MySQL Shell 8.0

概要

MySQL Shell は、MySQL の高度なクライアントおよびコードエディタです。 このドキュメントでは、MySQL Shell のコア機能について説明します。 mysql と同様の提供される SQL 機能に加えて、MySQL Shell は JavaScript および Python のスクリプト機能を提供し、MySQL を操作するための API を備えています。X DevAPI を使用すると、リレーショナルデータとドキュメントデータの両方を操作できます。ドキュメントストアとしての MySQL の使用 を参照してください。AdminAPI を使用すると、InnoDB クラスタ を作業できます。第6章「MySQL AdminAPI の使用」を参照してください。

MySQL Shell 8.0 は、MySQL Server 8.0 および 5.7 とともに使用することを強くお薦めします。 MySQL Shell 8.0 にアップグレードしてください。 MySQL Shell をまだインストールしていない場合は、「ダウンロードサイト」からダウンロードします。

各リリースでの変更の詳細は、「MySQL Shell リリースノート」を参照してください。

MySQL の使用方法のヘルプは、「MySQL フォーラム」を参照してください。この「MySQL フォーラム」では、他の MySQL ユーザーとの問題について説明できます。

ドキュメント生成日: 2022-06-14 (revision: 111)

目次

1 MySQL Shell の機能 1
2 MySQL Shell のインストール 5
2.1 Microsoft Windows への MySQL Shell のインストール 5
2.2 Linux への MySQL Shell のインストール 5
2.3 macOS への MySQL Shell のインストール 7
3 MySQL Shell コマンドの使用 9
3.1 MySQL Shell のコマンド 9
4 MySQL Shell スタートガイド 15
4.1 MySQL Shell の起動 15
4.2 MySQL Shell セッション 15
4.2.1 MySQL Shell 起動時の Session グローバルオブジェクトの作成 16
4.2.2 MySQL Shell の起動後の Session グローバルオブジェクトの作成 17
4.2.3 JavaScript および Python モードでのスクリプトセッション 18
4.3 MySQL Shell 接続 19
4.3.1 個々のパラメータを使用した接続 21
4.3.2 Unix ソケットおよび Windows Named Pipes を使用した接続 22
4.3.3 暗号化された接続の使用 22
4.3.4 圧縮接続の使用 23
4.4 プラガブルパスワードストア 26
4.4.1 プラガブルパスワード構成オプション 27
4.4.2 資格証明の使用 28
4.5 MySQL Shell グローバルオブジェクト 28
4.6 ページャの使用 29
5 MySQL Shell コードの実行 31
5.1 アクティブな言語 31
5.2 対話型コードの実行 32
5.3 コード自動補完 33
5.4 コードの編集 35
5.5 コード履歴 35
5.6 バッチコード実行 36
5.7 出力形式 38
5.7.1 テーブル形式 38
5.7.2 タブ区切り形式 39
5.7.3 垂直フォーマット 39
5.7.4 JSON 形式の出力 40
5.7.5 JSON ラッピング 41
5.7.6 結果メタデータ 42
5.8 API コマンドラインインタフェース 43
6 MySQL AdminAPI の使用 47
6.1 MySQL AdminAPI 47
6.2 MySQL InnoDB クラスタ 54
6.2.1 MySQL InnoDB クラスタ の要件 55
6.2.2 本番 InnoDB クラスタ のデプロイ 56
6.2.3 InnoDB クラスタ の監視 68
6.2.4 インスタンスの操作 77
6.2.5 InnoDB クラスタの操作 79
6.2.6 InnoDB クラスタ の構成 82
6.2.7 InnoDB クラスタ のトラブルシューティング 87
6.2.8 InnoDB クラスタ のアップグレード 91
6.2.9 メタデータのタグ付け 94
6.2.10 InnoDB クラスタ のヒント 97
6.2.11 既知の制限事項 100
6.3 MySQL InnoDB ReplicaSet 101
6.3.1 InnoDB ReplicaSet の概要 101
6.3.2 InnoDB ReplicaSet のデプロイ 102
6.3.3 ReplicaSet へのインスタンスの追加 104
6.3.4 既存のレプリケーション設定の採用 106

6.3.5 InnoDB ReplicaSet の操作 107
6.4 MySQL Router 110
6.4.1 MySQL Router のブートストラップ 110
6.4.2 AdminAPI および MySQL Router の使用 113
6.5 AdminAPI MySQL サンドボックス 115
7 MySQL Shell の拡張 119
7.1 MySQL Shell でのレポート 119
7.1.1 MySQL Shell レポートの作成 120
7.1.2 MySQL Shell レポートの登録 120
7.1.3 MySQL Shell レポートの永続化 122
7.1.4 MySQL Shell レポートの例 122
7.1.5 MySQL Shell レポートの実行 122
7.1.6 組込み MySQL Shell レポート 124
7.2 MySQL Shell への拡張オブジェクトの追加 126
7.2.1 ユーザー定義 MySQL Shell グローバルオブジェクトの作成 126
7.2.2 拡張オブジェクトの作成 127
7.2.3 拡張オブジェクトの永続化 129
7.2.4 MySQL Shell 拡張オブジェクトの例 129
7.3 MySQL Shell プラグイン 130
7.3.1 MySQL Shell プラグインの作成 130
7.3.2 プラグイングループの作成 131
7.3.3 MySQL Shell プラグインの例 132
8 MySQL Shell ユーティリティ 135
8.1 アップグレードチェッカユーティリティ 135
8.2 JSON インポートユーティリティ 141
8.2.1 mysqlsh コマンドインタフェースを使用した JSON ドキュメントのインポート 143
8.2.2 --import コマンドを使用した JSON ドキュメントのインポート 143
8.2.3 BSON データ型の表現の変換 145
8.3 テーブルエクスポートユーティリティ 145
8.4 パラレルテーブルインポートユーティリティ 149
8.5 インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティ 155
8.6 ダンプロードユーティリティ 163
9 MySQL Shell のロギングおよびデバッグ 173
9.1 アプリケーションログ 173
9.2 冗長出力 175
9.3 AdminAPI 操作のロギング 175
10 MySQL Shell のカスタマイズ 177
10.1 起動スクリプトの操作 177
10.2 モジュール検索パスの追加 178
10.2.1 モジュール検索パスの環境変数 179
10.2.2 起動スクリプトのモジュール検索パス変数 179
10.3 プロンプトのカスタマイズ 179
10.4 MySQL Shell オプションの構成 180
A MySQL Shell コマンドリファレンス 185
A.1 mysqlsh — MySQL Shell 185

第 1 章 MySQL Shell の機能

MySQL Shell では、次の機能を使用できます。

サポートされる言語

MySQL Shell は、JavaScript、Python および SQL で記述されたコードを処理します。 実行されたコードは、現在アクティブな言語に基づいて、これらの言語のいずれかとして処理されます。\という接頭辞が付いた特定の MySQL Shell コマンドもあり、現在選択されている言語に関係なく MySQL Shell を構成できます。 詳細は、セクション 3.1「MySQL Shell のコマンド」 を参照してください。

バージョン 8.0.18 から、MySQL Shell では Python 2.7 ではなく Python 3 が使用されます。 システムでサポートされている Python 3 のインストールを含むプラットフォームの場合、MySQL Shell では使用可能な最新バージョンが使用され、サポートされている最小バージョンの Python 3.6 が使用されます。 Python 3 が含まれていないプラットフォームの場合、MySQL Shell には Python 3.7.7 がバンドルされます。MySQL Shell は、Python 2.6 および Python 2.7 とのコード互換性を維持しているため、これらの古いバージョンのいずれかが必要な場合は、適切な Python バージョンを使用してソースから MySQL Shell をビルドできます。

対話型コードの実行

MySQL Shell には対話型コード実行モードが用意されており、ここで MySQL Shell プロンプトにコードを入力すると、入力した各ステートメントが処理され、処理の結果が画面に表示されます。 Unicode テキスト入力は、使用中の端末でサポートされている場合にサポートされます。 カラー端子がサポートされています。

コマンドを使用して複数行コードを記述し、MySQL Shell で複数行をキャッシュして単一のステートメントとして実行できます。 詳細は、複数行のサポート を参照してください。

バッチコード実行

MySQL Shell では、コードの対話型実行に加えて、様々なソースからコードを取得して処理することもできます。 この非対話的な方法でコードを処理する方法は、バッチ実行と呼ばれます。

バッチ実行モードは単一言語のスクリプト処理を目的としているため、フォーマットされていない出力を最小限に抑え、コマンドの実行を無効にすることに制限されます。 これらの制限を回避するには、対話型セッションであるかのように入力を実行するように MySQL Shell に指示する --interactive コマンドラインオプションを使用します。 このモードでは、各行が対話型セッションで入力されたかのように、入力は line by line で処理されます。 詳細は、セクション5.6「バッチコード実行」 を参照してください。

サポートされる API

MySQL Shell には、MySQL と対話するコードの開発に使用できる、JavaScript および Python に実装された次の API が含まれています。

- AdminAPI を使用すると、MySQL インスタンスを管理し、それらを使用して InnoDB クラスタs、InnoDB ReplicaSets を作成し、MySQL Router を統合できます。InnoDB クラスタ は、InnoDB ベースの MySQL データベースを使用して高可用性およびスケーラビリティのための統合ソリューションを提供します。InnoDB クラスタ は、MySQL の高度な専門知識を必要とせずに Group Replication を使用するための代替ソリューションです。 同様に、InnoDB ReplicaSet を使用すると、非同期 GTID ベースのレプリケーションを実行する一連の MySQL インスタンスを管理できます。AdminAPI には、InnoDB クラスタ および InnoDB ReplicaSet との統合をできるだけ簡単にするために、MySQL Router のユーザーを構成する操作も用意されています。 第6章「MySQL AdminAPI の使用」を参照してください。

- X DevAPI を使用すると、MySQL Shell が X プロトコル を使用して MySQL サーバーに接続している場合に、開発者はリレーショナルデータとドキュメントデータの両方を操作できます。 詳細は、ドキュメントストアとしての MySQL の使用を参照してください。 X DevAPI の概念および使用方法のドキュメントは、X DevAPI User Guide を参照してください。

X プロトコル のサポート

MySQL Shell は、X プロトコル をサポートするすべての MySQL 製品に統合コマンドラインクライアントを提供するように設計されています。 MySQL Shell の開発機能は、X プロトコル を使用するセッション用に設計されています。MySQL Shell は、クラシック MySQL プロトコル を使用して、X プロトコル をサポートしていない MySQL Server に接続することもできます。 クラシック MySQL プロトコル を使用して作成されたセッションでは、X DevAPI の最小限の機能セットを使用できます。

拡張機能

MySQL Shell の基本機能に対する拡張機能は、レポートおよび拡張オブジェクトの形式で定義できます。 レポートおよび拡張オブジェクトは、JavaScript または Python を使用して作成でき、アクティブな MySQL Shell 言語に関係なく使用できます。 レポートおよび拡張機能オブジェクトは、MySQL Shell の起動時に自動的にロードされるプラグインに保持できます。MySQL Shell には、使用可能な組込みレポートがいくつか用意されています。 詳しくは第7章「MySQL Shell の拡張」をご覧ください。

ユーティリティ

MySQL Shell には、MySQL を使用するための次のユーティリティが含まれています:

- MySQL サーバーインスタンスのアップグレード準備ができているかどうかを検証するアップグレードチェッカユーティリティ。 util.checkForServerUpgrade() を使用してアップグレードチェッカにアクセスします。
- JSON ドキュメントを MySQL Server コレクションまたはテーブルにインポートする JSON インポートユーティリティ。 util.importJSON() を使用してインポートユーティリティにアクセスします。
- 単一のデータファイルを分割し、複数のスレッドを使用してチャンクを MySQL テーブルにロードするパラレルテーブルインポートユーティリティ。

詳しくは第8章「MySQL Shell ユーティリティ」をご覧ください。

API コマンドライン統合

MySQL Shell では、mysqlsh を他のツールと簡単に統合できる API コマンド構文を使用して、その機能の多くを公開しています。 たとえば、この機能を使用して InnoDB クラスタ を管理する bash スクリプトを作成できます。 REPL インタフェースをバイパスして操作を MySQL Shell グローバルオブジェクトに直接渡すには、mysqlsh [options] -- shell_object object_method [method_arguments]構文を使用します。 セクション5.8「API コマンドラインインタフェース」を参照してください。

出力形式

MySQL Shell は、結果をテーブル、タブ付きまたは垂直形式で、または JSON 出力として返すことができます。 MySQL Shell を外部ツールと統合するために、コマンドラインから MySQL Shell を起動するときに、すべての出力に対して JSON ラッピングをアクティブ化できます。 詳細は、セクション5.7「出力形式」 を参照してください。

ロギングおよびデバッグ

MySQL Shell では、選択した詳細レベルで実行プロセスに関する情報をログに記録できます。 ロギング情報は、アプリケーションログファイル、追加の表示可能な宛先およびコンソールの任意の組合せに送信できます。 詳細は、第9章「MySQL Shell のロギングおよびデバッグ」 を参照してください。

グローバルセッション

MySQL Shell では、MySQL Server インスタンスへの接続はセッションオブジェクトによって処理されます。 MySQL Shell の起動時またはその後に実行できる MySQL Server インスタンスへの最初の接続を行うと、この接続を表す session という名前の MySQL Shell グローバルオブジェクトが作成されます。 このセッションは、すべての MySQL Shell 実行モードで使用できるため、グローバルセッションと呼ばれます。 SQL モードでは、グローバルセッション

はステートメントの実行に使用され、JavaScript モードおよび Python モードでは、session という名前のオブジェクトを介して使用できます。 mysqlx および mysql JavaScript および Python モジュールで使用可能な関数を使用してさらにセッションオブジェクトを作成し、これらのセッションオブジェクトのいずれかを session グローバルオブジェクトとして設定して、任意のモードで使用できます。 詳細は、セクション4.2「MySQL Shell セッション」を参照してください。

第 2 章 MySQL Shell のインストール

目次

2.1 Microsoft Windows への MySQL Shell のインストール 5
2.2 Linux への MySQL Shell のインストール 5
2.3 macOS への MySQL Shell のインストール 7

このセクションでは、MySQL Server の開発および管理をサポートする対話型の JavaScript、Python または SQL インタフェースである MySQL Shell をダウンロード、インストールおよび起動する方法について説明します。MySQL Shell は、個別にインストールできるコンポーネントです。

MySQL Shell は X プロトコル をサポートしており、JavaScript または Python で X DevAPI を使用して、ドキュメントストアとして機能する MySQL Server と通信するアプリケーションを開発できます。 MySQL をドキュメントストアとして使用する方法の詳細は、ドキュメントストアとしての MySQL の使用 を参照してください。

> **重要**
>
> Community および Commercial バージョンの MySQL Shell の場合: MySQL Shell をインストールする前に、Visual Studio 2015 用の Visual C++ Redistributable (Microsoft ダウンロードセンターで入手可能) が Windows システムにインストールされていることを確認してください。

要件

MySQL Shell は、64-bit プラットフォーム用の Microsoft Windows、Linux および macOS で使用できます。

2.1 Microsoft Windows への MySQL Shell のインストール

MSI インストーラを使用して Microsoft Windows に MySQL Shell をインストールするには、次の手順を実行します:

1. http://dev.mysql.com/downloads/shell/ からWindows (x86、64-bit)、MSI インストーラパッケージをダウンロードします。
2. プロンプトが表示されたら、実行をクリックします。
3. セットアップウィザードのステップに従います。

2.2 Linux への MySQL Shell のインストール

> **注記**
>
> MySQL Shell のインストールパッケージは、限られた数の Linux ディストリビューションでのみ使用でき、64-bit システムでのみ使用できます。

サポートされている Linux ディストリビューションの場合、Linux に MySQL Shell をインストールする最も簡単な方法は、「MySQL APT リポジトリ」または「MySQL Yum リポジトリ」を使用することです。 MySQL リポジトリを使用していないシステムでは、MySQL Shell を直接ダウンロードしてインストールすることもできます。

MySQL APT リポジトリを使用した MySQL Shell のインストール

「MySQL APT リポジトリ」でサポートされている Linux ディストリビューションの場合は、次のいずれかのパスに従います:

- システムにソフトウェアリポジトリとして「MySQL APT リポジトリ」がまだない場合は、次の手順を実行します:

- 「MySQL APT リポジトリの追加」に示されているステップに従って、次の点に特に注意してください:
  - 構成パッケージのインストール時に、リポジトリを構成するためのダイアログボックスが表示されたら、必要なリリースシリーズとして MySQL 8.0 を選択してください。
  - MySQL APT リポジトリのパッケージ情報を更新するステップをスキップしないでください:

    ```
    sudo apt-get update
    ```
- 次のコマンドを使用して MySQL Shell をインストールします:

  ```
  sudo apt-get install mysql-shell
  ```
- システムにソフトウェアリポジトリとして「MySQL APT リポジトリ」がすでに存在する場合は、次の手順を実行します:
  - MySQL APT リポジトリのパッケージ情報を更新します:

    ```
    sudo apt-get update
    ```
  - 次のコマンドを使用して MySQL APT リポジトリ構成パッケージを更新します:

    ```
    sudo apt-get install mysql-apt-config
    ```

    リポジトリの構成を求めるダイアログボックスが表示されたら、必要なリリースシリーズとして MySQL 8.0 を選択していることを確認します。
  - 次のコマンドを使用して MySQL Shell をインストールします:

    ```
    sudo apt-get install mysql-shell
    ```

MySQL Yum Repository を使用した MySQL Shell のインストール

「MySQL Yum リポジトリ」でサポートされている Linux ディストリビューションの場合は、次のステップに従って MySQL Shell をインストールします:

- 次のいずれかを実行します:
  - システムにソフトウェアリポジトリとして「MySQL Yum リポジトリ」がすでにあり、リポジトリが新しいリリースパッケージ mysql80-community-release で構成されている場合。
  - すでに「MySQL Yum リポジトリ」をソフトウェアリポジトリとしてシステムに所有していても、古いリリースのパッケージ mysql-community-release でリポジトリを構成している場合は、最初に MySQL Yum リポジトリを新しい mysql80-community-release パッケージで再構成することで、MySQL Shell をインストールするのが最も簡単です。 これを行うには、まず次のコマンドを使用して古いリリースのパッケージを削除する必要があります:

    ```
    sudo yum remove mysql-community-release
    ```

    dnf 対応システムの場合は、かわりに次の手順を実行します:

    ```
    sudo dnf erase mysql-community-release
    ```

    次に、「MySQL Yum リポジトリの追加」に示されているステップに従って、新しいリリースパッケージ mysql80-community-release をインストールします。
  - 「MySQL Yum リポジトリ」をソフトウェアリポジトリとしてシステムにまだ持っていない場合は、「MySQL Yum リポジトリの追加」で説明されているステップに従います。
- 次のコマンドを使用して MySQL Shell をインストールします:

  ```
  sudo yum install mysql-shell
  ```

  dnf 対応システムの場合は、かわりに次の手順を実行します:

  ```
  sudo dnf install mysql-shell
  ```

MySQL Developer ゾーンからの直接ダウンロードからの MySQL Shell のインストール

MySQL Shell をインストールするための RPM、Debian およびソースパッケージは、「MySQL Shell のダウンロード」からもダウンロードできます。

2.3 macOS への MySQL Shell のインストール

macOS に MySQL Shell をインストールするには、次の手順を実行します:

1. http://dev.mysql.com/downloads/shell/ からパッケージをダウンロードします。
2. ダウンロードした DMG をダブルクリックしてマウントします。 ファインダが開きます。
3. ファインダウィンドウに表示されている .pkg ファイルをダブルクリックします。
4. インストールウィザードのステップに従います。
5. インストーラが終了したら、DMG をイジェクトします。 (削除できます。)

第 3 章 MySQL Shell コマンドの使用

目次

3.1 MySQL Shell のコマンド 9

このセクションでは、対話型コードエディタから MySQL Shell を構成するコマンドについて説明します。 コマンドを使用すると、現在使用されている言語に関係なく、MySQL Shell を制御できます。 たとえば、オンラインヘルプの表示、サーバーへの接続、現在使用されている言語の変更、レポートの実行、ユーティリティの使用などを実行できます。 これらのコマンドは、mysqlsh コマンドオプションを使用して構成できる MySQL Shell 設定と似ている場合があります。付録A「MySQL Shell コマンドリファレンス」 を参照してください。

3.1 MySQL Shell のコマンド

MySQL Shell には、アクティブなプログラミング言語や MySQL Server 接続の構成など、コードエディタの実行環境を変更できるコマンドが用意されています。 次のテーブルに、現在選択されている言語に関係なく使用可能なコマンドを示します。 コマンドは実行モードから独立して使用可能である必要があるため、エスケープシーケンス (\文字) で始まります。

コマンド	Alias/Shortcut	説明
\help	\h または\?	MySQL Shell に関するヘルプを出力するか、オンラインヘルプを検索します。
\quit	\q または\exit	MySQL Shell を終了します。
\		SQL モードで、複数行モードを開始します。 空の行が入力されると、コードがキャッシュされて実行されます。
\status	\s	現在の MySQL Shell ステータスを表示します。
\js		実行モードを JavaScript に切り替えます。
\py		実行モードを Python に切り替えます。
\sql		実行モードを SQL に切り替えます。
\connect	\c	MySQL インスタンスに接続します。
\reconnect		同じ MySQL インスタンスに再接続します。
\disconnect		グローバルセッションを切断します。
\use	\u	使用するスキーマを指定します。
\source	\. または source (バックスラッシュなし)	アクティブな言語を使用してスクリプトファイルを実行します。
\warnings	\W	ステートメントによって生成された警告を表示します。
\nowarnings	\w	ステートメントによって生成された警告を表示しません。
\history		コマンドライン履歴を表示および編集します。
\rehash		オートコンプリート名前キャッシュを手動で更新します。

コマンド	Alias/Shortcut	説明
\option		MySQL Shell 構成オプションをクエリーおよび変更します。
\show		指定されたオプションと引数を使用して、指定されたレポートを実行します。
\watch		指定されたオプションと引数を使用して指定されたレポートを実行し、定期的に結果をリフレッシュします。
\edit	\e	デフォルトのシステムエディタでコマンドを開き、MySQL Shell に表示します。
\pager	\P	MySQL Shell がテキストの表示に使用するページャを構成します。
\nopager		MySQL Shell が使用するように構成されたページャを無効にします。
\system	\!	指定したオペレーティングシステムコマンドを実行し、MySQL Shell に結果を表示します。

Help コマンド

\help コマンドは、パラメータの有無にかかわらず使用できます。 パラメータを指定せずに使用すると、使用可能な MySQL Shell コマンド、グローバルオブジェクトおよびメインヘルプカテゴリに関する情報を含む一般的なヘルプメッセージが出力されます。

このパラメータをパラメータとともに使用すると、MySQL Shell が現在実行されているモードに基づいて使用可能なヘルプを検索するために使用されます。 パラメータには、ワード、コマンド、API 関数または SQL ステートメントの一部を指定できます。 次のカテゴリが存在します:

- AdminAPI - dba グローバルオブジェクトおよび AdminAPI の詳細を示します。これにより、InnoDB クラスタ および InnoDB ReplicaSet を使用できます。
- X DevAPI - mysqlx モジュールと、MySQL をドキュメントストアとして使用できる X DevAPI の機能の詳細を示します
- Shell Commands - に、使用可能な組込み MySQL Shell コマンドの詳細を示します。
- ShellAPI - には、shell および util のグローバルオブジェクトと、MySQL Servers で SQL を実行できるようにする mysql モジュールに関する情報が含まれています。
- SQL Syntax - SQL ステートメントの構文ヘルプを取得するエントリポイント。

API 関数など、トピックのヘルプを検索するには、関数名を pattern として使用します。 ワイルドカード文字 ? を使用して単一の文字を照合し、* を使用して検索で複数の文字を照合できます。 ワイルドカード文字は、パターン内で 1 回以上使用できます。 次のネームスペースは、ヘルプの検索時にも使用できます:

- dba for AdminAPI
- mysqlx for X DevAPI
- mysql for ShellAPI for クラシック MySQL プロトコル
- 他の ShellAPI クラス用の shell: Shell, Sys, Options
- MySQL Shell コマンド用の commands
- mysqlsh コマンドインタフェース用の cmdline

たとえば、トピックのヘルプを検索するには、\help pattern を発行し、次のようにします:

- x devapi を使用した X DevAPI のヘルプの検索
- \c を使用して、MySQL Shell \connect コマンドのヘルプを検索
- Cluster または dba.Cluster を使用して、AdminAPI dba.Cluster() 操作のヘルプを検索
- Table または mysqlx.Table を使用して、X DevAPI Table クラスのヘルプを検索
- MySQL Shell が JavaScript モードで実行されている場合は、isView、Table.isView または mysqlx.Table.isView を使用して、Table オブジェクトの isView 関数に関するヘルプを検索
- MySQL Shell が Python モードで実行されている場合は、is_view、Table.is_view または mysqlx.Table.is_view を使用して、Table オブジェクトの isView 関数に関するヘルプを検索
- MySQL Shell が SQL モードで実行されている場合、MySQL サーバーへのグローバルセッションが存在すると、SQL ヘルプが表示されます。 概要では、検索パターンとして sql syntax を使用します。

指定された検索パターンによっては、1 つまたは複数の結果が見つかります。 タイトルに検索パターンが含まれているヘルプトピックが 1 つだけの場合は、そのヘルプトピックが表示されます。 複数のトピックタイトルがパターンと一致するが、1 つが完全一致の場合、そのヘルプトピックが表示され、その後にパターン一致を含む他のトピックのリストがタイトルに表示されます。 完全一致が識別されない場合は、タイトルにパターン一致があるトピックのリストが表示されます。 トピックのリストが返された場合は、関連トピックのタイトルに一致する拡張検索パターンを指定してコマンドを再度入力することで、表示するトピックをリストから選択できます。

接続、再接続および切断コマンド

\connect コマンドは、MySQL Server への接続に使用されます。 セクション4.3「MySQL Shell 接続」を参照してください。

例:

```
\connect root@localhost:3306
```

パスワードが必要な場合は、パスワードの入力を求められます。

--mysqlx (--mx) オプションを使用して、X プロトコル を使用して MySQL サーバーインスタンスに接続するセッションを作成します。 例:

```
\connect --mysqlx root@localhost:33060
```

--mysql (--mc) オプションを使用して ClassicSession を作成すると、クラシック MySQL プロトコル を使用してサーバーで SQL を直接発行できます。 例:

```
\connect --mysql root@localhost:3306
```

短い形式のオプション (-mx および -mc) を使用した単一ダッシュの使用は、MySQL Shell のバージョン 8.0.13 から非推奨になりました。

\reconnect コマンドは、パラメータまたはオプションなしで指定されます。 サーバーへの接続が失われた場合は、\reconnect コマンドを使用できます。これにより、MySQL Shell は既存の接続パラメータを使用してセッションの再接続を複数回試行します。 これらの試行が失敗した場合は、\connect コマンドを使用して接続パラメータを指定することで、新しい接続を作成できます。

MySQL Shell 8.0.22 から使用可能な\disconnect コマンドも、パラメータまたはオプションなしで指定されます。 このコマンドは、現在接続している MySQL サーバーインスタンスから MySQL Shell グローバルセッション (session グローバルオブジェクトで表されるセッション) を切断して、接続をクローズできるようにしますが、引き続き MySQL Shell を使用します。

サーバーへの接続が失われた場合は、\reconnect コマンドを使用できます。これにより、MySQL Shell は既存の接続パラメータを使用してセッションの再接続を複数回試行します。 これらの試行が失敗した場合は、\connect コマンドを使用して接続パラメータを指定することで、新しい接続を作成できます。

ステータスコマンド

\status コマンドは、現在のグローバル接続に関する情報を表示します。 これには、接続されているサーバー、使用中の文字セット、稼働時間などに関する情報が含まれます。

ソースコマンド

\source コマンドまたはそのエイリアス\. を MySQL Shell 対話モードで使用して、特定のパスにあるスクリプトファイルからコードを実行できます。 例:

```
\source /tmp/mydata.sql
```

SQL、JavaScript または Python コードのいずれかを実行できます。 ファイル内のコードはアクティブな言語を使用して実行されるため、SQL コードを処理するには、MySQL Shell が SQL モードである必要があります。

> **警告**
>
> コードはアクティブな言語を使用して実行されるため、現在選択されている実行モード言語とは異なる言語でスクリプトを実行すると、予期しない結果になる可能性があります。

MySQL Shell 8.0.19 からは、mysql クライアントとの互換性のために、SQL モードでのみ、バックスラッシュおよびオプションの SQL デリミタを指定せずに source コマンドを使用してスクリプトファイルからコードを実行できます。source またはエイリアス\. (SQL デリミタを使用しない) は、SQL の MySQL Shell インタラクティブモードでスクリプトを直接実行する場合と、バッチモードで処理された SQL コードのファイルでファイル内からさらにスクリプトを実行する場合の両方で使用できます。 SQL モードの MySQL Shell では、次の 3 つのコマンドのいずれかを使用して、対話モードまたはバッチモードから/tmp/mydata.sql ファイルのスクリプトを実行できるようになりました:

```
source /tmp/mydata.sql;
source /tmp/mydata.sql
\. /tmp/mydata.sql
```

コマンド\source /tmp/mydata.sql も有効ですが、対話型モードでのみ有効です。

対話モードでは、\source、\. または source コマンド自体が MySQL Shell 履歴に追加されますが、実行されたスクリプトファイルの内容は履歴に追加されません。

コマンドの使用

\use コマンドを使用すると、アクティブなスキーマを選択できます。次に例を示します:

```
\use schema_name
```

\use コマンドでは、グローバル開発セッションがアクティブである必要があります。 \ use コマンドは、現在のスキーマを指定された schema_name に設定し、db 変数を選択されたスキーマを表すオブジェクトに更新します。

履歴コマンド

\history コマンドは、MySQL Shell で以前に発行したコマンドをリストします。 \history を発行すると、履歴エントリが発行された順序で履歴エントリが表示されます。履歴エントリ番号は、\history delete entry_number コマンドで使用できます。

\history コマンドには、次のオプションがあります:

- \history save を使用して履歴を手動で保存します。
- \history delete entrynumber を使用して、指定した番号の個々の履歴エントリを削除します。
- \history delete firstnumber-lastnumber を使用して、指定されたエントリ番号の範囲内の履歴エントリを削除します。 lastnumber が最後に見つかった履歴エントリ番号を超過すると、最後のエントリまでの履歴エントリが削除されます。
- \history delete number- を使用して、number から最後のエントリまでの履歴エントリを削除します。
- \history delete -number を使用して、最後のエントリから開始して作業中の指定した数の履歴エントリを削除します。 たとえば、\history delete -10 では、最新の 10 個の履歴エントリが削除されます。
- \history clear を使用して、履歴全体を削除します。

デフォルトでは、履歴はセッション間で保存されないため、MySQL Shell を終了すると、現在のセッション中に発行した内容の履歴が失われます。 セッション間で履歴を保持する場合は、MySQL Shell history.autoSave オプションを有効にします。 詳細は、セクション5.5「コード履歴」を参照してください。

Rehash コマンド

名前キャッシュのオートコンプリート機能を無効にした場合は、\rehash コマンドを使用してキャッシュを手動で更新します。 たとえば、\use schema コマンドを発行して新しいスキーマをロードした後、\rehash を発行してオートコンプリート名キャッシュを更新します。 このオートコンプリートがデータベースで使用されている名前を認識した後、テーブル名などのテキストをオートコンプリートできます。 セクション5.3「コード自動補完」を参照してください。

オプションコマンド

\option コマンドを使用すると、すべてのモードで MySQL Shellconfiguration オプションをクエリーして変更できます。 \option コマンドを使用して、設定されている構成オプションをリストし、その値が最後にどのように変更されたかを表示できます。 また、これを使用して、セッションに対して、または MySQL Shell 構成ファイルで永続的に、オプションを設定および設定解除することもできます。 手順および構成オプションのリストは、セクション10.4「MySQL Shell オプションの構成」 を参照してください。

ページャコマンド

外部ページャを使用して、オンラインヘルプや SQL クエリーの結果など、画面の長い出力を読み取るように MySQL Shell を構成できます。 セクション4.6「ページャの使用」を参照してください。

Show コマンドと Watch コマンド

\show コマンドは、組込み MySQL Shell レポートまたは MySQL Shell に登録されているユーザー定義レポートのいずれかの名前付きレポートを実行します。 コマンドの標準オプションと、レポートでサポートされているオプションまたは追加の引数を指定できます。 \watch コマンドは、\show コマンドと同じ方法でレポートを実行しますが、Ctrl + C を使用してコマンドを取り消すまで定期的に結果をリフレッシュします。 その手順は、セクション7.1.5「MySQL Shell レポートの実行」を参照してください。

コマンドの編集

\edit (\e) コマンドは、デフォルトのシステムエディタで編集用のコマンドを開き、MySQL Shell で編集したコマンドを実行用に表示します。 このコマンドは、キーの組合せ Ctrl-X Ctrl-E を使用して呼び出すこともできます。 詳細は、セクション5.4「コードの編集」を参照してください。

システムコマンド

\system (\!) コマンドは、コマンドの引数として指定したオペレーティングシステムコマンドを実行し、MySQL Shell のコマンドからの出力を表示します。コマンドを実行できなかった場合、MySQL Shell はエラーを返します。 コマンドからの出力は、オペレーティングシステムによって指定されたとおりに返され、出力を表示するように指定した MySQL ShellJSON ラッピング関数または外部ページャツールでは処理されません。

第 4 章 MySQL Shell スタートガイド

目次

4.1 MySQL Shell の起動 15
4.2 MySQL Shell セッション 15
4.2.1 MySQL Shell 起動時の Session グローバルオブジェクトの作成 16
4.2.2 MySQL Shell の起動後の Session グローバルオブジェクトの作成 17
4.2.3 JavaScript および Python モードでのスクリプトセッション 18
4.3 MySQL Shell 接続 19
4.3.1 個々のパラメータを使用した接続 21
4.3.2 Unix ソケットおよび Windows Named Pipes を使用した接続 22
4.3.3 暗号化された接続の使用 22
4.3.4 圧縮接続の使用 23
4.4 プラガブルパスワードストア 26
4.4.1 プラガブルパスワード構成オプション 27
4.4.2 資格証明の使用 28
4.5 MySQL Shell グローバルオブジェクト 28
4.6 ページャの使用 29

このセクションでは、MySQL Shell の開始方法、MySQL サーバーインスタンスへの接続方法およびセッションタイプの選択方法について説明します。

4.1 MySQL Shell の起動

MySQL Shell をインストールすると、mysqlsh コマンドが使用可能になります。 ターミナルウィンドウ (Windows ではコマンドプロンプト) を開き、次を発行して MySQL Shell を起動します:

```
> mysqlsh
```

これにより、デフォルトでは JavaScript モードでサーバーに接続せずに MySQL Shell が開きます。 モードを変更するには、\sql、\py および\js コマンドを使用します。

4.2 MySQL Shell セッション

MySQL Shell では、MySQL Server インスタンスへの接続はセッションオブジェクトによって処理されます。 次のタイプのセッションオブジェクトを使用できます:

- Session : このセッションオブジェクトタイプは、新しいアプリケーション開発で X プロトコル が使用可能な MySQL Server インスタンスと通信するために使用します。X プロトコル は、MySQL Server との最適な統合を提供します。 X プロトコル を使用できるようにするには、MySQL Server インスタンスに X プラグイン をインストールして有効にする必要があります。これは、MySQL 8.0 のデフォルトです。 MySQL 5.7 では、X プラグイン を手動でインストールする必要があります。 詳細は、X プラグイン を参照してください。X プラグインは、mysqlx_port によって指定されたポート (デフォルトは 33060) をリスニングするため、Session を使用した接続でこのポートを指定します。

- ClassicSession : このセッションオブジェクトタイプを使用して、X プロトコル が使用できない MySQL Server インスタンスと対話します。 このオブジェクトは、クラシック MySQL プロトコル を使用してサーバーに対して SQL を実行するためのものです。 この種のセッションで使用可能な開発 API は非常に制限されています。 たとえば、X DevAPI CRUD 操作、コレクション処理およびバインディングはサポートされていません。 開発の場合は、可能なかぎり Session オブジェクトを優先します。

> **重要**
>
> ClassicSession は MySQL Shell に固有であり、MySQL コネクタなどの X DevAPI の他の実装では使用できません。

MySQL Shell の起動時またはその後に実行できる MySQL Server インスタンスへの最初の接続を行うと、この接続を表す session という名前の MySQL Shell グローバルオブジェクトが作成されます。 この特定のセッションオブジェクトは、一度作成されるとすべての MySQL Shell 実行モードで使用できるため、グローバルです: SQL モード、JavaScript モードおよび Python モード。 したがって、それが表す接続はグローバルセッションと呼ばれます。 変数 session は、このセッションオブジェクトへの参照を保持し、JavaScript モードおよび Python モードの MySQL Shell で使用して接続を操作できます。

session グローバルオブジェクトは、MySQL Server インスタンスへの接続時に選択したプロトコルに従って、Session タイプのセッションオブジェクトまたは ClassicSession タイプのセッションオブジェクトのいずれかになります。 コマンドオプションを使用してプロトコルを選択するか、指定した接続データの一部としてセッションオブジェクトタイプを指定できます。 現在のグローバルセッションに関する情報を表示するには、次のコマンドを発行します:

```
mysql-js []> session

<ClassicSession:user@example.com:3330>
```

グローバルセッションが接続されると、セッションオブジェクトタイプおよびグローバルセッションが接続されている MySQL Server インスタンスのアドレスが表示されます。

プロトコルを明示的に選択するか、接続時に暗黙的に指定すると、MySQL Shell はそのプロトコルを使用して接続を作成しようとし、失敗した場合はエラーを返します。 接続パラメータにプロトコルが指定されていない場合、MySQL Shell はまず X プロトコル を使用して接続を試行し (Session タイプのセッションオブジェクトを戻します)、失敗した場合は クラシック MySQL プロトコル を使用して接続を試行します (ClassicSession タイプのセッションオブジェクトを戻します)。

接続試行の結果を確認するには、MySQL Shell\status コマンドまたは shell.status() メソッドを使用します。 これらは、session グローバルオブジェクトによって表される接続に関する接続プロトコルおよびその他の情報を表示します。または、session グローバルオブジェクトが MySQL サーバーに接続されていない場合は、「「未接続」」を返します。 例:

```
mysql-js []> shell.status()
MySQL Shell version 8.0.18

Session type:               X Protocol
Connection Id:              198
Current schema:
Current user:               user@example.com
SSL:                      Cipher in use: TLS_AES_256_GCM_SHA384 TLSv1.3
Using delimiter:            ;
Server version:             8.0.18 MySQL Community Server - GPL
Protocol version:           X Protocol
Client library:           8.0.18
Connection:                 TCP/IP
TCP port:                 33060
Server characterset:        utf8mb4
Schema characterset:        utf8mb4
Client characterset:      utf8mb4
Conn. characterset:       utf8mb4
Compression:              Enabled (zstd)
Uptime:                 31 min 42.0000 sec

Threads: 8  Questions: 2622  Slow queries: 0  Opens: 298  Flush tables: 3  Open tables: 217  Queries per second avg: 1.378
```

このセクションでは、MySQL Server インスタンスへの接続を表すセッションオブジェクトおよび session グローバルオブジェクトについて説明します。 MySQL Server インスタンスに接続するためのこのセクションで説明する各方法の完全な手順と例、および接続に使用できるその他のオプションについては、セクション4.3「MySQL Shell 接続」を参照してください。

4.2.1 MySQL Shell 起動時の Session グローバルオブジェクトの作成

コマンドラインから MySQL Shell を起動する場合、ユーザー名、ホスト、ポートなどの値ごとに個別のコマンドオプションを使用して接続パラメータを指定できます。 この方法で MySQL Shell を起動して MySQL Server インスタンスに接続する手順および例は、セクション4.3.1「個々のパラメータを使用した接続」 を参照してください。 この接続方法を使用する場合は、次のいずれかのオプションを追加して、起動時に作成するセッションオブジェクトのタイプを session グローバルオブジェクトとして選択できます:

- --mysqlx (--mx) は、X プロトコル を使用して MySQL Server インスタンスに接続する Session オブジェクトを作成します。
- --mysql (--mc) は、クラシック MySQL プロトコル を使用して MySQL Server インスタンスに接続する ClassicSession オブジェクトを作成します。

たとえば、次のコマンドは MySQL Shell を起動し、ポート 33060 でリスニングしているローカル MySQL Server インスタンスへの X プロトコル 接続を確立します:

```
shell> mysqlsh --mysqlx -u user -h localhost -P 33060
```

MySQL Shell を SQL モードで起動する場合、--sqlx および --sqlc オプションにはセッションオブジェクトタイプの選択肢が含まれるため、かわりにこれらのいずれかを指定して、MySQL Shell で接続に X プロトコル または クラシック MySQL プロトコル を使用できます。 すべての mysqlsh コマンドラインオプションのリファレンスは、セクション A.1「mysqlsh — MySQL Shell」 を参照してください。

個々のオプションを使用して接続パラメータを指定するかわりに、URI のような接続文字列を使用して指定することもできます。 この文字列は、オプションの --uri コマンドオプションを使用するかどうかにかかわらず、コマンドラインから MySQL Shell を起動するときに渡すことができます。 この接続方法を使用する場合、URI のような接続文字列の先頭に scheme 要素を含めて、作成するセッションオブジェクトのタイプを選択できます。mysqlx は X プロトコルを使用して Session オブジェクトを作成するか、mysql は クラシック MySQL プロトコル を使用して ClassicSession オブジェクトを作成します。 たとえば、次のいずれのコマンドも URI のような接続文字列を使用して MySQL Shell を起動し、ポート 3306 でリスニングしているローカル MySQL Server インスタンスへの クラシック MySQL プロトコル 接続を作成します:

```
shell> mysqlsh --uri mysql://user@localhost:3306
shell> mysqlsh mysql://user@localhost:3306
```

次の例のように、URI のような接続文字列の一部としてではなく、オプションとして接続プロトコルを指定することもできます:

```
shell> mysqlsh --mysql --uri user@localhost:3306
```

この方法で MySQL Server インスタンスに接続する手順および例は、URI 類似文字列またはキーと値のペアを使用したサーバーへの接続 を参照してください。

接続プロトコルを省略して、他の接続パラメータに基づいて MySQL Shell で自動的に検出されるようにできます。 たとえば、ポート 33060 を指定し、接続プロトコルを示すオプションがない場合、MySQL Shell は X プロトコル を使用して接続を試行します。 接続パラメータにプロトコルが指定されていない場合、MySQL Shell はまず X プロトコルを使用して接続を試行し、失敗した場合は クラシック MySQL プロトコル を使用して接続を試行します。

4.2.2 MySQL Shell の起動後の Session グローバルオブジェクトの作成

MySQL Server インスタンスに接続せずに MySQL Shell を起動した場合は、MySQL Shell \connect コマンドまたは shell.connect() メソッドを使用して接続を開始し、session グローバルオブジェクトを作成できます。 または、shell.getSession() メソッドは session グローバルオブジェクトを返します。

MySQL Shell\connect コマンドは、前述および URI 類似文字列またはキーと値のペアを使用したサーバーへの接続 で説明されているように、URI のような接続文字列とともに使用されます。 URI のような接続文字列の先頭に scheme 要素を含めて、作成するセッションオブジェクトのタイプを選択できます。次に例を示します:

```
mysql-js> \connect mysqlx://user@localhost:33060
```

または、scheme 要素を省略し、--mysqlx (--mx) オプションを使用して X プロトコル または --mysql (--mc) を使用して Session オブジェクトを作成して、クラシック MySQL プロトコル を使用して ClassicSession オブジェクトを作成できます。 例:

```
mysql-js> \connect --mysqlx user@localhost:33060
```

shell.connect() メソッドは、\connect コマンドのかわりに MySQL Shell で session グローバルオブジェクトを作成するために使用できます。 この接続方法では、選択したプロトコルが scheme 要素として指定された URI のような接続文字列を使用できます。 例:

```
mysql-js> shell.connect('mysqlx://user@localhost:33060')
```

shell.connect() メソッドでは、JavaScript で JSON オブジェクトとして、または Python でディクショナリとして提供されるキーと値のペアを使用して、接続パラメータを指定することもできます。 選択したプロトコル (mysqlx または mysql) が scheme キーの値として指定されます。 例:

```
mysql-js> shell.connect( {scheme:'mysqlx', user:'user', host:'localhost', port:33060} )
```

これらの方法で MySQL Server インスタンスに接続する手順および例は、URI 類似文字列またはキーと値のペアを使用したサーバーへの接続 を参照してください。

接続プロトコルを省略して、プロトコルのデフォルトポートの指定など、他の接続パラメータに基づいて MySQL Shell で自動的に検出されるようにできます。 接続に使用されたプロトコルを確認するには、MySQL Shell\status コマンドまたは shell.status() メソッドを使用します。

session グローバルオブジェクトがすでに存在する場合 (起動時またはその後に作成)、\connect コマンドまたは shell.connect() メソッドを使用して新しい接続を作成すると、MySQL Shell は session グローバルオブジェクトによって表される既存の接続をクローズします。 これは、shell.connect() メソッドによって作成された新しいセッションオブジェクトを別の変数に割り当てる場合でも当てはまります。 session グローバルオブジェクト (session 変数によって参照される) の値は、引き続き新しい接続の詳細で更新されます。 複数の同時接続を使用可能にする場合は、セクション4.2.3「JavaScript および Python モードでのスクリプトセッション」 で説明されている代替機能を使用してこれらを作成します。

4.2.3 JavaScript および Python モードでのスクリプトセッション

JavaScript および Python モードで使用可能な関数を使用して、選択したタイプの複数のセッションオブジェクトを作成し、変数に割り当てることができます。 これらのセッションオブジェクトを使用すると、単一の MySQL Shell インスタンスから複数の MySQL Server インスタンスまたは同じインスタンスを複数の方法で操作するための同時接続を確立および管理できます。

セッションオブジェクトを作成する関数は、mysqlx、mysql JavaScript および Python モジュールで使用できます。 これらのモジュールは、使用前にインポートする必要があります。これは、MySQL Shell を対話モードで使用する場合に自動的に実行されます。 mysqlx.getSession() 関数は、指定された接続データを使用して MySQL Server インスタンスへの X プロトコル 接続をオープンし、接続を表す Session オブジェクトを戻します。 関数 mysql.getClassicSession() および mysql.getSession() は、指定された接続データを使用して MySQL Server インスタンスへの クラシック MySQL プロトコル 接続を開き、ClassicSession オブジェクトを返して接続を表します。 これらの関数では、MySQL Shell が使用する接続プロトコルは、別のオプションを使用して選択されるのではなく、関数に組み込まれているため、ポートの正しいプロトコルと一致する適切な関数を選択する必要があります。

MySQL Shell 8.0.20 から、MySQL Shell は shell グローバルオブジェクトに独自の openSession() メソッドを提供します。JavaScript または Python モードで使用でき、shell.openSession() は X プロトコル と クラシック MySQL プロトコル の両方で使用できます。 接続データの一部として接続プロトコルを指定するか、MySQL Shell が他の接続パラメータ (プロトコルのデフォルトのポート番号など) に基づいて接続プロトコルを自動的に検出するようにします。

これらのすべての関数の接続データは、URI のような接続文字列として、またはキーと値のペアのディクショナリとして指定できます。 戻されたセッションオブジェクトには、割り当てる変数を使用してアクセスできます。 次の例では、mysql.getClassicSession() 関数を使用して クラシック MySQL プロトコル 接続をオープンする方法を示します。この関数は、接続を表す ClassicSession オブジェクトを返します:

```
mysql-js> var s1 = mysql.getClassicSession('user@localhost:3306', 'password');
mysql-js> s1
<ClassicSession:user@localhost:3306>
```

この例では、shell.openSession() を Python モードで使用して、接続に必要な圧縮で X プロトコル 接続をオープンする方法を示します。 Session オブジェクトが返されます:

```
mysql-py> s2 = shell.open_session('mysqlx://user@localhost:33060?compression=required', 'password')
mysql-py> s2
<Session:user@localhost:33060>
```

これらの関数を使用して JavaScript モードで作成したセッションオブジェクトは、JavaScript モードでのみ使用でき、セッションオブジェクトが Python モードで作成された場合も同様です。 SQL モードでは複数のセッションオブ

ジェクトを作成できません。 セッションオブジェクトは、作成したモードで割り当てられた変数を使用してのみ参照できますが、任意のモードで shell.setSession() メソッドを使用して、作成して変数に割り当てたセッションオブジェクトを session グローバルオブジェクトとして設定できます。 例:

```
mysql-js> var s3 = mysqlx.getSession('user@localhost:33060', 'password');
mysql-js> s3
<Session:user@localhost:33060>
mysql-js> shell.setSession(s3);
<Session:user@localhost:33060>
mysql-js> session
<Session:user@localhost:33060>
mysql-js> shell.status();
MySQL Shell version 8.0.18

Session type:               X Protocol
Connection Id:              5
Current schema:
Current user:               user@localhost
...
TCP port:                   33060
...
```

セッションオブジェクト s3 は session グローバルオブジェクトを使用して使用できるようになったため、それが表す X プロトコル 接続には任意の MySQL Shell モードからアクセスできます: SQL モード、JavaScript モードおよび Python モード。 この接続の詳細は、session グローバルオブジェクトによって表される接続の詳細のみを表示する shell.status() メソッドを使用して表示することもできます。 MySQL Shell インスタンスに複数のオープン接続があり、それらのいずれも session グローバルオブジェクトとして設定されていない場合、shell.status() メソッドは「「未接続」」を返します。

shell.setSession() を使用して設定したセッションオブジェクトは、session グローバルオブジェクトとして設定された既存のセッションオブジェクトを置き換えます。 置換されたセッションオブジェクトが最初に作成され、mysqlx 関数、mysql 関数または shell.openSession() のいずれかを使用して変数に割り当てられた場合、そのセッションオブジェクトは引き続き存在し、その接続は開いたままになります。 この接続は、最初に作成された MySQL Shell モードで引き続き使用でき、shell.setSession() を使用していつでも session グローバルオブジェクトに再度含めることができます。 置換されたセッションオブジェクトが shell.connect() メソッドで作成され、変数に割り当てられている場合も同様です。 置換されたセッションオブジェクトが、MySQL Shell の起動時、\connect コマンドの使用時、または shell.connect() メソッドの使用時に変数に割り当てずに作成された場合、その接続はクローズされ、再度使用する場合はセッションオブジェクトを再作成する必要があります。

4.3 MySQL Shell 接続

MySQL Shell は、X プロトコル と クラシック MySQL プロトコル の両方を使用して MySQL Server に接続できます。 MySQL Shell がグローバルに接続する MySQL サーバーインスタンスは、次の方法で指定できます:

- MySQL Shell を起動する場合は、コマンドパラメータを使用します。 セクション4.3.1「個々のパラメータを使用した接続」を参照してください。
- MySQL Shell の実行中に、\connect instance コマンドを使用します。 セクション3.1「MySQL Shell のコマンド」を参照してください。
- Python または JavaScript モードで実行している場合は、shell.connect() メソッドを使用します。

MySQL サーバーインスタンスに接続するこれらの方法では、すべての MySQL Shell 実行モードで使用できる接続であるグローバルセッションが作成されます: SQL モード、JavaScript モードおよび Python モード。 session という名前の MySQL Shell グローバルオブジェクトはこの接続を表し、変数 session はその接続への参照を保持します。 shell.openSession(), mysqlx.getSession(), mysql.getSession() または mysql.getClassicSession() 関数を使用して、MySQL サーバーインスタンスへの他の接続を表す複数の追加セッションオブジェクトを作成することもできます。 これらの接続は、作成したモードで使用でき、一度にいずれかの接続を MySQL Shell グローバルセッションとして割り当てて、すべてのモードで使用できます。 セッションオブジェクトの説明、グローバルセッションの操作方法、および MySQL Shell インスタンスから複数の接続を作成および管理する方法については、セクション 4.2「MySQL Shell セッション」 を参照してください。

MySQL サーバーインスタンスに接続するこれらの様々な方法はすべて、次のように接続を指定することをサポートします:

- URI のような文字列で指定されたパラメータは、myuser@example.com:3306/main-schema などの構文を使用します。 完全な構文については、URI 類似の接続文字列を使用した接続 を参照してください。
- キーと値のペアで指定されたパラメータは、{user:'myuser', host:'example.com', port:3306, schema:'main-schema'} などの構文を使用します。 これらのキーと値のペアは、実装用の言語自然構造で提供されます。 たとえば、キーと値のペアを JavaScript の JSON オブジェクトとして、または Python のディクショナリとして使用して、接続パラメータを指定できます。 完全な構文については、キーと値のペアを使用した接続 を参照してください。

詳しくはURI 類似文字列またはキーと値のペアを使用したサーバーへの接続をご覧ください。

> **重要**
>
> 接続の選択方法に関係なく、パスワードが MySQL Shell によってどのように処理されるかを理解することが重要です。 デフォルトでは、接続にはパスワードが必要とみなされます。パスワード (最大 128 文字) はログインプロンプトで要求され、セクション4.4「プラガブルパスワードストア」 を使用して格納できます。 指定したユーザーにパスワードなしのアカウントがある場合 (セキュアではなく推奨されません)、またはソケットピア資格証明認証が使用されている場合 (たとえば、Unix ソケット接続を使用している場合)、パスワードが指定されず、パスワードプロンプトが不要であることを明示的に指定する必要があります。 これを行うには、次のいずれかの方法を使用します:
>
> - URI のような接続文字列を使用して接続している場合は、文字列の user の後に:を配置しますが、その後にパスワードを指定しないでください。
> - キーと値のペアを使用して接続する場合は、password キーの後に"を使用して空の文字列を指定します。
> - 個々のパラメータを使用して接続する場合は、--no-password オプションを指定するか、空の値で --password= オプションを指定します。

接続のパラメータを指定しない場合は、次のデフォルトが使用されます:

- user のデフォルトは、現在のシステムユーザー名です。
- host のデフォルトは localhost です。
- port のデフォルトは、X プロトコル 接続を使用する場合は X プラグイン ポート 33060、クラシック MySQL プロトコル 接続を使用する場合はポート 3306 です。

接続タイムアウトを構成するには、connect-timeout 接続パラメータを使用します。 connect-timeout の値は、ミリ秒単位の時間枠を定義する負でない整数である必要があります。 タイムアウトのデフォルト値は 10000 ミリ秒 (10 秒) です。 例:

```
// Decrease the timeout to 2 seconds.
mysql-js> \connect user@example.com?connect-timeout=2000
// Increase the timeout to 20 seconds
mysql-js> \connect user@example.com?connect-timeout=20000
```

タイムアウトを無効にするには、connect-timeout の値を 0 に設定します。これは、基礎となるソケットがタイムアウトするまでクライアントが待機することを意味します (プラットフォームによって異なります)。

TCP 接続のかわりに、Unix ソケットファイルまたは Windows 名前付きパイプを使用して接続できます。 その手順は、セクション4.3.2「Unix ソケットおよび Windows Named Pipes を使用した接続」を参照してください。

MySQL サーバーインスタンスで暗号化された接続がサポートされている場合は、暗号化を使用するように接続を有効化および構成できます。 その手順は、セクション4.3.3「暗号化された接続の使用」を参照してください。

また、MySQL Shell と MySQL サーバーインスタンスの間で送信されるすべてのデータに対して、接続で圧縮を使用するようにリクエストすることもできます。 その手順は、セクション4.3.4「圧縮接続の使用」を参照してください。

サーバーへの接続が失われた場合は、\reconnect コマンドを使用できます。これにより、MySQL Shell は、既存の接続パラメータを使用して、現在のグローバルセッションに対して複数の再接続を試行します。 \reconnect コマンドは、パラメータまたはオプションなしで指定されます。 これらの試行が失敗した場合は、\connect コマンドを使用して接続パラメータを指定することで、新しい接続を作成できます。

4.3.1 個々のパラメータを使用した接続

接続文字列を使用して接続パラメータを指定するだけでなく、値ごとに個別のコマンドパラメータを使用して MySQL Shell を起動するときに接続データを定義することもできます。 MySQL Shell コマンドオプションの詳細は、セクションA.1「mysqlsh — MySQL Shell」 を参照してください。

次の接続関連パラメータを使用します:

- --user (-u) value
- --host (-h) value
- --port (-P) value
- --schema または --database (-D) value
- --socket (-S)

コマンドオプションの動作は、コマンドオプションを使用した MySQL Server への接続 で説明されている mysql クライアントで使用されるオプションと同様です。

次のコマンドオプションを使用して、接続にパスワードを指定するかどうか、およびその方法を制御します:

- --password=password (-ppassword) に値を指定すると、接続に使用するパスワード (128 文字以内) が指定されます。 長い形式の --password= では、オプションとその値の間に空白ではなく等号を使用する必要があります。 短い形式の -p では、オプションとその値の間に空白を入れないでください。 いずれの場合もスペースが使用される場合、値はパスワードとして解釈されず、別の接続パラメータとして解釈される可能性があります。

  コマンド行でのパスワード指定は、セキュアでないと考えるべきです。 パスワードセキュリティーのためのエンドユーザーガイドラインを参照してください。 オプションファイルを使用すれば、コマンド行でパスワードを指定することを回避できます。

- 値がなく等号がない --password または値がない -p は、パスワードプロンプトを要求します。

- --no-password または空の値を持つ --password= は、ユーザーがパスワードなしで接続していることを指定します。 サーバーに接続するときに、ユーザーがパスワードなしのアカウントを持っている場合 (セキュアではなく推奨されない)、またはソケットピア資格証明認証が使用されている場合 (Unix ソケット接続の場合)、次のいずれかの方法を使用して、パスワードが指定されず、パスワードプロンプトが不要であることを明示的に指定する必要があります。

--uri オプションの使用や --user などの個々のパラメータの指定など、パラメータが複数の方法で指定されている場合は、次のルールが適用されます:

- 引数が複数回指定されている場合は、最後の外観の値が使用されます。
- 個々の接続引数と --uri の両方が指定されている場合、--uri の値がベースとして使用され、個々の引数の値はベース URI のような文字列の特定のコンポーネントをオーバーライドします。

  たとえば、URI のような文字列から user をオーバーライドするには、次のようにします:

```
shell> mysqlsh --uri user@localhost:33065 --user otheruser
```

MySQL Shell からサーバーへの接続は暗号化でき、これらの機能をリクエストし、サーバーでサポートされている場合は圧縮できます。 暗号化された接続を確立する手順については、セクション4.3.3「暗号化された接続の使用」 を参照してください。 圧縮接続を確立する手順については、セクション4.3.4「圧縮接続の使用」 を参照してください。

次の例に、コマンドパラメータを使用して接続を指定する方法を示します。 ポート 33065 で指定されたユーザーとの X プロトコル 接続を確立しようとします:

```
shell> mysqlsh --mysqlx -u user -h localhost -P 33065
```

指定されたユーザーとの クラシック MySQL プロトコル 接続を確立しようとし、接続の圧縮をリクエストします:

```
shell> mysqlsh --mysql -u user -h localhost -C
```

4.3.2 Unix ソケットおよび Windows Named Pipes を使用した接続

Unix では、次の条件が満たされた場合、MySQL Shell 接続はデフォルトで Unix ソケットを使用します:

- TCP ポートが指定されていません。
- ホスト名が指定されていないか、localhost と同じです。
- ソケットファイルへのパスの有無にかかわらず、--socket または -S オプションが指定されています。

値なしで等号なしで --socket を指定した場合、または値なしで -S を指定した場合、プロトコルのデフォルトの Unix ソケットファイルが使用されます。 代替 Unix ソケットファイルへのパスを指定すると、そのソケットファイルが使用されます。

ホスト名が指定されているが、localhost ではない場合は、かわりに TCP 接続が確立されます。 この場合、TCP ポートが指定されていないと、デフォルト値 3306 が使用されます。

Windows では、クラシック MySQL プロトコル を使用する MySQL Shell 接続の場合、ホスト名をピリオド (.) として指定した場合。)、MySQL Shell は名前付きパイプを使用して接続します。

- URI のような接続文字列を使用して接続する場合は、user@. を指定
- キーと値のペアを使用して接続する場合は、{"host": "."} を指定
- 個々のパラメータを使用して接続する場合は、--host=. または -h . を指定

デフォルトでは、パイプ名 MySQL が使用されます。 代替の名前付きパイプは、--socket オプションを使用するか、URI のような接続文字列の一部として指定できます。

URI のような文字列では、パーセントエンコーディングを使用するか、パスをカッコで囲んで、Unix ソケットファイルまたは Windows 名前付きパイプへのパスをエンコードする必要があります。 カッコを使用すると、/ディレクトリセパレータ文字などの文字をパーセントエンコードする必要がなくなります。 Unix ソケットファイルへのパスがクエリー文字列の一部として URI のような文字列に含まれている場合、先頭のスラッシュはパーセントエンコードする必要がありますが、ホスト名を置き換える場合、次の例に示すように先頭のスラッシュはパーセントエンコードしないでください:

```
mysql-js> \connect user@localhost?socket=%2Ftmp%2Fmysql.sock
mysql-js> \connect user@localhost?socket=(/tmp/mysql.sock)
mysql-js> \connect user@/tmp%2Fmysql.sock
mysql-js> \connect user@(/tmp/mysql.sock)
```

Windows の場合のみ、次の例に示すように、名前付きパイプの先頭に文字\\.\を付加し、パーセントエンコーディングを使用してエンコードするか、カッコで囲む必要があります:

```
(\\.\named:pipe)
\\.\named%3Apipe
```

> **重要**
>
> Windows では、名前付きパイプを使用して 1 つ以上の MySQL Shell セッションが MySQL Server インスタンスに接続されていて、サーバーを停止する必要がある場合、まず MySQL Shell セッションを閉じる必要があります。 この方法でまだ接続されているセッションでは、シャットダウン手順中にサーバーがハングアップする可能性があります。 これが発生した場合は、MySQL Shell を終了すると、サーバーは停止手順を続行します。

Unix ソケットファイルおよび Windows 名前付きパイプとの接続の詳細は、コマンドオプションを使用した MySQL Server への接続 および URI 類似文字列またはキーと値のペアを使用したサーバーへの接続 を参照してください。

4.3.3 暗号化された接続の使用

TLS (SSL とも呼ばれる) 対応の MySQL サーバーに接続する場合は、暗号化された接続を使用できます。 MySQL Shell の構成の多くは、MySQL サーバーで使用されるオプションに基づいています。詳細は、暗号化された接続の使用 を参照してください。

MySQL Shell の起動時に暗号化された接続を構成するには、次のコマンドオプションを使用します:

- --ssl : 非推奨。将来のバージョンで削除されます。 --ssl-mode の使用。 このオプションは、暗号化された接続を有効または無効にします。
- --ssl-mode : このオプションは、サーバーへの接続の目的のセキュリティー状態を指定します。
- --ssl-ca=file_name: 信頼できる SSL 認証局のリストを含む PEM 形式のファイルへのパス。
- --ssl-capath=dir_name: PEM 形式の信頼できる SSL 認証局証明書を含むディレクトリへのパス。
- --ssl-cert=file_name: 暗号化された接続の確立に使用する PEM 形式の SSL 証明書ファイルの名前。
- --ssl-cipher=name: 暗号化された接続の確立に使用する SSL 暗号の名前。
- --ssl-key=file_name: 暗号化された接続の確立に使用する PEM 形式の SSL キーファイルの名前。
- --ssl-crl=name: PEM 形式の証明書失効リストを含むファイルへのパス。
- --ssl-crlpath=dir_name: PEM 形式の証明書失効リストを含むファイルを含むディレクトリへのパス。
- --tls-version=version: 暗号化された接続に許可される TLS プロトコル。カンマ区切りリストで指定します。 たとえば、--tls-version=TLSv1.1,TLSv1.2 です。
- --tls-ciphersuites=suites: 暗号化された接続に許可される TLS 暗号スイート。TLS 暗号スイート名のコロン区切りリストで指定します。 たとえば、--tls-ciphersuites=TLS_DHE_PSK_WITH_AES_128_GCM_SHA256:TLS_CHACHA20_POLY1305_SHA256 です。 バージョン 8.0.18 に追加されました。

または、SSL オプションを URI のような接続文字列の一部としてクエリー要素の一部としてエンコードできます。使用可能な SSL オプションは、前述のオプションと同じですが、前のハイフンなしで記述されます。 たとえば、ssl-ca は --ssl-ca と同等です。

URI のような文字列で指定するパスは、パーセントエンコードする必要があります。次に例を示します:

```
ssluser@127.0.0.1?ssl-ca%3D%2Froot%2Fclientcert%2Fca-cert.pem%26ssl-cert%3D%2Fro\
ot%2Fclientcert%2Fclient-cert.pem%26ssl-key%3D%2Froot%2Fclientcert%2Fclient-key
.pem
```

詳しくはURI 類似文字列またはキーと値のペアを使用したサーバーへの接続をご覧ください。

JavaScript または Python モードでスクリプトセッションの暗号化された接続を確立するには、connectionData ディクショナリに SSL 情報を設定します。 例:

```
mysql-js> var session=mysqlx.getSession({host: 'localhost',
                                         user: 'root',
                                         password: 'password',
                                         ssl_ca: "path_to_ca_file",
                                         ssl_cert: "path_to_cert_file",
                                         ssl_key: "path_to_key_file"});
```

ssl-mode が指定されておらず、ssl-ca も ssl-capath も指定されていない場合、mysqlx.getSession()、mysql.getSession() または mysql.getClassicSession() を使用して作成されたセッションでは ssl-mode=REQUIRED がデフォルトとして使用されます。 ssl-mode が提供されておらず、ssl-ca または ssl-capath が提供されている場合、作成されるセッションは ssl-mode=VERIFY_CA にデフォルト設定されます。

詳しくはキーと値のペアを使用した接続をご覧ください。

4.3.4 圧縮接続の使用

MySQL Shell 8.0.14 から、クラシック MySQL プロトコル を使用する MySQL Shell 接続、および X プロトコル を使用する MySQL Shell 接続の圧縮を MySQL Shell 8.0.20 からリクエストできます。 セッションの圧縮がリクエストされると、サーバーが圧縮をサポートし、圧縮アルゴリズムを MySQL Shell と一致させることができる場合、クライアントとサーバーの間で送信されるすべての情報が圧縮されます。 圧縮は、アップグレードチェッカユーティリティなどの MySQL Shell ユーティリティで使用される接続にリクエストされた場合にも適用されます。

X プロトコル 接続の場合、デフォルトでは圧縮が要求され、圧縮された接続のネゴシエーションが成功しない場合は圧縮されていない接続が許可されます。 クラシック MySQL プロトコル 接続の場合、デフォルトで圧縮は無効になっています。 接続が確立されると、MySQL Shell \status コマンドは圧縮がセッションに使用されているかどうかを表示します。 このコマンドは、接続が圧縮されているかどうかを示す Disabled または Enabled を示す Compression:行を表示します。 圧縮が有効な場合は、使用中の圧縮アルゴリズムも表示されます。

defaultCompress MySQL Shell 構成オプションを設定して、すべてのグローバルセッションの圧縮をリクエストできます。 X プロトコル 接続のデフォルトでは、MySQL Shell リリースでこれがサポートされている圧縮がリクエストされるため、この構成オプションは クラシック MySQL プロトコル 接続に対してのみ有効です。

X プロトコル 接続に対する接続圧縮の動作の詳細は、X プラグイン での接続圧縮 を参照してください。 クラシック MySQL プロトコル 接続に対する接続圧縮の動作、および MySQL Server インスタンスの圧縮設定と機能の詳細は、接続圧縮制御 を参照してください。

4.3.4.1 MySQL Shell 8.0.20 以降の圧縮制御

MySQL Shell 8.0.20 から、X プロトコル 接続および クラシック MySQL プロトコル 接続の場合、MySQL Server インスタンスへの接続を管理するセッションオブジェクトを作成するたびに、その接続の圧縮が必要か、優先か、無効かを指定できます。

- required はサーバーから圧縮接続をリクエストし、サーバーが圧縮をサポートしていないか、圧縮プロトコルで MySQL Shell と一致しない場合、接続は失敗します。
- サーバーが圧縮をサポートしていないか、圧縮プロトコルで MySQL Shell に同意できない場合、preferred はサーバーから圧縮接続をリクエストし、圧縮されていない接続にフォールバックします。 これは、X プロトコル 接続のデフォルトです。
- disabled は圧縮されていない接続をリクエストし、サーバーが圧縮された接続のみを許可している場合、接続は失敗します。 これは、クラシック MySQL プロトコル 接続のデフォルトです。

MySQL Shell 8.0.20 から、接続に使用できる圧縮アルゴリズムを選択することもできます。 デフォルトでは、MySQL Shell は zlib、LZ4 および zstd アルゴリズムを X プロトコル 接続用にサーバーに提案し、クラシック MySQL プロトコル 接続用に zlib および zstd アルゴリズムを提案します (LZ4 アルゴリズムはサポートしていません)。 これらのアルゴリズムの任意の組合せを指定できます。 圧縮アルゴリズムを指定する順序は、MySQL Shell がそれらを提案するプリファレンスの順序ですが、プロトコルおよびサーバー構成によっては、サーバーがこのプリファレンスの影響を受けない場合があります。

圧縮アルゴリズムまたはその組合せを指定すると、接続の圧縮が自動的にリクエストされるため、圧縮が必要か、優先か、無効かを指定するために別のパラメータを使用するかわりに、圧縮を行うことができます。 この接続圧縮制御方法では、オプション uncompressed (圧縮されていない接続を許可) を圧縮アルゴリズムのリストに追加することで、圧縮が必要か優先かを指定します。 uncompressed を含める場合は圧縮が推奨され、含めない場合は圧縮が必要です。 uncompressed を単独で渡して、圧縮が無効であることを指定することもできます。 圧縮が必要、優先または無効であることを別のパラメータで指定すると、圧縮アルゴリズムのリストで uncompressed を使用するよりも優先されます。

接続の数値圧縮レベルを指定することもできます。これは、X プロトコル 接続の圧縮アルゴリズム、または クラシック MySQL プロトコル 接続の zstd アルゴリズムにのみ適用されます。 X プロトコル 接続の場合、指定した圧縮レベルが最終的に選択されたアルゴリズムに対してサーバーで受け入れられないと、サーバーは X プラグイン での接続圧縮 にリストされている動作に従って適切な設定を選択します。 たとえば、MySQL Shell が zlib アルゴリズムの圧縮レベル 7 を要求し、サーバーの mysqlx_deflate_max_client_compression_level システム変数 (deflate、または zlib、圧縮の最大圧縮レベルを制限) がデフォルトの 5 に設定されている場合、サーバーは最大許容圧縮レベル 5 を使用します。

MySQL サーバーインスタンスがプロトコルの接続圧縮をサポートしていない場合 (MySQL 8.0.19 for X プロトコル 接続の前の場合)、または接続圧縮をサポートしているが接続アルゴリズムおよび圧縮レベルの指定をサポートしていない場合、MySQL Shell はサポートされていないパラメータを指定せずに接続を確立します。

MySQL Shell 8.0.20 からの接続の圧縮をリクエストするには、次のいずれかの方法を使用します:

- コマンドラインから MySQL Shell を起動し、別のコマンドオプションを使用して接続パラメータを指定する場合は、--compress (-C) オプションを使用して、接続の圧縮が必要か、優先か、無効かを指定します。 例:

```
shell> mysqlsh --mysqlx -u user -h localhost -C required
```

--compress (-C) オプションは、以前のリリースの MySQL Shell (MySQL 8.0.14 に戻る) と互換性があり、これらのリリースのブール設定を受け入れます。 MySQL Shell 8.0.20 から、パラメータを指定せずに --compress (-C) のみを指定した場合、接続には圧縮が必要です。

前述の X プロトコル 接続の例では、zlib、LZ4 および zstd アルゴリズムをこの優先順位でサーバーに提案しています。 圧縮アルゴリズムの代替の組合せが必要な場合は、--compression-algorithms オプションを使用して、許可されているアルゴリズムのカンマ区切りリストで文字列を指定することで、これを指定できます。 X プロトコル 接続の場合、zlib、lz4 および zstd を任意の組合せおよび順序で使用できます。 クラシック MySQL プロトコル 接続の場合、zlib および zstd を任意の組合せおよび順序で使用できます。 クラシック MySQL プロトコル 接続の次の例では、zstd アルゴリズムのみが許可されます:

```
shell> mysqlsh --mysql -u user -h localhost -C preferred --compression-algorithms=zstd
```

--compress (-C) オプションを指定せずに --compression-algorithms のみを使用して、圧縮をリクエストすることもできます。 この場合、非圧縮接続を許可する場合はアルゴリズムのリストに uncompressed を追加し、許可しない場合は省略します。 この形式の接続圧縮制御は、mysql、mysqlbinlog などの他の MySQL クライアントと互換性があります。 クラシック MySQL プロトコル 接続の次の例は、preferred が別のオプションとして指定されている (つまり、zstd アルゴリズムで圧縮を提案するが、圧縮されていない接続にフォールバックする) 前述の例と同じ効果があります:

```
shell> mysqlsh --mysql -u user -h localhost --compression-algorithms=zstd,uncompressed
```

--compression-level または --zstd-compression-level オプションを使用して圧縮レベルを構成できます。これらのオプションは、クラシック MySQL プロトコル 接続に対しては検証されますが、X プロトコル 接続に対しては検証されません。--compression-level では、X プロトコル 接続の場合はアルゴリズムの圧縮レベルに整数を指定し、クラシック MySQL プロトコル 接続の場合は zstd アルゴリズムにのみ整数を指定します。--zstd-compression-level では、zstd アルゴリズムの圧縮レベルに 1 から 22 の整数を指定し、mysql や mysqlbinlog などの他の MySQL クライアントと互換性があります。 たとえば、X プロトコル 接続の次の接続パラメータは、圧縮がグローバルセッションに必要であり、リクエストされた圧縮レベルが 5 の LZ4 または zstd アルゴリズムを使用する必要があることを指定します:

```
shell> mysqlsh --mysqlx -u user -h localhost -C required --compression-algorithms=lz4,zstd --compression-level=5
```

- URI に似た接続文字列を使用して、コマンドライン、MySQL Shell\connect コマンド、shell.connect(), shell.openSession(), mysqlx.getSession(), mysql.getSession() または mysql.getClassicSession() 関数のいずれかから接続パラメータを指定する場合は、クエリー文字列で compression パラメータを使用して、圧縮が必要か、優先か、無効かを指定します。 例:

```
mysql-js> \connect user@example.com?compression=preferred
```

```
shell> mysqlsh mysqlx://user@localhost:33060?compression=disabled
```

コマンドラインオプションと同様に、compression-algorithms パラメータを使用して圧縮アルゴリズムを選択し、compression-level パラメータを使用して圧縮レベルを選択します。 (URI のような接続文字列には、zstd 固有の圧縮レベルパラメータはありません。) compression パラメータを指定せずに compression-algorithms パラメータを使用することもできます。これには、圧縮されていない接続を許可または禁止する uncompressed オプションも含まれます。 たとえば、これらの両方の接続パラメータのセットでは、圧縮は推奨されますが、圧縮されていない接続が許可され、zlib および zstd アルゴリズムが受け入れられ、圧縮レベル 4 を使用するように指定します:

```
mysql-js> \connect user@example.com:33060?compression=preferred&compression-algorithms=zlib,zstd&compression-level=4
```

```
mysql-js> \connect user@example.com:33060?compression-algorithms=zlib,zstd,uncompressed&compression-level=4
```

- キーと値のペアを使用して、MySQL Shell\connect コマンド、shell.connect(), shell.openSession(), mysqlx.getSession(), mysql.getSession() または mysql.getClassicSession() 関数のいずれかで接続パラメータを指定する場合は、オプションのディクショナリで compression パラメータを使用して、圧縮が必要か、優先か、無効かを指定します。 例:

```
mysql-js>  var s1=mysqlx.getSession({host: 'localhost',
                                     user: 'root',
                                     password: 'password',
                                     compression: 'required'});
```

compression-algorithms パラメータを使用して圧縮アルゴリズムを選択し、compression-level パラメータを使用して、コマンドラインおよび URI のような接続文字列メソッドと同様に圧縮レベルを選択します。 (キーと値のペアの zstd 固有の圧縮レベルパラメータはありません。) compression パラメータを指定せずに compression-algorithms パラメータを使用することもできます。これには、圧縮されていない接続を許可または禁止する uncompressed オプションも含まれます。

4.3.4.2 8.0.19 を介した MySQL Shell 8.0.14 の圧縮制御

MySQL Shell 8.0.14 から 8.0.19 までのリリースでは、クラシック MySQL プロトコル を使用する接続に対してのみ圧縮をリクエストできます。 デフォルトでは、圧縮は要求されません。 これらのリリースでの圧縮では、zlib 圧縮アルゴリズムが使用されます。 これらのリリースでは圧縮を要求できないため、圧縮がサーバーでサポートされていない場合、セッションは圧縮されていない接続にフォールバックします。

これらの MySQL Shell リリースでは、圧縮制御は、接続の圧縮の有効化 (true を指定) または無効化 (false を指定) に制限されます。 この圧縮制御を含む MySQL Shell リリースを使用して、圧縮アルゴリズムのクライアントリクエストがサポートされている MySQL 8.0.18 以降のサーバーインスタンスに接続する場合、圧縮を有効にすることは、zlib、uncompressed のアルゴリズムセットを提案することと同等です。

MySQL Shell は、8.0.14 より前のリリースでは圧縮をリクエストできません。

MySQL Shell 8.0.14 から 8.0.19 の接続の圧縮をリクエストするには、次のいずれかの方法を使用します:

- コマンドラインから MySQL Shell を起動し、別のコマンドオプションを使用して接続パラメータを指定する場合は、--compress (-C) オプションを使用します。次に例を示します:

```
shell> mysqlsh --mysql -u user -h localhost -C
```

- URI に似た接続文字列を使用して、コマンドラインまたは MySQL Shell\connect コマンド、あるいは shell.connect() メソッドから接続パラメータを指定する場合は、クエリー文字列で compression=true パラメータを使用します:

```
mysql-js> \connect user@example.com?compression=true
```

```
shell> mysqlsh mysql://user@localhost:3306?compression=true
```

- MySQL Shell\connect コマンドまたは mysql.getClassicSession() メソッドを使用して、キーと値のペアを使用して接続パラメータを指定する場合は、オプションのディクショナリで compression パラメータを使用します:

```
mysql-js> var s1=mysql.getClassicSession({host: 'localhost',
                                          user: 'root',
                                          password: 'password',
                                          compression: 'true'});
```

4.4 プラガブルパスワードストア

MySQL Shell をより柔軟かつセキュアに使用するために、キーチェーンなどのシークレットストアを使用してサーバー接続のパスワードを永続化できます。 対話形式で接続のパスワードを入力すると、接続の資格証明としてサーバー URL とともに格納されます。 例:

```
mysql-js> \connect user@localhost:3310
Creating a session to 'user@localhost:3310'
Please provide the password for 'user@localhost:3310': ********
Save password for 'user@localhost:3310'? [Y]es/[N]o/Ne[v]er (default No): y
```

サーバー URL のパスワードが格納されると、MySQL Shell はセッションを開くたびに、構成済の Secret Store Helper からパスワードを取得して、対話形式でパスワードを入力せずにサーバーにログインします。 MySQL Shell によって実行されるスクリプトについても同じことが保持されます。 Secret Store Helper が構成されていない場合、パスワードは対話形式でリクエストされます。

> **重要**
>
> MySQL Shell では、サーバー URL とパスワードはシークレットストアを介してのみ永続化され、パスワード自体は永続化されません。

パスワードは、手動で入力した場合にのみ永続化されます。 サーバー URI のような接続文字列を使用して、または mysqlsh の実行時にコマンドラインでパスワードを指定した場合、そのパスワードは永続化されません。

MySQL Shell への接続に使用できるパスワードの最大長は 128 文字です。

MySQL Shell では、次のシークレットストアの組込みサポートが提供されます:

- MySQL サーバーでサポートされているすべてのプラットフォーム (MySQL クライアントパッケージがインストールされている場合) で使用可能な MySQL login-path で、永続記憶域を提供します。 mysql_config_editor — MySQL 構成ユーティリティーを参照してください。
- macOS キーチェーン。「ここ」を参照してください。
- Windows API については、「ここ」を参照してください。

MySQL Shell が対話モードで実行されている場合、新しいセッションが開始され、ユーザーにパスワードの入力を求めるプロンプトが表示されるたびにパスワードの取得が実行されます。 プロンプトが表示される前に、Secret Store Helper に対して、セッション URL を使用してパスワードをクエリーします。 一致するものが見つかった場合は、このパスワードを使用してセッションを開きます。 取得したパスワードが無効な場合は、メッセージがログに追加され、パスワードがシークレットストアから消去され、MySQL Shell によってパスワードの入力を求められます。

MySQL Shell が非対話モードで実行されている場合 (たとえば、--no-wizard が使用された場合)、パスワードの取得は対話モードと同じ方法で実行されます。 ただし、この場合、Secret Store Helper で有効なパスワードが見つからないと、MySQL Shell はパスワードなしでセッションを開こうとします。

サーバー URL のパスワードは、MySQL サーバーへの接続が成功し、パスワードがシークレットストアヘルパーによって取得されなかった場合は常に格納できます。 パスワードを格納するかどうかは、ここで説明する credentialStore.savePasswords および credentialStore.excludeFilters に基づいて決定されます。

パスワードの自動格納および自動取得は、次の場合に実行されます:

- mysqlsh は、最初のセッションの確立時に接続オプションを使用して起動されます
- 組込みの\connect コマンドを使用する場合
- shell.connect() メソッドを使用する場合
- 接続が必要な AdminAPI メソッドを使用する場合

4.4.1 プラガブルパスワード構成オプション

プラガブルパスワードストアを構成するには、shell.options インタフェースを使用します。セクション10.4「MySQL Shell オプションの構成」 を参照してください。 次のオプションは、プラガブルパスワードストアを構成します。

shell.options.credentialStore.helper = "login-path"

パスワードの格納および取得に使用されるシークレットストアヘルパーを指定する文字列。 デフォルトでは、このオプションは、現在のプラットフォームのデフォルトヘルパーを識別する特殊な値 default に設定されています。 shell.listCredentialHelpers() メソッドによって返される任意の値に設定できます。 この値を無効な値または不明なヘルパーに設定すると、例外が発生します。 mysqlsh の起動時に無効な値が検出されると、エラーが表示され、パスワードの格納および取得は無効になります。 パスワードの自動格納および取得を無効にするには、このオプションを特別な値 <disabled> に設定します。たとえば、次のように発行します:

```
shell.options.set("credentialStore.helper", "<disabled>")
```

このオプションを無効にすると、ここで説明するすべての資格証明ストア MySQL Shell メソッドを使用した場合に例外が発生します。

shell.options.credentialStore.savePasswords = "value"

パスワードの自動格納を制御する文字列。 有効な値は次のとおりです:

- always - パスワードは、シークレットストアですでに使用可能な場合、またはサーバー URL が credentialStore.excludeFilters 値と一致する場合を除き、常に格納されます。
- never - パスワードは格納されません。
- prompt - 対話モードでは、サーバー URL が shell.credentialStore.excludeFilters の値と一致しない場合、パスワードを格納するかどうかを尋ねるプロンプトが表示されます。 考えられる回答は、このパスワードを保存する yes、このパスワードを保存しない no、このパスワードを保存しない never および credentialStore.excludeFilters に URL を追加する never です。 credentialStore.excludeFilters の変更された値は永続化されません。つまり、MySQL Shell が再起動されるまで有効です。 MySQL Shell が非対話型モードで実行されている場合 (たとえば、--no-wizard オプションが使用された場合)、credentialStore.savePasswords オプションは常に never です。

このオプションのデフォルト値は prompt です。

shell.options.credentialStore.excludeFilters = ["*@myserver.com:*"];

パスワードの自動格納から除外するサーバー URL を指定する文字列のリスト。 各文字列は、明示的な URL または glob パターンのいずれかです。 格納されようとしているサーバー URL がこのオプションのいずれかの文字列と一致する場合、格納されません。 有効なワイルドカード文字は次のとおりです: 任意の数の任意の文字に一致する *、および単一の文字に一致する ?。

このオプションのデフォルト値は空のリストです。

4.4.2 資格証明の使用

次の関数を使用すると、プラガブルパスワードストアを操作できます。 使用可能なシークレットストアヘルパーをリストしたり、資格証明をリスト、格納および取得できます。

var list = shell.listCredentialHelpers();

文字列のリストを返します。各文字列は、現在のプラットフォームで使用可能な Secret Store Helper の名前です。 default および <disabled> の特別な値はリストには含まれていませんが、credentialStore.helper オプションの有効な値です。

shell.storeCredential(url[, password]);

現在の Secret Store Helper (credentialStore.helper) を使用して、指定された資格証明を格納します。 ストア操作が失敗した場合 (たとえば、現在のヘルパーが無効な場合)、エラーがスローされます。 URL がすでにシークレットストアにある場合は、上書きされます。 このメソッドは、credentialStore.savePasswords および credentialStore.excludeFilters オプションの現在の値を無視します。 パスワードが指定されていない場合、MySQL Shell によってパスワードの入力が求められます。

shell.deleteCredential(url);

現在の Secret Store Helper (credentialStore.helper) を使用して、指定された URL の資格証明を削除します。 削除操作が失敗した場合 (たとえば、現在のヘルパーが無効であるか、指定された URL の資格証明がない場合) は、エラーがスローされます。

shell.deleteAllCredentials();

現在の Secret Store Helper (credentialStore.helper) によって管理されているすべての資格証明を削除します。 削除操作が失敗した場合 (たとえば、現在のヘルパーが無効な場合)、エラーがスローされます。

var list = shell.listCredentials();

現在の Secret Store Helper (credentialStore.helper) によって格納されている資格証明のすべての URL のリストを返します。

4.5 MySQL Shell グローバルオブジェクト

MySQL Shell には、JavaScript モードと Python モードの両方に存在する多数の組込みグローバルオブジェクトが含まれています。 組込み MySQL Shell グローバルオブジェクトは次のとおりです:

- session は、グローバルセッションの確立時に使用でき、グローバルセッションを表します。
- dba では、AdminAPI を使用して InnoDB クラスタ および InnoDB ReplicaSet の管理機能にアクセスできます。 第6章「MySQL AdminAPI の使用」を参照してください。
- cluster は InnoDB クラスタ を表します。 MySQL Shell の起動時に --cluster オプションが指定された場合にのみ移入されます。
- rs は、InnoDB ReplicaSet (バージョン 8.0.20 で追加) を表します。 MySQL Shell の起動時に --replicaset オプションが指定された場合にのみ移入されます。
- db は、デフォルトのデータベースが指定された X プロトコル 接続を使用してグローバルセッションが確立されたときに使用でき、そのスキーマを表します。
- shell では、次のような様々な MySQL Shell 関数にアクセスできます:
  - shell.options には、MySQL Shell プリファレンスを設定および設定解除する関数が用意されています。 セクション10.4「MySQL Shell オプションの構成」を参照してください。
  - shell.reports では、組込みまたはユーザー定義の MySQL Shell レポートが関数として提供され、レポートの名前が関数として使用されます。 セクション7.1「MySQL Shell でのレポート」を参照してください。
- util には、アップグレードチェッカユーティリティ、JSON インポートユーティリティ、パラレルテーブルインポートユーティリティなど、様々な MySQL Shell ツールが用意されています。 第8章「MySQL Shell ユーティリティ」を参照してください。

> **重要**
>
> MySQL Shell グローバルオブジェクトの名前はグローバル変数として予約されているため、変数の名前などとして使用しないでください。 いずれかのグローバル変数を割り当てた場合、前述の機能をオーバーライドし、リストアするには、MySQL Shell を再起動する必要があります。

独自の拡張オブジェクトを作成し、追加の MySQL Shell グローバルオブジェクトとして登録して、グローバルコンテキストで使用できるようにすることもできます。 これを行う手順は、セクション7.2「MySQL Shell への拡張オブジェクトの追加」 を参照してください。

4.6 ページャの使用

less や more などの外部ページャツールを使用するように MySQL Shell を構成できます。 ページャを構成すると、オンラインヘルプまたは SQL 操作の結果からテキストを表示するために MySQL Shell で使用されます。 次の構成の可能性を使用します:

- ページングされた出力を表示する外部コマンドを指定する文字列である shell.options[pager] = "" MySQL Shell オプションを構成します。 この文字列には、外部ページャコマンドに渡されるコマンドライン引数をオプションで含めることができます。 新しい値の正確性はチェックされません。 空の文字列を指定すると、ページャが無効になります。

  デフォルト値: 文字列が空です。

- PAGER 環境変数を構成します。これは、shell.options["pager"]オプションのデフォルト値をオーバーライドします。 shell.options["pager"]が永続化されていた場合は、PAGER 環境変数よりも優先されます。

  PAGER 環境変数は、MySQL Shell で想定されているのと同じコンテキストで Unix システムで一般的に使用され、競合は発生しません。

- --pager MySQL Shell オプションを構成します。これは、shell.options["pager"]オプションが永続化され、PAGER 環境変数が構成されている場合でも、このオプションの初期値をオーバーライドします。
- \pager | \P command MySQL Shell コマンドを使用して、shell.options["pager"]オプションの値を設定します。 引数を指定せずにコールした場合、shell.options["pager"]オプション (起動時に MySQL Shell が保持していたオプション) の初期値をリストアします。 文字列は"文字でマークすることも、マークしないこともできます。 たとえば、ページャを構成するには、次のようにします:

- 初期ページャをリストアするには、command を渡さないか、空の文字列を渡してください
- more コマンドをページャとして使用するように MySQL Shell を構成するには、more を渡します
- more -10 を渡して、オプション -10 を指定して more コマンドをページャとして使用するように MySQL Shell を構成

外部ページャツールに渡される MySQL Shell 出力は、フィルタリングなしで転送されます。 MySQL Shell で色付きプロンプトを使用している場合 (セクション10.3「プロンプトのカスタマイズ」 を参照)、出力には ANSI エスケープシーケンスが含まれます。 一部のページャでは、-R オプションを使用して解釈を有効にできる less など、これらのエスケープシーケンスがデフォルトで解釈されない場合があります。more は、デフォルトで ANSI エスケープシーケンスを解釈します。

第 5 章 MySQL Shell コードの実行

目次

5.1 アクティブな言語 31
5.2 対話型コードの実行 32
5.3 コード自動補完 33
5.4 コードの編集 35
5.5 コード履歴 35
5.6 バッチコード実行 36
5.7 出力形式 38
5.7.1 テーブル形式 38
5.7.2 タブ区切り形式 39
5.7.3 垂直フォーマット 39
5.7.4 JSON 形式の出力 40
5.7.5 JSON ラッピング 41
5.7.6 結果メタデータ 42
5.8 API コマンドラインインタフェース 43

このセクションでは、MySQL Shell でのコード実行の仕組みについて説明します。

5.1 アクティブな言語

MySQL Shell では SQL、JavaScript または Python コードを実行できますが、一度にアクティブにできる言語は 1 つのみです。 アクティブモードによって、実行されるステートメントの処理方法が決まります:

- SQL モードを使用している場合、ステートメントは SQL として処理され、実行のために MySQL サーバーに送信されます。
- JavaScript モードを使用している場合、ステートメントは JavaScript コードとして処理されます。
- Python モードを使用している場合、ステートメントは Python コードとして処理されます。

> **注記**
>
> バージョン 8.0.18 から、MySQL Shell は Python 3 を使用します。 システムでサポートされている Python 3 のインストールを含むプラットフォームの場合、MySQL Shell では使用可能な最新バージョンが使用され、サポートされている最小バージョンの Python 3.4.3 が使用されます。 Python 3 が含まれていないプラットフォームの場合、MySQL Shell には Python 3.7.4 がバンドルされます。MySQL Shell は、Python 2.6 および Python 2.7 とのコード互換性を維持しているため、これらの古いバージョンのいずれかが必要な場合は、適切な Python バージョンを使用してソースから MySQL Shell をビルドできます。

MySQL Shell を対話モードで実行する場合は、次のコマンドを入力して特定の言語をアクティブ化します: \sql, \js, \py。

MySQL Shell をバッチモードで実行する場合は、次のコマンドラインオプションのいずれかを渡して特定の言語をアクティブ化します: --js、--py または --sql。 何も指定されていない場合のデフォルトモードは JavaScript です。

MySQL Shell を使用して、ファイル code.sql のコンテンツを SQL として実行します。

```
shell> mysqlsh --sql < code.sql
```

MySQL Shell を使用して、ファイル code.js のコンテンツを JavaScript コードとして実行します。

```
shell> mysqlsh < code.js
```

MySQL Shell を使用して、ファイル code.py のコンテンツを Python コードとして実行します。

```
shell> mysqlsh --py < code.py
```

MySQL Shell 8.0.16 から、\sql コマンドの直後に SQL ステートメントを入力することで、別の言語がアクティブなときに単一の SQL ステートメントを実行できます。 例:

```
mysql-py> \sql select * from sakila.actor limit 3;
```

SQL ステートメントには追加の引用符は必要なく、ステートメントデリミタはオプションです。 このコマンドは、単一行の単一の SQL クエリーのみを受け入れます。 この形式では、MySQL Shell は\sql コマンドを入力した場合のようにはモードを切り替えません。 SQL ステートメントが実行されると、MySQL Shell は JavaScript または Python モードのままになります。

MySQL Shell 8.0.18 から、任意の言語がアクティブなときに、\system または\! コマンドの直後に実行するコマンドを入力することで、オペレーティングシステムコマンドを実行できます。 例:

```
mysql-py> \system echo Hello from MySQL Shell!
```

MySQL Shell は、オペレーティングシステムコマンドからの出力を表示するか、コマンドを実行できなかった場合はエラーを返します。

5.2 対話型コードの実行

MySQL Shell のデフォルトモードでは、コマンドプロンプトで入力したデータベース操作を対話形式で実行できます。 これらの操作は、現在の セクション5.1「アクティブな言語」 に応じて、JavaScript、Python または SQL で記述できます。 実行されると、操作の結果が画面に表示されます。

他の言語インタプリタと同様に、MySQL Shell は構文に関して非常に厳格です。 たとえば、次の JavaScript スニペットは、MySQL サーバーへのセッションを開き、コレクション内のドキュメントを読み取って出力します:

```
var mySession = mysqlx.getSession('user:pwd@localhost');
var result = mySession.getSchema('world_x').getCollection('countryinfo').find().execute();
var record = result.fetchOne();
while(record){
  print(record);
  record = result.fetchOne();
}
```

前述のように、find() へのコールの後に execute() 関数が続きます。 CRUD データベースコマンドは、実際には execute() がコールされたときに MySQL Server でのみ実行されます。 ただし、MySQL Shell を対話形式で操作する場合は、ステートメントで Return を押すたびに execute() が暗黙的にコールされます。 その後、操作の結果がフェッチされ、画面に表示されます。 execute() をコールする必要があるかどうかのルールは、次のとおりです:

- この方法で MySQL Shell を使用する場合、execute() のコールはオプションになります:
  - Collection.add()
  - Collection.find()
  - Collection.remove()
  - Collection.modify()
  - Table.insert()
  - Table.select()
  - Table.delete()
  - Table.update()
- オブジェクトが変数に割り当てられている場合、自動実行は無効になります。 このような場合、操作を実行するには execute() をコールする必要があります。
- 行が処理され、関数が使用可能な Result オブジェクトを返すと、結果オブジェクトに含まれる情報が画面に自動的に表示されます。 Result オブジェクトを返す関数には、次のものがあります:
  - SQL 実行および CRUD 操作 (前述)

- mysql モジュールと mysqlx モジュールの両方のセッションオブジェクトのトランザクション処理および削除機能: -
  - startTransaction()
  - コミット()
  - ロールバック()
  - dropSchema()
  - dropCollection()
  - ClassicSession.runSql()

前述のルールに基づいて、コレクション内のドキュメントのセッション、クエリーおよび印刷を確立するために対話モードの MySQL Shell で必要なステートメントは次のとおりです:

```
mysql-js> var mySession = mysqlx.getSession('user:pwd@localhost');
mysql-js> mySession.getSchema('world_x').getCollection('countryinfo').find();
```

execute() をコールする必要はなく、Result オブジェクトが自動的に出力されます。

複数行のサポート

複数行にまたがるステートメントを指定できます。 Python または JavaScript モードでは、関数定義、if/then ステートメント、for ループなどでステートメントのブロックが開始されると、複数行モードが自動的に有効になります。 SQL モードでは、コマンド\が発行されると、複数行モードが開始されます。

複数行モードが開始されると、その後に入力されるステートメントがキャッシュされます。

例:

```
mysql-sql> \
... create procedure get_actors()
... begin
...   select first_name from sakila.actor;
... end
...
```

> **注記**
>
> クエリーで\sql コマンドを使用して、別の言語がアクティブなときに単一の SQL ステートメントを実行する場合、複数行モードは使用できません。 このコマンドは、単一行の単一の SQL クエリーのみを受け入れます。

5.3 コード自動補完

MySQL Shell では、「タブ」キーを押してカーソルの前のテキストの自動補完をサポートしています。 セクション3.1「MySQL Shell のコマンド」 は、任意の言語モードで自動補完できます。 たとえば、\con と入力し、Tab キーを押すと、\connect がオートコンプリートされます。 自動補完は、現在の セクション5.1「アクティブな言語」 に応じて、SQL、JavaScript および Python 言語のキーワードに使用できます。

オートコンプリートでは、次のテキストオブジェクトがサポートされます:

- SQL モード - 自動補完では、現在アクティブなスキーマのスキーマ名、テーブル名、カラム名が認識されます。
- JavaScript および Python モードでは、自動補完はオブジェクトメンバーを認識します。次に例を示します:
  - session, db, dba, shell, mysql, mysqlx などのグローバルオブジェクト名。
  - session.connect()、dba.configureLocalInstance() などのグローバルオブジェクトのメンバー。

- グローバルユーザー定義変数
- shell.options.verbose などの連鎖オブジェクトプロパティ参照。
- col.find().where().execute().fetchOne() などの連鎖した X DevAPI メソッドコール。

デフォルトでは、自動補完は有効になっています。この動作を変更するには、オートコンプリートの構成 を参照してください。

オートコンプリートをアクティブにすると、カーソルの前のテキストが 1 つの一致する可能性がある場合、そのテキストは自動的に入力されます。 オートコンプリートで複数の一致が見つかると、端末がビープ音を鳴らすか点滅します。 Tab キーを再度押すと、使用可能な完了のリストが表示されます。 一致するものが見つからない場合、自動補完は行われません。

自動補完 SQL

MySQL Shell が SQL モードの場合、オートコンプリートは、一致する可能性のあるすべての完了を含む単語を完了しようとします。 SQL モードでは、次を自動完了できます:

- SQL キーワード - 既知の SQL キーワードのリスト。 照合では大文字と小文字は区別されません。
- SQL スニペット - SHOW CREATE TABLE, ALTER TABLE, CREATE TABLE など、特定の共通スニペット。
- テーブル名 - アクティブなスキーマがあり、データベース名キャッシュが無効になっていない場合、アクティブなスキーマのすべてのテーブルが可能な補完として使用されます。

特別な例外として、バックティックが見つかった場合は、テーブル名のみが完了とみなされます。 SQL モードでは、自動補完はコンテキストに対応していません。つまり、SQL 文法に基づく補完のフィルタリングはありません。 つまり、オートコンプリート SEL は SELECT を返しますが、selfies というテーブルを含めることもできます。

JavaScript および Python の自動補完

JavaScript モードと Python モードの両方で、完了する文字列は、「タブ」が押されたときに現在のカーソル位置から右から左に決定されます。 メソッドコール内のコンテンツは無視されますが、構文的に正しい必要があります。 つまり、文字列、コメントおよびネストされたメソッドコールはすべて適切にクローズされ、バランスがとれている必要があります。 これにより、連鎖メソッドを適切に処理できます。 たとえば、次のように発行するとします:

```
print(db.user.select().where("user in ('foo', 'bar')").e
```

「タブ」キーを押すと、自動補完でテキスト db.user.select().where().e の完了が試行されますが、この無効なコードでは未定義の動作が発生します。 . で区切られたトークン間の空白 (改行を含む) は無視されます。

オートコンプリートの構成

デフォルトでは、自動完了エンジンは有効になっています。 このセクションでは、オートコンプリートを無効にする方法および\rehash MySQL Shell コマンドの使用方法について説明します。 オートコンプリートでは、MySQL Shell が認識しているデータベース名オブジェクトのキャッシュが使用されます。 オートコンプリートが有効な場合、この名前キャッシュは自動的に更新されます。 たとえば、スキーマをロードするたびに、自動補完エンジンはスキーマ内のテキストオブジェクトに基づいて名前キャッシュを更新し、テーブル名などをオートコンプリートできるようにします。

この動作を無効にするには、次のようにします:

- --no-name-cache コマンドオプションを使用して MySQL Shell を起動します。
- MySQL Shell の実行中に自動補完が無効になるように、shell.options の autocomplete.nameCache および devapi.dbObjectHandles キーを変更します。

オートコンプリート名キャッシュが無効になっている場合は、\rehash を発行して、テキストオブジェクトのオートコンプリートが認識するように手動で更新できます。 これにより、現在アクティブなスキーマに基づいて名前キャッシュが強制的にリロードされます。

MySQL Shell の実行中にオートコンプリートを無効にするには、次の shell.options キーを使用します:

- autocomplete.nameCache: boolean は、SQL で使用するオートコンプリート名キャッシュを切り替えます。
- devapi.dbObjectHandles: boolean は、db.mytable、db.mycollection などの X DevAPI db オブジェクトで使用するオートコンプリート名キャッシュを切り替えます。

両方のキーがデフォルトで true に設定され、--no-name-cache コマンドオプションが使用されている場合は false に設定されます。 MySQL Shell の実行中に SQL のオートコンプリート名キャッシュを変更するには、次のコマンドを発行します:

```
shell.options['autocomplete.nameCache']=true
```

\rehash コマンドを使用して、名前キャッシュを手動で更新します。

MySQL Shell の実行中に JavaScript および Python のオートコンプリート名キャッシュを変更するには、次のコマンドを発行します:

```
shell.options['devapi.dbObjectHandles']=true
```

再度、\rehash コマンドを使用して、名前キャッシュを手動で更新できます。

5.4 コードの編集

MySQL Shell \edit コマンド (MySQL Shell 8.0.18 から使用可能) は、編集のためにデフォルトのシステムエディタでコマンドを開き、編集したコマンドを MySQL Shell で実行できるように表示します。 このコマンドは、短い形式の\eまたはキーの組合せ Ctrl-X Ctrl-E を使用して呼び出すこともできます。 コマンドに引数を指定すると、このテキストはエディタに配置されます。 引数を指定しない場合、MySQL Shell 履歴の最後のコマンドがエディタに配置されます。

EDITOR および VISUAL 環境変数は、デフォルトのシステムエディタを識別するために使用されます。 これらの環境変数からデフォルトのシステムエディタを識別できない場合、MySQL Shell は Windows では notepad.exe を使用し、その他のプラットフォームでは vi を使用します。 コマンドの編集は一時ファイルで行われ、MySQL Shell は後で削除します。

編集が終了したら、ファイルを保存してエディタを閉じる必要があります。その後、MySQL Shell は、Enter を押して実行する準備ができた編集済テキストを表示します。続行しない場合は、Ctrl-C を押して取り消します。

たとえば、ユーザーはカラムのカスタムセットを使用して MySQL Shell 組込みレポート threads を実行し、システムエディタでコマンドを開き、一部のカラムの表示名を追加します:

```
\show threads --foreground -o tid,cid,user,host,command,state,lastwait,lastwaitl
\e
\show threads --foreground -o tid=thread_id,cid=conn_id,user,host,command,state,lastwait=last_wait_event,lastwaitl=wait_length
```

5.5 コード履歴

MySQL Shell で発行するコードは履歴に格納され、上下の矢印キーを使用してアクセスできます。 増分履歴検索機能を使用して履歴を検索することもできます。 履歴を検索するには、Ctrl+R を使用して逆方向に検索するか、Ctrl+S を使用して履歴を順方向に検索します。 検索がアクティブになったら、文字を入力すると、履歴内で一致する文字列が検索され、最初の一致が表示されます。 Ctrl+S または Ctrl+R を使用して、現在の検索語にさらに一致するものを検索します。 さらに文字を入力すると、検索がさらに絞り込まれます。 検索中に矢印キーを押すと、現在の検索結果から履歴をステップ実行できます。 Enter を押して、表示された一致を受け入れます。 Ctrl+C を使用して検索を取り消します。

history.maxSize MySQL Shell 構成オプションでは、履歴に格納するエントリの最大数を設定します。 デフォルトは 1000 です。 履歴エントリの数が構成されている最大数を超えると、最も古いエントリが削除されて破棄されます。 最大値を 0 に設定すると、履歴エントリは格納されません。

デフォルトでは、履歴はセッション間で保存されないため、MySQL Shell を終了すると、現在のセッション中に発行した履歴は失われます。 MySQL Shell history.autoSave オプションを有効にすると、セッション間の履歴を保存できます。 たとえば、この変更を永続的にするには、次のコマンドを発行します:

```
mysqlsh-js> \option --persist history.autoSave=1
```

history.autoSave オプションを有効にすると、履歴は MySQL Shell 構成パス (Linux および macOS の場合は ~/.mysqlsh ディレクトリ、Windows の場合は %AppData%\MySQL\mysqlsh フォルダ) に格納されます。 このパスは、環境変数 MYSQLSH_USER_CONFIG_HOME を定義することで、すべてのプラットフォームでオーバーライドできます。 保存された履歴は MySQL Shell によって自動的に作成され、所有者ユーザーのみが読み取ることができます。 履歴ファイルの読取りまたは書込みができない場合、MySQL Shell はエラーメッセージをログに記録し、読取りまたは書込み操作をスキップします。 8.0.16 より前のバージョンでは、履歴エントリは、すべての MySQL Shell 言語で発行されたコードを含む単一の history ファイルに保存されていました。 MySQL Shell バージョン 8.0.16 以降では、履歴はアクティブな言語ごとに分割され、ファイルの名前は history.sql、history.js および history.py になります。

MySQL Shell \history コマンドを発行すると、\history delete entry_number コマンドで使用できる履歴エントリ番号とともに、発行された順序で履歴エントリが表示されます。 個々の履歴エントリ、指定した数値範囲の履歴エントリまたは履歴の末尾を手動で削除できます。\history clear を使用して、履歴全体を手動で削除することもできます。MySQL Shell を終了すると、history.autoSave 構成オプションが true に設定されている場合、履歴ファイルに残っている履歴エントリが保存され、その番号付けは 1 から始まるようにリセットされます。 shell.options["history.autoSave"]構成オプションが false(デフォルト) に設定されている場合、履歴ファイルはクリアされます。

MySQL Shell プロンプトで対話形式で入力したコードのみが履歴に追加されます。 間接的または内部的に実行されるコード (\source コマンドの実行時など) は、履歴に追加されません。 複数行コードを発行すると、履歴エントリ内の改行文字が削除されます。 同じコードが複数回発行された場合は、履歴に一度のみ格納されるため、重複が削減されます。

--histignore コマンドオプションを使用して、履歴に追加されるエントリをカスタマイズできます。 また、MySQL Shell を SQL モードで使用する場合は、履歴に追加しない文字列を構成できます。 この履歴無視リストは、\sql コマンドをクエリーとともに使用して、別の言語がアクティブなときに単一の SQL ステートメントを実行する場合にも適用されます。

デフォルトでは、IDENTIFIED または PASSWORD の glob パターンに一致する文字列は履歴に追加されません。 一致する文字列をさらに構成するには、--histignore コマンドオプションまたは shell.options["history.sql.ignorePattern"]を使用します。 コロン (:) で区切って、複数の文字列を指定できます。 履歴照合では、大/小文字を区別しない glob パターン (照合など) が使用されます。 サポートされているワイルドカードは、* (0 文字以上に一致) および ? (1 文字に完全に一致) です。 デフォルトの文字列は"*IDENTIFIED*:*PASSWORD*"として指定されます。

履歴無視リストに設定されているフィルタに関係なく、最後に実行されたステートメントは常に上向き矢印を押してリコールできるため、すべての入力を再入力せずに修正できます。 フィルタリングが最後に実行されたステートメントに適用される場合、別のステートメントが入力されるとすぐに、またはステートメントの実行直後に MySQL Shell を終了すると、そのステートメントは履歴から削除されます。

5.6 バッチコード実行

MySQL Shell では、対話型コードの実行に加えて、次からバッチコードを実行できます:

- 処理のためにロードされたファイル。
- 実行のために標準入力にリダイレクトされるコードを含むファイル。
- 実行のために標準入力にリダイレクトされる別のソースからのコード。

> **ヒント**
>
> ファイルのバッチ実行のかわりに、端末から MySQL Shell を制御することもできます。セクション5.8「API コマンドラインインタフェース」 を参照してください。

バッチモードでは、セクション5.2「対話型コードの実行」 で説明されているすべてのコマンドロジックを使用できず、アクティブな言語の有効なコードのみを実行できます。 SQL コードを処理する場合、次のロジックを使用してステートメントによってステートメントが実行されます: read/process/print result. SQL 以外のコードを処理する場合は、入力ソースから完全にロードされ、ユニットとして実行されます。 --interactive (または -i) コマンドラインオプ

ションを使用して、対話モードで発行されているかのように入力ソースを処理するように MySQL Shell を構成します。これにより、対話モードで提供されるすべての機能をバッチ処理で使用できます。

> 注記
>
> この場合、ソースは行単位で読み取られ、対話型パイプラインを使用して処理されます。

入力は、MySQL Shell で選択された現在のプログラミング言語 (デフォルトは JavaScript) に基づいて処理されます。defaultMode MySQL Shell 構成オプションを使用して、デフォルトのプログラミング言語を変更できます。 拡張子が .js、.py および .sql のファイルは、デフォルトのプログラミング言語に関係なく、常に適切な言語モードで処理されます。

この例では、バッチ処理のためにファイルから JavaScript コードをロードする方法を示します:

```
shell> mysqlsh --file code.js
```

ここでは、JavaScript ファイルが標準入力にリダイレクトされて実行されます:

```
shell> mysqlsh < code.js
```

次の例では、実行のために SQL コードを標準入力にリダイレクトする方法を示します:

```
shell> echo "show databases;" | mysqlsh --sql --uri user@192.0.2.20:33060
```

MySQL Shell 8.0.22 から、--pym コマンドラインオプションを使用して、指定した Python モジュールを Python モードでスクリプトとして実行できます。 このオプションは、Python -m コマンドラインオプションと同様に機能します。

実行可能スクリプト

Linux では、#! 行をスクリプトの最初の行として含めることで、MySQL Shell で実行される実行可能スクリプトを作成できます。 この行には、MySQL Shell へのフルパスを指定し、--file オプションを含める必要があります。 例:

```
#!/usr/local/mysql-shell/bin/mysqlsh --file
print("Hello World\n");
```

スクリプトファイルは、ファイルシステムで実行可能としてマークされている必要があります。 スクリプトを実行すると、MySQL Shell が起動され、スクリプトの内容が実行されます。

スクリプトでの SQL 実行

X プロトコル セッションに対する SQL クエリーの実行では、通常、sql() 関数が使用されます。この関数は、SQL ステートメントを文字列として取り、クエリーのバインドと実行に使用する SqlExecute オブジェクトを返し、結果を返します。 この方法は、Using SQL with Session で説明されています。 ただし、クラシック MySQL プロトコル セッションに対する SQL クエリーの実行では、runSql() 関数を使用します。この関数は、SQL ステートメントとそのパラメータを取得し、指定されたパラメータを指定されたクエリーにバインドし、クエリーを単一のステップで実行して結果を返します。

MySQL サーバーへの接続に使用されるプロトコルに依存しない MySQL Shell スクリプトを作成する必要がある場合、MySQL Shell には X プロトコル 用の session.runSql() 関数が用意されており、クラシック MySQL プロトコル セッションの runSql() 関数と同様に機能します。 この関数は、sql() のかわりに MySQL Shell でのみ使用できるため、スクリプトは X プロトコル セッションまたは クラシック MySQL プロトコル セッションのいずれかで動作します。 Session.runSql() は、クラシック MySQL プロトコル 関数によって返される ClassicResult オブジェクトの仕様に一致する SqlResult オブジェクトを返すため、結果は同じ方法で処理できます。

> 注記
>
> Session.runSql() は、JavaScript および Python の MySQL Shell X DevAPI 実装専用であり、標準 X DevAPI の一部ではありません。

クエリー結果を参照するには、クラシック MySQL プロトコル と X プロトコル の両方で機能する fetchOneObject() 関数を使用できます。 この関数は、次の結果をスクリプトオブジェクトとして返します。 カラム名はディクショナリ

内のキーとして (有効な識別子の場合はオブジェクト属性として) 使用され、行値はディクショナリ内の属性値として使用されます。 オブジェクトに対して行われた更新は、データベースで永続化されません。

たとえば、MySQL Shell スクリプトの次のコードは、X プロトコル セッションまたは クラシック MySQL プロトコル セッションと連携して、特定の国から市区町村の名前を取得および出力します:

```
var resultSet = mySession.runSql("SELECT * FROM city WHERE countrycode = ' AUT'");
var row = resultSet.fetchOneObject();
print(row['Name']);
```

5.7 出力形式

MySQL Shell では、結果をテーブル、タブ付きまたは垂直形式、あるいはプリティまたは RAW JSON 出力として出力できます。 MySQL Shell 8.0.14 から、MySQL Shell 構成オプション resultFormat を使用して、これらの出力形式のいずれかをすべてのセッションまたは現在のセッションの永続的なデフォルトとして指定できます。 このオプションの変更はすぐに有効になります。 MySQL Shell の構成オプションを設定する手順は、セクション10.4「MySQL Shell オプションの構成」 を参照してください。 または、コマンドラインオプション --result-format またはそのエイリアス (--table, --tabbed, --vertical) を起動時に使用して、セッションの出力形式を指定できます。 コマンドラインオプションのリストは、セクションA.1「mysqlsh — MySQL Shell」 を参照してください。

resultFormat 構成オプションが指定されていない場合、MySQL Shell が対話モードのとき、結果セットを印刷するためのデフォルトの書式はフォーマットされたテーブルであり、MySQL Shell がバッチモードのとき、結果セットを印刷するためのデフォルトの書式はタブ区切りの出力です。 resultFormat 構成オプションを使用してデフォルトを設定すると、このデフォルトは対話モードとバッチモードの両方で適用されます。

MySQL Shell 関数 shell.dumpRows() では、MySQL Shell でサポートされている任意の出力形式でクエリーによって返された結果セットを書式設定し、コンソールにダンプできます。 (結果セットは関数によって使用されることに注意してください。)

MySQL Shell を外部ツールと統合するには、--json オプションを使用して、コマンドラインから MySQL Shell を起動したときのすべての MySQL Shell 出力の JSON ラッピングを制御できます。 JSON ラッピングがオンの場合、MySQL Shell は整形出力 JSON (デフォルト) または RAW JSON を生成し、resultFormat MySQL Shell 構成オプションの値は無視されます。 JSON ラッピングがオフになっているか、セッションに対してリクエストされなかった場合、結果セットは resultFormat 構成オプションで指定された形式で通常どおりに出力されます。

outputFormat 構成オプションは非推奨になりました。 このオプションは、JSON ラッピング関数と結果出力関数を組み合せたものです。 このオプションが MySQL Shell 構成ファイルまたはスクリプトでまだ指定されている場合、動作は次のようになります:

- json または json/raw の値を使用すると、outputFormat は JSON ラップをプリティ JSON または RAW JSON でそれぞれアクティブ化します。
- table、tabbed または vertical の値を使用すると、outputFormat は JSON ラッピングをオフにし、セッションの resultFormat 構成オプションを適切な値に設定します。

5.7.1 テーブル形式

MySQL Shell が対話モードの場合、結果セットの出力にはテーブル形式がデフォルトで使用されます。 クエリーの結果は、ビューを改善して分析を支援するために書式設定されたテーブルとして表示されます。

バッチモードでの実行時にこの出力形式を取得するには、--result-format=table コマンドラインオプション (またはそのエイリアス --table) を使用して MySQL Shell を起動するか、MySQL Shell 構成オプション resultFormat を table に設定します。

例 5.1 テーブル形式での出力

```
MySQL  localhost:33060+ ssl  world_x  JS > shell.options.set('resultFormat','table')
MySQL  localhost:33060+ ssl  world_x  JS > session.sql("select * from city where countrycode='AUT'")
+------+------------+-------------+---------------+------------------------+
| ID   | Name       | CountryCode | District      | Info                   |
+------+------------+-------------+---------------+------------------------+
| 1523 | Wien       | AUT         | Wien          | {"Population": 1608144} |
| 1524 | Graz       | AUT         | Steiermark    | {"Population": 240967}  |
```

```
| 1525 | Linz       | AUT         | North Austria | {"Population": 188022}  |
| 1526 | Salzburg   | AUT         | Salzburg      | {"Population": 144247}  |
| 1527 | Innsbruck  | AUT         | Tiroli        | {"Population": 111752}  |
| 1528 | Klagenfurt | AUT         | Kärnten       | {"Population": 91141}   |
+------+------------+-------------+---------------+-------------------------+
6 rows in set (0.0030 sec)
```

5.7.2 タブ区切り形式

MySQL Shell をバッチモードで実行している場合は、デフォルトでタブ区切り形式が結果セットの印刷に使用され、自動分析の出力が向上します。

対話モードで実行しているときにこの出力形式を取得するには、--result-format=tabbed コマンドラインオプション (またはその エイリアス --tabbed) を使用して MySQL Shell を起動するか、MySQL Shell 構成オプション resultFormat を tabbed に設定します。

例 5.2 タブ区切り形式での出力

```
MySQL  localhost:33060+ ssl  world_x  JS > shell.options.set('resultFormat','tabbed')
MySQL  localhost:33060+ ssl  world_x  JS > session.sql("select * from city where countrycode='AUT'")
ID      Name    CountryCode     District        Info
1523    Wien    AUT     Wien    {"Population": 1608144}
1524    Graz    AUT     Steiermark      {"Population": 240967}
1525    Linz    AUT     North Austria   {"Population": 188022}
1526    Salzburg        AUT     Salzburg        {"Population": 144247}
1527    Innsbruck       AUT     Tiroli  {"Population": 111752}
1528    Klagenfurt      AUT     Kärnten {"Population": 91141}
6 rows in set (0.0041 sec)
```

5.7.3 垂直フォーマット

垂直フォーマットオプションでは、\G クエリー終了記号を SQL クエリーに使用する場合と同じ方法で、水平テーブルではなく垂直方向に結果セットが出力されます。 縦方向の書式は、長いテキスト行が出力の一部であるほど読みやすくなります。

この出力形式を取得するには、--result-format=vertical コマンドラインオプション (またはそのエイリアス --vertical) を使用して MySQL Shell を起動するか、MySQL Shell 構成オプション resultFormat を vertical に設定します。

例 5.3 垂直形式での出力

```
MySQL  localhost:33060+ ssl  world_x  JS > shell.options.set('resultFormat','vertical')
MySQL  localhost:33060+ ssl  world_x  JS > session.sql("select * from city where countrycode='AUT'")
*************************** 1. row ***************************
       ID: 1523
     Name: Wien
CountryCode: AUT
 District: Wien
     Info: {"Population": 1608144}
*************************** 2. row ***************************
       ID: 1524
     Name: Graz
CountryCode: AUT
 District: Steiermark
     Info: {"Population": 240967}
*************************** 3. row ***************************
       ID: 1525
     Name: Linz
CountryCode: AUT
 District: North Austria
     Info: {"Population": 188022}
*************************** 4. row ***************************
       ID: 1526
     Name: Salzburg
CountryCode: AUT
 District: Salzburg
     Info: {"Population": 144247}
*************************** 5. row ***************************
       ID: 1527
     Name: Innsbruck
```

```
CountryCode: AUT
  District: Tiroli
     Info: {"Population": 111752}
*************************** 6. row ***************************
        ID: 1528
      Name: Klagenfurt
CountryCode: AUT
  District: Kärnten
     Info: {"Population": 91141}
6 rows in set (0.0027 sec)
```

5.7.4 JSON 形式の出力

MySQL Shell には、結果セットを出力するためのいくつかの JSON 形式オプションが用意されています:

json または json/pretty　　これらのオプションは、どちらも整形出力 JSON を生成します。

ndjson または json/raw　　これらのオプションはどちらも改行で区切られた RAW JSON を生成します。

json/array　　このオプションは、JSON 配列にラップされた RAW JSON を生成します。

これらの出力形式を選択するには、MySQL Shell を --result-format=value コマンドラインオプションで起動するか、MySQL Shell 構成オプション resultFormat を設定します。

バッチモードでは、MySQL Shell を外部ツールと統合するために、--json オプションを使用して、コマンドラインから MySQL Shell を起動するときのすべての出力の JSON ラッピングを制御できます。 JSON ラッピングがオンの場合、MySQL Shell は整形出力 JSON (デフォルト) または RAW JSON を生成し、resultFormat MySQL Shell 構成オプションの値は無視されます。 その手順は、セクション5.7.5「JSON ラッピング」を参照してください。

例 5.4 整形出力 JSON 形式での出力 (json または json/pretty)

```
MySQL  localhost:33060+ ssl  world_x  JS > shell.options.set('resultFormat','json')
MySQL  localhost:33060+ ssl  world_x  JS > session.sql("select * from city where countrycode='AUT'")
{
    "ID": 1523,
    "Name": "Wien",
    "CountryCode": "AUT",
    "District": "Wien",
    "Info": {
        "Population": 1608144
    }
}
{
    "ID": 1524,
    "Name": "Graz",
    "CountryCode": "AUT",
    "District": "Steiermark",
    "Info": {
        "Population": 240967
    }
}
{
    "ID": 1525,
    "Name": "Linz",
    "CountryCode": "AUT",
    "District": "North Austria",
    "Info": {
        "Population": 188022
    }
}
{
    "ID": 1526,
    "Name": "Salzburg",
    "CountryCode": "AUT",
    "District": "Salzburg",
    "Info": {
        "Population": 144247
    }
}
{
    "ID": 1527,
```

```
    "Name": "Innsbruck",
    "CountryCode": "AUT",
    "District": "Tiroli",
    "Info": {
        "Population": 111752
    }
}
{
    "ID": 1528,
    "Name": "Klagenfurt",
    "CountryCode": "AUT",
    "District": "Kärnten",
    "Info": {
        "Population": 91141
    }
}
6 rows in set (0.0031 sec)
```

例 5.5 改行デリミタを使用した RAW JSON 形式での出力 (ndjson または json/raw)

```
MySQL  localhost:33060+ ssl  world_x  JS > shell.options.set('resultFormat','ndjson')
MySQL  localhost:33060+ ssl  world_x  JS > session.sql("select * from city where countrycode='AUT'")
{"ID":1523,"Name":"Wien","CountryCode":"AUT","District":"Wien","Info":{"Population":1608144}}
{"ID":1524,"Name":"Graz","CountryCode":"AUT","District":"Steiermark","Info":{"Population":240967}}
{"ID":1525,"Name":"Linz","CountryCode":"AUT","District":"North Austria","Info":{"Population":188022}}
{"ID":1526,"Name":"Salzburg","CountryCode":"AUT","District":"Salzburg","Info":{"Population":144247}}
{"ID":1527,"Name":"Innsbruck","CountryCode":"AUT","District":"Tiroli","Info":{"Population":111752}}
{"ID":1528,"Name":"Klagenfurt","CountryCode":"AUT","District":"Kärnten","Info":{"Population":91141}}
6 rows in set (0.0032 sec)
```

例 5.6 JSON 配列でラップされた RAW JSON 形式の出力 (json/array)

```
MySQL  localhost:33060+ ssl  world_x  JS > shell.options.set('resultFormat','json/array')
MySQL  localhost:33060+ ssl  world_x  JS > session.sql("select * from city where countrycode='AUT'")
[
{"ID":1523,"Name":"Wien","CountryCode":"AUT","District":"Wien","Info":{"Population":1608144}},
{"ID":1524,"Name":"Graz","CountryCode":"AUT","District":"Steiermark","Info":{"Population":240967}},
{"ID":1525,"Name":"Linz","CountryCode":"AUT","District":"North Austria","Info":{"Population":188022}},
{"ID":1526,"Name":"Salzburg","CountryCode":"AUT","District":"Salzburg","Info":{"Population":144247}},
{"ID":1527,"Name":"Innsbruck","CountryCode":"AUT","District":"Tiroli","Info":{"Population":111752}},
{"ID":1528,"Name":"Klagenfurt","CountryCode":"AUT","District":"Kärnten","Info":{"Population":91141}}
]
6 rows in set (0.0032 sec)
```

5.7.5 JSON ラッピング

MySQL Shell を外部ツールと統合するには、--json オプションを使用して、コマンドラインから MySQL Shell を起動したときのすべての MySQL Shell 出力の JSON ラッピングを制御できます。 --json オプションは、指定されている MySQL Shell セッションに対してのみ有効になります。

--json、--json=pretty または --json=raw を指定すると、セッションの JSON ラッピングが有効になります。 --json=pretty を使用するか、値を指定しない場合、整形出力 JSON が生成されます。 --json=raw では、RAW JSON が生成されます。

JSON ラッピングがオンの場合、構成ファイルまたはコマンドライン (--result-format オプションまたはそのエイリアスのいずれかを使用) で resultFormat MySQL Shell 構成オプションに指定された値は無視されます。

--json=off を指定すると、セッションの JSON ラッピングがオフになります。 JSON ラッピングがオフになっているか、セッションに対してリクエストされなかった場合、結果セットは resultFormat MySQL Shell 構成オプションで指定された形式で通常どおりに出力されます。

例 5.7 プリティ印刷 JSON ラッピングを使用した MySQL Shell 出力 (--json または --json=pretty)

```
shell> echo "select * from world_x.city where countrycode='AUT'" | mysqlsh --json --sql --uri user@localhost:33060
or
shell> echo "select * from world_x.city where countrycode='AUT'" | mysqlsh --json=pretty --sql --uri user@localhost:33060
{
    "hasData": true,
    "rows": [
        {
```

```
        "ID": 1523,
        "Name": "Wien",
        "CountryCode": "AUT",
        "District": "Wien",
        "Info": {
            "Population": 1608144
        }
    },
    {
        "ID": 1524,
        "Name": "Graz",
        "CountryCode": "AUT",
        "District": "Steiermark",
        "Info": {
            "Population": 240967
        }
    },
    {
        "ID": 1525,
        "Name": "Linz",
        "CountryCode": "AUT",
        "District": "North Austria",
        "Info": {
            "Population": 188022
        }
    },
    {
        "ID": 1526,
        "Name": "Salzburg",
        "CountryCode": "AUT",
        "District": "Salzburg",
        "Info": {
            "Population": 144247
        }
    },
    {
        "ID": 1527,
        "Name": "Innsbruck",
        "CountryCode": "AUT",
        "District": "Tiroli",
        "Info": {
            "Population": 111752
        }
    },
    {
        "ID": 1528,
        "Name": "Klagenfurt",
        "CountryCode": "AUT",
        "District": "Kärnten",
        "Info": {
            "Population": 91141
        }
    }
  ],
  "executionTime": "0.0067 sec",
  "affectedRowCount": 0,
  "affectedItemsCount": 0,
  "warningCount": 0,
  "warningsCount": 0,
  "warnings": [],
  "info": "",
  "autoIncrementValue": 0
}
```


例 5.8 RAW JSON ラップを使用した MySQL Shell 出力 (--json=raw)


```
shell> echo "select * from world_x.city where countrycode='AUT'" | mysqlsh --json=raw --sql --uri user@localhost:33060
{"hasData":true,"rows":[{"ID":1523,"Name":"Wien","CountryCode":"AUT","District":"Wien","Info":{"Population":1608144}},{"ID":1524,"Name":"Graz","CountryCode":"AUT
```


5.7.6 結果メタデータ

操作が実行されると、返される結果に加えて、いくつかの追加情報が返されます。 これには、次のいずれかの条件に該当する場合に、影響を受ける行の数、警告、期間などの情報が含まれます:

- JSON 形式が出力に使用されています
- MySQL Shell は対話モードで実行されています。

出力には「When JSON」形式が使用され、メタデータは JSON オブジェクトの一部として返されます。 対話型モードでは、メタデータは結果の後に出力されます。

5.8 API コマンドラインインタフェース

MySQL Shell では、mysqlsh を他のツールと簡単に統合できる API コマンド構文を使用して、その機能の多くを公開しています。 この機能は --execute オプションの使用と似ていますが、コマンドインタフェースでは簡略化された引数構文が使用され、端末で必要になる可能性のある引用符およびエスケープが削減されます。 たとえば、bash スクリプトを使用して InnoDB クラスタ を作成する場合は、この機能を使用できます。

次の組込み MySQL Shell グローバルオブジェクトを使用できます:

- session - 現在のグローバルセッションを表します。
- db - デフォルトデータベースが指定された X プロトコル 接続を使用してグローバルセッションが確立された場合、そのセッションのデフォルトデータベースを表します。
- cluster - InnoDB クラスタ を表します。
- dba - AdminAPI を使用した InnoDB クラスタ管理機能へのアクセスを提供します。 第6章「MySQL AdminAPI の使用」を参照してください。
- shell - global は、MySQL Shell オプションを構成するための shell.options (セクション10.4「MySQL Shell オプションの構成」 を参照)、および MySQL Shell レポートを実行するための shell.reports (セクション7.1「MySQL Shell でのレポート」 を参照) などの MySQL Shell 機能へのアクセスを提供します。
- util - MySQL Shell ユーティリティへのアクセスを提供します。 第8章「MySQL Shell ユーティリティ」を参照してください。

API コマンドライン統合構文

次の特別な構文を使用してコマンドラインで MySQL Shell を起動すると、-- はオプションのリストの最後と、コマンドおよびその引数として扱われた後のすべてを示します。

```
mysqlsh [options] -- shell_object object_method [arguments]
```

ここで、次のことが適用されます:

- shell_object は、MySQL Shell グローバルオブジェクトにマップされる文字列です。
- object_method は、shell_object によって提供されるメソッドの名前です。 メソッド名は、JavaScript、Python または代替コマンドライン入力のフレンドリ形式のいずれかに従って指定できます。既知のメソッドはすべて小文字を使用し、単語はハイフンで区切られます。 object_method の名前は、標準の JavaScript スタイルの camelCase 名から自動的に変換されます。この場合、大/小文字の変更はすべて - に置き換えられ、小文字になります。 たとえば、getCluster は get-cluster になります。
- arguments は、コール時に object_method に渡される引数です。

shell_object は公開されたグローバルオブジェクトのいずれかと一致する必要があり、object_method は有効な形式 (JavaScript、Python またはコマンドライン対応) のいずれかのグローバルオブジェクトメソッドと一致する必要があります。 有効なグローバルオブジェクトとそのメソッドに対応していない場合、MySQL Shell はステータス 10 で終了します。

API コマンドライン統合引数の構文

arguments リストはオプションで、すべての引数は、このセクションで説明するコマンドラインでの使用に適した構文に従う必要があります。 たとえば、システムシェル (bash、cmd など) によって処理される特殊文字は避けてくだ

さい。引用符が必要な場合は、親シェルの引用符のみを考慮するようにしてください。 つまり、「foo bar」 が bash でパラメータとして使用されている場合、引用符は削除され、エスケープが処理されます。

引数のリストでは、2 つのタイプの引数を使用できます: 位置指定引数と名前付き引数。 位置引数は、文字列、数値、ブール、null などの単純な型です。 名前付き引数はキーと値のペアで、値は単純な型です。 使用方法は、次のパターンに従う必要があります:

```
[ positional_argument ]* [ { named_argument* } ]* [ named_argument ]*
```

この構文を使用するためのルールは次のとおりです:

- 構文のすべての部分はオプションであり、任意の順序で指定できます
- 中カッコのネストは禁止されています
- 名前付き引数として指定するすべてのキー値は、スコープ内で一意の名前を持つ必要があります。 スコープは、グループ化されていないか、グループ内 (中カッコ内) にあります。

これらの引数は、次の方法でメソッドコールに渡される引数に変換されます:

- グループ化されていない名前付き引数はすべて、それらが出現する場所に関係なく単一のディクショナリに結合され、最後のパラメータとしてメソッドに渡されます
- 中カッコ内にグループ化された名前付き引数は、単一のディクショナリに結合されます
- グループ化された名前付き引数によって生成された位置指定引数およびディクショナリは、コマンドラインに表示された順序で arguments リストに挿入されます

API インタフェースの例

API 統合を使用すると、--execute オプションを使用するよりも、MySQL Shell コマンドをコールする方が簡単で扱いにくくなります。 次の例では、この機能の使用方法を示します:

- サーバーインスタンスがアップグレードに適していることを確認し、その後の処理のために結果を JSON として返すには:

  ```
  $ mysqlsh -- util check-for-server-upgrade { --user=root --host=localhost --port=3301 } --password='password' --outputFormat=JSON --config-path=/etc/mysql/my.cn
  ```

  これは、MySQL Shell の同等のコマンドにマップされます:

  ```
  mysql-js> util.checkForServerUpgrade({user:'root', host:'localhost', port:3301}, {password:'password', outputFormat:'JSON', configPath:'/etc/mysql/my.cnf'})
  ```

- InnoDB クラスタ サンドボックスインスタンスをデプロイするには、ポート 1234 でリスニングし、接続に使用するパスワードを指定します:

  ```
  $ mysqlsh -- dba deploy-sandbox-instance 1234 --password=password
  ```

  これは、MySQL Shell の同等のコマンドにマップされます:

  ```
  mysql-js> dba.deploySandboxInstance(1234, {password: password})
  ```

- ポート 1234 でリスニングし、mycluster という名前を指定してサンドボックスインスタンスを使用して InnoDB クラスタ を作成するには:

  ```
  $ mysqlsh root@localhost:1234 -- dba create-cluster mycluster
  ```

  これは、MySQL Shell の同等のコマンドにマップされます:

  ```
  mysql-js> dba.createCluster('mycluster')
  ```

- ポート 1234 でリスニングしているサンドボックスインスタンスを使用して InnoDB クラスタ のステータスを確認するには:

  ```
  $ mysqlsh root@localhost:1234 -- cluster status
  ```

  これは、MySQL Shell の同等のコマンドにマップされます:

```
mysql-js> cluster.status()
```

- コマンド履歴をオンにするように MySQL Shell を構成するには:

```
$ mysqlsh -- shell.options set_persist history.autoSave true
```

これは、MySQL Shell の同等のコマンドにマップされます:

```
mysql-js> shell.options.set_persist('history.autoSave', true);
```

第 6 章 MySQL AdminAPI の使用

目次

6.1 MySQL AdminAPI 47
6.2 MySQL InnoDB クラスタ 54
6.2.1 MySQL InnoDB クラスタ の要件 55
6.2.2 本番 InnoDB クラスタ のデプロイ 56
6.2.3 InnoDB クラスタ の監視 68
6.2.4 インスタンスの操作 77
6.2.5 InnoDB クラスタの操作 79
6.2.6 InnoDB クラスタ の構成 82
6.2.7 InnoDB クラスタ のトラブルシューティング 87
6.2.8 InnoDB クラスタ のアップグレード 91
6.2.9 メタデータのタグ付け 94
6.2.10 InnoDB クラスタ のヒント 97
6.2.11 既知の制限事項 100
6.3 MySQL InnoDB ReplicaSet 101
6.3.1 InnoDB ReplicaSet の概要 101
6.3.2 InnoDB ReplicaSet のデプロイ 102
6.3.3 ReplicaSet へのインスタンスの追加 104
6.3.4 既存のレプリケーション設定の採用 106
6.3.5 InnoDB ReplicaSet の操作 107
6.4 MySQL Router 110
6.4.1 MySQL Router のブートストラップ 110
6.4.2 AdminAPI および MySQL Router の使用 113
6.5 AdminAPI MySQL サンドボックス 115

この章では、MySQL Shell で提供される MySQL AdminAPI について説明します。この AdminAPI を使用すると、InnoDB クラスタ、InnoDB ReplicaSet、および MySQL Router の統合のために MySQL インスタンスを管理できます。

6.1 MySQL AdminAPI

このセクションでは、AdminAPI の概要と開始に必要な知識について説明します。

MySQL Shell には、dba グローバル変数とそれに関連付けられたメソッドを介してアクセスする AdminAPI が含まれています。 dba 変数メソッドは、InnoDB クラスタ および InnoDB ReplicaSet のデプロイ、構成および管理を可能にする操作を提供します。 たとえば、dba.createCluster() メソッドを使用して InnoDB クラスタ を作成します。 また、AdminAPI では、InnoDB クラスタ と InnoDB ReplicaSet の統合を可能にするユーザーの作成や更新など、一部の MySQL Router 関連タスクの管理がサポートされています。

MySQL Shell には、ネイティブ SQL モードに加えて、JavaScript および Python のスクリプト言語モードが用意されています。 このガイド全体を通して、MySQL Shell は主に JavaScript モードで使用されます。 MySQL Shell を起動すると、デフォルトで JavaScript モードになります。 JavaScript モードの場合は\js、Python モードの場合は\py を発行して、モードを切り替えます。 \js を発行して、JavaScript モードであることを確認します。

> **重要**
>
> MySQL Shell ではソケット接続を介してサーバーに接続できますが、AdminAPI ではサーバーインスタンスへの TCP 接続が必要です。 ソケットベースの接続は AdminAPI ではサポートされていません。

このセクションでは、MySQL Shell について理解していることを前提としています。詳細は、MySQL Shell 8.0 を参照してください。MySQL Shell では、AdminAPI のオンラインヘルプも提供されます。 使用可能なすべての dba コマ

ンドをリストするには、dba.help() メソッドを使用します。 特定の方法のオンラインヘルプを参照するには、一般的な形式の object.help('methodname') を使用します。 例:

```
mysql-js> dba.help('getCluster')

Retrieves a cluster from the Metadata Store.

SYNTAX

  dba.getCluster([name][, options])

WHERE

  name: Parameter to specify the name of the cluster to be returned.
  options: Dictionary with additional options.

>trimmed for brevity<
```

このドキュメントに加えて、MySQL Shell JavaScript API リファレンスまたは MySQL Shell Python API リファレンス (「コネクタおよび API」から入手可能) のすべての AdminAPI メソッドの開発者用ドキュメントがあります。

このセクションは、InnoDB クラスタ または InnoDB ReplicaSet の使用に適用され、次のもので構成されます:

- デプロイメントシナリオ
- コンポーネントのインストール
- ホスト名の構成
- インスタンスの指定
- 設定の永続化
- ハンドラオブジェクトの取得
- 管理用のユーザーアカウントの作成
- 詳細ロギング
- プライマリの検索
- スクリプト AdminAPI

デプロイメントシナリオ

AdminAPI では、次のデプロイメントシナリオがサポートされます:

- 本番デプロイメント: 完全な本番環境を使用する場合は、必要な数のマシンを構成してから、サーバーインスタンスをマシンにデプロイする必要があります。
- サンドボックスのデプロイメント: 完全本番デプロイメントにコミットする前にデプロイメントをテストする場合、提供されているサンドボックス機能を使用すると、ローカルマシンにテスト環境をすばやく設定できます。 サンドボックスサーバーインスタンスは必要な構成で作成され、採用されているテクノロジを試すことができます。

> **重要**
>
> サンドボックスデプロイメントは、完全本番環境での使用には適していません。

コンポーネントのインストール

AdminAPI に必要なソフトウェアコンポーネントのインストール方法は、使用するデプロイメントのタイプによって異なります。 本番デプロイメントの場合は、各マシンにコンポーネントをインストールします。 本番デプロイメントでは、MySQL サーバーインスタンスを実行している複数のリモートホストマシンを使用するため、コンポーネントのインストールなどのタスクを実行するには、SSH や Windows リモートデスクトップなどのツールを使用して各マシンに接続する必要があります。 サンドボックスデプロイメントの場合は、コンポーネントを単一のマシンにインス

トールします。 サンドボックスデプロイメントは単一のマシンに対してローカルであるため、インストールはローカルマシンで一度のみ実行する必要があります。 次のインストール方法を使用できます:

次のドキュメントを使用して、コンポーネントをダウンロードおよびインストールします:

- MySQL Server - MySQL のインストールとアップグレード を参照してください。
- MySQL Shell - 第2章「MySQL Shell のインストール」 を参照してください。
- MySQL Router - Installing MySQL Router を参照してください。

> **重要**
>
> 一致するバージョンのコンポーネントを常に使用します。たとえば、MySQL Router 8.0.29 とともに MySQL 8.0.29 を実行するインスタンスを管理するには、MySQL Shell 8.0.29 を実行します。

必要なソフトウェアをインストールしたら、セクション6.2「MySQL InnoDB クラスタ」 または セクション6.3「MySQL InnoDB ReplicaSet」 のどちらに従うかを選択します。

ホスト名の構成

本番デプロイメントでは、使用するインスタンスは個別のマシンで実行されるため、各マシンには一意のホスト名が必要であり、サーバーインスタンスを実行する他のマシンのホスト名を解決できる必要があります。 そうでない場合は、次のことができます:

- 各マシンを構成して、相互の IP をホスト名にマップします。 詳細は、オペレーティングシステムのドキュメントを参照してください。 これは推奨される解決策です。
- DNS サービスの設定
- 各インスタンスの MySQL 構成の report_host 変数を、適切な外部から到達可能なアドレスに構成

AdminAPI では、ホスト名のかわりに IP アドレスを使用できます。 MySQL Shell 8.0.18 から、ターゲット MySQL Server のバージョンが 8.0.13 より高い場合、AdminAPI は IPv6 アドレスをサポートします。 MySQL Shell 8.0.18 以上を使用している場合、すべてのクラスタインスタンスで 8.0.14 以上が実行されていると、IPv6 またはホスト名を使用して、インスタンス接続文字列および localAddress、groupSeeds、ipAllowlist などのオプションを指定して IPv6 アドレスに解決できます。 IPv6 の使用の詳細は、IPv6 および IPv6 と IPv4 の混合グループのサポート を参照してください。 以前のバージョンでは、IPv4 アドレスのみサポートされていました。

MySQL サーバーのホスト名が正しく構成されているかどうかを確認するには、次のクエリーを実行して、インスタンスが他のサーバーに自身のアドレスをレポートする方法を確認し、返されたアドレスを使用して他のホストからその MySQL サーバーへの接続を試行します:

```
SELECT coalesce(@@report_host, @@hostname);
```

インスタンスの指定

AdminAPI を使用するコア概念の 1 つは、InnoDB クラスタ または InnoDB ReplicaSet を構成する MySQL インスタンスへの接続を理解することです。 管理時のインスタンスへの接続およびインスタンス間の接続の要件は、次のとおりです:

- TCP/IP 接続のみがサポートされ、Unix ソケットまたは名前付きパイプの使用はサポートされていません。InnoDB クラスタ および InnoDB ReplicaSet は、広域ネットワーク上で実行されているローカルエリアネットワークでの使用を目的としていますが、お薦めしません。
- クラシック MySQL プロトコル 接続のみがサポートされており、X プロトコル はサポートされていません。

> **ヒント**
>
> アプリケーションで X プロトコル を使用できます。この要件は、AdminAPI を使用した管理操作用です。

MySQL Shell を使用すると、様々な API を操作でき、URI 類似文字列またはキーと値のペアを使用したサーバーへの接続 で説明されているように接続の指定がサポートされます。 URI のような文字列またはキーと値のペアを使用して接続を指定できます。 追加の接続パラメータ は AdminAPI ではサポートされていません。 このドキュメントでは、URI のような接続文字列を使用した AdminAPI を示します。 たとえば、ユーザー myuser として www.example.com の MySQL サーバーインスタンスに接続するには、3306 のポートで接続文字列を使用します:

```
myuser@www.example.com:3306
```

この接続文字列を dba.configureInstance() などの AdminAPI 操作で使用するには、接続文字列を一重引用符 (') または二重引用符 (\") で囲むなどして、接続文字列を文字列として解釈する必要があります。 AdminAPI の JavaScript 実装を使用している場合には、次のコマンドを発行します:

```
MySQL JS > dba.configureInstance('myuser@www.example.com:3306')
```

MySQL Shell をデフォルトの対話モードで実行している場合は、パスワードの入力を求められます。AdminAPI では MySQL Shell セクション4.4「プラガブルパスワードストア」 がサポートされており、インスタンスへの接続に使用したパスワードを格納すると、プロンプトは表示されなくなります。

設定の永続化

InnoDB クラスタ、InnoDB ReplicaSet およびそのサーバーインスタンスを操作するために使用する AdminAPI コマンドによって、インスタンス上の MySQL の構成が変更されます。 MySQL Shell のインスタンスへの接続方法およびインスタンスにインストールされている MySQL のバージョンに応じて、これらの構成変更をインスタンスに自動的に永続化できます。 インスタンスに設定を永続化すると、インスタンスの再起動後に構成の変更が保持されます。バックグラウンド情報は、SET PERSIST を参照してください。 これは、信頼性のある使用のために不可欠です。たとえば、設定が永続化されていない場合、構成の変更が失われるため、再起動後にクラスタに追加されたインスタンスは再結合されません。

次の要件を満たすインスタンスでは、構成変更の永続化が自動的にサポートされます:

- インスタンスで MySQL バージョン 8.0.11 以上が実行されている
- persisted_globals_load が ON に設定されている
- インスタンスが --no-defaults オプションで起動されていません

これらの要件を満たさないインスタンスは、構成変更の永続化を自動的にサポートしておらず、AdminAPI 操作によってインスタンス設定の変更が永続化されると、次のような警告が表示されます:

```
WARNING: On instance 'localhost:3320' membership change cannot be persisted since MySQL version 5.7.21
does not support the SET PERSIST command (MySQL version >= 8.0.5 required). Please use the
<Dba>.configureLocalInstance command locally to persist the changes.
```

MySQL Shell が現在実行されている MySQL インスタンス (つまり、ローカルインスタンス) に対して AdminAPI コマンドが発行されると、MySQL Shell は構成の変更をインスタンスに直接保持します。 構成変更の自動永続化をサポートするローカルインスタンスでは、構成変更はインスタンス mysqld-auto.cnf ファイルに永続化され、構成変更にそれ以上のステップは必要ありません。 構成変更の自動永続化をサポートしていないローカルインスタンスでは、変更をローカルで行う必要があります。dba.configureLocalInstance() でのインスタンスの構成 を参照してください。

リモートインスタンス (つまり、MySQL Shell が現在実行されているインスタンス以外のインスタンス) に対して実行する場合、インスタンスが構成変更の永続化を自動的にサポートしている場合、AdminAPI コマンドは、インスタンスの mysql-auto.conf オプションファイルに対する構成変更を永続化します。 リモートインスタンスが構成変更の永続化を自動的にサポートしていない場合、AdminAPI コマンドはインスタンスオプションファイルを自動的に構成できません。 つまり、AdminAPI コマンドは、現在の構成を表示するためなど、インスタンスから情報を読み取ることができますが、構成への変更をインスタンスオプションファイルに永続化することはできません。 この場合、変更をローカルに永続化する必要があります。dba.configureLocalInstance() でのインスタンスの構成 を参照してください。

ハンドラオブジェクトの取得

AdminAPI を使用している場合、InnoDB クラスタ または InnoDB ReplicaSet を表すハンドラオブジェクトを使用します。 このオブジェクトを変数に割り当て、使用可能な操作を使用して InnoDB クラスタ または InnoDB ReplicaSet

を監視および管理します。 ハンドラオブジェクトを取得できるようにするには、InnoDB クラスタ または InnoDB ReplicaSet に属するいずれかのインスタンスへの接続を確立します。 たとえば、dba.createCluster() を使用してクラスタを作成する場合、この操作は変数に割り当てることができる Cluster オブジェクトを返します。 このオブジェクトを使用して、インスタンスの追加やクラスタステータスの確認など、クラスタを操作します。 MySQL Shell の再起動後など、後日クラスタを再度取得する場合は、dba.getCluster([name],[options]) 関数を使用します。 例:

```
mysql-js> var cluster1 = dba.getCluster()
```

同様に、dba.getReplicaSet() 操作を使用して InnoDB ReplicaSet を取得します。 例:

```
mysql-js> var replicaset1 = dba.getReplicaSet()
```

name を指定しない場合、デフォルトオブジェクトが返されます。 デフォルトでは、MySQL Shell はハンドラの取得時にプライマリインスタンスへの接続を試行します。 この動作を構成するには、connectToPrimary オプションを設定します。 connectToPrimary が true で、アクティブなグローバル MySQL Shell セッションがプライマリインスタンスに対するものでない場合、MySQL Shell はプライマリインスタンスに対するクエリーを実行します。 クラスタにクォーラムがない場合、操作は失敗します。 connectToPrimary が false の場合、取得されたオブジェクトはアクティブセッション、つまり MySQL Shell の現在のグローバルセッションと同じインスタンスを使用します。 connectToPrimary が指定されていない場合、MySQL Shell は connectToPrimary を true として扱い、false である connectToPrimary にフォールバックします。

セカンダリに強制的に接続するには、セカンダリインスタンスへの接続を確立し、次を発行して connectToPrimary オプションを使用します:

```
mysql-js> shell.connect(secondary_member)
mysql-js> var cluster1 = dba.getCluster(testCluster, {connectToPrimary:false})
```

> **ヒント**
>
> セカンダリインスタンスには super_read_only=ON があるため、変更を書き込むことはできません。

管理用のユーザーアカウントの作成

インスタンスの管理に使用されるユーザーアカウントは root アカウントである必要はありませんが、完全な MySQL 管理者権限 (SUPER, GRANT OPTION, CREATE, DROP など) に加えて、メタデータテーブルに対する完全な読取りおよび書込み権限がユーザーに割り当てられている必要があります。 この手順では、ユーザー icadmin を InnoDB クラスタ の例に、rsadmin を InnoDB ReplicaSet の例に示します。

> **重要**
>
> 管理者のユーザー名とパスワードは、すべてのインスタンスで同じである必要があります。

8.0.20 以降のバージョンでは、setupAdminAccount(user) 操作を使用して、InnoDB クラスタ または InnoDB ReplicaSet の管理に必要な権限を持つ MySQL ユーザーアカウントを作成またはアップグレードします。 setupAdminAccount() 操作を使用するには、root などのユーザーを作成する権限を持つ MySQL ユーザーとして接続する必要があります。 setupAdminAccount(user) 操作では、dba.upgradeMetadata() 操作の前に、必要な権限を持つ既存の MySQL アカウントをアップグレードすることもできます。

必須の user 引数は、アカウントの管理に使用するために作成またはアップグレードする MySQL アカウントの名前です。 setupAdminAccount() 操作で受け入れられるユーザー名の形式は、標準の MySQL アカウント名の形式に従います。アカウント名の指定 を参照してください。 ユーザー引数の形式は username[@host]です。ここで、host はオプションであり、指定しない場合は % ワイルドカード文字にデフォルト設定されます。

たとえば、変数 myCluster に割り当てられた InnoDB クラスタ を管理する icadmin という名前のユーザーを作成するには、次のように発行します:

```
mysql-js> myCluster.setupAdminAccount('icadmin')

Missing the password for new account icadmin@%. Please provide one.
Password for new account: ********
Confirm password: ********

Creating user icadmin@%.
```

```
Setting user password.
Account icadmin@% was successfully created.
```

たとえば、8.0.20 より前のバージョンで作成された管理ユーザーがすでに存在する場合は、setupAdminAccount() 操作で update オプションを使用して、既存のユーザーの権限をアップグレードします。 これは、管理ユーザーに互換性を持たせるために、アップグレード時に関連します。 たとえば、icadmin issue という名前のユーザーをアップグレードするには、次のようにします:

```
mysql-js> myCluster.setupAdminAccount('icadmin', {'update':1})
Updating user icadmin@%.
Account icadmin@% was successfully updated.
```

8.0.20 より前のバージョンでは、管理用のユーザーを作成するには、dba.configureInstance() 操作で clusterAdmin オプションを使用することをお薦めします。 clusterAdmin オプションは、適切な権限を持つユーザーを作成する権限を持つユーザーに基づく MySQL Shell 接続で使用する必要があります。この例では、root ユーザーが使用されます。 例:

```
mysql-js> dba.configureInstance('root@ic-1:3306', {clusterAdmin: "'icadmin'@'ic-1%'"});
```

setupAdminAccount() 操作および clusterAdmin オプションで受け入れられるユーザー名の形式は、標準の MySQL アカウント名形式に従います。アカウント名の指定 を参照してください。

読取り操作のみが必要な場合 (監視目的など)、より制限された権限を持つアカウントを使用できます。 AdminAPI のユーザーの構成を参照してください。

詳細ロギング

本番デプロイメントを使用する場合は、MySQL Shell の冗長ロギングを構成すると便利です。 たとえば、ログ内の情報は、InnoDB クラスタ の一部として機能するようにサーバーインスタンスを準備する際に発生する可能性のある問題を見つけて解決するのに役立ちます。 冗長ロギングレベルで MySQL Shell を起動するには、--log-level オプションを使用します:

```
shell> mysqlsh --log-level=DEBUG3
```

DEBUG3 レベルをお薦めします。詳細は、--log-level を参照してください。 DEBUG3 が設定されている場合、MySQL Shell ログファイルには、各 AdminAPI コールの一部として実行される SQL クエリーを含む Debug: execute_sql(...) などの行が含まれます。 MySQL Shell によって生成されるログファイルは、Unix ベースのシステムの場合は ~/.mysqlsh/mysqlsh.log にあり、Microsoft Windows システムの場合は %APPDATA%\MySQL\mysqlsh\mysqlsh.log にあります。 詳しくは第9章「MySQL Shell のロギングおよびデバッグ」をご覧ください。

MySQL Shell ログレベルの有効化に加えて、各コマンドの発行後に AdminAPI が MySQL Shell で提供する出力量を構成できます。 AdminAPI 出力の量を有効にするには、MySQL Shell で次のコマンドを発行します:

```
mysql-js> dba.verbose=2
```

これにより、AdminAPI コールからの最大出力が可能になります。 使用可能な出力レベルは次のとおりです:

- デフォルトは 0 または OFF です。 これは最小限の出力を提供し、トラブルシューティングを行わない場合に推奨されるレベルです。
- 1 または ON を指定すると、各コールから AdminAPI に冗長出力が追加されます。
- 2 は、AdminAPI への各コールの実行内容に関する完全な情報を提供するデバッグ出力を冗長出力に追加します。

MySQL Shell では、オプションで、AdminAPI 操作で使用される SQL ステートメントをログに記録でき (サンドボックス操作を除く)、実行時に端末に表示することもできます。 これを行うように MySQL Shell を構成するには、セクション9.3「AdminAPI 操作のロギング」 を参照してください。

プライマリの検索

単一プライマリ InnoDB クラスタ または InnoDB ReplicaSet を使用している場合は、構成の変更をメタデータに書き込むことができるように、管理タスクのためにプライマリインスタンスに接続する必要があります。 現在のプライマリを検索するには、次の手順を実行します:

- MySQL Shell の起動時に --redirect-primary オプションを使用して、ターゲットサーバーが InnoDB クラスタ または InnoDB ReplicaSet の一部であることを確認します。 ターゲットインスタンスがプライマリでない場合、MySQL Shell はプライマリを検索してそれに接続します。
- ターゲットインスタンスがクラスタまたは ReplicaSet に属しているかどうかをチェックする shell.connectToPrimary([instance, password]) 操作 (バージョン 8.0.20 で追加) を使用します。 その場合、MySQL Shell はプライマリに対して新しいセッションを開き、アクティブなグローバル MySQL Shell セッションを確立されたセッションに設定して戻します。

  instance が指定されていない場合、操作はアクティブなグローバル MySQL Shell セッションの使用を試みます。 instance が指定されておらず、アクティブなグローバル MySQL Shell セッションがない場合は、例外がスローされます。 ターゲットインスタンスがクラスタまたは ReplicaSet に属していない場合、操作はエラーで失敗します。
- ステータス操作を使用して、結果でプライマリを検索し、そのインスタンスに手動で接続します。

スクリプト AdminAPI

このセクションに示す対話型モードに加えて、MySQL Shell では batch mode でのスクリプトの実行がサポートされています。 これにより、MySQL Shell --file オプションを使用して実行できる JavaScript または Python で記述されたスクリプトを使用して、AdminAPI を使用したプロセスを自動化できます。 例:

```
shell> mysqlsh --file setup-innodb-cluster.js
```

> **注記**
>
> スクリプトファイル名の後に指定されたコマンドラインオプションは、MySQL Shell ではなくスクリプトに渡されます。 これらのオプションには、JavaScript の os.argv 配列または Python の sys.argv 配列を使用してアクセスできます。 どちらの場合も、配列で最初に選択されるオプションはスクリプト名です。

スクリプトファイルの例の内容を次に示します:

```
print('InnoDB クラスタ sandbox set up\n');
print('=================================\n');
print('Setting up a MySQL InnoDB Cluster with 3 MySQL Server sandbox instances,\n');
print('installed in ~/mysql-sandboxes, running on ports 3310, 3320 and 3330.\n\n');

var dbPass = shell.prompt('Please enter a password for the MySQL root account: ', {type:"password"});

try {
  print('\nDeploying the sandbox instances.');
  dba.deploySandboxInstance(3310, {password: dbPass});
  print('.');
  dba.deploySandboxInstance(3320, {password: dbPass});
  print('.');
  dba.deploySandboxInstance(3330, {password: dbPass});
  print('.\nSandbox instances deployed successfully.\n\n');

  print('Setting up InnoDB Cluster...\n');
  shell.connect('root@localhost:3310', dbPass);

  var cluster = dba.createCluster("prodCluster");

  print('Adding instances to the Cluster.');
  cluster.addInstance({user: "root", host: "localhost", port: 3320, password: dbPass});
  print('.');
  cluster.addInstance({user: "root", host: "localhost", port: 3330, password: dbPass});
  print('.\nInstances successfully added to the Cluster.');

  print('\nInnoDB Cluster deployed successfully.\n');
} catch(e) {
  print('\nThe InnoDB Cluster could not be created.\n\nError: ' +
  + e.message + '\n');
}
```

AdminAPI は、MySQL Shell セクション5.8「API コマンドラインインタフェース」 でもサポートされます。 これにより、AdminAPI を環境に簡単に統合できます。 たとえば、ポート 1234 でリスニングしているサンドボックスインスタンスを使用して InnoDB クラスタ のステータスを確認するには、次のようにします:

```
$ mysqlsh root@localhost:1234 -- cluster status
```

これは、MySQL Shell の同等のコマンドにマップされます:

```
mysql-js> cluster.status()
```

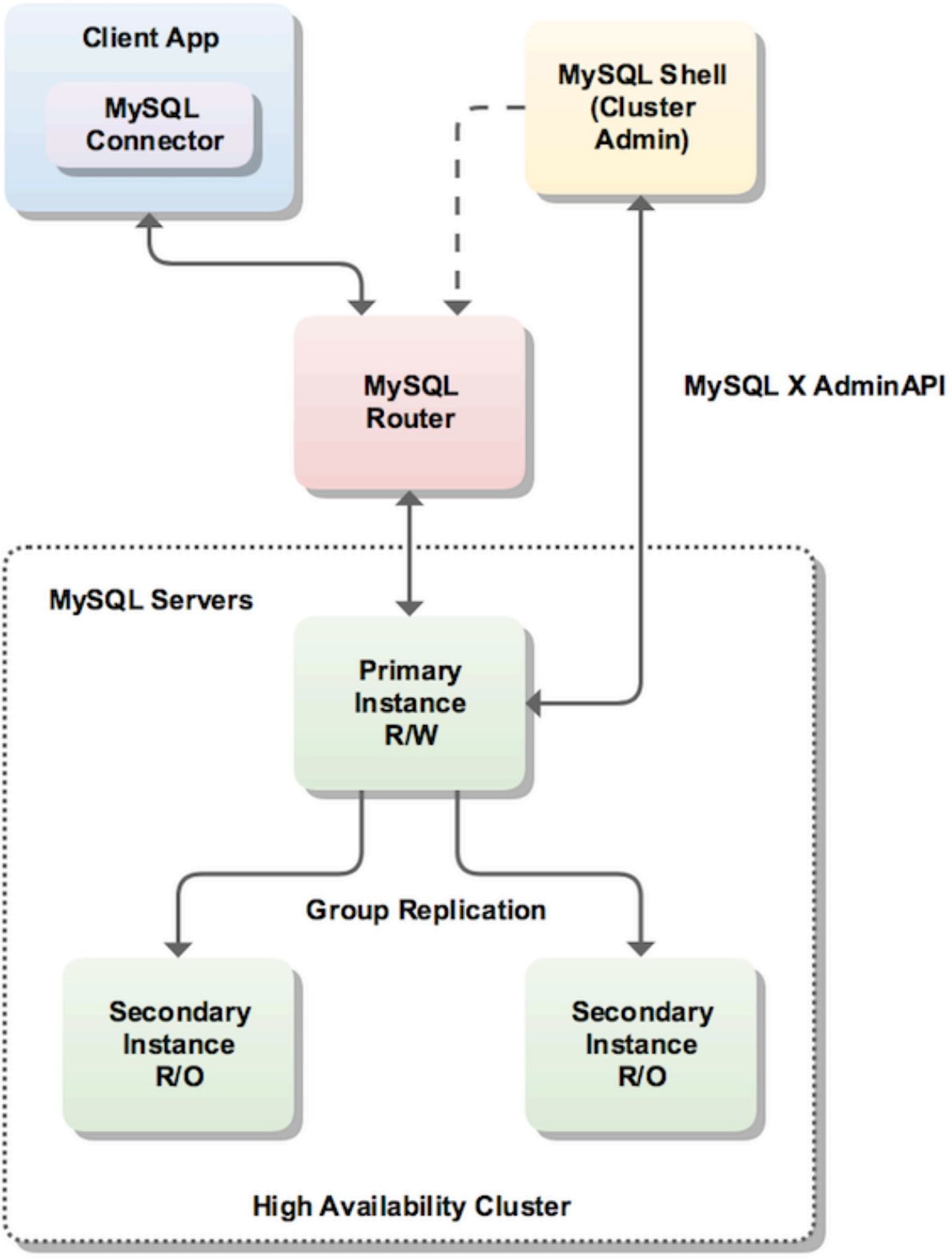
6.2 MySQL InnoDB クラスタ

MySQL InnoDB クラスタ は、MySQL の完全な高可用性ソリューションを提供します。 MySQL Shell に含まれる AdminAPI を使用すると、InnoDB クラスタ として機能するように少なくとも 3 つの MySQL サーバーインスタンスのグループを簡単に構成および管理できます。 この手順では、ホスト名 ic-number を例で使用します。 各 MySQL サーバーインスタンスは、組込みフェイルオーバーを使用して InnoDB クラスタs 内でデータをレプリケートするメカニズムを提供する MySQL Group Replication を実行します。AdminAPI では、InnoDB クラスタs で Group Replication を直接操作する必要はありませんが、詳細は グループレプリケーション を参照してください。MySQL Router は、デプロイするクラスタに基づいて自動的に構成し、クライアントアプリケーションをサーバーインスタンスに透過的に接続できます。 サーバーインスタンスで予期しない障害が発生した場合、クラスタは自動的に再構成されます。 デフォルトの単一プライマリモードでは、InnoDB クラスタ には単一の読取り/書込みサーバーインスタンスがあります - プライマリ。 複数のセカンダリサーバーインスタンスがプライマリのレプリカです。 プライマリに障害が発生すると、セカンダリはプライマリのロールに自動的に昇格されます。MySQL Router はこれを検出し、クライアントアプリケーションを新しいプライマリに転送します。 上級ユーザーは、複数のプライマリを持つようにクラスタを構成することもできます。

> **重要**
>
> InnoDB クラスタ では、MySQL NDB Cluster はサポートされていません。NDB Cluster は、NDB ストレージエンジンに加えて、MySQL Server 8.0 で提供されていない NDB Cluster 固有の多くのプログラムに依存します。NDB は、MySQL NDB Cluster 配布の一部としてのみ使用できます。 また、MySQL Server 8.0 で提供される MySQL サーバーバイナリ (mysqld) は NDB Cluster では使用できません。 MySQL NDB Cluster の詳細は、MySQL NDB Cluster 8.0 を参照してください。MySQL Server NDB Cluster と比較した InnoDB の使用 では、InnoDB ストレージエンジンと NDB ストレージエンジンの違いに関する情報を提供します。

次の図は、これらのテクノロジの連携の概要を示しています:

図 6.1 InnoDB クラスタ の概要



6.2.1 MySQL InnoDB クラスタ の要件

InnoDB クラスタ の本番デプロイメントをインストールする前に、使用するサーバーインスタンスが次の要件を満たしていることを確認します。

- InnoDB クラスタ は Group Replication を使用するため、サーバーインスタンスは同じ要件を満たす必要があります。 グループレプリケーションの要件 を参照してください。AdminAPI には、インスタンスがグループレプリケーション要件を満たしていることを確認する dba.checkInstanceConfiguration() メソッドと、要件を満たすようにインスタンスを構成する dba.configureInstance() メソッドが用意されています。

  > 注記
  >
  > サンドボックスデプロイメントを使用する場合、インスタンスはこれらの要件を自動的に満たすように構成されます。

- グループレプリケーションメンバーには、InnoDB 以外のストレージエンジン (MyISAM など) を使用してテーブルを含めることができます。 このようなテーブルは、グループレプリケーションでは書き込むことができないため、InnoDB クラスタ の使用時には書き込まれません。 InnoDB クラスタ を使用してこのようなテーブルに書き込むことができるようにするには、InnoDB クラスタ でインスタンスを使用する前に、このようなすべてのテーブルを InnoDB に変換します。

- パフォーマンススキーマは、InnoDB クラスタ で使用する任意のインスタンスで有効にする必要があります。

- MySQL Shell が InnoDB クラスタ で使用するサーバーの構成に使用するプロビジョニングスクリプトには、Python へのアクセスが必要です。 Windows の場合、MySQL Shell には Python が含まれており、ユーザーの構成は必要ありません。 Unix では、Python はシェル環境の一部として検出される必要があります。 システムで Python が正しく構成されていることを確認するには、次のコマンドを発行します:

  ```
  $ /usr/bin/env python
  ```

  Python インタプリタが起動した場合、それ以上のアクションは必要ありません。 前述のコマンドが失敗した場合は、/usr/bin/python と選択した Python バイナリの間にソフトリンクを作成します。 詳細は、サポートされる言語を参照してください。

- バージョン 8.0.17 からは、インスタンスは InnoDB クラスタ 内で一意の server_id を使用する必要があります。 Cluster.addInstance(instance) 操作を使用する場合、instance の server_id がクラスタ内のインスタンスによってすでに使用されていると、操作はエラーで失敗します。

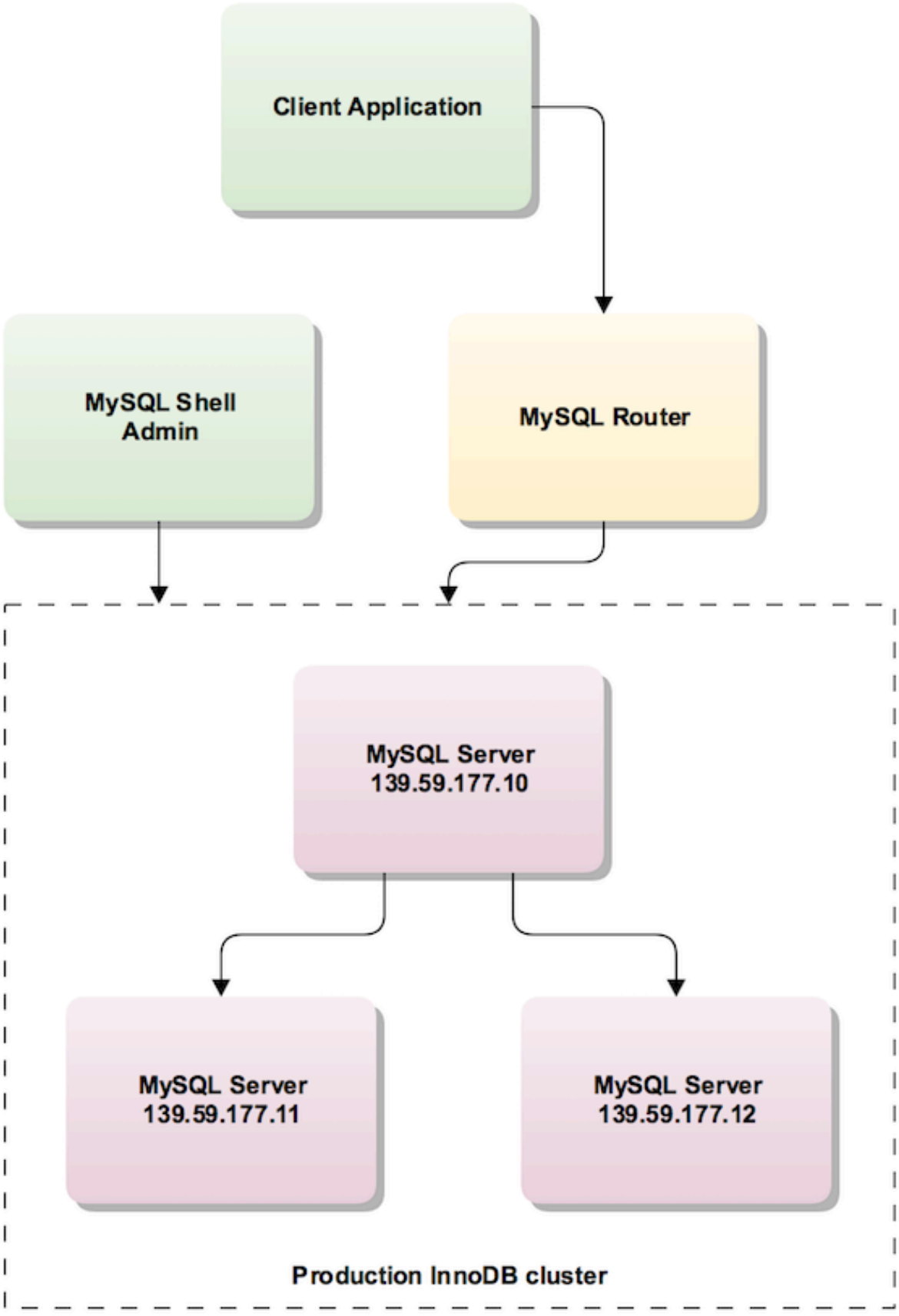
- バージョン 8.0.23 からは、パラレルレプリケーションアプライヤを使用するようにインスタンスを構成する必要があります。 パラレルレプリケーションアプリケーションの構成を参照してください。

- InnoDB クラスタ のインスタンスを構成するプロセス中に、インスタンスの使用に必要なシステム変数の大部分が構成されます。 ただし、AdminAPI では transaction_isolation システム変数は構成されません。つまり、REPEATABLE READ にデフォルト設定されます。 これは単一プライマリクラスタには影響しませんが、マルチプライマリクラスタを使用している場合は、アプリケーションで REPEATABLE READ セマンティクスに依存しないかぎり、READ COMMITTED 分離レベルを使用することをお薦めします。 グループレプリケーションの制限事項を参照してください。

6.2.2 本番 InnoDB クラスタ のデプロイ

本番環境で作業している場合、InnoDB クラスタ を構成する MySQL サーバーインスタンスは、セクション 6.5「AdminAPI MySQL サンドボックス」 で説明されているように、単一のマシンではなく、ネットワークの一部として複数のホストマシンで実行されます。 これらの手順に進む前に、サーバーインスタンスとしてクラスタに追加する各マシンに必要なソフトウェアをインストールする必要があります。コンポーネントのインストール を参照してください。

次の図は、このセクションで使用するシナリオを示しています:


Client Application
MySQL Shell
Admin
MySQL Router
MySQL Server
139.59.177.10
MySQL Server
139.59.177.11
MySQL Server
139.59.177.12
Production InnoDB cluster

> **重要**
>
> すべてのインスタンスがローカルにデプロイされるサンドボックスデプロイメントとは異なり、AdminAPI がローカルファイルアクセス権を持ち、構成の変更を永続化できるマシンでは、本番デプロイメントでインスタンスの構成の変更を永続化する必要があります。 これを行う方法は、インスタンスで実行されている MySQL のバージョンによって異なります。設定の永続化 を参照してください。

サーバー接続情報を AdminAPI に渡すには、URI のような接続文字列またはデータディクショナリを使用します。URI 類似文字列またはキーと値のペアを使用したサーバーへの接続 を参照してください。 このドキュメントでは、URI のような文字列を示します。

このセクションでは、次のことを前提としています:

- インスタンスへ MySQL コンポーネントがインストールされている
- MySQL Shell がインストールされ、およびインスタンスを指定して接続できる
- 適切な管理ユーザーが作成されている

6.2.2.1 新しい本番 InnoDB クラスタ のデプロイ

次の各セクションでは、新しい本番 InnoDB クラスタ をデプロイする方法について説明します。

- 本番インスタンスの構成
- クラスタの作成
- クラスタへのインスタンスの追加
- InnoDB クラスタによって作成されたユーザーアカウント
- InnoDB クラスタ ポートの構成

本番インスタンスの構成

AdminAPI には、インスタンスが InnoDB クラスタ 使用のために適切に構成されているかどうかをチェックし、InnoDB クラスタ と互換性のない設定が見つかった場合にインスタンスを構成する dba.configureInstance() 関数が用意されています。 インスタンスに対して dba.configureInstance() コマンドを実行すると、インスタンスを InnoDB クラスタ の使用に使用できるようにするために必要なすべての設定がチェックされます。 インスタンスで構成の変更が不要な場合は、インスタンスの構成を変更する必要はなく、dba.configureInstance() コマンド出力によって、インスタンスで InnoDB クラスタ を使用する準備ができていることが確認されます。 インスタンスを InnoDB クラスタ と互換性を持たせるために変更が必要な場合は、互換性のない設定のレポートが表示され、コマンドでインスタンスオプションファイルを変更できます。 MySQL Shell のインスタンスへの接続方法およびインスタンスで実行されている MySQL のバージョンに応じて、これらの変更をリモートインスタンスオプションファイルに永続化することで永続化できます。設定の永続化 を参照してください。 構成変更の永続化をサポートしていないインスタンスでは、インスタンスをローカルに構成する必要があります。dba.configureLocalInstance() でのインスタンスの構成 を参照してください。 または、インスタンスオプションファイルを手動で変更することもできます。詳細は、オプションファイルの使用 を参照してください。 構成の変更方法に関係なく、構成の変更が検出されるように、MySQL の再起動が必要になる場合があります。

dba.configureInstance() コマンドの構文は次のとおりです:

```
dba.configureInstance([instance][, options])
```

ここで、instance はインスタンス定義で、options は操作を構成するための追加オプションを含むデータディクショナリです。 このコマンドは、操作結果に関する説明テキストメッセージを返します。

instance 定義はインスタンスの接続データです。URI 類似文字列またはキーと値のペアを使用したサーバーへの接続 を参照してください。 ターゲットインスタンスがすでに InnoDB クラスタ に属している場合、エラーが生成され、プロセスは失敗します。

オプションディクショナリには次のものを含めることができます:

- mycnfPath - インスタンスの MySQL オプションファイルのパス。

- outputMycnfPath - インスタンスの MySQL オプションファイルを書き込む代替出力パス。
- password - 接続で使用されるパスワード。
- clusterAdmin - 作成する InnoDB クラスタ 管理者ユーザーの名前。 サポートされているフォーマットは、標準の MySQL アカウント名フォーマットです。 ユーザー名およびホスト名の識別子または文字列をサポートします。 デフォルトでは、引用符で囲まれていない場合、入力は文字列であるとみなされます。 管理用のユーザーアカウントの作成を参照してください。
- clusterAdminPassword - clusterAdmin を使用して作成される InnoDB クラスタ 管理者アカウントのパスワード。 このオプションを使用して指定できますが、これは潜在的なセキュリティリスクです。 このオプションを指定せずに clusterAdmin オプションを指定すると、対話型プロンプトでパスワードの入力を求められます。
- 非推奨であり、将来のバージョンでの削除がスケジュールされています

  clearReadOnly - super_read_only をオフに設定する必要があることを確認するために使用されるブール値。スーパー読取り専用およびインスタンス を参照してください。
- interactive - ユーザーにプロンプトが表示されず、確認プロンプトが表示されないように、コマンド実行で対話型ウィザードを無効にするために使用されるブール値。
- restart - 操作を終了するためにターゲットインスタンスのリモート再起動を実行する必要があることを示すブール値。

接続パスワードはインスタンス定義に含めることができますが、これはセキュアではないためお薦めしません。 MySQL Shell セクション4.4「プラガブルパスワードストア」 を使用して、インスタンスパスワードを安全に格納します。

インスタンスに対して dba.configureInstance() が発行されると、このコマンドはインスタンス設定が InnoDB クラスタ の使用に適しているかどうかをチェックします。 InnoDB クラスタ で必要な設定を示すレポートが表示されます。 インスタンスの設定を変更する必要がない場合は、InnoDB クラスタ で使用でき、クラスタの作成 に進むことができます。 インスタンス設定が InnoDB クラスタ の使用に対して有効でない場合、dba.configureInstance() コマンドは変更が必要な設定を表示します。 インスタンスを構成する前に、次の情報を含むテーブルに示されている変更を確認するよう求められます:

- Variable - 無効な構成変数。
- Current Value - 無効な構成変数の現在の値。
- Required Value - 構成変数に必要な値。

続行方法は、インスタンスが永続化設定をサポートしているかどうかによって異なります。設定の永続化 を参照してください。 MySQL Shell が現在実行されている MySQL インスタンス (つまり、ローカルインスタンス) に対して dba.configureInstance() を発行すると、インスタンスの自動構成が試行されます。 リモートインスタンスに対して dba.configureInstance() が発行されたときに、インスタンスが構成変更の永続化を自動的にサポートしている場合は、これを選択できます。 リモートインスタンスが、InnoDB クラスタ で使用できるように構成するための変更の永続化をサポートしていない場合は、インスタンスをローカルに構成する必要があります。 dba.configureLocalInstance() でのインスタンスの構成を参照してください。

一般に、dba.configureInstance() でオプションファイルを構成した後にインスタンスを再起動する必要はありませんが、特定の設定によっては再起動が必要になる場合があります。 この情報は、dba.configureInstance() の発行後に生成されるレポートに表示されます。 インスタンスが RESTART ステートメントをサポートしている場合、MySQL Shell はインスタンスを停止してから起動できます。 これにより、インスタンスオプションファイルに加えられた変更が mysqld によって確実に検出されます。 詳細は、RESTART を参照してください。

> 注記
>
> RESTART ステートメントを実行すると、インスタンスへの現在の接続が失われます。 自動再接続が有効な場合、サーバーの再起動後に接続が再確立されます。 それ以外の場合は、接続を手動で再確立する必要があります。

dba.configureInstance() メソッドは、クラスタのメンバー間の接続に使用される適切なユーザーがクラスタの使用に使用できることを検証します。管理用のユーザーアカウントの作成 を参照してください。

クラスタを管理するユーザーを指定しない場合は、対話型モードでウィザードを使用して次のいずれかのオプションを選択できます:

- root ユーザーのリモート接続を有効にします。本番環境ではお薦めしません
- 新規ユーザーの作成
- ユーザーを手動で作成する必要がある自動構成はありません

> **ヒント**
>
> インスタンスに super_read_only=ON がある場合は、AdminAPI で super_read_only=OFF を設定できることを確認する必要がある場合があります。 詳しくはスーパー読取り専用およびインスタンスをご覧ください。

クラスタの作成

インスタンスを準備したら、MySQL Shell が接続されているインスタンスをクラスタのシードインスタンスとして使用して、dba.createCluster() 関数を使用してクラスタを作成します。 シードインスタンスは、クラスタに追加した他のインスタンスにレプリケートされ、シードインスタンスのレプリカになります。 この手順では、ic-1 インスタンスがシードとして使用されます。 dba.createCluster(name) MySQL Shell を発行すると、MySQL Shell の現在のグローバルセッションに接続されているサーバーインスタンスへの クラシック MySQL プロトコル セッションが作成されます。 たとえば、testCluster というクラスタを作成し、返されたクラスタを cluster という変数に割り当てるには、次のようにします:

```
mysql-js> var cluster = dba.createCluster('testCluster')
Validating instance at icadmin@ic-1:3306...
This instance reports its own address as ic-1
Instance configuration is suitable.
Creating InnoDB cluster 'testCluster' on 'icadmin@ic-1:3306'...
Adding Seed Instance...
Cluster successfully created. Use Cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to
one server failure.
```

返されたクラスタを変数に割り当てるこのパターンを使用すると、Cluster オブジェクトメソッドを使用してクラスタに対してさらに操作を実行できます。 返された Cluster オブジェクトは、MySQL Shell グローバルセッションから独立した新しいセッションを使用します。 これにより、MySQL Shell グローバルセッションを変更した場合、Cluster オブジェクトはインスタンスへのセッションを維持します。

クラスタを管理できるようにするには、必要な権限を持つ適切なユーザーがいることを確認する必要があります。 推奨される方法は、管理ユーザーを作成することです。 インスタンスの構成時に管理ユーザーを作成しなかった場合は、Cluster.setupAdminAccount() 操作を使用します。 たとえば、変数 cluster に割り当てられた InnoDB クラスタ を管理できる icadmin という名前のユーザーを作成するには、次のように発行します:

```
mysql-js> cluster.setupAdminAccount(icadmin)
```

クラスタ管理ユーザーの詳細は、AdminAPI のユーザーの構成 を参照してください。

dba.createCluster() 操作では、MySQL Shellinteractive オプションがサポートされます。 interactive がオンの場合、プロンプトは次の状況で表示されます:

- クラスタに属するインスタンスで実行され、adoptFromGr オプションが false の場合、既存のクラスタを採用するかどうかを尋ねられます
- force オプションが使用されていない (true に設定されていない) 場合、マルチプライマリクラスタの作成を確認するよう求められます

> **注記**
>
> メタデータにアクセスできないというエラーが発生した場合は、ループバックネットワークインタフェースが構成されている可能性があります。 InnoDB クラスタ を正しく使用するには、ループバックインタフェースを無効にします。

クラスタが作成されたことを確認するには、クラスタインスタンスの status() 関数を使用します。 Cluster.status() によるクラスタステータスの確認を参照してください。

> **ヒント**
>
> サーバーインスタンスがクラスタに属したら、MySQL Shell および AdminAPI を使用してのみ管理することが重要です。 クラスタに追加されたインスタンスでのグループレプリケーションの構成の手動変更の試行はサポートされていません。 同様に、AdminAPI を使用してインスタンスを構成した後の、server_uuid などの InnoDB クラスタ に重要なサーバー変数の変更はサポートされていません。

MySQL Shell 8.0.14 以降を使用してクラスタを作成する場合、インスタンスにアクセスできなくなった場合など、インスタンスがクラスタから削除される前にタイムアウトを設定できます。 シードインスタンスで group_replication_member_expel_timeout 変数を構成する dba.createCluster() 操作に expelTimeout オプションを渡します。 expelTimeout オプションには、0 から 3600 の範囲の整数値を指定できます。 expelTimeout が構成されたクラスタに追加される MySQL サーバー 8.0.13 以降を実行しているすべてのインスタンスは、シードインスタンスで構成されているものと同じ expelTimeout 値を持つように自動的に構成されます。

dba.createCluster() に渡すことができるその他のオプションの詳細は、セクション6.2.5「InnoDB クラスタの操作」を参照してください。

クラスタへのインスタンスの追加

クラスタにインスタンスを追加するには、Cluster.addInstance(instance) 関数を使用します。ここで、instance は構成済インスタンスへの接続情報です。本番インスタンスの構成 を参照してください。 バージョン 8.0.17 から、Group Replication はインスタンスのパッチバージョンを考慮する互換性ポリシーを実装し、Cluster.addInstance() 操作はこれを検出し、非互換性が発生した場合はエラーで操作を終了します。 インスタンスでの MySQL バージョンの確認 および グループ内の異なるメンバーバージョンの組合せ を参照

1 つのインスタンスの障害を許容するには、クラスタ内に 3 つ以上のインスタンスが必要です。 さらにインスタンスを追加すると、インスタンスの障害に対する許容範囲が増加します。 クラスタにインスタンスを追加するには、次のコマンドを発行します:

```
mysql-js> cluster.addInstance('icadmin@ic-2:3306')
A new instance will be added to the InnoDB cluster. Depending on the amount of
data on the cluster this might take from a few seconds to several hours.
Please provide the password for 'icadmin@ic-2:3306': ********
Adding instance to the cluster ...
Validating instance at ic-2:3306...
This instance reports its own address as ic-2
Instance configuration is suitable.
The instance 'icadmin@ic-2:3306' was successfully added to the cluster.
```

新しいインスタンスがクラスタに追加されると、必要に応じて、このインスタンスのローカルアドレスがすべてのオンラインクラスタインスタンスの group_replication_group_seeds 変数に自動的に追加され、新しいインスタンスを使用してグループに再度参加できるようになります。

> **注記**
>
> group_replication_group_seeds にリストされているインスタンスは、リストに表示される順序に従って使用されます。 これにより、ユーザー指定の設定が最初に使用され、優先されます。 詳しくはInnoDB クラスタs のカスタマイズをご覧ください。

MySQL 8.0.17 以降を使用している場合は、クラスタとの同期に必要なトランザクションをインスタンスがリカバリする方法を選択できます。 結合インスタンスが以前にクラスタによって処理されたすべてのトランザクションをリカバリした場合にのみ、オンラインインスタンスとして参加し、トランザクションの処理を開始できます。 詳細は、セクション6.2.2.2「InnoDB クラスタ での MySQL クローンの使用」を参照してください。

また、8.0.17 以降では、Cluster.addInstance() の動作を構成して、リカバリ操作をバックグラウンドで続行したり、MySQL Shell で様々なレベルの進行状況を監視できます。

クラスタからインスタンスをリカバリするために選択したオプションに応じて、MySQL Shell に異なる出力が表示されます。 インスタンス ic-2 をクラスタに追加し、ic-1 がシードまたはドナーであるとします。

- MySQL クローンを使用してクラスタからインスタンスをリカバリする場合、出力は次のようになります:

```
Validating instance at ic-2:3306...
This instance reports its own address as ic-2:3306
```

```
Instance configuration is suitable.
A new instance will be added to the InnoDB cluster. Depending on the amount of
data on the cluster this might take from a few seconds to several hours.
Adding instance to the cluster...
Monitoring recovery process of the new cluster member. Press ^C to stop monitoring and let it continue in background.
Clone based state recovery is now in progress.
NOTE: A server restart is expected to happen as part of the clone process. If the
server does not support the RESTART command or does not come back after a
while, you may need to manually start it back.
* Waiting for clone to finish...
NOTE: ic-2:3306 is being cloned from ic-1:3306
** Stage DROP DATA: Completed
** Clone Transfer
FILE COPY  ############################################################  100%  Completed
PAGE COPY  ############################################################  100%  Completed
REDO COPY  ############################################################  100%  Completed
NOTE: ic-2:3306 is shutting down...
* Waiting for server restart... ready
* ic-2:3306 has restarted, waiting for clone to finish...
** Stage RESTART: Completed
* Clone process has finished: 2.18 GB transferred in 7 sec (311.26 MB/s)
State recovery already finished for 'ic-2:3306'
The instance 'ic-2:3306' was successfully added to the cluster.
```

サーバーの再起動に関する警告が表示されます。場合によっては、インスタンスを手動で再起動する必要があります。 RESTART ステートメントを参照してください。

- 増分リカバリを使用してクラスタからインスタンスをリカバリする場合、出力は次のようになります:

```
Incremental distributed state recovery is now in progress.
* Waiting for incremental recovery to finish...
NOTE: 'ic-2:3306' is being recovered from 'ic-1:3306'
* Distributed recovery has finished
```

リカバリフェーズの監視を取り消すには、CONTROL+C を発行します。 これにより監視は停止されますが、リカバリプロセスはバックグラウンドで続行されます。 Cluster.addInstance() 操作で waitRecovery 整数オプションを使用して、リカバリフェーズに関するコマンドの動作を制御できます。 次の値を使用できます:

- 0: 待機せず、リカバリプロセスをバックグラウンドで終了させます
- 1: リカバリプロセスが終了するまで待機
- 2: リカバリプロセスが終了するまで待機し、詳細な静的進捗情報を表示
- 3: リカバリプロセスが終了するまで待機し、詳細な動的進捗情報 (進捗バー) を表示

デフォルトでは、MySQL Shell が実行されている標準出力が端末を参照する場合、waitRecovery オプションのデフォルトは 3 です。 それ以外の場合は、デフォルトで 2 に設定されます。 リカバリ操作の監視を参照してください。

インスタンスが追加されたことを確認するには、クラスタインスタンスの status() 関数を使用します。 たとえば、2 番目のインスタンスを追加した後のサンドボックスクラスタのステータス出力は次のとおりです:

```
mysql-js> cluster.status()
{
    "clusterName": "testCluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "ic-1:3306",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "ic-1:3306": {
                "address": "ic-1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-2:3306": {
                "address": "ic-2:3306",
                "mode": "R/O",
```

```
            "readReplicas": {},
            "role": "HA",
            "status": "ONLINE"
        }
    }
  },
  "groupInformationSourceMember": "mysql://icadmin@ic-1:3306"
}
```

続行方法は、インスタンスが MySQL Shell が実行されているインスタンスに対してローカルであるかリモートであるか、およびインスタンスが構成変更の自動永続化をサポートしているかどうかによって異なります。設定の永続化 を参照してください。 インスタンスで構成変更の永続化が自動的にサポートされている場合、設定を手動で永続化する必要はなく、さらにインスタンスを追加するか、次のステップに進むことができます。 インスタンスで構成変更の永続化が自動的にサポートされない場合は、インスタンスをローカルに構成する必要があります。 dba.configureLocalInstance() でのインスタンスの構成を参照してください。 これは、クラスタから離れる場合にインスタンスがクラスタに再参加するようにするために不可欠です。

> **ヒント**
>
> インスタンスに super_read_only=ON がある場合は、AdminAPI で super_read_only=OFF を設定できることを確認する必要がある場合があります。 詳しくはスーパー読取り専用およびインスタンスをご覧ください。

クラスタをデプロイしたら、高可用性を提供するように MySQL Router を構成できます。セクション6.4「MySQL Router」 を参照してください。

InnoDB クラスタによって作成されたユーザーアカウント

グループレプリケーションの使用の一環として、InnoDB クラスタ は、クラスタ内のサーバー間の接続を可能にする内部リカバリユーザーを作成します。 これらのユーザーはクラスタの内部にあり、生成されたユーザーのユーザー名は mysql_innodb_cluster_server_id@% のネーミングスキームに従います (server_id はインスタンスに対して一意です)。 8.0.17 より前のバージョンでは、生成されたユーザーのユーザー名は mysql_innodb_cluster_r[10_numbers]のネーミングスキームに従いました。 内部ユーザーに使用されるホスト名は、ipAllowlist オプションが構成されているかどうかによって異なります。 ipAllowlist が構成されていない場合、デフォルトで AUTOMATIC に設定され、ホスト名の値にワイルドカードの % 文字と localhost の両方を使用して内部ユーザーが作成されます。 ipAllowlist が構成されている場合、ipAllowlist リスト内のアドレスごとに内部ユーザーが作成されます。 詳細は、サーバーの許可リストの作成を参照してください。

各内部ユーザーにはランダムに生成されたパスワードがあります。 バージョン 8.0.18 から、AdminAPI を使用して内部ユーザーに対して生成されたパスワードを変更できます。 回復アカウントのパスワードのリセットを参照してください。 ランダムに生成されたユーザーには、次の権限が付与されます:

```
GRANT REPLICATION SLAVE ON *.* to internal_user;
```

内部ユーザーアカウントがシードインスタンスに作成され、クラスタ内の他のインスタンスにレプリケートされます。 内部ユーザーは次のとおりです:

- dba.createCluster() を発行して新しいクラスタを作成するときに生成されます
- Cluster.addInstance() を発行してクラスタに新しいインスタンスを追加するときに生成されます。

また、ipAllowlist オプションを使用してホスト名を指定すると、Cluster.rejoinInstance() 操作によって新しい内部ユーザーが生成されることもあります。 たとえば、次のように発行します:

```
Cluster.rejoinInstance({ipAllowlist: "192.168.1.1/22"});
```

使用されている ipAllowlist 値を考慮して、既存のすべての内部ユーザーが削除され、新しい内部ユーザーが作成されます。

Group Replication に必要な内部ユーザーの詳細は、分散リカバリのユーザー資格証明 を参照してください。

InnoDB クラスタ ポートの構成

クラスタに属するインスタンスは、異なるタイプの通信に異なるポートを使用します。 クラシック MySQL プロトコル 経由のクライアント接続に使用される 3306 のデフォルトの port と、デフォルトで 33060 に設定され、X プロトコル クライアント接続に使用される mysqlx_port に加えて、クライアント接続に使用されないクラスタ内のインス

タンス間の内部接続用のポートもあります。 このポートは、group_replication_local_address システム変数を構成する localAddress オプションによって構成され、クラスタ内のインスタンスが相互に通信できるように、このポートをオープンする必要があります。 たとえば、ファイアウォールがこのポートをブロックしている場合、インスタンスは相互に通信できず、クラスタは機能しません。 同様に、インスタンスが SELinux を使用している場合は、インスタンスが相互に通信できるように、InnoDB クラスタ で使用されるすべての必須ポートが開いていることを確認する必要があります。 MySQL 機能の TCP ポートコンテキストの設定 および「MySQL Shell ポートリファレンス」を参照してください。

クラスタを作成するか、クラスタにインスタンスを追加する場合、デフォルトでは、localAddress ポートはターゲットインスタンスの port 値に 10 を乗算して結果に追加することで計算されます。 たとえば、ターゲットインスタンスの port がデフォルト値 3306 の場合、計算された localAddress ポートは 33061 です。 クラスタインスタンスで使用されるポート番号が、localAddress の計算方法と互換性があることを確認する必要があります。 たとえば、クラスタの作成に使用されているサーバーインスタンスの port 番号が 6553 より大きい場合、計算された localAddress ポート番号が最大有効ポート 65535 を超えているため、dba.createCluster() 操作は失敗します。 この状況を回避するには、InnoDB クラスタ に使用するインスタンスで低い port 値を使用するか、localAddress 値を手動で割り当てます。 次に例を示します:

```
mysql-js> dba.createCluster('testCluster', {'localAddress':'icadmin@ic-1:33061'}
```

6.2.2.2 InnoDB クラスタ での MySQL クローンの使用

MySQL 8.0.17 では、InnoDB クラスタ は MySQL クローンプラグインを統合して、参加インスタンスの自動プロビジョニングを提供します。 インスタンスがクラスタと同期できるようにクラスタデータを取得するプロセスは、分散リカバリと呼ばれます。 インスタンスでクラスタトランザクションをリカバリする必要がある場合は、ドナー (データを提供するクラスタインスタンス) と受信者 (ドナーからデータを受信するインスタンス) を区別します。 以前のバージョンでは、グループレプリケーションは、結合しているインスタンスがクラスタに結合できるようにクラスタと同期するために必要なトランザクションをリカバリするために非同期レプリケーションのみを提供していました。 以前に処理されたトランザクションが大量にあるクラスタでは、クラスタに参加する前に、新しいインスタンスがすべてのトランザクションをリカバリするのに時間がかかる場合があります。 または、GTID をパージしたクラスタ (定期的な保守の一部など) で、新しいインスタンスのリカバリに必要なトランザクションの一部が欠落している可能性があります。 このような場合の唯一の代替方法は、グループレプリケーションでの MySQL Enterprise Backup の使用 に示すように、MySQL Enterprise Backup などのツールを使用してインスタンスを手動でプロビジョニングすることでした。

MySQL クローンは、インスタンスがクラスタとの同期に必要なトランザクションをリカバリするための代替方法を提供します。 非同期レプリケーションを使用してトランザクションをリカバリするかわりに、MySQL クローンはドナーインスタンス上のデータのスナップショットを取得し、そのスナップショットを受信者に転送します。

> **警告**
>
> レシーバ内の以前のデータはすべて、クローン操作中に破棄されます。 ただし、テーブルに格納されていないすべての MySQL 設定は保持されます。

クローン操作によってスナップショットが受信者に転送されると、スナップショットの転送中にクラスタでトランザクションが処理された場合、非同期レプリケーションを使用して、受信者とクラスタの同期に必要なデータがリカバリされます。 これは、非同期レプリケーションを使用してすべてのトランザクションをリカバリするインスタンスよりもはるかに効率的であり、パージされた GTID によって発生する問題を回避して、InnoDB クラスタ の新しいインスタンスを迅速にプロビジョニングできます。 詳細は、クローンプラグインおよび分散リカバリのためのクローニングを参照してください

MySQL クローンの使用とは対照的に、増分リカバリは、クラスタに参加しているインスタンスが非同期レプリケーションのみを使用してクラスタからインスタンスをリカバリするプロセスです。 InnoDB クラスタ が MySQL クローンを使用するように構成されている場合、クラスタに参加するインスタンスは、MySQL クローンまたは増分リカバリのいずれかを使用してクラスタトランザクションをリカバリします。 デフォルトでは、クラスタは最適な方法を自動的に選択しますが、オプションでこの動作を構成してクローニングを強制できます。これにより、結合インスタンスによってすでに処理されているトランザクションが置換されます。 MySQL Shell を対話モードで使用している場合、デフォルトでは、クラスタがリカバリを続行できるかどうかがわからない場合は、対話型プロンプトが表示されます。 このセクションでは、提供される様々なオプションと、選択できるオプションに影響する様々なシナリオについて説明します。

また、RECOVERING 状態のメンバーに対する Cluster.status() の出力には、MySQL クローンを使用しているか増分リカバリを使用しているかにかかわらず、リカバリ操作を簡単に監視できるリカバリ進捗情報が含まれていま

す。InnoDB クラスタ は、Cluster.status() の出力で MySQL クローンを使用するインスタンスに関する追加情報を提供します。

MySQL クローンを使用するクラスタの操作

MySQL クローンを使用する InnoDB クラスタ では、次の追加動作が提供されます。

dba.createCluster() および MySQL クローン

バージョン 8.0.17 からは、MySQL クローンプラグインが使用可能なインスタンスに新しいクラスタが作成されると、デフォルトで自動的にインストールされ、クラスタはクローニングをサポートするように構成されます。 InnoDB クラスタ リカバリアカウントは、必要な BACKUP_ADMIN 権限で作成されます。

disableClone ブールオプションを true に設定して、クラスタの MySQL クローンを無効にします。 この場合、この構成のメタデータエントリが追加され、MySQL クローンプラグインがインストールされている場合はアンインストールされます。 disableClone オプションは、dba.createCluster() を発行するとき、または Cluster.setOption() を使用してクラスタを実行しているときにいつでも設定できます。

Cluster.addInstance(instance) および MySQL クローン

新しいインスタンスが MySQL 8.0.17 以降を実行しており、MySQL 8.0.17 以降を実行しているドナーがクラスタ内に少なくとも 1 人 (group_replication_group_seeds リストに含まれている) 存在する場合、MySQL クローンを結合 instance に使用できます。 MySQL クローンを使用するクラスタは、クラスタへのインスタンスの追加 に記載されている動作に従い、クラスタからのインスタンスのリカバリに必要なデータの転送方法の選択肢が追加されています。 Cluster.addInstance(instance) の動作は、次の要因によって異なります:

- MySQL クローンがサポートされているかどうか。
- 増分リカバリが可能かどうか。バイナリログの可用性によって異なります。 たとえば、ドナーインスタンスに必要なすべてのバイナリログ (GTID_PURGED が空) がある場合、増分リカバリが可能です。 すべてのバイナリログが必要なクラスタインスタンスがない場合、増分リカバリはできません。
- 増分リカバリが適切かどうか。 増分リカバリが可能な場合でも、インスタンス上のデータと競合する可能性があるため、ドナーおよびレシーバ上の GTID セットがチェックされ、増分リカバリが適切であることが確認されます。 比較の結果は次のようになります:
  - 新規: レシーバに空の GTID_EXECUTED GTID セットがあります
  - 同一: 受け側にドナー GTID セットと同じ GTID セットがあります
  - 回収可能: 受け側にトランザクションが欠落している GTID セットがありますが、これらはドナーから回収できます
  - 回復不能: トランザクションが欠落している GTID セットがドナーにあります。パージされた可能性があります
  - 多様: ドナーとレシーバの GTID セットが相違しています

  比較の結果が同一またはリカバリ可能と判断された場合、増分リカバリが適切とみなされます。 比較の結果がリカバリ不能または分散と判断された場合、増分リカバリは適切とみなされません。

  新規とみなされるインスタンスの場合、バイナリログがパージされたかどうか、または GTID_PURGED 変数と GTID_EXECUTED 変数がリセットされたかどうかを判断できないため、増分リカバリは適切とみなされません。 または、バイナリログおよび GTID が有効になる前に、サーバーがすでにトランザクションを処理している可能性があります。 したがって、対話型モードでは、増分リカバリを使用することを確認する必要があります。
- gtidSetIsComplete オプションの状態。 完全な GTID セットを使用してクラスタが作成されていることが確実であるため、余分な確認なしで GTID セットが空のインスタンスを追加できる場合は、クラスタレベルの gtidSetIsComplete ブールオプションを true に設定します。

> **警告**
>
> gtidSetIsComplete オプションを true に設定すると、含まれているデータに関係なく、結合サーバーがリカバリされ、注意して使用されます。 トランザクションを適用したインスタンスを追加しようとすると、データ破損のリスクがあります。

これらの要因の組合せは、Cluster.addInstance() の発行時にインスタンスがクラスタに参加する方法に影響します。recoveryMethod オプションはデフォルトで auto に設定されています。つまり、MySQL Shell 対話モードでは、クラスタはクラスタからインスタンスをリカバリするための最適な方法を選択し、続行方法を指示するプロンプトが表示されます。 つまり、クラスタでは、最適なアプローチとサーバーでサポートされている内容に基づいて、MySQL クローンまたは増分リカバリを使用することをお薦めします。 対話モードを使用せず、MySQL Shell をスクリプト化している場合は、recoveryMethod を使用するリカバリのタイプに設定する必要があります - clone または incremental。このセクションでは、考えられる様々なシナリオについて説明します。

MySQL Shell を対話モードで使用している場合、インスタンスを追加するために使用可能なすべてのオプションを含むメインプロンプトは次のとおりです:

```
Please select a recovery method [C]lone/[I]ncremental recovery/[A]bort (default Clone):
```

前述の要因によっては、これらのオプションの一部が提供されない場合があります。 このセクションで後述するシナリオでは、提供されるオプションについて説明します。 このプロンプトで提供されるオプションは次のとおりです:

- クローン: クラスタに追加するインスタンスにドナーをクローニングし、インスタンスに含まれるトランザクションを削除するには、このオプションを選択します。 MySQL クローンプラグインが自動的にインストールされます。InnoDB クラスタ リカバリアカウントは、必要な BACKUP_ADMIN 権限で作成されます。 空 (トランザクションを処理していない) または保持しないトランザクションを含むインスタンスを追加する場合は、クローンオプションを選択します。 次に、クラスタは MySQL クローンを使用して、参加しているインスタンスをドナークラスタメンバーからのスナップショットで完全に上書きします。 この方法をデフォルトで使用し、このプロンプトを無効にするには、cluster recoveryMethod オプションを clone に設定します。
- 増分リカバリでは、このオプションを選択して増分リカバリを使用し、非同期レプリケーションを使用して、クラスタで処理されたすべてのトランザクションを結合インスタンスにリカバリします。 増分リカバリは、クラスタで処理されたすべての更新が GTID を有効にして実行されたことが確実な場合に適しており、パージされたトランザクションはなく、新しいインスタンスにはクラスタまたはそのサブセットと同じ GTID セットが含まれています。この方法をデフォルトで使用するには、recoveryMethod オプションを incremental に設定します。

前述の要因の組合せは、次のようにプロンプトで使用可能なこれらのオプションに影響します:

> 注記
>
> group_replication_clone_threshold システム変数が AdminAPI の外部で手動で変更されている場合、クラスタは次のシナリオのかわりにクローンリカバリを使用することを決定できます。

- 次の場合
  - 増分リカバリが可能です
  - 増分リカバリが適切ではありません
  - クローンがサポートされています

  いずれかのオプションを選択できます。 デフォルトの MySQL クローンを使用することをお薦めします。
- 次の場合
  - 増分リカバリが可能です
  - 増分リカバリが適切です

  プロンプトが表示されず、増分リカバリが使用されます。
- 次の場合
  - 増分リカバリが可能です
  - 増分リカバリが適切ではありません
  - クローンはサポートされていないか、無効です

MySQL クローンを使用してインスタンスをクラスタに追加することはできません。 プロンプトが表示され、増分リカバリを続行することをお薦めします。

- 次の場合
  - 増分リカバリはできません
  - クローンはサポートされていないか、無効です

  インスタンスをクラスタに追加できず、およびエラー: ターゲットインスタンスをターゲットクラスタに追加する前に、クローニングするか完全にプロビジョニングする必要があります。 Cluster.addInstance: インスタンスプロビジョニングが必要です (RuntimeError) が示されます。 これは、バイナリログがすべてのクラスタインスタンスからパージされた結果である可能性があります。 クラスタをアップグレードするか、disableClone オプションを false に設定して、MySQL クローンを使用することをお薦めします。

- 次の場合
  - 増分リカバリはできません
  - クローンがサポートされています

  MySQL クローンは、インスタンスをクラスタに追加するためにのみ使用できます。 これは、たとえばパージされたときに、クラスタにバイナリログがないことが原因である可能性があります。

プロンプトからオプションを選択すると、デフォルトで、クラスタからトランザクションをリカバリするインスタンスの進行状況が表示されます。 この監視により、リカバリフェーズが機能していること、およびインスタンスがクラスタに参加してオンラインになるまでにかかる時間を確認できます。 リカバリフェーズの監視を取り消すには、CONTROL+C を発行します。

Cluster.checkInstanceState() および MySQL クローン

MySQL クローンを使用しているクラスタに対してインスタンスを検証するために Cluster.checkInstanceState() 操作を実行するときに、インスタンスにバイナリログがない場合 (たとえば、パージされたがクローンが使用可能で無効化されていない (disableClone は false) は、クローンを使用できることを示す警告を表示します。 例:

```
The cluster transactions cannot be recovered on the instance, however,
Clone is available and can be used when adding it to a cluster.

{
"reason": "all_purged",
"state": "warning"
}
```

同様に、クローンが使用できないか無効になっており、バイナリログがパージされたなどの理由で使用できないインスタンスでは、出力には次のものが含まれます:

```
The cluster transactions cannot be recovered on the instance.

{
"reason": "all_purged",
"state": "warning"
}
```

dba.checkInstanceConfiguration() および MySQL クローン

MySQL クローンは使用可能だが無効になっているインスタンスに対して dba.checkInstanceConfiguration() 操作を実行すると、警告が表示されます。

6.2.2.3 グループレプリケーションデプロイメントの採用

グループレプリケーションの既存のデプロイメントがあり、それを使用してクラスタを作成する場合は、dba.createCluster() 関数に adoptFromGR オプションを渡します。 作成された InnoDB クラスタ は、レプリケーショングループが単一プライマリとして実行されているか、マルチプライマリとして実行されているかに一致します。

既存のグループレプリケーショングループを採用するには、MySQL Shell を使用してグループメンバーに接続します。 次の例では、単一プライマリグループが採用されています。 gr-member-1 がグループプライマリとして機能している間に、セカンダリインスタンスである gr-member-2 に接続します。 adoptFromGR オプションを渡して、dba.createCluster() を使用してクラスタを作成します。 例:

```
mysql-js> var cluster = dba.createCluster('prodCluster', {adoptFromGR: true});

A new InnoDB cluster will be created on instance 'root@gr-member-2:3306'.

Creating InnoDB cluster 'prodCluster' on 'root@gr-member-2:3306'...
Adding Seed Instance...

Cluster successfully created. Use cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to
one server failure.
```

> **ヒント**
>
> インスタンスに super_read_only=ON がある場合は、AdminAPI で super_read_only=OFF を設定できることを確認する必要がある場合があります。 詳しくはスーパー読取り専用およびインスタンスをご覧ください。

新しいクラスタはグループのモードと一致します。 採用されたグループがシングルプライマリモードで実行されていた場合は、シングルプライマリクラスタが作成されます。 採用されたグループがマルチプライマリモードで実行されていた場合は、マルチプライマリクラスタが作成されます。

6.2.3 InnoDB クラスタ の監視

このセクションでは、AdminAPI を使用して InnoDB クラスタ を監視する方法について説明します。

- Cluster.describe() の使用
- Cluster.status() によるクラスタステータスの確認
- リカバリ操作の監視
- InnoDB クラスタ およびグループのレプリケーションプロトコル
- インスタンスでの MySQL バージョンの確認

Cluster.describe() の使用

InnoDB クラスタ 自体の構造に関する情報を取得するには、Cluster.describe() 関数を使用します:

```
mysql-js> cluster.describe();
{
    "clusterName": "testCluster",
    "defaultReplicaSet": {
        "name": "default",
        "topology": [
            {
                "address": "ic-1:3306",
                "label": "ic-1:3306",
                "role": "HA"
            },
            {
                "address": "ic-2:3306",
                "label": "ic-2:3306",
                "role": "HA"
            },
            {
                "address": "ic-3:3306",
                "label": "ic-3:3306",
                "role": "HA"
            }
        ]
    }
}
```

この関数の出力には、すべての構成情報などを含む InnoDB クラスタ の構造が表示されます。 アドレス、ラベルおよびロールの値は、Cluster.status() によるクラスタステータスの確認 で説明されている値と一致します。

Cluster.status() によるクラスタステータスの確認

クラスタオブジェクトには、クラスタの実行方法を確認できる status() メソッドが用意されています。 InnoDB クラスタ のステータスを確認するには、そのインスタンスに接続して InnoDB クラスタ オブジェクトへの参照を取得する必要があります。 ただし、クラスタの構成を変更する場合は、R/W インスタンスに接続する必要があります。 status() を発行すると、接続しているサーバーインスタンスが認識しているクラスタのビューに基づいてクラスタのステータスが取得され、ステータスレポートが出力されます。

> **重要**
>
> クラスタ内のインスタンスの状態は、ステータスレポートに表示される情報に直接影響します。 したがって、接続しているインスタンスのステータスが ONLINE であることを確認してください。

InnoDB クラスタ の実行方法の詳細は、クラスタ status() メソッドを使用してください:

```
mysql-js> var cluster = dba.getCluster()
mysql-js> cluster.status()
{
    "clusterName": "testcluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "ic-1:3306",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "ic-1:3306": {
                "address": "ic-1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-2:3306": {
                "address": "ic-2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-3:3306": {
                "address": "ic-3:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    },
    "groupInformationSourceMember": "mysql://icadmin@ic-1:3306"
}
```

Cluster.status() の出力には、次の情報が表示されます:

- clusterName: dba.createCluster() 中にこのクラスタに割り当てられた名前。
- defaultReplicaSet : InnoDB クラスタ に属し、データセットを含むサーバーインスタンス。
- primary: クラスタがシングルプライマリモードで動作している場合にのみ表示されます。 現在のプライマリインスタンスのアドレスを表示します。 このフィールドが表示されない場合、クラスタはマルチプライマリモードで動作しています。
- ssl: セキュアな接続がクラスタで使用されているかどうか。 createCluster() または addInstance() 中に memberSslMode オプションがどのように構成されたかに応じて、REQUIRED または DISABLED の値が表示され

ます。 このパラメータによって返される値は、インスタンス上の group_replication_ssl_mode サーバー変数の値に対応します。 クラスタの保護を参照してください。

- status: クラスタのこの要素のステータス。 クラスタ全体について、このクラスタによって提供される高可用性について説明します。 ステータスは次のいずれかです:
  - ONLINE: インスタンスはオンラインで、クラスタに参加しています。
  - OFFLINE: インスタンスは他のインスタンスへの接続を失いました。
  - RECOVERING: インスタンスは、ONLINE メンバーになる前に必要なトランザクションを取得して、クラスタと同期しようとしています。
  - UNREACHABLE: インスタンスはクラスタとの通信を失いました。
  - ERROR: リカバリフェーズ中またはトランザクションの適用中にインスタンスでエラーが発生しました。

    > **重要**
    >
    > インスタンスが ERROR 状態になると、super_read_only オプションは ON に設定されます。 ERROR の状態のままにするには、super_read_only=OFF を使用してインスタンスを手動で構成する必要があります。

  - (MISSING): 構成済クラスタの一部であるが、現在使用できないインスタンスの状態。

    > **注記**
    >
    > MISSING の状態は InnoDB クラスタ に固有であり、Group Replication によって生成される状態ではありません。MySQL Shell はこの状態を使用して、メタデータに登録されているが、ライブクラスタビューに見つからないインスタンスを示します。

- topology: クラスタに追加されたインスタンス。
- Host name of instance : インスタンスのホスト名 (localhost:3310 など)。
- role: このインスタンスがクラスタ内で提供する機能。 現在は HA のみで、高可用性を実現しています。
- mode: サーバーが読取り/書込み ("R/W") か読取り専用 ("R/O") か。 バージョン 8.0.17 から、これはインスタンス上の super_read_only 変数の現在の状態、およびクラスタにクォーラムがあるかどうかから導出されます。 以前のバージョンでは、mode の値は、インスタンスがプライマリインスタンスとして機能していたかセカンダリインスタンスとして機能していたかから導出されていました。 通常、インスタンスがプライマリの場合、モードは R/W で、インスタンスがセカンダリの場合、モードは R/O です。 表示可能なクォーラムがないクラスタ内のインスタンスは、super_read_only 変数の状態に関係なく、R/O としてマークされます。
- groupInformationSourceMember: URI のような接続文字列として表示される、クラスタに関する情報の取得に使用される内部接続。 通常は、クラスタの作成に最初に使用される接続です。

クラスタの詳細を表示するには、extended オプションを使用します。 バージョン 8.0.17 からは、extended オプションで整数またはブール値がサポートされます。 Cluster.status({'extended':value}) が提供する追加情報を構成するには、次の値を使用:

- 0: 追加情報を無効にします (デフォルト)
- 1: には、Group Replication Protocol Version、Group name、クラスタメンバー UUID、クラスタメンバー役割および Group Replication によって報告される状態、およびフェンシングされたシステム変数のリストに関する情報が含まれています
- 2: 接続および適用者によって処理されたトランザクションに関する情報が含まれます
- 3: には、各クラスタメンバーによって実行されるレプリケーションに関するより詳細な統計が含まれます。

ブール値を使用して extended を設定することは、整数値 0 および 1 を設定することと同等です。 8.0.17 より前のバージョンでは、extended オプションはブールのみでした。 同様に、以前のバージョンでは、queryMembers ブールオプションを使用して、クラスタ内のインスタンスの詳細情報を提供していました。これは、extended を 3 に設定することと同等です。 queryMembers オプションは非推奨であり、将来のリリースで削除される予定です。

Cluster.status({'extended':1}) を発行するか、extended オプションが true に設定されている場合、出力には次のものが含まれます:

- defaultReplicaSet オブジェクトの次の追加属性:
  - GRProtocolVersion は、クラスタで使用されるグループレプリケーションプロトコルバージョンです。

    > **ヒント**
    >
    > InnoDB クラスタ は、自動的に使用される Group Replication Protocol のバージョンを管理します。詳細は、InnoDB クラスタ およびグループのレプリケーションプロトコル を参照してください。

  - groupName はグループ名 (UUID) です。
- topology オブジェクトの各オブジェクトについて、次の追加属性:
  - fenceSysVars は、AdminAPI によって構成されたフェンシングされたシステム変数の名前を含むリストです。 現在考慮されるフェンシングされたシステム変数は、read_only、super_read_only および offline_mode です。 システム変数は、その値に関係なくリストされます。
  - インスタンスごとの instanceErrors。インスタンスで検出可能な診断情報が表示されます。 たとえば、インスタンスがセカンダリで、super_read_only 変数が ON に設定されていない場合、警告が表示されます。 この情報は、エラーのトラブルシューティングに使用できます。
  - memberId 各クラスタメンバー UUID。
  - Group Replication プラグインによって報告されたメンバーロールを memberRole します。replication_group_members テーブルの MEMBER_ROLE カラムを参照してください。
  - Group Replication プラグインによって報告されたメンバー状態を memberState します。replication_group_members テーブルの MEMBER_STATE カラムを参照してください。

リカバリおよび通常のトランザクションの I/O,アプライヤワーカースレッド統計とラグ、適用側コーディネータ統計 (パラレルレプリケーションアプライヤが有効な場合)、エラー、および受信側と適用側のスレッドからのその他の情報に関する情報を表示するには、extended に 2 または 3 の値を使用します。 これらの値を使用すると、クラスタ内の各インスタンスへの接続がオープンされ、追加のインスタンス固有の統計をクエリーすることができます。 出力に含まれる正確な統計は、インスタンスの状態と構成およびサーバーバージョンによって異なります。この情報は、replication_group_member_stats テーブルに示されている情報と一致します。詳細は、一致するカラムの説明を参照してください。 ONLINE であるインスタンスには、出力に transactions セクションが含まれます。RECOVERING であるインスタンスには、出力に recovery セクションが含まれます。 どちらの場合も、extended を 2 に設定すると、これらのセクションには次の内容を含めることができます:

- appliedCount: COUNT_TRANSACTIONS_REMOTE_APPLIED を参照
- checkedCount: COUNT_TRANSACTIONS_CHECKED を参照
- committedAllMembers: TRANSACTIONS_COMMITTED_ALL_MEMBERS を参照
- conflictsDetectedCount: COUNT_CONFLICTS_DETECTED を参照
- inApplierQueueCount: COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE を参照
- inQueueCount: COUNT_TRANSACTIONS_IN_QUEUE を参照
- lastConflictFree: LAST_CONFLICT_FREE_TRANSACTION を参照
- proposedCount: COUNT_TRANSACTIONS_LOCAL_PROPOSED を参照
- rollbackCount: COUNT_TRANSACTIONS_LOCAL_ROLLBACK を参照

extended を 3 に設定すると、connection セクションに replication_connection_status テーブルの情報が表示されます。 値 3 は、非推奨の queryMembers オプションを true に設定することと同等です。 connection セクションには、次のものを含めることができます:

currentlyQueueing セクションには、現在キューに入れられているトランザクションに関する情報が表示されます:

- immediateCommitTimestamp: QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP を参照
- immediateCommitToNowTime: QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP から NOW() を引いた値を参照
- originalCommitTimestamp: QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP を参照
- originalCommitToNowTime: QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP から NOW() を引いた値を参照
- startTimestamp: QUEUEING_TRANSACTION_START_QUEUE_TIMESTAMP を参照
- transaction: QUEUEING_TRANSACTION を参照
- lastHeartbeatTimestamp: LAST_HEARTBEAT_TIMESTAMP を参照

lastQueued セクションには、最後にキューに入れられたトランザクションに関する情報が表示されます:

- endTimestamp: LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP を参照
- immediateCommitTimestamp: LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP を参照
- immediateCommitToEndTime: LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP マイナス NOW()
- originalCommitTimestamp: LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP を参照
- originalCommitToEndTime: LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP マイナス NOW()
- queueTime: LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP マイナス LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP
- startTimestamp: LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP を参照
- transaction: LAST_QUEUED_TRANSACTION を参照
- receivedHeartbeats: COUNT_RECEIVED_HEARTBEATS を参照
- receivedTransactionSet: RECEIVED_TRANSACTION_SET を参照
- threadId: THREAD_ID を参照

マルチスレッドレプリカを使用しているインスタンスには、ワーカースレッドに関する情報を含む workers セクションがあり、replication_applier_status_by_worker テーブルに表示される情報と一致します。

lastApplied セクションには、ワーカーによって最後に適用されたトランザクションに関する次の情報が表示されます:

- applyTime: LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP から LAST_APPLIED_TRANSACTION_START_APPLY_TIMESTAMP を引いた値を参照
- endTimestamp: LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP を参照
- immediateCommitTimestamp: LAST_APPLIED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP を参照
- immediateCommitToEndTime: LAST_APPLIED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP から NOW() を引いた値を参照
- originalCommitTimestamp: LAST_APPLIED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP を参照
- originalCommitToEndTime: LAST_APPLIED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP から NOW() を引いた値を参照
- startTimestamp: LAST_APPLIED_TRANSACTION_START_APPLY_TIMESTAMP を参照

- transaction: LAST_APPLIED_TRANSACTION を参照

currentlyApplying セクションには、ワーカーによって現在適用されているトランザクションに関する次の情報が表示されます:

- immediateCommitTimestamp: APPLYING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP を参照
- immediateCommitToNowTime: APPLYING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP から NOW() を引いた値を参照
- originalCommitTimestamp: APPLYING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP を参照
- originalCommitToNowTime: APPLYING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP から NOW() を引いた値を参照
- startTimestamp: APPLYING_TRANSACTION_START_APPLY_TIMESTAMP を参照
- transaction: APPLYING_TRANSACTION を参照

lastProcessed セクションには、ワーカーが最後に処理したトランザクションに関する次の情報が表示されます:

- bufferTime: LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP マイナス LAST_PROCESSED_TRANSACTION_START_BUFFER_TIMESTAMP
- endTimestamp: LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP を参照
- immediateCommitTimestamp: LAST_PROCESSED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP を参照
- immediateCommitToEndTime: LAST_PROCESSED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP マイナス LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP
- originalCommitTimestamp: LAST_PROCESSED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP を参照
- originalCommitToEndTime: LAST_PROCESSED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP マイナス LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP
- startTimestamp: LAST_PROCESSED_TRANSACTION_START_BUFFER_TIMESTAMP を参照
- transaction: LAST_PROCESSED_TRANSACTION を参照

パラレルレプリケーションアプライヤが有効になっている場合は、transactions または recovery の workers 配列内のオブジェクト数が構成済ワーカー数と一致し、追加のコーディネータオブジェクトが含まれます。 表示される情報は、replication_applier_status_by_coordinator テーブルの情報と一致します。 オブジェクトには、次のものを含めることができます:

currentlyProcessing セクションには、ワーカーが処理しているトランザクションに関する次の情報が表示されます:

- immediateCommitTimestamp: PROCESSING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP を参照
- immediateCommitToNowTime: PROCESSING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP マイナス NOW()
- originalCommitTimestamp: PROCESSING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP を参照
- originalCommitToNowTime: PROCESSING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP マイナス NOW()
- startTimestamp: PROCESSING_TRANSACTION_START_BUFFER_TIMESTAMP を参照
- transaction: PROCESSING_TRANSACTION を参照

replication_applier_status_by_worker テーブルでエラーが検出された場合、worker オブジェクトには次の情報が含まれます:

- lastErrno: LAST_ERROR_NUMBER を参照
- lastError: LAST_ERROR_MESSAGE を参照

- lastErrorTimestamp: LAST_ERROR_TIMESTAMP を参照

replication_connection_status テーブルでエラーが検出された場合、connection オブジェクトには次の情報が含まれます:

- lastErrno: LAST_ERROR_NUMBER を参照
- lastError: LAST_ERROR_MESSAGE を参照
- lastErrorTimestamp: LAST_ERROR_TIMESTAMP を参照

replication_applier_status_by_coordinator テーブルでエラーが検出された場合、coordinator オブジェクトには次の情報が含まれます:

- lastErrno: LAST_ERROR_NUMBER を参照
- lastError: LAST_ERROR_MESSAGE を参照
- lastErrorTimestamp: LAST_ERROR_TIMESTAMP を参照

リカバリ操作の監視

Cluster.status() の出力には、RECOVERING 状態のインスタンスのリカバリ操作の進行状況に関する情報が表示されます。 MySQL クローンまたは増分リカバリのいずれかを使用してリカバリするインスタンスの情報が表示されます。 次のフィールドをモニターします:

- recoveryStatusText フィールドには、使用されているリカバリのタイプに関する情報が含まれます。 MySQL クローンが機能している場合、このフィールドには「「クローニング進行中」」と表示されます。 増分リカバリが機能している場合、このフィールドには「「分散リカバリの進行中」」と表示されます。
- MySQL クローンが使用されている場合、recovery フィールドには次のフィールドを含むディクショナリが含まれます:
  - cloneStartTime: クローンプロセスの開始のタイムスタンプ
  - cloneState: クローン進行状況の状態
  - currentStage: クローンプロセスが到達した現在のステージ
  - currentStageProgress: 現在のステージの進行状況 (完了率)
  - currentStageState: 現在のステージ状態

  簡潔にするために切り捨てられた Cluster.status() 出力の例:

```
...
"recovery": {
"cloneStartTime": "2019-07-15 12:50:22.730",
"cloneState": "In Progress",
"currentStage": "FILE COPY",
"currentStageProgress": 61.726837675213865,
"currentStageState": "In Progress"
},
"recoveryStatusText": "Cloning in progress",
...
```

- 増分リカバリが使用されており、extended オプションが 1 以上に設定されている場合、recovery フィールドには次のフィールドを含むディクショナリが含まれます:
  - state: group_replication_recovery チャネルの状態
  - recoveryChannel : 増分リカバリを実行しているインスタンス、またはリカバリチャネルのステータスがオフでないインスタンスに対して表示されます。 増分リカバリでは受信者スレッドを使用してソースからトランザクションを受信し、適用者スレッドでは受信したトランザクションをインスタンスに適用します。 次の情報が表示されます:

- applierQueuedTransactionSetSize: 適用を待機している、現在キューに入っているトランザクションの数。
- applierState: レプリケーションアプライアンスの現在の状態 (ON または OFF)。
- applierStatus: アプライヤスレッドの現在のステータス。 applierThreadState フィールドに表示される状態の集計。 次のいずれかを指定できます:
  - APPLIED_ALL: 適用を待機中のキュー済トランザクションはありません
  - APPLYING: 適用中のトランザクションがあります
  - ON: スレッドは接続されており、キューに入っているトランザクションはありません
  - ERROR: トランザクションの適用中にエラーが発生しました
  - OFF: アプライヤスレッドが無効です
- applierThreadState: 任意の適用者スレッドの現在の状態。 アプライヤスレッドが実行している処理に関する詳細情報を提供します。 詳細は、レプリケーション SQL スレッドの状態を参照してください。
- receiverStatus: 受信者スレッドの現在のステータス。 receiverThreadState フィールドに表示される状態の集計。 次のいずれかを指定できます:
  - ON: 受信側スレッドは正常に接続され、受信する準備ができています
  - CONNECTING: 受信者スレッドがソースに接続しています
  - ERROR: トランザクションの受信中にエラーが発生しました
  - OFF: 受信者スレッドが正常に切断されました
- receiverThreadState: 受信者スレッドの現在の状態。 受信者スレッドが実行している処理に関する詳細情報を提供します。 詳細は、レプリケーション I/O スレッドの状態を参照してください。
- source: 適用されるトランザクションのソース。

簡潔にするために切り捨てられた Cluster.status() 出力の例:

```
...
"recovery": {
                "recoveryChannel": {
                    "applierQueuedTransactionSetSize": 2284,
                    "applierStatus": "APPLYING",
                    "applierThreadState": "Opening tables",
                    "receiverStatus": "ON",
                    "receiverThreadState": "Queueing master event to the relay log",
                    "source": "ic-2:3306"
                },
                "state": "ON"
            },
...
```

InnoDB クラスタ およびグループのレプリケーションプロトコル

MySQL 8.0.16 から、グループレプリケーションにはグループの通信プロトコルの概念があります。バックグラウンド情報は、グループ通信プロトコルバージョンの設定 を参照してください。 Group Replication 通信プロトコルのバージョンは通常、明示的に管理する必要があり、グループでサポートする最も古い MySQL Server バージョンに対応するように設定する必要があります。 ただし、クラスタトポロジが AdminAPI 操作を使用して変更されるたびに、InnoDB クラスタ はそのメンバーの通信プロトコルバージョンを自動的かつ透過的に管理します。 クラスタでは、現在クラスタの一部であるか参加しているすべてのインスタンスでサポートされている最新の通信プロトコルバージョンが常に使用されます。

- クラスタに対してインスタンスが追加、削除または再結合されたり、再スキャンまたは再起動操作が実行されると、通信プロトコルバージョンは、現在最も古い MySQL Server バージョンであるインスタンスでサポートされているバージョンに自動的に設定されます。

- クラスタからインスタンスを削除してローリングアップグレードを実行し、それらをアップグレードしてクラスタに再度追加すると、古い MySQL Server バージョンの残りのインスタンスがアップグレード前にクラスタから削除されたときに、通信プロトコルバージョンが自動的にアップグレードされます。

クラスタで使用されている通信プロトコルのバージョンを確認するには、extended オプションを有効にして Cluster.status() 関数を使用します。 クラスタにクォーラムがあり、アクセスできないクラスタメンバーがない場合、通信プロトコルバージョンが GRProtocolVersion フィールドに返されます。

インスタンスでの MySQL バージョンの確認

次の操作では、インスタンスで実行されている MySQL Server のバージョンに関する情報をレポートできます:

- Cluster.status()
- Cluster.describe()
- Cluster.rescan()

動作は、Cluster オブジェクトセッションの MySQL Server バージョンによって異なります。

- Cluster.status()

  次のいずれかの要件が満たされると、topology オブジェクトのインスタンス JSON オブジェクトごとに version 文字列属性が返されます:

  - Cluster オブジェクトの現在のセッションは、8.0.11 以降のバージョンです。
  - Cluster オブジェクトの現在のセッションで、バージョン 8.0.11 より前のバージョンが実行されていますが、extended オプションが 3 に設定されています (または、非推奨の queryMembers が true です)。

  たとえば、バージョン 8.0.16 を実行しているインスタンスでは、次のようになります:

```
"topology": {
    "ic-1:3306": {
        "address": "ic-1:3306",
        "mode": "R/W",
        "readReplicas": {},
        "role": "HA",
        "status": "ONLINE",
        "version": "8.0.16"
}
```

  たとえば、バージョン 5.7.24 を実行しているインスタンスでは、次のようになります:

```
"topology": {
    "ic-1:3306": {
        "address": "ic-1:3306",
        "mode": "R/W",
        "readReplicas": {},
        "role": "HA",
        "status": "ONLINE",
        "version": "5.7.24"
}
```

- Cluster.describe()

  Cluster オブジェクトの現在のセッションがバージョン 8.0.11 以降の場合、topology オブジェクトのインスタンス JSON オブジェクトごとに version 文字列属性が返されます

  たとえば、バージョン 8.0.16 を実行しているインスタンスでは、次のようになります:

```
"topology": [
    {
        "address": "ic-1:3306",
        "label": "ic-1:3306",
        "role": "HA",
        "version": "8.0.16"
    }
```

```
]
```

- Cluster.rescan()

  Cluster オブジェクトの現在のセッションがバージョン 8.0.11 以降で、 Cluster.rescan() 操作によってクラスタに属していないインスタンスが検出された場合、newlyDiscoveredInstance オブジェクトのインスタンス JSON オブジェクトごとに version 文字列属性が返されます。

  たとえば、バージョン 8.0.16 を実行しているインスタンスでは、次のようになります:

```
"newlyDiscoveredInstances": [
    {
        "host": "ic-4:3306",
        "member_id": "82a67a06-2ba3-11e9-8cfc-3c6aa7197deb",
        "name": null,
        "version": "8.0.16"
    }
]
```

6.2.4 インスタンスの操作

このセクションでは、インスタンスに適用される AdminAPI 操作について説明します。 InnoDB クラスタ で使用する前にインスタンスを構成したり、インスタンスの状態を確認したりできます。

- dba.checkInstanceConfiguration() の使用
- dba.configureLocalInstance() でのインスタンスの構成
- インスタンスの状態の確認

dba.checkInstanceConfiguration() の使用

サーバーインスタンスから本番デプロイメントを作成する前に、各インスタンスの MySQL が正しく構成されていることを確認する必要があります。 インスタンスの構成の一部として構成をチェックする dba.configureInstance() に加えて、dba.checkInstanceConfiguration(instance) 関数を使用できます。 これにより、インスタンスの構成を変更せずに、instance が セクション6.2.1「MySQL InnoDB クラスタ の要件」 を満たしていることが保証されます。 インスタンス上のデータはチェックされません。詳細は、インスタンスの状態の確認 を参照してください。

instance への接続に使用するユーザーには、AdminAPI のユーザーの構成 での構成など、適切な権限が必要です。 実行中の MySQL Shell でこれを発行する方法を次に示します:

```
mysql-js> dba.checkInstanceConfiguration('icadmin@ic-1:3306')
Please provide the password for 'icadmin@ic-1:3306': ***
Validating MySQL instance at ic-1:3306 for use in an InnoDB cluster...

This instance reports its own address as ic-1
Clients and other cluster members will communicate with it through this address by default.
If this is not correct, the report_host MySQL system variable should be changed.

Checking whether existing tables comply with Group Replication requirements...
No incompatible tables detected

Checking instance configuration...

Some configuration options need to be fixed:
+--------------------------+---------------+----------------+--------------------------------------------------+
| Variable                 | Current Value | Required Value | Note                                             |
+--------------------------+---------------+----------------+--------------------------------------------------+
| enforce_gtid_consistency | OFF           | ON             | Update read-only variable and restart the server |
| gtid_mode                | OFF           | ON             | Update read-only variable and restart the server |
| server_id                | 1             |                | Update read-only variable and restart the server |
+--------------------------+---------------+----------------+--------------------------------------------------+

Please use the dba.configureInstance() command to repair these issues.

{
    "config_errors": [
        {
            "action": "restart",
```

```
        "current": "OFF",
        "option": "enforce_gtid_consistency",
        "required": "ON"
      },
      {
        "action": "restart",
        "current": "OFF",
        "option": "gtid_mode",
        "required": "ON"
      },
      {
        "action": "restart",
        "current": "1",
        "option": "server_id",
        "required": ""
      }
    ],
    "status": "error"
}
```


クラスタの一部として使用する予定のサーバーインスタンスごとに、このプロセスを繰り返します。 dba.checkInstanceConfiguration() の実行後に生成されたレポートには、続行する前に必要な構成変更に関する情報が表示されます。 レポートの config_error セクションの action フィールドには、構成ファイルに対する変更を検出するために、インスタンス上の MySQL の再起動が必要かどうかが示されます。

dba.configureLocalInstance() でのインスタンスの構成

構成変更の永続化を自動的にサポートしていないインスタンス (設定の永続化 を参照) では、サーバーに接続し、MySQL Shell を実行し、インスタンスにローカルに接続して dba.configureLocalInstance() を発行する必要があります。 これにより、リモートインスタンスに対して次のコマンドを実行した後、MySQL Shell はインスタンスオプションファイルを変更できます:

- dba.configureInstance()
- dba.createCluster()
- Cluster.addInstance()
- Cluster.removeInstance()
- Cluster.rejoinInstance()

> **重要**
>
> インスタンスオプションファイルへの構成変更の永続化に失敗すると、次回の再起動後にインスタンスがクラスタに再参加しなくなる可能性があります。

SSH などを使用してリモートマシンにログインし、root ユーザーとして MySQL Shell を実行してから、ローカル MySQL サーバーに接続することをお薦めします。 たとえば、--uri オプションを使用してローカル instance に接続します:

```
shell> sudo -i mysqlsh --uri=instance
```

または、\connect コマンドを使用してローカルインスタンスにログインします。 次に、instance がローカルインスタンスへの接続情報である dba.configureInstance(instance) を発行して、ローカルインスタンスオプションファイルに加えられた変更を永続化します。

```
mysql-js> dba.configureLocalInstance('icadmin@ic-2:3306')
```

構成変更の永続化を自動的にサポートしていないクラスタ内のインスタンスごとに、このプロセスを繰り返します。 たとえば、構成変更の永続化を自動的にサポートしないクラスタに 2 つのインスタンスを追加する場合は、各サーバーに接続し、インスタンスを再起動する前に InnoDB クラスタ に必要な構成変更を永続化する必要があります。 同様に、インスタンス数の変更など、クラスタ構造を変更する場合は、サーバーインスタンスごとにこのプロセスを繰り返して、クラスタ内のインスタンスごとに InnoDB クラスタ メタデータを更新する必要があります。

インスタンスの状態の確認

cluster.checkInstanceState() 関数を使用すると、インスタンス上の既存のデータがクラスタへの参加を妨げないことを検証できます。 このプロセスは、クラスタによってすでに処理されている GTID と比較して、インスタンスグローバルトランザクション識別子 (GTID) の状態を検証することで機能します。 GTID の詳細は、GTID 形式および格納 を参照してください。 このチェックでは、トランザクションを処理したインスタンスをクラスタに追加できるかどうかを判断できます。

実行中の MySQL Shell でこれを発行する方法を次に示します:

```
mysql-js> cluster.checkInstanceState('icadmin@ic-4:3306')
```

この関数の出力は、次のいずれかです:

- OK 新規: インスタンスは GTID トランザクションを実行していないため、クラスタによって実行された GTID と競合できません
- OK リカバリ可能: インスタンスが、クラスタシードインスタンスの実行済 GTID と競合しない GTID を実行しました
- ERROR diverged: インスタンスは、クラスタシードインスタンスの実行済 GTID と相違する GTID を実行しました
- ERROR lost_transactions: インスタンスには、クラスタシードインスタンスの実行済 GTID より多くの GTID が実行されています

OK ステータスのインスタンスは、インスタンス上のデータがクラスタと一貫性があるため、クラスタに追加できます。 つまり、チェック中のインスタンスは、クラスタによって実行された GTID と競合するトランザクションを実行しておらず、残りのクラスタインスタンスと同じ状態にリカバリできます。

6.2.5 InnoDB クラスタの操作

このセクションでは、InnoDB クラスタ の使用方法および一般的な管理タスクの処理方法について説明します。

- InnoDB クラスタからのインスタンスの削除
- InnoDB クラスタの解決
- クラスタトポロジの変更

InnoDB クラスタからのインスタンスの削除

必要に応じて、いつでもクラスタからインスタンスを削除できます。 これは、次の例のように、Cluster.removeInstance(instance) メソッドを使用して実行できます:

```
mysql-js> cluster.removeInstance('root@localhost:3310')

The instance will be removed from the InnoDB cluster. Depending on the instance
being the Seed or not, the Metadata session might become invalid. If so, please
start a new session to the Metadata Storage R/W instance.

Attempting to leave from the Group Replication group...

The instance 'localhost:3310' was successfully removed from the cluster.
```

オプションで、interactive オプションを渡して、クラスタからのインスタンスの削除を確認するプロンプトを表示するかどうかを制御できます。 対話モードでは、インスタンスにアクセスできない場合に備えて、インスタンスの削除を続行するかどうかを尋ねるプロンプトが表示されます。 cluster.removeInstance() 操作により、ONLINE であるすべてのクラスタメンバーのメタデータおよびインスタンス自体からインスタンスが削除されます。

削除するインスタンスにまだ適用する必要があるトランザクションがある場合、AdminAPI は、トランザクション (GTID) が適用されるまで、MySQL Shell dba.gtidWaitTimeout オプションで構成された秒数まで待機します。 MySQL Shell dba.gtidWaitTimeout オプションのデフォルト値は 60 秒です。デフォルトの変更の詳細は、セクション10.4「MySQL Shell オプションの構成」 を参照してください。 トランザクションの適用を待機しているときに dba.gtidWaitTimeout で定義されたタイムアウト値に達し、force オプションが false (または定義されていない) の場合は、エラーが発行され、削除操作が中断されます。 トランザクションの適用を待機しているときに dba.gtidWaitTimeout で定義されたタイムアウト値に達し、force オプションが true に設定されている場合、操作はエラーなしで続行され、インスタンスがクラスタから削除されます。

> **重要**
>
> force オプションは、未処理のトランザクションや UNREACHABLE であるインスタンスなどのエラーを無視し、クラスタでインスタンスを再利用しない場合にのみ、Cluster.removeInstance(instance) で使用する必要があります。 クラスタからインスタンスを削除するときにエラーを無視すると、インスタンスがクラスタと同期しなくなり、後でクラスタに再参加できなくなる可能性があります。 force オプションは、クラスタでインスタンスを使用しないことを計画している場合にのみ使用します。それ以外の場合は、常にインスタンスのリカバリを試行し、インスタンスが使用可能で正常な場合、つまりステータスが ONLINE の場合にのみ削除する必要があります。

InnoDB クラスタの解決

InnoDB クラスタ を解決するには、読取り/書込みインスタンス (単一プライマリクラスタのプライマリなど) に接続し、Cluster.dissolve() コマンドを使用します。 これにより、クラスタに関連付けられているすべてのメタデータおよび構成が削除され、インスタンスでのグループレプリケーションが無効になります。 インスタンス間でレプリケートされたデータは削除されません。

> **重要**
>
> クラスタのディゾルブを元に戻す方法はありません。 再度作成するには、dba.createCluster() を使用します。

Cluster.dissolve() 操作では、ONLINE または到達可能なインスタンスのみを構成できます。 Cluster.dissolve() コマンドを発行したメンバーがクラスタのメンバーに到達できない場合は、ディゾルブ操作の続行方法を決定する必要があります。 クラスタから欠落として識別されたインスタンスを再結合する可能性がある場合は、ディゾルブ操作を取り消し、まず欠落しているインスタンスをオンラインに戻してから、ディゾルブ操作を続行することを強くお薦めします。 これにより、すべてのインスタンスのメタデータを正しく更新でき、スプリットブレイン状況が発生する可能性がなくなります。 ただし、到達できないクラスタのインスタンスが永続的に残っている場合は、選択肢はなく、ディゾルブ操作を強制することができます。つまり、欠落しているインスタンスは無視され、操作の影響を受けるのはオンラインインスタンスのみです。

> **警告**
>
> クラスタインスタンスを無視するようにディゾルブ操作を強制すると、ディゾルブ操作中に到達できなかったインスタンスが引き続き動作し、スプリットブレイン状況のリスクが発生する可能性があります。 インスタンスが再度オンラインになる可能性がないことが確実な場合にのみ、欠落しているインスタンスを無視するようにディゾルブ操作を強制します。

対話型モードでは、ディゾルブ操作中にクラスタのメンバーにアクセスできない場合、対話型プロンプトが表示されます。次に例を示します:

```
mysql-js> Cluster.dissolve()
The cluster still has the following registered instances:
{
    "clusterName": "testCluster",
    "defaultReplicaSet": {
        "name": "default",
        "topology": [
            {
                "address": "ic-1:3306",
                "label": "ic-1:3306",
                "role": "HA"
            },
            {
                "address": "ic-2:3306",
                "label": "ic-2:3306",
                "role": "HA"
            },
            {
                "address": "ic-3:3306",
                "label": "ic-3:3306",
                "role": "HA"
            }
        ]
    }
```

```
}
WARNING: You are about to dissolve the whole cluster and lose the high
availability features provided by it. This operation cannot be reverted. All
members will be removed from the cluster and replication will be stopped,
internal recovery user accounts and the cluster metadata will be dropped. User
data will be maintained intact in all instances.

Are you sure you want to dissolve the cluster? [y/N]: y

ERROR: The instance 'ic-2:3306' cannot be removed because it is on a '(MISSING)'
state. Please bring the instance back ONLINE and try to dissolve the cluster
again. If the instance is permanently not reachable, then you can choose to
proceed with the operation and only remove the instance from the Cluster
Metadata.

Do you want to continue anyway (only the instance metadata will be removed)?
[y/N]: y

Instance 'ic-3:3306' is attempting to leave the cluster...  Instance 'ic-1:3306'
is attempting to leave the cluster...

WARNING: The cluster was successfully dissolved, but the following instance was
skipped: 'ic-2:3306'. Please make sure this instance is permanently unavailable
or take any necessary manual action to ensure the cluster is fully dissolved.
```

この例では、クラスタは 3 つのインスタンスで構成されており、そのうちの 1 つはディゾルブが発行されたときにオフラインでした。 エラーが捕捉され、続行方法を選択できます。 この場合、欠落している ic-2 インスタンスは無視され、到達可能なメンバーのメタデータが更新されます。

MySQL Shell が非対話モードで実行されている場合 (バッチファイルの実行時など)、force オプションを使用して Cluster.dissolve() 操作の動作を構成できます。 到達不能なインスタンスをディゾルブ操作で強制的に無視するには、次のように発行します:

```
mysql-js> Cluster.dissolve({force: true})
```

到達可能なインスタンスはすべてクラスタから削除され、到達不可能なインスタンスは無視されます。 このセクションでは、クラスタからの欠落しているインスタンスの強制的な削除に関する警告は、この解決操作を強制するこの手法にも同様に適用されます。

Cluster.dissolve() 操作で interactive オプションを使用して、MySQL Shell が実行されているモードをオーバーライドすることもできます。たとえば、バッチスクリプトの実行時に対話型プロンプトが表示されるようにします。 例:

```
mysql-js> Cluster.dissolve({interactive: true})
```

dba.gtidWaitTimeout MySQL Shell オプションでは、Cluster.dissolve() 操作がクラスタからターゲットインスタンスを削除する前にクラスタトランザクションの適用を待機する時間を構成しますが、これはターゲットインスタンスが ONLINE の場合のみです。 削除されるインスタンスのいずれかにクラスタトランザクションが適用されるのを待機しているときにタイムアウトに達すると、強制を除いて エラーが発行されます: true が使用され、その場合は エラーがスキップされます。

> **注記**
>
> cluster.dissolve() を発行すると、Cluster オブジェクトに割り当てられた変数は無効になります。

クラスタトポロジの変更

デフォルトでは、InnoDB クラスタ はシングルプライマリモードで実行され、クラスタには読取りおよび書込みクエリー (R/W) を受け入れる単一のプライマリサーバーがあり、クラスタ内の残りのすべてのインスタンスは読取りクエリー (R/O) のみを受け入れます。 マルチプライマリモードで実行するようにクラスタを構成すると、クラスタ内のすべてのインスタンスがプライマリになります。つまり、読取りクエリーと書込みクエリー (R/W) の両方を受け入れます。 クラスタのすべてのインスタンスで MySQL サーバーバージョン 8.0.15 以降が実行されている場合は、クラスタがオンラインの間にクラスタのトポロジを変更できます。 以前のバージョンでは、構成を変更するためにクラスタを完全に開放して再作成する必要がありました。 これは、オンライングループの構成 で説明されている UDF を介して公開されるグループアクションコーディネータを使用するため、オンライングループを構成するためのルールに従う必要があります。

> 注記
>
> マルチプライマリモードは拡張モードとみなされます

通常、単一プライマリクラスタは、予期しない停止などが原因で、現在のプライマリが予期せずクラスタを離れたときに新しいプライマリを選択します。 選択プロセスは通常、新しいプライマリになる現在のセカンダリを選択するために使用されます。 選択プロセスをオーバーライドし、特定のサーバーを強制的に新しいプライマリにするには、Cluster.setPrimaryInstance(instance) 関数を使用します。ここで、instance は、新しいプライマリになるインスタンスへの接続を指定します。 これにより、基礎となるグループレプリケーショングループを構成して、選択プロセスをバイパスして特定のインスタンスを新しいプライマリとして選択できます。

次の操作を使用して、単一プライマリとマルチプライマリの間でクラスタが実行されているモード (トポロジとも呼ばれる) を変更できます:

- Cluster.switchToMultiPrimaryMode():クラスタをマルチプライマリモードに切り替えます。 すべてのインスタンスがプライマリになります。
- Cluster.switchToSinglePrimaryMode([instance]):クラスタをシングルプライマリモードに切り替えます。 instance が指定されている場合は、プライマリになり、他のすべてのインスタンスがセカンダリになります。 instance が指定されていない場合、新しいプライマリはメンバーの重みが最も高いインスタンス (およびメンバーの重みが結び付けられている場合は UUID が最も低いインスタンス) です。

6.2.6 InnoDB クラスタ の構成

このセクションでは、AdminAPI を使用して InnoDB クラスタ を構成する方法について説明します。

- InnoDB クラスタ のオプションの設定
- InnoDB クラスタs のカスタマイズ
- 選任プロセスの設定
- フェイルオーバーの一貫性の構成
- インスタンスの自動再結合の構成
- パラレルレプリケーションアプリケーションの構成
- クラスタの保護
- サーバーの許可リストの作成

InnoDB クラスタ のオプションの設定

インスタンスがオンラインのときに、InnoDB クラスタ の設定を確認および変更できます。 クラスタの現在の設定を確認するには、次の操作を使用します:

- Cluster.options():クラスタとそのインスタンスの構成オプションをリストします。 ブールオプション all を指定して、すべての Group Replication システム変数に関する情報を出力に含めることもできます。

InnoDB クラスタ のオプションは、インスタンスをオンラインのまま、クラスタレベルまたはインスタンスレベルで構成できます。 これにより、InnoDB クラスタ オプションを変更するために、インスタンスを削除して再構成し、再度追加する必要がなくなります。 次の操作を使用します:

- Cluster.setOption(option, value):すべてのクラスタインスタンスの設定をグローバルに変更するか、clusterName などのクラスタグローバル設定を変更します。
- 個々のクラスタインスタンスの設定を変更する Cluster.setInstanceOption(instance, option, value)

リストされている操作で InnoDB クラスタ オプションを使用する方法は、オプションをすべてのインスタンスで同じになるように変更できるかどうかによって異なります。 これらのオプションは、クラスタレベル (すべてのインスタンス) とインスタンスレベルの両方で変更できます:

- autoRejoinTries: 促進後にインスタンスがクラスタへの再参加を試行する回数を定義する整数値。 インスタンスの自動再結合の構成を参照してください。
- exitStateAction: Group Replication の終了状態アクションを示す文字列値。 インスタンスの自動再結合の構成を参照してください。
- memberWeight: フェイルオーバー時の自動プライマリ選択の重みの割合を示す整数値。 選任プロセスの設定を参照してください。
- tag:option : クラスタに関連付ける組込みタグおよびユーザー定義タグ。 セクション6.2.9「メタデータのタグ付け」を参照してください。

これらのオプションは、クラスタレベルでのみ変更できます:

- clusterName: クラスタ名を定義する文字列値
- disableClone: クラスタでクローンの使用を無効にするために使用されるブール値。 dba.createCluster() および MySQL クローンを参照してください。
- expelTimeout: クラスタから削除する前に、クラスタメンバーが応答しないメンバーを待機する期間 (秒) を定義する整数値。 クラスタの作成を参照してください。
- failoverConsistency: クラスタが提供する一貫性保証を示す文字列値。 インスタンスの自動再結合の構成を参照してください。

このオプションは、インスタンスごとのレベルでのみ変更できます:

- label: インスタンスの文字列識別子

InnoDB クラスタs のカスタマイズ

クラスタを作成してインスタンスを追加すると、グループ名、ローカルアドレス、シードインスタンスなどの値が AdminAPI によって自動的に構成されます。 これらのデフォルト値はほとんどのデプロイメントで推奨されますが、上級ユーザーは次のオプションを dba.createCluster() および Cluster.addInstance() に渡すことでデフォルトをオーバーライドできます。

InnoDB クラスタ によって作成されたレプリケーショングループの名前をカスタマイズするには、dba.createCluster() コマンドに groupName オプションを渡します。 これにより、group_replication_group_name システム変数が設定されます。 名前は有効な UUID である必要があります。

インスタンスが他のインスタンスからの接続用に提供するアドレスをカスタマイズするには、localAddress オプションを dba.createCluster() および cluster.addInstance() コマンドに渡します。 host:port の形式でアドレスを指定します。 これにより、インスタンスに group_replication_local_address システム変数が設定されます。 アドレスは、クラスタ内のすべてのインスタンスからアクセス可能であり、内部クラスタ通信専用に予約されている必要があります。 つまり、インスタンスとの通信にこのアドレスを使用しないでください。

インスタンスがクラスタに参加するときにシードとして使用されるインスタンスをカスタマイズするには、groupSeeds オプションを dba.createCluster() および Cluster.addInstance() 操作に渡します。 シードインスタンスは、新しいインスタンスがクラスタに参加したときに接続され、新しいインスタンスにデータを提供するために使用されます。 シードインスタンスのアドレスは、host1:port1、host2:port2 などのカンマ区切りリストとして指定されます。 これにより、group_replication_group_seeds システム変数が構成されます。 新しいインスタンスがクラスタに追加されると、必要に応じて新しいインスタンスを使用してグループに再度参加できるように、このインスタンスのローカルアドレスがすべてのオンラインクラスタメンバーのグループシードのリストに自動的に追加されます。

> 注記
>
> シードリスト内のインスタンスは、リストに表示される順序に従って使用されます。 つまり、ユーザー指定のシードが最初に使用され、自動的に追加されたインスタンスよりも優先されます。

詳細は、これらの AdminAPI オプションで構成されるシステム変数のドキュメントを参照してください。

選任プロセスの設定

オプションで、単一プライマリクラスタが新しいプライマリを選択する方法を構成できます。たとえば、あるインスタンスをフェイルオーバー先の新しいプライマリとして優先できます。 memberWeight オプションを使用して、クラスタの作成時に dba.createCluster() および Cluster.addInstance() メソッドに渡します。 memberWeight オプションは、フェイルオーバー時の自動プライマリ選択のパーセンテージ加重である 0 から 100 までの整数値を受け入れます。 インスタンスに memberWeight によって設定されたより大きい事前番号がある場合、単一プライマリクラスタでプライマリとして選択される可能性が高くなります。 プライマリ選択が行われると、複数のインスタンスが同じ memberWeight 値を持つ場合、インスタンスは辞書順 (最低) でサーバー UUID に基づいて優先順位が付けられ、最初のインスタンスが選択されます。

memberWeight の値を設定すると、インスタンスの group_replication_member_weight システム変数が構成されます。 グループレプリケーションでは、値の範囲が 0 から 100 に制限され、高い値または低い値が指定された場合は自動的に調整されます。 値が指定されていない場合、グループレプリケーションではデフォルト値の 50 が使用されます。 詳しくはシングルプライマリモードをご覧ください。

たとえば、現在のプライマリである ic-1 が予期せず memberWeight を使用する場合に、ic-3 がフェイルオーバー先の優先インスタンスであるクラスタを構成するには、次のようにします:

```
dba.createCluster('cluster1', {memberWeight:35})
var mycluster = dba.getCluster()
mycluster.addInstance('icadmin@ic2', {memberWeight:25})
mycluster.addInstance('icadmin@ic3', {memberWeight:50})
```

フェイルオーバーの一貫性の構成

グループレプリケーションでは、プライマリフェイルオーバーが単一プライマリモードで発生した場合にフェイルオーバー保証 (最終的または「「書き込みを読む」」) を指定できます (トランザクション一貫性保証の構成 を参照)。作成時に InnoDB クラスタ のフェイルオーバー保証を構成するには、consistency オプション (バージョン 8.0.16 より前のこのオプションは failoverConsistency オプションで、現在は非推奨) を dba.createCluster() 操作に渡して、シードインスタンスで group_replication_consistency システム変数を構成します。 このオプションは、単一プライマリグループで新しいプライマリが選択されたときに使用される新しいフェンシングメカニズムの動作を定義します。 フェンシングは、古いプライマリ (「「書き込みを読む」」とも呼ばれる) からの保留中の変更のバックログが適用されるまで、新しいプライマリからの接続の書込みおよび読取りを制限します。 フェンシングメカニズムが設定されている間は、バックログが適用されている間、アプリケーションは短時間前に進む時間を事実上認識しません。 これにより、アプリケーションは新しく選択されたプライマリから失効した情報を読み取らないようになります。

consistency オプションは、ターゲット MySQL サーバーのバージョンが 8.0.14 以上で、consistency オプションで構成されたクラスタに追加されたインスタンスが、そのオプションをサポートするすべてのクラスタメンバーで group_replication_consistency が同じになるように自動的に構成されている場合にのみサポートされます。
変数のデフォルト値は Group Replication によって制御され、EVENTUAL の場合は、consistency オプションを BEFORE_ON_PRIMARY_FAILOVER に変更してフェンシングメカニズムを有効にします。 または、consistency=0 for EVENTUAL および consistency=1 for BEFORE_ON_PRIMARY_FAILOVER を使用します。

> 注記
>
> マルチプライマリ InnoDB クラスタ で consistency オプションを使用しても効果はありませんが、後でクラスタをシングルプライマリモードに変更できるため、Cluster.switchToSinglePrimaryMode() 操作を使用できます。

インスタンスの自動再結合の構成

MySQL 8.0.16 以降を実行しているインスタンスでは、グループレプリケーションの自動再結合機能がサポートされているため、明示的にクラスタに自動的に再参加するようにインスタンスを構成できます。 背景情報は、障害検出およびネットワークパーティション化へのレスポンス を参照してください。AdminAPI には、削除後にインスタンスがクラスタへの再参加を試行する回数を構成する autoRejoinTries オプションが用意されています。 デフォルトでは、インスタンスはクラスタに自動的に再参加しません。 次のコマンドを使用して、クラスタレベルまたは個々のインスタンスのいずれかで autoRejoinTries オプションを構成できます:

- dba.createCluster()
- Cluster.addInstance()
- Cluster.setOption()

- Cluster.setInstanceOption()

autoRejoinTries オプションは、0 から 2016 までの正の整数値を受け入れ、デフォルト値は 0 です。つまり、インスタンスは自動的に再結合を試みません。 自動再結合機能を使用している場合、クラスタは障害、特に信頼できないネットワークなどの一時的な障害に対してより許容範囲が高くなります。 ただし、クォーラムが失われた場合は、インスタンスを再結合するために大部分が必要になるため、メンバーがクラスタに自動的に再参加しないようにする必要があります。

MySQL バージョン 8.0.12 以降を実行しているインスタンスには、AdminAPI exitStateAction オプションを使用して構成できる group_replication_exit_state_action 変数があります。 これは、クラスタを予期せず終了した場合のインスタンスの動作を制御します。 デフォルトでは、exitStateAction オプションは READ_ONLY,です。つまり、クラスタを予期せず終了したインスタンスは読取り専用になります。 exitStateAction が OFFLINE_MODE (MySQL 8.0.18 から使用可能) に設定されている場合、クラスタを予期せずに残すインスタンスは読取り専用になり、オフラインモードになります。このモードでは、既存のクライアントが切断され、新しい接続は受け入れられません (管理者権限を持つクライアントを除く)。 exitStateAction が ABORT_SERVER に設定されている場合、クラスタを予期せず終了すると、インスタンスは MySQL を停止し、クラスタに再参加する前に再起動する必要があります。 自動再結合機能を使用している場合、exitStateAction オプションで構成されたアクションは、すべての再結合試行がクラスタに失敗した場合にのみ発生することに注意してください。

インスタンスに接続し、AdminAPI を使用して構成しようとする可能性がありますが、その時点でインスタンスがクラスタに再参加している可能性があります。 これは、次のいずれかの操作を使用するたびに発生する可能性があります:

- Cluster.status()
- dba.getCluster()
- Cluster.rejoinInstance()
- Cluster.addInstance()
- Cluster.removeInstance()
- Cluster.rescan()
- Cluster.checkInstanceState()

これらの操作では、インスタンスがクラスタに自動的に再参加している間に、追加情報が提供される場合があります。 また、Cluster.removeInstance() を使用している場合、ターゲットインスタンスがクラスタに自動的に再参加すると、force:true を渡さないかぎり操作は中断されます。

パラレルレプリケーションアプリケーションの構成

バージョンから、8.0.23 インスタンスはパラレルレプリケーションアプリケーションスレッド (マルチスレッドレプリカと呼ばれることもあります) をサポートし、有効にします。 複数のレプリカアプライアンススレッドをパラレルに使用すると、レプリケーションアプライアンスと増分リカバリの両方のスループットが向上します。

つまり、8.0.23 以降を実行しているインスタンスでは、次のシステム変数を構成する必要があります:

- binlog_transaction_dependency_tracking=WRITESET
- slave_preserve_commit_order=ON
- slave_parallel_type=LOGICAL_CLOCK
- transaction_write_set_extraction=XXHASH64

デフォルトでは、適用者スレッドの数 (slave_parallel_workers システム変数で構成) は 4 に設定されています。

8.0.23 より前のバージョンの MySQL サーバーおよび MySQL Shell を実行しているクラスタをアップグレードする場合、インスタンスはパラレルレプリケーションアプライヤを使用するように構成されません。 パラレルアプライヤが有効になっていない場合、Cluster.status() 操作の出力では、次のようなメッセージが instanceErrors フィールドに表示されます:

```
...
```

```
"instanceErrors": [
 "NOTE: The required parallel-appliers settings are not enabled on
  the instance. Use dba.configureInstance() to fix it."
...
```

この場合、パラレルレプリケーションアプライヤを使用するようにインスタンスを再構成する必要があります。 InnoDB クラスタ に属するインスタンスごとに、dba.configureInstance(instance) を発行して構成を更新します。 通常、dba.configureInstance() はインスタンスをクラスタに追加する前に使用されますが、この特別なケースでは、インスタンスを削除する必要はなく、構成の変更はオンライン中に行われます。

パラレルレプリケーションアプライヤに関する情報は、Cluster.status(extended=1) 操作の出力に表示されます。 たとえば、パラレルレプリケーションアプライヤが有効な場合、インスタンスの topology セクションの出力には、applierWorkerThreads の下のスレッド数が表示されます。 パラレルレプリケーションアプライヤ用に構成されたシステム変数は、Cluster.options() 操作の出力に表示されます。

インスタンスがパラレルレプリケーションアプライヤに使用するスレッドの数は、applierWorkerThreads オプションを使用して構成できます (デフォルトは 4 スレッドです)。 このオプションは、0 から 1024 の範囲の整数を受け入れ、dba.configureInstance() および dba.configureReplicaSetInstance() 操作でのみ使用できます。 たとえば、8 つのスレッドを使用するには、次のコマンドを発行します:

```
mysql-js> dba.configureInstance(instance, {applierWorkerThreads: 8, restart: true})
```

> **注記**
>
> パラレルレプリケーションアプライヤで使用されるスレッド数の変更は、インスタンスが再起動されてクラスタに再結合された後にのみ発生します。

パラレルレプリケーションアプライヤを無効にするには、applierWorkerThreads オプションを 0 に設定します。

クラスタの保護

セキュアな接続を使用するようにサーバーインスタンスを構成できます。 MySQL でのセキュアな接続の使用に関する一般情報は、暗号化された接続の使用 を参照してください。 このセクションでは、暗号化された接続を使用するようにクラスタを構成する方法について説明します。 追加のセキュリティ可能性は、クラスタにアクセスできるサーバーを構成することです。サーバーの許可リストの作成 を参照してください。

> **重要**
>
> 暗号化された接続を使用するようにクラスタを構成したら、サーバーを ipAllowlist に追加する必要があります。

dba.createCluster() を使用してクラスタを設定する場合、サーバーインスタンスが暗号化を提供すると、シードインスタンスで自動的に有効になります。 memberSslMode オプションを dba.createCluster() メソッドに渡して、別の SSL モードを指定します。 クラスタの SSL モードは、作成時にのみ設定できます。 memberSslMode オプションは、使用する SSL モードを構成する文字列で、デフォルトは AUTO です。 許可される値は、DISABLED、REQUIRED および AUTO です。 これらのモードは次のように定義されます:

- createCluster({memberSslMode:'DISABLED'}) を設定すると、クラスタ内のシードインスタンスで SSL 暗号化が無効になります。
- createCluster({memberSslMode:'REQUIRED'}) を設定すると、クラスタ内のシードインスタンスに対して SSL 暗号化が有効になります。 有効にできない場合は、エラーが発生します。
- createCluster({memberSslMode:'AUTO'}) (デフォルト) を設定すると、SSL 暗号化は、サーバーインスタンスでサポートされている場合は自動的に有効になり、サーバーでサポートされていない場合は無効になります。

> **注記**
>
> 商用バージョンの MySQL を使用する場合、SSL はデフォルトで有効になっており、すべてのインスタンスに対して許可リストを構成する必要がある場合があります。 サーバーの許可リストの作成を参照してください。

cluster.addInstance() および cluster.rejoinInstance() コマンドを発行すると、シードインスタンスに検出された設定に基づいて、インスタンスの SSL 暗号化が有効または無効になります。

adoptFromGR オプションを指定して createCluster() を使用して既存のグループレプリケーショングループを採用する場合、採用されたクラスタで SSL 設定は変更されません:

- memberSslMode は、adoptFromGR では使用できません。
- 採用されたクラスタの SSL 設定が MySQL Shell でサポートされている設定と異なる場合 (つまり、グループレプリケーションリカバリおよびグループ通信の SSL)、両方の設定は変更されません。 つまり、採用されたクラスタの設定を手動で変更しないかぎり、新しいインスタンスをクラスタに追加できません。

MySQL Shell は、グループレプリケーションリカバリとグループ通信の両方でクラスタの SSL を常に有効または無効にします。Secure Socket Layer (SSL) を使用したグループ通信接続の保護 を参照してください。 検証が実行され、新しいインスタンスをクラスタに追加するときにシードインスタンスの設定が異なる場合 (たとえば、adoptFromGR を使用した dba.createCluster() の結果として)、エラーが発行されます。 クラスタ内のすべてのインスタンスで SSL 暗号化を有効または無効にする必要があります。 新しいインスタンスをクラスタに追加するときに、この不変条件が確実に保持されるように検証が実行されます。

dba.deploySandboxInstance() コマンドは、デフォルトで SSL 暗号化サポートを使用してサンドボックスインスタンスのデプロイを試行します。 不可能な場合、サーバーインスタンスは SSL サポートなしでデプロイされます。 ignoreSslError オプションを false に設定すると、サンドボックスインスタンスが SSL サポートとともにデプロイされ、SSL サポートを提供できない場合はエラーが発行されます。 ignoreSslError が true(デフォルト) の場合、SSL サポートを提供できず、サーバーインスタンスが SSL サポートなしでデプロイされていると、操作中にエラーは発行されません。

サーバーの許可リストの作成

クラスタの createCluster()、addInstance() および rejoinInstance() メソッドを使用する場合、オプションで、許可リストと呼ばれる、クラスタに属する承認済サーバーのリストを指定できます。 許可リストをこの方法で明示的に指定することで、許可リスト内のサーバーのみがクラスタに接続できるため、クラスタのセキュリティを向上させることができます。 ipAllowlist オプション (以前の ipWhitelist、現在は非推奨) を使用して、インスタンスに group_replication_ip_allowlist システム変数を構成します。 デフォルトでは、明示的に指定しない場合、allowlist は、サーバーがネットワークインタフェースを持つプライベートネットワークアドレスに自動的に設定されます。 allowlist を構成するには、メソッドの使用時に ipAllowlist オプションを使用して追加するサーバーを指定します。 IP アドレスは IPv4 形式で指定する必要があります。 サーバーを引用符で囲んだカンマ区切りリストとして渡します。 例:

```
mysql-js> cluster.addInstance("icadmin@ic-3:3306", {ipAllowlist: "203.0.113.0/24, 198.51.100.110"})
```

これにより、アドレス 203.0.113.0/24 および 198.51.100.110 のサーバーからの接続のみを受け入れるようにインスタンスが構成されます。 allowlist には、別のサーバーによって接続リクエストが行われた場合にのみ解決されるホスト名を含めることもできます。

> **警告**
>
> ホスト名は本質的に許可リストの IP アドレスより安全性が低くなります。 MySQL は、適切なレベルの保護を提供する FCrDNS 検証を実行しますが、特定のタイプの攻撃によって危険にさらされる可能性があります。 厳密に必要な場合にのみ許可リストにホスト名を指定し、名前解決に使用されるすべてのコンポーネント (DNS サーバーなど) が制御下に保持されていることを確認します。 外部コンポーネントを使用しないように、hosts ファイルを使用して名前解決をローカルに実装することもできます。

6.2.7 InnoDB クラスタ のトラブルシューティング

このセクションでは、InnoDB クラスタ のトラブルシューティング方法について説明します。

- クラスタへのインスタンスの再参加
- クォーラム損失からのクラスタのリストア
- メジャーな停止からのクラスタの再起動
- クラスタの再スキャン

クラスタへのインスタンスの再参加

たとえば、接続が失われ、なんらかの理由で自動的にクラスタに再参加できなかったために、インスタンスがクラスタを離れた場合は、後でクラスタに再参加する必要がある場合があります。 インスタンスをクラスタに再結合するには、Cluster.rejoinInstance(instance) を発行します。

> ヒント
>
> インスタンスに super_read_only=ON がある場合は、AdminAPI で super_read_only=OFF を設定できることを確認する必要がある場合があります。 詳しくはスーパー読取り専用およびインスタンスをご覧ください。

インスタンスの構成が永続化されていない場合 (設定の永続化 を参照)、インスタンスの再起動時にクラスタに自動的に再参加しません。 解決策は、インスタンスがクラスタに再度追加され、変更が永続化されるように、cluster.rejoinInstance() を発行することです。 InnoDB クラスタ 構成がインスタンスオプションファイルに永続化されると、クラスタに自動的に再結合されます。

なんらかの方法で変更されたインスタンスを再結合する場合は、再結合プロセスが正しく機能するようにインスタンスを変更する必要がある場合があります。 たとえば、MySQL Enterprise Backup バックアップをリストアすると、server_uuid が変更されます。 このようなインスタンスを再結合しようとすると、InnoDB クラスタ インスタンスが server_uuid 変数によって識別されるため、失敗します。 このような状況では、古いインスタンスの server_uuid に関する情報を InnoDB クラスタ メタデータから削除してから、Cluster.rescan() を実行し、新しい server_uuid を使用してインスタンスをメタデータに追加する必要があります。 例:

```
cluster.removeInstance("root@instanceWithOldUUID:3306", {force: true})

cluster.rescan()
```

この場合、インスタンスはクラスタの観点からアクセスできず、InnoDB クラスタ メタデータから削除する必要があるため、force オプションを Cluster.removeInstance() メソッドに渡す必要があります。

クォーラム損失からのクラスタのリストア

インスタンスに障害が発生すると、クラスタのクォーラムが失われる可能性があります。これは、新しいプライマリで投票する機能です。 これは、グループレプリケーション操作で投票するクラスタを構成するインスタンスの大部分がなくなった場合に発生する可能性があります。 Fault-toleranceを参照してください。 クラスタがクォーラムを失うと、インスタンスの追加、再参加、削除などによって、クラスタとの書込みトランザクションを処理したり、クラスタトポロジを変更することはできなくなります。 ただし、InnoDB クラスタ メタデータを含むオンラインのインスタンスがある場合は、クォーラムを使用してクラスタをリストアできます。 これは、InnoDB クラスタ メタデータを含むインスタンスに接続でき、そのインスタンスがクラスタのリストアに使用する他のインスタンスに接続できることを前提としています。

> 重要
>
> この操作は、不適切に使用された場合にスプリットブレインシナリオが作成される可能性があり、最後の手段とみなす必要があるため、危険になる可能性があります。 ネットワーク内のどこかで動作しているが、自分の場所からアクセスできないこのグループのパーティションがないことを絶対に確認してください。

クラスタメタデータを含むインスタンスに接続し、Cluster.forceQuorumUsingPartitionOf(instance) 操作を使用して instance のメタデータに基づいてクラスタをリストアし、指定されたインスタンス定義の観点から ONLINE であるすべてのインスタンスがリストアされたクラスタに追加されます。

```
mysql-js> cluster.forceQuorumUsingPartitionOf("icadmin@ic-1:3306")

 Restoring replicaset 'default' from loss of quorum, by using the partition composed of [icadmin@ic-1:3306]

 Please provide the password for 'icadmin@ic-1:3306': ******
 Restoring the InnoDB cluster ...

 The InnoDB cluster was successfully restored using the partition from the instance 'icadmin@ic-1:3306'.

 WARNING: To avoid a split-brain scenario, ensure that all other members of the replicaset
 are removed or joined back to the group that was restored.
```

インスタンスがクラスタに自動的に追加されない場合 (設定が永続化されていない場合など) は、Cluster.rejoinInstance() を使用してインスタンスを手動でクラスタに追加しなおします。

リストアされたクラスタは、クラスタを構成していたすべての元のインスタンスで構成されている場合があり、必ずしも構成されている必要はありません。 たとえば、元のクラスタが次の 5 つのインスタンスで構成されているとします:

- ic-1
- ic-2
- ic-3
- ic-4
- ic-5

クラスタでは、ic-1、ic-2 および ic-3 があるパーティションを形成し、ic-4 および ic-5 が別のパーティションを形成するスプリットブレインシナリオが発生します。 ic-1 に接続し、Cluster.forceQuorumUsingPartitionOf('icadmin@ic-1:3306') を発行してクラスタをリストアする場合、結果のクラスタは次の 3 つのインスタンスで構成されます:

- ic-1
- ic-2
- ic-3

これは、ic-1 では、ic-2 および ic-3 が ONLINE として認識され、ic-4 および ic-5 が表示されないためです。

メジャーな停止からのクラスタの再起動

クラスタで完全な停止が発生した場合、dba.rebootClusterFromCompleteOutage() を使用してクラスタが正しく再構成されていることを確認できます。 この操作では、MySQL Shell が現在接続しているインスタンスを取得し、そのメタデータを使用してクラスタをリカバリします。 クラスタインスタンスが完全に停止している場合は、インスタンスを起動してからクラスタを起動できるようにする必要があります。 たとえば、サンドボックスクラスタが実行されていたマシンが再起動され、インスタンスがポート 3310、3320 および 3330 にあった場合は、次を発行します:

```
mysql-js> dba.startSandboxInstance(3310)
mysql-js> dba.startSandboxInstance(3320)
mysql-js> dba.startSandboxInstance(3330)
```

これにより、サンドボックスインスタンスが確実に実行されます。 本番デプロイメントの場合は、MySQL Shell の外部でインスタンスを起動する必要があります。 インスタンスが起動したら、GTID スーパーセット (停止前に最も多くのトランザクションを適用したインスタンス) を使用してインスタンスに接続する必要があります。 GTID スーパーセットを含むインスタンスが不明な場合は、任意のインスタンスに接続し、接続しているインスタンスに GTID スーパーセットが含まれているかどうかを検出する dba.rebootClusterFromCompleteOutage() 操作からの対話型メッセージに従います。 次を発行して、クラスタを再起動します:

```
mysql-js> var cluster = dba.rebootClusterFromCompleteOutage();
```

その後、dba.rebootClusterFromCompleteOutage() 操作は次のステップに従って、クラスタが正しく再構成されていることを確認します:

- MySQL Shell が現在接続されているインスタンスで見つかった InnoDB クラスタ メタデータがチェックされ、GTID スーパーセット (つまり、クラスタによって適用されたトランザクション) が含まれているかどうかが確認されます。 現在接続されているインスタンスに GTID スーパーセットが含まれていない場合、操作はその情報で中止されます。 詳細は、後続の段落を参照してください。
- インスタンスに GTID スーパーセットが含まれている場合、クラスタはインスタンスのメタデータに基づいてリカバリされます。
- MySQL Shell を対話型モードで実行している場合、ウィザードが実行され、現在アクセス可能なクラスタのインスタンスがチェックされ、検出されたインスタンスを再起動されたクラスタに再結合するかどうかが尋ねられます。
- 同様に、対話型モードでも、ウィザードは現在到達できないインスタンスを検出し、再起動されたクラスタからそのようなインスタンスを削除するかどうかを尋ねます。

MySQL Shell 対話型モードを使用していない場合は、rejoinInstances および removeInstances オプションを使用して、クラスタの再起動時に結合または削除する必要があるインスタンスを手動で構成できます。

「アクティブセッションインスタンスは、クラスタメタデータの ONLINE インスタンスと比較して最も更新されません。」などのエラーが発生した場合、接続しているインスタンスには、クラスタによって適用される GTID スーパーセットのトランザクションがありません。 この状況では、MySQL Shell をエラーメッセージに示されたインスタンスに接続し、そのインスタンスから dba.rebootClusterFromCompleteOutage() を発行します。

> ヒント
>
> 対話型ウィザードを使用するのではなく GTID スーパーセットを持つインスタンスを手動で検出するには、各インスタンスの gtid_executed 変数を確認します。 たとえば、次のコマンドを発行します:
>
> ```
> mysql-sql> SHOW VARIABLES LIKE 'gtid_executed';
> ```
>
> トランザクションの最大「GTID セット」を適用したインスタンスに GTID スーパーセットが含まれています。

このプロセスが失敗し、クラスタメタデータが不正に破損した場合は、メタデータを削除し、クラスタを最初から再度作成する必要がある場合があります。 dba.dropMetadataSchema() を使用してクラスタメタデータを削除できます。

> 警告
>
> dba.dropMetadataSchema() メソッドは、クラスタをリストアできない場合に、最後の手段としてのみ使用してください。 元に戻すことはできません。

クラスタで MySQL Router を使用している場合、メタデータを削除すると、現在のすべての接続が切断され、新しい接続は禁止されます。 これにより、完全な停止が発生します。

クラスタの再スキャン

構成の問題を解決するためにインスタンス構成を手動で変更するなどして、AdminAPI コマンドの外部でクラスタに構成変更を加えた場合、またはインスタンスの損失後に、InnoDB クラスタ メタデータがインスタンスの現在の構成と一致するように更新する必要があります。 このような場合は、Cluster.rescan() 操作を使用します。これにより、InnoDB クラスタ メタデータを手動で更新するか、対話型ウィザードを使用して更新できます。 Cluster.rescan() 操作では、メタデータに登録されていない新しいアクティブインスタンスを検出して追加したり、メタデータにまだ登録されている (アクティブでなくなった) 古いインスタンスを検出して削除できます。 コマンドで検出されたインスタンスに応じてメタデータを自動的に更新することも、メタデータに追加またはメタデータから削除するインスタンスアドレスのリストを指定することもできます。 メタデータに格納されているトポロジモードを更新することもできます。たとえば、AdminAPI の外部で単一プライマリモードからマルチプライマリモードに変更した後などです。

コマンドの構文は Cluster.rescan([options]) です。 options ディクショナリでは、次のものがサポートされています:

- interactive: コマンド実行でウィザードを無効または有効にするために使用されるブール値。 プロンプトと確認を提供するかどうかを制御します。 デフォルト値は、shell.options.useWizards で指定された MySQL Shell ウィザードモードと同じです。
- addInstances: メタデータに追加する新しいアクティブインスタンスの接続データを含むリスト、または欠落しているインスタンスをメタデータに自動的に追加する 「auto」。 値 「auto」 では、大/小文字は区別されません。
  - リストで指定されたインスタンスは、確認を求められることなくメタデータに追加されます
  - 対話モードでは、addInstances オプションに含まれていない新しく検出されたインスタンスの追加を確認するプロンプトが表示されます
  - 非対話型モードでは、addInstances オプションに含まれていない新しく検出されたインスタンスが出力にレポートされますが、追加を求めるプロンプトは表示されません
- removeInstances: メタデータから削除する廃止されたインスタンスの接続データを含むリスト、またはメタデータから不要なインスタンスを自動的に削除する 「auto」。
  - リストで指定されたインスタンスは、確認を求められることなくメタデータから削除されます

- 対話型モードでは、removeInstances オプションに含まれていない不要なインスタンスの削除を確認するプロンプトが表示されます
- 非対話型モードでは、removeInstances オプションに含まれていない廃止されたインスタンスは出力にレポートされますが、削除を求めるプロンプトは表示されません

- updateTopologyMode: メタデータのトポロジモード (単一プライマリまたはマルチプライマリ) を、クラスタで使用されているトポロジモードと一致するように更新する (true) か更新しない (false) かを示すために使用されるブール値。 デフォルトでは、メタデータは更新されません (false)。
  - 値が true の場合、InnoDB クラスタ メタデータはグループレプリケーションで使用されている現在のモードと比較され、必要に応じてメタデータが更新されます。 このオプションを使用して、AdminAPI 外部のクラスタのトポロジモードを変更した後にメタデータを更新します。
  - 値が false の場合、クラスタトポロジモードに関する InnoDB クラスタ メタデータは、クラスタグループレプリケーショングループで使用されるトポロジと異なる場合でも更新されません
  - このオプションが指定されておらず、メタデータのトポロジモードがクラスタグループレプリケーショングループで使用されるトポロジと異なる場合は、次のようになります:
    - 対話モードでは、メタデータのトポロジモードの更新を確認するプロンプトが表示されます
    - 非対話型モードでは、クラスタグループレプリケーショングループで使用されるトポロジと InnoDB クラスタ メタデータに違いがある場合、それがレポートされ、メタデータは変更されません
  - メタデータトポロジモードがグループレプリケーションモードと一致するように更新されると、InnoDB クラスタおよび自動増分 で説明されているように、すべてのインスタンスの自動増分設定が更新されます。

6.2.8 InnoDB クラスタ のアップグレード

このセクションでは、クラスタをアップグレードする方法について説明します。 InnoDB クラスタ のアップグレードプロセスの多くは、グループレプリケーションのアップグレード に記載されているのと同じ方法でインスタンスをアップグレードすることで構成されます。 このセクションでは、InnoDB クラスタ のアップグレードに関するその他の考慮事項に焦点を当てます。 アップグレードを開始する前に、MySQL Shell セクション8.1「アップグレードチェッカユーティリティ」 を使用して、インスタンスのアップグレード準備ができていることを確認できます。

バージョン 8.0.19 から、クラスタに対して MySQL Router をブートストラップしようとしたときに、メタデータバージョンが 0.0.0 であることが検出された場合、これはメタデータのアップグレードが進行中であることを示し、ブートストラップは失敗します。 メタデータのアップグレードが完了するまで待ってから、ブートストラップを再試行してください。 MySQL Router が正常に (ブートストラップではなく) 動作している場合、メタデータバージョンが 0.0.0 (アップグレード進行中) であることが検出されても、開始しようとしていたメタデータのリフレッシュは続行されません。 かわりに、MySQL Router はキャッシュされた最後のメタデータを引き続き使用します。 既存のすべてのユーザー接続が維持され、新しい接続はキャッシュされたメタデータに従ってルーティングされます。 メタデータバージョンが 0.0.0 でなくなると、メタデータのリフレッシュが再開されます。 通常の (ブートストラップではない) モードでは、MySQL Router はバージョン 1.x.x と 2.x.x。メタデータ、およびバージョンは TTL リフレッシュ間でバージョンを変更できます。 これにより、クラスタのアップグレード中にルーティングが続行されます。

バージョン 5.7 や 8.0 など、複数の MySQL バージョンを実行するクラスタ内にインスタンスを含めることはできますが、このような混在は長時間使用する場合にはお薦めしません。 たとえば、バージョンが混在するクラスタでは、バージョン 5.7 を実行しているインスタンスがクラスタから離れ、リカバリ操作に MySQL クローンが使用される場合、BACKUP_ADMIN 権限が要件になるため、下位バージョンを実行しているインスタンスはクラスタに参加できなくなります。 複数のバージョンでクラスタを実行することは、あるバージョンから別のバージョンへの移行を支援する一時的な状況として意図されているため、長期的な使用には依存しないでください。

6.2.8.1 ローリングアップグレード

8.0.19 より前の MySQL Shell バージョンによってデプロイされたクラスタのメタデータスキーマをアップグレードする場合、既存の MySQL Router インスタンスのローリングアップグレードが必要です。 このプロセスにより、アップグレード中のアプリケーションの中断が最小限に抑えられます。 ローリングアップグレード処理は、次の順序で実行する必要があります:

1. 最新の MySQL Shell バージョンを実行し、グローバルセッションをクラスタに接続して、dba.upgradeMetadata() を発行します。 このステップでは、クラスタ用に構成された MySQL Router アカウントの権限が、新しいバージョンと互換性を持つように変更されていることを確認します。 古い MySQL Router インスタンスが検出されると、アップグレード機能は停止します。この時点で、MySQL Shell でアップグレードプロセスを停止して後で再開できます。

2. 検出された最新でない MySQL Router インスタンスを最新バージョンにアップグレードします。 MySQL Shell バージョンと同じ MySQL Router バージョンを使用することをお薦めします。

3. dba.upgradeMetadata() 操作を続行または再起動して、メタデータのアップグレードを完了します。

6.2.8.2 InnoDB クラスタ メタデータのアップグレード

AdminAPI の進化に伴い、一部のリリースでは、既存のクラスタのメタデータをアップグレードして、新しいバージョンの MySQL Shell との互換性を確保する必要がある場合があります。 たとえば、バージョン 8.0.19 に InnoDB ReplicaSet を追加することは、メタデータスキーマがバージョン 2.0 にアップグレードされていることを意味します。 InnoDB ReplicaSet を使用するかどうかに関係なく、以前のバージョンの MySQL Shell を使用してデプロイされたクラスタで MySQL Shell 8.0.19 以降を使用するには、クラスタのメタデータをアップグレードする必要があります。

> **警告**
>
> メタデータをアップグレードしないと、MySQL Shell 8.0.19 を使用して、以前のバージョンで作成されたクラスタの構成を変更できません。 たとえば、読取り操作は、Cluster.status()、Cluster.describe()、Cluster.options() などのクラスタに対してのみ実行できます。

この dba.upgradeMetadata() 操作では、クラスタ MySQL Shell で現在接続されているメタデータスキーマのバージョンが、この MySQL Shell バージョンでサポートされているメタデータスキーマのバージョンと比較されます。 インストールされているメタデータのバージョンが低い場合は、アップグレードプロセスが開始されます。 その後、dba.upgradeMetadata() 操作によって、自動的に作成された MySQL Router ユーザーが適切な権限を持つようにアップグレードされます。 mysql_router_で始まらない名前で手動で作成された MySQL Router ユーザーは、自動的にはアップグレードされません。 これは、クラスタをアップグレードする際の重要なステップであり、MySQL Router アプリケーションのみをアップグレードできます。 クラスタに登録されている MySQL Router インスタンスのうち、メタデータのアップグレードが必要なものに関する情報を取得するには、次のコマンドを発行します:

```
cluster.listRouters({'onlyUpgradeRequired':'true'})
{
    "clusterName": "mycluster",
    "routers": {
        "example.com::": {
            "hostname": "example.com",
            "lastCheckIn": "2019-11-26 10:10:37",
            "roPort": 6447,
            "roXPort": 64470,
            "rwPort": 6446,
            "rwXPort": 64460,
            "version": "8.0.18"
        }
    }
}
```

> **警告**
>
> 新しいメタデータを使用しているクラスタは、以前の MySQL Shell バージョンでは管理できません。たとえば、バージョン 8.0.19 にアップグレードすると、バージョン 8.0.18 以前を使用してクラスタを管理できなくなります。

クラスタメタデータをアップグレードするには、MySQL Shell グローバルセッションをクラスタに接続し、dba.upgradeMetadata() 操作を使用してクラスタメタデータを新しいメタデータにアップグレードします。 例:

```
mysql-js> \connect user@example.com:3306

mysql-js> dba.upgradeMetadata()
InnoDB Cluster Metadata Upgrade
```

```
The cluster you are connected to is using an outdated metadata schema version
1.0.1 and needs to be upgraded to 2.0.0.

Without doing this upgrade, no AdminAPI calls except read only operations will
be allowed.

The grants for the MySQL Router accounts that were created automatically when
bootstrapping need to be updated to match the new metadata version's
requirements.
Updating router accounts...
NOTE: 2 router accounts have been updated.

Upgrading metadata at 'example.com:3306' from version 1.0.1 to version 2.0.0.
Creating backup of the metadata schema...
Step 1 of 1: upgrading from 1.0.1 to 2.0.0...
Removing metadata backup...
Upgrade process successfully finished, metadata schema is now on version 2.0.0
```

権限のないクラスタ管理ユーザーに関連するエラーが発生した場合は、更新オプションを指定して Cluster.setupAdminAccount() 操作を使用し、ユーザーに正しい権限を付与します。 AdminAPI のユーザーの構成を参照してください。

6.2.8.3 InnoDB クラスタ のアップグレードのトラブルシューティング

このセクションでは、アップグレードプロセスのトラブルシューティングについて説明します。

ホスト名の変更の処理

MySQL Shell は、指定された接続パラメータのホスト値を、AdminAPI 操作、つまりメタデータへのインスタンスの登録 (dba.createCluster() および Cluster.addInstance() 操作) に使用されるターゲットホスト名として使用します。 ただし、接続パラメータに使用される実際のホストは、Group Replication によって使用または報告される hostname と一致しない可能性があります。この場合は、定義時に report_host システム変数の値が使用されます (つまり、NULL ではありません)。それ以外の場合は、hostname の値が使用されます。 したがって、AdminAPI は、インスタンス接続パラメータのホスト値を使用するかわりに、同じロジックに従ってメタデータにターゲットインスタンスを登録し、インスタンスの group_replication_local_address 変数のデフォルト値として登録するようになりました。 report_host 変数が空に設定されている場合、グループレプリケーションではホストに空の値が使用されますが、AdminAPI では (dba.checkInstanceConfiguration(), dba.configureInstance(), dba.createCluster() などのコマンドで) ホスト名が使用された値としてレポートされ、これは Group Replication によって報告された値と矛盾しています。 report_host システム変数に空の値が設定されている場合、エラーが生成されます。 (Bug #28285389)

8.0.16 より前のバージョンの MySQL Shell を使用して作成されたクラスタの場合、バージョン 8.0.16 以上を使用して実行された完全な停止からクラスタを再起動しようとすると、このエラーが発生します。 これは、メタデータ値がインスタンスによって報告された report_host または hostname 値と一致しないことが原因です。 回避策は次のとおりです:

1. 「seed」 であるインスタンス、つまり最新の GTID セットを持つインスタンスを識別します。 dba.rebootClusterFromCompleteOutage() 操作は、インスタンスがシードであるかどうかを検出し、現行のセッションが最新のインスタンスに接続されていない場合はエラーを生成します。

2. report_host システム変数を、ターゲットインスタンスのメタデータスキーマに格納されている値に設定します。 この値は、クラスタ作成時にインスタンス定義で使用される hostname:port ペアです。 この値は、mysql_innodb_cluster_metadata.instances テーブルをクエリーすることで参照できます。

   たとえば、次の一連のコマンドを使用してクラスタが作成されたとします:

   ```
   mysql-js> \c clusterAdmin@localhost:3306
   mysql-js> dba.createCluster("myCluster")
   ```

   したがって、メタデータに格納されるホスト名の値は 「localhost」 であり、そのため、シードで report_host を 「localhost」 に設定する必要があります。

3. シードインスタンスのみを使用してクラスタを再起動します。 対話型プロンプトでは、残りのインスタンスはクラスタに追加されません。

4. Cluster.rescan() を使用して、他のインスタンスをクラスタに追加します。

5. クラスタからシードインスタンスを削除
6. シードインスタンスで mysqld を停止し、強制 report_host 設定を削除するか (ステップ 2)、メタデータ値に以前格納されていた値で置き換えます。
7. シードインスタンスを再起動し、Cluster.addInstance() を使用してクラスタに追加しなおします

これにより、クラスタを最新の MySQL Shell バージョンにスムーズかつ完全にアップグレードできます。 ユースケースに依存する別の可能性は、クラスタの作成時にメタデータスキーマに登録されているものと一致するように、すべてのクラスタメンバーで report_host の値を設定することです。

6.2.9 メタデータのタグ付け

バージョン 8.0.21 から、構成可能なタグフレームワークを使用して、クラスタまたは ReplicaSet のメタデータを追加情報でマークできるようになりました。 タグを使用すると、カスタムキーと値のペアをクラスタ、ReplicaSet またはインスタンスに関連付けることができます。 タグは MySQL Router での使用のために予約されており、互換性のある MySQL Router でアプリケーションからのインスタンスの非表示をサポートできます。 次のタグは、この目的で予約されています:

- _hidden は、クライアントアプリケーションで使用可能な宛先のリストからインスタンスを除外するように MySQL Router に指示
- _disconnect_existing_sessions_when_hidden は、非表示とマークされたインスタンスから既存の接続を切断するようルーターに指示

詳細は、ルーティングからのインスタンスの削除を参照してください。

また、タグフレームワークはユーザーが構成できます。 カスタムタグは任意の ASCII 文字で構成でき、クラスタ、ReplicaSets またはその特定のインスタンスに関連付けることができるキーと値のペアのディクショナリとして機能する namespace を提供します。 タグ値には任意の JSON 値を指定できます。 これにより、メタデータの上部に独自の属性を追加できます。

タグの表示

Cluster.options() 操作では、個々のクラスタインスタンスおよびクラスタ自体に割り当てられたタグに関する情報が表示されます。 たとえば、myCluster に割り当てられた InnoDB クラスタ では、次のように表示されます:

```
mysql-js> myCluster.options()
{
    "cluster": {
        "name": "test1",
        "tags": {
            "ic-1:3306": [
                {
                    "option": "_disconnect_existing_sessions_when_hidden",
                    "value": true
                },
                {
                    "option": "_hidden",
                    "value": false
                }
            ],
            "ic-2:3306": [],
            "ic-3:3306": [],
            "global": [
                {
                    "option": "location:",
                    "value": "US East"
                }
            ]
        }
    }
}
```

このクラスタには、値 US East を持つ location という名前のグローバルタグがあり、インスタンス ic-1 にタグが付けられています。

クラスタインスタンスでのタグの設定

インスタンスレベルでタグを設定できます。これにより、たとえば、インスタンスを使用不可としてマークして、アプリケーションおよびルーターでオフラインとして処理できます。 Cluster.setInstanceOption(instance, option, value) 操作を使用して、インスタンスのタグの値を設定します。 instance 引数は、ターゲットインスタンスへの接続文字列です。 option 引数は、namespace:option 形式の文字列である必要があります。 value パラメータは、指定した namespace の option に割り当てる必要がある値です。 値が null の場合、option は指定された namespace から削除されます。 クラスタに属するインスタンスの場合、setInstanceOption() 操作は tag ネームスペースのみを受け入れます。 その他のネームスペースは ArgumentError になります。

たとえば、myCluster インスタンス ic-1 でタグ test を true に設定するには、次のように発行します:

```
mysql-js> myCluster.setInstanceOption(icadmin@ic-1:3306, "tag:test", true);
```

ルーティングからのインスタンスの削除

AdminAPI と MySQL Router が連携して動作している場合、インスタンスを非表示としてマークし、ルーティングから削除できる特定のタグがサポートされます。次に、MySQL Router は、このようなタグ付けされたインスタンスをルーティング先候補リストから除外します。 これにより、サーバーインスタンスを安全にオフラインにできるため、たとえばサーバーのアップグレードや構成の変更などのメンテナンスタスクの実行中に、アプリケーションおよび MySQL Router でサーバーインスタンスを無視できます。

_hidden タグが true に設定されている場合、クライアントアプリケーションで使用可能な宛先のリストからインスタンスを除外するように MySQL Router に指示します。 インスタンスはオンラインのままですが、新しい着信接続のためにルーティングされません。 _disconnect_existing_sessions_when_hidden タグは、インスタンスへの既存の接続を閉じる方法を制御します。 このタグは true とみなされ、_hidden タグが true の場合、InnoDB クラスタ または InnoDB ReplicaSet に対してブートストラップされた MySQL Routers に対して、インスタンスから既存の接続を切断するように指示します。 _disconnect_existing_sessions_when_hidden が false の場合、_hidden が true であれば、インスタンスへの既存のクライアント接続はクローズされません。 予約済の_hidden および_disconnect_existing_sessions_when_hidden タグはインスタンスに固有であり、クラスタレベルでは使用できません。

> **警告**
>
> use_gr_notifications MySQL Router オプションが有効な場合、デフォルトで 60 秒に設定されます。 つまり、タグを設定すると、MySQL Router が変更を検出するまで最大 60 秒かかります。 待機時間を短縮するには、use_gr_notifications を小さい値に変更します。

たとえば、myCluster に割り当てられた InnoDB クラスタ の一部である ic-1 インスタンスをルーティング先から削除するとします。 setInstanceOption() 操作を使用して、_hidden および_disconnect_existing_sessions_when_hidden タグを有効にします:

```
mysql-js> myCluster.setInstanceOption(icadmin@ic-1:3306, "tag:_hidden", true);
```

オプションをチェックすることで、メタデータの変更を確認できます。 たとえば、ic-1 に加えられた変更は、次のようにオプションに表示されます:

```
mysql-js> myCluster.options()
{
    "cluster": {
        "name": "test1",
        "tags": {
            "ic-1:3306": [
                {
                    "option": "_disconnect_existing_sessions_when_hidden",
                    "value": true
                },
                {
                    "option": "_hidden",
                    "value": true
                }
            ],
            "ic-2:3306": [],
            "ic-3:3306": [],
            "global": []
```

```
    }
  }
}
```

MySQL Router がメタデータの変更を検出したことを確認するには、ログファイルを表示します。 ic-1 に対する変更を検出した MySQL Router には、次のような変更が表示されます:

```
2020-07-03 16:32:16 metadata_cache INFO [7fa9d164c700] Potential changes detected in cluster 'testCluster' after metadata refresh
2020-07-03 16:32:16 metadata_cache INFO [7fa9d164c700] view_id = 4, (3 members)
2020-07-03 16:32:16 metadata_cache INFO [7fa9d164c700]     ic-1:3306 / 33060 - mode=RW
2020-07-03 16:32:16 metadata_cache INFO [7fa9d164c700]     ic-1:3306 / 33060 - mode=RO
2020-07-03 16:32:16 metadata_cache INFO [7fa9d164c700]     ic-1:3306 / 33060 - mode=RO hidden=yes disconnect_when_hidden=yes
2020-07-03 16:32:16 routing INFO [7fa9d164c700] Routing routing:testCluster_x_ro listening on 64470 got request to disconnect invalid connections: metadata change
2020-07-03 16:32:16 routing INFO [7fa9d164c700] Routing routing:testCluster_x_rw listening on 64460 got request to disconnect invalid connections: metadata change
2020-07-03 16:32:16 routing INFO [7fa9d164c700] Routing routing:testCluster_rw listening on 6446 got request to disconnect invalid connections: metadata change
2020-07-03 16:32:16 routing INFO [7fa9d164c700] Routing routing:testCluster_ro listening on 6447 got request to disconnect invalid connections: metadata change
```

インスタンスをオンラインに戻すには、setInstanceOption() 操作を使用してタグを削除し、MySQL Router によってインスタンスがルーティング先に自動的に追加され、アプリケーションに対してオンラインになります。 例:

```
mysql-js> myCluster.setInstanceOption(icadmin@ic-1:3306, "tag:_hidden", false);
```

オプションを再度チェックして、メタデータの変更を確認します:

```
mysql-js> myCluster.options()
{
    "cluster": {
        "name": "test1",
        "tags": {
            "ic-1:3306": [
                {
                    "option": "_disconnect_existing_sessions_when_hidden",
                    "value": true
                },
                {
                    "option": "_hidden",
                    "value": false
                }
            ],
            "ic-2:3306": [],
            "ic-3:3306": [],
            "global": []
        }
    }
}
```

クラスタでのタグの設定

Cluster.setOption(option, value) 操作では、クラスタ全体のネームスペースオプションの値を変更できます。 option 引数は、namespace:option 形式の文字列である必要があります。 value パラメータは、指定した namespace の option に割り当てられる値です。 値が null の場合、option は指定された namespace から削除されます。 クラスタの場合、setOption() 操作は tag ネームスペースを受け入れます。 その他のネームスペースは ArgumentError になります。

> **ヒント**
>
> クラスタレベルで設定されたタグは、インスタンスレベルで設定されたタグをオーバーライドしません。 Cluster.setOption() を使用して、インスタンスレベルで設定されたすべてのタグを削除することはできません。

すべてのインスタンスをオンラインにする必要はなく、クラスタにクォーラムがあることのみが必要です。 myCluster に割り当てられた InnoDB クラスタ に location タグを US East に設定してタグ付けするには、次のコマンドを発行します:

```
mysql-js> myCluster.setOption("tag:location", "US East")
mysql-js> myCluster.options()
{
    "cluster": {
        "name": "test1",
        "tags": {
            "ic-1:3306": [],
```

```
        "ic-2:3306": [],
        "ic-3:3306": [],
        "global": [
          {
            "option": "location:",
            "value": "US East"
          }
        ]
      }
    }
}
```

ユーザー定義タグ付け

AdminAPI では、特定のクラスタ、ReplicaSet またはインスタンスに関連付けられたキーと値のペアに情報を格納できる tag ネームスペースがサポートされています。 tag ネームスペースの下のオプションは制約されません。つまり、有効な MySQL ASCII 識別子であるかぎり、選択した情報でタグ付けできます。 名前が次の構文に従っているかぎり、タグには任意の名前と値を使用できます: _または文字の後に英数字と_文字が続きます。

namespace オプションは、namespace:option という形式のコロン区切りの文字列です。ここで、namespace はネームスペースの名前、option は実際のオプション名です。 タグは、インスタンスレベル、クラスタレベルまたは ReplicaSet レベルで設定および削除できます。

タグ名には、文字またはアンダースコアで始まり、オプションで英数字および_文字が続く任意の値 (^[a-zA-Z_][0-9a-zA-Z_]* など) を指定できます。 _のアンダースコア文字で始めることができるのは組込みタグのみです。

カスタムタグの使用方法はユーザーによって異なります。 クラスタにカスタムタグを設定して、そのタグが動作しているリージョンをマークできます。 たとえば、クラスタ上で EMEA の値を持つ location という名前のカスタムタグを設定できます。

6.2.10 InnoDB クラスタ のヒント

このセクションでは、InnoDB クラスタ の使用時に知っておくとよい情報について説明します。

- スーパー読取り専用およびインスタンス
- AdminAPI のユーザーの構成
- InnoDB クラスタ および自動増分
- InnoDB クラスタおよびバイナリログのパージ
- 回復アカウントのパスワードのリセット

スーパー読取り専用およびインスタンス

グループレプリケーションが停止するたびに、インスタンスへの書込みが行われないように、super_read_only 変数が ON に設定されます。 このようなインスタンスを次の AdminAPI コマンドで使用しようとすると、インスタンスに super_read_only=OFF を設定することを選択できます:

- dba.configureInstance()
- dba.configureLocalInstance()
- dba.dropMetadataSchema()

AdminAPI が super_read_only=ON を持つインスタンスを検出すると、対話モードで super_read_only=OFF を設定することを選択できます。 例:

```
mysql-js> var myCluster = dba.dropMetadataSchema()
Are you sure you want to remove the Metadata? [y/N]: y
The MySQL instance at 'localhost:3310' currently has the super_read_only system
variable set to protect it from inadvertent updates from applications. You must
first unset it to be able to perform any changes to this instance.
For more information see:
https://dev.mysql.com/doc/refman/en/server-system-variables.html#sysvar_super_read_only.
```

```
Do you want to disable super_read_only and continue? [y/N]: y

Metadata Schema successfully removed.
```

インスタンスに対する現在のアクティブセッションの数が表示されます。 アプリケーションが誤ってインスタンスに書き込めないようにする必要があります。 y に応答することで、AdminAPI がインスタンスに書き込めることを確認します。 リストされているインスタンスに複数のオープンセッションがある場合は、AdminAPI に super_read_only=OFF の設定を許可する前に注意してください。

AdminAPI のユーザーの構成

InnoDB クラスタ または InnoDB ReplicaSet に属するインスタンスを使用するには、インスタンスを管理する適切な権限を持つユーザーでインスタンスに接続する必要があります。AdminAPI には、適切なユーザーを管理する次の方法が用意されています:

- 8.0.20 以降のバージョンでは、setupAdminAccount(user) 操作を使用して、InnoDB クラスタ または InnoDB ReplicaSet の管理に必要な権限を持つ MySQL ユーザーアカウントを作成またはアップグレードします。
- 8.0.20 より前のバージョンでは、管理用のユーザーを作成するには、dba.configureInstance() 操作で clusterAdmin オプションを使用することをお薦めします。

詳細は、管理用のユーザーアカウントの作成を参照してください。 管理ユーザーを手動で構成する場合、そのユーザーには次の権限 (すべて GRANT OPTION を含む) が必要です:

- RELOAD, SHUTDOWN, PROCESS, FILE, SELECT, SUPER, REPLICATION SLAVE, REPLICATION CLIENT, REPLICATION_APPLIER, CREATE USER, SYSTEM_VARIABLES_ADMIN, PERSIST_RO_VARIABLES_ADMIN, BACKUP_ADMIN, CLONE_ADMIN および EXECUTE の *.* に対するグローバル権限。

  > **注記**
  >
  > SUPER には、次の必要な権限が含まれます: SYSTEM_VARIABLES_ADMIN, SESSION_VARIABLES_ADMIN, REPLICATION_SLAVE_ADMIN, GROUP_REPLICATION_ADMIN, REPLICATION_SLAVE_ADMIN, ROLE_ADMIN。

- mysql_innodb_cluster_metadata.*、mysql_innodb_cluster_metadata_bkp.* および mysql_innodb_cluster_metadata_previous.* のスキーマ固有の権限は ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE VIEW, DELETE, DROP, EVENT, EXECUTE, INDEX, INSERT, LOCK TABLES, REFERENCES, SHOW VIEW, TRIGGER, UPDATE で、mysql.* の権限は INSERT, UPDATE, DELETE です。

  > **注記**
  >
  > この権限のリストは、現在のバージョンの MySQL Shell に基づいています。 権限はリリース間で変更される可能性があります。 したがって、アカウントを管理するための推奨方法は、管理用のユーザーアカウントの作成 で説明されている操作を使用することです

監視目的でユーザーを作成する場合など、読取り操作のみが必要な場合は、より制限された権限を持つアカウントを使用できます。 InnoDB クラスタ の監視に必要な権限をユーザー your_user に付与するには、次のコマンドを発行します:

```
GRANT SELECT ON mysql_innodb_cluster_metadata.* TO your_user@'%';
GRANT SELECT ON performance_schema.global_status TO your_user@'%';
GRANT SELECT ON performance_schema.global_variables TO your_user@'%';
GRANT SELECT ON performance_schema.replication_applier_configuration TO your_user@'%';
GRANT SELECT ON performance_schema.replication_applier_status TO your_user@'%';
GRANT SELECT ON performance_schema.replication_applier_status_by_coordinator TO your_user@'%';
GRANT SELECT ON performance_schema.replication_applier_status_by_worker TO your_user@'%';
GRANT SELECT ON performance_schema.replication_connection_configuration TO your_user@'%';
GRANT SELECT ON performance_schema.replication_connection_status TO your_user@'%';
GRANT SELECT ON performance_schema.replication_group_member_stats TO your_user@'%';
GRANT SELECT ON performance_schema.replication_group_members TO your_user@'%';
GRANT SELECT ON performance_schema.threads TO your_user@'%' WITH GRANT OPTION;
```

詳細は、アカウント管理ステートメントを参照してください。

InnoDB クラスタ および自動増分

InnoDB クラスタ の一部としてインスタンスを使用している場合、マルチプライマリクラスタの自動増分衝突が最大 9 (グループレプリケーショングループの最大サポートサイズ) になる可能性を回避するために、auto_increment_increment および auto_increment_offset 変数が構成されます。 これらの変数の構成に使用されるロジックは、次のように要約できます:

- グループがシングルプライマリモードで実行されている場合は、auto_increment_increment を 1 に、auto_increment_offset を 2 に設定します。
- グループがマルチプライマリモードで実行されている場合、クラスタに 7 つ以下のインスタンスがあると、auto_increment_increment は 7 に設定され、auto_increment_offset は 1 + server_id % 7 に設定されます。 マルチプライマリクラスタに 8 つ以上のインスタンスがある場合、auto_increment_increment をインスタンス数に設定し、auto_increment_offset を 1 + server_id % をインスタンス数に設定します。

InnoDB クラスタおよびバイナリログのパージ

MySQL 8 では、バイナリログは (binlog_expire_logs_seconds で定義されているように) 自動的にパージされます。 つまり、binlog_expire_logs_seconds より長い時間実行されていたクラスタには、最終的に、インスタンスによって適用されたすべてのトランザクションを含む完全なバイナリログを持つインスタンスが含まれていない可能性があります。 これにより、インスタンスをクラスタに参加させる前に、たとえば MySQL Enterprise Backup を使用してインスタンスを自動的にプロビジョニングする必要がある場合があります。 8.0.17 以降を実行しているインスタンスは、増分リカバリに依存しない自動プロビジョニングソリューションを提供することでこの問題を解決する MySQL クローンプラグインをサポートしています。セクション6.2.2.2「InnoDB クラスタ での MySQL クローンの使用」 を参照してください。 8.0.17 より前のバージョンを実行しているインスタンスは増分リカバリのみをサポートしているため、インスタンスが実行されている MySQL のバージョンによっては、インスタンスを自動的にプロビジョニングする必要がある場合があります。 そうしないと、Cluster.addInstance() などの分散リカバリに依存する操作が失敗する可能性があります。

以前のバージョンの MySQL を実行しているインスタンスでは、バイナリログのパージに次のルールが使用されます:

- 8.0.1 より前のバージョンを実行しているインスタンスでは、expire_logs_days のデフォルト値が 0 であるため、バイナリログの自動パージは行われません。
- 8.0.1 より後のバージョンを実行しているが、8.0.4 より前のバージョンを実行しているインスタンスでは、expire_logs_days のデフォルト値が 30 であるため、30 日後にバイナリログがパージされます。
- binlog_expire_logs_seconds のデフォルト値は 2592000 で、expire_logs_days のデフォルト値は 0 であるため、8.0.10 より後のバージョンを実行しているインスタンスは 30 日後にバイナリログをパージします。

したがって、クラスタでバイナリログが実行されている期間によっては、パージされた可能性があり、インスタンスを手動でプロビジョニングする必要がある場合があります。 同様に、バイナリログを手動でパージした場合も、同じ状況が発生する可能性があります。 したがって、分散リカバリのために MySQL クローンによって提供される自動プロビジョニングを最大限に活用し、InnoDB クラスタ のインスタンスをプロビジョニングする際の停止時間を最小限に抑えるために、8.0.17 より後のバージョンの MySQL にアップグレードすることを強くお薦めします。

回復アカウントのパスワードのリセット

バージョン 8.0.18 から、Cluster.resetRecoveryAccountsPassword() 操作を使用して、カスタムパスワード存続期間ポリシーに従うなど、InnoDB クラスタ によって作成された内部リカバリアカウントのパスワードをリセットできます。 Cluster.resetRecoveryAccountsPassword() 操作を使用して、クラスタで使用されるすべての内部リカバリアカウントのパスワードをリセットします。 この操作では、オンラインの各インスタンスの内部リカバリアカウントに新しいランダムパスワードが設定されます。 インスタンスに到達できない場合、操作は失敗します。 force オプションを使用してこのようなインスタンスを無視できますが、これはお薦めしません。この操作を使用する前に、インスタンスをオンラインに戻す方が安全です。 この操作は、InnoDB クラスタによって作成されたパスワードにのみ適用され、手動で作成されたパスワードの更新には使用できません。

> 注記
>
> リカバリアカウントのパスワードをパスワード verification-required ポリシーに関係なく変更できるようにするには、この操作を実行するユーザーには、特に CREATE USER で必要

なすべての管理権限が必要です。 つまり、password_require_current システム変数が有効かどうかには関係ありません。

6.2.11 既知の制限事項

このセクションでは、InnoDB クラスタ の既知の制限事項について説明します。 InnoDB クラスタ ではグループレプリケーションを使用するため、制限にも注意する必要があります。グループレプリケーションの制限事項 を参照してください。

- グローバルセッションの作成時にセッションタイプが指定されていない場合、MySQL Shell では、最初に NodeSession の作成を試行し、失敗した場合は ClassicSession の作成を試行する自動プロトコル検出が提供されます。 読取り/書込みポートが 1 つおよび読取り専用ポートが 2 つある 3 つのサーバーインスタンスで構成される InnoDB クラスタでは、MySQL Shell が読取り専用インスタンスのいずれかにのみ接続する可能性があります。 したがって、グローバルセッションの作成時には、常にセッションタイプを指定することをお薦めします。

- 非サンドボックスサーバーインスタンス (dba.deploySandboxInstance() を使用せずに手動で構成したインスタンス) をクラスタに追加する場合、MySQL Shell はインスタンス構成ファイルで構成の変更を永続化できません。 これにより、次のいずれかまたは両方のシナリオが発生します:

  1. グループレプリケーション構成はインスタンス構成ファイルに永続化されず、再起動時にインスタンスはクラスタに再参加しません。

  2. インスタンスはクラスタ使用に対して有効ではありません。 インスタンスは dba.checkInstanceConfiguration() を使用して検証でき、MySQL Shell では、インスタンスをクラスタで使用できるようにするために必要な構成変更が行われますが、これらの変更は構成ファイルに永続化されないため、再起動が発生すると失われます。

  a のみが発生した場合、インスタンスは再起動後にクラスタに再参加しません。

  b も発生し、再起動後にインスタンスがクラスタに再参加しなかった場合、この状況では推奨される dba.rebootClusterFromCompleteOutage() を使用してクラスタをオンラインに戻すことはできません。 これは、インスタンスが MySQL Shell によって行われた構成変更を失い、永続化されなかったため、インスタンスはクラスタ用に構成される前に前の状態に戻ります。 これにより、Group Replication が応答を停止し、最終的にコマンドがタイムアウトします。

  この問題を回避するには、構成の変更を永続化するために、クラスタにインスタンスを追加する前に dba.configureInstance() を使用することを強くお薦めします。

- オプションファイルを指定するための --defaults-extra-file オプションの使用は、InnoDB クラスタ サーバーインスタンスではサポートされていません。InnoDB クラスタ では、インスタンス上の単一のオプションファイルのみがサポートされ、追加のオプションファイルはサポートされません。 したがって、インスタンスオプションファイルを操作する場合は、メインファイルを指定する必要があります。 複数のオプションファイルを使用する場合は、ファイルを手動で構成し、複数のオプションファイルの使用の優先順位ルールを考慮して正しく更新されていることを確認し、目的の設定が認識されない余分なオプションファイル内のオプションによって誤って上書きされないようにします。

- 実際のネットワークインタフェースと一致しない IP アドレスに解決されるホスト名を持つインスタンスを使用しようとすると、「このインスタンスは自身の住所を the hostname として報告」というエラーで失敗します。 これは、Group Replication 通信レイヤーではサポートされません。 Debian ベースのインスタンスでは、localhost が存在しない IP (127.0.1.1 など) に解決されるため、インスタンスは user@localhost などのアドレスを使用できません。 これは、通常は単一マシン上のローカルインスタンスを使用するサンドボックスデプロイメントの使用に影響します。

  回避策は、マシンの実際の IP アドレスを使用するように各インスタンスで report_host システム変数を構成することです。 マシンの IP を取得し、各インスタンスの my.cnf ファイルに report_host=IP of your machine を追加します。 変更するには、インスタンスが再起動されていることを確認する必要があります。

- Cluster.addInstance() を実行して dba.createCluster() を実行するか、既存の InnoDB クラスタ にインスタンスを追加すると、次のエラーが MySQL エラーログに記録されます:

```
2020-02-10T10:53:43.727246Z 12 [ERROR] [MY-011685] [Repl] Plugin
group_replication reported: 'The group name option is mandatory'
2020-02-10T10:53:43.727292Z 12 [ERROR] [MY-011660] [Repl] Plugin
group_replication reported: 'Unable to start Group Replication on boot'
```

これらのメッセージは無害であり、AdminAPI が Group Replication を起動する方法に関連しています。

- サンドボックスデプロイメントを使用する場合、各サンドボックスインスタンスは、$PATH のローカル mysql-sandboxes ディレクトリにある mysqld バイナリのコピーを使用します。 アップグレード後などに mysqld のバージョンが変更された場合、前のバージョンに基づくサンドボックスの起動に失敗します。 これは、サンドボックスバイナリが basedir の下にある依存関係と比較して古くなっているためです。 サンドボックスインスタンスは本番用に設計されていないため、一時的とみなされ、アップグレードではサポートされません。

  この問題を回避するには、アップグレードした mysqld バイナリを各サンドボックスの bin ディレクトリに手動でコピーします。 次に、dba.startSandboxInstance() を発行してサンドボックスを起動します。 操作はタイムアウトで失敗し、エラーログには次の情報が含まれます:

  ```
  2020-03-26T11:43:12.969131Z 5 [System] [MY-013381] [Server] Server upgrade
  from '80019' to '80020' started.
  2020-03-26T11:44:03.543082Z 5 [System] [MY-013381] [Server] Server upgrade
  from '80019' to '80020' completed.
  ```

  操作はタイムアウトで失敗したようですが、サンドボックスは正常に起動しました。

- InnoDB クラスタ は、手動で構成された非同期レプリケーションチャネルを管理しません。 グループレプリケーションおよび AdminAPI では、非同期レプリケーションがプライマリでのみアクティブであり、状態がインスタンス間でレプリケートされないことは保証されません。 これにより、レプリケーションが機能しなくなり、分割ブレーンが発生する可能性がある様々なシナリオが発生する可能性があります。 したがって、ある InnoDB クラスタ と別の InnoDB クラスタ の間のレプリケーションもサポートされていません。

6.3 MySQL InnoDB ReplicaSet

このセクションでは、バージョン 8.0.19 で追加された InnoDB ReplicaSet について説明します。

6.3.1 InnoDB ReplicaSet の概要

AdminAPI には、InnoDB クラスタ と同様の方法で非同期 GTID ベースレプリケーションを実行する MySQL インスタンスのセットを管理できるようにする InnoDB ReplicaSet のサポートが含まれています。 InnoDB ReplicaSet は、単一のプライマリおよび複数のセカンダリ (従来は MySQL レプリケーションソースおよびレプリカと呼ばれていました) で構成されます。 InnoDB ReplicaSet のステータスを確認したり、障害発生時に新しいプライマリに手動でフェイルオーバーするなど、ReplicaSet オブジェクトおよび AdminAPI 操作を使用して ReplicaSets を管理します。 InnoDB クラスタ と同様に、MySQL Router では InnoDB ReplicaSet に対するブートストラップがサポートされているため、手動で構成しなくても InnoDB ReplicaSet を使用するように MySQL Router を自動的に構成できます。 これにより、InnoDB ReplicaSet は、MySQL レプリケーションおよび MySQL Router を迅速かつ簡単に起動して実行できるため、読取りのスケールアウトに適しており、InnoDB クラスタ で提供される高可用性を必要としないユースケースでは手動フェイルオーバー機能を提供します。

AdminAPI を使用した InnoDB ReplicaSet のデプロイに加えて、既存のレプリケーション設定を採用できます。AdminAPI は、レプリケーション設定のトポロジに基づいて InnoDB ReplicaSet を構成します。 レプリケーション設定が採用されたら、最初からデプロイされた InnoDB ReplicaSet と同じ方法で管理します。 これにより、新しい ReplicaSet を作成せずに、AdminAPI および MySQL Router を利用できます。 詳細は、セクション6.3.4「既存のレプリケーション設定の採用」 を参照してください。

InnoDB ReplicaSet の制限事項

InnoDB ReplicaSet には InnoDB クラスタ と比較していくつかの制限があるため、可能なかぎり InnoDB クラスタ をデプロイすることをお薦めします。 通常、InnoDB ReplicaSet 自体は高可用性を提供しません。 InnoDB ReplicaSet の制限事項は次のとおりです:

- 自動フェイルオーバーは行われません。 プライマリが使用できなくなった場合は、変更を再度行う前に、AdminAPI を使用してフェイルオーバーを手動でトリガーする必要があります。 ただし、セカンダリインスタンスは読取り可能なままです。

- 予期しない停止または使用不可による部分的なデータ損失からの保護はありません。 停止時までにまだ適用されていないトランザクションは失われる可能性があります。

- 予期しない終了または使用不可の後は、不整合から保護しません。 (たとえば、ネットワークパーティションが原因で) 元のプライマリがまだ使用可能な間にフェイルオーバーによってセカンダリが昇格されると、スプリットブレインのために不整合が発生する可能性があります。
- InnoDB ReplicaSet は、マルチプライマリモードをサポートしていません。 すべてのメンバーで書込みが可能なクラシックレプリケーショントポロジでは、データ整合性を保証できません。
- InnoDB ReplicaSet は非同期レプリケーションに基づいているため、読取りスケールアウトが制限されています。 そのため、Group Replication の場合とは異なり、フロー制御のチューニングはできません。
- すべてのセカンダリメンバーが単一のソースからレプリケートされます。 一部の特定のシナリオまたはユースケースでは、これがソースに影響する可能性があります。 たとえば、非常に小規模な更新が多数行われています。

6.3.2 InnoDB ReplicaSet のデプロイ

InnoDB ReplicaSet は、InnoDB クラスタ と同様の方法でデプロイします。 まず、いくつかの MySQL サーバーインスタンスを構成します。最小インスタンスは 2 つです。セクション6.1「MySQL AdminAPI」 を参照してください。 一方はプライマリとして、このチュートリアル rs-1 では機能します。もう一方のインスタンスはセカンダリとして機能し、このチュートリアル rs-2 ではプライマリによって適用されるトランザクションをレプリケートします。 これは、非同期 MySQL レプリケーションから認識されるソースおよびレプリカと同等です。 次に、MySQL Shell を使用していずれかのインスタンスに接続し、ReplicaSet を作成します。 ReplicaSet を作成したら、それにインスタンスを追加できます。

InnoDB ReplicaSet はサンドボックスインスタンスと互換性があり、テスト目的などでローカルにデプロイするために使用できます。 手順については、サンドボックスインスタンスのデプロイを参照してください。 ただし、このチュートリアルでは、各インスタンスが異なるホストで実行されている本番 InnoDB ReplicaSet をデプロイすることを前提としています。

InnoDB ReplicaSet の前提条件

InnoDB ReplicaSet を使用するには、次の前提条件に注意する必要があります:

- MySQL バージョン 8.0 以降を実行しているインスタンスのみがサポートされます
- GTID ベースのレプリケーションのみがサポートされており、バイナリログファイルの位置レプリケーションは InnoDB ReplicaSet と互換性がありません
- 行ベースのレプリケーション (RBR) のみがサポートされ、ステートメントベースのレプリケーション (SBR) はサポートされていません
- レプリケーションフィルタはサポートされていません
- 管理対象外レプリケーションチャネルはどのインスタンスでも許可されていません
- ReplicaSet は最大 1 つのプライマリインスタンスで構成され、1 つまたは複数のセカンダリがサポートされます。 ReplicaSet に追加できるセカンダリ数に制限はありませんが、ReplicaSet に接続されている各 MySQL Router は各インスタンスを監視する必要があります。 したがって、ReplicaSet に追加されるインスタンスが多いほど、より多くの監視を行う必要があります。
- ReplicaSet は、MySQL Shell によって完全に管理される必要があります。 たとえば、レプリケーションアカウントは MySQL Shell によって作成および管理されます。 たとえば、SQL ステートメントを直接使用してプライマリを変更するなど、MySQL Shell 外部のインスタンスに対する構成変更はサポートされていません。 InnoDB ReplicaSet を使用する場合は、常に MySQL Shell を使用してください。

AdminAPI および InnoDB ReplicaSet を使用すると、基礎となる概念を深く理解しなくても、MySQL レプリケーションを使用できます。 ただし、背景情報については、レプリケーション を参照してください。

InnoDB ReplicaSet インスタンスの構成

dba.configureReplicaSetInstance(instance) を使用して、レプリカセットで使用する各インスタンスを構成します。MySQL Shell は、インスタンスに接続して構成するか、instance を渡して特定のリモートインスタンスを構成できます。 ReplicaSet でインスタンスを使用するには、永続化設定をサポートする必要があります。 設定の永続化を参照してください。

管理タスクのためにインスタンスに接続する場合、適切な権限を持つユーザーが必要です。 ReplicaSet を管理するユーザーを作成するには、setupAdminAccount() 操作を使用することをお薦めします。 管理用のユーザーアカウントの作成を参照してください。 または、clusterAdmin オプションが指定されている場合は、dba.configureReplicaSetInstance() 操作で管理者アカウントをオプションで作成できます。 アカウントは、InnoDB ReplicaSet の管理に必要な正しい権限セットで作成されます。

> ヒント
>
> 管理者アカウントは、同じクラスタまたはレプリカセットのすべてのインスタンスで同じユーザー名とパスワードを持つ必要があります。

rsadmin という名前のクラスタ管理者で、rs-1:3306 でインスタンスを構成するには、次のコマンドを発行します:

```
mysql-js> dba.configureReplicaSetInstance('root@rs-1:3306', {clusterAdmin: "'rsadmin'@'rs-1%'"});
```

対話型プロンプトは、指定されたユーザーに必要なパスワードを要求します。 MySQL Shell が現在接続されているインスタンスを構成するには、null のインスタンス定義を指定できます。 たとえば、次のコマンドを発行します:

```
mysql-js> dba.configureReplicaSetInstance('', {clusterAdmin: "'rsadmin'@'rs-1%'"});
```

対話型プロンプトは、指定されたユーザーに必要なパスワードを要求します。 これにより、MySQL Shell が現在接続されているインスタンスが InnoDB ReplicaSet での使用に有効かどうかがチェックされます。 可能であれば、InnoDB ReplicaSet と互換性のない設定が構成されます。 クラスタ管理者アカウントは、InnoDB ReplicaSet に必要な権限で作成されます。

InnoDB ReplicaSet の作成

インスタンスを構成したら、InnoDB クラスタ で使用される MySQL Group Replication ではなく、インスタンスに接続し、dba.createReplicaSet() を使用して MySQL 非同期レプリケーションを使用する管理対象 ReplicaSet を作成します。 MySQL Shell が現在接続されている MySQL インスタンスは、ReplicaSet の初期プライマリとして使用されます。

dba.createReplicaSet() 操作では、インスタンスの状態および構成が管理対象 ReplicaSet と互換性があることを確認するために複数のチェックが実行され、互換性がある場合はインスタンスでメタデータスキーマが初期化されます。 操作をチェックするが、実際にはインスタンスを変更しない場合は、dryRun オプションを使用します。 これにより、MySQL Shell が ReplicaSet を作成するために実行するアクションがチェックされ、表示されます。 ReplicaSet が正常に作成されると、ReplicaSet オブジェクトが返されます。 したがって、返された ReplicaSet を変数に割り当てることをお薦めします。 これにより、ReplicaSet.status() 操作をコールするなどして、ReplicaSet を操作できます。 インスタンス rs-1 で example という名前の ReplicaSet を作成し、rs 変数に割り当てるには、次のように発行します:

```
mysql-js> \connect root@rs-1:3306
...
mysql-js> var rs = dba.createReplicaSet("example")
A new replicaset with instance 'rs-1:3306' will be created.

* Checking MySQL instance at rs-1:3306

This instance reports its own address as rs-1:3306
rs-1:3306: Instance configuration is suitable.

* Updating metadata...

ReplicaSet object successfully created for rs-1:3306.
Use rs.add_instance() to add more asynchronously replicated instances to this replicaset
and rs.status() to check its status.
```

操作が成功したことを確認するには、返された ReplicaSet オブジェクトを操作します。 たとえば、ReplicaSet に関する情報を表示する ReplicaSet.status() 操作を提供します。 返された ReplicaSet はすでに変数 rs に割り当てられているため、次のコマンドを発行します:

```
mysql-js> rs.status()
{
    "replicaSet": {
        "name": "example",
        "primary": "rs-1:3306",
        "status": "AVAILABLE",
```

```
        "statusText": "All instances available.",
        "topology": {
            "rs-1:3306": {
                "address": "rs-1:3306",
                "instanceRole": "PRIMARY",
                "mode": "R/W",
                "status": "ONLINE"
            }
        },
        "type": "ASYNC"
    }
}
```

この出力は、example という名前の ReplicaSet が作成され、プライマリが rs-1 であることを示しています。 現在、インスタンスは 1 つのみで、次のタスクは ReplicaSet にインスタンスを追加することです。

6.3.3 ReplicaSet へのインスタンスの追加

ReplicaSet を作成したら、ReplicaSet.addInstance() 操作を使用して、ReplicaSet の現在のプライマリの読取り専用セカンダリレプリカとしてインスタンスを追加できます。 ReplicaSet のプライマリは、この操作中にアクセス可能で使用可能である必要があります。 MySQL レプリケーションは、ランダムパスワードを使用して自動的に作成された MySQL アカウントを使用して、追加されたインスタンスとプライマリの間で構成されます。 インスタンスをオペレーショナルセカンダリにするには、プライマリと同期している必要があります。 このプロセスはリカバリと呼ばれ、InnoDB ReplicaSet では、recoveryMethod オプションを使用して構成する様々な方法がサポートされます。

インスタンスが ReplicaSet に参加できるようにするには、様々な前提条件を満たす必要があります。 これらは ReplicaSet.addInstance() によって自動的にチェックされ、問題が見つかった場合は操作が失敗します。 インスタンスを追加する前に、dba.configureReplicaSetInstance() を使用してバイナリログおよびレプリケーション関連のオプションを検証および構成します。MySQL Shell は、ReplicaSet ハンドルオブジェクトの取得に使用したものと同じユーザー名とパスワードを使用してターゲットインスタンスに接続します。 ReplicaSet のすべてのインスタンスには、同じ権限付与およびパスワードを持つ同じ管理者アカウントが必要です。 必要な権限を持つカスタム管理者アカウントは、インスタンスが dba.configureReplicaSetInstance() で構成されている間に作成できます。 InnoDB ReplicaSet インスタンスの構成を参照してください。

InnoDB ReplicaSet のリカバリ方法

新しいインスタンスを InnoDB ReplicaSet に追加する場合は、それに含まれる既存のデータをプロビジョニングする必要があります。 これは、次のいずれかの方法を使用して自動的に実行できます:

- MySQL クローン:オンラインインスタンスからスナップショットを取得し、新しいインスタンスのデータをスナップショットに置き換えます。 MySQL クローンは、新しい空白インスタンスを InnoDB ReplicaSet に結合する場合に適しています。 InnoDB ReplicaSet によって適用されるすべてのトランザクションの完全なバイナリログがあることに依存するわけではありません。

  > **警告**
  >
  > 追加されるインスタンス上の以前のデータはすべて、クローン操作中に破棄されます。 ただし、テーブルに格納されていないすべての MySQL 設定は保持されます。

- 増分リカバリ: MySQL レプリケーションに依存して、欠落しているすべてのトランザクションを新しいインスタンスに適用します。 新しいインスタンスで欠落しているトランザクションの量が少ない場合、これが最速の方法になる可能性があります。 ただし、この方法は、InnoDB ReplicaSet 内の少なくとも 1 つのオンラインインスタンスに、InnoDB ReplicaSet のトランザクション履歴全体を含む完全なバイナリログがある場合にのみ使用できます。バイナリログがすべてのメンバーからパージされた場合、またはバイナリログがインスタンスにすでに存在する後にのみ有効化された場合、このメソッドは使用できません。 適用するトランザクションの量が非常に多い場合、インスタンスが InnoDB ReplicaSet に参加するまでに長い遅延が発生する可能性があります。

インスタンスが ReplicaSet に参加している場合、リカバリは InnoDB クラスタ とほぼ同じ方法で使用されます。MySQL Shell は、適切なリカバリ方法を自動的に選択しようとします。 安全に方法を選択できない場合は、MySQL Shell によって使用する内容の入力を求められます。 詳細は、セクション6.2.2.2「InnoDB クラスタ での MySQL クローンの使用」を参照してください。 このセクションでは、ReplicaSet にインスタンスを追加する場合の相違点について説明します。

ReplicaSet へのインスタンスの追加

ReplicaSet.addInstance(instance) 操作を使用して、セカンダリインスタンスを ReplicaSet に追加します。 URI のような接続文字列として instance を指定します。 指定するユーザーは必要な権限を持っている必要があり、ReplicaSet のすべてのインスタンスで同じである必要があります。InnoDB ReplicaSet インスタンスの構成 を参照してください。

たとえば、ユーザー rsadmin を使用して rs-2 でインスタンスを追加するには、次のように発行します:

```
mysql-js> rs.addInstance('rsadmin@rs-2')

Adding instance to the replicaset...

* Performing validation checks

This instance reports its own address as rsadmin@rs-2
rsadmin@rs-2: Instance configuration is suitable.

* Checking async replication topology...

* Checking transaction state of the instance...

NOTE: The target instance 'rsadmin@rs-2' has not been pre-provisioned (GTID set
is empty). The Shell is unable to decide whether replication can completely
recover its state.  The safest and most convenient way to provision a new
instance is through automatic clone provisioning, which will completely
overwrite the state of 'rsadmin@rs-2' with a physical snapshot from an existing
replicaset member. To use this method by default, set the 'recoveryMethod'
option to 'clone'.

WARNING: It should be safe to rely on replication to incrementally recover the
state of the new instance if you are sure all updates ever executed in the
replicaset were done with GTIDs enabled, there are no purged transactions and
the new instance contains the same GTID set as the replicaset or a subset of it.
To use this method by default, set the 'recoveryMethod' option to 'incremental'.
Please select a recovery method [C]lone/[I]ncremental recovery/[A]bort (default Clone):
```

この場合、リカバリ方法を指定しなかったため、最適な方法が示されます。 この例では、ReplicaSet に参加するインスタンスに既存のトランザクションがないため、クローンオプションを選択します。 したがって、結合中のインスタンスからデータを削除するリスクはありません。

```
Please select a recovery method [C]lone/[I]ncremental recovery/[A]bort (default Clone): C
* Updating topology
Waiting for clone process of the new member to complete. Press ^C to abort the operation.
* Waiting for clone to finish...
NOTE: rsadmin@rs-2 is being cloned from rsadmin@rs-1
** Stage DROP DATA: Completed
** Clone Transfer
FILE COPY  ############################################################  100%  Completed
PAGE COPY  ############################################################  100%  Completed
REDO COPY  ############################################################  100%  Completed
** Stage RECOVERY: \
NOTE: rsadmin@rs-2 is shutting down...

* Waiting for server restart... ready
* rsadmin@rs-2 has restarted, waiting for clone to finish...
* Clone process has finished: 59.63 MB transferred in about 1 second (~1.00 B/s)

** Configuring rsadmin@rs-2 to replicate from rsadmin@rs-1
** Waiting for new instance to synchronize with PRIMARY...

The instance 'rsadmin@rs-2' was added to the replicaset and is replicating from rsadmin@rs-1.
```

インスタンスが InnoDB ReplicaSet の使用に有効であると仮定すると、リカバリは続行されます。 この場合、新しく結合されたインスタンスは MySQL クローンを使用して、まだプライマリから適用されていないすべてのトランザクションをコピーし、ReplicaSet をオンラインインスタンスとして結合します。 確認するには、ReplicaSet.status() 操作を使用します:

```
mysql-js> rs.status()
{
    "replicaSet": {
        "name": "example",
        "primary": "rs-1:3306",
        "status": "AVAILABLE",
```

```
        "statusText": "All instances available.",
        "topology": {
            "rs-1:3306": {
                "address": "rs-1:3306",
                "instanceRole": "PRIMARY",
                "mode": "R/W",
                "status": "ONLINE"
            },
            "rs-2:3306": {
                "address": "rs-2:3306",
                "instanceRole": "SECONDARY",
                "mode": "R/O",
                "replication": {
                    "applierStatus": "APPLIED_ALL",
                    "applierThreadState": "Replica has read all relay log; waiting for more updates",
                    "receiverStatus": "ON",
                    "receiverThreadState": "Waiting for source to send event",
                    "replicationLag": null
                },
                "status": "ONLINE"
            }
        },
        "type": "ASYNC"
    }
}
```


この出力は、example という名前の ReplicaSet が 2 つの MySQL インスタンスで構成され、プライマリが rs-1 であることを示しています。 現在、プライマリのレプリカであるセカンダリインスタンスが rs-2 にあります。 ReplicaSet はオンラインであり、プライマリとセカンダリが同期していることを意味します。 この時点で、ReplicaSet はトランザクションを処理する準備ができています。

最適なリカバリ方法を選択しようとする対話型 MySQL Shell モードをオーバーライドする場合は、recoveryMethod オプションを使用して、ReplicaSet に参加するために必要なデータをインスタンスがリカバリする方法を構成します。 詳細は、セクション6.2.2.2「InnoDB クラスタ での MySQL クローンの使用」を参照してください。

6.3.4 既存のレプリケーション設定の採用

新しい InnoDB ReplicaSet を作成するかわりに、dba.createReplicaSet() で adoptFromAR オプションを使用して既存のレプリケーション設定を採用することもできます。 レプリケーション設定がスキャンされ、InnoDB ReplicaSet の前提条件 と互換性がある場合は、AdminAPI によって必要なメタデータが作成されます。 レプリケーション設定が採用されると、InnoDB ReplicaSet の管理にのみ AdminAPI を使用できます。

既存のレプリケーション設定を InnoDB ReplicaSet に変換するには、プライマリ (ソースとも呼ばれる) に接続します。 レプリケーショントポロジは、MySQL Shell グローバルセッションが接続されているインスタンスから開始して、自動的にスキャンおよび検証されます。 すべてのインスタンスの構成は、InnoDB ReplicaSet の使用と互換性があることを確認するために、採用時にチェックされます。 すべてのレプリケーションチャネルがアクティブであり、GTID セットを介して検証されたトランザクションセットに一貫性がある必要があります。 インスタンスは同じ状態であるか、収束可能であるとみなされます。 トポロジの一部であるすべてのインスタンスは、ReplicaSet に自動的に追加されます。 この操作によって採用された ReplicaSet に加えられた変更は、メタデータスキーマの作成のみです。 既存のレプリケーションチャネルは採用中に変更されませんが、後続のプライマリスイッチ操作中に変更できます。

たとえば、example1 および example2 上の MySQL サーバーインスタンスで構成されるレプリケーショントポロジを InnoDB ReplicaSet に採用するには、example1 のプライマリに接続し、次を発行します:

```
mysql-js> rs = dba.createReplicaSet('testadopt', {'adoptFromAR':1})
A new replicaset with the topology visible from 'example1:3306' will be created.

* Scanning replication topology...
** Scanning state of instance example1:3306
** Scanning state of instance example2:3306

* Discovering async replication topology starting with example1:3306
Discovered topology:
- example1:3306: uuid=00371d66-3c45-11ea-804b-080027337932 read_only=no
- example2:3306: uuid=59e4f26e-3c3c-11ea-8b65-080027337932 read_only=no
    - replicates from example1:3306
 source="localhost:3310" channel= status=ON receiver=ON applier=ON
```

```
* Checking configuration of discovered instances...

This instance reports its own address as example1:3306
example1:3306: Instance configuration is suitable.

This instance reports its own address as example2:3306
example2:3306: Instance configuration is suitable.

* Checking discovered replication topology...
example1:3306 detected as the PRIMARY.
Replication state of example2:3306 is OK.

Validations completed successfully.

* Updating metadata...

ReplicaSet object successfully created for example1:3306.
Use rs.add_instance() to add more asynchronously replicated instances to
this replicaset and rs.status() to check its status.
```

InnoDB ReplicaSet が採用されたら、最初から作成された ReplicaSet を使用するのと同じ方法で使用できます。 この時点から、AdminAPI のみを使用して InnoDB ReplicaSet を管理する必要があります。

6.3.5 InnoDB ReplicaSet の操作

InnoDB ReplicaSet は、InnoDB クラスタ とほぼ同じ方法で操作します。 たとえば、ReplicaSet へのインスタンスの追加 に表示されているように、ReplicaSet オブジェクトを変数に割り当て、ReplicaSet.addInstance() などの ReplicaSet を管理する操作をコールして、InnoDB クラスタ の Cluster.addInstance() と同等のインスタンスを追加します。 したがって、セクション6.2.5「InnoDB クラスタの操作」 のドキュメントの多くは、InnoDB ReplicaSet にも適用されます。 ReplicaSet オブジェクトでは、次の操作がサポートされています:

- \help ReplicaSet または ReplicaSet.help() と\help dba または dba.help() を使用して、ReplicaSet オブジェクトおよび AdminAPI のオンラインヘルプを取得します。 セクション6.1「MySQL AdminAPI」を参照してください。
- name または ReplicaSet.getName() を使用して、ReplicaSet オブジェクトの名前をすばやく確認できます。 たとえば、次は同等です:

  ```
  mysql-js> rs.name
  example
  mysql-js> rs.getName()
  example
  ```

- 様々なレベルの詳細を取得する extended オプションをサポートする ReplicaSet.status() 操作を使用して、ReplicaSet に関する情報を確認します。 例:
  - extended のデフォルトは 0(通常の詳細レベル) です。 デフォルト以外または予期しないレプリケーション設定およびステータスに加えて、インスタンスおよびレプリケーションのステータスに関する基本情報のみが含まれます。
  - extended を 1 に設定すると、メタデータバージョン、サーバー UUID、ラグやワーカースレッドなどのレプリケーション情報、インスタンスのステータスの導出に使用される RAW 情報、アプライヤキューのサイズ、予期しない書込みから保護するシステム変数の値などが含まれます。
  - extended を 2 に設定すると、暗号化された接続などの重要なレプリケーション関連の構成設定が含まれます。

  ReplicaSet.status(extended=1) の出力は Cluster.status(extended=1) と非常に似ていますが、主な違いは、増分リカバリ中に使用する InnoDB クラスタ とは異なり、InnoDB ReplicaSet は常に MySQL レプリケーションに依存するため、replication フィールドを使用できることです。 フィールドの詳細は、Cluster.status() によるクラスタステータスの確認 を参照してください。
- ReplicaSet.addInstance() および ReplicaSet.removeInstance() 操作を使用して、ReplicaSet に使用されているインスタンスを変更します。 ReplicaSet へのインスタンスの追加およびInnoDB クラスタからのインスタンスの削除を参照してください。
- ReplicaSet.rejoinInstance() を使用して、フェイルオーバー後などに削除されたインスタンスを ReplicaSet に追加します。

- ReplicaSet.setPrimaryInstance() 操作を使用して、ReplicaSet のプライマリの別のインスタンスへの変更を安全に実行します。 ReplicaSet プライマリの計画済変更を参照してください。
- ReplicaSet.forcePrimaryInstance() 操作を使用して、プライマリの強制フェイルオーバーを実行します。 ReplicaSet でのプライマリインスタンスの強制を参照してください。
- InnoDB クラスタ とまったく同じ方法で、ReplicaSet に対してブートストラップされた MySQL Router インスタンスを操作します。 ReplicaSet.listRouters() および ReplicaSet.removeRouterMetadata() の詳細は、クラスタルーターの操作 を参照してください。 InnoDB ReplicaSet での MySQL Router の使用の詳細は、MySQL Router での ReplicaSets の使用 を参照してください。
- バージョン 8.0.23 から、InnoDB ReplicaSet はパラレルレプリケーションアプライヤ (マルチスレッドレプリカと呼ばれることもあります) をサポートし、有効にします。 InnoDB ReplicaSet でパラレルレプリケーションアプライヤを使用するには、インスタンスに正しい設定が構成されている必要があります。 以前のバージョンからアップグレードする場合、インスタンスには更新された構成が必要です。 InnoDB ReplicaSet に属するインスタンスごとに、dba.configureReplicaSetInstance(instance) を発行して構成を更新します。 通常、dba.configureReplicaSetInstance() はレプリカセットにインスタンスを追加する前に使用されますが、この特別なケースでは、インスタンスを削除する必要はなく、構成の変更はオンライン中に行われます。 詳細は、パラレルレプリケーションアプリケーションの構成を参照してください。

  InnoDB ReplicaSet インスタンスは、replication フィールドの下の ReplicaSet.status(extended=1) 操作の出力でパラレルレプリケーションアプライヤに関する情報を報告します。

詳細は、リンクされた InnoDB クラスタ のセクションを参照してください。

次の操作は InnoDB ReplicaSet に固有であり、ReplicaSet オブジェクトに対してのみコールできます:

ReplicaSet プライマリの計画済変更

ReplicaSet.setPrimaryInstance() 操作を使用して、ReplicaSet のプライマリの別のインスタンスへの変更を安全に実行します。 現在のプライマリはセカンダリに降格され、読取り専用になりますが、昇格されたインスタンスは新しいプライマリになり、読取り/書込みになります。 他のすべてのセカンダリインスタンスは、新しいプライマリからレプリケートするように更新されます。ReplicaSet に対してブートストラップされた MySQL Router インスタンスは、新しいプライマリへの読取り/書込みクライアントのリダイレクトを自動的に開始します。

プライマリを安全に変更できるようにするには、すべてのレプリカセットインスタンスが MySQL Shell から到達可能であり、一貫性のある GTID_EXECUTED セットを持つ必要があります。 プライマリが使用できず、リストアする方法がない場合は、かわりに強制フェイルオーバーが唯一のオプションである可能性があります。ReplicaSet でのプライマリインスタンスの強制 を参照してください。

プライマリの変更中、昇格されたインスタンスは古いプライマリと同期され、トポロジの変更がコミットされる前にプライマリに存在するすべてのトランザクションが適用されます。 この同期化ステップに時間がかかりすぎるか、セカンダリインスタンスで実行できない場合、操作は中断されます。 このような状況でフェイルオーバーを可能にするには、これらの問題のあるセカンダリインスタンスを修復するか、ReplicaSet から削除する必要があります。

ReplicaSet でのプライマリインスタンスの強制

プライマリで予期しない障害が発生した場合に自動フェイルオーバーをサポートする InnoDB クラスタ とは異なり、InnoDB ReplicaSet には、グループレプリケーションによって提供されるプロトコルなどの自動障害検出またはコンセンサスベースのプロトコルはありません。 プライマリが使用できない場合は、手動フェイルオーバーが必要です。 プライマリを失った InnoDB ReplicaSet は事実上読取り専用であり、書込みの変更を可能にするには、新しいプライマリを選択する必要があります。 プライマリに接続できず、ReplicaSet プライマリの計画済変更 で説明されているように ReplicaSet.setPrimaryInstance() を使用して新しいプライマリへのスイッチオーバーを安全に実行できない場合は、ReplicaSet.forcePrimaryInstance() 操作を使用してプライマリの強制フェイルオーバーを実行します。 これは、現在のプライマリが使用できず、どのような方法でもリストアできない障害タイプのシナリオでのみ使用する必要がある最後のリゾート操作です。

> 警告
>
> 強制フェイルオーバーは潜在的に破壊的なアクションであり、注意して使用する必要があります。

ターゲットインスタンスが指定されていない (または null である) 場合、最新のインスタンスが自動的に選択され、新しいプライマリに昇格されます。 ターゲットインスタンスが指定されている場合はプライマリに昇格され、他の到達可能なセカンダリインスタンスは新しいプライマリからレプリケートするように切り替えられます。 ターゲットインスタンスは、到達可能なインスタンス間で最新の GTID_EXECUTED セットを持っている必要があります。そうでない場合、操作は失敗します。

フェイルオーバーは、古いプライマリとの同期や更新を行わずにセカンダリインスタンスを昇格するため、計画されたプライマリ変更とは異なります。 これには次のような大きな影響があります:

- 古いプライマリで障害が発生した時点でセカンダリによってまだ適用されていないトランザクションは失われます。
- 古いプライマリがまだ実行中でトランザクションを処理している場合は、スプリットブレインが存在し、古いプライマリと新しいプライマリのデータセットが相違します。

最新の既知のプライマリがまだ到達可能な場合、ReplicaSet.forcePrimaryInstance() 操作は失敗し、スプリットブレイン状況のリスクが軽減されます。 ただし、このようなシナリオを回避または最小化するために、古いプライマリに他のインスタンスからアクセスできないようにするのは管理者の責任です。

強制フェイルオーバーの後、古いプライマリは新しいプライマリによって無効とみなされ、レプリカセットに含めることはできなくなります。 後でインスタンスをリカバリする方法が見つかった場合は、ReplicaSet から削除し、新しいインスタンスとして再追加する必要があります。 フェイルオーバー中に新しいプライマリに切り替えることができなかったセカンダリインスタンスがあった場合は、それらも無効とみなされます。

フェイルオーバー後にデータが失われる可能性があります。これは、古いプライマリに、昇格されるセカンダリにまだレプリケートされていないトランザクションがある可能性があるためです。 さらに、障害が発生したとみなされたインスタンスがまだトランザクションを処理できる場合、たとえば、そのインスタンスが配置されているネットワークはまだ機能していますが、MySQL Shell からアクセスできないため、昇格されたインスタンスとの相違は続行されます。 インスタンスでのトランザクションセットの相違後のリカバリには手動操作が必要であり、障害が発生したインスタンスをリカバリできる場合でも状況によっては実行できなかった可能性があります。 多くの場合、強制フェイルオーバーが必要な障害からリカバリする最も高速で簡単な方法は、このような相違されたトランザクションを破棄し、新しく昇格されたプライマリから新しいインスタンスを再プロビジョニングすることです。

InnoDB ReplicaSet ロック

バージョン 8.0.20 から、AdminAPI はロックメカニズムを使用して、InnoDB ReplicaSet で様々な操作が同時に変更を実行しないようにしています。 以前は、MySQL Shell の異なるインスタンスが同時に InnoDB ReplicaSet に接続し、AdminAPI 操作を同時に実行できました。 これにより、ReplicaSet.addInstance() および ReplicaSet.setPrimaryInstance() がパラレルで実行された場合など、一貫性のないインスタンスの状態および エラーが発生する可能性があります。

InnoDB ReplicaSet 操作には、次のロックがあります:

- dba.upgradeMetadata() および dba.createReplicaSet() は、グローバルに排他的な操作です。 これは、MySQL Shell が InnoDB ReplicaSet でこれらの操作を実行する場合、InnoDB ReplicaSet またはそのインスタンスに対して他の操作を実行できないことを意味します。
- ReplicaSet.forcePrimaryInstance() および ReplicaSet.setPrimaryInstance() は、プライマリを変更する操作です。 これは、MySQL Shell がこれらの操作を InnoDB ReplicaSet に対して実行する場合、プライマリまたはインスタンス変更操作を変更する他の操作は、最初の操作が完了するまで実行できないことを意味します。
- ReplicaSet.addInstance(), ReplicaSet.rejoinInstance() および ReplicaSet.removeInstance() は、インスタンスを変更する操作です。 つまり、MySQL Shell がインスタンスでこれらの操作を実行すると、インスタンスはそれ以降のインスタンス変更操作のためにロックされます。 ただし、このロックはインスタンスレベルでのみ行われ、InnoDB ReplicaSet 内の複数のインスタンスがこのタイプの操作のいずれかを同時に実行できます。 つまり、InnoDB ReplicaSet のインスタンスごとに、一度に実行できるインスタンス変更操作は最大 1 つです。
- dba.getReplicaSet() および ReplicaSet.status() は InnoDB ReplicaSet の読取り操作であり、ロックは必要ありません。

実際には、同時に実行できない別の操作の実行中に InnoDB ReplicaSet 関連の操作を実行しようとすると、必要なリソースのロックの取得に失敗したことを示すエラーが表示されます。 この場合、ロックを保持する実行中の操作が完了するまで待機してから、次の操作の実行を試行する必要があります。 例:

```
mysql-js> rs.addInstance("admin@rs2:3306");

ERROR: The operation cannot be executed because it failed to acquire the lock on
instance 'rs1:3306'. Another operation requiring exclusive access to the
instance is still in progress, please wait for it to finish and try again.

ReplicaSet.addInstance: Failed to acquire lock on instance 'rs1:3306' (MYSQLSH
51400)
```

この例では、ReplicaSet.setPrimaryInstance() 操作 (または他の同様の操作) がまだ実行中であったなどの理由で、プライマリインスタンス (rs1:3306) のロックを取得できなかったため、ReplicaSet.addInstance() 操作が失敗しました。

ReplicaSets のタグ付け

タグ付けは、ReplicaSets とそのインスタンスでサポートされます。 タグ付けのために、ReplicaSets は setOption()、setInstanceOption() および options() 操作をサポートしています。 これらの操作は、通常、Cluster と同等の方法で機能します。 詳細は、セクション6.2.9「メタデータのタグ付け」を参照してください。 このセクションでは、ReplicaSets のタグの使用における相違点について説明します。

> **重要**
>
> ReplicaSets およびそのインスタンス用に構成できる他のオプションはありません。 ReplicaSets では、InnoDB クラスタ のオプションの設定 に記載されているオプションはサポートされていません。 サポートされているオプションは、ここで説明するタグ付けのみです。

ReplicaSet.options() 操作では、個々の ReplicaSet インスタンスおよび ReplicaSet 自体に割り当てられたタグに関する情報が表示されます。

ReplicaSet.setOption() および ReplicaSet.setInstanceOption() の option 引数では、tag ネームスペースのオプションのみがサポートされ、それ以外の場合はエラーがスローされます。

ReplicaSet.setInstanceOption(instance, option, value) および ReplicaSet.setOption(option, value) の操作は、Cluster の同等の操作と同じように動作します。

ルーティングからのインスタンスの削除 で説明されているように、インスタンスの非表示に違いはありません。 たとえば、ReplicaSet インスタンス rs-1 を非表示にするには、次のコマンドを発行します:

```
mysql-js> myReplicaSet.setInstanceOption(icadmin@rs-1:3306, "tag:_hidden", true);
```

ReplicaSet に対してブートストラップされた MySQL Router は、変更を検出し、rs-1 インスタンスをルーティング先から削除します。

6.4 MySQL Router

このセクションでは、MySQL Router を InnoDB クラスタ および InnoDB ReplicaSet と統合する方法について説明します。 MySQL Router の背景情報は、MySQL Router 8.0 を参照してください。

6.4.1 MySQL Router のブートストラップ

InnoDB ReplicaSet または InnoDB クラスタ に対して MySQL Router をブートストラップして、ルーティングを自動的に構成します。 ブートストラッププロセスは、MySQL Router を実行する特定の方法で、通常のルーティングは開始されず、かわりにメタデータに基づいて mysqlrouter.conf ファイルが構成されます。 コマンドラインで MySQL Router を強化するには、mysqlrouter コマンドの起動時に --bootstrap オプションを渡し、メタデータからトポロジ情報を取得して、サーバーインスタンスへのルーティング接続を構成します。 または、Windows では、MySQL Installer を使用して MySQL Router をブートストラップします。MySQL Installer での MySQL Router の構成 を参照してください。 MySQL Router がブートストラップされると、クライアントアプリケーションはパブリッシュするポートに接続します。MySQL Router は、着信ポートに基づいてクライアント接続をインスタンスに自動的にリダイレクトします。たとえば、6646 は、クラシック MySQL プロトコル を使用した読取り/書込み接続にデフォルトで使用されます。 たとえば、インスタンスに予期しない障害が発生したためにトポロジが変更された場合、MySQL Router はその変更を検出し、ルーティングを残りのインスタンスに自動的に調整します。 これにより、クライアントアプリケーションでフェイルオーバーを処理したり、基礎となるトポロジに注意する必要がなくなります。 詳細は、Routing for MySQL InnoDB Clusterを参照してください。

> **注記**
>
> サーバーインスタンスにリダイレクトするように MySQL Router を手動で構成しないでください。 これにより、MySQL Router はメタデータから構成を取得できるため、常に --bootstrap オプションを使用してください。 Cluster Metadata and Stateを参照してください。

MySQL Router ユーザーの構成

MySQL Router が InnoDB クラスタ または InnoDB ReplicaSet に接続する場合、適切な権限を持つユーザーアカウントが必要です。 MySQL Router バージョン 8.0.19 からは、この内部ユーザーは --account オプションを使用して指定できます。 以前のバージョンでは、MySQL Router はクラスタのブートストラップごとに内部アカウントを作成したため、時間の経過とともに多数のアカウントが構築される可能性がありました。 MySQL Shell バージョン 8.0.20 から、AdminAPI を使用して MySQL Router に必要なユーザーアカウントを設定できます。 setupRouterAccount(user, [options]) 操作を使用して、MySQL ユーザーアカウントを作成するか、既存のアカウントをアップグレードし、MySQL Router で InnoDB クラスタ または InnoDB ReplicaSet の操作に使用できるようにします。 これは、InnoDB クラスタ および InnoDB ReplicaSet で MySQL Router を構成するための推奨方法です。

変数 testCluster によって参照される InnoDB クラスタ に myRouter1 という名前の新しい MySQL Router アカウントを追加するには、次のコマンドを発行します:

```
mysqlsh> testCluster.setupRouterAccount(myRouter1)
```

この場合、ドメインは指定されないため、アカウントはワイルドカード (%) 文字を使用して作成されるため、作成されたユーザーはどのドメインからでも接続できます。 example.com ドメインからのみ接続できるようにアカウントを制限するには、次のコマンドを発行します:

```
mysqlsh> testCluster.setupRouterAccount(myRouter1@example.com)
```

この操作では、パスワードの入力を求められ、適切な権限を持つ MySQL Router ユーザーが設定されます。 InnoDB クラスタ または InnoDB ReplicaSet に複数のインスタンスがある場合、作成された MySQL Router ユーザーはすべてのインスタンスに伝播されます。

MySQL Router ユーザーがすでに構成されている場合 (たとえば、8.0.20 より前のバージョンを使用していた場合)、setupRouterAccount() 操作を使用して既存のユーザーを再構成できます。 この場合は、true に設定された update オプションを渡します。 たとえば、myOldRouter ユーザーを再構成するには、次のように発行します:

```
mysqlsh> testCluster.setupRouterAccount(myOldRouter, {'update':1})
```

MySQL Router のデプロイ

MySQL Router の推奨デプロイメントは、アプリケーションと同じホスト上にあります。 サンドボックスデプロイメントを使用する場合、すべてが単一のホストで実行されるため、MySQL Router を同じホストにデプロイします。 本番デプロイメントを使用する場合は、クライアントアプリケーションのホストに使用する各マシンに MySQL Router インスタンスをデプロイすることをお薦めします。 アプリケーションインスタンスが接続する共通マシンに MySQL Router をデプロイすることもできます。 詳細は、Installing MySQL Routerを参照してください。

InnoDB クラスタ または InnoDB ReplicaSet に基づいて MySQL Router をブートストラップするには、オンラインインスタンスへの URI のような接続文字列が必要です。 mysqlrouter コマンドを実行し、--bootstrap=instance オプションを指定します。ここで、instance はオンラインインスタンスへの URI のような接続文字列です。MySQL Router はインスタンスに接続し、含まれているメタデータキャッシュプラグインを使用して、サーバーインスタンスアドレスとそのロールのリストで構成されるメタデータを取得します。 例:

```
shell> mysqlrouter --bootstrap icadmin@ic-1:3306 --user=mysqlrouter
```

MySQL Router で使用するインスタンスパスワードおよび暗号化キーの入力を求められます。 この暗号化キーは、MySQL Router がクラスタへの接続に使用するインスタンスパスワードを暗号化するために使用されます。 クライアント接続に使用できるポートも表示されます。 ブートストラップ関連のその他のオプションは、Bootstrapping Options を参照してください。

> **ヒント**
>
> この時点で、MySQL Router は接続をルーティングするために起動されていません。 ブートストラップは別のプロセスです。

MySQL Router ブートストラッププロセスでは、前述の icadmin@ic-1:3306 の例で、--bootstrap オプションに渡されたアドレスから取得されたメタデータに基づいて設定された mysqlrouter.conf ファイルが作成されます。 MySQL Router では、取得されたメタデータに基づいて、metadata_cache セクションを含む mysqlrouter.conf ファイルが自動的に構成されます。 MySQL Router 8.0.14 以降を使用している場合、--bootstrap オプションは、dynamic_state によって構成されたパスでアクティブな MySQL メタデータサーバーアドレスを追跡および格納するように MySQL Router を自動的に構成します。 これにより、MySQL Router の再起動時に、どの MySQL メタデータサーバーアドレスが最新であるかが確実に認識されます。 詳細は、dynamic_state のドキュメントを参照してください。

以前の MySQL Router バージョンでは、メタデータサーバー情報は MySQL Router の初期ブートストラップ操作中に定義され、クラスタ内のすべてのサーバーインスタンスのアドレスを含む bootstrap_server_addresses として構成ファイルに静的に格納されていました。 例:

```
[metadata_cache:prodCluster]
router_id=1
bootstrap_server_addresses=mysql://icadmin@ic-1:3306,mysql://icadmin@ic-2:3306,mysql://icadmin@ic-3:3306
user=mysql_router1_jy95yozko3k2
metadata_cluster=prodCluster
ttl=300
```

> **ヒント**
>
> MySQL Router 8.0.13 以前を使用している場合、MySQL Router のブートストラップ後に別のサーバーインスタンスを追加してクラスタのトポロジを変更するときは、更新されたメタデータに基づいて bootstrap_server_addresses を更新する必要があります。 --bootstrap オプションを使用して MySQL Router を再起動するか、mysqlrouter.conf ファイルの bootstrap_server_addresses セクションを手動で編集して MySQL Router を再起動します。

生成された MySQL Router 構成により、クラスタへの接続に使用する TCP ポートが作成されます。 デフォルトでは、クラシック MySQL プロトコル と X プロトコル の両方を使用してクラスタと通信するためのポートが作成されます。 X プロトコル を使用するには、サーバーインスタンスに X プラグイン がインストールおよび構成されている必要があります。これは、MySQL 8.0 以降のデフォルトです。 デフォルトで使用可能な TCP ポートは次のとおりです:

- 6446 - MySQL Router が受信接続をプライマリサーバーインスタンスにリダイレクトする クラシック MySQL プロトコル 読取り/書込みセッションの場合。
- 6447 - MySQL Router が受信接続をセカンダリサーバーインスタンスのいずれかにリダイレクトする クラシック MySQL プロトコル 読取り専用セッションの場合。
- 64460 - MySQL Router が受信接続をプライマリサーバーインスタンスにリダイレクトする X プロトコル 読取り/書込みセッションの場合。
- 64470 - MySQL Router が受信接続をセカンダリサーバーインスタンスのいずれかにリダイレクトする X プロトコル 読取り専用セッションの場合。

MySQL Router の構成によっては、ポート番号が前述のものと異なる場合があります。 たとえば、--conf-base-port オプションまたは group_replication_single_primary_mode 変数を使用する場合です。 MySQL Router を起動すると、正確なポートがリストされます。

着信接続のリダイレクト方法は、使用されている基礎となるトポロジによって異なります。 たとえば、単一プライマリクラスタを使用している場合、MySQL Router はデフォルトで X プロトコル および クラシック MySQL プロトコル ポートを公開します。これらのポートは、クライアントが読取り/書込みセッションのために接続し、クラスタの単一プライマリにリダイレクトされます。 マルチプライマリクラスタでは、読取り/書込みセッションはラウンドロビン方式でプライマリインスタンスのいずれかにリダイレクトされます。 たとえば、ポート 6446 への最初の接続は ic-1 インスタンスにリダイレクトされ、ポート 6446 への次の接続は ic-2 インスタンスにリダイレクトされます。 受信読取り専用接続の場合、MySQL Router は接続をセカンダリインスタンスのいずれかにラウンドロビン方式でリダイレクトします。 この動作を変更するには、routing_strategy オプションを参照してください。

ブートストラップして構成したら、MySQL Router を起動します。 --bootstrap オプションを指定してシステム全体のインストールを使用した場合は、次のコマンドを発行します:

```
shell> mysqlrouter &
```

--directory オプションを使用して MySQL Router をディレクトリにインストールした場合は、インストール先のディレクトリにある start.sh スクリプトを使用します。 または、システムのブート時に MySQL Router を自動的に起動す

るようにサービスを設定します。Starting MySQL Router を参照してください。 前述のように、MySQL Shell などの MySQL クライアントをいずれかの着信 MySQL Router ポートに接続し、クライアントがいずれかのサーバーインスタンスに透過的に接続される方法を確認できるようになりました。

```
shell> mysqlsh --uri root@localhost:6442
```

実際に接続しているインスタンスを確認するには、port ステータス変数に対して SQL クエリーを発行します。

```
mysql-js> \sql
Switching to SQL mode... Commands end with ;
mysql-sql> select @@port;
+--------+
| @@port |
+--------+
|   3310 |
+--------+
```

MySQL Router での ReplicaSets の使用

MySQL Router 8.0.19 以降を使用して InnoDB ReplicaSet に対してブートストラップできます。セクション 6.4「MySQL Router」 を参照してください。 生成される MySQL Router 構成ファイルの唯一の違いは、cluster_type オプションの追加です。 MySQL Router が ReplicaSet に対してブートストラップされると、生成される構成ファイルには次のものが含まれます:

```
cluster_type=rs
```

MySQL Router を InnoDB ReplicaSet とともに使用する場合は、次の点に注意してください:

- MySQL Router の読取り/書込みポートは、クライアント接続を ReplicaSet のプライマリインスタンスに転送
- ReplicaSet のセカンダリインスタンスへの MySQL Router ダイレクトクライアント接続の読取り専用ポート (ただし、プライマリに転送することもできます)
- MySQL Router は、プライマリインスタンスから ReplicaSet トポロジに関する情報を取得
- プライマリインスタンスが使用できなくなり、別のインスタンスがに昇格されると、MySQL Router は自動的にリカバリ

InnoDB クラスタ とまったく同じ方法で、ReplicaSet に対してブートストラップされた MySQL Router インスタンスを操作します。 ReplicaSet.listRouters() および ReplicaSet.removeRouterMetadata() の詳細は、クラスタルーターの操作 を参照してください。

6.4.2 AdminAPI および MySQL Router の使用

このセクションでは、MySQL Router および AdminAPI の使用方法について説明します。

InnoDB クラスタ の高可用性のテスト

InnoDB クラスタ の高可用性が機能するかどうかをテストするには、インスタンスを強制終了して予期しない停止をシミュレートします。 クラスタは、インスタンスがクラスタを離れたことを検出し、それ自体を再構成します。 クラスタ自体の再構成方法は、単一プライマリクラスタとマルチプライマリクラスタのどちらを使用しているか、およびインスタンスがクラスタ内で機能するロールによって正確に異なります。

シングルプライマリモードの場合:

- 現在のプライマリがクラスタから離れると、セカンダリインスタンスのいずれかが新しいプライマリとして選択され、インスタンスの優先順位は最も低い server_uuid になります。MySQL Router は、新しく選択されたプライマリに読取り/書込み接続をリダイレクトします。
- 現在のセカンダリがクラスタから離れると、MySQL Router はインスタンスへの読取り専用接続のリダイレクトを停止します。

詳細は、シングルプライマリモード を参照してください。

マルチプライマリモードの場合:

- 現在の「R/W」インスタンスがクラスタから離れると、MySQL Router は読取り/書込み接続を他のプライマリにリダイレクトします。 残りのインスタンスがクラスタ内の最後のプライマリであった場合、クラスタは完全に失われ、MySQL Router ポートに接続できません。

詳細は、マルチプライマリモード を参照してください。

クラスタから離れるインスタンスをシミュレートするには様々な方法があります。たとえば、インスタンス上の MySQL サーバーを強制的に停止したり、サンドボックスのデプロイメントをテストする場合は AdminAPI dba.killSandboxInstance() を使用できます。 この例では、3 つのサーバーインスタンスを持つ単一プライマリサンドボックスクラスタデプロイメントがあり、ポート 3310 でリスニングしているインスタンスが現在のプライマリであると想定しています。 予期せずにクラスタから離れるインスタンスをシミュレートします:

```
mysql-js> dba.killSandboxInstance(3310)
```

クラスタは変更を検出し、新しいプライマリを自動的に選択します。 セッションがデフォルトの読取り/書込み クラシック MySQL プロトコル ポートであるポート 6446 に接続されている場合、MySQL Router はクラスタトポロジへの変更を検出し、新しく選択されたプライマリにセッションをリダイレクトする必要があります。 これを確認するには、\sql コマンドを使用して MySQL Shell で SQL モードに切り替え、インスタンスの port 変数を選択して、セッションがリダイレクトされたインスタンスを確認します。 元のプライマリへの接続が失われたため、最初の SELECT ステートメントが失敗することに注意してください。 これは、現在のセッションがクローズされたことを意味し、MySQL Shell は自動的に再接続し、コマンドを再発行すると新しいポートが確認されます。

```
mysql-js> \sql
Switching to SQL mode... Commands end with ;
mysql-sql> SELECT @@port;
ERROR: 2013 (HY000): Lost connection to MySQL server during query
The global session got disconnected.
Attempting to reconnect to 'root@localhost:6446'...
The global session was successfully reconnected.
mysql-sql> SELECT @@port;
+--------+
| @@port |
+--------+
|   3330 |
+--------+
1 row in set (0.00 sec)
```

この例では、ポート 3330 のインスタンスが新しいプライマリとして選択されています。 これは、InnoDB クラスタが自動フェイルオーバーを提供し、MySQL Router が新しいプライマリインスタンスに自動的に再接続し、高可用性を備えていることを示しています。

クラスタルーターの操作

クラスタまたは ReplicaSet に対して MySQL Router の複数のインスタンスをブートストラップできます。 バージョン 8.0.19 から、登録されているすべての MySQL Router インスタンスのリストを表示するには、次のコマンドを発行します:

```
Cluster.listRouters()
```

結果には、メタデータ内の名前、ホスト名、ポートなど、登録されている各 MySQL Router インスタンスに関する情報が表示されます。 たとえば、次のように発行します:

```
mysql-js> Cluster.listRouters()
{
    "clusterName": "example",
    "routers": {
        "ic-1:3306": {
            "hostname": "ic-1:3306",
            "lastCheckIn": "2020-01-16 11:43:45",
            "roPort": 6447,
            "roXPort": 64470,
            "rwPort": 6446,
            "rwXPort": 64460,
            "version": "8.0.19"
        }
    }
}
```

返される情報は次のとおりです:

- MySQL Router インスタンスの名前。
- メタデータに格納されている MySQL Router から定期 ping によって生成される最終チェックインタイムスタンプ
- MySQL Router インスタンスが実行されているホスト名
- MySQL Router が クラシック MySQL プロトコル 接続用に公開する読取り専用および読取り/書込みポート
- MySQL Router が X プロトコル 接続用に公開する読取り専用および読取り/書込みポート
- この MySQL Router インスタンスのバージョン。 version を返すためのサポートが 8.0.19 で追加されました。 この操作を以前のバージョンの MySQL Router に対して実行する場合、バージョンフィールドは null です。

また、Cluster.listRouters() 操作では、MySQL Shell でサポートされているメタデータバージョンをサポートしていないインスタンスのリストを表示できます。 たとえば、Cluster.listRouters({'onlyUpgradeRequired':'true'}) を発行して、onlyUpgradeRequired オプションを使用します。 返されるリストには、メタデータのアップグレードが必要な Cluster に登録された MySQL Router インスタンスのみが表示されます。 セクション6.2.8.2「InnoDB クラスタ メタデータのアップグレード」を参照してください。

MySQL Router インスタンスはメタデータから自動的に削除されないため、たとえば、より多くのインスタンスをブートストラップすると、InnoDB クラスタ メタデータには、増加するインスタンスへの参照数が含まれます。 登録された MySQL Router インスタンスをクラスタメタデータから削除するには、バージョン 8.0.19 で追加された Cluster.removeRouterMetadata(router) 操作を使用します。 Cluster.listRouters() 操作を使用して、削除する MySQL Router インスタンスの名前を取得し、router として渡します。 たとえば、クラスタに登録された MySQL Router インスタンスが次のようになっているとします:

```
mysql-js> Cluster.listRouters(){

    "clusterName": "testCluster",
    "routers": {
        "myRouter1": {
            "hostname": "example1.com",
            "lastCheckIn": null,
            "routerId": "1",
            "roPort": "6447",
            "rwPort": "6446"
            "version": null
        },
        "myRouter2": {
            "hostname": "example2.com",
            "lastCheckIn": "2019-11-27 16:25:00",
            "routerId": "3",
            "roPort": "6447",
            "rwPort": "6446"
            "version": "8.0.19"
        }
    }
}
```

「myRouter1」 という名前のインスタンスに 「lastCheckIn」 および 「version」 用の null があるという事実に基づいて、次のコマンドを発行してメタデータからこの古いインスタンスを削除することにしました:

```
mysql-js> cluster.removeRouterMetadata('myRouter1')
```

指定した MySQL Router インスタンスは、InnoDB クラスタ メタデータから削除することでクラスタから登録解除されます。

6.5 AdminAPI MySQL サンドボックス

このセクションでは、AdminAPI でサンドボックスデプロイメントを設定する方法について説明します。 最初に MySQL のローカルサンドボックスインスタンスをデプロイおよび使用することは、AdminAPI の探索を開始するのに適した方法です。 本番サーバーにデプロイする前に、機能をローカルで完全にテストできます。AdminAPI には、ローカルにデプロイされたシナリオで InnoDB クラスタ および InnoDB ReplicaSet と連携するように正しく構成されたサンドボックスインスタンスを作成するための組込み機能があります。

> **重要**
>
> サンドボックスインスタンスは、テスト目的でローカルマシンでのデプロイおよび実行にのみ適しています。 本番環境では、MySQL Server インスタンスはネットワーク上の様々なホストマシンにデプロイされます。 詳しくはセクション6.2.2「本番 InnoDB クラスタ のデプロイ」をご覧ください。

インスタンスを操作して接続文字列で指定する本番デプロイメントとは異なり、サンドボックスインスタンスは MySQL Shell を実行しているマシンと同じマシンでローカルに実行されます。 したがって、サンドボックスインスタンスを指定するには、MySQL サンドボックスインスタンスがリスニングしているポート番号を指定します。

- サンドボックスインスタンスのデプロイ
- サンドボックスインスタンスの管理

サンドボックスインスタンスのデプロイ

MySQL AdminAPI では、dba グローバル変数が MySQL Shell に追加され、サンドボックスインスタンスを管理するための機能が提供されます。 この設定例では、dba.deploySandboxInstance(port_number) を使用して 3 つのサンドボックスインスタンスを作成します。 ポート 3310 にバインドされている新しいサンドボックスインスタンスをデプロイするには、次を発行します:

```
mysql-js> dba.deploySandboxInstance(3310)
```

deploySandboxInstance() に渡される引数は、MySQL Server インスタンスが接続をリスニングする TCP ポート番号です。 デフォルトでは、サンドボックスは Unix システムの $HOME/mysql-sandboxes/port という名前のディレクトリに作成されます。 Microsoft Windows システムの場合、ディレクトリは %userprofile%\MySQL\mysql-sandboxes\port です。

インスタンスの root ユーザーパスワードの入力を求められます。

> **重要**
>
> 各サンドボックスインスタンスはルートユーザーとパスワードを使用し、連携する必要があるすべてのサンドボックスインスタンスで同じである必要があります。 これは本番環境ではお薦めしません。

別のサンドボックスサーバーインスタンスをデプロイするには、ポート 3310 でサンドボックスインスタンスに対してステップを繰り返し、インスタンスごとに異なるポート番号を選択します。 追加のサンドボックスインスタンスごとに、次のようにします:

```
mysql-js> dba.deploySandboxInstance(port_number)
```

このチュートリアルに従うには、3 つのサンドボックスサーバーインスタンスにポート番号 3310、3320 および 3330 を使用します。 次のコマンドを発行します:

```
mysql-js> dba.deploySandboxInstance(3320)
mysql-js> dba.deploySandboxInstance(3330)
```

テスト目的で単一のホストで複数のサンドボックスを実行する場合など、サンドボックスが格納されているディレクトリを変更するには、MySQL Shell sandboxDir オプションを使用します。 たとえば、/home/user/sandbox1 ディレクトリでサンドボックスを使用するには、次のコマンドを発行します:

```
mysql-js> shell.options.sandboxDir='/home/user/sandbox1'
```

その後のサンドボックス関連のすべての操作は、/home/user/sandbox1 で見つかったインスタンスに対して実行されます。

サンドボックスをデプロイすると、MySQL Shell は mysqld バイナリを検索し、それを使用してサンドボックスインスタンスを作成します。 MySQL Shell が mysqld バイナリを検索する場所を構成するには、PATH 環境変数を構成します。 これは、本番環境にデプロイする前に、新しいバージョンの MySQL をローカルでテストする場合に役立ちます。 たとえば、パス/home/user/mysql-latest/bin/mysqld で mysqld バイナリを使用するには、次のようにします:

```
PATH=/home/user/mysql-latest/bin/mysqld:$PATH
```

次に、PATH 環境変数が設定されている端末から MySQL Shell を実行します。 デプロイするサンドボックスでは、構成されたパスにある mysqld バイナリが使用されます。

サンドボックスインスタンスの管理

サンドボックスインスタンスの実行後は、次を使用していつでもステータスを変更できます:

- サンドボックスインスタンスを停止するには、dba.stopSandboxInstance(instance) を使用します。 これにより、dba.killSandboxInstance(instance) とは異なり、インスタンスが正常に停止します。
- サンドボックスインスタンスを起動するには、dba.startSandboxInstance(instance) を使用します。
- サンドボックスインスタンスを強制終了するには、dba.killSandboxInstance(instance) を使用します。 これにより、インスタンスは正常に停止せずに停止され、予期しない停止のシミュレーションに役立ちます。
- サンドボックスインスタンスを削除するには、dba.deleteSandboxInstance(instance) を使用します。 これにより、サンドボックスインスタンスがファイルシステムから完全に削除されます。

第 7 章 MySQL Shell の拡張

目次

7.1 MySQL Shell でのレポート 119
7.1.1 MySQL Shell レポートの作成 120
7.1.2 MySQL Shell レポートの登録 120
7.1.3 MySQL Shell レポートの永続化 122
7.1.4 MySQL Shell レポートの例 122
7.1.5 MySQL Shell レポートの実行 122
7.1.6 組込み MySQL Shell レポート 124
7.2 MySQL Shell への拡張オブジェクトの追加 126
7.2.1 ユーザー定義 MySQL Shell グローバルオブジェクトの作成 126
7.2.2 拡張オブジェクトの作成 127
7.2.3 拡張オブジェクトの永続化 129
7.2.4 MySQL Shell 拡張オブジェクトの例 129
7.3 MySQL Shell プラグイン 130
7.3.1 MySQL Shell プラグインの作成 130
7.3.2 プラグイングループの作成 131
7.3.3 MySQL Shell プラグインの例 132

MySQL Shell の基本機能に対する拡張機能は、レポートおよび拡張オブジェクトの形式で定義できます。 レポートおよび拡張オブジェクトは、JavaScript または Python を使用して作成でき、アクティブな MySQL Shell 言語に関係なく使用できます。 レポートおよび拡張機能オブジェクトは、MySQL Shell の起動時に自動的にロードされるプラグインに保持できます。

- MySQL Shell レポートは、MySQL Shell 8.0.16 から入手できます。 セクション7.1「MySQL Shell でのレポート」を参照してください。
- 拡張オブジェクトは、MySQL Shell 8.0.17 から使用できます。 セクション7.2「MySQL Shell への拡張オブジェクトの追加」を参照してください。
- レポートおよび拡張オブジェクトは、MySQL Shell 8.0.17 から MySQL Shell プラグインとして格納できます。 セクション7.3「MySQL Shell プラグイン」を参照してください。

7.1 MySQL Shell でのレポート

MySQL Shell を使用すると、ステータスやパフォーマンス情報など、MySQL サーバーからのライブ情報を表示するレポートを設定および実行できます。MySQL Shell レポート機能では、組込みレポートとユーザー定義レポートの両方がサポートされます。 レポート機能は、MySQL Shell 8.0.16 から使用できます。 レポートは、MySQL Shell 対話型プロンプトで直接作成することも、MySQL Shell の起動時に自動的にロードされるスクリプトで定義することもできます。

レポートは、目的の出力を生成する操作を実行するプレーンな JavaScript または Python 関数です。 関数を MySQL Shell レポートとして登録するには、JavaScript の shell.registerReport() メソッドまたは Python の shell.register_report() メソッドを使用します。セクション7.1.1「MySQL Shell レポートの作成」 には、レポートを作成、登録および格納する手順があります。 レポートは、MySQL Shell プラグインの一部として格納できます (セクション7.3「MySQL Shell プラグイン」 を参照)。

サポートされている言語 (JavaScript、Python または SQL) で記述されたレポートは、アクティブな MySQL Shell 言語に関係なく実行できます。 レポートは、MySQL Shell \show コマンドを使用して一度実行するか、\watch コマンドを使用して MySQL Shell セッションで継続的に実行およびリフレッシュできます。 これらは、shell.reports オブジェクトを使用して API 関数としてアクセスすることもできます。セクション7.1.5「MySQL Shell レポートの実行」 では、これらの各方法でレポートを実行する方法について説明します。

MySQL Shell には、セクション7.1.6「組込み MySQL Shell レポート」 で説明されている多数の組込みレポートが含まれています。

7.1.1 MySQL Shell レポートの作成

サポートされているスクリプト言語 (JavaScript および Python) のいずれかで、MySQL Shell のユーザー定義レポートを作成および登録できます。 レポート機能は、同じ API フロントエンドスキームを使用して組込みレポートおよびユーザー定義レポートを処理します。

レポートでは、受け入れるレポート固有のオプションのリストを指定できます。また、指定した数の追加引数を受け入れることもできます。 レポートでは、これらの入力の両方をサポートすることも、いずれもサポートしないこともできます。 レポートのヘルプをリクエストすると、MySQL Shell では、オプションと引数のリスト、およびレポートの登録時に提供されるこれらの説明が提供されます。

署名

MySQL Shell レポートとして登録する Python または JavaScript 関数のシグネチャは、次のようにする必要があります:

```
Dict report(Session session, List argv, Dict options);
```

ここでは:

- session は、レポートの実行に使用される MySQL Shell セッションオブジェクトです。
- argv は、レポートに渡される追加の引数の文字列値を含むオプションのリストです。
- options は、レポート固有のオプションとその値に対応するキー名と値を持つオプションのディクショナリです。

レポートタイプ

レポート関数は、登録時に使用するタイプに応じて、特定の形式でデータを返すことが期待されます:

リストタイプ　出力をリストのリストとして返します。最初のリストはカラムの名前で構成され、残りは行のコンテンツです。MySQL Shell では、デフォルトで出力がテーブル形式で表示されるか、\show または\watch コマンドで --vertical または --E オプションが指定されている場合は垂直形式で表示されます。 行の値は、アイテムの文字列表現に変換されます。 行の要素数がカラム名の数より少ない場合、欠落している要素は NULL とみなされます。 行の要素数がカラム名の数より多い場合、余分な要素は無視されます。 このレポートを登録するときは、「list」 タイプを使用します。

レポートタイプ　単一アイテムを含むリストとしてフリーフォーム出力を返します。MySQL Shell では、YAML を使用してこの出力が表示されます。 このレポートを登録するときは、「report」 タイプを使用します。

印刷タイプ　出力を画面に直接出力し、出力がすでに表示されていることを示す空のリストを MySQL Shell に返します。 このレポートを登録するときは、「print」 タイプを使用します。

出力を提供するには、レポートの API 関数で、キー report を含むディクショナリと、返されるリスト内の各セクション目に対して 1 つずつ JSON オブジェクトのリストを返す必要があります。 リストタイプにはリストごとに 1 つの要素を使用し、レポートタイプには単一の要素を使用し、印刷タイプには要素を使用しません。

7.1.2 MySQL Shell レポートの登録

ユーザー定義レポートを MySQL Shell に登録するには、Python の JavaScript または shell.register_report() で shell.registerReport() メソッドをコールします。 メソッドの構文は次のとおりです:

```
shell.registerReport(name, type, report[, description])
```

ここでは:

- name は、レポートの一意の名前を示す文字列です。
- type は、「list」、「report」 または 「print」 のいずれかの出力形式を決定するレポートタイプを示す文字列です。
- report は、レポートの起動時にコールされる関数です。

- description は、レポートがサポートするオプション、レポートが受け入れる追加の引数、および MySQL Shell ヘルプシステムで提供されるヘルプ情報を指定するために使用できるオプションを含むディクショナリです。

name、type および report パラメータはすべて必須です。 レポート名は次の要件を満たしている必要があります:

- MySQL Shell インストールで一意である必要があります。
- 有効なスクリプト識別子である必要があるため、最初の文字は文字またはアンダースコア文字で、その後に任意の数の文字、数字またはアンダースコア文字が続く必要があります。
- 大文字と小文字を混在させることはできますが、小文字に変換する場合は、MySQL Shell インストールで一意である必要があります。

レポート名では、登録プロセス中や、\show および\watch コマンドを使用してレポートを実行する際に、大/小文字は区別されません。 shell.reports オブジェクトで対応する API 関数をコールする場合、レポート名では大/小文字が区別されます。 Python モードか JavaScript モードかにかかわらず、レポートの登録に使用された正確な名前を使用して関数をコールする必要があります。

オプションのディクショナリには次のキーが含まれており、これらはすべてオプションです:

brief　　レポートの簡単な説明。

details　　文字列の配列として提供されるレポートの詳細な説明。 これは、\show コマンドで\help コマンドまたは --help オプションを使用する場合に提供されます。

options　　レポートが受け入れることができるレポート固有のオプション。 配列内の各ディクショナリには 1 つのオプションが記述されており、次のキーが含まれている必要があります:

- name (string、必須): 長い形式のオプションの名前。有効なスクリプト識別子である必要があります。
- brief (string、オプション): オプションの簡単な説明。
- shortcut (string、オプション): オプションの代替名 (単一の英数字)。
- details (文字列の配列、オプション): オプションの詳細な説明。 これは、\show コマンドで\help コマンドまたは --help オプションを使用する場合に提供されます。
- type (string、オプション): オプションの値タイプ。 許可される値は「string」、「bool」、「integer」 および 「float」 で、type が指定されていない場合、デフォルトは 「string」 です。 「bool」 が指定されている場合、このオプションはスイッチとして機能: 指定しない場合、false にデフォルト設定され、\show または\watch コマンドを使用してレポートを実行する場合は true にデフォルト設定され (値は受け入れられません)、shell.reports オブジェクトを使用してレポートを実行する場合は有効な値が必要です。
- required (bool、オプション): オプションが必要かどうか。 required が指定されていない場合、デフォルトで false に設定されます。 オプションタイプが 「bool」 の場合、required は true にできません。
- values (文字列の配列、オプション): オプションに使用できる値のリスト。 このキーを持つことができるのは、「string」 タイプのオプションのみです。 values が指定されていない場合、このオプションは任意の値を受け入れます。

argc　　レポートで想定される追加の引数の数を指定する文字列。次のいずれかを指定できます:

- 単一の数値として指定される引数の正確な数。
- アスタリスクとして指定されたゼロ個以上の引数。
- 引数番号の範囲。ダッシュで区切られた 2 つの数値として指定されます (例: 「1-5」)。
- 最小値で最大値が指定されていない引数番号の範囲。数値とアスタリスクで区切ります (「1-*」 など)。

7.1.3 MySQL Shell レポートの永続化

MySQL Shell レポートは、レポートに使用されるスクリプト言語と一致するように、JavaScript コードの場合は .js のファイル拡張子、Python コードの場合は .py のファイル拡張子で保存する必要があります。 ファイル拡張子は大/小文字が区別されません。

レポートを永続化するには、MySQL Shell プラグインにレポートを追加することをお薦めします。 プラグインおよびプラグイングループは、MySQL Shell の起動時に自動的にロードされ、それらが定義および登録する関数はすぐに使用可能になります。 MySQL Shell プラグインでは、初期化スクリプトを含むファイルの名前は、言語に応じて init.js または init.py である必要があります。 MySQL Shell プラグインの使用手順については、セクション7.3「MySQL Shell プラグイン」 を参照してください。

かわりに、レポートを含むスクリプトを MySQL Shell ユーザー構成パスの init.d フォルダに直接格納することもできます。 MySQL Shell が起動すると、.js または .py ファイル拡張子を持つ init.d フォルダにあるすべてのファイルが自動的に処理され、それらのファイル内の機能が使用可能になります。 (この場所では、ファイル名は MySQL Shell には関係ありません。) デフォルトの MySQL Shell ユーザー構成パスは、Unix では ~/.mysqlsh/、Windows では %AppData%\MySQL\mysqlsh\です。 ユーザー構成パスは、環境変数 MYSQLSH_USER_CONFIG_HOME を定義することで、すべてのプラットフォームでオーバーライドできます。

7.1.4 MySQL Shell レポートの例

このユーザー定義レポートの例 sessions は、現在存在するセッションを示しています。

```
def sessions(session, args, options):
    sys = session.get_schema('sys')
    session_view = sys.get_table('session')
    query = session_view.select(
        'thd_id', 'conn_id', 'user', 'db', 'current_statement',
        'statement_latency AS latency', 'current_memory AS memory')
    if (options.has_key('limit')):
        limit = int(options['limit'])
        query.limit(limit)

    result = query.execute()
    report = [result.get_column_names()]
    for row in result.fetch_all():
        report.append(list(row))

    return {'report': report}

shell.register_report(
    'sessions',
    'list',
    sessions,
    {
        'brief': 'Shows which sessions exist.',
        'details': ['You need the SELECT privilege on sys.session view and the underlying tables and functions used by it.'],
        'options': [
            {
                'name': 'limit',
                'brief': 'The maximum number of rows to return.',
                'shortcut': 'l',
                'type': 'integer'
            }
        ],
        'argc': '0'
    }
)
```

7.1.5 MySQL Shell レポートの実行

MySQL Shell に登録されている組込みレポートおよびユーザー定義レポートは、\show または\watch コマンドを使用して任意の対話型 MySQL Shell モード (JavaScript、Python または SQL) で実行するか、JavaScript または Python スクリプトの shell.reports オブジェクトを使用してコールできます。 パラメータを指定せずに\show コマンドまたは\watch コマンドを実行すると、使用可能な組込みレポートおよびユーザー定義レポートがすべてリストされます。

Show および Watch コマンドの使用

\show および\watch コマンドを使用するには、アクティブな MySQL セッションが使用可能である必要があります。

\show コマンドは、組込み MySQL Shell レポートまたは MySQL Shell に登録されているユーザー定義レポートのいずれかの名前付きレポートを実行します。 レポートでサポートされているオプションまたは追加の引数を指定できます。 たとえば、次のコマンドは組込みレポート query を実行します。このレポートは、引数として単一の SQL ステートメントを取ります:

```
\show query show session status
```

レポート名では大文字と小文字は区別されず、ダッシュとアンダースコアは同じものとして扱われます。

\show コマンドには、次の標準オプションもあります:

- --vertical (または -E) では、テーブル形式ではなく垂直形式でリストを返すレポートの結果が表示されます。
- --help では、指定したレポートに提供されているヘルプが表示されます。 (または、レポート機能のヘルプを表示するレポートの名前を指定して\help コマンドを使用することもできます。)

標準オプションおよびレポート固有のオプションは、引数の前に指定します。 たとえば、次のコマンドは組込みレポート query を実行し、結果を垂直形式で返します:

```
\show query --vertical show session status
```

\watch コマンドは、\show コマンドと同じ方法でレポートを実行しますが、Ctrl + C を使用してコマンドを取り消すまで定期的に結果をリフレッシュします。 \watch コマンドには、次のようにリフレッシュ動作を制御するための追加の標準オプションがあります:

- --interval=float (または -i float) は、リフレッシュの間に待機する秒数を指定します。 デフォルトは 2 秒です。 小数秒を 0.1 秒の最小間隔で指定でき、間隔は 86400 秒 (24 時間) まで設定できます。

  --nocls では、リフレッシュ前に画面がクリアされないように指定されているため、以前の結果は引き続き表示されます。

たとえば、次のコマンドは組込みレポート query を使用してステートメントカウンタ変数を表示し、0.5 秒ごとに結果をリフレッシュします:

```
\watch query --interval=0.5 show global status like 'Com%'
```

引用符はサーバーによって直接ではなくコマンドハンドラによって解釈されるため、クエリーで使用される場合は、引用符の前にバックスラッシュ (\) を付けてエスケープする必要があります。

shell.reports オブジェクトの使用

MySQL Shell に登録されている組込み MySQL Shell レポートおよびユーザー定義レポートには、shell.reports オブジェクトの API 関数としてアクセスすることもできます。 shell.reports オブジェクトは、JavaScript および Python モードで使用でき、登録時に指定されたレポート名を関数名として使用します。 この関数のシグネチャは次のとおりです:

```
Dict report(Session session, List argv, Dict options);
```

ここでは:

- session は、レポートの実行に使用される MySQL Shell セッションオブジェクトです。
- argv は、レポートに渡される追加の引数の文字列値を含むリストです。
- options は、レポート固有のオプションとその値に対応するキー名と値を含むディクショナリです。 短い形式のオプションは、shell.reports オブジェクトでは使用できません。

戻り値は、キー report を含むディクショナリと、レポートを含む JSON オブジェクトのリストです。 レポートのリストタイプには各リストの要素があり、レポートタイプには単一の要素があり、印刷タイプには要素がありません。

shell.reports オブジェクトでは、オプションのディクショナリが存在する場合、追加の引数がない場合でも argv リストが必要です。 \help report_name コマンドを使用して、レポート機能のヘルプを表示し、レポートに引数またはオプションが必要かどうかを確認します。

たとえば、次のコードは、現在存在するセッションを示す sessions という名前のユーザー定義レポートを実行します。 レポートを実行するための MySQL Shell セッションオブジェクトが作成されます。 レポート固有のオプションを使用して、返される行数を 10 に制限します。 追加の引数はないため、argv リストは存在しますが空です。

```
report = shell.reports.sessions(shell.getSession(), [], {'limit':10});
```

7.1.6 組込み MySQL Shell レポート

MySQL Shell には、次の情報を表示する組込みレポートが含まれています:

- 指定した SQL クエリーの結果 (MySQL Shell 8.0.16 から入手可能な query)。
- 接続された MySQL サーバー (MySQL Shell 8.0.18 から入手可能な threads) 内の現在のスレッドのリスト。
- 指定したスレッド (MySQL Shell 8.0.18 から入手可能な thread) に関する詳細情報。

ユーザー定義レポートと同様に、組込みレポートは、MySQL Shell \show コマンドを使用して一度実行するか、\watch コマンドを使用して MySQL Shell セッションで継続的に実行およびリフレッシュできます。 組込みレポートでは、説明に特に記載がないかぎり、レポート固有のオプションに加えて、\show および\watch コマンドの標準オプションがサポートされます。 これらは、shell.reports オブジェクトを使用して API 関数としてアクセスすることもできます。セクション7.1.5「MySQL Shell レポートの実行」 では、これらの各方法でレポートを実行する方法について説明します。

7.1.6.1 組込み MySQL Shell レポート: クエリー

組込み MySQL Shell レポート query は、MySQL Shell 8.0.16 から使用できます。 引数として指定された単一の SQL ステートメントを実行し、MySQL Shell レポート機能を使用して結果を返します。 query レポートは、すぐに使用できるように単純なレポートを生成する便利な方法として使用できます。

query レポートにはレポート固有のオプションはありませんが、セクション7.1.5「MySQL Shell レポートの実行」 で説明されているように、\show および\watch コマンドの標準オプションを使用できます。

たとえば、次のコマンドでは、query レポートを使用してステートメントカウンタ変数を表示し、0.5 秒ごとに結果をリフレッシュします:

```
\watch query --interval=0.5 show global status like 'Com%'
```

7.1.6.2 組込み MySQL Shell レポート: Threads

組込み MySQL Shell レポート threads は、MySQL Shell 8.0.18 から使用できます。 レポートの実行に使用されるユーザーアカウントに属する、接続された MySQL サーバー内の現在のスレッドがリストされます。 このレポートは、サポートされているすべての MySQL 5.7 および MySQL 8.0 バージョンを実行しているサーバーで機能します。 ターゲットサーバーの MySQL Server バージョンで使用できない情報項目がある場合、レポートはその情報を残します。

threads レポートには、MySQL パフォーマンススキーマを含む様々なソースから取得された各スレッドの情報が表示されます。 レポート固有のオプションを使用すると、フォアグラウンドスレッド、バックグラウンドスレッドまたはすべてのスレッドの表示を選択できます。 スレッドごとにデフォルトの情報セットをレポートすることも、使用可能な多数の選択肢からレポートに含める特定の情報を選択することもできます。 出力をフィルタ、ソートおよび制限できます。 レポート固有のオプションおよびレポートに含めることができる情報の完全なリストの詳細は、次の MySQL Shell コマンドのいずれかを発行してレポートのヘルプを表示します:

```
\help threads
\show threads --help
```

セクション7.1.5「MySQL Shell レポートの実行」 で説明されているように、threads レポートでは、レポート固有のオプションに加えて、\show および\watch コマンドの標準オプションも使用できます。 threads レポートはリストタイプであり、デフォルトでは結果はテーブルとして返されますが、--vertical (または -E) オプションを使用して垂直形式で表示できます。

threads レポートでは、MySQL Server format_statement() 関数が使用されます (format_statement() 関数 を参照)。 レポートに表示される切り捨てられたステートメントは、MySQL Server sys_config テーブルの statement_truncate_len オプションの設定 (デフォルトは 64 文字) に従って切り捨てられます。

次のリストは、threads レポートのレポート固有のオプションで提供される機能をまとめたものです。 オプションの詳細および短い形式は、レポートのヘルプを参照してください:

--foreground, --background, --all
: フォアグラウンドスレッドのみ、バックグラウンドスレッドのみ、またはすべてのスレッドをリストします。 かわりに --format オプションを使用して独自のフィールドの選択を指定しないかぎり、レポートにはスレッドタイプの選択に適したフィールドのデフォルトセットが表示されます。

--format
: カラム (および必要に応じて表示名) のカンマ区切りリストとして指定して、スレッドごとに表示する独自のカスタム情報セットを定義します。 レポートのヘルプには、レポートのカスタマイズに含めることができるすべてのカラムがリストされます。

--where, --order-by, --desc, --limit
: 論理式 (--where) を使用して返される結果をフィルタするか、選択したカラム (--order-by) でソートするか、昇順の --desc ではなく降順でソートするか、返されるスレッドの数 (--limit) を制限します。

たとえば、次のコマンドは、threads レポートを実行して、スレッド ID、生成スレッドの ID、接続 ID、ユーザー名とホスト名、クライアントプログラム名、スレッドが実行しているコマンドのタイプ、およびスレッドによって割り当てられたメモリーで構成される情報のカスタムセットとともに、すべてのフォアグラウンドスレッドを表示します:

```
mysql-js> \show threads --foreground -o tid,ptid,cid,user,host,progname,command,memory
```

7.1.6.3 組込み MySQL Shell レポート: Thread

組込み MySQL Shell レポート thread は、MySQL Shell 8.0.18 から使用できます。 接続された MySQL サーバー内の特定のスレッドに関する詳細情報を提供します。 このレポートは、サポートされているすべての MySQL 5.7 および MySQL 8.0 バージョンを実行しているサーバーで機能します。 ターゲットサーバーの MySQL Server バージョンで使用できない情報項目がある場合、レポートはその情報を残します。

thread レポートには、MySQL パフォーマンススキーマを含む様々なソースから導出された、選択したスレッドとそのアクティビティの情報が表示されます。 デフォルトでは、レポートには現在の接続で使用されているスレッドに関する情報が表示されます。または、スレッドをその ID または接続 ID で識別できます。 1 つ以上のカテゴリの情報を選択するか、スレッドに関する使用可能なすべての情報を表示できます。 レポート固有のオプションおよびレポートに含めることができる情報の詳細は、次の MySQL Shell コマンドのいずれかを発行してレポートのヘルプを表示します:

```
\help thread
\show thread --help
```

セクション7.1.5「MySQL Shell レポートの実行」 で説明されているように、thread レポートでは、レポート固有のオプションに加えて、\show および\watch コマンドのほとんどの標準オプションを使用できます。 例外は、\show コマンドの --vertical (または -E) オプションであり、受け入れられません。 thread レポートには、異なるセクションに表示される垂直リストおよびテーブルを含むカスタム出力形式があり、この出力形式は変更できません。

threads レポートでは、MySQL Server format_statement() 関数が使用されます (format_statement() 関数 を参照)。 レポートに表示される切り捨てられたステートメントは、MySQL Server sys_config テーブルの statement_truncate_len オプションの設定 (デフォルトは 64 文字) に従って切り捨てられます。

次のリストは、threads レポートのレポート固有のオプションで提供される機能をまとめたものです。 オプションの詳細および短い形式は、レポートのヘルプを参照してください:

--tid, --cid
: レポートするスレッド ID または接続 ID を指定します。

--general
: スレッドに関する基本情報を表示します。 この情報は、次のいずれのオプションも使用しない場合、デフォルトで返されます。

--brief
: スレッドの簡単な説明を 1 行に表示します。

--client
: クライアント接続およびクライアントセッションに関する情報を表示します。

--innodb
: スレッドを使用している現在の InnoDB トランザクションに関する情報を表示します (存在する場合)。

--locks
: スレッドによってブロックおよびブロックされたロックに関する情報を表示します。

--prep-stmts	スレッドに割り当てられたプリペアドステートメントに関する情報を表示します。
--status	スレッドのセッションステータス変数に関する情報を表示します。 照合する接頭辞のリストを指定できます。この場合、一致する変数のみが表示されます。
--vars	スレッドのセッションシステム変数に関する情報を表示します。 照合する接頭辞のリストを指定できます。この場合、一致する変数のみが表示されます。
--user-vars	スレッドのユーザー定義変数に関する情報を表示します。 照合する接頭辞のリストを指定できます。この場合、一致する変数のみが表示されます。
--all	簡単な説明を除いて、前述のすべての情報を表示します。

たとえば、次のコマンドはスレッド ID 53 のスレッドに対して thread レポートを実行し、スレッドに関する一般情報、クライアント接続の詳細、およびスレッドがブロックしているロックまたはブロックしているロックに関する情報を返します:

```
mysql-py>  \show thread --tid 53 --general --client --locks
```

7.2 MySQL Shell への拡張オブジェクトの追加

MySQL Shell 8.0.17 から、拡張オブジェクトを定義し、ユーザー定義の MySQL Shell グローバルオブジェクトの一部として使用できるようにすることができます。 拡張オブジェクトを作成して登録すると、JavaScript モードと Python モードの両方で使用できます。

拡張オブジェクトは、1 つ以上のメンバーで構成されます。 メンバーには、基本データ型の値、ネイティブの JavaScript または Python で記述された関数、または別の拡張オブジェクトを指定できます。 組込みグローバルオブジェクト shell で提供される関数を使用して、拡張オブジェクトを構築および登録します。 MySQL Shell に登録した後、オブジェクトにメンバーを追加することで、オブジェクトの拡張を続行できます。

> 注記
>
> 関数を含む拡張オブジェクトを MySQL Shell グローバルオブジェクトとして直接登録できます。 ただし、拡張オブジェクトを適切に管理するには、すべての拡張オブジェクトのエントリポイントとして機能する、または少数のトップレベル拡張オブジェクトを作成し、これらのトップレベル拡張オブジェクトを MySQL Shell グローバルオブジェクトとして登録すると便利です。 その後、現在および将来の拡張オブジェクトを、適切なトップレベル拡張オブジェクトのメンバーとして追加できます。 この構造では、MySQL Shell グローバルオブジェクトとして登録された最上位の拡張オブジェクトによって、開発者は様々なタイミングで作成され、様々な MySQL Shell プラグインに格納された様々な拡張オブジェクトを追加できます。

7.2.1 ユーザー定義 MySQL Shell グローバルオブジェクトの作成

拡張オブジェクトのエントリポイントとして機能する新しい MySQL Shell グローバルオブジェクトを作成するには、まず Python の JavaScript または shell.create_extension_object() の組込み shell.createExtensionObject() 関数を使用して、新しいトップレベル拡張オブジェクトを作成します:

```
shell.createExtensionObject()
```

次に、Python の JavaScript または shell.register_global() で shell.registerGlobal() メソッドをコールして、この最上位の拡張オブジェクトを MySQL Shell グローバルオブジェクトとして登録します。 メソッドの構文は次のとおりです:

```
shell.registerGlobal(name, object[, definition])
```

ここでは:

- name は、グローバルオブジェクトの名前 (およびクラス) を示す文字列です。 名前は有効なスクリプト識別子である必要があるため、最初の文字は文字またはアンダースコア文字で、その後に任意の数の文字、数字またはアンダースコア文字が続く必要があります。 この名前は MySQL Shell インストール内で一意である必要があるため、組込み MySQL Shell グローバルオブジェクト (db, dba, cluster, session, shell, util など) の名前にすることはできず、ユーザー定義 MySQL Shell グローバルオブジェクトにすでに使用されている名前にすることもできません。 次の例は、グローバルオブジェクトを登録する前に名前がすでに存在するかどうかを確認する方法を示しています。

> **重要**
>
> グローバルオブジェクトの登録に使用する名前は、JavaScript モードと Python モードの両方でオブジェクトにアクセスするときにそのまま使用されます。 したがって、グローバルオブジェクト (ext など) には単純な一言の名前を使用することをお薦めします。 キャメルケースまたはスネークケース (myCustomObject など) で複合名でグローバルオブジェクトを登録する場合は、グローバルオブジェクトを使用するときに、登録された名前を指定する必要があります。 メンバーに使用される名前のみが、言語に適した方法で処理されます。

- object は、MySQL Shell グローバルオブジェクトとして登録する拡張オブジェクトです。 拡張オブジェクトは一度のみ登録できます。
- definition は、MySQL Shell ヘルプシステムで提供されるグローバルオブジェクトのヘルプ情報を含むオプションのディクショナリです。 ディクショナリには、次のキーが含まれます:
  - brief (string、オプション): ヘルプ情報として提供されるグローバルオブジェクトの簡単な説明。
  - details (文字列のリスト、オプション): ヘルプ情報として提供されるグローバルオブジェクトの詳細な説明。

7.2.2 拡張オブジェクトの作成

新しい拡張オブジェクトを作成して、1 つ以上の関数、データ型またはその他の拡張オブジェクトを提供するには、Python の JavaScript または shell.create_extension_object() で組込み shell.createExtensionObject() 関数を使用します:

```
shell.createExtensionObject()
```

拡張オブジェクトにメンバーを追加するには、Python の JavaScript または shell.add_extension_object_member() で組込み shell.addExtensionObjectMember() 関数を使用します:

```
shell.addExtensionObjectMember(object, name, member[, definition])
```

ここでは:

- object は、新しいメンバーが追加される拡張オブジェクトです。
- name は、新しいメンバーの名前です。 名前は有効なスクリプト識別子である必要があるため、最初の文字は文字またはアンダースコア文字で、その後に任意の数の文字、数字またはアンダースコア文字が続く必要があります。名前は、同じ拡張オブジェクトにすでに追加されているメンバー間で一意である必要があり、メンバーが関数の場合、名前は定義された関数の名前と一致する必要はありません。 Python を使用してメンバーを定義および追加する場合でも、名前はキャメルケースで指定することをお薦めします。 キャメルケースでメンバー名を指定すると、MySQL Shell でネーミング規則を自動的に適用できます。MySQL Shell では、メンバーはキャメルケースを使用して JavaScript モードで使用可能になり、スネークケースを使用して Python モードで使用可能になります。
- member は新しいメンバーの値で、次のいずれかを指定できます:
  - サポートされている基本データ型。 サポートされているデータ型は、「none」 または 「null」、「bool」、「number」(整数または浮動小数点)、「string」、「array」 および「dictionary」です。
  - JavaScript または Python 関数。 インタフェース (パラメータおよび戻り値) が 表7.1「拡張オブジェクトでサポートされているデータ型のペア」 でサポートされているデータ型に制限されている場合は、拡張オブジェクトにメンバーとして追加される関数本体でネイティブコードを使用できます。 インタフェースで他のデータ型を使用すると、動作が未定義になる可能性があります。
  - 別の拡張オブジェクト。
- definition はオプションのディクショナリで、メンバーのヘルプ情報を含めることができます。また、メンバーが関数の場合は、関数が受け取るパラメータのリストです。 ヘルプ情報は、次の属性を使用して定義されます:
  - brief は、メンバーの簡単な説明です。
  - details は、文字列のリストとして提供されるメンバーの詳細な説明です。 これは、MySQL Shell \help コマンドを使用する場合に提供されます。

関数のパラメータは、次の属性を使用して定義されます:

- parameters は、関数が受け取る各パラメータを記述するディクショナリのリストです。 各ディクショナリには 1 つのパラメータが記述され、次のキーを含めることができます:
  - name (string、必須): パラメータの名前。
  - type (string、必須): パラメータのデータ型。「string」、「integer」、「bool」、「float」、「array」、「dictionary」または「object」のいずれかです。 タイプが「object」の場合は、class または classes キーも使用できます。 タイプが「string」の場合は、values キーも使用できます。 タイプが「dictionary」の場合は、options キーも使用できます。
  - class (文字列、オプション、データ型が「object」の場合に使用可能): パラメータとして許可されるオブジェクトタイプを定義します。
  - classes (文字列のリスト、オプション、データ型が「object」の場合に使用可能): パラメータとして許可されるオブジェクトタイプを定義するクラスのリスト。 class および classes でサポートされているオブジェクト型は、Session, ClassicSession, Table や Collection などの MySQL Shell API によって公開されるオブジェクト型です。 このリストにない関数にオブジェクト型が渡されると、エラーが発生します。
  - values (文字列のリスト、オプション、データ型が「string」の場合に使用可能): パラメータに有効な値のリスト。 このリストにない関数に値が渡されると、エラーが発生します。
  - options (オプションのリスト、オプション、データ型が「dictionary」の場合に使用可能): パラメータに許可されているオプションのリスト。 オプションではパラメータと同じ定義構造が使用されますが、オプションに required が指定されていない場合は、デフォルトで false に設定される点が異なります。MySQL Shell では、エンドユーザーが指定したオプションが検証され、このリストにない関数にオプションが渡されるとエラーが発生します。 8.0.19 を介した MySQL Shell 8.0.17 では、データ型が「dictionary」の場合、このパラメータは必須ですが、MySQL Shell 8.0.20 からはオプションです。 オプションのリストを指定せずにディクショナリを作成した場合、エンドユーザーがディクショナリに対して指定したオプションは、検証なしで MySQL Shell によって関数に直接渡されます。
  - required (bool、オプション): パラメータが必須かどうか。 パラメータに required が指定されていない場合は、デフォルトで true に設定されます。
  - brief (string、オプション): ヘルプ情報として提供されるパラメータの簡単な説明。
  - details (文字列のリスト、オプション): ヘルプ情報として提供されるパラメータの詳細な説明。

拡張オブジェクトは、MySQL Shell グローバルオブジェクトとして登録されるか、MySQL Shell グローバルオブジェクトとして登録される別の拡張オブジェクトにメンバーとして追加されるまで、構成中とみなされます。 まだ登録されていない拡張オブジェクトを MySQL Shell で使用しようとすると、エラーが返されます。

クロス言語に関する考慮事項

拡張オブジェクトには、Python で定義されたメンバーと JavaScript で定義されたメンバーを混在させることができます。MySQL Shell は、一方の言語から他方の言語へのデータの転送をパラメータおよび戻り値として管理します。表 7.1「拡張オブジェクトでサポートされているデータ型のペア」 には、言語間でデータを転送する際に MySQL Shell でサポートされるデータ型と、相互の表現として使用されるペアが表示されます:

表 7.1 拡張オブジェクトでサポートされているデータ型のペア

JavaScript	Python
Boolean	Boolean
文字列	文字列
Integer	Long
数値	Float
Null	なし
Array	リスト

JavaScript	Python
マップ	辞書

拡張オブジェクトは、両方の言語で文字どおり同じオブジェクトです。

7.2.3 拡張オブジェクトの永続化

拡張オブジェクトを定義および登録するスクリプトには、スクリプトに使用される言語と一致するように、JavaScript コードの場合は .js のファイル拡張子、Python コードの場合は .py のファイル拡張子が必要です。 ファイル拡張子は大/小文字が区別されません。

拡張オブジェクトを永続化するには、それを MySQL Shell プラグインに追加することをお薦めします。 プラグインおよびプラグイングループは、MySQL Shell の起動時に自動的にロードされ、それらが定義および登録する関数はすぐに使用可能になります。 MySQL Shell プラグインでは、初期化スクリプトを含むファイルの名前は、言語に応じて init.js または init.py である必要があります。 プラグインには 1 つの言語のコードのみを含めることができるため、Python で定義されたメンバーと JavaScript で定義されたメンバーが混在する拡張オブジェクトを作成する場合は、メンバーを個別の言語に適したプラグインとして格納する必要があります。 MySQL Shell プラグインの使用手順については、セクション7.3「MySQL Shell プラグイン」 を参照してください。

かわりに、拡張オブジェクトを含むスクリプトを MySQL Shell ユーザー構成パスの init.d フォルダに直接格納できます。 MySQL Shell が起動すると、.js または .py ファイル拡張子を持つ init.d フォルダにあるすべてのファイルが自動的に処理され、登録した関数が使用可能になります。 (この場所では、ファイル名は MySQL Shell には関係ありません。) デフォルトの MySQL Shell ユーザー構成パスは、Unix では ~/.mysqlsh/、Windows では %AppData%\MySQL\mysqlsh\です。 ユーザー構成パスは、環境変数 MYSQLSH_USER_CONFIG_HOME を定義することで、すべてのプラットフォームでオーバーライドできます。

7.2.4 MySQL Shell 拡張オブジェクトの例

例 7.1 拡張オブジェクトの作成および登録 - Python

この例では、ユーザー定義の MySQL Shell グローバルオブジェクト demo を介して使用可能にする関数 hello_world() を作成します。 このコードは、新しい拡張オブジェクトを作成し、そのオブジェクトに hello_world() 関数をメンバーとして追加してから、その拡張オブジェクトを MySQL Shell グローバルオブジェクト demo として登録します。

```
# Define a hello_world function that will be exposed by the global object 'demo'
def hello_world():
    print("Hello world!")

# Create an extension object where the hello_world function will be registered
plugin_obj = shell.create_extension_object()

shell.add_extension_object_member(plugin_obj, "helloWorld", hello_world,
                                  {"brief": "Prints 'Hello world!'", "parameters": []})

# Registering the 'demo' global object
shell.register_global("demo", plugin_obj,
                      {"brief": "A demo plugin that showcases MySQL Shell's plugin feature."})
```

メンバー名は、shell.add_extension_object_member() 関数のキャメルケースで指定されることに注意してください。 Python モードでメンバーをコールする場合、メンバー名にスネークケースを使用すると、MySQL Shell によって変換が自動的に処理されます。 JavaScript モードでは、関数は次のようにコールされます:

```
mysql-js> demo.helloWorld()
```

Python モードでは、関数は次のようにコールされます:

```
mysql-py> demo.hello_world()
```

例 7.2 拡張オブジェクトの作成および登録 - JavaScript

この例では、関数 listTables() をメンバーとして使用して拡張オブジェクトを作成し、MySQL Shell グローバルオブジェクト tools として直接登録します:

```
// Define a listTables function that will be exposed by the global object tools

function listTables(session, schemaName, options) {
...
```

```
}

// Create an extension object and add the listTables function to it as a member

var object = shell.createExtensionObject()

shell.addExtensionObjectMember(object, "listTables", listTables,

                {
                  brief:"Retrieves the tables from a given schema.",
                  details: ["Retrieves the tables of the schema named schemaName.",
                            "If excludeCollections is true, the collection tables will not be returned"],
                  parameters:
                  [
                    {
                      name: "session",
                      type: "object",
                      class: "Session",
                      brief: "An X Protocol session object."
                    },
                    {
                      name: "schemaName",
                      type: "string",
                      brief: "The name of the schema from which the table list will be pulled."
                    },
                    {
                      name: "options",
                      type: "dictionary",
                      brief: "Additional options that affect the function behavior.",
                      options: [
                        {
                          name: "excludeViews",
                          type: "bool",
                          brief: "If set to true, the views will not be included on the list, default is false",
                        },
                        {
                          name: "excludeCollections",
                          type: "bool",
                          brief: "If set to true, the collections will not be included on the list, default is false",
                        }
                      ]
                    },
                  ]
                });


// Register the extension object as the global object "tools"

shell.registerGlobal("tools", object, {brief:"Global object for ExampleCom administrator tools",
                details:[
                  "Global object to access homegrown ExampleCom administrator tools.",
                  "Add new tools to this global object as members with shell.addExtensionObjectMember()."]})
```

JavaScript モードでは、関数は次のようにコールされます:

```
mysql-js> tools.listTables(session, "world_x", {excludeViews: true})
```

Python モードでは、関数は次のようにコールされます:

```
mysql-py> tools.list_tables(session, "world_x", {"excludeViews": True})
```

7.3 MySQL Shell プラグイン

MySQL Shell 8.0.17 から、起動時にロードされるユーザー定義プラグインを使用して MySQL Shell を拡張できます。プラグインは JavaScript または Python のいずれかで記述でき、プラグインに含まれる関数は JavaScript モードと Python モードの両方で MySQL Shell で使用できます。

7.3.1 MySQL Shell プラグインの作成

MySQL Shell プラグインを使用すると、MySQL Shell レポートとして登録される関数 (セクション7.1「MySQL Shell でのレポート」 を参照)、およびユーザー定義の MySQL Shell グローバルオブジェクトによって使用可能になる拡張

オブジェクトのメンバーである関数 (セクション7.2「MySQL Shell への拡張オブジェクトの追加」 を参照) を含めることができます。 単一のプラグインに複数の関数を含めて登録し、レポートと拡張オブジェクトのメンバーを混在させることができます。 MySQL Shell プラグインによってレポートまたは拡張オブジェクトのメンバーとして登録された関数は、MySQL の起動が完了するとすぐに使用できます。

MySQL Shell プラグインは、言語 (init.js または init.py ファイル) に適した初期化スクリプトを含むフォルダです。 初期化スクリプトはプラグインのエントリポイントです。 プラグインには 1 つの言語のコードのみを含めることができるため、Python で定義されたメンバーと JavaScript で定義されたメンバーが混在する拡張オブジェクトを作成する場合は、メンバーを個別の言語に適したプラグインとして格納する必要があります。

MySQL Shell プラグインを起動時に自動的にロードするには、そのフォルダが MySQL Shell ユーザー構成パスの plugins フォルダの下にある必要があります。MySQL Shell は、この場所で初期化スクリプトを検索します。MySQL Shell では、名前がドット (.) で始まる plugins の場所にあるフォルダは無視されますが、それ以外の場合、プラグインフォルダに使用する名前は重要ではありません。

plugins フォルダのデフォルトパスは、Unix では ~/.mysqlsh/plugins で、Windows では %AppData%\MySQL\mysqlsh\plugins です。 ユーザー構成パスは、環境変数 MYSQLSH_USER_CONFIG_HOME を定義することで、すべてのプラットフォームでオーバーライドできます。 この変数の値は、Windows 上の %AppData%\MySQL\mysqlsh\または Unix 上の ~/.mysqlsh/に置き換わります。

プラグインのロード中にエラーが検出されると、警告が表示され、MySQL Shell アプリケーションログにエラーの詳細が表示されます。 ロードプロセスの詳細を表示するには、MySQL Shell の起動時に --log-level=debug オプションを使用します。

MySQL Shell プラグインがロードされると、次のオブジェクトをグローバル変数として使用できます:

- 組込みグローバルオブジェクト shell、dba および util。
- シェル API メインモジュール mysql。
- X DevAPI メインモジュール mysqlx。
- AdminAPI メインモジュール dba。

7.3.1.1 共通コードおよびパッケージ

MySQL Shell プラグインまたはプラグイングループの一部である Python コードで共通コードまたは内部パッケージを使用する場合は、パッケージ名間の潜在的な競合を避けるために、命名およびインポートのために次の要件に従う必要があります:

- プラグインまたはプラグイングループの最上位フォルダと、パッケージとして認識される各内部フォルダは、Python PEP 8 スタイルガイドに従って、文字、数字、およびアンダースコアのみを使用した有効な通常のパッケージ名である必要があります。
- パッケージとして認識される各内部フォルダには、__init__.py というファイルが含まれている必要があります。
- インポート時には、パッケージ名のフルパスを指定する必要があります。 たとえば、ext という名前のプラグイングループに、sample という名前のモジュールを含む src という名前の内部パッケージを持つ demo という名前のプラグインが含まれている場合、そのモジュールは次のようにインポートする必要があります:

```
from ext.demo.src import sample
```

7.3.2 プラグイングループの作成

複数の MySQL Shell プラグインのフォルダを plugins フォルダの下の格納フォルダに配置することで、プラグイングループを作成できます。 プラグイングループには、JavaScript を使用して定義されたプラグインと Python を使用して定義されたプラグインを混在させることができます。 プラグイングループを使用すると、次のような共通のものを持つプラグインを編成できます:

- 特定のテーマに関するレポートを提供するプラグイン。
- 同じ共通コードを再利用するプラグイン。

- 同じ拡張オブジェクトに関数を追加するプラグイン。

plugins フォルダのサブディレクトリに初期化スクリプト (init.js または init.py ファイル) が含まれていない場合、MySQL Shell はそれをプラグイングループとして扱い、そのサブフォルダでプラグインの初期化スクリプトを検索します。 格納フォルダには、プラグイングループ内のプラグインによって共有されるコードを持つほかのファイルを含めることができます。 プラグインサブフォルダの場合と同様に、名前がドット (.) で始まる場合、格納フォルダは無視されますが、それ以外の場合、名前は MySQL Shell にとって重要ではありません。

たとえば、ユーザー定義の MySQL Shell グローバルオブジェクト ext によって提供されるすべての関数を構成するプラグイングループは、次のように構造化できます:

- フォルダ C:\Users\exampleuser\AppData\Roaming\MySQL\mysqlsh\plugins\ext は、プラグイングループの格納フォルダです。
- プラグインの共通コードは、C:\Users\exampleuser\AppData\Roaming\MySQL\mysqlsh\plugins\ext\common.py のこのフォルダに格納されます
- プラグイングループ内のプラグインは、それぞれ C:\Users\exampleuser\AppData\Roaming\MySQL\mysqlsh\plugins\ext\helloWorld\init.py などの init.py ファイルを含む ext フォルダのサブフォルダに格納されます。
- プラグインは、ext.common から共通コードをインポートし、その機能を使用します。

7.3.3 MySQL Shell プラグインの例

例 7.3 レポートおよび拡張オブジェクトを含む MySQL Shell プラグイン

この例では、現在実行中のプロセスを表示する関数 show_processes() と、指定された ID を持つプロセスを強制終了する関数 kill_process() を定義します。show_processes() は MySQL Shell レポートになり、kill_process() は拡張オブジェクトによって提供される関数になります。

このコードは、shell.register_report() メソッドを使用して、show_processes() を MySQL Shell レポート proc として登録します。 kill_process() を ext.process.kill() として登録するには、グローバルオブジェクト ext および拡張オブジェクト process がすでに存在するかどうかをチェックし、存在しない場合は作成して登録します。 その後、kill_process() 関数がメンバーとして process 拡張オブジェクトに追加されます。

プラグインコードはファイル ~/.mysqlsh/plugins/ext/process/init.py として保存されます。 起動時に、MySQL Shell は plugins フォルダ内のフォルダをトラバースし、この init.py ファイルを検索してコードを実行します。 レポート proc および関数 kill() が登録され、使用できるようになります。 グローバルオブジェクト ext および拡張オブジェクト process は、別のプラグインによってまだ登録されていない場合は作成および登録され、それ以外の場合は既存のオブジェクトが使用されます。

```
# Define a show_processes function that generates a MySQL Shell report

def show_processes(session, args, options):
  query = "SELECT ID, USER, HOST, COMMAND, INFO FROM INFORMATION_SCHEMA.PROCESSLIST"
  if (options.has_key('command')):
    query += " WHERE COMMAND = '%s'" % options['command']

  result = session.sql(query).execute();
  report = []
  if (result.has_data()):
    report = [result.get_column_names()]
    for row in result.fetch_all():
        report.append(list(row))

  return {"report": report}


# Define a kill_process function that will be exposed by the global object 'ext'

def kill_process(session, id):
  result = session.sql("KILL CONNECTION %d" % id).execute()


# Register the show_processes function as a MySQL Shell report
```

```
shell.register_report("proc", "list", show_processes, {"brief":"Lists the processes on the target server.",
                                                        "options": [{
                                                          "name": "command",
                                                          "shortcut": "c",
                                                          "brief": "Use this option to list processes over specific commands."
                                                        }]})


# Register the kill_process function as ext.process.kill()

# Check if global object 'ext' has already been registered
if 'ext' in globals():
    global_obj = ext
else:
    # Otherwise register new global object named 'ext'
    global_obj = shell.create_extension_object()
    shell.register_global("ext", global_obj,
        {"brief":"MySQL Shell extension plugins."})

# Add the 'process' extension object as a member of the 'ext' global object
try:
    plugin_obj = global_obj.process
except IndexError:
    # If the 'process' extension object has not been registered yet, do it now
    plugin_obj = shell.create_extension_object()
    shell.add_extension_object_member(global_obj, "process", plugin_obj,
        {"brief": "Utility object for process operations."})

# Add the kill_process function to the 'process' extension object as member 'kill'
try:
    shell.add_extension_object_member(plugin_obj, "kill", kill_process, {"brief": "Kills the process with the given ID.",
                                                      "parameters": [
                                                       {
                                                         "name":"session",
                                                         "type":"object",
                                                         "class":"Session",
                                                         "brief": "The session to be used on the operation."
                                                       },
                                                       {
                                                         "name":"id",
                                                         "type":"integer",
                                                         "brief": "The ID of the process to be killed."
                                                       }
                                                       ]
                                                      })
except Exception as e:
    shell.log("ERROR", "Failed to register ext.process.kill ({0}).".
        format(str(e).rstrip()))
```

ここで、ユーザーは MySQL Shell \show コマンドを使用してレポート proc を実行し、ext.process.kill() 関数を使用してリストされているプロセスのいずれかを停止します:

```
mysql-py> \show proc
+----+-----------------+----------------+---------+-----------------------------------------------------------------------------------------+
| ID | USER            | HOST           | COMMAND | INFO                                                                                    |
+----+-----------------+----------------+---------+-----------------------------------------------------------------------------------------+
| 66 | root            | localhost:53998 | Query   | PLUGIN: SELECT ID, USER, HOST, COMMAND, INFO FROM INFORMATION_SCHEMA.PROCESSLIST |
| 67 | root            | localhost:34022 | Sleep   | NULL                                                                                    |
| 4  | event_scheduler | localhost       | Daemon  | NULL                                                                                    |
+----+-----------------+----------------+---------+-----------------------------------------------------------------------------------------+

mysql-py> ext.process.kill(session, 67)
mysql-py> \show proc
+----+-----------------+----------------+---------+-----------------------------------------------------------------------------------------+
| ID | USER            | HOST           | COMMAND | INFO                                                                                    |
+----+-----------------+----------------+---------+-----------------------------------------------------------------------------------------+
| 66 | root            | localhost:53998 | Query   | PLUGIN: SELECT ID, USER, HOST, COMMAND, INFO FROM INFORMATION_SCHEMA.PROCESSLIST |
| 4  | event_scheduler | localhost       | Daemon  | NULL                                                                                    |
+----+-----------------+----------------+---------+-----------------------------------------------------------------------------------------+
```

第 8 章 MySQL Shell ユーティリティ

目次

8.1 アップグレードチェッカユーティリティ 135
8.2 JSON インポートユーティリティ 141
8.2.1 mysqlsh コマンドインタフェースを使用した JSON ドキュメントのインポート 143
8.2.2 --import コマンドを使用した JSON ドキュメントのインポート 143
8.2.3 BSON データ型の表現の変換 145
8.3 テーブルエクスポートユーティリティ 145
8.4 パラレルテーブルインポートユーティリティ 149
8.5 インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティ . 155
8.6 ダンプロードユーティリティ 163

MySQL Shell には、MySQL を操作するユーティリティが含まれています。 MySQL Shell 内からユーティリティにアクセスするには、JavaScript および Python モードで使用可能な util グローバルオブジェクトを使用しますが、SQL モードでは使用できません。 util グローバルオブジェクトには、次の関数があります:

checkForServerUpgrade()	MySQL サーバーインスタンスのアップグレード準備ができているかどうかを確認できるアップグレードチェッカユーティリティ。 セクション8.1「アップグレードチェッカユーティリティ」を参照してください。
importJSON()	JSON ドキュメントを MySQL Server コレクションまたはテーブルにインポートできる JSON インポートユーティリティ。 セクション8.2「JSON インポートユーティリティ」を参照してください。
exportTable()	MySQL リレーショナルテーブルをデータファイルにエクスポートするテーブルエクスポートユーティリティ。その後、MySQL Shell のパラレルテーブルインポートユーティリティを使用してターゲット MySQL サーバーのテーブルにアップロードしたり、別のアプリケーションにデータをインポートしたり、単一のデータテーブルの軽量論理バックアップとしてデータをインポートできます。 セクション 8.3「テーブルエクスポートユーティリティ」を参照してください。
importTable()	単一のデータファイルを分割し、複数のスレッドを使用してチャンクを MySQL テーブルにロードするパラレルテーブルインポートユーティリティ。 セクション 8.4「パラレルテーブルインポートユーティリティ」を参照してください。
dumpInstance(), dumpSchemas(), dumpTables()	すべてのスキーマ、選択したスキーマまたは選択したテーブルおよびビューを MySQL インスタンスから Oracle Cloud Infrastructure Object Storage バケットまたはローカルファイルのセットにエクスポートできるインスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティ。 セクション8.5「インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティ」を参照してください。
loadDump()	MySQL Shell インスタンスダンプユーティリティおよびスキーマダンプユーティリティを使用してダンプされたスキーマを MySQL インスタンスにインポートできるダンプロードユーティリティ。 セクション8.6「ダンプロードユーティリティ」を参照してください。

8.1 アップグレードチェッカユーティリティ

util.checkForServerUpgrade() 関数は、MySQL サーバーインスタンスのアップグレード準備ができているかどうかを確認できるアップグレードチェッカユーティリティです。 MySQL Shell 8.0.13 から、最初の MySQL Server 8.0 General Availability (GA) リリース (8.0.11) から現在の MySQL Shell リリース番号と一致する MySQL Server リリース番号まで、アップグレード先のターゲット MySQL Server リリースを選択できます。 アップグレードチェッカユーティリティは、指定されたターゲットリリースに関連する自動チェックを実行し、手動で行う必要がある関連チェックをさらにアドバイスします。

アップグレードチェッカユーティリティを使用して、MySQL 5.7 サーバーインスタンスの互換性エラーおよびアップグレードの問題を確認できます。 MySQL Shell 8.0.13 から、これを使用して、MySQL 8.0 リリースシリーズ内の別

の GA ステータスリリースで MySQL 8.0 サーバーインスタンスをチェックすることもできます。 MySQL Server インスタンスを指定せずに checkForServerUpgrade() を起動すると、グローバルセッションに現在接続されているインスタンスがチェックされます。 現在接続されているインスタンスを表示するには、\status コマンドを発行します。

> 注記
>
> 1. アップグレードチェッカユーティリティでは、MySQL 5.7 より前のバージョンの MySQL Server インスタンスのチェックはサポートされていません。
> 2. MySQL Server は GA リリース間のアップグレードのみをサポートしています。 MySQL 5.7 または 8.0 の GA 以外のリリースからのアップグレードはサポートされません。 サポートされているアップグレードパスの詳細は、アップグレードパス を参照してください。

MySQL Shell 8.0.16 から、アップグレードチェッカユーティリティはサーバーインスタンスの構成ファイル (my.cnf または my.ini) を確認できます。 このユーティリティは、構成ファイルで定義されているが、ターゲットの MySQL Server リリースで削除されているシステム変数をチェックします。また、構成ファイルで定義されておらず、ターゲットの MySQL Server リリースでデフォルト値が異なるシステム変数もチェックします。 これらのチェックでは、checkForServerUpgrade() を起動するときに、構成ファイルへのファイルパスを指定する必要があります。

アップグレードチェッカユーティリティは、TCP ソケットまたは Unix ソケットを使用して、X プロトコル 接続または クラシック MySQL プロトコル 接続を介して動作できます。 事前に接続を作成することも、関数の引数として指定することもできます。 ユーティリティは常に新しいセッションを作成してサーバーに接続するため、MySQL Shell グローバルセッションは影響を受けません。

MySQL Shell 8.0.20 までは、アップグレードチェッカユーティリティの実行に使用されるユーザーアカウントに ALL 権限が必要です。 MySQL Shell 8.0.21 からは、ユーザーアカウントに RELOAD、PROCESS および SELECT 権限が必要です。

アップグレードチェッカユーティリティでは、テキスト形式 (デフォルト) または JSON 形式 (devops 自動化で使用するための解析および処理が簡単な場合があります) で出力を生成できます。

アップグレードチェッカユーティリティには、次のシグネチャがあります:

```
checkForServerUpgrade (ConnectionData connectionData, Dictionary options)
```

どちらの引数もオプションです。 接続がまだ存在しない場合、最初の接続データが提供されます。2 番目の接続データは、次のオプションを指定するために使用できるディクショナリです:

password
: アップグレードチェッカユーティリティの実行に使用されるユーザーアカウントのパスワード。 このディクショナリオプションを使用するか、接続詳細の一部としてパスワードを指定できます。 パスワードを指定しない場合、ユーティリティはサーバーへの接続時にパスワードの入力を求めます。

targetVersion
: アップグレード先のターゲット MySQL Server バージョン。 MySQL Shell 8.0.22 では、リリース 8.0.11 (最初の MySQL Server 8.0 GA リリース)、8.0.12, 8.0.13, 8.0.14, 8.0.15, 8.0.16, 8.0.17, 8.0.18, 8.0.19, 8.0.20, 8.0.21 または 8.0.22 を指定できます。 短い形式のバージョン番号 8.0 を指定するか、targetVersion オプションを省略すると、ユーティリティは現在の MySQL Shell リリース番号と一致する MySQL Server リリース番号へのアップグレードをチェックします。

configPath
: 確認する MySQL サーバーインスタンスの my.cnf または my.ini 構成ファイルへのローカルパス (C:\ProgramData\MySQL\MySQL Server 8.0\my.ini など)。 ファイルパスを省略し、アップグレードチェッカユーティリティで構成ファイルを必要とするチェックを実行する必要がある場合、そのチェックは失敗し、ファイルパスを指定する必要があることを知らせるメッセージが表示されます。

outputFormat
: アップグレードチェッカユーティリティからの出力が返される形式。 このオプションを省略した場合のデフォルトはテキスト形式 (TEXT) です。 JSON を指定すると、かわりに、アップグレードチェッカユーティリティの JSON 出力 にリストされている形式で整形式の JSON 出力が返されます。

たとえば、次のコマンドは、グローバルセッションに現在接続されている MySQL サーバーインスタンスを確認し、テキスト形式で出力します:

```
mysqlsh> \status
MySQL Shell version 8.0.22
...
Server version:         5.7.25-log MySQL Community Server (GPL)
...
mysqlsh> util.checkForServerUpgrade()
```

次のコマンドは、URI user@example.com:3306 の MySQL サーバーをチェックして、最初の MySQL Server 8.0 GA ステータスリリース (8.0.11) にアップグレードします。 ユーザーパスワードと構成ファイルパスは、オプションディクショナリの一部として提供され、出力はデフォルトのテキスト形式で返されます:

```
mysqlsh> util.checkForServerUpgrade('user@example.com:3306', {"password":"password", "targetVersion":"8.0.11", "configPath":"C:\ProgramData\MySQL\MyS
```

次のコマンドは、現在の MySQL Shell リリース番号 (デフォルト) と一致する MySQL Server リリース番号にアップグレードするために同じ MySQL サーバーをチェックし、その後の処理のために JSON 出力を返します:

```
mysqlsh> util.checkForServerUpgrade('user@example.com:3306', {"password":"password", "outputFormat":"JSON", "configPath":"C:\ProgramData\MySQL\MyS
```

MySQL 8.0.13 から、mysqlsh コマンドインタフェースを使用して、コマンドラインからアップグレードチェッカユーティリティを起動できます。 この構文の詳細は、セクション5.8「API コマンドラインインタフェース」 を参照してください。 次の例では、MySQL サーバーのリリース 8.0.21 へのアップグレードをチェックし、JSON 出力を返します:

```
mysqlsh -- util checkForServerUpgrade user@localhost:3306 --target-version=8.0.21 --output-format=JSON --config-path=/etc/mysql/my.cnf
```

次の例に示すように、中カッコを使用して名前付きオプションとして接続データをグループ化することもできます。 これは、メソッド名に camelCase ではなく小文字とハイフンを使用できることも示しています:

```
mysqlsh -- util check-for-server-upgrade { --user=user --host=localhost --port=3306 } --target-version=8.0.21 --output-format=JSON --config-path=/etc/mysql/my.c
```

次の例では、Unix ソケット接続を使用し、コマンドラインからユーティリティを起動するための古い形式を示しますが、これはまだ有効です:

```
./bin/mysqlsh --socket=/tmp/mysql.sock --user=user -e "util.checkForServerUpgrade()"
```

アップグレードチェッカユーティリティのヘルプを表示するには、次のコマンドを発行します:

```
mysqlsh> util.help("checkForServerUpgrade")
```

util.checkForServerUpgrade() は値を返さなくなりました (MySQL Shell 8.0.13 の前に、値 0、1 または 2 が返されました)。

アップグレードチェッカユーティリティを起動すると、MySQL Shell はサーバーインスタンスに接続し、アップグレード用のインストールの準備 で説明されている設定をテストします。 例:

```
The MySQL server at example.com:3306, version
5.7.25-enterprise-commercial-advanced - MySQL Enterprise Server - Advanced Edition (Commercial),
will now be checked for compatibility issues for upgrade to MySQL 8.0.22...

1) Usage of old temporal type
 No issues found

2) Usage of db objects with names conflicting with new reserved keywords
 Warning: The following objects have names that conflict with new reserved keywords.
 Ensure queries sent by your applications use `quotes` when referring to them or they will result in errors.
 More information: https://dev.mysql.com/doc/refman/en/keywords.html

 dbtest.System - Table name
 dbtest.System.JSON_TABLE - Column name
 dbtest.System.cube - Column name

3) Usage of utf8mb3 charset
 Warning: The following objects use the utf8mb3 character set. It is recommended to convert them to use
 utf8mb4 instead, for improved Unicode support.
 More information: https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/charset-unicode-utf8mb3.html

 dbtest.view1.col1 - column's default character set: utf8

4) Table names in the mysql schema conflicting with new tables in 8.0
 No issues found
```

```
5) Partitioned tables using engines with non native partitioning
  Error: In MySQL 8.0 storage engine is responsible for providing its own
  partitioning handler, and the MySQL server no longer provides generic
  partitioning support. InnoDB and NDB are the only storage engines that
  provide a native partitioning handler that is supported in MySQL 8.0. A
  partitioned table using any other storage engine must be altered—either to
  convert it to InnoDB or NDB, or to remove its partitioning—before upgrading
  the server, else it cannot be used afterwards.
  More information:
    https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/upgrading-from-previous-series.html#upgrade-configuration-changes

  dbtest.part1_hash - MyISAM engine does not support native partitioning

6) Foreign key constraint names longer than 64 characters
  No issues found

7) Usage of obsolete MAXDB sql_mode flag
  No issues found

8) Usage of obsolete sql_mode flags
  No issues found

9) ENUM/SET column definitions containing elements longer than 255 characters
  No issues found

10) Usage of partitioned tables in shared tablespaces
  Error: The following tables have partitions in shared tablespaces. Before upgrading to 8.0 they need
  to be moved to file-per-table tablespace. You can do this by running query like
  'ALTER TABLE table_name REORGANIZE PARTITION X INTO
    (PARTITION X VALUES LESS THAN (30) TABLESPACE=innodb_file_per_table);'
  More information: https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/mysql-nutshell.html#mysql-nutshell-removals

  dbtest.table1 - Partition p0 is in shared tablespace tbsp4
  dbtest.table1 - Partition p1 is in shared tablespace tbsp4

11) Circular directory references in tablespace data file paths
  No issues found

12) Usage of removed functions
  Error: Following DB objects make use of functions that have been removed in
    version 8.0. Please make sure to update them to use supported alternatives
    before upgrade.
  More information:
    https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/mysql-nutshell.html#mysql-nutshell-removals

  dbtest.view1 - VIEW uses removed function PASSWORD

13) Usage of removed GROUP BY ASC/DESC syntax
  Error: The following DB objects use removed GROUP BY ASC/DESC syntax. They need to be altered so that
  ASC/DESC keyword is removed from GROUP BY clause and placed in appropriate ORDER BY clause.
  More information: https://docs.oracle.com/cd/E17952_01/mysql-8.0-relnotes-en/news-8-0-13.html#mysqld-8-0-13-sql-syntax

  dbtest.view1 - VIEW uses removed GROUP BY DESC syntax
  dbtest.func1 - FUNCTION uses removed GROUP BY ASC syntax

14) Removed system variables for error logging to the system log configuration
  No issues found

15) Removed system variables
  Error: Following system variables that were detected as being used will be
    removed. Please update your system to not rely on them before the upgrade.
  More information: https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/added-deprecated-removed.html#optvars-removed

  log_builtin_as_identified_by_password - is set and will be removed
  show_compatibility_56 - is set and will be removed

16) System variables with new default values
  Warning: Following system variables that are not defined in your
    configuration file will have new default values. Please review if you rely on
    their current values and if so define them before performing upgrade.
  More information: https://mysqlserverteam.com/new-defaults-in-mysql-8-0/

  back_log - default value will change
  character_set_server - default value will change from latin1 to utf8mb4
```

```
  collation_server - default value will change from latin1_swedish_ci to
    utf8mb4_0900_ai_ci
  event_scheduler - default value will change from OFF to ON
[...]

17) Zero Date, Datetime, and Timestamp values
  Warning: By default zero date/datetime/timestamp values are no longer allowed
    in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included in
    SQL_MODE by default. These modes should be used with strict mode as they will
    be merged with strict mode in a future release. If you do not include these
    modes in your SQL_MODE setting, you are able to insert
    date/datetime/timestamp values that contain zeros. It is strongly advised to
    replace zero values with valid ones, as they may not work correctly in the
    future.
  More information:
    https://lefred.be/content/mysql-8-0-and-wrong-dates/

  global.sql_mode - does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE
    which allows insertion of zero dates
  session.sql_mode -  of 2 session(s) does not contain either NO_ZERO_DATE or
    NO_ZERO_IN_DATE which allows insertion of zero dates
  dbtest.date1.d - column has zero default value: 0000-00-00

18) Schema inconsistencies resulting from file removal or corruption
  No issues found

19) Tables recognized by InnoDB that belong to a different engine
  No issues found

20) Issues reported by 'check table x for upgrade' command
  No issues found

21) New default authentication plugin considerations
  Warning: The new default authentication plugin 'caching_sha2_password' offers
    more secure password hashing than previously used 'mysql_native_password'
    (and consequent improved client connection authentication). However, it also
    has compatibility implications that may affect existing MySQL installations.
    If your MySQL installation must serve pre-8.0 clients and you encounter
    compatibility issues after upgrading, the simplest way to address those
    issues is to reconfigure the server to revert to the previous default
    authentication plugin (mysql_native_password). For example, use these lines
    in the server option file:

    [mysqld]
    default_authentication_plugin=mysql_native_password

    However, the setting should be viewed as temporary, not as a long term or
    permanent solution, because it causes new accounts created with the setting
    in effect to forego the improved authentication security.
    If you are using replication please take time to understand how the
    authentication plugin changes may impact you.
  More information:
    https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/upgrading-from-previous-series.html#upgrade-caching-sha2-password-compatibility-issues
    https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/upgrading-from-previous-series.html#upgrade-caching-sha2-password-replication

Errors:   7
Warnings: 36
Notices:  0

7 errors were found. Please correct these issues before upgrading to avoid compatibility issues.
```

- この例では、サーバーインスタンスで実行されたチェックで、チェックされたサーバーで見つかったアップグレードシナリオのエラーが返されたため、サーバーインスタンスをターゲットの MySQL 8.0 リリースにアップグレードする前に変更が必要です。
- レポートのエラー数をクリアするために必要な変更を行った場合は、さらに変更を加えて警告を削除することも検討する必要があります。 これらの構成の改善により、サーバーインスタンスとターゲットリリースとの互換性が向上します。 ただし、サーバーインスタンスは、警告を削除せずに正常にアップグレードできます。
- この例に示すように、アップグレードチェッカユーティリティでは、自動化できず、手動で行う必要がある、警告または通知 (情報) レベルのいずれかとして評価される、さらに関連性のあるチェックに関するアドバイスおよび指示も提供される場合があります。

アップグレードチェッカユーティリティの JSON 出力

outputFormat ディクショナリオプションを使用して JSON 出力を選択した場合、アップグレードチェッカユーティリティによって返される JSON オブジェクトには、次のキーと値のペアがあります:

- serverAddress: チェックされた MySQL サーバーインスタンスへの MySQL Shell 接続のホスト名およびポート番号。
- serverVersion: チェックされたサーバーインスタンスの MySQL バージョンが検出されました。
- targetVersion: アップグレードチェックのターゲット MySQL バージョン。
- errorCount: ユーティリティで検出されたエラーの数。
- warningCount: ユーティリティで検出された警告の数。
- noticeCount: ユーティリティによって検出された通知の数。
- summary: テキスト出力の最後に提供されるサマリーステートメントのテキスト (「既知の互換性エラーまたは問題が見つかりませんでした。」など)。
- checksPerformed: JSON オブジェクトの配列。自動的にチェックされた個々のアップグレードの問題 (削除された関数の使用など) ごとに 1 つずつ。 各 JSON オブジェクトには、次のキーと値のペアがあります:
  - id: 一意の文字列であるチェックの ID。
  - title: チェックの短い説明。
  - status: チェックが正常に実行された場合は「OK」、それ以外の場合は「ERROR」。
  - description: アドバイスを組み込むチェック (使用可能な場合) の詳細な説明、またはチェックの実行に失敗した場合は エラーメッセージ。
  - documentationLink: 使用可能な場合は、詳細情報またはアドバイスを含むドキュメントへのリンク。
  - detectedProblems: チェックの結果として見つかった エラー、警告または通知を表す JSON オブジェクトの配列 (空の可能性があります)。 各 JSON オブジェクトには、次のキーと値のペアがあります:
    - level: メッセージレベル。エラー、警告または通知のいずれか。
    - dbObject: メッセージが関連するデータベースオブジェクトを識別する文字列。
    - description: 使用可能な場合は、データベースオブジェクトに関する問題の特定の説明を含む文字列。
- manualChecks: JSON オブジェクトの配列で、アップグレードパスに関連し、手動でチェックする必要がある個々のアップグレードの問題 (たとえば、MySQL 8.0 でのデフォルトの認証プラグインの変更) ごとに使用されます。 各 JSON オブジェクトには、次のキーと値のペアがあります:
  - id: 一意の文字列である手動チェックの ID。
  - title: 手動チェックの簡単な説明。
  - description: 手動チェックの詳細な説明と、情報およびアドバイス。
  - documentationLink: 使用可能な場合は、詳細情報またはアドバイスを含むドキュメントへのリンク。

8.2 JSON インポートユーティリティ

MySQL Shell 8.0.13 で導入された MySQL Shell JSON インポートユーティリティ util.importJSON() を使用すると、JSON ドキュメントをファイル (または FIFO 特殊ファイル) または標準入力から MySQL Server コレクションまたはリレーショナルテーブルにインポートできます。 このユーティリティは、指定された JSON ドキュメントが整形式であることをチェックしてターゲットデータベースに挿入するため、複数の INSERT ステートメントを使用したり、スクリプトを記述してこのタスクを実行する必要がなくなります。

MySQL Shell 8.0.14 から、インポートユーティリティは JSON ドキュメントで表される BSON (バイナリ JSON) データ型を処理できます。 BSON ドキュメントで使用されるデータ型はすべて JSON でネイティブにサポートされているわけではありませんが、JSON 形式の拡張機能を使用して表すことができます。 インポートユーティリティでは、JSON 拡張を使用して BSON データ型を表すドキュメントを処理し、それらを同一または互換性のある MySQL 表現に変換し、その表現を使用してデータ値をインポートできます。 変換された結果のデータ値は、式およびインデックスで使用でき、SQL ステートメントおよび X DevAPI 関数で操作できます。

JSON ドキュメントは、既存のテーブルまたはコレクション、またはインポート用に作成された新しいテーブルまたはコレクションにインポートできます。 ターゲットのテーブルまたはコレクションが指定したデータベースに存在しない場合は、デフォルトのコレクションまたはテーブル構造を使用してユーティリティによって自動的に作成されます。 デフォルトのコレクションは、schema オブジェクトから createCollection() 関数をコールすることで作成されます。 デフォルトテーブルは次のように作成されます:

```
CREATE TABLE `dbname`.`tablename` (
  target_column JSON,
  id INTEGER AUTO_INCREMENT PRIMARY KEY
) CHARSET utf8mb4 ENGINE=InnoDB;
```

デフォルトのコレクション名またはテーブル名は、指定されたインポートファイルの名前 (ファイル拡張子なし) で、デフォルトの target_column 名は doc です。

BSON 型の JSON 拡張機能を MySQL 型に変換するには、インポートユーティリティの実行時に convertBsonTypes オプションを指定する必要があります。 特定の BSON データ型のマッピングおよび変換を制御するための追加オプションが使用可能です。 BSON タイプの JSON 拡張子を持つドキュメントをインポートし、このオプションを使用しない場合、ドキュメントは入力ファイルで表されるのと同じ方法でインポートされます。

JSON インポートユーティリティには、サーバーへの既存の X プロトコル 接続が必要です。 ユーティリティは、クラシック MySQL プロトコル 接続を介して動作できません。

MySQL Shell API では、JSON インポートユーティリティは util グローバルオブジェクトの関数であり、次のシグネチャを持ちます:

```
importJSON (path, options)
```

path は、インポートする JSON ドキュメントを含むファイルのファイルパスを指定する文字列です。 これには、ディスクに書き込まれたファイルまたは FIFO 特殊ファイル (名前付きパイプ) を指定できます。 標準入力は、ユーティリティの --import コマンドライン起動でのみインポートできます。

options はインポートオプションのディクショナリで、空の場合は省略できます。 (MySQL 8.0.14 より前は、ディクショナリが必要でした。) JSON ドキュメントをインポートする場所と方法を指定するには、次のオプションを使用できます:

schema: "db_name"
: ターゲットデータベースの名前。 このオプションを省略すると、MySQL Shell は URI のような接続文字列、\use コマンドまたは MySQL Shell オプションで指定されているように、現在のセッションで使用されているスキーマ名を識別して使用しようとします。 スキーマ名が指定されておらず、セッションから識別できない場合は、エラーが返されます。

collection: "collection_name"
: ターゲット収集の名前。 これは、テーブルとカラムを指定するかわりに使用できます。 コレクションが存在しない場合は、ユーティリティによって作成されます。 collection、table または tableColumn のいずれのオプションも指定しない場合、ユーティリティはデフォルトで、指定されたインポートファイルの名前 (ファイル拡張子なし) でターゲットコレクションを使用または作成します。

table: "table_name"	ターゲットテーブルの名前。 これは、コレクションを指定するかわりに使用できます。 テーブルが存在しない場合は、ユーティリティによって作成されます。
tableColumn: "column_name"	JSON ドキュメントがインポートされるターゲットテーブルのカラムの名前。 テーブルがすでに存在する場合は、指定したカラムがテーブルに存在する必要があります。 table オプションを指定して tableColumn オプションを省略すると、デフォルトのカラム名 doc が使用されます。 tableColumn オプションを指定して table オプションを省略した場合、指定したインポートファイルの名前 (ファイル拡張子なし) がテーブル名として使用されます。
convertBsonTypes: true	拡張機能を使用して JSON 形式に表される BSON データ型を認識および変換します。 このオプションのデフォルトは false です。 convertBsonTypes: true を指定すると、表現された BSON タイプはそれぞれ同一または互換性のある MySQL 表現に変換され、その表現を使用してデータ値がインポートされます。 特定の BSON データ型のマッピングおよび変換を制御するための追加オプションが使用可能です。これらの制御オプションおよびデフォルトの型変換のリストは、セクション 8.2.3「BSON データ型の表現の変換」 を参照してください。 convertBsonOid オプションも true に設定する必要があります。これは、convertBsonTypes: true を指定した場合のオプションのデフォルト設定です。 BSON 型の JSON 拡張子を持つドキュメントをインポートし、convertBsonTypes: true を使用しない場合、ドキュメントは入力ファイルに埋め込まれた JSON ドキュメントとして表されるのと同じ方法でインポートされます。
convertBsonOid: true	MongoDB 拡張 JSON 厳密モードで表される、ドキュメントの _id 値として使用される 12 バイト BSON 型である MongoDB ObjectIDs を認識および変換します。 このオプションのデフォルトは convertBsonTypes オプションの値であるため、このオプションを true に設定すると、MongoDB ObjectIDs も自動的に変換されます。 MongoDB からデータをインポートする場合、MySQL Server では _id 値を varbinary(32) タイプに変換する必要があるため、BSON タイプを変換しない場合は、convertBsonOid を常に true に設定する必要があります。
extractOidTime: "field_name"	ドキュメントの _id フィールドの MongoDB ObjectID に含まれるタイムスタンプ値を認識して抽出し、インポートされたデータの別のフィールドに配置します。extractOidTime は、タイムスタンプを含むドキュメント内のフィールドに名前を付けます。 タイムスタンプは ObjectID の最初の 4 バイトで、変更されません。このオプションを使用するように convertBsonOid: true を設定する必要があります。これは、convertBsonTypes が true に設定されている場合のデフォルトです。

次の例 (MySQL ShellJavaScript モードの最初の例と MySQL ShellPython モードの次の例) では、/tmp/products.json ファイルの JSON ドキュメントを mydb データベースの products コレクションにインポートします:

```
mysql-js> util.importJson("/tmp/products.json", {schema: "mydb", collection: "products"})
```

```
mysql-py> util.import_json("/tmp/products.json", {"schema": "mydb", "collection": "products"})
```

MySQL Shell JavaScript モードの次の例にはオプションが指定されていないため、ディクショナリは省略されます。mydb は、MySQL Shell セッションのアクティブなスキーマです。 したがって、ユーティリティは、ファイル/tmp/stores.json 内の JSON ドキュメントを mydb データベース内の stores という名前のコレクションにインポートします:

```
mysql-js> \use mydb
mysql-js> util.importJson("/tmp/stores.json")
```

MySQL ShellJavaScript モードの次の例では、ファイル/europe/regions.json の JSON ドキュメントを mydb データベースの regions というリレーショナルテーブルの jsondata カラムにインポートします。 JSON 拡張によってドキュメントで表される BSON データ型は、MySQL 表現に変換されます:

```
mysql-js> util.importJson("/europe/regions.json", {schema: "mydb", table: "regions", tableColumn: "jsondata", convertBsonTypes: true});
```

MySQL ShellJavaScript モードの次の例では、BSON データ型の JSON 表現を MySQL 表現に変換せずに、同じインポートを実行します。 ただし、ドキュメント内の MongoDB ObjectIDs は MySQL の要求に応じて変換され、タイムスタンプも抽出されます:

```
mysql-js> util.importJson("/europe/regions.json", {schema: "mydb", table: "regions", tableColumn: "jsondata", convertBsonOid: true, extractOidTime: "idTime"});
```

インポートが完了するか、Ctrl+C を持つユーザーまたはエラーによってインポートが途中で停止されると、正常にインポートされた JSON ドキュメントの数と該当するエラーメッセージを示すメッセージがユーザーに返されます。 関数自体が void を返すか、エラーの場合は例外を返します。

JSON インポートユーティリティは、コマンドラインからも起動できます。 コマンドラインの起動には、2 つの代替形式を使用できます。 ファイル (または FIFO 特殊ファイル) からの入力のみを受け入れる mysqlsh コマンドインタフェース、または標準入力またはファイルからの入力を受け入れる --import コマンドを使用できます。

8.2.1 mysqlsh コマンドインタフェースを使用した JSON ドキュメントのインポート

mysqlsh コマンドインタフェースを使用して、次のように JSON インポートユーティリティを起動します:

```
mysqlsh user@host:port/mydb -- util importJson <path> [options]
or
mysqlsh user@host:port/mydb -- util import-json <path> [options]
```

この構文の詳細は、セクション5.8「API コマンドラインインタフェース」 を参照してください。 JSON インポートユーティリティの場合は、次のようにパラメータを指定します:

user　　JSON インポートユーティリティの実行に使用されるユーザーアカウントのユーザー名。

ホスト　　MySQL サーバーのホスト名。

port　　MySQL サーバーへの MySQL Shell 接続のポート番号。 この接続のデフォルトポートは 33060 です。

mydb　　ターゲットデータベースの名前。 コマンドラインから JSON インポートユーティリティを起動する場合は、ターゲットデータベースを指定する必要があります。 URI のような接続文字列に指定することも、追加の --schema コマンドラインオプションを使用することもできます。

パス　　インポートする JSON ドキュメントを含むファイル (または FIFO 特殊ファイル) のファイルパス。

options　　--collection、--table および --tableColumn オプションでは、ターゲットコレクションまたはターゲットテーブルとカラムを指定します。 mysqlsh コマンドインタフェースを使用して JSON インポートユーティリティを起動した場合の関係およびデフォルトは、対応するオプションが MySQL Shell セッションで使用された場合と同じです。 これらのオプションのいずれも指定しない場合、ユーティリティはデフォルトで、指定されたインポートファイルの名前 (ファイル拡張子なし) でターゲットコレクションを使用または作成します。

--convertBsonTypes オプションは、拡張機能を使用して表される BSON データ型を JSON 形式に変換します。 特定の BSON データ型の追加の制御オプションも指定できます。これらの制御オプションおよびデフォルトの型変換のリストは、セクション8.2.3「BSON データ型の表現の変換」 を参照してください。 --convertBsonTypes を指定すると、--convertBsonOid オプションが自動的にオンに設定されます。 MySQL Server では_id 値を varbinary(32) タイプに変換する必要があるため、MongoDB からデータをインポートする際に BSON タイプを変換しない場合は、--convertBsonOid を指定する必要があります。--extractOidTime=field_name を使用すると、_id 値から別のフィールドにタイムスタンプを抽出できます。

次の例では、ファイル products.json の JSON ドキュメントを mydb データベースの products コレクションにインポートします:

```
mysqlsh user@localhost/mydb -- util importJson products.json --collection=products
```

8.2.2 --import コマンドを使用した JSON ドキュメントのインポート

--import コマンドは、JSON インポートユーティリティのコマンドライン起動用の mysqlsh コマンドインタフェースのかわりに使用できます。 このコマンドは、オプション名を使用せずに短い形式の構文を提供し、標準入力から JSON ドキュメントを受け入れます。 構文は次のとおりです:

```
mysqlsh user@host:port/mydb --import <path> [target] [tableColumn] [options]
```

mysqlsh コマンドインタフェースと同様に、URI のような接続文字列で、または追加の --schema コマンドラインオプションを使用して、ターゲットデータベースを指定する必要があります。 --import コマンドの最初のパラメータは、

インポートする JSON ドキュメントを含むファイルのファイルパスです。 標準入力から JSON ドキュメントを読み取るには、ファイルパスのかわりにダッシュ (-) を指定します。 入力ストリームの終わりは、ファイルの終わりを示すインジケータです。これは、Unix システムでは Ctrl+D、Windows システムでは Ctrl+Z です。

パス (標準入力の場合は -) を指定すると、次のパラメータはターゲットコレクションまたはテーブルの名前になります。 標準入力を使用する場合は、ターゲットを指定する必要があります。

- 標準入力を使用し、指定したターゲットが指定したスキーマに存在するリレーショナルテーブルである場合、ドキュメントはそのテーブルにインポートされます。 カラム名を指定するパラメータをさらに指定できます。この場合、指定したカラムがインポート先に使用されます。 それ以外の場合は、既存のテーブルに存在する必要があるデフォルトのカラム名 doc が使用されます。 ターゲットが既存のテーブルでない場合、ユーティリティは指定されたターゲット名のコレクションを検索し、そこにドキュメントをインポートします。 そのようなコレクションが見つからない場合、ユーティリティは指定されたターゲット名でコレクションを作成し、そこにドキュメントをインポートします。 テーブルを作成してテーブルにインポートするには、さらにパラメータとしてカラム名を指定する必要もあります。この場合、ユーティリティは指定されたテーブル名でリレーショナルテーブルを作成し、指定されたカラムにデータをインポートします。

- ファイルパスとターゲットを指定すると、ユーティリティは指定されたターゲット名のコレクションを検索します。 見つからない場合、ユーティリティはデフォルトでその名前のコレクションを作成し、そこにドキュメントをインポートします。 ファイルをテーブルにインポートするには、さらにパラメータとしてカラム名を指定する必要もあります。この場合、ユーティリティは既存のリレーショナルテーブルを検索してインポートするか、指定されたテーブル名でリレーショナルテーブルを作成し、指定されたカラムにデータをインポートします。

- ファイルパスを指定し、ターゲットを指定しない場合、ユーティリティは、指定されたインポートファイルの名前 (ファイル拡張子なし) を持つ、指定されたスキーマ内の既存のコレクションを検索します。 ドキュメントが見つかった場合は、そのドキュメントにインポートされます。 指定されたインポートファイルの名前のコレクションが指定されたスキーマに見つからない場合、ユーティリティはその名前のコレクションを作成し、そのコレクションにドキュメントをインポートします。

BSON (バイナリ JSON) データ型の表現を含むドキュメントをインポートする場合は、オプション --convertBsonOid、--extractOidTime=field_name、--convertBsonTypes および セクション8.2.3「BSON データ型の表現の変換」 にリストされている制御オプションも指定できます。

次の例では、標準入力から JSON ドキュメントを読み取り、mydb データベースの territories という名前のターゲットにインポートします。 territories という名前のコレクションまたはテーブルが見つからない場合、ユーティリティは territories という名前のコレクションを作成し、そのコレクションにドキュメントをインポートします。 ドキュメントを作成して territories という名前のリレーショナルテーブルにインポートする場合は、さらにパラメータとしてカラム名を指定する必要があります。

```
mysqlsh user@localhost/mydb --import - territories
```

次の例では、ファイルパスとターゲットを使用して、ファイル/europe/regions.json の JSON ドキュメントを mydb データベースの regions というリレーショナルテーブルの jsondata カラムにインポートします。 スキーマ名は、URI のような接続文字列ではなく、--schema コマンドラインオプションを使用して指定します:

```
mysqlsh user@localhost:33062 --import /europe/regions.json regions jsondata --schema=mydb
```

次の例では、ファイルパスが指定されていますが、ターゲットが指定されていない場合、JSON ドキュメントがファイル/europe/regions.json にインポートされます。 指定された mydb データベースに regions という名前のコレクションまたはテーブル (拡張子のない指定されたインポートファイルの名前) が見つからない場合、ユーティリティは regions という名前のコレクションを作成し、そのコレクションにドキュメントをインポートします。 regions という名前のコレクションがすでに存在する場合、ユーティリティはそのコレクションにドキュメントをインポートします。

```
mysqlsh user@localhost/mydb --import /europe/regions.json
```

MySQL Shell は、「127.0.0.1 の MySQL Server で、ファイル/europe/regions.json"からテーブル`mydb`.`regions`にインポートしています:33062」など、インポートのパラメータを確認するメッセージを返します。

インポートが完了するか、Ctrl+C を持つユーザーまたはエラーによってインポートが途中で停止されると、正常にインポートされた JSON ドキュメントの数と該当するエラーメッセージを示すメッセージがユーザーに返されます。 インポートが正常に終了した場合はゼロが返され、エラーが発生した場合はゼロ以外の終了コードが返されます。

8.2.3 BSON データ型の表現の変換

convertBsonTypes: true (--convertBsonTypes) オプションを指定して JSON 拡張で表される BSON データ型を変換すると、デフォルトで BSON 型は次のようにインポートされます:

日付 (「date」)	フィールドの値を含む単純な値。
タイムスタンプ (「timestamp」)	time_t 値を使用して作成された MySQL タイムスタンプ。
小数 (「decimal」)	小数値の文字列表現を含む単純な値。
Integer (「int」 または 「long」)	整数値。
正規表現 (「regex」 とオプション)	正規表現のみを含み、オプションを無視する文字列。 オプションが存在する場合は、警告が出力されます。
バイナリデータ (「binData」)	Base64 string.
ObjectID (「objectId」)	フィールドの値を含む単純な値。

次の制御オプションを指定して、これらの BSON タイプのマッピングおよび変換を調整できます。次のいずれかの制御オプションを使用するには、convertBsonTypes: true (--convertBsonTypes) を指定する必要があります:

ignoreDate: true (--ignoreDate)	BSON 「date」 タイプの変換を無効にします。 データは、入力ファイルとまったく同じように埋込み JSON ドキュメントとしてインポートされます。
ignoreTimestamp: true (--ignoreTimestamp)	BSON 「timestamp」 タイプの変換を無効にします。 データは、入力ファイルとまったく同じように埋込み JSON ドキュメントとしてインポートされます。
decimalAsDouble: true (--decimalAsDouble)	BSON 「decimal」 タイプの値を文字列ではなく MySQL DOUBLE タイプに変換します。
ignoreRegex: true (--ignoreRegex)	正規表現 (BSON 「regex」 タイプ) の変換を無効にします。 データは、入力ファイルとまったく同じように埋込み JSON ドキュメントとしてインポートされます。
ignoreRegexOptions: false (--ignoreRegexOptions=false)	正規表現に関連付けられたオプションを文字列に含め、正規表現自体も含めます (/<regular expression>/<options>の形式)。 デフォルトでは、オプションは無視されます (ignoreRegexOptions: true) が、オプションが存在する場合は警告が出力されます。ignoreRegexOptions を指定するには、ignoreRegex を false のデフォルトに設定する必要があります。
ignoreBinary: true (--ignoreBinary)	BSON 「binData」 タイプの変換を無効にします。 データは、入力ファイルとまったく同じように埋込み JSON ドキュメントとしてインポートされます。

次の例では、ファイル/europe/regions.json から mydb データベースの regions というリレーショナルテーブルの jsondata カラムにドキュメントをインポートします。 JSON 拡張によって表される BSON データ型は、埋込み JSON ドキュメントとしてインポートされる正規表現を除き、MySQL 表現に変換されます:

```
mysqlsh user@localhost/mydb --import /europe/regions.json regions jsondata --convertBsonTypes --ignoreRegex
```

8.3 テーブルエクスポートユーティリティ

MySQL Shell 8.0.22 で導入された MySQL Shell テーブルエクスポートユーティリティ util.exportTable() は、MySQL リレーショナルテーブルをローカルサーバーまたは Oracle Cloud Infrastructure Object Storage バケット上のデータファイルにエクスポートします。 その後、MySQL Shell パラレルテーブルインポートユーティリティ util.importTable() (セクション8.4「パラレルテーブルインポートユーティリティ」 を参照) を使用して、データをターゲット MySQL サーバー上のテーブルにアップロードできます。このユーティリティでは、パラレル接続を使用して、大規模なデータファイルの高速データインポートを提供します。 データファイルは、別のアプリケーションにデータをインポートしたり、単一のデータテーブルの軽量論理バックアップとして使用することもできます。

デフォルトでは、テーブルエクスポートユーティリティは、MySQL Shell パラレルテーブルインポートユーティリティのデフォルト形式でデータファイルを生成します。 DOS または UNIX システムの CSV ファイルおよび TSV

ファイルのエクスポートには、事前設定オプションを使用できます。 テーブルエクスポートユーティリティで JSON データを生成できません。 SELECT...INTO OUTFILE ステートメントと同様にフィールド処理オプションおよび行処理オプションを設定して、任意の形式でデータファイルを作成することもできます。

テーブルエクスポートファイルの宛先を選択する場合、MySQL DB システムにインポートするには、パラレルテーブルインポートユーティリティを実行する MySQL Shell インスタンスが、MySQL DB システムにアクセスできる Oracle Cloud Infrastructure Compute インスタンスにインストールされている必要があります。 テーブルをオブジェクトストレージバケット内のファイルにエクスポートする場合、コンピュートインスタンスからオブジェクトストレージバケットにアクセスできます。 ローカルシステムにテーブルエクスポートファイルを作成する場合は、コンピュートインスタンスに選択したオペレーティングシステムに応じて、選択したコピーユーティリティを使用して Oracle Cloud Infrastructure Compute インスタンスに転送する必要があります。

テーブルエクスポートユーティリティを使用したエクスポートには、次の要件が適用されます:

- ソース MySQL インスタンスおよびアップグレード先 MySQL インスタンスには、MySQL 5.7 以上が必要です。
- Oracle Cloud Infrastructure Object Storage バケットへのファイルの転送に使用されるアップロード方法のファイルサイズ制限は、1.2 TiB です。

テーブルエクスポートユーティリティは、MySQL Shell グローバルセッションを使用して、エクスポートを実行するターゲット MySQL サーバーの接続詳細を取得します。 ユーティリティを実行する前に、(X プロトコル 接続または クラシック MySQL プロトコル 接続を持つことができる) グローバルセッションをオープンする必要があります。 ユーティリティはスレッドごとに独自のセッションを開き、接続圧縮や SSL オプションなどのオプションをグローバルセッションからコピーし、グローバルセッションをこれ以上使用しません。 データ転送の最大速度を制限して、ネットワーク上の負荷を分散できます。

MySQL Shell API では、テーブルエクスポートユーティリティはutilグローバルオブジェクトの関数であり、次のシグネチャを持ちます:

```
util.exportTable(table, outputUrl[, options])
```

table は、データファイルにエクスポートするリレーショナルデータテーブルの名前です。 テーブル名は有効なスキーマ名で修飾でき、必要に応じてバックティック文字で引用符で囲むことができます。 スキーマを省略すると、MySQL Shell グローバルセッションのアクティブなスキーマが使用されます。

データをローカルファイルシステムにエクスポートする場合、outputUrl は、エクスポートされたデータファイルへのパスとファイル名自体を指定する文字列で、適切な拡張子が付いています。 絶対パスまたは現在の作業ディレクトリからの相対パスを指定できます。 ローカルディレクトリパスの前に file:// スキーマを付けることができます。 MySQL ShellJavaScript モードのこの例では、ユーザーはデフォルトの言語を使用して hr スキーマから employees テーブルをエクスポートします。 ファイルはユーザーホームディレクトリの exports ディレクトリに書き込まれ、次の形式のファイルに適した .txt 拡張子が付与されます:

```
shell-js> util.exportTable("hr.employees", "file:///home/hanna/exports/employees.txt")
```

エクスポートを実行する前にターゲットディレクトリが存在する必要がありますが、空である必要はありません。 エクスポートされたデータファイルがすでに存在する場合は、上書きされます。 ローカルディレクトリへのエクスポートの場合、データファイルはアクセス権限 rw-r----- で作成されます (これらがサポートされているオペレーティングシステム上)。 ファイルの所有者は、MySQL Shell を実行しているユーザーアカウントです。

Oracle Cloud Infrastructure Object Storage バケットにデータをエクスポートする場合、outputUrl はバケット内のデータファイルの名前で、適切なファイル拡張子が含まれます。 ディレクトリセパレータを含めて、ディレクトリ構造をシミュレートできます。 osBucketName オプションを使用してオブジェクトストレージバケットの名前を指定し、osNamespace オプションを使用してバケットのネームスペースを識別します。 MySQL ShellPython モードのこの例では、ユーザーは hr スキーマから TSV 形式のファイルとして employees テーブルをオブジェクトストレージバケット hanna-bucket にエクスポートします:

```
shell-py> util.export_table("hr.employees", "dump/employees.tsv", {
      > dialect: "tsv", "osBucketName": "hanna-bucket", "osNamespace": "idx28w1ckztq" })
```

オブジェクトストレージバケットのネームスペースは、Oracle Cloud Infrastructure コンソールのバケット詳細ページの「バケット情報」タブに表示されるか、Oracle Cloud Infrastructure コマンドラインインタフェースを使用して取得できます。 オブジェクトストレージバケットへの接続は、デフォルトの Oracle Cloud Infrastructure CLI 構成ファイル

のデフォルトプロファイル、または ociConfigFile および ociProfile オプションを使用して指定する代替詳細を使用して確立されます。 CLI 構成ファイルの設定手順については、「SDK および CLI 構成ファイル」を参照してください。

options はオプションのディクショナリで、空の場合は省略できます。 テーブルエクスポートユーティリティでは、次のオプションを使用できます:

dialect: [default|csv|csv-unix|tsv]

エクスポートされたデータファイルの形式のフィールド処理オプションと行処理オプションのセットを指定します。 linesTerminatedBy, fieldsTerminatedBy, fieldsEnclosedBy, fieldsOptionallyEnclosed および fieldsEscapedBy オプションのいずれかまたは複数を指定して設定を変更することで、選択した言語をさらにカスタマイズするためのベースとして使用できます。

デフォルトの言語では、そのステートメントのデフォルト設定で SELECT...INTO OUTFILE ステートメントを使用して作成されるものと一致するデータファイルが生成されます。.txt は、これらの出力ファイルに割り当てる適切なファイル拡張子です。 DOS または UNIX システム (.csv) および TSV ファイル (.tsv) の CSV ファイルをエクスポートするには、他のダイアレクトを使用できます。

各言語に適用される設定は次のとおりです:

表 8.1 テーブルエクスポートユーティリティの言語設定

dialect	linesTerminatedBy	fieldsTerminatedBy	fieldsEnclosedBy	fieldsOptionallyEnclosed	fieldsEscapedBy
default	[LF]	[TAB]	[空]	false	\
csv	[CR][LF]	,	"	true	\
csv-unix	[LF]	,	"	false	\
tsv	[CR][LF]	[TAB]	"	true	\

注記

1. ダイアレクトのキャリッジリターンおよびラインフィードの値は、オペレーティングシステムに依存しません。
2. linesTerminatedBy, fieldsTerminatedBy, fieldsEnclosedBy, fieldsOptionallyEnclosed および fieldsEscapedBy オプションを使用する場合、コマンドインタプリタのエスケープ規則に応じて、バックスラッシュ文字 (\) をオプション値で使用する場合は二重にする必要があります。
3. SELECT...INTO OUTFILE ステートメントを使用した MySQL サーバーと同様に、MySQL Shell では、指定したフィールド処理および行処理オプションは検証されません。 これらのオプションを正しく選択しないと、データが部分的または誤ってエクスポートされる可能性があります。 エクスポートを開始する前に必ず設定を確認し、後で結果を確認してください。

linesTerminatedBy: "characters"

ユーティリティがエクスポートされたデータファイルの各行を終了するために使用する 1 つ以上の文字 (または空の文字列)。 デフォルトは、指定された方言、または方言オプションが省略されている場合は改行文字 (\n) です。 このオプションは、SELECT...INTO OUTFILE ステートメントの LINES TERMINATED BY オプションと同等です。 ユーティリティでは、空の文字列に設定されている SELECT...INTO OUTFILE ステートメントの LINES STARTING BY オプションに相当するものは提供されないことに注意してください。

fieldsTerminatedBy: "characters"

ユーティリティがエクスポートされたデータファイルの各フィールドを終了するために使用する 1 つ以上の文字 (または空の文字列)。 デフォルトは、指定された言語、または言語オプションが省略されている場合はタブ文字 (\t) です。 このオプ

	ションは、SELECT...INTO OUTFILE ステートメントの FIELDS TERMINATED BY オプションと同等です。
fieldsEnclosedBy: "character"	ユーティリティがエクスポートされたデータファイル内の各フィールドを囲む単一の文字 (または空の文字列)。 デフォルトは、指定された言語に対するものであり、dialect オプションが省略されている場合は空の文字列です。 このオプションは、SELECT...INTO OUTFILE ステートメントの FIELDS ENCLOSED BY オプションと同等です。
fieldsOptionallyEnclosed: [true | false]	fieldsEnclosedBy に指定された文字が、エクスポートされたデータファイル (false) のすべてのフィールドを囲むか、または CHAR, BINARY, TEXT や ENUM (true) などの文字列データ型を持つフィールドのみを囲むか。 デフォルトは、指定された言語、または dialect オプションが省略されている場合は false です。 このオプションにより、fieldsEnclosedBy オプションは SELECT...INTO OUTFILE ステートメントの FIELDS OPTIONALLY ENCLOSED BY オプションと同等になります。
fieldsEscapedBy: "character"	エクスポートされたデータファイルでエスケープシーケンスを開始する文字。 デフォルトは、指定された言語の場合と同様です。または、dialect オプションが省略されている場合はバックスラッシュ (\)) です。 このオプションは、SELECT...INTO OUTFILE ステートメントの FIELDS ESCAPED BY オプションと同等です。 このオプションを空の文字列に設定した場合、文字はエスケープされません。SELECT...INTO OUTFILE で使用される特殊文字はエスケープする必要があるため、これはお薦めしません。
osBucketName: "string"	エクスポートされたデータファイルが書き込まれる Oracle Cloud Infrastructure Object Storage バケットの名前。 デフォルトでは、~/.oci/config にある Oracle Cloud Infrastructure CLI 構成ファイルの[DEFAULT]プロファイルを使用して、バケットへの接続が確立されます。 ociConfigFile および ociProfile オプションを使用して、接続に使用される代替プロファイルを置換できます。 CLI 構成ファイルの設定手順については、「SDK および CLI 構成ファイル」を参照してください。
osNamespace: "string"	osBucketName によって指定されたオブジェクトストレージバケットが配置される Oracle Cloud Infrastructure ネームスペース。 オブジェクトストレージバケットのネームスペースは、Oracle Cloud Infrastructure コンソールのバケット詳細ページの「バケット情報」タブに表示されるか、Oracle Cloud Infrastructure コマンドラインインタフェースを使用して取得できます。
ociConfigFile: "string"	デフォルトの場所の ~/.oci/config ではなく、接続に使用するプロファイルを含む Oracle Cloud Infrastructure CLI 構成ファイル。
ociProfile: "string"	接続に使用される Oracle Cloud Infrastructure CLI 構成ファイル内の[DEFAULT]プロファイルではなく、接続に使用する Oracle Cloud Infrastructure プロファイルのプロファイル名。
maxRate: "string"	エクスポート時のデータ読取りスループットのスレッド当たりの最大バイト数 (毎秒)。 単位接尾辞 k (キロバイト)、M (メガバイト)、および G (ギガバイト) を使用できます (たとえば、100M を設定すると、スループットがスレッド当たり 100 メガバイト/秒に制限されます)。 0 (デフォルト値) を設定するか、オプションを空の文字列に設定すると、制限は設定されません。
showProgress: [true | false]	エクスポートの進行状況情報を表示 (true) または非表示 (false) します。 MySQL Shell が対話型モードの場合など、stdout が端末 (tty) の場合、デフォルトは true です。それ以外の場合は false です。 進捗情報には、エクスポートされる行の推定合計数、これまでにエクスポートされた行数、完了率およびスループット (行およびバイト/秒) が含まれます。
compression: "string"	エクスポートされたデータファイルの書込み時に使用する圧縮タイプ。 デフォルトでは、圧縮なし (none) が使用されます。 または、gzip 圧縮 (gzip) または zstd 圧縮 (zstd) を使用します。
defaultCharacterSet: "string"	エクスポートのために MySQL Shell によってサーバーに開かれるセッション接続中に使用される文字セット。 デフォルトは utf8mb4 です。 システム変数

character_set_client、character_set_connection および character_set_results のセッション値は、接続ごとにこの値に設定されます。 文字セットは、character_set_client システム変数で許可され、MySQL インスタンスでサポートされている必要があります。

8.4 パラレルテーブルインポートユーティリティ

MySQL Shell 8.0.17 で導入された MySQL Shell パラレルテーブルインポートユーティリティ util.importTable() は、大規模なデータファイルの MySQL リレーショナルテーブルへの高速データインポートを提供します。 このユーティリティは、入力データファイルを分析してチャンクに配布し、パラレル接続を使用してチャンクをターゲット MySQL サーバーにアップロードします。 このユーティリティは、LOAD DATA ステートメントを使用した標準のシングルスレッドアップロードよりも数回高速に大規模データインポートを完了できます。

パラレルテーブルインポートユーティリティを実行する場合、データファイルのフィールドと MySQL テーブルのカラムの間のマッピングを指定します。 LOAD DATA ステートメントの場合と同様に、フィールド処理および行処理のオプションを設定して、任意の形式でデータファイルを処理できます。 複数のファイルの場合、すべてのファイルは同じ形式である必要があります。 ユーティリティのデフォルト言語は、そのステートメントのデフォルト設定を使用して SELECT...INTO OUTFILE ステートメントを使用して作成されたファイルにマップされます。 ユーティリティには、CSV ファイル (DOS または UNIX システムで作成された)、TSV ファイルおよび JSON の標準データ形式にマップする事前設定済のダイアレクトもあり、必要に応じてフィールド処理および行処理オプションを使用してこれらをカスタマイズできます。 JSON データは、document-per-line 形式である必要があります。

パラレルテーブルインポートユーティリティが導入された後にいくつかの関数が追加されているため、最新バージョンの MySQL Shell を使用してユーティリティを完全に機能させてください。

入力前処理
: MySQL Shell 8.0.22 から、パラレルテーブルインポートユーティリティは、LOAD DATA ステートメントと同じ方法で、入力前処理のためにデータファイルからカラムを取得できます。 選択したデータを破棄するか、データを変換してターゲットテーブルのカラムに割り当てることができます。

Oracle Cloud Infrastructure Object Storage インポート
: MySQL Shell 8.0.20 までは、クライアントホストからローカルディスクとしてアクセス可能な場所からデータをインポートする必要があります。 MySQL Shell 8.0.21 から、osBucketName オプションで指定された Oracle Cloud Infrastructure Object Storage バケットからデータをインポートすることもできます。

複数のデータファイルのインポート
: MySQL Shell 8.0.22 まで、パラレルテーブルインポートユーティリティでは、単一の入力データファイルを単一のリレーショナルテーブルにインポートできます。 MySQL Shell 8.0.23 からは、このユーティリティは指定されたファイルのリストをインポートすることもでき、ワイルドカードパターンマッチングをサポートしてすべての関連ファイルを場所から含めます。 ユーティリティの単一の実行によってアップロードされた複数のファイルは、単一のリレーショナルテーブルに配置されるため、たとえば、複数のホストからエクスポートされたデータを、分析に使用する単一のテーブルにマージできます。

圧縮ファイル処理
: MySQL Shell 8.0.21 までは、パラレルテーブルインポートユーティリティは、圧縮されていない入力データファイルのみを受け入れます。 ユーティリティは、データファイルを分析してチャンクに配布し、チャンクをターゲット MySQL サーバーのリレーショナルテーブルにアップロードして、チャンクをパラレル接続間で分割します。 MySQL Shell 8.0.22 から、ユーティリティは、gzip (.gz) および zstd (.zst) 形式で圧縮されたデータファイルを受け入れて、ファイル拡張子に基づいてフォーマットを自動的に検出することもできます。 ユーティリティは、圧縮形式でストレージから圧縮ファイルをアップロードし、転送のその部分の帯域幅を節約します。 圧縮ファイルはチャンクに分散できないため、かわりにユーティリティはパラレル接続を使用して複数のファイルを解凍し、ターゲットサーバーに同時にアップロードします。 入力データファイルが 1 つしかない場合、圧縮ファイルのアップロードで使用できるのは 1 つの接続のみです。

MySQL Shell のパラレルテーブルインポートユーティリティでは、MySQL Shell テーブルエクスポートユーティリティからの出力がサポートされています。このユーティリティでは、生成されたデータファイルを出力として圧縮し、ローカルフォルダまたはオブジェクトストレージバケットにエクスポートできます。 パラレルテーブルインポートユーティリティのデフォルトの言語は、テーブルエクスポートユーティリティによって生成される出力ファイルの

デフォルトです。 パラレルテーブルインポートユーティリティを使用して、他のソースからファイルをアップロードすることもできます。

MySQL Shell ダンプロードユーティリティ util.loadDump() は、MySQL Shell インスタンスダンプユーティリティ util.dumpInstance()、スキーマダンプユーティリティ util.dumpSchemas() およびテーブルダンプユーティリティ util.dumpTables() によって生成されたチャンク出力ファイルとメタデータの組合せをインポートするように設計されています。 ターゲットサーバーにアップロードする前にチャンク出力ファイルのデータを変更する場合は、パラレルテーブルインポートユーティリティをダンプロードユーティリティと組み合せて使用できます。 これを行うには、まずダンプロードユーティリティを使用して、選択したテーブルの DDL のみをロードし、ターゲットサーバーにテーブルを作成します。 次に、パラレルテーブルインポートユーティリティを使用して、テーブルの出力ファイルからデータを取得および変換し、ターゲットテーブルにインポートします。 必要に応じて、データを変更する他のテーブルに対してこのプロセスを繰り返します。 最後に、ダンプロードユーティリティを使用して、変更しない残りのテーブル (変更したテーブルを除く) の DDL およびデータをロードします。 手順の詳細は、ダンプしたデータの変更 を参照してください。

パラレルテーブルインポートユーティリティには、ターゲット MySQL サーバーへの既存の クラシック MySQL プロトコル 接続が必要です。 各スレッドは、独自のセッションを開いてデータのチャンクを MySQL サーバーに送信するか、圧縮ファイルの場合は複数のファイルを並行して送信します。 スレッド数、各チャンクで送信されるバイト数およびスレッド当たりのデータ転送の最大速度を調整して、ネットワーク上の負荷とデータ転送の速度を均衡化できます。 ユーティリティは、LOAD DATA ステートメントをサポートしていない X プロトコル 接続を介して動作することはできません。

インポートするデータファイルは、次のいずれかの場所にある必要があります:

- クライアントホストからローカルディスクとしてアクセス可能な場所。
- URL で指定された、HTTP または HTTPS を介してクライアントホストにアクセス可能なリモートの場所。 パターン一致は、この方法でアクセスされるファイルではサポートされていません。
- Oracle Cloud Infrastructure Object Storage バケット (MySQL Shell 8.0.21 から)。

データは、アクティブな MySQL セッションが接続されている MySQL サーバーの単一のリレーショナルテーブルにインポートされます。

パラレルテーブルインポートユーティリティでは、LOAD DATA LOCAL INFILE ステートメントを使用してデータをアップロードするため、ターゲットサーバーで local_infile システム変数を ON に設定する必要があります。 これを行うには、パラレルテーブルインポートユーティリティを実行する前に、SQL モードで次のステートメントを発行します:

```
SET GLOBAL local_infile = 1;
```

LOAD DATA LOCAL でセキュリティ上の既知の潜在的な問題を回避するために、MySQL サーバーがファイル転送リクエストを含むパラレルテーブルインポートユーティリティの LOAD DATA リクエストに応答すると、ユーティリティは事前に決定されたデータチャンクのみを送信し、サーバーによって試行された特定のリクエストは無視します。 詳細は、LOAD DATA LOCAL のセキュリティー上の考慮事項を参照してください。

関数

MySQL Shell API では、パラレルテーブルインポートユーティリティは util グローバルオブジェクトの関数であり、次のシグネチャを持ちます:

```
importTable ({file_name | file_list}, options)
```

file_name は、インポートするデータを含む単一ファイルの名前とパスを指定する文字列です。 または、file_list は、複数のデータファイルを指定するファイルパスの配列です。 Windows では、ファイルパスでバックスラッシュをエスケープする必要があります。または、代わりにスラッシュを使用できます。

- ローカルディスク上のクライアントホストからアクセス可能なファイルの場合、ディレクトリパスに接頭辞として file://スキーマを付けるか、デフォルトのスキーマを使用できます。 この方法でアクセスされるファイルの場合、ファイルパスには、パターン一致のためのワイルドカード * (複数文字) および ? (単一文字) を含めることができます。

- HTTP または HTTPS を介してクライアントホストにアクセスできるファイルの場合は、必要に応じて、URL または URL のリストを接頭辞として http://または https://スキーマを付けて http[s]://host.domain[:port]/path の形式で指定します。 この方法でアクセスされるファイルの場合、パターン一致は使用できません。 HTTP サーバーは Range リクエストヘッダーをサポートし、Content-Range レスポンスヘッダーをクライアントに返す必要があります。
- Oracle Cloud Infrastructure Object Storage バケット内のファイルの場合は、バケット内のファイルへのパスを指定し、osBucketName オプションを使用してバケット名を指定します。

options はインポートオプションのディクショナリで、空の場合は省略できます。 オプションは、例の後にリストされています。

この関数は、void、またはエラーの場合は例外を返します。 インポートが Ctrl+C を持つユーザーによって途中で停止された場合、またはエラーが発生した場合、ユーティリティはデータの送信を停止します。 サーバーが受信したデータの処理を終了すると、その時点で各スレッドによってインポートされていたチャンク、完了率およびターゲットテーブルで更新されたレコード数を示すメッセージが返されます。

例

次の例では、MySQL ShellJavaScript モードの最初の例と MySQL ShellPython モードの次の例で、単一の CSV ファイル/tmp/productrange.csv のデータを mydb データベースの products テーブルにインポートし、ファイルのヘッダー行をスキップします:

```
mysql-js> util.importTable("/tmp/productrange.csv", {schema: "mydb", table: "products", dialect: "csv-unix", skipRows: 1, showProgress: true})
```

```
mysql-py> util.import_table("/tmp/productrange.csv", {"schema": "mydb", "table": "products", "dialect": "csv-unix", "skipRows": 1, "showProgress": True})
```

次の MySQL Shell Python モードの例では、CSV ファイルの言語のみを指定します。mydb は、MySQL Shell セッションのアクティブなスキーマです。 したがって、ユーティリティは、ファイル/tmp/productrange.csv 内のデータを mydb データベース内の productrange テーブルにインポートします:

```
mysql-py> \use mydb
mysql-py> util.import_table("/tmp/productrange.csv", {"dialect": "csv-unix"})
```

MySQL ShellPython モードの次の例では、個別に名前が付けられたファイル、ワイルドカードパターンマッチングを使用して指定されたファイルの範囲、圧縮されたファイルなどの複数のファイルからデータをインポートします:

```
mysql-py> util.import_table(
    [
        "data_a.csv",
        "data_b*",
        "data_c*",
        "data_d.tsv.zst",
        "data_e.tsv.zst",
        "data_f.tsv.gz",
        "/backup/replica3/2021_01_12/data_g.tsv",
        "/backup/replica3/2021_01_13/*.tsv",
    ],
    {"schema": "mydb", "table": "productrange"}
)
```

パラレルテーブルインポートユーティリティは、mysqlsh コマンドインタフェースを使用してコマンドラインから起動することもできます。 このインタフェースを使用して、次の例のようにユーティリティを起動します:

```
mysqlsh mysql://root:@127.0.0.1:3366 --ssl-mode=DISABLED -- util import-table /r/mytable.dump --schema=mydb --table=regions --bytes-per-chunk=10M --lines
```

複数のデータファイルをインポートする場合、次の例に示すように、ワイルドカードパターンマッチングを使用して指定されたファイルの範囲は、引用符で囲まれていれば MySQL Shellglob パターンマッチングロジックによって展開されます。 それ以外の場合は、mysqlsh コマンドを入力したユーザーシェルのパターン一致ロジックによって展開されます。

```
mysqlsh mysql://root:@127.0.0.1:3366 -- util import-table data_a.csv "data_b*" data_d.tsv.zst --schema=mydb --table=productrange --osBucketName=mybucket
```

mysqlsh コマンドインタフェースを使用してパラレルテーブルインポートユーティリティを起動する場合、配列値は受け入れられないため、columns オプションはサポートされません。したがって、データファイルの入力行には、ターゲットテーブルのすべてのカラムに一致するフィールドが含まれている必要があります。 前述の例に示すように、ラインフィード文字は、この関数をサポートするシェル (bash, ksh, mksh や zsh など) で ANSI-C 引用符を使用

して渡す必要があります。 このインタフェースの詳細は、セクション5.8「API コマンドラインインタフェース」 を参照してください。

オプション

パラレルテーブルインポートユーティリティでは、次のインポートオプションを使用してデータのインポート場所とインポート方法を指定できます:

schema: "db_name"

接続された MySQL サーバー上のターゲットデータベースの名前。 このオプションを省略すると、ユーティリティは、接続 URI 文字列、\use コマンドまたは MySQL Shell オプションで指定された、現在の MySQL Shell セッションで使用されているスキーマ名を識別して使用しようとします。 スキーマ名が指定されておらず、セッションから識別できない場合は、エラーが返されます。

table: "table_name"

ターゲットリレーショナルテーブルの名前。 このオプションを省略すると、テーブル名は拡張子のないデータファイルの名前とみなされます。 ターゲットテーブルがターゲットデータベースに存在する必要があります。

columns: array of column names

ターゲットリレーショナルテーブルのカラムにマップされた順序で指定された、インポートファイルのカラム名を含む文字列の配列。 このオプションは、インポートされたデータにターゲットテーブルのすべてのカラムが含まれていない場合、またはインポートされたデータのフィールドの順序がテーブルのカラムの順序と異なる場合に使用します。 このオプションを省略した場合、入力行にはターゲットテーブルの各カラムに一致するフィールドが含まれると想定されます。

MySQL Shell 8.0.22 からは、このオプションを使用して、LOAD DATA ステートメントの場合と同じ方法で、入力前処理のためにインポートファイルからカラムを取得できます。 配列内のカラム名のかわりに整数値を使用すると、インポートファイル内のそのカラムは、@1 などのユーザー変数@int として取得されます。 選択したデータを破棄するか、decodeColumns オプションを使用してデータを変換し、ターゲットテーブルのカラムに割り当てることができます。

MySQL ShellJavaScript モードのこの例では、インポートファイルの 2 番目と 4 番目のカラムがユーザー変数@1 および@2 に割り当てられ、ターゲットテーブルのどのカラムにも割り当てる decodeColumns オプションはないため、これらは破棄されます。

```
mysql-js> util.importTable('file.txt', {
          table: 't1',
          columns: ['column1', 1, 'column2', 2, 'column3']
          });
```

decodeColumns: dictionary

columns オプションによってユーザー変数として取得されたインポートファイルカラムをターゲットテーブルのカラムに割り当て、LOAD DATA ステートメントの SET 句と同じ方法で前処理変換を指定する、キーと値のペアのディクショナリ。 このオプションは、MySQL Shell 8.0.22 から使用できます。

MySQL ShellJavaScript モードのこの例では、データファイルの最初の入力カラムがターゲットテーブルの最初のカラムとして使用されます。 columns オプションによって変数@1 に割り当てられた 2 番目の入力カラムは、ターゲットテーブルの 2 番目のカラムの値として使用される前に除算操作の対象となります。

```
mysql-js> util.importTable('file.txt', {
          columns: ['column1', 1],
          decodeColumns: {'column2': '@1 / 100'}
          });
```

MySQL ShellJavaScript モードのこの例では、データファイルの入力カラムが両方とも変数に割り当てられ、様々な方法で変換されて、ターゲットテーブルのカラムへの移入に使用されます:

```
mysql-js> util.importTable('file.txt', {
          table: 't1',
          columns: [1, 2],
```

```
        decodeColumns: {
          'a': '@1',
          'b': '@2',
          'sum': '@1 + @2',
          'multiple': '@1 * @2',
          'power': 'POW(@1, @2)'
        }
      });
```

skipRows: number

インポートファイルの先頭、または複数のインポートファイルの場合はファイルリストに含まれるすべてのファイルの先頭で、この数のデータ行をスキップします。このオプションを使用して、アップロードからテーブルへのカラム名を含む初期ヘッダー行を省略できます。 デフォルトでは、行はスキップされません。

replaceDuplicates: [true|false]

既存の行と同じ値または一意インデックスを持つ入力行を置換する (true) かスキップする (false) か。 デフォルトは false です。

dialect: [default|csv|csv-unix|tsv|json]

指定したファイル形式に適した一連のフィールド処理および行処理オプションを使用します。 linesTerminatedBy, fieldsTerminatedBy, fieldsEnclosedBy, fieldsOptionallyEnclosed および fieldsEscapedBy オプションのいずれかまたは複数を指定して設定を変更することで、選択した言語をさらにカスタマイズするためのベースとして使用できます。 デフォルトの言語は、そのステートメントのデフォルト設定で SELECT...INTO OUTFILE ステートメントを使用して作成されたファイルにマップされます。 これは、MySQL Shell テーブルエクスポートユーティリティによって生成される出力ファイルのデフォルトです。 CSV ファイル (DOS または UNIX システムで作成)、TSV ファイルおよび JSON データに適したその他のダイアレクトを使用できます。 各言語に適用される設定は次のとおりです:

表 8.2 パラレルテーブルインポートユーティリティの言語設定

dialect	linesTerminatedBy	fieldsTerminatedBy	fieldsEnclosedBy	fieldsOptionallyEnclosed	fieldsEscapedBy
default	[LF]	[TAB]	[空]	false	\
csv	[CR][LF]	,	"	true	\
csv-unix	[LF]	,	"	false	\
tsv	[CR][LF]	[TAB]	"	true	\
json	[LF]	[LF]	[空]	false	[空]

注記

1. ダイアレクトのキャリッジリターンおよびラインフィードの値は、オペレーティングシステムに依存しません。
2. linesTerminatedBy, fieldsTerminatedBy, fieldsEnclosedBy, fieldsOptionallyEnclosed および fieldsEscapedBy オプションを使用する場合、コマンドインタプリタのエスケープ規則に応じて、バックスラッシュ文字 (\) をオプション値で使用する場合は二重にする必要があります。
3. LOAD DATA ステートメントを使用した MySQL サーバーと同様に、MySQL Shell では、指定したフィールド処理および行処理オプションは検証されません。これらのオプションを正しく選択しないと、データが誤ったフィールドに部分的にインポートされたり、正しくインポートされない場合があります。 インポートを開始する前に必ず設定を確認し、後で結果を確認してください。

linesTerminatedBy: "characters"

入力データファイルの各行を終了する 1 つ以上の文字 (または空の文字列)。 デフォルトは、指定された方言、または方言オプションが省略されている場合は

	改行文字 (\n) です。 このオプションは、LOAD DATA ステートメントの LINES TERMINATED BY オプションと同等です。 ユーティリティでは、空の文字列に設定されている LOAD DATA ステートメントの LINES STARTING BY オプションに相当するものは提供されないことに注意してください。
fieldsTerminatedBy: "characters"	入力データファイルの各フィールドを終了する 1 つ以上の文字 (または空の文字列)。 デフォルトは、指定された言語、または言語オプションが省略されている場合はタブ文字 (\t) です。 このオプションは、LOAD DATA ステートメントの FIELDS TERMINATED BY オプションと同等です。
fieldsEnclosedBy: "character"	入力データファイルの各フィールドを囲む単一の文字 (または空の文字列)。 デフォルトは、指定された言語に対するものであり、dialect オプションが省略されている場合は空の文字列です。 このオプションは、LOAD DATA ステートメントの FIELDS ENCLOSED BY オプションと同等です。
fieldsOptionallyEnclosed: [true | false]	fieldsEnclosedBy に指定された文字が入力データファイル (false) 内のすべてのフィールドを囲むか、場合によってのみフィールドを囲むか (true)。 デフォルトは、指定された言語、または dialect オプションが省略されている場合は false です。 このオプションにより、fieldsEnclosedBy オプションは LOAD DATA ステートメントの FIELDS OPTIONALLY ENCLOSED BY オプションと同等になります。
fieldsEscapedBy: "character"	入力データファイル内のエスケープシーケンスを開始する文字。 これを指定しない場合、エスケープシーケンスの解釈は行われません。 デフォルトは、指定された言語の場合と同様です。または、dialect オプションが省略されている場合はバックスラッシュ (\)) です。 このオプションは、LOAD DATA ステートメントの FIELDS ESCAPED BY オプションと同等です。
osBucketName: "string"	MySQL Shell 8.0.21 に追加されました。 入力データファイルが存在する Oracle Cloud Infrastructure Object Storage バケットの名前。 デフォルトでは、~/.oci/config にある Oracle Cloud Infrastructure CLI 構成ファイルの[DEFAULT]プロファイルを使用して、バケットへの接続が確立されます。 ociConfigFile および ociProfile オプションを使用して、接続に使用される代替プロファイルを置換できます。 CLI 構成ファイルの設定手順については、「SDK および CLI 構成ファイル」を参照してください。
osNamespace: "string"	MySQL Shell 8.0.21 に追加されました。 osBucketName によって指定されたオブジェクトストレージバケットが配置される Oracle Cloud Infrastructure ネームスペース。 オブジェクトストレージバケットのネームスペースは、Oracle Cloud Infrastructure コンソールのバケット詳細ページの「バケット情報」タブに表示されるか、Oracle Cloud Infrastructure コマンドラインインタフェースを使用して取得できます。
ociConfigFile: "string"	MySQL Shell 8.0.21 に追加されました。 デフォルトの場所の ~/.oci/config ではなく、接続に使用するプロファイルを含む Oracle Cloud Infrastructure CLI 構成ファイル。
ociProfile: "string"	MySQL Shell 8.0.21 に追加されました。 接続に使用される Oracle Cloud Infrastructure CLI 構成ファイル内の[DEFAULT]プロファイルではなく、接続に使用する Oracle Cloud Infrastructure プロファイルのプロファイル名。
characterSet: "charset"	MySQL Shell 8.0.21 に追加されました。 このオプションでは、インポート時に入力データが解釈される文字セットエンコーディングを指定します。 このオプションを binary に設定すると、インポート中に変換は行われません。 このオプションを省略すると、インポートでは、character_set_database システム変数で指定された文字セットを使用して入力データが解釈されます。
bytesPerChunk: "size"	複数の入力データファイルのリストの場合、このオプションは使用できません。 単一の入力データファイルの場合、このオプションでは、スレッドがターゲットサーバーへの LOAD DATA コールごとに送信するバイト数 (および行の終わりに到達するために必要な追加バイト数) を指定します。 このユーティリティは、スレッドがターゲットサーバーに取得して送信するために、データをこのサイズのチャンクに分散します。 チャンクサイズは、バイト数で指定するか、接尾辞 k (キ

	口バイト)、M (メガバイト)、G (ギガバイト) を使用して指定できます。 たとえば、bytesPerChunk="2k"では、スレッドは約 2 キロバイトのチャンクを送信します。 最小チャンクサイズは 131072 バイトで、デフォルトのチャンクサイズは 50M です。
threads: number	入力ファイルのデータをターゲットサーバーに送信するために使用するパラレルスレッドの最大数。 スレッド数を指定しない場合、デフォルトの最大値は 8 です。 複数の入力データファイルのリストについては、指定した数または最大数のスレッドが作成されます。 単一の入力データファイルの場合、ユーティリティは次の式を使用して、この最大数までの適切なスレッド数を計算します: `min{max{1, threads}, chunks}}` ここで、threads はスレッドの最大数、chunks はデータが分割されるチャンクの数で、ファイルサイズを bytesPerChunk サイズで除算してから 1 を加算して計算されます。 この計算により、スレッドの最大数が実際に送信されるチャンクの数を超えた場合、ユーティリティは必要以上のスレッドを作成しません。 圧縮されたファイルはチャンクに配布できないため、ユーティリティはそのかわりにパラレル接続を使用して複数のファイルを一度にアップロードします。 入力データファイルが 1 つしかない場合、圧縮ファイルのアップロードで使用できるのは 1 つの接続のみです。
maxRate: "rate"	データスループットの最大制限 (バイト/秒/スレッド)。 クライアントホストまたはターゲットサーバーのネットワーク、I/O または CPU の飽和を回避する必要がある場合は、このオプションを使用します。 最大速度はバイト数で指定することも、接尾辞 k (キロバイト)、M (メガバイト)、G (ギガバイト) を使用して指定することもできます。 たとえば、maxRate="5M"では、各スレッドが 5MB のデータ/秒に制限され、8 つのスレッドで 40MB/秒の転送速度が得られます。 デフォルトは 0 で、これは制限がないことを意味します。
showProgress: [true | false]	インポートの進行状況情報を表示 (true) または非表示 (false) します。 デフォルトは、stdout が端末 (tty) の場合は true、それ以外の場合は false です。

8.5 インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティ

MySQL Shell インスタンスダンプユーティリティ util.dumpInstance() およびスキーマダンプユーティリティ util.dumpSchemas() は、MySQL Shell 8.0.21 で導入され、オンプレミス MySQL インスタンスから Oracle Cloud Infrastructure Object Storage バケットまたはローカルファイルのセットへのすべてのスキーマまたは選択したスキーマのエクスポートをサポートしています。 MySQL Shell 8.0.22 で導入されたテーブルダンプユーティリティ util.dumpTables() では、スキーマから選択したテーブルまたはビューに対して同じ操作がサポートされます。 エクスポートした項目は、MySQL Shellセクション8.6「ダンプロードユーティリティ」util.loadDump() を使用して、MySQL Database Service DB システム (短縮形は MySQL DB システム) または MySQL Server インスタンスにインポートできます。

MySQL Shell インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティは、Oracle Cloud Infrastructure Object Storage ストリーミング、MySQL Database Service の互換性チェックと変更、複数スレッドによるパラレルダンプ、および mysqldump で提供されていないファイル圧縮を提供します。 ダンプ中に進捗情報が表示されます。 選択したダンプオプションのセットでドライランを実行して、実行されるアクション、ダンプされる項目、および (インスタンスダンプユーティリティおよびスキーマダンプユーティリティの) これらのオプションを使用して実際にユーティリティを実行するときに修正する必要がある MySQL Database Service の互換性の問題に関する情報を表示できます。

ダンプファイルの宛先を選択する場合、MySQL DB システムにインポートするには、ダンプロードユーティリティを実行する MySQL Shell インスタンスが、MySQL DB システムにアクセスできる Oracle Cloud Infrastructure Compute インスタンスにインストールされている必要があります。 インスタンス、スキーマまたはテーブルをオブジェクトストレージバケットにダンプする場合、コンピュートインスタンスからオブジェクトストレージバケットにアクセスできます。 ローカルシステムにダンプファイルを作成する場合、コンピュートインスタンスに選択したオペレーティングシステムに応じて、選択したコピーユーティリティを使用して Oracle Cloud Infrastructure Compute インスタンスにダンプファイルを転送する必要があります。

MySQL Shell インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティによって作成されるダンプは、スキーマ構造を指定する DDL ファイルと、データを含むタブ区切りの .tsv ファイルで構成されます。 エクスポートされたスキーマにエクスポートされたデータを移入することとは別に、エクスポートされたスキーマを設定する場合は、DDL ファイルのみまたはデータファイルのみを生成することもできます。 データ整合性のためにダンプ中にバックアップのためにインスタンスをロックするかどうかを選択できます。 デフォルトでは、ダンプユーティリティはテーブルデータを複数のデータファイルにチャンク化し、ファイルを圧縮します。

MySQL インスタンスのスキーマの大部分を代替方法としてダンプする必要がある場合は、スキーマダンプユーティリティではなくインスタンスダンプユーティリティを使用して、ダンプしないスキーマをリストする excludeSchemas オプションを指定できます。 同様に、スキーマ内のほとんどのテーブルをダンプする必要がある場合は、テーブルダンプユーティリティではなく、excludeTables オプションを指定してスキーマダンプユーティリティを使用できます。 information_schema, mysql, ndbinfo, performance_schema および sys スキーマは、常にインスタンスダンプから除外されます。 mysql.apply_status, mysql.general_log, mysql.schema テーブルおよび mysql.slow_log テーブルのデータは、DDL ステートメントは含まれますが、常にスキーマダンプから除外されます。 ユーザーとそのロール、権限付与、イベント、ルーチンおよびトリガーを含めるか除外するかを選択することもできます。

デフォルトでは、ダンプ出力のすべてのタイムスタンプデータでタイムゾーンが UTC に標準化されるため、異なるタイムゾーンを持つサーバー間でのデータの移動および複数のタイムゾーンを持つデータの処理が容易になります。 必要に応じて、tzUtc: false オプションを使用して元のタイムスタンプを保持できます。

MySQL Shell 8.0.22 から、インスタンスまたはスキーマを Oracle Cloud Infrastructure Object Storage バケットにエクスポートするときに、ダンプ中に各アイテムに対して事前認証済リクエスト URL を生成できます。 MySQL Shell ダンプロードユーティリティ util.loadDump() を実行するユーザーアカウントは、これらを使用して、追加のアクセス権限なしでダンプファイルをロードします。 デフォルトでは、ocimds オプションが true に設定されており、osBucketName オプションを使用してオブジェクトストレージバケット名が指定されている場合、MySQL Shell インスタンスダンプユーティリティおよびスキーマダンプユーティリティはダンプファイルの事前認証済リクエスト URL を生成し、単一のマニフェストファイルにリストします。 ダンプロードユーティリティはマニフェストファイルを参照して URL を取得し、ダンプファイルをロードします。 事前認証済リクエスト URL を生成または非アクティブ化する手順は、ociParManifest オプションの説明を参照してください。

インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティを使用するダンプには、次の要件が適用されます:

- MySQL 5.7 以上は、ソース MySQL インスタンスとアップグレード先 MySQL インスタンスの両方に必要です。
- インスタンスまたはスキーマ内のオブジェクト名は、latin1 または utf8 の文字セットである必要があります。
- データの整合性は、InnoDB ストレージエンジンを使用するテーブルに対してのみ保証されます。
- ユーティリティの実行に使用されるユーザーアカウントが、関連するすべてのスキーマに対して持っている必要がある最小限の権限セットは、次のとおりです: BACKUP_ADMIN, EVENT, RELOAD, SELECT, SHOW VIEW および TRIGGER。 consistent オプションが false に設定されている場合、BACKUP_ADMIN および RELOAD 権限は必要ありません。 consistent オプションが true(デフォルト) に設定されている場合、ダンプされたすべてのテーブルに対する LOCK TABLES 権限は、RELOAD 権限のかわりに使用できます (後者が使用できない場合)。
- Oracle Cloud Infrastructure Object Storage バケットへのファイルの転送に使用されるアップロード方法のファイルサイズ制限は、1.2 TiB です。 MySQL Shell 8.0.21 では、マルチパートサイズ設定は、複数のファイルパートの数値制限が最初に適用され、約 640 GB の制限が作成されることを意味します。 MySQL Shell 8.0.22 から、マルチパートサイズの設定が変更され、フルファイルサイズ制限が可能になりました。
- ユーティリティは、安全でないデータ型のカラムをテキスト形式 (BLOB など) で Base64 に変換します。 したがって、これらのカラムのサイズは、ターゲットの MySQL インスタンスで構成されている max_allowed_packet システム変数の値の約 0.74 倍 (バイト単位) を超えないようにする必要があります。
- テーブルダンプユーティリティの場合、エクスポートされたビューおよびトリガーは、他のビューまたはテーブルの参照に修飾名を使用しないでください。
- MySQL DB システムにインポートする場合は、MySQL Database Service との互換性を確保するために、ocimds オプションを true に設定します。
- MySQL Database Service との互換性のために、すべてのテーブルで InnoDB ストレージエンジンを使用する必要があります。 ocimds オプションはダンプで見つかった例外をチェックし、compatibility オプションはダンプファイルを変更してほかのストレージエンジンを InnoDB に置き換えます。

- インスタンスダンプユーティリティおよびスキーマダンプユーティリティでは、MySQL Database Service との互換性のために、インスタンスまたはスキーマ内のすべてのテーブルが MySQL データディレクトリにあり、デフォルトのスキーマ暗号化を使用する必要があります。 ocimds オプションは、これらの要件を適用するようにダンプファイルを変更します。
- MySQL Database Service との互換性のために、テーブルスペースや権限などの項目には、他の多くのセキュリティ関連の制限および要件が適用されます。 ocimds オプションはダンプ中に見つかった例外をチェックし、compatibility オプションはダンプファイルを自動的に変更して互換性の問題の一部を解決します。 一部の変更は手動で行う必要がある (または望ましい) 場合があります。 詳細は、compatibility オプションの説明を参照してください。

インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティは、MySQL Shell グローバルセッションを使用して、エクスポートを実行するターゲット MySQL サーバーの接続詳細を取得します。 いずれかのユーティリティを実行する前に、(X プロトコル 接続または クラシック MySQL プロトコル 接続を持つことができる) グローバルセッションをオープンする必要があります。 ユーティリティはスレッドごとに独自のセッションを開き、接続圧縮や SSL オプションなどのオプションをグローバルセッションからコピーし、グローバルセッションをこれ以上使用しません。

MySQL Shell API では、インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティはutilグローバルオブジェクトの関数であり、次のシグネチャを持ちます:

```
util.dumpInstance(outputUrl[, options])
util.dumpSchemas(schemas, outputUrl[, options])
util.dumpTables(schema, tables, outputUrl[, options])
```

スキーマダンプユーティリティの場合、schemas は、MySQL インスタンスからダンプするスキーマのリストを指定します。

テーブルダンプユーティリティの場合、schema はダンプする項目を含むスキーマを指定し、tables はダンプするテーブルまたはビューを指定する文字列の配列です。 MySQL Shell 8.0.23 からは、ダンプロードユーティリティの schema オプションを使用して別のターゲットスキーマにロードできますが、テーブルダンプにはターゲットの MySQL インスタンスで指定されたスキーマを設定するために必要な情報が含まれます。 MySQL Shell 8.0.22 にはスキーマ情報が含まれていないため、このユーティリティで生成されたダンプファイルを既存のターゲットスキーマにロードする必要があります。

ローカルファイルシステムにダンプする場合、outputUrl はダンプファイルを配置するローカルディレクトリへのパスを指定する文字列です。 絶対パスまたは現在の作業ディレクトリからの相対パスを指定できます。 ローカルディレクトリパスの前に file:///スキーマを付けることができます。 この例では、接続された MySQL インスタンスがローカルディレクトリにダンプされますが、MySQL Database Service との互換性のためにダンプファイルにいくつかの変更が加えられています。 ユーザーは最初に予行演習を実行してスキーマを検査し、互換性の問題を表示してから、適切な互換性オプションを適用してダンプを実行し、問題を削除します:

```
shell-js> util.dumpInstance("C:/Users/hanna/worlddump", {dryRun: true, ocimds: true})
Checking for compatibility with MySQL Database Service 8.0.21
...
Compatibility issues with MySQL Database Service 8.0.21 were found. Please use the
'compatibility' option to apply compatibility adaptations to the dumped DDL.
Util.dumpInstance: Compatibility issues were found (RuntimeError)
shell-js> util.dumpInstance("C:/Users/hanna/worlddump", {
      > ocimds: true, compatibility: ["strip_definers", "strip_restricted_grants"]})
```

エクスポートを実行する前に、ターゲットディレクトリを空にする必要があります。 ディレクトリがまだ親ディレクトリに存在しない場合は、ユーティリティによって作成されます。 ローカルディレクトリへのエクスポートの場合、ダンプ中に作成されたディレクトリはアクセス権限 rwxr-x--- で作成され、ファイルはアクセス権限 rw-r----- (これらがサポートされているオペレーティングシステム上) で作成されます。 ファイルおよびディレクトリの所有者は、MySQL Shell を実行しているユーザーアカウントです。

テーブルダンプユーティリティを使用すると、スキーマ間でテーブルを転送する場合など、スキーマから個々のテーブルを選択できます。 MySQL ShellJavaScript モードのこの例では、hr スキーマのテーブル employees および salaries がローカルディレクトリ emp にエクスポートされ、ユーティリティによって現在の作業ディレクトリに作成されます:

```
shell-js> util.dumpTables("hr", [ "employees", "salaries" ], "emp")
```

Oracle Cloud Infrastructure Object Storage バケットにダンプする場合、outputUrl は、ディレクトリ構造をシミュレートするためにバケット内のダンプファイルの接頭辞として使用されるパスです。 osBucketName オプションを使用し

てオブジェクトストレージバケットの名前を指定し、osNamespace オプションを使用してバケットのネームスペースを識別します。 MySQL ShellPython モードのこの例では、ユーザーは world スキーマを接続された MySQL インスタンスから Object Storage バケットにダンプしますが、前の例と同じ互換性の変更があります:

```
shell-py> util.dump_schemas(["world"], "worlddump", {
       > "osBucketName": "hanna-bucket", "osNamespace": "idx28w1ckztq",
       > "ocimds": "true", "compatibility": ["strip_definers", "strip_restricted_grants"]})
```

オブジェクトストレージバケットでは、ダンプファイルはすべて接頭辞 worlddump とともに表示されます。次に例を示します:

```
worlddump/@.done.json
worlddump/@.json
worlddump/@.post.sql
worlddump/@.sql
worlddump/world.json
worlddump/world.sql
worlddump/world@city.json
worlddump/world@city.sql
worlddump/world@city@@0.tsv.zst
worlddump/world@city@@0.tsv.zst.idx
...
```

オブジェクトストレージバケットのネームスペースは、Oracle Cloud Infrastructure コンソールのバケット詳細ページの「バケット情報」タブに表示されるか、Oracle Cloud Infrastructure コマンドラインインタフェースを使用して取得できます。 オブジェクトストレージバケットへの接続は、デフォルトの Oracle Cloud Infrastructure CLI 構成ファイルのデフォルトプロファイル、または ociConfigFile および ociProfile オプションを使用して指定する代替詳細を使用して確立されます。 CLI 構成ファイルを設定する手順については、「SDK および CLI 構成ファイル」を参照してください

options はオプションのディクショナリで、空の場合は省略できます。 特に指定がないかぎり、インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティには次のオプションを使用できます:

dryRun: [true | false]	指定されたオプションセットでダンプされる内容、および MySQL Database Service の互換性チェックの結果 (ocimds オプションが指定されている場合) に関する情報を表示しますが、ダンプは続行しません。 このオプションを設定すると、ダンプを開始する前に互換性の問題をすべてリストできます。 デフォルトは false です。
osBucketName: "string"	ダンプが書き込まれる Oracle Cloud Infrastructure Object Storage バケットの名前。 デフォルトでは、~/.oci/config にある Oracle Cloud Infrastructure CLI 構成ファイルの[DEFAULT]プロファイルを使用して、バケットへの接続が確立されます。 ociConfigFile および ociProfile オプションを使用して、接続に使用される代替プロファイルを置換できます。 CLI 構成ファイルの設定手順については、「SDK および CLI 構成ファイル」を参照してください。
osNamespace: "string"	osBucketName によって指定されたオブジェクトストレージバケットが配置される Oracle Cloud Infrastructure ネームスペース。 オブジェクトストレージバケットのネームスペースは、Oracle Cloud Infrastructure コンソールのバケット詳細ページの「バケット情報」タブに表示されるか、Oracle Cloud Infrastructure コマンドラインインタフェースを使用して取得できます。
ociConfigFile: "string"	デフォルトの場所の ~/.oci/config ではなく、接続に使用するプロファイルを含む Oracle Cloud Infrastructure CLI 構成ファイル。
ociProfile: "string"	接続に使用される Oracle Cloud Infrastructure CLI 構成ファイル内の[DEFAULT]プロファイルではなく、接続に使用する Oracle Cloud Infrastructure プロファイルのプロファイル名。
threads: int	MySQL インスタンスからデータのチャンクをダンプするために使用するパラレルスレッドの数。 各スレッドは、MySQL インスタンスへの独自の接続を持ちます。 デフォルトは 4 です。
maxRate: "string"	ダンプ中のデータ読取りスループットのスレッド当たりの最大バイト数/秒。 単位接尾辞 k (キロバイト)、M (メガバイト)、および G (ギガバイト) を使用できます

(たとえば、100M を設定すると、スループットがスレッド当たり 100 メガバイト/秒に制限されます)。 0 (デフォルト値) を設定するか、オプションを空の文字列に設定すると、制限は設定されません。

showProgress: [true | false]
: ダンプの進行状況情報を表示 (true) または非表示 (false) にします。 MySQL Shell が対話型モードの場合など、stdout が端末 (tty) の場合、デフォルトは true です。 それ以外の場合は false です。 進捗情報には、ダンプされる行の推定合計数、これまでにダンプされた行数、完了率およびスループット (行およびバイト/秒) が含まれます。

compression: "string"
: ダンプのデータファイルを書き込むときに使用する圧縮タイプ。 デフォルトでは、zstd 圧縮 (zstd) が使用されます。 または、gzip 圧縮 (gzip) を使用するか、圧縮なし (none) を使用します。

excludeSchemas: array of strings
: (インスタンスダンプユーティリティのみ)ダンプから名前付きスキーマを除外します。 information_schema, mysql, ndbinfo, performance_schema および sys スキーマは、常にインスタンスダンプから除外されることに注意してください。 名前付きスキーマが存在しないか除外されている場合、ユーティリティはその項目を無視します。

excludeTables: array of strings
: (インスタンスダンプユーティリティおよびスキーマダンプユーティリティのみ)指定されたテーブルをダンプから除外します。 テーブル名は、有効なスキーマ名で修飾し、必要に応じてバックティック文字で引用符で囲む必要があります。 DDL ステートメントは含まれますが、mysql.apply_status, mysql.general_log, mysql.schema および mysql.slow_log tables のデータは常にスキーマダンプから除外されることに注意してください。 excludeTables オプションで指定されたテーブルには、ダンプ内に DDL ファイルまたはデータファイルがありません。 指定したテーブルがスキーマに存在しない場合、またはスキーマがダンプに含まれていない場合、ユーティリティはその項目を無視します。

all: [true | false]
: (テーブルダンプユーティリティのみ) このオプションを true に設定すると、指定したスキーマのすべてのビューおよびテーブルがダンプに含まれます。 このオプションを使用する場合は、tables パラメータを空の配列に設定します。 デフォルトは false です。

users: [true | false]
: (インスタンスダンプユーティリティのみ) Include (true) または exclude (false) ユーザーとそのロールおよび権限をダンプに含めます。 デフォルトは true であるため、ユーザーはデフォルトで含まれます。 スキーマダンプユーティリティおよびテーブルダンプユーティリティでは、ダンプにユーザー、ロールおよび権限は含まれません。 MySQL Shell 8.0.22 から、excludeUsers または includeUsers オプションを使用して、ダンプファイルに除外または含める個々のユーザーアカウントを指定できます。 これらのオプションを MySQL Shell ダンプロードユーティリティ util.loadDump() とともに使用して、ターゲットの MySQL インスタンスの要件に応じて、インポート時に個々のユーザーアカウントを除外または含めることもできます。

> **注記**
>
> MySQL Shell 8.0.21 では、root ユーザーアカウントまたは別の制限付きユーザーアカウント名がダンプファイルに存在する場合、ユーザーを MySQL DB システムにインポートしようとするとインポートが失敗するため、そのリリースではユーザーの MySQL DB システムへのインポートはサポートされていません。

excludeUsers: array of strings
: (インスタンスダンプユーティリティのみ)指定されたユーザーアカウントをダンプファイルから除外します。 このオプションは MySQL Shell 8.0.22 から使用でき、これを使用して、MySQL DB システムへのインポートが許可されていないユーザーアカウント、またはターゲットの MySQL インスタンスにすでに存在するか不要なユーザーアカウントを除外できます。 各ユーザーアカウント文字列は、ユーザー名とホスト名で定義されたアカウントの場合は"'user_name'@'host_name'"の形式で、ユーザー名のみで定義されたアカウント

	の場合は"'user_name'" ("'user_name'@'%'"と同等) で指定します。 指定されたユーザーアカウントが存在しない場合、ユーティリティはアイテムを無視します。
includeUsers: array of strings	(インスタンスダンプユーティリティのみ)ダンプファイルに指定されたユーザーアカウントのみを含めます。 excludeUsers オプションの場合と同様に、各ユーザーアカウント文字列を指定します。 このオプションは MySQL Shell 8.0.22 から使用でき、ダンプに必要なユーザーアカウントが少ない場合は、excludeUsers のかわりに使用できます。 両方のオプションを指定することもできます。この場合、includeUsers 文字列と excludeUsers 文字列の両方で一致するユーザーアカウントは除外されます。
events: [true | false]	ダンプ内の各スキーマに対する(インスタンスダンプユーティリティおよびスキーマダンプユーティリティのみ) include (true) または exclude (false) イベント。 デフォルトは true です。
routines: [true | false]	ダンプ内の各スキーマに対する(インスタンスダンプユーティリティおよびスキーマダンプユーティリティのみ) include (true) または exclude (false) 関数およびストアドプロシージャ。 デフォルトは true です。 routines が true に設定されている場合でも、ユーザー定義関数は含まれないことに注意してください。
triggers: [true | false]	ダンプ内の各テーブルに (true) トリガーを含めるか、(false) トリガーを除外します。 デフォルトは true です。
defaultCharacterSet: "string"	ダンプのために MySQL Shell によってサーバーに開かれるセッション接続中に使用される文字セット。 デフォルトは utf8mb4 です。 システム変数 character_set_client、character_set_connection および character_set_results のセッション値は、接続ごとにこの値に設定されます。 文字セットは、character_set_client システム変数で許可され、MySQL インスタンスでサポートされている必要があります。
tzUtc: [true | false]	ダンプの先頭にステートメントを含めて、タイムゾーンを UTC に設定します。 ダンプ出力のすべてのタイムスタンプデータは、このタイムゾーンに変換されます。デフォルトは true であるため、タイムスタンプデータはデフォルトで変換されます。 タイムゾーンを UTC に設定すると、異なるタイムゾーンを持つサーバー間でのデータの移動や、複数のタイムゾーンを持つ一連のデータの処理が容易になります。 必要に応じて元のタイムスタンプを保持するには、このオプションを false に設定します。
consistent: [true | false]	ダンプ中にインスタンスをバックアップ用にロックして、一貫性のあるデータダンプを有効 (true) または無効 (false) にします。 デフォルトは true です。 true が設定されている場合、ユーティリティは FLUSH TABLES WITH READ LOCK ステートメントを使用してグローバル読取りロックを設定します。 各スレッドのトランザクションは、SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ および START TRANSACTION WITH CONSISTENT SNAPSHOT ステートメントを使用して開始されます。 すべてのスレッドがトランザクションを開始すると、インスタンスはバックアップ用にロックされ、グローバル読取りロックが解放されます。
ddlOnly: [true | false]	このオプションを true に設定すると、ダンプされた項目の DDL ファイルのみがダンプに含まれ、データはダンプされません。 デフォルトは false です。
dataOnly: [true | false]	このオプションを true に設定すると、ダンプされた項目のデータファイルのみがダンプに含まれ、DDL ファイルは含まれません。 デフォルトは false です。
chunking: [true | false]	各テーブルのデータを複数のファイルに分割するテーブルデータのチャンク化を有効 (true) または無効 (false) にします。 デフォルトは true であるため、チャンク化はデフォルトで有効になっています。 bytesPerChunk を使用してチャンクサイズを指定します。 テーブルデータを別々のファイルにチャンク化するには、テーブルに対して主キーまたは一意インデックスを定義する必要があります。このテーブルは、データを順序付けおよびチャンク化するインデックスカラムを選択するためにユーティリティで使用されます。 テーブルにこれらのいずれも含まれていない場合は、警告が表示され、テーブルデータが単一のファイルに書き込まれます。

チャンクオプションを false に設定した場合、チャンクは実行されず、ユーティリティによってテーブルごとに 1 つのデータファイルが作成されます。

bytesPerChunk: "string"

チャンクが有効な場合に各データファイルに書き込まれるおおよそのバイト数を設定します。 単位接尾辞 k(キロバイト)、M(メガバイト)、G(ギガバイト) を使用できます。 デフォルトは MySQL Shell 8.0.22 (MySQL Shell 8.0.21 では 32 MB) からの 64 MB (64M) で、最小は 128 KB (128k) です。 このオプションを指定すると、chunking は暗黙的に true に設定されます。 このユーティリティは、圧縮が適用される前に、各テーブルのデータをこの量のデータを含むファイルにチャンク化することを目的としています。 チャンクサイズは平均で、テーブル統計および実行計画の見積りに基づいて計算されます。

ocimds: [true | false]

このオプションを true に設定すると、MySQL Database Service との互換性のチェックおよび変更が可能になります。 デフォルトは false です。 MySQL Shell 8.0.23 からは、このオプションはすべてのユーティリティで使用でき、そのリリースより前は、インスタンスダンプユーティリティおよびスキーマダンプユーティリティでのみ使用できます。

このオプションが true, DATA DICTIONARY, INDEX DICTIONARY に設定され、CREATE TABLE ステートメントの ENCRYPTION オプションが DDL ファイルでコメントアウトされている場合、すべてのテーブルが MySQL データディレクトリに配置され、デフォルトのスキーマ暗号化が使用されるようにします。 チェックは、InnoDB 以外の CREATE TABLE ステートメント内のストレージエンジン、ユーザーまたはロールへの不適切な権限の付与、およびその他の互換性の問題に対して実行されます。 非準拠の SQL ステートメントが見つかった場合は、例外が発生し、ダンプが停止されます。 ダンププロセスが開始される前に、dryRun オプションを使用してダンプ内の項目に関するすべての問題をリストします。 compatibility オプションを使用して、ダンプ出力の問題を自動的に修正します。

MySQL Shell 8.0.22 から、このオプションが true に設定され、osBucketName オプションを使用してオブジェクトストレージバケット名が指定されている場合、ociParManifest オプションもデフォルトで true に設定されます。つまり、ダンプ内のすべての項目に対して事前認証済リクエストが生成され、これらのリクエスト URL を使用してのみダンプファイルにアクセスできます。

compatibility: array of strings

ダンプ出力のすべてのテーブルに MySQL Database Service との互換性のために指定された要件を適用し、ダンプファイルを necessary.From MySQL Shell 8.0.23 として変更します。このオプションはすべてのユーティリティで使用でき、そのリリースより前は、インスタンスダンプユーティリティおよびスキーマダンプユーティリティでのみ使用できます。

次の変更は、カンマ区切りリストとして指定できます:

force_innodb	CREATE TABLE ステートメントを使用していないすべてのテーブルに InnoDB ストレージエンジンを使用するように変更します。
skip_invalid_accounts	MySQL Database Service でサポートされていない外部認証プラグインで作成されたユーザーアカウントを削除します。
strip_definers	ビュー、ルーチン、イベントおよびトリガーから DEFINER 句を削除して、これらのオブジェクトがデフォルト定義者 (スキーマを起動するユーザー) で作成されるようにし、ビューおよびルーチンの SQL SECURITY 句を変更して、DEFINER のかわりに INVOKER を指定します。 MySQL Database Service では、スキーマをロードするユーザー以外の定義者を使用してこれらのオブジェクトを作成するには、特別な権限が必要です。 セキュリティモデルで、ビューおよびルーチンにアカウントクエリーま

たはコールする権限よりも多くの権限が必要な場合は、ロードする前にスキーマを手動で変更する必要があります。

strip_restricted_grants
: MySQL Database Service によって制限されている特定の権限を GRANT ステートメントから削除して、ユーザーおよびそのロールにこれらの権限を付与できないようにします (これにより、ユーザーの作成が失敗します)。 MySQL Shell 8.0.22 からは、Oracle Cloud Infrastructure Compute インスタンスの管理ユーザーアカウント自体に関連する権限がない場合、このオプションによってシステムスキーマ (mysql および sys) の REVOKE ステートメントも削除されるため、削除できません。

strip_role_admin
: GRANT ステートメントから ROLE_ADMIN 権限を削除します。 この権限は、MySQL Database Service によって制限できます。

strip_tablespaces
: すべてのテーブルがデフォルトのテーブルスペースに作成されるように、GRANT ステートメントから TABLESPACE 句を削除します。 MySQL Database Service には、テーブルスペースに対するいくつかの制限があります。

ociParManifest: [true | false]
: このオプションを true に設定すると、ダンプ内のすべてのアイテムに対する読取りアクセスの事前認証済リクエスト (オブジェクト読取り PAR) と、すべての事前認証済リクエスト URL をリストするマニフェストファイルが生成されます。 事前認証済リクエストは、デフォルトで 1 週間後に期限切れになります。これは、ociParExpireTime オプションを使用して変更できます。

  このオプションは MySQL Shell 8.0.22 から使用でき、(osBucketName オプションを設定して) オブジェクトストレージバケットにエクスポートする場合にのみ使用できます。 MySQL Shell 8.0.23 からは、このオプションはすべてのユーティリティで使用でき、MySQL Shell 8.0.22 では、インスタンスダンプユーティリティおよびスキーマダンプユーティリティでのみ使用できます。

  ocimds オプションが true に設定され、osBucketName オプションを使用してオブジェクトストレージバケット名が指定されている場合、ociParManifest はデフォルトで true に設定され、それ以外の場合はデフォルトで false に設定されます。

  オブジェクトストレージバケットへの接続に使用される Oracle Cloud Infrastructure プロファイルで指定されたユーザー (DEFAULT ユーザーまたは ociProfile オプションで指定された別のユーザー) は、事前認証済リクエストの作成者です。 このユーザーには、「事前認証済リクエストの使用」で説明されているように、バケット内のオブジェクトと対話するための PAR_MANAGE 権限および適切な権限が必要です。 オブジェクトの事前認証済リクエスト URL の作成に問題がある場合は、関連付けられたファイルが削除され、ダンプが停止されます。

  生成されたダンプファイルをロードできるようにするには、「事前認証済リクエストの使用」の手順に従って、マニフェストファイルオブジェクト (@.manifest.json) の事前認証済読取りリクエストを作成します。 ダンプが完了する前にダンプのロードを開始する場合は、ダンプの進行中にこれを実行できます。 必要な権限を持つユーザーアカウントを使用して、この事前認証済読取りリクエストを作成できます。 その後、事前認証済リクエスト URL をダンプロードユーティリティで使用

して、マニフェストファイルを介してダンプファイルにアクセスする必要があります。 URL は作成時にのみ表示されるため、永続記憶域にコピーします。

> **重要**
>
> このアクセス方法を使用する前に、バケットまたはオブジェクトに対する事前認証済アクセスのビジネス要件およびセキュリティの影響を評価します。
>
> 事前認証済リクエスト URL は、リクエストで識別されたターゲットへの URL アクセス権を持つすべてのユーザーに付与します。 マニフェストファイル用に作成する事前認証済 URL、およびマニフェストファイル内のエクスポートされたアイテム用の事前認証済 URL の配布を慎重に管理します。

ociParExpireTime: "string"

ociParManifest オプションが true に設定されている場合に生成される事前認証済リクエスト URL の有効期限。 デフォルトは、UTC 形式の現在の時間に 1 週間を加えたものです。

このオプションは、MySQL Shell 8.0.22 から使用できます。 MySQL Shell 8.0.23 からは、このオプションはすべてのユーティリティで使用でき、MySQL Shell 8.0.22 では、インスタンスダンプユーティリティおよびスキーマダンプユーティリティでのみ使用できます。

有効期限は、事前認証済リクエストの作成時に Oracle Cloud Infrastructure で必要とされる RFC 3339 タイムスタンプとして書式設定する必要があります。 書式は、YYYY-MM-DDTHH-MM-SS の直後に文字 Z (UTC 時間の場合) または[+|-]hh:mm で表される ローカル時間の UTC オフセット (2020-10-01T00:09:51.000+02:00 など) が続きます。MySQL Shell では有効期限は検証されませんが、フォーマットエラーにより、ダンプ内の最初のファイルに対する事前認証済リクエストの作成が失敗し、ダンプが停止されます。

8.6 ダンプロードユーティリティ

MySQL Shell ダンプロードユーティリティ util.loadDump() は、MySQL Shell 8.0.21 で導入され、MySQL Database Service DB システム (MySQL DB システム、短縮形) または MySQL Shell セクション8.5「インスタンスダンプユーティリティ、スキーマダンプユーティリティおよびテーブルダンプユーティリティ」 を使用してダンプされたスキーマまたはテーブルの MySQL Server インスタンスへのインポートをサポートしています。 ダンプロードユーティリティでは、リモート記憶域からのデータストリーミング、テーブルまたはテーブルチャンクのパラレルロード、進行状況トラッキング、再開およびリセット機能、およびダンプの実行中の同時ロードのオプションが提供されます。

MySQL DB システムにインポートするには、ダンプロードユーティリティを実行する MySQL Shell インスタンスが、MySQL DB システムにアクセスできる Oracle Cloud Infrastructure Compute インスタンスにインストールされている必要があります。 ダンプファイルが Oracle Cloud Infrastructure Object Storage バケットにある場合は、コンピュートインスタンスからオブジェクトストレージバケットにアクセスできます。 ダンプファイルがローカルシステムにある場合は、コンピュートインスタンスに選択したオペレーティングシステムに応じて、選択したコピーユーティリティを使用して Oracle Cloud Infrastructure Compute インスタンスに転送する必要があります。 MySQL Database Service との互換性のために、MySQL Shell インスタンスダンプユーティリティまたはスキーマダンプユーティリティで ocimds オプションを true に設定してダンプが作成されていることを確認します。MySQL Shell テーブルダンプユーティリティでは、このオプションは使用しません。

> **注記**
>
> 1. ダンプロードユーティリティでは LOAD DATA LOCAL INFILE ステートメントが使用されるため、インポート中は、ターゲット MySQL インスタンスの local_infile システム変数のグローバル設定を ON にする必要があります。 デフォルトでは、このシステム変数は標準の MySQL DB システム構成で ON に設定されています。
>
> 2. ターゲットの MySQL インスタンスでは、ダンプロードユーティリティは、sql_require_primary_key システム変数が ON に設定されているかどうかをチェッ

クし、設定されている場合は、ダンプファイルに主キーのないテーブルがあるとエラーを返します。 デフォルトでは、このシステム変数は標準の MySQL DB システム構成で OFF に設定されています。

3. ダンプロードユーティリティは、ソース MySQL インスタンスの gtid_executed GTID セットをターゲット MySQL インスタンスに自動的に適用しません。 GTID セットは、@.json ダンプファイルの gtidExecuted フィールドとして、MySQL Shell インスタンスダンプユーティリティ、スキーマダンプユーティリティまたはテーブルダンプユーティリティのダンプメタデータに含まれます。 レプリケーションで使用するためにこれらの GTID をターゲット MySQL インスタンスに適用するには、MySQL Shell 8.0.22 から、updateGtidSet オプションを使用して、ターゲット MySQL インスタンスのリリースに応じて gtid_purged GTID セットに追加するか、gtid_purged GTID セットを置換します。 権限の制限のため、これは現在 MySQL DB システムではサポートされていません。 MySQL Shell 8.0.21 では GTID セットを手動でインポートできますが、これは MySQL DB システムではサポートされていません。

インスタンスダンプユーティリティまたはスキーマダンプユーティリティによって生成される出力の場合、MySQL Shell ダンプロードユーティリティは DDL ファイルおよびタブ区切りの .tsv データファイルを使用して、ターゲットの MySQL インスタンスにサーバーインスタンスまたはスキーマを設定し、データをロードします。 DDL ファイルのみを含むダンプまたはデータファイルのみを使用して、これらのタスクを個別に実行できます。 ダンプロードユーティリティでは、DDL ファイルおよびデータファイルを、両方の種類のファイルを含む通常のダンプから個別に適用することもできます。

MySQL Shell テーブルダンプユーティリティによって生成される出力の場合、MySQL Shell 8.0.23 からのダンプには、最初にテーブルを含むスキーマの設定に必要な情報が含まれます。 デフォルトでは、そのリリースから、ターゲットの MySQL インスタンスにスキーマがまだ存在しない場合は再作成されます。 または、ダンプロードユーティリティで schema オプションを指定して、ターゲット MySQL インスタンスの代替スキーマにテーブルをロードできます。 MySQL Shell 8.0.22 では、テーブルダンプユーティリティファイルにスキーマ情報が含まれていないため、ターゲットスキーマがターゲットの MySQL インスタンスに存在する必要があります。 このリリースでは、デフォルトでグローバルシェルセッションの現在のスキーマがターゲットスキーマとして使用されるか、schema オプションを使用してスキーマに名前を付けることができます。

ダンプロードユーティリティのその他のオプションを使用して、インポートをカスタマイズできます:

- インポートまたはインポートから除外する個々のテーブルまたはスキーマを選択できます。
- ユーザーとそのロールおよび権限はデフォルトで除外されますが、インポートを選択できます。
- ターゲット MySQL インスタンスのデータには、ダンプファイルで使用されているものとは異なる文字セットを指定できます。
- データがすでにロードされている場合でも、ANALYZE TABLE ヒストグラムを更新できます。
- SET sql_log_bin=0 ステートメントを使用したインポート中に、ターゲット MySQL インスタンスでバイナリロギングをスキップすることを選択できます。

選択したダンプロードオプションのセットを使用してドライランを実行し、これらのオプションを使用してユーティリティを実際に実行したときに実行されるアクションを表示できます。

waitDumpTimeout オプションを使用すると、まだ作成中のダンプを適用できます。 テーブルは使用可能になるとロードされ、新しいデータがダンプの場所に到着しなくなった後、ユーティリティは指定された秒数待機します。 タイムアウトが経過すると、ユーティリティはダンプが完了したとみなし、インポートを停止します。

インポートの進行状態は永続的な進行状態ファイルに格納され、正常に完了したステップと中断または失敗したステップが記録されます。 デフォルトでは、進捗状態ファイルは load-progress.server_uuid.json という名前でダンプディレクトリに作成されますが、別の名前と場所を選択することもできます。 ダンプロードユーティリティは、ダンプのインポートを再開または再試行するときに進行状態ファイルを参照し、完了したステップをスキップします。 部分的にロードされたテーブルの重複除外は自動的に管理されます。 Ctrl + C を使用して進行中のダンプを中断した場合、そのキーの組合せを最初に使用しても、ユーティリティによって新しいタスクは開始されませんが、既存のタスクは続行されます。 Ctrl + C を再度押すと、既存のタスクが停止し、エラーメッセージが表示されます。 いずれの場合も、ユーティリティは停止した場所からインポートを再開できます。

進行状態をリセットしてダンプのインポートを最初から再開することを選択できますが、この場合、ユーティリティはすでに作成されて重複除外を管理していないオブジェクトをスキップしません。 これを行う場合、正しいインポートを保証するには、以前にロードされたすべてのオブジェクト (スキーマ、テーブル、ユーザー、ビュー、トリガー、ルーチン、イベントなど) をターゲット MySQL インスタンスから手動で削除する必要があります。 それ以外の場合、ダンプファイル内のオブジェクトがターゲット MySQL インスタンスにすでに存在すると、インポートはエラーで停止します。 適切な注意が必要な場合は、ignoreExistingObjects オプションを使用してユーティリティレポートでオブジェクトを複製しますが、スキップしてインポートを続行できます。 ユーティリティでは、ターゲット MySQL インスタンスとダンプファイルのオブジェクトの内容が異なるかどうかはチェックされないため、インポート結果に不正または無効なデータが含まれる可能性があります。

> **重要**
>
> ダンプの停止とダンプの再開の間にダンプファイル内のデータを変更しないでください。 データの変更後にダンプを再開すると動作が未定義になり、データの不整合やデータの損失が発生する可能性があります。 ダンプを部分的にロードした後にデータを変更する必要がある場合は、部分的なインポート中に作成されたすべてのオブジェクトを手動で削除し (進捗状態ファイルにリストされています)、resetProgress オプションを指定してダンプロードユーティリティを実行し、最初から再開します。

ダンプのデータファイル内のデータをターゲットの MySQL インスタンスにインポートする前に変更する必要がある場合は、MySQL シェルのパラレルテーブルインポートユーティリティ util.importTable() とダンプロードユーティリティを組み合せて変更できます。 これを行うには、まずダンプロードユーティリティを使用して、選択したテーブルの DDL のみをロードし、ターゲットサーバーにテーブルを作成します。 次に、パラレルテーブルインポートユーティリティを使用して、テーブルの出力ファイルからデータを取得および変換し、ターゲットテーブルにインポートします。 必要に応じて、データを変更する他のテーブルに対してこのプロセスを繰り返します。 最後に、ダンプロードユーティリティを使用して、変更しない残りのテーブル (変更したテーブルを除く) の DDL およびデータをロードします。 手順の詳細は、ダンプしたデータの変更 を参照してください。

オブジェクトストレージバケットからの事前認証済リクエストを含むダンプファイルを MySQL DB システムにロードするには、マニフェストファイルオブジェクト (@.manifest.json) 用に作成された事前認証済読取りリクエスト URL が必要です。 また、オブジェクトストレージバケット内のダンプファイルと同じ接頭辞付きの場所に、テキストファイルに対する事前認証済の読取り/書込みリクエストを作成します。 これはダンプロードユーティリティの進捗状態ファイルで、事前認証済リクエストを含むダンプファイルをロードする場合に必要です。 リクエストの作成に必要な権限を持つ任意のユーザーアカウントを使用できます。 テキストファイルには任意の名前を付けることができ、ファイルを作成するか、ユーティリティで作成できます。 ファイルのコンテンツは JSON 形式になるため、.json ファイル拡張子は使用する場合に適しています (progress.json など)。

ダンプファイル (デフォルト) とともに進捗状態ファイルを格納するかわりに、ダンプロードユーティリティを実行する場所にあるローカルファイルを使用できます。 進捗状態ファイルの事前認証済読取り/書込みリクエストを作成する権限がない場合、このメソッドを使用して進捗を格納できます。 ローカルファイルを使用する場合、ダンプロードユーティリティを別の場所から実行してもダンプを再開できないことに注意してください。

事前認証済リクエストでは、ダンプロードユーティリティを実行するときに、@.manifest.json ファイルの事前認証済リクエスト URL としてダンプ URL を指定します。 また、進行状態ファイル (progressFile オプション) をオブジェクトストレージバケット内のファイルの事前認証済リクエスト URL として、またはローカルシステム上のファイル (このオプションを選択した場合) として指定します。 ダンプロードユーティリティを実行するユーザーアカウントは、追加のアクセス権限なしでマニフェストファイル内の URL を使用してダンプファイルをロードできます。 ダンプがまだ進行中の場合、ダンプロードユーティリティは、オブジェクトストレージバケットではなくマニフェストファイルへの新しい追加を監視して待機します。

ダンプの DDL ファイルは単一のスレッドによってロードされますが、データは選択したスレッド数 (デフォルトは 4) によってパラレルにロードされます。 ダンプの作成時にテーブルデータがチャンク化された場合は、テーブルに複数のスレッドを使用できます。それ以外の場合は、各スレッドが一度に 1 つのテーブルをロードします。 ダンプロードユーティリティは、並列性を最大化するためにスレッド間のデータインポートをスケジュールします。 ダンプファイルが MySQL Shell ダンプユーティリティによって圧縮されている場合、ダンプロードユーティリティはそれらの解凍を処理します。

デフォルトでは、テーブルの全文インデックスは、テーブルが完全にロードされた後にのみ作成され、インポートが高速化されます。 各テーブルが完全にロードされるまで、すべてのインデックス作成 (プライマリインデックスを除く) を遅延するように選択できます。 テーブルのインポート中にすべてのインデックスを作成することもできます。 また、インポート中にインデックスの作成を無効にし、ロード後にテーブル構造を変更する場合などは、後でインデックスを作成することもできます。

データロードのパフォーマンスをさらに向上させるために、インポート中にターゲット MySQL インスタンスの InnoDB redo ログを無効にできます。 これは (本番システムではなく) 新しい MySQL Server インスタンスでのみ行う必要があり、この機能は MySQL DB システムでは使用できないことに注意してください。 詳細は、redo ロギングの無効化を参照してください。

ダンプロードユーティリティは、MySQL Shell グローバルセッションを使用して、ダンプのインポート先のターゲット MySQL インスタンスの接続詳細を取得します。 ユーティリティを実行する前に、(X プロトコル 接続または クラシック MySQL プロトコル 接続を持つことができる) グローバルセッションをオープンする必要があります。 ユーティリティはスレッドごとに独自のセッションを開き、接続圧縮や SSL オプションなどのオプションをグローバルセッションからコピーし、グローバルセッションをこれ以上使用しません。

MySQL Shell API では、ダンプロードユーティリティは util グローバルオブジェクトの関数であり、次のシグネチャを持ちます:

```
util.loadDump(url[, options])
```

ユーティリティを実行している Oracle Cloud Infrastructure Compute インスタンスのファイルシステムにあるダンプをインポートする場合、url はダンプファイルを含むローカルディレクトリへのパスを指定する文字列です。 ローカルディレクトリパスの前に file://スキーマを付けることができます。 MySQL ShellJavaScript モードのこの例では、ダンプファイルがローカルディレクトリから接続された MySQL インスタンスにロードされるときに問題がないことを確認するための予行演習が実行されます:

```
shell-js> util.loadDump("/mnt/data/worlddump", {dryRun: true})
```

Oracle Cloud Infrastructure Object Storage バケットからダンプをインポートする場合、url は、ダンプの作成時に outputUrl パラメータを使用して割り当てられたバケット内のダンプファイルのパス接頭辞です。 osBucketName オプションを使用してオブジェクトストレージバケットの名前を指定し、osNamespace オプションを使用してバケットのネームスペースを識別します。 MySQL ShellJavaScript モードのこの例では、8 つのスレッドを使用して、接頭辞が worlddump のダンプがオブジェクトストレージバケットから接続された MySQL DB システムにロードされます:

```
shell-js> util.loadDump("worlddump", {
      > threads: 8, osBucketName: "hanna-bucket", osNamespace: "idx28w1ckztq"})
```

オブジェクトストレージバケットのネームスペースは、Oracle Cloud Infrastructure コンソールのバケット詳細ページの「バケット情報」タブに表示されるか、Oracle Cloud Infrastructure コマンドラインインタフェースを使用して取得できます。 オブジェクトストレージバケットへの接続は、デフォルトの Oracle Cloud Infrastructure CLI 構成ファイルのデフォルトプロファイル、または ociConfigFile および ociProfile オプションを使用して指定する代替詳細を使用して確立されます。 CLI 構成ファイルを設定する手順については、「SDK および CLI 構成ファイル」を参照してください

options はオプションのディクショナリで、空の場合は省略できます。 次のオプションを使用できます。

dryRun: [true | false]	ダンプの内容に基づいて返されるが、インポートを続行しないエラーを含む、指定されたオプションおよびダンプファイルで実行されるアクションに関する情報を表示します。 デフォルトは false です。
osBucketName: "string"	ダンプファイルがある Oracle Cloud Infrastructure Object Storage バケットの名前。 デフォルトでは、~/.oci/config にある Oracle Cloud Infrastructure CLI 構成ファイルの[DEFAULT]プロファイルを使用して、バケットへの接続が確立されます。 ociConfigFile および ociProfile オプションを使用して、接続に使用される代替プロファイルを置換できます。 CLI 構成ファイルの設定手順については、「SDK および CLI 構成ファイル」を参照してください。
osNamespace: "string"	osBucketName によって指定されたオブジェクトストレージバケットが配置される Oracle Cloud Infrastructure ネームスペース。 オブジェクトストレージバケットのネームスペースは、Oracle Cloud Infrastructure コンソールのバケット詳細ページの「バケット情報」タブに表示されるか、Oracle Cloud Infrastructure コマンドラインインタフェースを使用して取得できます。
ociConfigFile: "string"	デフォルトの場所の ~/.oci/config ではなく、接続に使用するプロファイルを含む Oracle Cloud Infrastructure CLI 構成ファイル。
ociProfile: "string"	接続に使用される Oracle Cloud Infrastructure CLI 構成ファイル内の[DEFAULT]プロファイルではなく、接続に使用する Oracle Cloud Infrastructure プロファイルのプロファイル名。

threads: int

データのチャンクをターゲット MySQL インスタンスにアップロードするために使用するパラレルスレッドの数。 各スレッドは、MySQL インスタンスへの独自の接続を持ちます。 ダンプがチャンク化を有効にして作成された場合 (デフォルト)、ユーティリティは複数のスレッドを使用してテーブルのデータをロードできます。それ以外の場合、スレッドは 1 つのテーブルにのみ使用されます。

progressFile: "string"

ダンプロードユーティリティの進捗状態ファイルのローカルファイルの場所。インポートの進捗状態を保持します。 デフォルトでは、進捗状態ファイルは load-progress.server_uuid.json という名前でダンプディレクトリに作成されますが、このオプションを使用して変更できます。 progressFile を空の文字列に設定すると、進行状況の追跡が無効になります。つまり、ダンプロードユーティリティは部分的に完了したインポートを再開できません。

showProgress: [true | false]

インポートの進行状況情報を表示 (true) または非表示 (false) します。 MySQL Shell が対話型モードの場合など、stdout が端末 (tty) の場合、デフォルトは true です。それ以外の場合は false です。 進捗情報には、アクティブスレッドの数とそのアクション、これまでにロードされたデータの量、完了率およびスループット率が含まれます。 進捗情報が表示されない場合でも、進捗状態はダンプロードユーティリティの進捗状態ファイルに記録されます。

resetProgress: [true | false]

このオプションを true に設定すると、進行状態がリセットされ、インポートが最初から再開されます。 デフォルトは false です。 このオプションでは、ダンプロードユーティリティはすでに作成されているオブジェクトをスキップせず、重複除外を管理しないことに注意してください。 このオプションを使用する場合は、正しいインポートを確実にするために、まず、以前にロードされたすべてのオブジェクト (スキーマ、テーブル、ユーザー、ビュー、トリガー、ルーチン、イベントなど) をターゲット MySQL インスタンスから手動で削除する必要があります。 それ以外の場合、ダンプファイル内のオブジェクトがターゲット MySQL インスタンスにすでに存在すると、インポートはエラーで停止します。 適切な注意が必要な場合は、ignoreExistingObjects オプションを使用してユーティリティレポートでオブジェクトを複製しますが、スキップしてインポートを続行できます。

waitDumpTimeout: int

このオプションを設定すると、ダンプの場所にアップロードされたすべてのデータチャンクが処理された後、ユーティリティがそれ以降のデータを待機するタイムアウト (秒) を指定することで、同時ロードがアクティブ化されます。 これにより、作成中のダンプをユーティリティでインポートできます。 データは使用可能になると処理され、ダンプの場所にデータが表示されずにタイムアウトを超えるとインポートは停止します。 デフォルト設定の 0 は、アップロードされたすべてのデータチャンクが処理され、それ以上のデータを待機しない場合に、ユーティリティがダンプを完了としてマークすることを意味します。

ignoreExistingObjects: [true | false]

MySQL インスタンスのターゲットスキーマにすでに存在するオブジェクトが含まれている場合でも、ダンプをインポートします。 デフォルトは false です。つまり、インポートが進行状態ファイルを使用した前回の試行から再開されないかぎり、エラーが発行され、重複オブジェクトが検出されるとインポートが停止します。この場合、チェックはスキップされます。 このオプションを true に設定すると、重複オブジェクトがレポートされますが、エラーは生成されず、インポートは続行されます。 ユーティリティでは、ターゲット MySQL インスタンスとダンプファイルのオブジェクトの内容が異なるかどうかはチェックされないため、このオプションは注意して使用する必要があります。そのため、インポート結果に不正または無効なデータが含まれる可能性があります。 別の方法として、excludeTables オプションを使用して、ダンプファイル内のオブジェクトがターゲット MySQL インスタンス内のインポート済オブジェクトと同じであることを確認したときにすでにロードしたテーブルを除外する方法もあります。 ダンプを再起動する前に、重複するオブジェクトをターゲット MySQL インスタンスから削除することをお薦めします。

ignoreVersion: [true | false]

データのダンプ元の MySQL インスタンスのメジャーバージョン番号が、データのアップロード先の MySQL インスタンスのメジャーバージョン番号と異なる場合でも、ダンプをインポートします。 デフォルトは false で、メジャーバージョン番号が異なる場合、エラーが発行され、インポートは続行されません。 このオプションを true に設定すると、警告が発行され、インポートが続行されます。 インポー

トは、ダンプファイル内のスキーマに新しいメジャーバージョンとの互換性の問題がない場合にのみ成功することに注意してください。

MySQL Shell 8.0.23 からは、このオプションを使用して、ocimds オプションを使用せずに作成されたダンプを MySQL Database Service インスタンスにインポートすることもできます。

ignoreVersion オプションを使用してインポートを試行する前に、MySQL Shell アップグレードチェッカユーティリティ checkForServerUpgrade() を使用して、ソース MySQL インスタンスのスキーマを確認します。 スキーマをダンプしてターゲット MySQL インスタンスにインポートする前に、ユーティリティで特定された互換性の問題を修正します。

updateGtidSet: [off | append | replace]

ダンプメタデータに記録されているソース MySQL インスタンスの gtid_executed GTID セットを、ターゲット MySQL インスタンスの gtid_purged GTID セットに適用します。 gtid_purged GTID セットは、サーバーに適用されたが、サーバー上のバイナリログファイルには存在しないすべてのトランザクションの GTID を保持します。 このオプションは MySQL Shell 8.0.22 から使用できますが、そのリリースでは、権限の制限のため、MySQL DB システムではサポートされていません。 MySQL 8.0.23 から、このオプションを MySQL DB システムインスタンスにも使用できます。 デフォルトは off で、GTID セットが適用されないことを意味します。

このオプションは、MySQL Shell インスタンスダンプユーティリティまたはスキーマダンプユーティリティによって生成されたダンプに対してのみ、MySQL Shell テーブルダンプユーティリティによって生成されたダンプには使用しないでください。 また、グループレプリケーションがターゲットの MySQL インスタンスで実行されている場合は、このオプションを使用しないでください。

MySQL DB システムインスタンスではない MySQL インスタンスの場合、GTID セットを更新するために append または replace を設定するときに、skipBinlog オプションも true に設定します。 これにより、ソースサーバー上の GTID がターゲットサーバー上の GTID と一致することが保証されます。 MySQL DB システムインスタンスの場合、このオプションは使用されません。

MySQL 8.0 のターゲット MySQL インスタンスの場合、オプションを append に設定できます。これにより、ソース MySQL インスタンスの gtid_executed GTID セットがターゲット MySQL インスタンスの gtid_purged GTID セットに追加されます。 適用する gtid_executed GTID セットは、@.json ダンプファイルの gtidExecuted フィールドに表示され、ターゲット MySQL インスタンスにすでに存在する gtid_executed セットと交差しないようにする必要があります。 たとえば、別のソース MySQL インスタンスから、他のソースサーバーのスキーマをすでに持つターゲット MySQL インスタンスにスキーマをインポートする場合に、このオプションを使用できます。

MySQL 8.0 のターゲット MySQL インスタンスに replace を使用して、ターゲット MySQL インスタンスの gtid_purged GTID セットをソース MySQL インスタンスの gtid_executed GTID セットに置き換えることもできます。 これを行うには、ソース MySQL インスタンスの gtid_executed GTID セットがターゲット MySQL インスタンスの gtid_purged GTID セットのスーパーセットであり、gtid_purged GTID セットにないターゲット gtid_executed GTID セットのトランザクションのセットと交差していない必要があります。

MySQL 5.7 のターゲット MySQL インスタンスの場合、オプションを replace に設定します。これにより、ターゲット MySQL インスタンスに設定されている gtid_purged GTID が、ソース MySQL インスタンスの gtid_executed GTID セットに置き換えられます。 MySQL 5.7 でこれを行うには、ターゲット MySQL インスタンス上の gtid_executed および gtid_purged GTID セットが空である必要があるた

め、以前に GTID セットをインポートしていない状態でインスタンスを使用しないでください。

MySQL Shell 8.0.21 では、このオプションを使用できない場合、GTID セットを MySQL Server インスタンスに手動で適用できます (グループレプリケーションが使用されている場合を除く)。 MySQL DB システムの場合、この方法はサポートされていません。 GTID セットを適用するには、インポート後に、MySQL Shell\sql コマンドを使用して (または SQL モードを開始して)、接続された MySQL インスタンスで次のステートメントを発行し、ダンプメタデータの@.json ダンプファイルの gtidExecuted フィールドから gtid_executed GTID セットをコピーします:

```
shell-js> \sql SET @@GLOBAL.gtid_purged= "+gtidExecuted_set";
```

このステートメントは、MySQL 8.0 から機能し、ソース MySQL Server インスタンス gtid_executed GTID セットをターゲット MySQL インスタンス gtid_purged GTID セットに追加します。 MySQL 5.7 の場合、プラス記号 (+) は省略する必要があり、ターゲット MySQL インスタンスの gtid_executed および gtid_purged GTID セットは空である必要があります。 詳細は、ターゲット MySQL インスタンスのリリースにおける gtid_purged システム変数の説明を参照してください。

skipBinlog: [true | false]

SET sql_log_bin=0 ステートメントを発行して、インポート中にユーティリティで使用されるセッションのターゲット MySQL インスタンスでバイナリロギングをスキップします。 デフォルトは false であるため、バイナリロギングはデフォルトでアクティブです。 MySQL DB システムの場合、このオプションは使用されず、true に設定しようとするとインポートはエラーで停止します。 他の MySQL インスタンスの場合、updateGtidSet オプションを使用するか手動で、ソース MySQL インスタンスから gtid_executed GTID セットをターゲット MySQL インスタンスに適用する場合は、常に skipBinlog を true に設定します。 GTID がターゲット MySQL インスタンス (gtid_mode=ON) で使用されている場合、このオプションを true に設定すると、インポートの実行中に新しい GTID が生成および割り当てられないため、ソースサーバーから設定された元の GTID を使用できます。 ユーザーアカウントには、sql_log_bin システム変数の設定に必要な権限が必要です。

loadIndexes: [true | false]

テーブルのセカンダリインデックスを作成 (true) するか、作成 (false) しないでください。 デフォルトは true です。 このオプションが false に設定されている場合、セカンダリインデックスはインポート中に作成されないため、後で作成する必要があります。 これは、DDL ファイルとデータファイルを個別にロードする場合、および DDL ファイルのロード後にテーブル構造を変更する場合に便利です。 その後、loadIndexes を true に設定し、deferTableIndexes を all に設定してダンプロードユーティリティを再度実行することで、セカンダリインデックスを作成できます。

deferTableIndexes: [off | fulltext | all]

テーブルデータがロードされるまでセカンダリインデックスの作成を延期します。これにより、ロード時間を短縮できます。off は、テーブルのロード中にすべてのインデックスが作成されることを意味します。 デフォルト設定の fulltext では、全文インデックスのみが遅延されます。all は、すべてのセカンダリインデックスを遅延し、テーブルのロード中にのみプライマリインデックスを作成します。また、自動インクリメント値を含むカラムに (MySQL Shell 8.0.22 から) 定義されたインデックスも作成します。 MySQL Shell 8.0.21 では、自動増分値を含む一意のキーカラムがある場合、all を設定しないでください。

analyzeTables: [off | on | histogram]

ロードされたテーブルに対して ANALYZE TABLE を実行します。on はすべてのテーブルを分析し、histogram はダンプにヒストグラム情報が格納されているテーブルのみを分析します。 デフォルトは off です。 このオプションを指定してダンプロードユーティリティを実行すると、データがすでにロードされている場合でもテーブルを分析できます。

characterSet: "string"

LOAD DATA ステートメントの CHARACTER SET オプションなどで、ターゲット MySQL インスタンスへのインポートに使用される文字セット。 デフォルトは、ダンプが MySQL Shell インスタンスダンプユーティリティ、スキーマダンプユーティリティまたはテーブルダンプユーティリティによって作成されたときに使用さ

	れたダンプメタデータで指定された文字セットで、デフォルトでは utf8mb4 が使用されます。 文字セットは、character_set_client システム変数で許可され、MySQL インスタンスでサポートされている必要があります。
schema: "string"	MySQL Shell テーブルダンプユーティリティによって生成されたダンプをロードする必要がある既存のターゲットスキーマ。 MySQL Shell 8.0.23 からは、テーブルダンプユーティリティのダンプファイルには、最初にテーブルが含まれていたスキーマの設定に必要な情報が含まれているため、このオプションは必要ありません。 デフォルトでは、そのリリースから、ターゲットの MySQL インスタンスにスキーマがまだ存在しない場合は再作成されます。 または、schema オプションを指定して、ターゲット MySQL インスタンスの代替スキーマにテーブルをロードできます。 MySQL Shell 8.0.22 では、テーブルダンプユーティリティのダンプファイルにスキーマ情報が含まれていないため、ターゲットスキーマがターゲットの MySQL インスタンスに存在する必要があります。 このリリースでは、デフォルトでグローバルシェルセッションの現在のスキーマがターゲットスキーマとして使用されるか、schema オプションを使用してターゲットスキーマを指定できます。
excludeSchemas: array of strings	指定したスキーマをインポートから除外します。 information_schema, mysql, ndbinfo, performance_schema および sys スキーマは、常に MySQL Shell インスタンスダンプユーティリティによって作成されたダンプから除外されることに注意してください。 ダンプファイルに名前付きスキーマが存在しない場合、ユーティリティはその項目を無視します。
includeSchemas: array of strings	ダンプファイルから名前付きスキーマのみをロードします。 両方のオプションを指定できます。この場合、includeSchemas 文字列と excludeSchemas 文字列の両方で一致するスキーマ名は除外されます。
excludeTables: array of strings	指定したテーブルをインポートから除外します。 テーブル名は、有効なスキーマ名で修飾し、必要に応じてバックティック文字で引用符で囲む必要があります。 mysql.apply_status, mysql.general_log, mysql.schema および mysql.slow_log tables のデータは、DDL ステートメントは含まれていますが、常に MySQL Shell スキーマダンプユーティリティによって作成されたダンプから除外されることに注意してください。 excludeTables オプションで指定されたテーブルは、ターゲットの MySQL インスタンスにアップロードされません。 指定したテーブルがスキーマに存在しない場合、またはスキーマがダンプファイルに存在しない場合、ダンプロードユーティリティは項目を無視します。
includeTables: array of strings	ダンプファイルから指定されたテーブルのみをロードします。 テーブル名は、有効なスキーマ名で修飾し、必要に応じてバックティック文字で引用符で囲む必要があります。 両方のオプションを指定できます。この場合、includeTables 文字列と excludeTables 文字列の両方で一致するテーブル名は除外されます。
loadDdl: [true | false]	このオプションを true に設定すると、ダンプから DDL ファイルのみがインポートされ、データはインポートされません。 デフォルトは false です。
loadData: [true | false]	このオプションを true に設定すると、ダンプからデータファイルのみがインポートされ、DDL ファイルはインポートされません。 デフォルトは false です。
loadUsers: [true | false]	(true) をインポートするか、(false) ユーザーとそのロールおよび権限をターゲットの MySQL インスタンスにインポートしないでください。 デフォルトは false であるため、ユーザーはデフォルトでインポートされません。 現在のユーザーのステートメントはスキップされます。 MySQL Shell 8.0.22 からは、ターゲットの MySQL インスタンスにユーザーがすでに存在する場合、エラーが返され、ダンプファイルからのユーザー権限は適用されません。 MySQL Shell 8.0.22 から、ダン

プロードユーティリティの excludeUsers または includeUsers オプションを使用して、インポートに除外または含めるユーザーアカウントを指定できます。

> 注記
>
> MySQL Shell 8.0.21 では、root ユーザーアカウントまたは別の制限付きユーザーアカウント名がダンプファイルに存在する場合、ユーザーを MySQL DB システムにインポートしようとするとインポートが失敗するため、そのリリースではユーザーの MySQL DB システムへのインポートはサポートされていません。

MySQL Shell スキーマダンプユーティリティおよびテーブルダンプユーティリティでは、ダンプにユーザー、ロールおよび付与は含まれませんが、インスタンスダンプユーティリティではデフォルトで可能であり、実行できます。 MySQL Shell 8.0.22 から、excludeUsers および includeUsers オプションをインスタンスダンプユーティリティで使用して、ダンプファイルから名前付きユーザーアカウントを除外または含めることもできます。

true を指定したが、指定したダンプファイルにユーザーアカウントが含まれていない場合、MySQL Shell 8.0.23 の前に、ユーティリティはエラーを返してインポートを停止します。 MySQL Shell 8.0.23 からは、かわりにユーティリティは警告を返して続行します。

excludeUsers: array of strings

指定されたユーザーアカウントをインポートから除外します。 このオプションは MySQL Shell 8.0.22 から使用でき、これを使用して、MySQL DB システムへのインポートが許可されていないユーザーアカウント、またはターゲットの MySQL インスタンスにすでに存在するか不要なユーザーアカウントを除外できます。 各ユーザーアカウント文字列は、ユーザー名とホスト名で定義されたアカウントの場合は"'user_name'@'host_name'"の形式で、ユーザー名のみで定義されたアカウントの場合は"'user_name'" ("'user_name'@'%'"と同等) で指定します。 指定されたユーザーアカウントがダンプファイルに存在しない場合、ユーティリティは項目を無視します。

includeUsers: array of strings

指定されたユーザーアカウントのみをインポートに含めます。 excludeUsers オプションの場合と同様に、各ユーザーアカウント文字列を指定します。 このオプションは MySQL Shell 8.0.22 から使用でき、ターゲット MySQL インスタンスで必要なユーザーアカウントが少ない場合は、excludeUsers のかわりに使用できます。 両方のオプションを指定することもできます。この場合、includeUsers 文字列と excludeUsers 文字列の両方で一致するユーザーアカウントは除外されます。

ダンプしたデータの変更

MySQL シェルのパラレルテーブルインポートユーティリティ util.importTable() をダンプロードユーティリティ util.loadDump() と組み合せて使用すると、チャンク出力ファイルのデータをターゲットの MySQL インスタンスにアップロードする前に変更できます。 この方法では、一度に 1 つのテーブルのデータを変更できます。 MySQL Shell 8.0.23 から動作する次の手順に従います:

1. loadDdl オプションを指定してダンプロードユーティリティを使用して DDL ファイルをロードし、選択したテーブルをデータなしでターゲット MySQL インスタンスに作成します。

```
shell-js> util.loadDump("/mnt/data/proddump", {
      > includeTables: ["product.pricing"],
      > loadDdl: true,
      > loadData: false});
```

2. パラレルテーブルインポートユーティリティを使用して、テーブルのデータを取得および変換し、ターゲット MySQL インスタンスの空のテーブルにインポートします。 この例では、pricing テーブルのデータは、ワイルドカードパターンマッチングを使用して指定された複数の圧縮ファイルにあります。 ダンプファイルの id および prodname カラムの値は、ターゲットテーブルの同じカラムにそのまま割り当てられます。 ダンプファイルの price カラムの値が取得され、変数@1 に割り当てられます。 その後、decodeColumns オプションを使用して価格を標準金額で減額し、減額された価格をターゲットテーブルの price カラムに配置します。

```
shell-js> util.importTable ("/mnt/data/proddump/product@pricing@*.zst", {
      > schema: "product",
      > table: "pricing",
      > columns: ["id",  "prodname",  1],
      > decodeColumns: { "price": "0.8 * @1"}});
```

3. データを変更する必要があるダンプファイル内の他のテーブルについて、必要に応じてステップ 1 と 2 を繰り返します。

4. 変更する必要があるすべてのテーブルおよびデータのアップロードが終了したら、ダンプロードユーティリティを使用して、変更する必要がない残りのテーブルの DDL とデータの両方をロードします。 前のステップで変更したテーブルは除外してください。

```
shell-js> util.loadDump("/mnt/data/proddump", {excludeTables: ["product.pricing"]});
```

第 9 章 MySQL Shell のロギングおよびデバッグ

目次

9.1 アプリケーションログ 173
9.2 冗長出力 175
9.3 AdminAPI 操作のロギング 175

MySQL Shell ロギング機能を使用して、実行中の MySQL Shell の状態を確認し、問題をトラブルシューティングできます。

デフォルトでは、MySQL Shell はロギング情報をロギングレベル 5 (エラー、警告および情報メッセージ) でアプリケーションログファイルに送信します。 オプションの表示可能な追加の場所に情報を送信し、(MySQL 8.0.17 から) 詳細出力としてコンソールに送信するように MySQL Shell を構成することもできます。

各宛先に送信される詳細のレベルを制御できます。 アプリケーションログおよび表示可能な追加の場所については、使用可能な任意のレベルを詳細の最大レベルとして指定できます。 詳細出力の場合は、詳細の最大レベルにマップする設定を指定できます。 次の詳細レベルを使用できます:

表 9.1 MySQL Shell でのロギングレベル

ロギングレベル - 数値	ロギングレベル - テキスト	意味	詳細設定
1	none	ロギングなし	0
2	internal	内部エラー	1
3	エラー	エラー	1
4	warning	警告	1
5	info	情報	1
6	debug	デバッグ	2
7	debug2	Debug2	3
8	debug3	Debug3	4

デフォルトでは、MySQL Shell は、AdminAPI 操作の過程で実行される SQL ステートメントを記録または出力しません。 操作中に返されるメッセージに加えて、SQL 実行に関してこれらの操作の進行状況を監視する場合は、MySQL Shell 8.0.18 からこれらのステートメントのロギングをアクティブ化できます。 ロギングレベルが 5 以上に設定されている場合、ステートメントは情報メッセージとして MySQL Shell アプリケーションログファイルに書き込まれます。 冗長設定が 1 以上の場合は、冗長出力としてコンソールに送信されます。

アプリケーションログおよびオプションの追加の宛先 (Unix ベースのシステムでは stderr、Windows システムでは OutputDebugString() 機能) を構成する手順は、セクション9.1「アプリケーションログ」 を参照してください。

ロギング情報を冗長出力としてコンソールに送信する手順については、セクション9.2「冗長出力」 を参照してください。

AdminAPI 操作によって実行される SQL ステートメントのロギングをアクティブ化する手順は、セクション 9.3「AdminAPI 操作のロギング」 を参照してください。

9.1 アプリケーションログ

MySQL Shell アプリケーションログファイルの場所はユーザー構成パスで、ファイルの名前は mysqlsh.log です。 デフォルトでは、MySQL Shell はロギング情報をロギングレベル 5 (エラー、警告および情報メッセージ) でこのファイルに送信します。 送信されるロギング情報のレベルを変更したり、アプリケーションログファイルへのロギングを無効にするには、次のいずれかのオプションを選択します:

- MySQL Shell の起動時に --log-level コマンドラインオプションを使用します。

- MySQL Shell \option コマンドを使用して、logLevel MySQL Shell 構成オプションを設定します。 このコマンドの使用手順については、セクション10.4「MySQL Shell オプションの構成」 を参照してください。
- shell.options オブジェクトを使用して、logLevel MySQL Shell 構成オプションを設定します。 この構成インタフェースの使用手順については、セクション10.4「MySQL Shell オプションの構成」 を参照してください。

使用可能なロギングレベルは、表9.1「MySQL Shell でのロギングレベル」 にリストされています。 オプションにロギングレベル 1 または none を指定すると、アプリケーションログファイルへのロギングは無効になります。 他のすべての値はロギングを有効のままにし、ログファイルの詳細レベルを設定します。 このオプションには値が必要です。

--log-level コマンドラインオプションを使用すると、テキスト名または同等の数値を使用してロギングレベルを指定できるため、次の例でも同じ効果が得られます:

```
shell> mysqlsh --log-level=4
shell> mysqlsh --log-level=warning
```

logLevel MySQL Shell 構成オプションでは、数値のロギングレベルのみを指定できます。

ロギングレベルの先頭に@ (アットマーク) を付加すると、ログエントリは表示可能な追加の場所に出力され、MySQL Shell ログファイルに書き込まれます。 次の例でも同じ効果があります:

```
shell> mysqlsh --log-level=@8
shell> mysqlsh --log-level=@debug3
```

Unix ベースのシステムでは、ログエントリは、現在 MySQL Shell に設定されている出力形式で stderr に出力されます。 これは、--json コマンドラインオプションを使用して MySQL Shell を起動して JSON ラッピングがアクティブ化されていないかぎり、resultFormat MySQL Shell 構成オプションの値です。

Windows システムでは、ログエントリは OutputDebugString() 関数を使用して出力されます。この関数の出力は、アプリケーションデバッガ、システムデバッガまたはデバッグ出力用のキャプチャツールで表示できます。

MySQL Shell のログファイル形式はプレーンテキストで、エントリには問題のタイムスタンプと説明、および前述のリストのロギングレベルが含まれます。 例:

```
2016-04-05 22:23:01: Error: Default Domain: (shell):1:8: MySQLError: You have an error
in your SQL syntax; check the manual that corresponds to your MySQL server version for
the right syntax to use near '' at line 1 (1064) in session.sql("select * from t
limit").execute().all();
```

Windows でのログファイルの場所

Windows では、アプリケーションログファイルへのデフォルトパスは %APPDATA%\MySQL\mysqlsh\mysqlsh.log です。 システム上の %APPDATA% の場所を検索するには、コマンドラインからエコーします。 例:

```
C:>echo %APPDATA%

C:\Users\exampleuser\AppData\Roaming
```

Windows では、パスは、MySQL\mysqlsh が追加されたユーザー固有の %APPDATA% フォルダです。 前述の例を使用すると、パスは C:\Users\exampleuser\AppData\Roaming\MySQL\mysqlsh\mysqlsh.log になります。

アプリケーションログファイルを別の場所に格納する場合は、環境変数 MYSQLSH_USER_CONFIG_HOME を定義することで、デフォルトのユーザー構成パスをオーバーライドできます。 この変数の値は、Windows 上の %AppData%\MySQL\mysqlsh\に置き換わります。

Unix ベースシステムでのログファイルの場所

Unix を実行しているマシンの場合、アプリケーションログファイルのデフォルトパスは ~/.mysqlsh/mysqlsh.log です。ここで、「~」 はユーザーホームディレクトリを表します。 環境変数 HOME は、ユーザーホームディレクトリも表します。 ユーザーホームディレクトリに .mysqlsh を追加すると、ログへのデフォルトパスが決まります。

アプリケーションログファイルを別の場所に格納する場合は、環境変数 MYSQLSH_USER_CONFIG_HOME を定義することで、デフォルトのユーザー構成パスをオーバーライドできます。 この変数の値は、Unix 上の ~/.mysqlsh/に置き換わります。

9.2 冗長出力

MySQL 8.0.17 から、デバッグに役立つ MySQL Shell ロギング情報をコンソールに送信できます。 コンソールに送信されるロギングメッセージには、verbose:接頭辞が付けられます。 コンソールにロギング情報を送信しても、アプリケーションログファイルに送信されます。

ロギング情報を冗長出力としてコンソールに送信するには、次のいずれかのオプションを選択します:

- MySQL Shell の起動時に --verbose コマンドラインオプションを使用します。
- MySQL Shell \option コマンドを使用して、verbose MySQL Shell 構成オプションを設定します。 このコマンドの使用手順については、セクション10.4「MySQL Shell オプションの構成」 を参照してください。
- shell.options オブジェクトを使用して、verbose MySQL Shell 構成オプションを設定します。 この構成インタフェースの使用手順については、セクション10.4「MySQL Shell オプションの構成」 を参照してください。

使用可能な設定は、表9.1「MySQL Shell での口ギングレベル」 にリストされています。 verbose オプションの設定では、次の詳細レベルでメッセージが表示されます:

0 メッセージはありません。 アプリケーションログの口ギングレベル 1 と同等です。

1 内部エラー、エラー、警告および情報メッセージ。 アプリケーションログの口ギングレベル 5 と同等です。

2 debug メッセージを追加します。 アプリケーションログの口ギングレベル 6 と同等です。

3 debug2 メッセージを追加します。 アプリケーションログの口ギングレベル 7 と同等です。

4 debug3 メッセージ (詳細の最上位レベル) を追加します。 アプリケーションログの口ギングレベル 8 と同等です。

verbose オプションがコマンド行または構成ファイルで設定されていない場合、またはオプションに 0 の設定を指定した場合、コンソールへの冗長出力は無効になります。 他のすべての値を指定すると、冗長出力が有効になり、コンソールに送信されるメッセージの詳細レベルが設定されます。 MySQL Shell (--verbose) の起動時にコマンドラインオプションとして許可されるが、オプションを設定する他の方法では許可されないオプションを指定した場合は、1 (内部エラー、エラー、警告および情報メッセージ) が使用されます。

9.3 AdminAPI 操作の口ギング

MySQL Shell 8.0.18 から、AdminAPI 操作の過程で実行される SQL ステートメントを MySQL Shell 口ギング情報の一部として含めることができます。 デフォルトでは、MySQL Shell はこれらのステートメントを口グに記録せず、操作中に返されたメッセージのみを口グに記録します。 これらのステートメントの口ギングをアクティブ化すると、エラーの問題診断に役立つ SQL 実行の観点から操作の進行状況を確認できます。

AdminAPI 操作から SQL ステートメントの口ギングをアクティブ化すると、口ギングレベルが 5 (MySQL Shell 口ギングレベルのデフォルト) 以上に設定されている場合、ステートメントは情報メッセージとして MySQL Shell アプリケーションログファイルに書き込まれます。 口ギングレベルで追加の表示可能な場所が指定された場合、そこにもステートメントが送信されます。 verbose オプションが 1 以上に設定されている場合は、ステートメントも冗長出力としてコンソールに送信されます。 SQL ステートメントに含まれるパスワードは、口ギングおよび表示のためにマスクされ、記録または表示されません。

AdminAPI サンドボックス操作 (dba.deploySandboxInstance(), dba.startSandboxInstance(), dba.stopSandboxInstance(), dba.killSandboxInstance() および dba.deleteSandboxInstance()) によって実行される SQL ステートメントは、通常の AdminAPI 操作の口ギングをアクティブ化した場合でも、常に口ギングおよび冗長出力から除外されます。

AdminAPI 操作によって実行された SQL ステートメントを口グに記録するには、次のいずれかのオプションを選択します:

- MySQL Shell の起動時に --dba-log-sql コマンドラインオプションを使用します。
- MySQL Shell \option コマンドを使用して、dba.logSql MySQL Shell 構成オプションを設定します。 このコマンドの使用手順については、セクション10.4「MySQL Shell オプションの構成」 を参照してください。

- shell.options オブジェクトを使用して、dba.logSql MySQL Shell 構成オプションを設定します。 この構成インタフェースの使用手順については、セクション10.4「MySQL Shell オプションの構成」 を参照してください。

このオプションで使用可能な設定は次のとおりです:

0 AdminAPI 操作によって実行された SQL ステートメントを口グに記録しません。 この設定は、オプションがコマンドラインまたは構成ファイルで設定されていない場合のデフォルトの動作であり、一時的に必要な場合にのみ、使用後にこのタイプの口ギングを非アクティブ化するように設定できます。

1 SELECT ステートメント、SHOW ステートメントおよびサンドボックス操作によって実行されるステートメントを除き、AdminAPI 操作によって実行される SQL ステートメントを口グに記録します。

2 通常の AdminAPI 操作によって実行される SQL ステートメントを、SELECT および SHOW ステートメントを含めて完全に記録しますが、サンドボックス操作によって実行されるステートメントは記録しません。

MySQL Shell (--dba-log-sql) の起動時にコマンド行オプションで許可される値なしでオプションを指定したが、オプションを設定する他の方法では許可されない場合は、設定 1 が使用されます。

第 10 章 MySQL Shell のカスタマイズ

目次

10.1 起動スクリプトの操作 177
10.2 モジュール検索パスの追加 178
10.2.1 モジュール検索パスの環境変数 179
10.2.2 起動スクリプトのモジュール検索パス変数 179
10.3 プロンプトのカスタマイズ 179
10.4 MySQL Shell オプションの構成 180

MySQL Shell には、プリファレンスに合せて動作およびコード実行環境を変更するための次のカスタマイズオプションが用意されています:

- MySQL Shell が JavaScript または Python モードで起動されたときに実行される起動スクリプトを作成します。 セクション10.1「起動スクリプトの操作」を参照してください。
- JavaScript または Python モードの非標準モジュール検索パスを追加します。 セクション10.2「モジュール検索パスの追加」を参照してください。
- MySQL Shell プロンプトをカスタマイズします。 セクション10.3「プロンプトのカスタマイズ」を参照してください。
- 構成オプションを設定して、現在のセッションの MySQL Shell 動作を変更するか、永続的に変更します。 セクション10.4「MySQL Shell オプションの構成」を参照してください。

10.1 起動スクリプトの操作

MySQL Shell を JavaScript または Python モードで起動した場合、および JavaScript または Python モードに初めて切り替えた場合、MySQL Shell は実行する起動スクリプトを検索します。 起動スクリプトは、MySQL Shell が最初に対応する言語モードに入ったときに実行される指示を含む JavaScript または Python 固有のスクリプトです。 起動スクリプトを使用すると、次のいずれかの方法で JavaScript または Python コード実行環境をカスタマイズできます:

- Python または JavaScript モジュールの検索パスを追加します。
- グローバル関数または変数の定義。
- JavaScript または Python を使用して、その他の可能な初期化を実行します。

関連する起動スクリプトは、JavaScript または Python モードで MySQL Shell を起動または再起動したとき、および MySQL Shell の実行中にこれらのモードを初めて他のモードに変更したときにロードされます。 この後、MySQL Shell は起動スクリプトを再度検索しないため、起動スクリプトへの更新を実装するには、関連するモードにすでに入っている場合は MySQL Shell を再起動する必要があります。 MySQL Shell を SQL モードで起動するか、そのモードに切り替えると、起動スクリプトはロードされません。

起動スクリプトはオプションであり、カスタマイズに使用する場合は作成できます。 起動スクリプトの名前は次のようにする必要があります:

- JavaScript モードの場合: mysqlshrc.js
- Python モードの場合: mysqlshrc.py

次に示す任意の場所に起動スクリプトを配置できます。MySQL Shell は、指定された順序で、ファイル名が mysqlshrc の起動スクリプトと、初期化されるスクリプトモードに一致するファイル拡張子 (MySQL Shell が言語モードを指定せずに起動された場合、デフォルトで .js) を検索します。 MySQL Shell では、スクリプトモードで検出されたすべての適切な起動スクリプトが検出された順序で実行されることに注意してください。 2 つの異なる起動スクリプトで何かが定義されている場合は、後で実行されるスクリプトが優先されます。

1. プラットフォーム標準グローバル構成パス内。
   - Windows の場合: %PROGRAMDATA%\MySQL\mysqlsh\mysqlshrc.[js|py]

- Unix の場合: /etc/mysql/mysqlsh/mysqlshrc.[js|py]

2. MySQL Shell ホームフォルダの share/mysqlsh サブディレクトリ (環境変数 MYSQLSH_HOME で定義するか、MySQL Shell で識別できます)。 MYSQLSH_HOME が定義されていない場合、MySQL Shell は、mysqlsh バイナリを含む bin という名前のフォルダの親フォルダとして独自のホームフォルダを識別します (そのようなフォルダが存在する場合)。 (多くの標準インストールでは、MYSQLSH_HOME を定義する必要はありません。)

   - Windows の場合: %MYSQLSH_HOME%\share\mysqlsh\mysqlshrc.[js|py]
   - Unix の場合: $MYSQLSH_HOME/share/mysqlsh/mysqlshrc.[js|py]

3. mysqlsh バイナリを含むフォルダ内 (ただし、オプション 2 で説明されている MySQL Shell ホームフォルダが、予想される標準の場所で MySQL Shell によって指定も識別もされていない場合のみ)。

   - Windows の場合: <mysqlsh binary path>\mysqlshrc.[js|py]
   - Unix の場合: <mysqlsh binary path>/mysqlshrc.[js|py]

4. MySQL Shell ユーザー構成パス。環境変数 MYSQLSH_USER_CONFIG_HOME で定義されます。

   - Windows の場合: %MYSQLSH_USER_CONFIG_HOME%\mysqlshrc.[js|py]
   - Unix の場合: $MYSQLSH_USER_CONFIG_HOME/mysqlshrc.[js|py]

5. プラットフォームの標準ユーザー構成パス (オプション 4 で説明されている MySQL Shell ユーザー構成パスが指定されていない場合のみ)。

   - Windows の場合: %APPDATA%\MySQL\mysqlsh\mysqlshrc.[js|py]
   - Unix の場合: $HOME/.mysqlsh/mysqlshrc.[js|py]

10.2 モジュール検索パスの追加

JavaScript で require() 関数を使用する場合、または Python で import 関数を使用する場合、sys.path 変数にリストされている既知のモジュール検索パスを使用して、指定したモジュールが検索されます。MySQL Shell は sys.path 変数を初期化して、次のモジュール検索パスを含めます:

- モジュール検索パス環境変数 (JavaScript モードの場合は MYSQLSH_JS_MODULE_PATH、Python モードの場合は PYTHONPATH) で指定されたフォルダ。
- JavaScript の場合、MySQL Shell ホームフォルダのサブフォルダ share/mysqlsh/modules/js、または mysqlsh バイナリを含むフォルダのサブフォルダ/modules/js(ホームフォルダが存在しない場合)。
- Python の場合、Python 標準インポート機構の場合と同様に、インストールに依存するデフォルトパス。

MySQL Shell では、require() または import 関数を使用して組込みモジュール mysql および mysqlx をロードすることもでき、sys.path 変数を使用してこれらのモジュールを指定する必要はありません。

JavaScript モードの場合、MySQL Shell は、指定された場所 (優先順位に従って) で見つかった最初のモジュール、指定された名前にファイル拡張子 .js が付いたファイル、または指定された名前のフォルダに含まれる init.js ファイルをロードします。 Python モードでは、Python 標準インポート機構を使用して、MySQL Shell のすべてのモジュールをロードします。

JavaScript モードでは、MySQL Shell 8.0.19 から、MySQL Shell は require() 関数による ローカルモジュールのロードもサポートしています。 ./または ../の接頭辞が付いたモジュール名またはパスをバッチモードで指定すると、MySQL Shell は、現在実行されている JavaScript ファイルまたはモジュールを含むフォルダ内で指定されたモジュールを検索します。 対話型モードでは、これらの接頭辞のいずれかが指定されると、MySQL Shell は現在の作業ディレクトリ内を検索します。 そのフォルダにモジュールが見つからない場合、MySQL Shell は、sys.path 変数で指定された既知のモジュール検索パスの確認に進みます。

既知のモジュール検索パスを sys.path 変数に追加するには、JavaScript モードまたは Python モードのモジュール検索パス環境変数に追加するか (セクション10.2.1「モジュール検索パスの環境変数」 を参照)、MySQL Shell 起動スクリプト (JavaScript モードまたは Python モードの場合) を使用して直接 sys.path 変数に追加します (セクション

10.2.2「起動スクリプトのモジュール検索パス変数」 を参照)。 実行時に sys.path 変数を変更して、require() または import 関数の動作をすぐに変更することもできます。

10.2.1 モジュール検索パスの環境変数

モジュール検索パスにフォルダを追加するには、適切な言語固有のモジュール検索パス環境変数にフォルダを追加します。MySQL Shell を起動または再起動すると、MySQL Shell は既知のモジュール検索パスにこれらのフォルダを含めます。 検索パスにすぐに追加する場合は、sys.path 変数を直接変更します。

JavaScript の場合は、MYSQLSH_JS_MODULE_PATH 環境変数にフォルダを追加します。 この変数の値は、セミコロン文字で区切られたパスのリストです。

Python の場合は、PYTHONPATH 環境変数にフォルダを追加します。 この変数の値は、Windows プラットフォームではセミコロン文字、Unix プラットフォームではコロン文字で区切られたパスのリストです。

JavaScript の場合、環境変数に追加されたフォルダは sys.path 変数値の最後に配置され、Python の場合は最初に配置されます。

モジュールをロードするための Python の動作は MySQL Shell によって制御されないことに注意してください。Python の通常のインポート動作が適用されます。

10.2.2 起動スクリプトのモジュール検索パス変数

sys.path 変数は、MySQL Shell 起動スクリプト mysqlshrc.js(JavaScript モードの場合) または mysqlshrc.py(Python モードの場合) を使用してカスタマイズできます。 起動スクリプトとその場所の詳細は、セクション10.1「起動スクリプトの操作」 を参照してください。 起動スクリプトを使用して、モジュールパスを sys.path 変数に直接追加できます。

各起動スクリプトは関連する言語モードでのみ使用されるため、mysqlshrc.js for JavaScript モードで指定されたモジュール検索パスは、mysqlshrc.py にもリストされている場合、Python モードでのみ使用できます。

Python の場合は、mysqlshrc.py ファイルを変更して、必要なパスを sys.path 配列に追加します:

```
# Import the sys module
import sys

# Append the additional module paths
sys.path.append('~/custom/python')
sys.path.append('~/other/custom/modules')
```

JavaScript の場合は、mysqlshrc.js ファイルを変更して、必要なパスを sys.path 配列に追加します:

```
// Append the additional module paths
 sys.path = [...sys.path, '~/custom/js'];
 sys.path = [...sys.path, '~/other/custom/modules'];
```

sys.path 配列に追加する相対パスは、現在の作業ディレクトリに対して相対的に解決されます。

起動スクリプトは、JavaScript または Python モードで MySQL Shell を起動または再起動したとき、および MySQL Shell の実行中にこれらのモードのいずれかに初めて変更したときにロードされます。 この後、MySQL Shell は起動スクリプトを再度検索しないため、起動スクリプトへの更新を実装するには、関連するモードにすでに入っている場合は MySQL Shell を再起動する必要があります。 または、実行時に sys.path 変数を変更できます。この場合、require() または import 関数はただちに新しい検索パスを使用します。

10.3 プロンプトのカスタマイズ

MySQL Shell のプロンプトは、プロンプトテーマファイルを使用してカスタマイズできます。 プロンプトテーマファイルをカスタマイズするには、MYSQLSH_PROMPT_THEME 環境変数をプロンプトテーマファイル名に設定するか、テーマファイルを Linux および Mac の ~/.mysqlsh/prompt.json ディレクトリまたは Windows の %AppData%\MySQL\mysqlsh\prompt.json ディレクトリにコピーします。

ディレクトリのユーザー構成パスは、環境変数 MYSQLSH_USER_CONFIG_HOME を定義することで、すべてのプラットフォームでオーバーライドできます。 この変数の値は、Windows 上の %AppData%\MySQL\mysqlsh\または Unix 上の ~/.mysqlsh/に置き換わります。

プロンプトテーマファイルの形式は README.prompt ファイルで説明されており、プロンプトテーマファイルの例もいくつか含まれています。 起動時にプロンプトテーマファイルにエラーが見つかった場合は、エラーメッセージが出力され、デフォルトのプロンプトテーマが使用されます。 プロンプトテーマファイルの例には、特殊なフォント (SourceCodePro+Powerline+Awesome+Regular.ttf など) が必要なものがあります。 MYSQLSH_PROMPT_THEME 環境変数を空の値に設定すると、MySQL Shell では色のない最小限のプロンプトが使用されます。

カラー表示は、端末から使用可能なサポートによって異なります。 ほとんどの端末は、Linux および Mac で 256 色をサポートしています。 Windows では、カラーサポートには ANSI/VT100 エスケープをサポートするサードパーティ製ターミナルプログラムまたは Windows 10 が必要です。 デフォルトでは、MySQL Shell は端末タイプを検出し、色を適切に処理しようとします。 端末タイプで自動検出が機能しない場合、またはアクセシビリティ要件やその他の目的で色モードを変更する場合は、MySQL Shell で強制的に特定の色モードを使用するように環境変数 MYSQLSH_TERM_COLOR_MODE を定義できます。 この環境変数に指定できる値は、rgb, 256, 16 および nocolor です。

10.4 MySQL Shell オプションの構成

特定のプログラミング言語まで起動したり、特定の形式で出力を提供したりするなど、プリファレンスに一致するように MySQL Shell を構成できます。 構成オプションは、現在のセッションに対してのみ設定することも、MySQL Shell 構成ファイルへの変更を永続化することで永続的に設定することもできます。 すべてのオプションのオンラインヘルプが提供されています。 MySQL Shell \option コマンドを使用してオプションを構成できます。このコマンドは、すべての MySQL Shell モードで構成オプションのクエリーおよび変更に使用できます。 または、JavaScript および Python モードでは、shell.options オブジェクトを使用します。

有効な構成オプション

次の構成オプションは、\option コマンドまたは shell.options スクリプトインタフェースのいずれかを使用して設定できます:

optionName	DefaultValue	型	影響
autocomplete.nameCache	true	ブール	自動補完のためにデータベース名キャッシュを有効にします。
batchContinueOnError	false	boolean (READ ONLY)	SQL バッチモードでは、エラーが見つかった場合に処理を強制的に続行します。
credentialStore.excludeFilters	empty	array	自動パスワード記憶域が無効になっている URL の配列で、* および ? の glob 文字がサポートされています。
credentialStore.helper	プラットフォームに依存	文字列	パスワードのフェッチまたは格納に使用される資格証明ヘルパーの名前。 プラットフォームのデフォルトヘルパーを使用するために、特別な値 default がサポートされています。 特殊な値 disabled は、資格証明ストアを無効にします。
credentialStore.savePasswords	false	文字列	自動パスワード記憶域、サポートされている値を制御: always、prompt または never。
dba.gtidWaitTimeout	60	0 より大きい整数	GTID トランザクションが適用されるのを待機する時間 (秒)。AdminAPI 操作で必要な場合 (セクション 6.2.5「InnoDB クラスタの操作」 を参照)。

optionName	DefaultValue	型	影響
dba.logSql	0	0 から 2 の範囲の整数	AdminAPI 操作によって実行される SQL ステートメントをログに記録します (第9章「MySQL Shell のロギングおよびデバッグ」 を参照)。
dba.restartWaitTimeout	60	0 より大きい整数	リカバリ操作中にトランザクションが適用されるのを待機する時間 (秒)。 結合インスタンスが大量のデータをリカバリする必要がある場合に、より長いタイムアウトを構成するために使用します。 セクション 6.2.2.2「InnoDB クラスタでの MySQL クローンの使用」 を参照してください。
defaultCompress	false	ブール	すべてのグローバルセッションでクライアントとサーバーの間で送信される情報の圧縮をリクエストします。 クラシック MySQL プロトコル 接続にのみ影響します (セクション 4.3.4「圧縮接続の使用」 を参照)。
defaultMode	なし	string (sql、js または py)	MySQL Shell の起動時に使用するモード (SQL、JavaScript または Python)。
devapi.dbObjectHandles	true	ブール	X DevAPI db オブジェクトのテーブルおよびコレクション名ハンドルを有効にします。
history.autoSave	false	ブール	アプリケーションの終了時に、MySQL Shell コード履歴のエントリを保存 (true) またはクリア (false) します (セクション5.5「コード履歴」 を参照)。
history.maxSize	1000	整数	MySQL Shell コード履歴に格納するエントリの最大数。
history.sql.ignorePattern	*IDENTIFIED* : *PASSWORD*	文字列	これらのパターンに一致する文字列は、MySQL Shell コード履歴に追加されません。
interactive	true	boolean (READ ONLY)	対話型モードを有効にします。
logLevel	値が必要です	1 から 8 の範囲の整数	アプリケーションログのロギングレベルを設定します (第9章「MySQL Shell のロギングおよびデバッグ」 を参照)。

optionName	DefaultValue	型	影響
pager	なし	文字列	指定した外部ページャツールを使用して、テキストおよび結果を表示します。ツールのコマンドライン引数を追加できます (セクション4.6「ページャの使用」 を参照)。
passwordsFromStdin	false	ブール	端末ではなく stdin からパスワードを読み取ります。
resultFormat	テーブル	string (table, tabbed, vertical, json | json/pretty, ndjson | json/raw, json/array)	結果セットを出力するためのデフォルトの出力形式 (セクション5.7「出力形式」 を参照)。
sandboxDir	プラットフォームに依存	文字列	サンドボックスディレクトリ。 Windows ではデフォルトは C:\Users\MyUser\MySQL\mysql-sandboxes で、Unix システムではデフォルトは $HOME/mysql-sandboxes です。
showColumnTypeInfo	false	ブール	SQL モードでは、結果セットのカラムメタデータを表示します。
showWarnings	true	ブール	SQL モードでは、SQL 警告があれば自動的に表示されます。
useWizards	true	ブール	ウィザードモードを有効にします。
verbose	1	0 から 4 までの整数	コンソールへの詳細出力を有効にし、詳細レベルを設定します (第9章「MySQL Shell のロギングおよびデバッグ」 を参照)。

> 注記
>
> 文字列値では大文字と小文字が区別されます。
>
> 「READ ONLY」 としてリストされたオプションは変更できません。
>
> outputFormat オプションは非推奨になりました。 かわりに resultFormat を使用してください。

\option コマンドの使用

MySQL Shell \option コマンドを使用すると、すべてのモードで構成オプションのクエリーおよび変更が可能になり、JavaScript および Python モードに加えて SQL モードからの構成が可能になります。

このコマンドは次のように使用します:

- \option -h, --help [filter] - filter に一致するオプションのヘルプを出力します。
- \option -l, --list [--show-origin] - すべてのオプションをリストします。--show-origin は、値が最後にどのように変更されたかに関する情報でリストを拡張します。可能な値は次のとおりです:
  - Command line

- Compiled default
- Configuration file
- Environment variable
- User defined

- \option option_name - オプションの現在の値を出力します。
- \option [--persist] option_name value or name=value - オプションの値を設定し、--persist が指定されている場合は構成ファイルに保存します。
- \option --unset [--persist] <option_name> - オプション値をデフォルトにリセットし、--persist が指定されている場合は、MySQL Shell 構成ファイルからオプションを削除します。

> 注記
>
> option_name および filter の値では、大/小文字が区別されます。

option_name で使用可能な値のリストは、有効な構成オプション を参照してください。

shell.options 構成インタフェースの使用

shell.options オブジェクトは、MySQL Shell オプション値を変更するために JavaScript および Python モードで使用できます。 特定の方法を使用して、次のようにオプションまたはキーと値のペアを構成できます:

```
MySQL JS > shell.options['history.autoSave']=1
```

キーと値のペアのインタフェースに加えて、次のメソッドを使用できます:

- shell.options.set(optionName, value) - このセッションの optionName を value に設定します。変更は構成ファイルに保存されません。
- shell.options.setPersist(optionName, value) - このセッションの optionName を value に設定し、変更を構成ファイルに保存します。 Python モードでは、メソッドは shell.options.set_persist です。
- shell.options.unset(optionName) - optionName をこのセッションのデフォルト値にリセットします。変更は構成ファイルに保存されません。
- shell.options.unsetPersist(optionName) - optionName をこのセッションのデフォルト値にリセットし、変更を構成ファイルに保存します。 Python モードでは、メソッドは shell.options.unset_persist です。

オプション名は文字列として扱われるため、'文字で囲む必要があります。 optionName で使用可能な値のリストは、有効な構成オプション を参照してください。

コマンドを使用して、次のように MySQL Shell オプションを構成します:

```
MySQL JS > shell.options.set('history.maxSize', 5000)
MySQL JS > shell.options.setPersist('useWizards', 'true')
MySQL JS > shell.options.setPersist('history.autoSave', 1)
```

次のように、オプションをデフォルト値に戻します:

```
MySQL JS > shell.options.unset('history.maxSize')
MySQL JS > shell.options.unsetPersist('useWizards')
```

構成ファイル

MySQL Shell 構成ファイルには、セッション間で永続化されるようにするためのオプションの値が格納されます。 値は起動時に読み取られ、永続化機能を使用すると、設定は構成ファイルに保存されます。

構成ファイルの場所はユーザー構成パスで、ファイルの名前は options.json です。 環境変数 MYSQLSH_USER_CONFIG_HOME を定義してデフォルトのユーザー構成パスがオーバーライドされていない場合、構成ファイルへのパスは次のようになります:

- Windows の場合 %APPDATA%\MySQL\mysqlsh
- Unix の場合 ~/.mysqlsh ここで、~ はユーザーのホームディレクトリを表します。

構成ファイルは、構成オプションを初めてカスタマイズするときに作成されます。 このファイルは MySQL Shell によって内部的に保持されるため、手動で編集しないでください。 起動時に認識されないオプションまたは誤った値のオプションが構成ファイルに見つかった場合、MySQL Shell はエラーで終了します。

付録 A MySQL Shell コマンドリファレンス

目次

A.1 mysqlsh — MySQL Shell 185

この付録では、mysqlsh コマンドについて説明します。

A.1 mysqlsh — MySQL Shell

MySQL Shell は、MySQL 用の高度なコマンドラインクライアントおよびコードエディタです。 SQL に加えて、MySQL Shell には JavaScript および Python のスクリプト機能も用意されています。 MySQL Shell の使用の詳細は、MySQL Shell 8.0 を参照してください。 MySQL Shell が X プロトコル を介して MySQL Server に接続されている場合、X DevAPI を使用してリレーショナルデータとドキュメントデータの両方を操作できます。ドキュメントストアとしての MySQL の使用 を参照してください。MySQL Shell は AdminAPI が含まれていて InnoDB クラスタ が作業できます。第6章「MySQL AdminAPI の使用」 を参照してください。

ここで説明するオプションの多くは、MySQL Shell と MySQL Server インスタンス間の接続に関連しています。 詳しくはセクション4.3「MySQL Shell 接続」をご覧ください。

mysqlsh では、次のコマンドラインオプションがサポートされます。

表 A.1 「mysqlsh のオプション」

オプション名	説明	導入
--	API コマンドライン統合の開始	
--auth-method	使用する認証方式	
--cluster	InnoDB クラスタへの接続	8.0.4
--column-type-info	結果セットのカラムのメタデータの印刷	8.0.14
--compress	クライアントとサーバー間で送信される情報をすべて圧縮	8.0.14
--connect-timeout	グローバルセッションの接続タイムアウト	8.0.13
--credential-store-helper	パスワード用のシークレットストアヘルパー	8.0.12
--database	使用するスキーマ (--schema のエイリアス)	
--dba	MySQL 5.7 サーバーとの接続時に X プロトコルを有効にします	
--dba-log-sql	AdminAPI 操作によって実行される SQL ステートメントを記録	8.0.18
--dbpassword	サーバーに接続する際に使用するパスワード	
--dbuser	サーバーへの接続時に使用する MySQL ユーザー名	
--execute	コマンドを実行して終了	
--file	バッチモードで処理するファイル	
--force	エラーが発生した場合でも SQL およびバッチモードで続行	
--get-server-public-key	サーバーから RSA 公開キーをリクエスト	

オプション名	説明	導入
--help	ヘルプメッセージを表示して終了	
--histignore	履歴に追加されない文字列	8.0.3
--host	MySQL サーバーインスタンスが存在するホスト	
--import	ファイルまたは標準入力からの JSON ドキュメントのインポート	8.0.13
--interactive	バッチモードでのインタラクティブモードのエミュレート	
--js, --javascript	JavaScript モードで起動	
--json	JSON 形式での出力の出力	
--log-level	ロギングレベルの指定	
-ma	セッションのトランスポートプロトコルの自動検出	8.0.3
--mysql, -mc	クラシック MySQL プロトコル を使用したセッションの作成	8.0.3
--mysqlx, -mx	X プロトコルを使用したセッションの作成	8.0.3
--name-cache	アクティブなデフォルトスキーマに基づいたテーブル名の自動ロードの有効化	8.0.4
--no-name-cache	オートコンプリートの無効化	8.0.4
--no-password	この接続のパスワードが指定されていません	
--no-wizard, --nw	対話型ウィザードの無効化	
--pager	出力の表示に使用される外部ページャツール	8.0.13
--password	サーバーへの接続時に使用するパスワード (--dbpassword のエイリアス)	
--passwords-from-stdin	stdin からパスワードを読み取ります	
--port	接続用の TCP/IP ポート番号	
--py, --python	Python モードで起動	
--quiet-start	紹介情報を印刷せずに開始	
--recreate-schema	スキーマの削除および再作成	
--redirect-primary	InnoDB クラスタプライマリへの接続の確認	8.0.4
--redirect-secondary	InnoDB クラスタセカンダリへの接続の確認	
--result-format	このセッションの出力形式を設定	8.0.14
--save-passwords	パスワードをシークレットストアに格納する方法	8.0.12
--schema	使用するスキーマ	
--server-public-key-path	RSA 公開鍵を含むファイルへのパス名	
--show-warnings	存在する場合、各ステートメントの後に警告を表示 (SQL モード)	

オプション名	説明	導入
--socket	使用する Unix ソケットファイルまたは Windows 名前付きパイプ (クラシック MySQL プロトコルのみ)	
--sql	SQL モードで開始し、接続に使用するプロトコルを自動検出	
--sqlc	クラシック MySQL プロトコル 接続を使用した SQL モードでの起動	
--sqlx	X プロトコル接続を使用した SQL モードでの起動	8.0.3
--ssl-ca	信頼できる SSL 認証局のリストを含むファイル	
--ssl-capath	信頼できる SSL 認証局の証明書ファイルを含むディレクトリ	
--ssl-cert	X.509 証明書を含むファイル	
--ssl-cipher	使用する SSL 暗号の名前	
--ssl-crl	証明書失効リストを含むファイル	
--ssl-crlpath	証明書失効リストファイルを含むディレクトリ	
--ssl-key	X.509 キーを含むファイル	
--ssl-mode	サーバーへの接続に必要なセキュリティ状態	
--tabbed	タブ区切り形式で出力を表示	
--table	出力を表形式で表示します	
--tls-version	暗号化された接続に許可される TLS プロトコル	
--uri	URI 形式のセッション情報	
--user	サーバーへの接続時に使用する MySQL ユーザー名 (--dbuser のエイリアス)	
--verbose	コンソールへの詳細出力をアクティブ化します	8.0.17
--version	バージョン情報を表示して終了	
--vertical	すべての SQL 結果を縦に表示	

- --help, -?

  ヘルプメッセージを表示して終了します。

- --

  mysqlsh オプションのリストの最後と、MySQL ShellAPI コマンド行統合のためのコマンドとその引数の開始をマークします。 次の構文を使用して、コマンドラインから MySQL Shell グローバルオブジェクトのメソッドを実行できます:

  ```
  mysqlsh [options]  -- object method [arguments]
  ```

  詳しくはセクション5.8「API コマンドラインインタフェース」をご覧ください。

- --auth-method=method

  アカウントに使用する認証方式。 アカウントパスワードに使用される認証プラグインによって異なります。 クラシック MySQL プロトコル を使用した MySQL Shell 接続の場合は、caching_sha2_password などの認証プラグイ

ンの名前を指定します。 X プロトコル を使用する MySQL Shell 接続の場合は、次のいずれかのオプションを指定します:

AUTO	ライブラリで認証方式を選択します。
FALLBACK	ライブラリで認証方式を選択しますが、MySQL 5.7 と互換性のない認証方式は使用しません。
FROM_CAPABILITIES	サーバーインスタンスによって通知される機能を使用して、ライブラリで認証方法を選択できるようにします。
MYSQL41	プレーンテキストパスワードを送信しない、MySQL 4.1 以降でサポートされているチャレンジレスポンス認証プロトコルを使用します。 このオプションは、mysql_native_password 認証プラグインを使用するアカウントと互換性があります。
PLAIN	認証用のプレーンテキストパスワードを送信します。 このオプションは、暗号化された接続でのみ使用します。 SSL 接続がある場合、このオプションを使用して、caching_sha2_password 認証プラグインを使用するアカウントのキャッシュされた資格証明で認証できます。 Caching SHA-2 認証プラグインでの X プラグイン の使用を参照してください。
SHA256_MEMORY	メモリーに格納されているハッシュパスワードを使用して認証します。 このオプションは、非 SSL 接続がある caching_sha2_password 認証プラグインを使用するアカウントに対して、キャッシュされた資格証明で認証するために使用できます。 Caching SHA-2 認証プラグインでの X プラグイン の使用を参照してください。

- --cluster

  ターゲットサーバーが InnoDB クラスタの一部であることを確認し、その場合は cluster グローバル変数をクラスタオブジェクトに設定します。

- --column-type-info

  SQL モードでは、クエリーに対して返された結果セットを出力する前に、結果セット内の各カラムのメタデータ (カラムタイプや照合順序など) を出力します。

  カラムタイプは、MySQL Shell (Type) で使用されるタイプと元のデータベース (DBType) で使用されるタイプの両方として返されます。 クラシック MySQL プロトコル を使用する MySQL Shell 接続の場合、DBType はプロトコルによって返され、X プロトコル 接続の場合、DBType は使用可能な情報から推測されます。 カラムの長さ (Length) はバイト単位で返されます。

- --compress[={required|preferred|disabled}], -C [{required|preferred|disabled}]

  この接続を使用してクライアントとサーバー間で送信される情報の圧縮を制御します。 8.0.19 を介した MySQL Shell 8.0.14 では、このオプションは クラシック MySQL プロトコル 接続にのみ使用でき、required、preferred および disabled オプションは使用しません。 これらのリリースでは、--compress を指定すると、可能であれば圧縮がアクティブ化されます。 MySQL Shell 8.0.20 からは、X プロトコル 接続にも使用でき、オプションで required、preferred または disabled を指定できます。 --compress のみが MySQL Shell 8.0.20 から指定されている場合、その意味は --compress=required です。 すべてのリリースでの MySQL Shell 圧縮制御の使用の詳細は、セクション4.3.4「圧縮接続の使用」 を参照してください。

- --connect-timeout=ms

  コマンドライン引数で指定されたグローバルセッションの確立を MySQL Shell が待機する時間 (ミリ秒) を構成します。

- --credential-store-helper=helper

  パスワードの格納および取得に使用されるシークレットストアヘルパー。 セクション4.4「プラガブルパスワードストア」を参照してください。

- --database=name, -D name

使用するデフォルトスキーマ。 これは --schema のエイリアスです。

- --dba=enableXProtocol

  MySQL 5.7 サーバーとの接続で X プラグイン を有効にして、後続の接続に X プロトコル 接続を使用できるようにします。 クラシック MySQL プロトコル を使用した接続が必要です。 X プラグイン がデフォルトで有効になっている MySQL 8.0 サーバーには関係ありません。

- --dba-log-sql[=0|1|2]

  AdminAPI 操作によって実行される SQL ステートメントをログに記録します (サンドボックス操作を除く)。 デフォルトでは、--log-level および --verbose オプションが設定されている場合でも、このカテゴリのステートメントは MySQL Shell アプリケーションログファイルに書き込まれず、コンソールに冗長出力として送信されません。 オプションの値は 0 から 2 の範囲の整数です。0 はこのカテゴリのステートメントを記録または表示しません。これは、オプションを指定しない場合のデフォルトの動作です。1 は、SELECT ステートメントおよび SHOW ステートメントを除き、AdminAPI 操作によって実行される SQL ステートメントをログに記録します (これは、コマンドラインで値なしでオプションを指定した場合のデフォルト設定です)。2 は、通常の AdminAPI 操作によって実行される SQL ステートメントをすべてログに記録します。 詳しくは第9章「MySQL Shell のロギングおよびデバッグ」をご覧ください。

- --dbpassword[=password]

  MySQL Shell のバージョン 8.0.13 では非推奨です。 かわりに --password[=password]を使用してください。

- --dbuser=user_name

  MySQL Shell のバージョン 8.0.13 では非推奨です。 かわりに --user=user_name を使用してください。

- --execute=command, -e command

  現在アクティブな言語を使用してコマンドを実行し、終了します。 このオプションは、--file=file_name オプションと相互に排他的です。

- --file=file_name, -f file_name

  バッチモードで処理するファイルを指定します。 この後に指定したオプションは、処理済ファイルの引数として使用されます。

- --force

  エラーが発生した場合でも、SQL およびバッチモードで処理を続行します。

- --histignore=strings

  MySQL Shell 履歴に追加されない文字列を指定します。 文字列はコロンで区切られます。 照合では大文字と小文字は区別されず、ワイルドカード * および ? を使用できます。 無視されるデフォルトの文字列は、「*IDENTIFIED*:*PASSWORD*」 として指定されます。 セクション5.5「コード履歴」を参照してください。

- --host=host_name, -h host_name

  指定されたホストの MySQL サーバーに接続します。 Windows では、--host=. または -h . (ピリオドとしてホスト名を指定) を指定すると、MySQL Shell はデフォルトの名前付きパイプ (MySQL という名前) または --socket オプションを使用して指定した代替の名前付きパイプを使用して接続します。

- --get-server-public-key

  MySQL Shell は --get-server-public-key と同等です。

  --server-public-key-path=file_name が指定され、有効な公開キーファイルが指定されている場合は、--get-server-public-key よりも優先されます。

  > **重要**
  >
  > クラシック MySQL プロトコル 接続でのみサポートされます。

SHA-2 プラガブル認証のキャッシュを参照してください。

- --import

  JSON インポートユーティリティを使用して、JSON ドキュメントをファイルまたは標準入力から MySQL Server コレクションまたはリレーショナルテーブルにインポートします。 その手順は、セクション8.2「JSON インポートユーティリティ」を参照してください。

- --interactive[=full], -i

  バッチモードで対話モードをエミュレートします。

- --js, --javascript

  JavaScript モードで起動します。

- --json[={off|pretty|raw}]

  このセッションからの MySQL Shell 出力の JSON ラッピングを制御します。 このオプションは、テストの一環としてなど、MySQL Shell を他のプログラムとインタフェースするためのものです。 JSON 形式を使用するようにクエリー結果の出力を変更するには、--result-format を参照してください。

  --json オプションに値または pretty の値がない場合、出力は整形出力 JSON として生成されます。 値が raw の場合、出力は RAW JSON 形式で生成されます。 いずれの場合も、--result-format オプションとそのエイリアス、および resultFormat MySQL Shell 構成オプションの値は無視されます。 値が off の場合、JSON ラッピングは実行されず、結果セットは --result-format オプションまたは resultFormat 構成オプションで指定された形式で通常どおり出力されます。

- --log-level=N

  MySQL Shell アプリケーションログファイルのロギングレベルを変更するか、ファイルへのロギングを無効にします。 このオプションには、1 から 8 の範囲の整数、none, internal, error, warning, info, debug, debug2 または debug3 のいずれかの値が必要です。 1 または none を指定すると、アプリケーションログファイルへのロギングが無効になります。 このオプションを指定しない場合、レベル 5 (info) がデフォルトです。 第9章「MySQL Shell のロギングおよびデバッグ」を参照してください。

- -ma

  MySQL Shell のバージョン 8.0.13 では非推奨です。 X プロトコル を使用してセッション接続を自動的に作成し、X プロトコル が使用できない場合は クラシック MySQL プロトコル にフォールバックします。

- --mysql, --mc

  クラシック MySQL プロトコル 接続を使用するように、起動時に作成されるグローバルセッションを設定します。 --mc オプションは、MySQL Shell 8.0.13 の以前の単一ハイフンの -mc オプションに置き換わります。

- --mysqlx, --mx

  X プロトコル 接続を使用するために起動時に作成されるグローバルセッションを設定します。 --mx オプションは、MySQL Shell 8.0.13 の以前の単一ハイフンの -mx オプションに置き換わります。

- --name-cache

  アクティブなデフォルトスキーマに基づいたテーブル名の自動ロードを有効にします。

- --no-name-cache, -A

  アクティブなデフォルトスキーマおよび DevAPI db オブジェクトに基づいて、自動補完のためのテーブル名のロードを無効にします。 \rehash を使用して、名前情報を手動でリロードします。

- --no-password

  サーバーへの接続時に、ユーザーがパスワードなしのアカウントを持っている場合 (セキュアではなく推奨されない)、またはソケットピア資格証明認証が使用されている場合 (Unix ソケット接続の場合)、--no-password を使用してパスワードが指定されず、パスワードプロンプトが不要であることを明示的に指定する必要があります。

- --no-wizard, -nw

  接続、dba.configureInstance()、Cluster.rebootClusterFromCompleteOutage() などの操作によって提供される対話型ウィザードを無効にします。 このオプションは、MySQL Shell をスクリプト化し、対話型プロンプトを表示しない場合に使用します。 詳細は、セクション5.6「バッチコード実行」 および セクション5.8「API コマンドラインインタフェース」 を参照してください。

- --pager=name

  MySQL Shell で使用される外部ページャツール。SQL モードで実行されるステートメントおよびオンラインヘルプなどの選択された他のコマンドのテキスト出力を表示します。 ページャを設定しない場合は、PAGER 環境変数で指定されたページャが使用されます。 セクション4.6「ページャの使用」を参照してください。

- --passwords-from-stdin

  端末からではなく標準入力からパスワードを読み取ります。 このオプションは、パスワードプロンプトなどの他のパスワードの動作には影響しません。

- --password[=password], -ppassword

  サーバーに接続する際に使用するパスワードです。 MySQL Shell への接続に使用できるパスワードの最大長は 128 文字です。

  - --password=password (-ppassword) に値を指定すると、接続に使用されるパスワードが指定されます。 長い形式の --password= では、オプションとその値の間に空白ではなく等号を使用する必要があります。 短い形式の -p では、オプションとその値の間に空白を入れないでください。 いずれの場合もスペースが使用される場合、値はパスワードとして解釈されず、別の接続パラメータとして解釈される可能性があります。

    コマンド行でのパスワード指定は、セキュアでないと考えるべきです。 パスワードセキュリティーのためのエンドユーザーガイドラインを参照してください。 オプションファイルを使用すれば、コマンド行でパスワードを指定することを回避できます。

  - 値がなく等号がない --password または値がない -p は、パスワードプロンプトを要求します。

  - 空の値を持つ --password= は、ユーザーがパスワードなしで接続していることを指定する --no-password と同じ効果があります。 サーバーに接続するときに、ユーザーがパスワードなしのアカウントを持っている場合 (セキュアではなく推奨されない)、またはソケットピア資格証明認証が使用されている場合 (Unix ソケット接続の場合)、次のいずれかの方法を使用して、パスワードが指定されず、パスワードプロンプトが不要であることを明示的に指定する必要があります。

- --port=port_num, -P port_num

  接続に使用する TCP/IP ポート番号。 デフォルトはポート 33060 です。

- --py, --python

  Python モードで起動します。

- --pym

  指定した Python モジュールを MySQL ShellPython モードでスクリプトとして実行します。--pym は、Python -m コマンドラインオプションと同じ方法で動作します。 このオプションは、MySQL Shell 8.0.22 から使用できます。

- --quiet-start[=1|2]

  紹介情報を印刷せずに開始します。MySQL Shell は通常、製品に関する情報、セッションに関する情報 (デフォルトのスキーマや接続 ID など)、警告メッセージ、および起動時と接続時に返されるエラーを出力します。 --quiet-

start を値なしまたは値 1 で指定すると、MySQL Shell 製品に関する情報は出力されませんが、セッション情報、警告およびエラーが出力されます。 値が 2 の場合、エラーのみが出力されます。

- --recreate-schema

  URI のような接続文字列の一部として、または --schema、--database または -D オプションを使用して、接続オプションで指定されたスキーマを削除して再作成します。 スキーマが存在する場合は削除されます。

- --redirect-primary

  ターゲットサーバーが InnoDB クラスタ または InnoDB ReplicaSet の一部であることを確認し、プライマリでない場合はプライマリを検索して接続します。このオプションの使用時に次のいずれかに該当する場合、MySQL Shell はエラーで終了します:

  - インスタンスが指定されていない
  - InnoDB クラスタ では、グループレプリケーションはアクティブではありません
  - InnoDB クラスタメタデータが存在しません
  - クォーラムがありません

- --replicaset

  ターゲットサーバーが InnoDB ReplicaSet に属していることを確認し、属している場合は、rs グローバル変数に InnoDB ReplicaSet を移入します。 その後、rs.status() を発行するなどして、rs グローバル変数を使用して InnoDB ReplicaSet を管理できます。

- --redirect-secondary

  ターゲットサーバーが単一プライマリ InnoDB クラスタまたは InnoDB ReplicaSet の一部であることを確認し、セカンダリでない場合はセカンダリを検索して接続します。このオプションの使用時に次のいずれかに該当する場合、MySQL Shell はエラーで終了します:

  - InnoDB クラスタ では、グループレプリケーションはアクティブではありません
  - InnoDB クラスタメタデータが存在しません
  - クォーラムがありません
  - クラスタはシングルプライマリモードではなく、マルチプライマリモードで実行されています
  - たとえば、サーバーインスタンスが 1 つしかないため、セカンダリは使用できません

- --result-format={table|tabbed|vertical|json|json/pretty|ndjson|json/raw|json/array}

  このセッションの resultFormat MySQL Shell 構成オプションの値を設定します。 形式は次のとおりです:

table	対話型モードのデフォルト。ただし、構成ファイルの resultFormat 構成オプションに別の値が永続的に設定されていないかぎり、そのデフォルトが適用されます。 --table エイリアスも使用できます。
tabbed	バッチモードのデフォルト。ただし、構成ファイルの resultFormat 構成オプションに別の値が永続的に設定されていないかぎり、そのデフォルトが適用されます。 --tabbed エイリアスも使用できます。
vertical	SQL クエリーの\G 終端文字と同等の出力を生成します。 --vertical または -E のエイリアスも使用できます。
json または json/pretty	整形出力 JSON を生成します。
ndjson または json/raw	改行で区切られた RAW JSON を生成します。
json/array	JSON 配列にラップされた RAW JSON を生成します。

  --json コマンドラインオプションを使用してセッションの出力の JSON ラッピングをアクティブ化する場合、--result-format オプションとそのエイリアス、および resultFormat 構成オプションの値は無視されます。

- --save-passwords={always|prompt|never}

  パスワードを自動的にシークレットストアに格納するかどうかを制御します。always は、パスワードがストアにすでに存在する場合、またはサーバー URL がフィルタによって除外されている場合を除き、パスワードが常に格納されることを意味します。never は、パスワードが格納されないことを意味します。prompt(デフォルト) は、ユーザーがパスワードを格納するかどうかを尋ねられることを意味します。 セクション4.4「プラガブルパスワードストア」を参照してください。

- --schema=name, -D name

  使用するデフォルトスキーマ。

- --server-public-key-path=file_name

  MySQL Shell は --server-public-key-path と同等です。

  --server-public-key-path=file_name が指定され、有効な公開キーファイルが指定されている場合は、--get-server-public-key よりも優先されます。

  > **重要**
  >
  > クラシック MySQL プロトコル 接続でのみサポートされます。

  caching_sha2_password plugin SHA-2 プラガブル認証のキャッシュ を参照してください。

- --show-warnings={true|false}

  true(デフォルト) が SQL モードで指定されている場合、MySQL Shell では各 SQL ステートメントの後に警告が表示されます (存在する場合)。 false を指定した場合、警告は表示されません。

- --socket[=path], -S [path]

  Unix では、パスが指定されている場合、パスは接続に使用する Unix ソケットファイルの名前です。 値なしで等号なしで --socket を指定した場合、または値なしで -S を指定した場合は、適切なプロトコルのデフォルトの Unix ソケットファイルが使用されます。

  Windows の場合、パスは接続に使用する名前付きパイプの名前です。 パイプ名では大文字と小文字は区別されません。 Windows では、パスを指定する必要があり、--socket オプションは クラシック MySQL プロトコル セッションでのみ使用できます。

  Unix では localhost 以外のポートまたはホスト名を指定し、Windows ではピリオド (.) を指定した場合は、ソケットを指定できません。

- --sql

  SQL モードで開始し、接続情報の一部として指定されていない場合に使用するプロトコルを自動検出します。 使用するプロトコルが指定されていない場合、デフォルトで X プロトコル 接続に設定され、クラシック MySQL プロトコル 接続に戻ります。 接続で特定のプロトコルを強制的に使用するには、--sqlx または --sqlc のオプションを参照してください。 または、URI のような接続文字列の一部として使用するプロトコルを指定するか、--port オプションを使用します。 詳細は、セクション4.3「MySQL Shell 接続」 および「MySQL Shell ポートリファレンス」を参照してください。

- --sqlc

  たとえば、X プロトコル をサポートしていないサーバーで MySQL Shell を使用するために、クラシック MySQL プロトコル を使用するように接続を強制する SQL モードで起動します。 接続の一部としてポートを指定しない場合、このオプションを指定すると、MySQL Shell はデフォルトの クラシック MySQL プロトコル ポート (通常 3306) を使用します。 接続先のポートは クラシック MySQL プロトコル をサポートしている必要があるため、たとえば、指定する接続で X プロトコル のデフォルトポート 33060 が使用されている場合、接続はエラーで失敗します。 詳細は、セクション4.3「MySQL Shell 接続」 および「MySQL Shell ポートリファレンス」を参照してください。

- --sqlx

  SQL モードで起動し、X プロトコル を使用するように接続を強制します。 接続の一部としてポートを指定しない場合、このオプションを指定すると、MySQL Shell はデフォルトの X プロトコル ポート (通常 33060) を使用します。 接続先のポートは X プロトコル をサポートする必要があるため、たとえば、指定した接続で クラシック MySQL プロトコル のデフォルトポート 3306 が使用されている場合、接続はエラーで失敗します。 詳細は、セクション4.3「MySQL Shell 接続」 および「MySQL Shell ポートリファレンス」を参照してください。

- --ssl*

  --ssl で始まるオプションは、SSL を使用してサーバーに接続することを許可するかどうかを指定し、SSL 鍵および証明書を検索する場所を指定します。 mysqlsh の SSL オプションは、MySQL Server の SSL オプションと同じように機能します。詳細は、暗号化接続のコマンドオプション を参照してください。

  mysqlsh は、次の SSL オプションを受け入れます: --ssl-mode, --ssl-ca, --ssl-capath, --ssl-cert, --ssl-cipher, --ssl-crl, --ssl-crlpath, --ssl-key, --tls-version。

- --tabbed

  対話モードで結果をタブ区切り形式で表示します。 そのモードのデフォルトはテーブル形式です。 このオプションは、--result-format=tabbed オプションのエイリアスです。

- --table

  バッチモードで結果をテーブル形式で表示します。 そのモードのデフォルトはタブ区切り形式です。 このオプションは、--result-format=table オプションのエイリアスです。

- --uri=str

  起動時に接続を作成し、URI 類似文字列またはキーと値のペアを使用したサーバーへの接続 で説明されている URI のような文字列に接続オプションを指定します。

- --user=user_name, -u user_name

  サーバーへの接続時に使用する MySQL ユーザー名。

- --verbose[=0|1|2|3|4]

  コンソールへの詳細出力をアクティブ化し、詳細のレベルを指定します。 値は 0 から 4 の範囲の整数です。0 はメッセージを表示しません。これは、オプションを指定しない場合のデフォルトの冗長性設定です。1 は、エラー、警告および情報メッセージを表示します (これは、値を指定せずにコマンドラインでオプションを指定した場合のデフォルト設定です)。2、3 および 4 は、より高いレベルのデバッグメッセージを追加します。 詳しくは第9章「MySQL Shell のロギングおよびデバッグ」をご覧ください。

- --version, -V

  MySQL Shell のバージョンを表示して終了します。

- --vertical, -E

  \G ターミネータが SQL クエリーに使用されている場合と同様に、結果を垂直に表示します。 このオプションは、--result-format=vertical オプションのエイリアスです。