と NUMERIC は NUMBER の同義
	INT、INTEGER、BIGINT など	NUMBER の同義。精度と位取りは指定できない

SAMPLE

分類	型	覚える点
	FLOAT、DOUBLE、REAL	浮動小数点。DOUBLE として保存される
文字列	VARCHAR、STRING、TEXT	既定の長さは16777216バイト、最大は134217728バイト
	CHAR、CHARACTER	VARCHAR の同義。既定の長さは VARCHAR(1)
日付と時刻	DATE、TIME、TIMESTAMP	TIMESTAMP は既定で TIMESTAMP_NTZ（TIMESTAMP_TYPE_MAPPING で変えられる）。DATETIME は TIMESTAMP_NTZ の同義
	TIMESTAMP_LTZ、TIMESTAMP_NTZ、TIMESTAMP_TZ	現地時間帯付き、時間帯なし、時間帯付き
半構造化	VARIANT、OBJECT、ARRAY	JSON などをそのまま持つ
地理空間	GEOGRAPHY、GEOMETRY	地球上の座標、平面の座標
ベクトル	VECTOR	埋め込み（AI の検索や類似度）用

標準のテーブルでは、PRIMARY KEY、UNIQUE、FOREIGN KEY を宣言できますが、強制されません。情報の提供とモデリングが目的で、巨大な分散の列指向テーブルで検査するとロードと DML が遅くなるからです。強制されるのは NOT NULL と CHECK で、NULL を NOT NULL の列に入れようとすると、エラーになります。重複した主キーの行は、そのまま入ります。ハイブリッドテーブルは、主キーが必須で、一意制約と外部キー制約も強制されます。[47]

SAMPLE

AI/ML とアプリケーション開発の機能 [1.6]

Snowflake の AI と ML の機能は、データを外へ出さずに Snowflake の中で動かせる点が共通しています。[48]

機能	目的
Snowflake Notebooks	分析の開発の画面
Streamlit in Snowflake	データのアプリ
Snowpark	コードでのデータ処理
Cortex	生成 AI（AI SQL 関数、Cortex Search、Cortex Analyst）
Snowflake ML	機械学習の流れ

Snowflake Notebooks

Snowflake Notebooks は、Snowsight の中の、Python、SQL、Markdown のセルを順に実行する開発の画面です。Snowflake にあるデータで、探索的な分析、機械学習モデルの開発、データエンジニアリングを行います。結果の可視化には、組み込みの Streamlit のほか、Altair、Matplotlib、seaborn が使えます。Git と連携し、スケジュール実行でパイプラインを自動にでき、同じロールを持つ他のユーザーと、ノートブックを見て協力できます。[49]

実行環境は2つあります。

	ウェアハウスランタイム	コンテナランタイム
計算	カーネルがノートブックのウェアハウスで動く	カーネルが Snowpark Container Services のコンピュートプールのノードで動く

SAMPLE

	ウェアハウスランタイム	コンテナランタイム
追加のライブラリ	Snowflake の Anaconda か、ステージから入れる	pip、conda、ステージから入れる
向く用途	すぐ始められる。一般提供のウェアハウス環境	柔軟。CPU と GPU の版があり、機械学習と深層学習向けのライブラリが入っている

どちらのランタイムでも、SQL と Snowpark のクエリは、ウェアハウスで実行されます。従来のノートブックは Legacy Notebooks と呼び名が変わり、次世代の Notebooks in Workspaces へ移行が進んでいます。2026年4月より後に作ったアカウントは、Legacy Notebooks を作れません。[49]

Streamlit in Snowflake

Streamlit は、機械学習やデータサイエンスの Web アプリを作るための、オープンソースの Python ライブラリです。Streamlit in Snowflake は、その Python のアプリを、データを動かさずに、Snowflake の中で作り、動かし、共有します。計算とストレージは Snowflake が管理し、ソースコードと環境の設定は、Snowflake のオブジェクトの中に保存されます。実行環境は、ウェアハウスかコンテナーを選べ、Snowpark、UDF、ストアドプロシージャ、Native App Framework と連携します。[50]

アプリはスキーマレベルのオブジェクトで、既定では所有者の権限で動きます。所有者のウェアハウスを使い、アプリを作ったデータベースとスキーマを使います。呼び出した人の権限や、呼び出した人の現在のデータベースは使いません。ストアドプロシージャの所有者の権限と同じ考え方で、サービスアカウントのトークンが要りません。[50]

SAMPLE

共有は、所有者がアプリを使えるロールを決めて行います。アプリを見る人は、画面に出るものを見て、操作できます。共有されたロールのユーザーは、所有者のロールの権限を、アプリを通して使えます（WITH GRANT が付いていない権限も同じです）。権限の広い所有者のロールでアプリを作ると、閲覧者に見せる範囲も広くなります。[51]

Snowpark

Snowpark は、Java、Python、Scala のライブラリで、Snowflake の中でデータを処理するプログラムを書けます。データを外のシステムへ移さず、処理は Snowflake のウェアハウスで動きます。Spark のような別のクラスターは要りません。Spark コネクタと比べると、Snowpark は UDF を含むすべての操作を Snowflake へプッシュダウンします。コードは、ローカルの Jupyter、VS Code、IntelliJ などで書けます。[48]

中心になる抽象は DataFrame です。DataFrame の操作は、サーバーで遅延して実行されます。`select` や `filter` などの変換を並べても、その時点では何も実行されません。`collect` や `show` のような処理（アクション）を呼んだ時に、SQL の文が Snowflake へ送られて、はじめて実行されます。結果の行だけがクライアントへ返るので、データの移動が少なくて済みます。[48]

```
from snowflake.snowpark.functions import col
df = session.table("sample_product_data").select(col("id"),
col("name"))
rows = df.collect()
```

2行目で DataFrame を作った時点では問い合わせは走らず、3行目の `collect` で実行されます。

SAMPLE

Snowpark の UDF とストアドプロシージャ、パッケージ

Snowpark のコードは、UDF やストアドプロシージャとして、Snowflake の中へ登録できます。Python の UDF では、Snowflake の Anaconda チャネルにある、サードパーティのパッケージを使えます。Python ワークシートで作る UDF では Anaconda のパッケージがすでに使える状態で、ローカルの開発環境では `session.add_packages` でセッションに追加します（ウェアハウスでの実行時に、Anaconda のパッケージが使われます）。Artifact Repository のパッケージも使えます。[52] Python のストアドプロシージャも、作る時に入れる Anaconda のパッケージを指定できます。[53]

UDF を SQL の式から呼ぶ点と、ストアドプロシージャを CALL で呼ぶ点は、Snowpark でも同じです。大きなメモリが要る処理は、先に見た Snowpark 最適化ウェアハウスで動かします。[52]

Cortex の AI SQL 関数

Cortex AI 関数は、OpenAI、Anthropic、Meta、Mistral AI、DeepSeek、xAI などの LLM で、文章と画像の分析を SQL から行います。LLM は、Snowflake のサービスの境界の内側に配備されています。利用できるリージョンは限られ、関数ごとにプレビューか一般提供かが違います。[54]

関数	役割
AI_COMPLETE	選んだ LLM で、文字列や画像に対する生成（補完）を行う。ほとんどの生成 AI の用途にはこれを使う
AI_CLASSIFY	文章や画像を、ユーザーが決めた分類に分ける
AI_FILTER	条件に合うかを真偽で返す。SELECT、WHERE、JOIN の ON で使える

SAMPLE

関数	役割
AI_AGG	列の多くの行をまとめて、プロンプトに従った洞察を返す。コンテキストの大きさの制限を受けない
AI_SUMMARIZE_AGG	多くの行をまとめた要約を返す。同じく制限を受けない
AI_SUMMARIZE	文章、画像、文書を要約する
AI_SENTIMENT	文章の感情を取り出す
AI_EXTRACT	文字列やファイルから情報を取り出す
AI_TRANSLATE（旧 SNOWFLAKE.CORTEX.TRANSLATE）	言語間の翻訳
AI_EMBED、AI_SIMILARITY	埋め込みベクトルの作成、2つの入力の類似度
AI_PARSE_DOCUMENT	ステージの文書から、OCR か、レイアウト付きでテキストを取り出す
AI_TRANSCRIBE	音声と動画を文字に起こす
AI_REDACT	個人を特定できる情報（PII）を伏せる

SAMPLE

補助の関数として、ステージのファイルを指す TO_FILE、入力のトークン数を返す AI_COUNT_TOKENS、プロンプトを組み立てる PROMPT があります。[54]

使い分けは、出力の形で決めます。

やりたいこと	使う関数
分類に分ける	AI_CLASSIFY
行を条件で絞る	AI_FILTER

やりたいこと	使う関数
行ごとに自由な文章を生成する	AI_COMPLETE
多くの行を1つにまとめる	AI_AGG、AI_SUMMARIZE_AGG
行ごとに要約する	AI_SUMMARIZE
翻訳する	AI_TRANSLATE

AI 関数は、大量の行を処理する用途に向きます。対話のように遅延が重要な用途には、REST API を使います。[54]

AI 関数の費用、モデル、必要なロール

費用は処理したトークンの数で決まり、トークンは文字のおよそ4文字分が目安です。AI_COMPLETE、AI_CLASSIFY、AI_FILTER、AI_AGG、AI_SUMMARIZE、AI_TRANSLATE は、入力と出力の両方のトークンが課金されます。AI_EMBED と AI_SIMILARITY は、入力だけが数えられます。AI_PARSE_DOCUMENT は、処理したページ数で課金されます。[55]

AI_CLASSIFY のラベルや説明は、1回の呼び出しではなく、処理した行ごとに入力として数えられます。AI_CLASSIFY、AI_FILTER、AI_AGG などは、裏でプロンプトを足すので、渡した文章より課金されるトークンが多くなります。AI_COUNT_TOKENS は、関数を動かす計算の費用だけで、トークンの追加の課金はありません。AI 関数を呼ぶクエリの間は、ウェアハウスの費用も引き続きかかりますが、MEDIUM より大きなウェアハウスは勧められません。大きくしても性能は上がらず、無駄な費用になります。費用は、ACCOUNT_USAGE の CORTEX_FUNCTIONS_USAGE_HISTORY ビューで見られます。METERING_DAILY_HISTORY の SERVICE_TYPE = 'AI_SERVICES' でも分かります。[55]

SAMPLE

AI 関数を呼ぶには、2つの条件が要ります。

- アカウントレベルの USE AI FUNCTIONS 権限（関数ごとの USE AI FUNCTION 名前でも可）
- SNOWFLAKE データベースの CORTEX_USER か AI_FUNCTIONS_USER のデータベースロール

USE AI FUNCTIONS は既定で PUBLIC ロールに付与され、CORTEX_USER も既定で PUBLIC に付与されているので、既定ではアカウントの全員が使えます。[56]

範囲を絞るなら、ACCOUNTADMIN が PUBLIC から取り消し、特定のロールへ付与します。CORTEX_USER は、ユーザーに直接付与できないので、アカウントロールを作ってそこへ付与し、そのロールをユーザーに渡します。USE AI FUNCTIONS があっても CORTEX_USER が無いユーザーは、AI_AGG と AI_SUMMARIZE_AGG だけは使えます。[56]

```
REVOKE USE AI FUNCTIONS ON ACCOUNT FROM ROLE PUBLIC;
GRANT USE AI FUNCTIONS ON ACCOUNT TO ROLE analyst;
GRANT DATABASE ROLE SNOWFLAKE.CORTEX_USER TO ROLE analyst;
```

自分のリージョンで処理できないモデルの呼び出しは、アカウントパラメーター CORTEX_ENABLED_CROSS_REGION で、別のリージョンでの処理を許可できます。[57]

Cortex Search

Cortex Search は、Snowflake のデータに対する、低遅延で質の高い「曖昧な」検索を提供します。ベクトルとキーワードのハイブリッド検索エンジンを、

SAMPLE

埋め込み、基盤の保守、検索品質の調整、索引の更新を気にせずに、短時間で用意できます。主な用途は2つで、LLM のチャットボットの RAG（検索拡張生成）の検索エンジンと、アプリの検索バーの裏側です。RAG では、Cortex Search が LLM に文脈を渡す検索役になり、答えが最新の自社データに基づきます。[58]

作り方は、SQL の CREATE CORTEX SEARCH SERVICE か、Snowsight の AI & ML の画面です。

```
CREATE OR REPLACE CORTEX SEARCH SERVICE transcript_search_service
  ON transcript_text
  ATTRIBUTES region
  WAREHOUSE = cortex_search_wh
  TARGET_LAG = '1 day'
  EMBEDDING_MODEL = 'snowflake-arctic-embed-l-v2.0'
  AS (SELECT transcript_text, region, agent_id FROM
support_transcripts);
```

句	意味
ON	検索する列
ATTRIBUTES	問い合わせの時に絞り込み（フィルター）に使える列。元のクエリの列に含める
WAREHOUSE	最初の作成と更新で、元のクエリを実行するウェアハウス
TARGET_LAG	元のテーブルの変更を確かめる頻度の目安
EMBEDDING_MODEL	埋め込みに使うモデル

SAMPLE

作成すると索引も作られるため、データが大きいと CREATE に数時間かかることもあります。サービスごとの専用のウェアハウス（MEDIUM より大きくないもの）が勧められ、画面で作る場合は、ロールに SNOWFLAKE.CORTEX_USER が要ります。SQL では SNOWFLAKE.CORTEX.SEARCH_PREVIEW などで問い合わせ、アプリからは REST API を使います。[58]

費用は5つの形で発生します。更新のためのウェアハウスの計算、列を埋め込む時のトークン、サービスを提供し続ける計算（ユーザーのウェアハウスとは別で、索引にしたデータの GB あたり月で請求されます）、元のクエリを実体化した表と索引の保存（TB あたりの定額）、変更の検知に使うクラウドサービスの計算です。複数の列に索引を付けるサービスは、列が多く、埋め込みが大きいほど費用が増えます。使っていない時の提供の計算の費用は、AUTO_SUSPEND（プレビュー機能。最小 1800 秒）を設定して抑えられます。[58]

Cortex Analyst

Cortex Analyst は、構造化データに対する自然言語の質問に答える、LLM を使った完全マネージドの機能です。業務の担当者が SQL を書かずに質問し、答えを得られます。質問から SQL を生成し（text-to-SQL）、その SQL を Snowflake のエンジンで実行します。Cortex Analyst は RBAC に従い、生成した SQL の実行も、確立されたアクセス制御に従います。権限の無いデータは返りません。既定では Snowflake が用意した LLM で動き、メタデータやプロンプトを含むデータは Snowflake のガバナンスの境界から出ません。顧客のデータでモデルを学習しません。REST API として提供され、Streamlit、Slack、Teams、独自のチャット画面などに組み込めます。[59]

SAMPLE

精度のかなめが、セマンティックモデルです。スキーマだけでは、業務の定義（粗利の意味など）が分からないので、LLM が正確な SQL を作りにくくなります。Cortex Analyst は、軽量な YAML のファイルで書いたセマンティックモデルで、業務の用語とデータベースの間を橋渡しします。現在は、同じ役割をデータベースのオブジェクトとして持つ、セマンティックビューが使われます。[59]

セマンティックビューは、スキーマレベルのオブジェクトで、CREATE SEMANTIC VIEW で作ります。作るロールには、スキーマの CREATE SEMANTIC VIEW と、定義に使うテーブルとビューの SELECT が要ります。[60] 定義するのは次のものです。

- 論理テーブル
- 論理テーブルの間の関係
- 事実（行レベルの属性）
- ディメンション
- メトリクス（業務の指標）

「純売上」の定義を1か所にまとめれば、どの BI やアプリでも同じ集計になります。セマンティックビューは SELECT でも問い合わせられ、リスティングで共有できます。Snowflake は、Cortex Analyst の全機能を備え、回答品質がより高い Cortex Agents への移行を勧めています。[61]

Snowflake ML

Snowflake ML は、Snowflake のガバナンスの効いたデータの上で、機械学習を端から端まで行う機能の集まりです。主な部品は次のとおりです。[62]

SAMPLE

部品	役割
特徴量ストア（Feature Store）	データから作る特徴量を、定義、保存、管理し、見つけて再利用する。バッチとストリーミングから増分更新できる
モデルレジストリ（Model Registry）	学習したモデルを、Snowflake の外で学習したものも含めて登録し、版を管理し、大規模な推論に使う
ML 関数	時系列の予測（FORECAST）と異常検知（ANOMALY_DETECTION）、分類（CLASSIFICATION）、重要な要因を見つける TOP INSIGHTS
コンテナランタイム	CPU と GPU での学習の環境。PyTorch、XGBoost、scikit-learn が入っている
ML ジョブ、実験、可観測性、リネージ	パイプラインの自動化、学習結果の記録、本番モデルの監視、元のデータからモデルまでの経路の追跡

ML 関数は、機械学習の専門知識なしに、予測と洞察を得る機能です。使う前に、セッションの AUTOCOMMIT が有効である必要があります（既定で有効です）。時系列の関数は、指標（売上など）が時間やほかの特徴とともにどう変わるかを学習します。使うと、ML モデルのインスタンスの保存の費用と、学習と予測の計算の費用が発生します。使わないモデルを削除して、保存の費用を減らします。Snowpark からは、`session.sql('call my_model!FORECAST(...)').collect()` のように、`session.sql` で呼びます。[63]

特徴量ストアは、特徴量の変換を標準化して中央に置くので、再利用が進み、重複が減り、学習時と推論時の特徴量が整合します。モデルレジストリに登録したモデルは、先に見たとおり、スキーマレベルのオブジェクトです。[62]

SAMPLE

サンプルはここまでです

続き(第2章〜巻末)は、SnowPro Core 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://docs.snowflake.com/en/user-guide/intro-key-concepts#snowflake-architecture ↩ ↩2 ↩3 ↩4
2. https://docs.snowflake.com/en/user-guide/cost-understanding-compute ↩ ↩2 ↩3 ↩4
3. https://docs.snowflake.com/en/user-guide/warehouses-overview ↩ ↩2
4. https://docs.snowflake.com/en/user-guide/warehouses-considerations ↩ ↩2
5. https://docs.snowflake.com/en/user-guide/intro-key-concepts#database-storage ↩
6. https://docs.snowflake.com/en/user-guide/intro-editions#feature-and-edition-matrix ↩ ↩2 ↩3 ↩4
7. https://docs.snowflake.com/en/user-guide/cost-understanding-data-storage ↩
8. https://docs.snowflake.com/en/user-guide/intro-editions#overview-of-editions ↩ ↩2

SAMPLE

9. https://docs.snowflake.com/en/user-guide/intro-regions ↩ ↩2 ↩3 ↩4
10. https://docs.snowflake.com/en/user-guide/ui-snowsight ↩
11. https://docs.snowflake.com/en/developer-guide/snowflake-cli/index ↩ ↩2
12. https://docs.snowflake.com/en/developer-guide/snowflake-cli/introduction ↩
13. https://docs.snowflake.com/en/developer-guide/snowflake-cli/connecting/configure-connections ↩
14. https://docs.snowflake.com/en/user-guide/vscode-ext ↩ ↩2 ↩3 ↩4 ↩5 ↩6
15. https://docs.snowflake.com/en/user-guide/security-access-control-overview ↩
16. https://docs.snowflake.com/en/user-guide/organizations ↩ ↩2 ↩3 ↩4 ↩5 ↩6
17. https://docs.snowflake.com/en/developer-guide/snowflake-ml/model-registry/overview ↩
18. https://docs.snowflake.com/en/developer-guide/udf/udf-overview ↩ ↩2
19. https://docs.snowflake.com/en/sql-reference/external-functions-introduction ↩

SAMPLE

20. https://docs.snowflake.com/en/developer-guide/stored-procedures-vs-udfs ↩ ↩2
21. https://docs.snowflake.com/en/developer-guide/stored-procedure/stored-procedures-rights ↩ ↩2 ↩3 ↩4
22. https://docs.snowflake.com/en/developer-guide/stored-procedure/stored-procedures-overview ↩ ↩2
23. https://docs.snowflake.com/en/user-guide/querying-sequences#sequence-semantics ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7
24. https://docs.snowflake.com/en/sql-reference/parameters#parameter-hierarchy-and-types ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7
25. https://docs.snowflake.com/en/sql-reference/parameters#statement-timeout-in-seconds ↩ ↩2 ↩3
26. https://docs.snowflake.com/en/sql-reference/session-variables ↩ ↩2 ↩3 ↩4 ↩5 ↩6
27. https://docs.snowflake.com/en/user-guide/warehouses-overview#warehouse-size ↩ ↩2 ↩3
28. https://docs.snowflake.com/en/sql-reference/sql/create-warehouse ↩ ↩2
29. https://docs.snowflake.com/en/user-guide/warehouses-multicluster ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7

SAMPLE

30. https://docs.snowflake.com/en/user-guide/cost-understanding-compute ↩ ↩2 ↩3

31. https://docs.snowflake.com/en/user-guide/warehouses-considerations ↩ ↩2 ↩3 ↩4

32. https://docs.snowflake.com/en/user-guide/warehouses-gen2 ↩ ↩2 ↩3 ↩4

33. https://docs.snowflake.com/en/user-guide/warehouses-snowpark-optimized ↩ ↩2

34. https://docs.snowflake.com/en/user-guide/warehouses-overview ↩

35. https://docs.snowflake.com/en/user-guide/warehouses-considerations ↩ ↩2

36. https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7

37. https://docs.snowflake.com/en/sql-reference/functions/system_clustering_information ↩

38. https://docs.snowflake.com/en/user-guide/tables-clustering-keys ↩ ↩2 ↩3 ↩4

39. https://docs.snowflake.com/en/user-guide/tables-temp-transient ↩ ↩2 ↩3 ↩4 ↩5

40. https://docs.snowflake.com/en/user-guide/data-failsafe#what-is-fail-safe ↩

SAMPLE

41. https://docs.snowflake.com/en/user-guide/tables-iceberg ↩ ↩2 ↩3 ↩4

42. https://docs.snowflake.com/en/user-guide/tables-external-intro ↩ ↩2 ↩3

43. https://docs.snowflake.com/en/user-guide/dynamic-tables-about ↩ ↩2 ↩3

44. https://docs.snowflake.com/en/user-guide/views-materialized ↩ ↩2 ↩3 ↩4 ↩5

45. https://docs.snowflake.com/en/user-guide/views-secure ↩ ↩2 ↩3

46. https://docs.snowflake.com/en/sql-reference/intro-summary-data-types ↩

47. https://docs.snowflake.com/en/sql-reference/constraints-overview ↩

48. https://docs.snowflake.com/en/developer-guide/snowpark/index ↩ ↩2 ↩3

49. https://docs.snowflake.com/en/user-guide/ui-snowsight/notebooks ↩ ↩2

50. https://docs.snowflake.com/en/developer-guide/streamlit/about-streamlit ↩ ↩2

51. https://docs.snowflake.com/en/developer-guide/streamlit/object-management/owners-rights ↩

SAMPLE

52. https://docs.snowflake.com/en/developer-guide/snowpark/python/creating-udfs ↩ ↩2

53. https://docs.snowflake.com/en/developer-guide/snowpark/python/creating-sprocs ↩

54. https://docs.snowflake.com/en/user-guide/snowflake-cortex/aisql#available-functions ↩ ↩2 ↩3

55. https://docs.snowflake.com/en/user-guide/snowflake-cortex/aisql-cost ↩ ↩2

56. https://docs.snowflake.com/en/user-guide/snowflake-cortex/aisql-privileges-and-access ↩ ↩2

57. https://docs.snowflake.com/en/sql-reference/parameters#cortex-enabled-cross-region ↩

58. https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-search/cortex-search-overview ↩ ↩2 ↩3

59. https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-analyst ↩ ↩2

60. https://docs.snowflake.com/en/sql-reference/sql/create-semantic-view ↩

61. https://docs.snowflake.com/en/user-guide/views-semantic/overview ↩

62. https://docs.snowflake.com/en/developer-guide/snowflake-ml/overview ↩ ↩2

SAMPLE

63. https://docs.snowflake.com/en/guides-overview-ml-functions ↩

SAMPLE

SnowPro Core Certification（COF-C03） 学習用テキスト(無料サンプル)

版

2026年11月版(Snowflake SnowPro Core Certification（COF-C03）学習ガイド（英語版 2026年7月8日更新・日本語版 2026年2月20日更新）準拠)

発行

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

お問い合わせ

info@shikaku-doujou.com

Snowflake、SnowPro、Snowsight、Snowpark、Snowpipe および Snowflake の製品名・サービス名は、Snowflake Inc. 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。

本教材は公式の学習ガイドの内容を基に作成したもので、Snowflake Inc. 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