Red Hat Integration 2020.q1

Data Virtualization のクライアントの開発

テクノロジープレビュー - クライアント開発者向けガイド

Last Updated: 2023-02-28

Red Hat Integration 2020.q1 Data Virtualization のクライアントの開発

テクノロジープレビュー - クライアント開発者向けガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

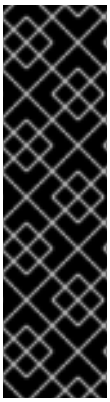
Data Virtualization に接続し、クライアントインターフェースを介してデータにアクセスする方法を説明します。

目次

第1章 DATA VIRTUALIZATION のクライアントの開発

本書は、Data Virtualization と対話するサードパーティーのアプリケーションの作成を試行する開発者向けです。接続メカニズム、JDBC API へのエクステンション、ODBC、SSL などの情報を見つけることができます。

Data Virtualization を調べる前に、VDB の基本的な構成要素をいくつか理解しておくことは非常に重要ですか？モデルとはどうでしょうか。これには、短い概要 を参照してください。



重要

Data Virtualization はテクノロジープレビューの機能です。テクノロジープレビュー機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

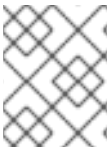
第2章 JDBC の互換性

Data Virtualization は、最新の仕様と互換性のある Java バージョンに基づいてほとんどの JDBC API を実装する強力な JDBC ドライバーを提供します。JDBC と連携するように設計されているほとんどのツールは、Data Virtualization ドライバーとシームレスに動作する必要があります。不明な場合は、実装済みの機能について は、「互換性のない JDBC メソッド 」を参照してください。

JDBC 以外でも、Data Virtualization は非同期処理、フェデレーションなどの機能のために JDBC 拡張も提供しています。

2.1. 生成されたキー

Data Virtualization は、JDBC ソースおよび SERIAL プライマリーキー列を使用して、Data Virtualization temp テーブルから生成されたキーを返すことができます。ただし、現在の実装は生成された最後のキーセットのみを返し、ソースから鍵の結果を直接返します。他の中間処理の表示は実行されません。ほとんどのシナリオ（単一のソース挿入）の場合は、この処理で十分です。生成した各キーを返す複数のテーブルを挿入するためにトリガーではなく FOR EACH ROW を使用する場合は、カスタムソリューションを開発しないといけない場合があります。返された生成されたキーも操作する UDF を開発できます。生成したキーを処理する **org.teiid.CommandContext** メソッドを参照してください。



注記

JDBC バッチ化された更新を使用して値をソーステーブルに挿入する場合は、Generated Keys を使用することはできません。

2.2. DATA VIRTUALIZATION サーバーへの接続

Data Virtualization JDBC API は、Data Virtualization にデプロイされた Virtual Database(VDB)への Java Database Connectivity(JDBC)アクセスを提供します。Data Virtualization JDBC API は JDBC 4.0 仕様と互換性がありますが、一部のメソッドと互換性がありません。アップデータブル結果セットや SQL3 データ型など、一部の高度な機能を使用することはできません。

Data Virtualization サーバーに接続する Java クライアントアプリケーションは、最低でも Java 1.8 JDK を使用する必要があります。以前のバージョンの Java は互換性がありません。ターゲット JRE と互換性のある以前の Teiid バージョンのクライアントドライバーの使用を試みることができます。

Data Virtualization に VDB をデプロイした後、クライアントアプリケーションは Data Virtualization Server に接続し、JDBC API を使用してデプロイされた VDB に対して SQL クエリーを発行することができます。JDBC を初めて使用する場合は、JDBC に関する Java のドキュメントを参照してください。Data Virtualization には、ダウンロード にある teiid-1.3.0-jdbc.jar が同梱されます。

また、コーディネートを使用して Maven リポジトリー https://oss.sonatype.org/content/repositories/releases/ から Data Virtualization JDBC を取得することもできます。

```
<dependency>
  <groupId>org.teiid</groupId>
  <artifactId>teiid</artifactId>
  <classifier>jdbc</classifier>
  <version>1.3.0</version>
</dependency>
```

クライアント JAR の重要なクラス：

- **org.teiid.jdbc.TeiidDriver**- DriverManager クラスを使用した JDBC 接続を許可します。
- **org.teiid.jdbc.TeiidDatasource**- DataSource XADataSource クラスを使用した JDBC 接続を許可します。このクラスを使用して管理または XA 接続を作成する必要があります。

Data Virtualization Server との接続を確立したら、標準の JDBC API クラスを使用してメタデータを補間し、クエリーを実行できます。

2.2.1. OpenTracing の互換性

OpenTracing はクライアントドライバーに対してオプションです。リモート接続がスパンを伝播させるには、ドライバーに適切な OpenTracing jar がクラスパスにある必要があります。これは、maven 依存関係を使用して実行できます。

```
<dependency>
  <groupId>io.opentracing</groupId>
  <artifactId>opentracing-util</artifactId>
  <version>${version.opentracing}</version>
</dependency>
```

ここで、version.opentracing(0.31 for Data Virtualization 11.0)は、プロジェクトのインテグレーション bom で定義されます。

または、Data Virtualization クライアント jar が使用されるツールまたは他の環境で必要に応じて、opentracing-util、opentracing-api、および opentracing-noop jar を手動で含めることができます。

クライアントおよびサーバーの OpenTracing サポートには、それぞれのランタイムに適切なトレースクライアントがインストールされ、GlobalTracer 経由で利用可能である必要があります。

2.2.2. ドライバー接続

org.teiid.jdbc.TeiidDriver をドライバークラスとして使用します。

JDBC 接続に以下の URL 形式を使用します。

```
jdbc:teiid:<vdb-name>[@mm[s]://<host>:<port>][;prop-name=prop-value]*
```

注記

JDBC クライアントには、JRE とサーバーの互換性に関する考慮事項があります。特に指定がない場合は、クライアント jar は通常、サーバーの 1 つのメジャーバージョンと後方互換性があります。修正や機能がクライアントに対して行われるため、クライアントの最新の状態を維持します。

URL コンポーネント

1. <vdb-name>: 接続先の VDB の名前。必要に応じて、VDB 名内にバージョン情報を含めることもできます。たとえば、「myvdb.2"」の場合、これは以下のように定義された「version=2"」接続プロパティーを指定することに相当します。ただし、この形式で vdb 名を使用すると、同時に「version」プロパティーは許可されません。
2. MM - Data Virtualization JDBC プロトコルを定義し、mmss がセキュアなチャンネルを定義します（詳細は「 SSL クライアント接続」 を参照）。

3. <host>: Data Virtualization Server がインストールされているサーバーを定義します。IPv6 バインディングアドレスをホスト名として使用している場合は、角括弧に配置します。例：[::1]

4. <port> - Data Virtualization Server が受信 JDBC 接続をリッスンするポートを定義します。

5. [prop-name=prop-value] - さらに、セミコロンで区切られた名前/値のペアをいくつでも指定できます [;]。互換性のある URL プロパティーはすべて、connection properties セクションで定義されます。プロパティー値は、予約文字が含まれる場合には URL エンコードする必要があります（例： '?'、'=', ';' など）。

2.2.2.1. ローカル接続

VM 内接続を作成するには、プロトコルおよび host/port: jdbc:teiid:vdb-name;props を省略します。

2.2.2.2. URL 接続プロパティー

以下の表は、Data Virtualization JDBC Driver URL 接続文字列または Data Virtualization JDBC Data Source クラスで使用できるすべての接続プロパティーを示しています。

表2.1 Connection Properties

プロパティー名	タイプ	説明
ApplicationName	**文字列**	クライアントアプリケーションの名前。管理者が接続を特定できるようにします。
FetchSize	**int**	結果セットのサイズです。500.gitops0 の場合はデフォルトのサイズで、デフォルトが使用されることを示します。
partialResultsMode	**boolean**	部分的な結果モードを有効/無効にします。デフォルト: falsePartial Results Mode セクションを参照してください。
autoCommitTxn	**文字列**	"autoCommit" が "true" に設定されている場合にのみ該当します。データの整合性を維持するために、実行されたコマンドを Data Virtualization エンジン内でトランザクション的にラップする方法を決定します。 ● ON: 常に分散トランザクションでコマンドをラップします。 ● OFF - 分散トランザクションでコマンドをラップしません。 ● DETECT（デフォルト）- 実行したコマンドが複数のソースにまたがる場合は、分散トランザクションを自動的に使用します。詳細のトランザクション
disableLocalTxn	**boolean**	"true" の場合は、ローカルトランザクションで autoCommit 設定、commit、および rollback は無視されます。デフォルト: false
user	**文字列**	ユーザー名

プロパティー名	タイプ	説明
パスワード	**文字列**	ユーザーの認証情報
ansiQuotedIdentifiers	**boolean**	SQL の二重引用符で囲んだエントリーの解析動作を設定します。デフォルトは true で、引用符付きのエントリーを識別子として解析します。false に設定すると、有効な文字列リテラルである引用符で囲まれた値が文字列リテラルとして解析されます。
version	**integer**	VDB のバージョン番号
resultSetCacheMode	**boolean**	ResultSet キャッシュはオン/オフになっています。デフォルト: false
autoFailover	**boolean**	true の場合、通信例外後に新しいサーバーインスタンスが自動的に選択されます。デフォルト: false接続をプールからパージできるため、通常は接続を管理する場合には必要ありません。埋め込みモードで true の場合、接続は同じ名前/バージョンの新しい VDB に再接続します。
SHOWPLAN	**文字列**	（通常は接続プロパティーとして設定されていません）ON、OFF,DEBUG; ● ON は、クエリー計画と結果を返します。 ● DEBUG はクエリープランナーのデバッグ情報をログに出力し、結果と共に返します。プランとログはどちらも JDBC API エクステンションで利用できます。 ● デフォルトは OFF です。
NoExec	**文字列**	（通常は接続プロパティーとして設定されていません）は ON、OFF、ON はクエリーの実行を防ぎますが、解析と計画は引き続き発生します。デフォルトは OFF です。
PassthroughAuthentication	**boolean**	「ローカル」接続にのみ適用されます。このオプションを「true」に設定すると、Data Virtualization は呼び出し元スレッドですでに認証されたセキュリティーコンテキストを検索します。見つかった場合は、そのユーザー認証情報を使用してセッションを作成します。また、データ仮想化は、接続のライフサイクル中に同じユーザーがこの接続を使用していることも確認します。呼び出しスレッドで異なるセキュリティーコンテキストを見つけた場合は、新規ユーザーが Data Virtualization にログインすることができるにも拘らず、接続上のアイデンティティーを切り替えます。

プロパティー名	タイプ	説明
useCallingThread	**boolean**	「ローカル」接続にのみ適用されます。このオプションが「true」に設定されている場合（デフォルト）、呼び出しスレッドを使用してクエリーを処理します。false の場合、エンジンスレッドが使用されます。
QueryTimeout	**integer**	デフォルトのクエリータイムアウト（秒単位）。0 でなければなりません。0 はタイムアウトなしであることを示します。**Statement.setQueryTimeout** で上書きできます。デフォルトは 0 です。
useJDBC4ColumnNameAndLabelSemantics	**boolean**	JDBC4 で変更され、エイリアスのない列名を ResultSetMetadata 列名として返すようになりました。この前に、列エイリアスを使用して列名が返されていました。このプロパティーを false に設定すると、JDBC3 以前との後方互換性が可能になります。デフォルトは true です。
jaasName	**文字列**	JAAS 設定名。GSS 認証を設定する場合にのみ適用されます。デフォルトは Data Virtualization です。GSS に必要な設定については、『セキュリティーガイド』を参照してください。
kerberosServicePrincipleName	**文字列**	Kerberos で認証された原則の名前。GSS 認証を設定する場合にのみ適用されます。GSS に必要な設定については、『セキュリティーガイド』を参照してください。
encryptRequest	**boolean**	SSL ソケット接続のみに適用されます。 「true」の場合、要求メッセージとすべての関連付けペイロードは、接続暗号ャーを使用して暗号化されます。 デフォルト: false
disableResultSetFetchSize	**boolean**	場合によっては、ツールによっては、結果の処理に望ましくないフェッチサイズを選択する場合があります。ResultSet.setFetchSize を許可する場合は true に設定します。デフォルト: false
loginTimeout	**integer**	ログインのタイムアウト（秒単位）。0 でなければなりません。0 は特定のタイムアウトなしを示しますが、他のタイムアウトが適用される可能性があります。接続をほぼ作成できない場合、タイムアウト値は例外が発生します。デフォルトの 0 は、ログインが永久に待機されることを意味するものではありません。通常、アクティブな vdb で、その時点でログインに失敗します。ただし、vdb バージョンを指定するローカル接続は、デフォルトで org.teiid.clientVdbLoadTimeoutMillis まで待機できます。

プロパティー名	タイプ	説明
reportAsViews	**boolean**	DatabaseMetaData が Data Virtualization ビューを VIEW テーブルタイプとして報告する場合。false の場合、Data Virtualization ビューが TABLE として報告されます。デフォルトは true です。

2.2.2.3. クライアント SSL 設定

以下のセクションでは、各 SSL モードに必要なプロパティーを定義します。SSL を有効にして Data Virtualization Server に接続する場合は、JDBC 接続 URL の **「mm」ではなく、「mms」**プロトコルを使用する **必要** があります。

```
jdbc:teiid:<myVdb>@mms://<host>:<port>
```

クライアントが 1 方向または 2 方向 SSL を有効にするように設定できるプロパティーは 2 種類あります。

2.2.2.3.1. オプション 1: Java SSL プロパティー

これらは任意の JVM で SSL を設定する標準的な Java 定義のシステムプロパティーであり、Data Virtualization は SSL の使用で一意ではありません。クライアントの仮想マシンプロセスに以下のシステムプロパティーを提供します。

1 方向 SSL

```
-Djavax.net.ssl.trustStore=<dir>/server.truststore (required)
-Djavax.net.ssl.trustStorePassword=<password> (optional)
-Djavax.net.ssl.keyStoreType (optional)
```

2 方向 SSL

```
-Djavax.net.ssl.keyStore=<dir>/client.keystore (required)
-Djavax.net.ssl.keyStrorePassword=<password> (optional)
-Djavax.net.ssl.trustStore=<dir>/server.truststore (required)
-Djavax.net.ssl.trustStorePassword=<password> (optioanl)
-Djavax.net.ssl.keyStroreType=<keystore type> (optional)
```

2.2.2.3.2. オプション 2: データ仮想化固有のプロパティー

上記の「javax」ベースのプロパティーがすでにホストプロセスによって使用されている場合は、このオプションを使用します。たとえば、クライアントアプリケーションが https プロトコルおよび上記の Java ベースのプロパティーに対して設定された Tomcat プロセスですでに使用されている場合、Data Virtualization 固有の証明書キーをこれらの https 証明書キーストアにインポートすることはできません。

このシナリオでは、さまざまな Data Virtualization 固有の SSL プロパティーのセットをシステムプロパティーとして設定するか、「teiid-client-settings.properties」ファイル内で定義します。「teiid-client-settings.properties」のサンプルファイルは、「teiid-<version>-client.jar」ファイルのルートにある「teiid-client-settings.orig.properties」です。このファイルを展開し、選択した SSL モードに必要なプ

ロパティーの値を変更し、このファイルをクライアントアプリケーションのクラスパスに置き、"teiid-<version>-client.jar" ファイルの前にこのファイルをクライアントアプリケーションのクラスパスに配置します。

「teiid-client-settings.properties」ファイルに設定できる SSL プロパティーと定義を以下に示します。

```
########################################
# SSL Settings
########################################

#
# The key store type.  Defaults to JKS
#

org.teiid.ssl.keyStoreType=JKS

#
# The key store algorithm, defaults to
# the system property "ssl.TrustManagerFactory.algorithm"
#

#org.teiid.ssl.algorithm=

#
# The classpath or filesystem location of the
# key store.
#
# This property is required only if performing 2-way
# authentication that requires a specific private
# key.
#

#org.teiid.ssl.keyStore=

#
# The key store password (not required)
#

#org.teiid.ssl.keyStorePassword=

#
# The key alias(not required, if given named certificate is used)
#

#org.teiid.ssl.keyAlias=

#
# The key password(not required, used if the key password is different than the keystore password)
#

#org.teiid.ssl.keyPassword=

#
# The classpath or filesystem location of the
# trust store.
#
```

```
# This property is required if performing 1-way
# authentication that requires trust not provided
# by the system defaults.
#

#org.teiid.ssl.trustStore=

#
# The trust store password (not required)
#

#org.teiid.ssl.trustStorePassword=

#
# The cipher protocol, defaults to TLSv3
#

org.teiid.ssl.protocol=TLSv1

#
# Whether to allow anonymous SSL
# (the TLS_DH_anon_WITH_AES_128_CBC_SHA cipher suite)
# defaults to true
#

org.teiid.ssl.allowAnon=true

#
# Whether to allow trust all server certificates
# defaults to false
#

#org.teiid.ssl.trustAll=false

#
# Whether to check for expired server certificates (no affect in anonymous mode or with trustAll=true)
# defaults to false
#

#org.teiid.ssl.checkExpired=false
```

1 方向 SSL

```
org.teiid.ssl.trustStore=<dir>/server.truststore (required)
```

2 方向 SSL

```
org.teiid.ssl.keyStore=<dir>/client.keystore (required)
org.teiid.ssl.trustStore=<dir>/server.truststore (required)
```

2.2.3. ソケットの追加クライアント設定

「teiid-client-settings.properties」ファイルを使用して、Data Virtualization 低レベルおよび SSL ソケット接続プロパティーを設定できます。現在、ドライバー/クラスローダーの組み合わせごとに単一

のプロパティーファイルのみが必要です。「teiid-client-settings.properties」のサンプルファイルは、「teiid-<version>-client.jar」ファイルのルートにある「teiid-client-settings.orig.properties」です。設定をカスタマイズするには、このファイルを展開してコピーを作成し、適切なプロパティー値を変更し、このファイルを "teiid-<version>-client.jar" ファイルの前にクライアントアプリケーションのクラスパスに配置します。通常、クライアントは SSL 以外のプロパティーを調整する必要はありません。プロパティーの参照方法は次のとおりです。

```
########################################
# Misc Socket Configuration
########################################

#
# The time in milliseconds for socket timeouts.
# Timeouts during the initialization, handshake, or
# a server ping may be treated as an error.
#
# This is the lower bound for all other timeouts
# the JDBC login timeout.
#
# Typically this should be left at the default of 1000
# (1 second). Setting this value too low may cause read
# errors.
#

org.teiid.sockets.soTimeout=1000

#
# Set the max time to live (in milliseconds) for non-execution
# synchronous calls.
#

org.teiid.sockets.synchronousttl=240000

#
# Set the socket receive buffer size (in bytes)
# 0 indicates that the default socket setting will be used.
#

org.teiid.sockets.receiveBufferSize=0

#
# Set the socket send buffer size (in bytes)
# 0 indicates that the default socket setting will be used.
#

org.teiid.sockets.sendBufferSize=0

#
# Set to true to enable Nagle's algorithm to conserve bandwidth
# by minimizing the number of segments that are sent.
#

org.teiid.sockets.conserveBandwidth=false

#
# Maximum number of bytes per server message.
```

```
# May need to be increased when using custom types and/or large batch sizes.
#

org.teiid.sockets.maxObjectSize=33554432
```

注記

「teiid-client-settings.properties」に記載されているプロパティーはすべて、System または env プロパティーとして設定することもできます。

2.3. 準備済みステートメント

Data Virtualization は、**java.sql.PreparedStatement** の標準的な実装を提供します。PreparedStatements は、サーバーがステートメントの解析、解決、計画をスキップできるため、共通のステートメント実行を迅速化するために非常に重要になります。PreparedStatement の使用に関する詳細は、Java ドキュメントを参照してください。

PreparedStatement の考慮事項

- Data Virtualization はサーバー側でプランのキャッシュを実行するため、クライアント側の Data Virtualization **PreparedStatements** をプールする必要はありません。
- キャッシュされたプランの数は設定可能です（『管理ガイド』を参照）、最近使用したもの (LRU)でパージされます。
- キャッシュされたプランは、クラスターを介して分散されません。各クラスターメンバーに新規プランを作成する必要があります。
- プランは VDB 全体または特定のセッションにのみキャッシュされます。プランのスコープは、計画プロセスで評価された関数に基づいて自動的に検出されます。
- **CallableStatement** で実行されるストアドプロシージャーでは、**PreparedStatement** としてのみプランがキャッシュされます。
- 関数署名のバインド変数タイプ。たとえば、「where t.col = abs(?)」は、関数に署名が 1 つしかない場合や、戻り値の型を判断できる述語で関数が使用されるかどうかを判別できます。より複雑な状況では、キャストまたは変換で型ヒント（例： upper（convert(?, string)）を追加しなければならない場合があります。
- クエリーで同じ値のバインディングがある場合、その使用状況をいくつかの方法で統合できます。
    - クエリーは匿名プロシージャーブロックとして囲みます。

```
BEGIN
  DECLARE string PARAM1 = cast(? as string);
  SELECT ... WHERE COLUMN1 = $1 AND COLUMN2 = $1 ...;
```

bind 変数のキャストに注意してください。これは、変数宣言からタイプを推測していないリゾルバーに関連した小規模な問題です。

- $n 位置バインディングの機能のように PostgreSQL を使用することもできます。

```
SELECT ... WHERE COLUMN1 = $1 AND COLUMN2 = $1 ...
```

2.4. 結果の制限

Data Virtualization の結果セットには、以下の制限が適用されます。

- TYPE_SCROLL_SENSITIVE は互換性がありません。
- **UPDATABLE** ResultSets には互換性がありません。
- 手順実行から複数の ResultSet を返すことはできません。

2.5. JDBC エクステンション

これらは Data Virtualization からの JDBC API へのカスタムエクステンションであり、さまざまな機能と互換性を提供します。

2.5.1. ステートメントエクステンション

Data Virtualization ステートメント拡張インターフェース **org.teiid.jdbc.TeiidStatement** は、JDBC 標準以外の機能を提供します。エクステンションインターフェースを使用するには、単に Connection によって返されたステートメントをキャストまたはウェイトします。エクステンションインターフェースでは、以下のメソッドが提供されます。

表2.2 Connection Properties

メソッド名	説明
getAnnotations	ステートメントが SHOWPLAN ON/DEBUG で最後に実行された場合は、クエリーエンジンアノテーションを取得します。各 **org.teiid.client.plan.Annotation** には、クエリー計画の説明、カテゴリー、重大度、およびクエリープランナーによる選択を理解するのに使用できる注記の解決が含まれます。
getDebugLog	ステートメントが SHOWPLAN DEBUG で最後に実行された場合は、デバッグログを取得します。
getExecutionProperty	このステートメントオブジェクトの実行プロパティーの現在の値を取得します。
getPlanDescription	ステートメントが SHOWPLAN ON/DEBUG で最後に実行された場合は、クエリー計画の説明を取得します。プランは、**org.teiid.client.plan.PlanNode** オブジェクトで構成されるツリーです。通常、**PlanNode.toString（）** または **PlanNode.toXml（）** は、プランをテキスト形式に変換するために使用されます。
getRequestIdentifier	このステートメントで実行されている最後のコマンドの識別子を取得します。まだコマンドが実行されていない場合は、null を返します。

メソッド名	説明
setExecutionProperty	このステートメントに execution プロパティーを設定します。詳細は、「実行プロパティー」 の項を参照してください。通常、実行プロパティーが実行されるステートメントにのみ実行プロパティーを適用しない限り、SET ステートメントを使用することが推奨されます。
setPayload	翻訳者に渡すコマンドごとのペイロードを設定します。現在、組み込みの使用は、Oracle データソースのヒントを送信することです。

2.5.2. 部分的な結果モード

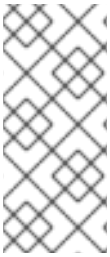
Data Virtualization サーバーで "partial results" クエリーモードを使用できます。このモードでは、一部のデータソースが利用できない場合でもサーバーが結果を返すようにクエリープロセッサーの動作が変更されます。

たとえば、異なるサプライヤーとデータオブジェクトに 2 つのデータソースが存在し、その 2 つのサプライヤーからの情報の間に統合を作成する仮想グループを作成しているとします。アプリケーションが部分的な結果クエリーモードを使用せずにクエリーを送信し、サプライヤーのデータベースのいずれかがダウンした場合に、仮想グループに対するクエリーは例外を返します。ただし、アプリケーションが "partial results" クエリーモードで同じクエリーを実行する場合、サーバーは実行中のデータソースからデータを返し、ダウンしているデータソースからデータを返しません。

「パート結果」モードを使用する場合、ソースが処理中に例外をスローすると、ユーザーのクエリーが失敗することはありません。代わりに、そのソースは、障害点の後に他の行が返さないものとして処理されます。通常、そのソースは 0 行を返します。

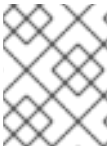
この動作は、これらの操作が不足している情報を便利な方法で処理するため、**UNION** または **OUTER JOIN** クエリーを使用する場合に最も役立ちます。他の種類のクエリーは、部分的な結果モードで使用した場合に、単に 0 行をユーザーに返し、ソースが利用できない場合にユーザーに返されます。

クエリーから除外されるソースごとに、ソースと失敗を記述する警告が生成されます。これらの警告は **Statement.getWarnings（）** メソッドから取得できます。このメソッドは **SQLWarning** オブジェクトを返しますが、「パート結果」の警告が発生した場合には、**org.teiid.jdbc.PartialResultsWarning** クラスのオブジェクトになります。このクラスは、名前で失敗したすべてのソースの一覧を取得し、各ソースから発生した特定の例外を取得するために使用できます。



注記

Data Virtualization は結果全体が形成される前にカーソルを有効にするため、最初の結果のバッチがクライアントに返されるまでデータソースの障害が決定されないことがあります。これは、Unions の場合に生じる可能性がありますが、結合はできません。すべての警告が累積されていることを確認するには、結果セット全体が読み込まれた後にステートメントを確認する必要があります。



注記

実行で他の警告が返されると、警告チェーンの最初の警告の後に部分的な結果の警告が発生することがあります。

部分的な結果モードはデフォルトでオフになっていますが、JDBC URL の DataSource または partialResultsMode=true の setPartialResultsMode("true")のいずれかを使用して Connection のすべてのクエリーに対してオンにすることができます。いずれの場合も、部分的な結果モードは、SET ステートメントで後で切り替えられます。

部分結果モードの設定

```
Statement statement = ...obtain statement from Connection...
statement.execute("set partialResultsMode true");
```

部分結果の警告の取得

```
statement.execute("set partialResultsMode true");
ResultSet results = statement.executeQuery("SELECT Name FROM Accounts");
while (results.next()) {
  ... //process the result set
}
SQLWarning warning = statement.getWarnings();
while(warning != null) {
  if (warning instanceof PartialResultsWarning) {
    PartialResultsWarning partialWarning = (PartialResultsWarning)warning;
    Collection failedConnectors = partialWarning.getFailedConnectors();
    Iterator iter = failedConnectors.iterator();
    while(iter.hasNext()) {
      String connectorName = (String) iter.next();
      SQLException connectorException = partialWarning.getConnectorException(connectorName);
      System.out.println(connectorName + ": " + connectorException.getMessage());
    }
  }
  warning = warning.getNextWarning();
}
```

警告

一部のインスタンスでは、通常 JDBC ソースが初期状態で利用できないと、Data Virtualization が適切なソース機能のセットを自動的に判断できなくなります。利用できないソースの機能が部分的な結果モードで有効ではないことを示す例外が表示される場合、トランスレーターでデータベースのバージョンまたは同様のプロパティーを手動で設定して、ソースが利用できない場合でも機能が認識されていることを確認する必要がある場合があります。

2.5.3. 非ブロッキングステートメント実行

JDBC クエリー実行は、ステートメントが実行されると、または resultset が反復されると、呼び出し

スレッドを無期限にブロックできます。場合によっては、呼び出しスレッドをこれらのブロック状態で保持したくないことがあります。埋め込み/ローカル接続を使用する場合は、オプションで **org.teiid.jdbc.TeiidStatement** および **org.teiid.jdbc.TeiidPreparedStatement** インターフェースを使用してクエリーを実行し、コールバック **org.teiid.jdbc.StatementCallback** でクエリーを実行できます。呼び出しスレッドは他の作業を自由に実行できます。コールバックは、必要に応じてエンジン処理スレッドにより実行されます。結果の処理自体がブロックされ、結果処理によってクエリー処理が同時に行われる場合、コールバックは onRow の処理をマルチスレッドに実装し、エンジンスレッドが続行されるようにします。

非ブロッキングの準備状態の実行

```
PreparedStatement stmt = c.prepareStatemen(sql);
Data VirtualizationPreparedStatement tStmt = stmt.unwrap(Data
VirtualizationPreparedStatement.class);
tStmt.submitExecute(new StatementCallback() {
    @Override
    public void onRow(Statement s, ResultSet rs) {
        //any logic that accesses the current row ...
        System.out.println(rs.getString(1));
    }

    @Override
    public void onException(Statement s, Exception e) throws Exception {
        s.close();
    }

    @Override
    public void onComplete(Statement s) throws Exception {
        s.close();
    }, new RequestOptions()
});
```

ブロック以外のロジックは、ステートメントの実行のみに限定されます。接続の作成やバッチ実行などの他の JDBC 操作にはノンブロッキングオプションはありません。

onRow メソッドの転送の位置にアクセスする場合（次の方法、isLast、isAfterLast、絶対）、まだ有効でない可能性があり、**org.teiid.jdbc.AsynchPositioningException** がスローされます。この例外は、取得した場合や、**Data VirtualizationResultSet.available（）** を呼び出して、必要な位置が有効かどうかを判断することで回復可能となります。

2.5.3.1. 継続的な実行

RequestOptions オブジェクトは、継続または **setContinuous** メソッドで連続した非同期h実行の特別なタイプを指定するために使用できます。**連続モードでは、ステートメントが継続的に再実行されます。これは、SQL プランを介して処理されるリアルタイムまたはその他のデータストリームを消費することを目的としています。継続的なクエリーは、エラーまたはステートメントが明示的に閉じられている場合にのみ終了します。継続的なクエリーの SQL は、他のステートメントとは異なります。マテリアル化されたビューやセッションスコープの一時テーブルなど、継続的なソースから取得が再利用するために適切にキャッシュされるように注意する必要があります。**

継続的なクエリーは、以下を行う必要があります。

- 結果セットを返します。
- 前方のみの結果セットで実行される
- トランザクションのスコープで使用できない

リソースの消費は継続的プランで異なるため、サーバーの最大アクティブなプランの制限にはカウントされません。通常、カスタムソースはデータストリームの提供に使用されます。詳細は、『ReusableExecutions』のセクションの「Developer's Guide」を参照してください。

クライアントが継続的なクエリーを終了する場合は、Statement.close（） メソッドまたは Statement.cancel（） メソッドを呼び出す必要があります。 通常、コールバックは結果を処理する必要がなくなったときに常に閉じます。

継続的な処理に関連する追加メソッドについては、StatementCallback として使用する場合は、「ContinuousStatementCallback」も参照してください。

2.5.4. 結果の拡張

Data Virtualization の結果セット拡張インターフェース org.teiid.jdbc.TeiidResultSet は、JDBC 標準以外の機能を提供します。拡張インターフェースを使用するには、Data Virtualization ステートメントによって返された結果セットをキャストまたはアンウェイトします。エクステンションインターフェースでは、以下のメソッドが提供されます。

表2.3 Connection Properties

メソッド名	説明
利用可能	ブロックまたは ResultSet に到達せずに読み取り（現在の後に）読み取りできる行の最小数を返します。

2.5.5. コネクションエクステンション

データ仮想化接続（ org.teiid.jdbc.TeiidConnection インターフェースによって定義）は、指定の接続を再認証するために changeUser メソッドと互換性があります。再認証に成功すると、現在の接続が指定のアイデンティティーで使用されます。既存のステートメント/結果セットは引き続き古いアイデンティティーで使用できます。

2.6. 互換性のない JDBC メソッド

JDK 1.6 の JDBC に基づくこの付録では、Data Virtualization と互換性がない JDBC メソッドのみを説明します。以下に指定しない限り、Data Virtualization は他のすべての JDBC メソッドと互換性があります。

コメントなしでリストされているこれらのメソッドは、サポートされないことを SQLException スローします。

指定された場合、一部のメソッドは例外をスローしませんが、予期しない動作を示す場合があります。引数を指定しないと、関連するすべての（上書きされた）メソッドに互換性がありません。引数がリストされている場合は、指定されたメソッドの形式のみが互換性されません。

2.6.1. 「java.sql」内の互換性のないクラスとメソッド

クラス名	メソッド
Blob	[source,java] ---- getBinaryStream(long, long)- throws SQLFeatureNotSupportedException setBinaryStream(long)- - throws SQLFeatureNotSupportedException setBytes - - throws SQLFeatureNotSupportedException truncate(long)- throws SQLFeatureNotSupportedException ----
CallableStatement	[source,java] ---- getObject(int parameterIndex, Map<String, Class<?>> map)- throws SQLFeatureNotSupportedException getRef - throws SQLFeatureNotSupportedException getRowId - throws SQLFeatureNotSupportedException getURL(String parameterName)- throws SQLFeatureNotSupportedException registerOutParameter - ignore registerOutParameter（String parameterName, *）- throws SQLFeatureNotSupportedException setRowId(String parameterName, RowId x)- throws SQLFeatureNotSupportedException setURL(String parameterName, URL val)- throws SQLFeatureNotSupportedException ----

クラス名	メソッド
Clob	[source,java] ---- getCharacterStream(long arg0, long arg1)- throws SQLFeatureNotSupportedException setAsciiStream(long arg0)- throws SQLFeatureNotSupportedException setCharacterStream(long arg0)- throws SQLFeatureNotSupportedException setString - throws SQLFeatureNotSupportedException truncate - throws SQLFeatureNotSupportedException ----
接続	[source,java] ---- createBlob - throws SQLFeatureNotSupportedException createClob - throws SQLFeatureNotSupportedException createNClob - throws SQLFeatureNotSupportedException createSQLXML - throws SQLFeatureNotSupportedException createStruct(String typeName, Object[] attributes)- throws SQLFeatureNotSupportedException getClientInfo - throws SQLFeatureNotSupportedException createClob - throws SQLFeatureNot SupportedException releaseSavepoint - throws SQLFeatureNotSupportedException rollback(Savepoint savepoint)- throws SQLFeatureNotSupportedException setHoldability - throws SQLFeatureNotSupportedException setSavepoint - throws SQLFeatureNotSupportedException setTypeMap - throws SQLFeatureNotSupportedException setRealOnly - effectively ignored ----
DatabaseMetaData	[source,java] ---- getAttributes - throws SQLFeatureNotSupportedException getClientInfoProperties - throws SQLFeatureNotSupportedException getRowIdLifetime - throws SQLFeatureNotSupportedException ----
NClob	**サポート対象外**
PreparedStatement	[source,java] ---- setRef - throws SQLFeatureNotSupportedException setRowId - throws SQLFeatureNotSupportedException setUnicodeStream - throws SQLFeatureNotSupportedException ----
Ref	**実装されていない**

クラス名	メソッド
ResultSet	[source,java] ---- deleteRow - throws SQLFeatureNotSupportedException getHoldability - throws SQLFeatureNotSupportedException getObject(, **Map<String, Class<?>> map)- throws SQLFeatureNotSupportedException getRef - throws SQLFeatureNotSupportedException getRowId - throws SQLFeatureNotSupportedException getUnicodeStream - throws SQLFeatureNotSupportedException getHoldability - throws SQLFeatureNot SupportedException getURL - throws SQLFeatureNotSupportedException insertRow - throws SQLFeatureNotSupportedException moveToInsertRow - throws SQLFeatureNotSupportedException refreshRow - throws SQLFeatureNotSupportedException rowDeleted - throws SQLFeatureNotSupportedException rowInserted - throws SQLFeatureNotSupportedException rowUpdated - throws SQLFeatureNotSupportedException setFetchDirection - throws SQLFeatureNotSupportedException rowInsertion - throws SQL FeatureNotSupportedException update** - throws SQLFeatureNotSupportedException ----
RowId	**サポート対象外**
Savepoint	**サポート対象外**
SQLData	**サポート対象外**
SQLInput	**サポート対象外**
SQLOutput	**サポート対象外**

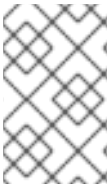
2.6.2. "javax.sql" の互換性のないクラスとメソッド

クラス名	メソッド
RowSet*	**サポート対象外**

第3章 ODBC 互換性

Open Database Connectivity(ODBC)は、1992 の SQL Access グループにより開発された標準のデータベースアクセス方法です。ODBC は Java の JDBC と同様に、データを処理するデータベース管理システム(DBMS)に関係なく一貫性のあるクライアントアクセスを可能にします。ODBC はドライバーを使用してアプリケーションのデータクエリーを DBMS が認識するコマンドに変換します。これを機能させるには、アプリケーションと DBMS の両方が ODBC に準拠する必要があります。つまり、アプリケーションは ODBC コマンドを実行し、DBMS がそれらを応答できる必要があります。

Data Virtualization は、PostgreSQLの ODBC ドライバーを介して、Data Virtualization ランタイムにデプロイされた VDB への ODBC アクセスを提供できます。これは、Data Virtualization にはソケットクライアント経由でアクセスできる PostgreSQL サーバーエミュレーション層があるためです。



注記

デフォルトでは、ODBC はポート 35432 で有効にされ、実行されます。

pg エミュレーションは完了していません。ODBC アクセスの目的は、pgsql クエリーではなく、Data Virtualization クエリーを発行するための JDBC 以外の接続を提供することです。多くの PostgreSQL コンストラクトを使用できますが、クエリーのデフォルトの動作は Data Virtualization の期待値に一致します。pgsql処理をさらにエミュレートするオプションのプロパティーについては、「システム プロパティー」を参照してください。

注記

ODBC でアンダースコア("") を使用した名前の処理。デフォルトでは、Data Virtualization にはエスケープ文字のようなデフォルトがありません。ODBC クライアントによっては、バックスラッシュがデフォルトで使用されていることが予想されます。これは PostgreSQL の動作です。これにより、名前に "" が指定されたオブジェクトに対してメタデータクエリーが発行され、何も返されない、または正しくない結果が返される可能性があります。org.teiid.backslashDefaultMatchEscape システムプロパティーを true に設定すると、PostgreSQL の動作をグローバルにエミュレートできます。現行セッションでプロパティーを変更するには、ODBC クライアントの問題の cast（teiid_session_set('backslashDefaultMatchEscape', true)をブール値として選択してから、他のステートメントの前に cast（teiid_session_set('backslashDefaultMatchEscape', true) ステートメントを選択します。

Postgres ODBC ドライバー 9.5 以降では、クライアントが E escaped リテラルを使用するため、この特別なプロパティーは必要ありません。

互換性は最後に 9.6 Postgres ODBC ドライバーで確認されました。必要に応じて、後続のクライアントバージョンを使用し、コミュニティーに問題を報告することが推奨されます。

3.1. 既知の制限事項

- 更新可能なカーソルはサポートされていません。この機能が無効でない場合は、pg system 列の ctid を含む解析エラーが発生します。
- LO サポートは利用できません。LOB は transport max lob size 設定を使用して、文字列または bytea として返されます。
- Data Virtualization オブジェクトタイプは PostgreSQL UNKNOWN タイプにマップされ、ODBC レイヤーでシリアライズできません。必要に応じて、teiid_session_set がオブジェクト値を返す場合に、キャスト/Convert を使用して型ヒントを提供する必要があります。"SELECT teiid_session_set('x', 'y')" は失敗しますが、"SELECT cast（teiid_session_set('x', 'y')" が成功します。
- マルチディメンションアレイはサポートされていません。

3.2. インストールシステム

アプリケーションが ODBC を使用できるようにするには、まずアプリケーションが実行中のマシンと同じマシンに ODBC ドライバーをインストールし、Data Virtualization VDB の接続プロファイルを表す Data Source Name(DSN)を作成する必要があります。

3.3. 設定

警告

デフォルトでは、クライアントは pg/ODBC インターフェースの Data Virtualization でプレーンテキストのパスワード認証を使用します。クライアント/サーバーが SSL または GSS 認証を使用するように設定されていない場合、パスワードはプレーンテキストでネットワーク上で送信されます。

ウィンドウクライアントについては、「データ ソース名の設定」を 参照してください。

DSN Less 接続 も参照してください。

3.3.1. 接続設定

使用できる説明とともに、利用可能な pg ドライバー接続オプションはすべて https://odbc.postgresql.org/docs/config.html で定義されます。コネクション文字列でこれらのプロパティーを使用する場合、プロパティー名は https://odbc.postgresql.org/docs/config-opt.html で定義されます。

ただし、Data Virtualization はすべてのプロパティーを適用しないため、Updatable Cursors などの一部の場合はクエリーに失敗します。

表3.1 データ仮想化の主な ODBC 設定

名前	説明
Updateable Cursors & Row Versioning	使用しないでください。
サーバー側の準備と固定ステートメントの使用とデッドプレマチャーの使用	「Use serverside prepare」を有効にし、「Parse Statements」/"Disallow Premature" を無効にすることが推奨されます。
SSL モード	セキュリティー保護された pg トランスポートポートに接続する場合は、必要になる場合があります。
カーソルの宣言/取得と最大数の使用	大規模な結果セットが使用される場合に、リソースの管理を改善するために使用する必要があります。

ロギング/デバッグ設定は必要に応じて使用できます。

「Show SystemTables」、「True is -1」、「Backend genetic optimizer」、「Bytea as LongVarBinary」、「Bools as Char」などのデータタイプ、メタデータ、または最適化を操作する設定は、Data Virtualization サーバーで無視され、クライアント側の効果はありません。これらまたはその他の設定を定義された設定が必要な場合は、製品/プロジェクトに関する問題を作成してください。

クライアント側の影響を与えるその他の設定（「LF <→ CR/LF 変換」など）は必要な場合に使用できますが、現時点では設定のサーバー側の使用はありません。

3.3.1.1. Data Virtualization 接続の設定

ほとんどの Data Virtualization 固有の接続プロパティーは、ODBC クライアント接続設定にマップされません。この状況で、ポスト接続 SET ステートメントを使用できない場合は、VDB 自体がデフォルトの 接続プロパティー ODBC を取得する可能性があります。form connection.XXX の VDB プロパティーを使用して、部分的な結果モードや結果セットキャッシュなどを制御します。

アプリケーション名は、一部のクライアントで設定できます。設定されていない場合は、SET ステートメント - "SET application_name name" - を使用して、接続が確立された後でも名前を設定できます。

3.4. DATA SOURCE NAME(DSN)の設定

利用可能なクライアント設定 オプションの説明は、「Data Virtualization 互換 オプション」を参照してください。

3.4.1. Windows Installation

ワークステーションに ODBC Driver Client ソフトウェアをインストールしたら、Data Virtualization Runtime に接続するように設定する必要があります。以下の手順は、Microsoft Windows Platform に固有のものです。

これを行うには、管理者権限でワークステーションにログインしている必要があり、Control Panel の Data Sources(ODBC) アプレットを使用して新しいデータソース名を追加する必要があります。

設定する各データソース名は、Data Virtualization System 内で 1 つの VDB のみにアクセスできます。複数の VDB を利用できるようにするには、複数のデータソース名を設定する必要があります。

データソース名(DSN)を作成するには、以下の手順を実行します。

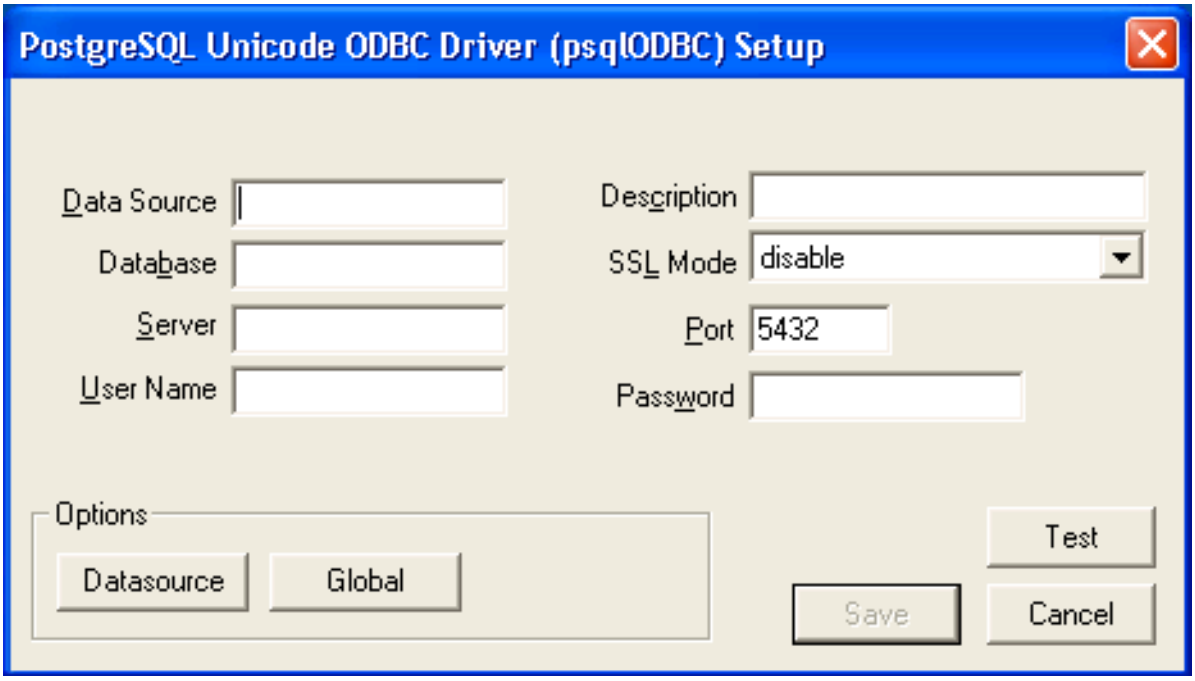
1. Start メニューから Settings > Control Panel を選択します。
2. コントロールパネルが表示されます。管理ツール をダブルクリックします。
3. 次に、Data Sources(ODBC) をダブルクリックします。

4. **ODBC Data Source Administrator** アプレットが表示されます。追加する **DSN** の種別に関連付けられたタブをクリックします。

5. **Create New Data Source** ダイアログボックスが表示されます。データソーステーブルを設定するドライバーの選択 で、**PostgreSQL Unicode** を選択します。

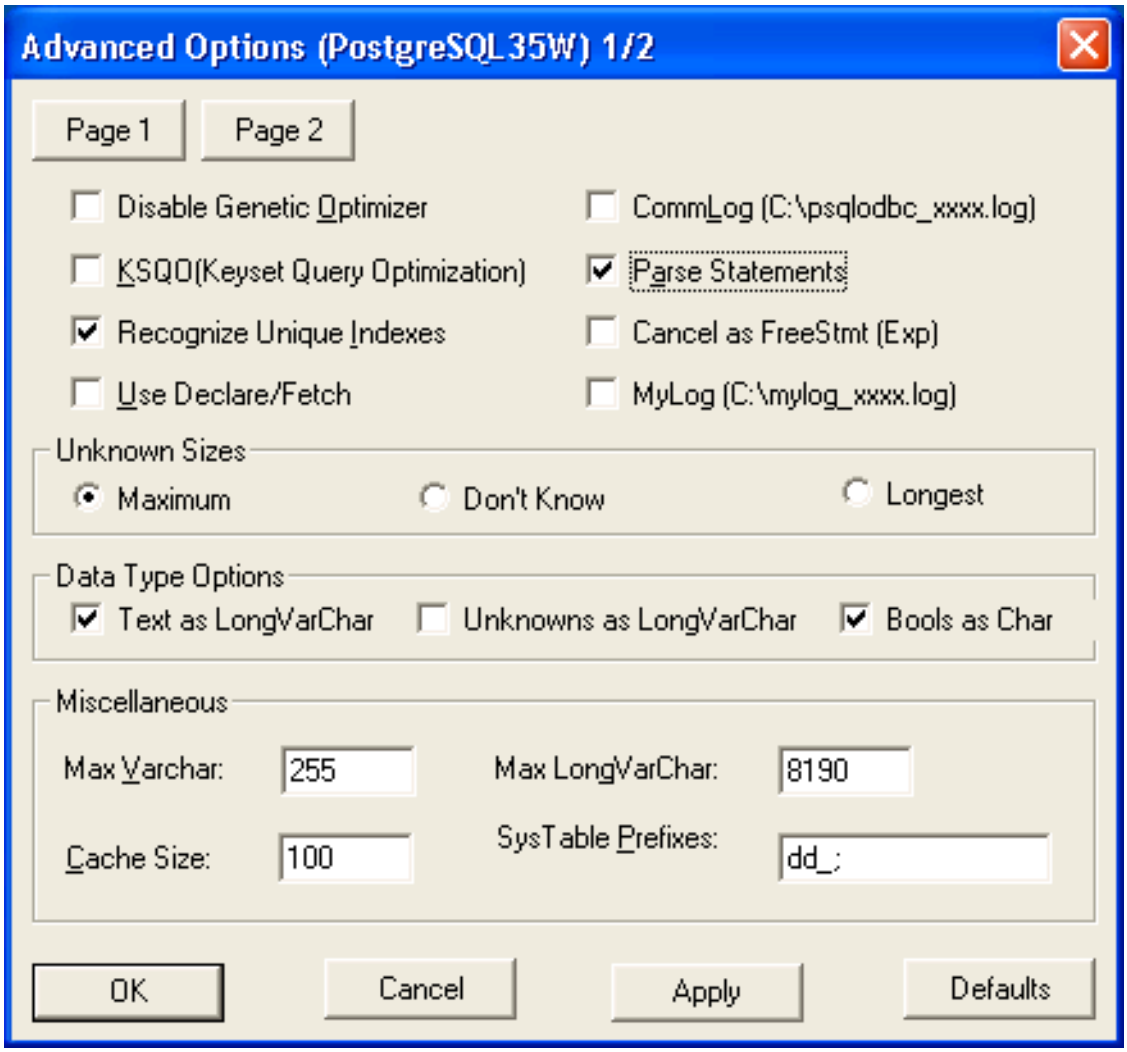
6. **Finish** をクリックします。

7. **PostgreSQL ODBC DSN Setup** ダイアログボックスが表示されます。

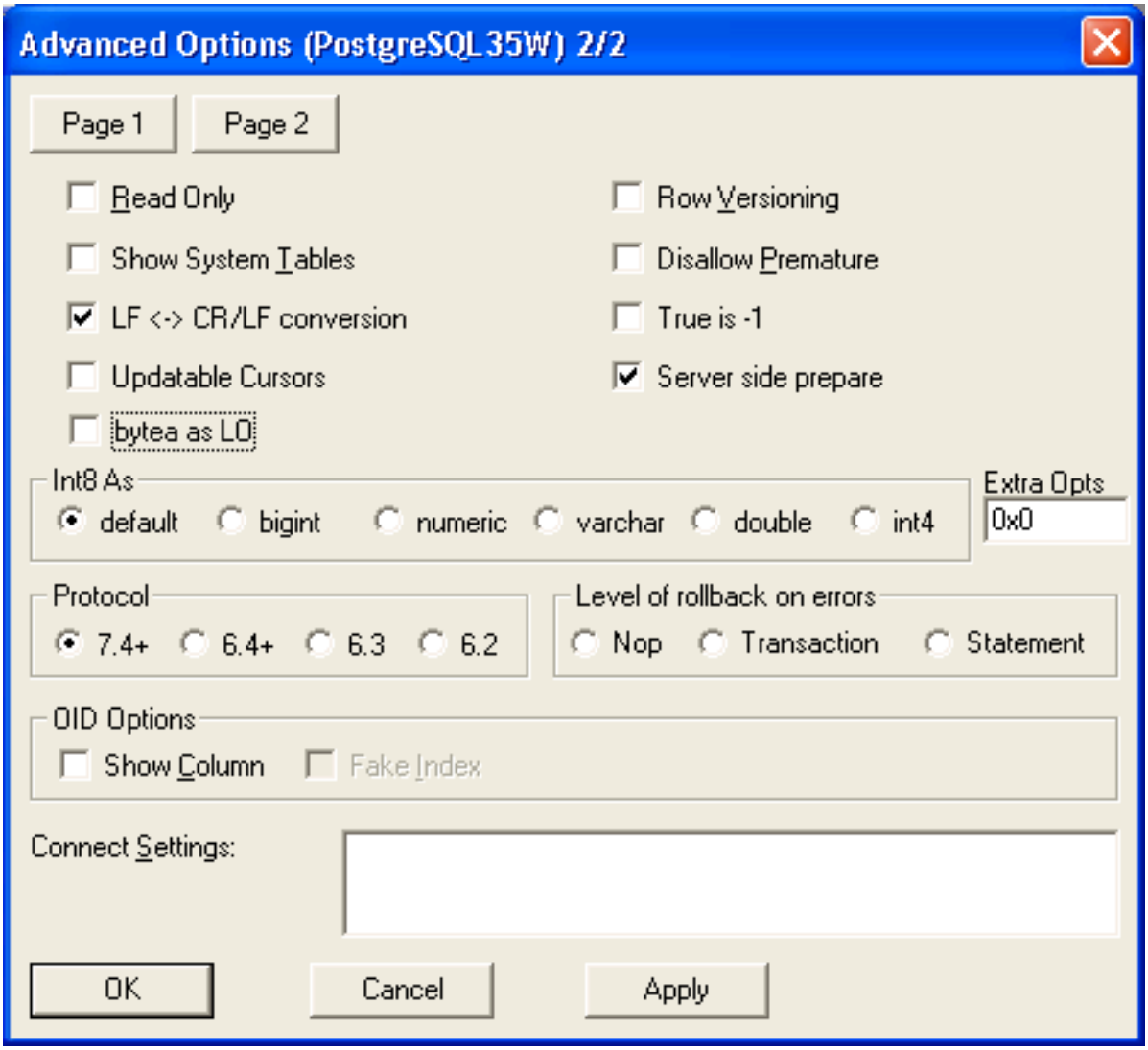


Data Source Name の編集ボックスに、このデータソースに割り当てる名前を入力します。**Database edit** ボックスに、このデータソースでアクセスする仮想データベースの名前を入力します。**Server edit** ボックスに、**Data Virtualization** ランタイムのホスト名または **IP** アドレスを入力します。ファイアウォールまたは **NAT** アドレス経由で接続する場合には、ファイアウォールアドレスまたは **NAT** アドレスを入力する必要があります。**Port edit** ボックスに、**Data Virtualization System** が **ODBC** 要求をリッスンするポート番号を入力します。デフォルトでは、**Data Virtualization** はポート **35432** で **ODBC** 要求をリッスンし、**User Name** および **Password edit** ボックスで、**Data Virtualization** ランタイムアクセスのユーザー名およびパスワードを指定します。データソースに関する説明を **Description** フィールドに入力します。

8. データソース ボタンをクリックすると、以下の図が表示されます。以下のようにオプションを設定します。



「page2"」をクリックし、表示されるようにオプションが選択されていることを確認します。



9. 「保存」をクリックすると、Data Virtualization の実行中に接続を検証するためにオプションで「テスト」をクリックできます。Data Virtualization の仮想データベースを、ODBC アプリケーションのデータソースとして設定している。Excel などのアプリケーションを使用して、VDB のデータをクエリーできるようになりました。

3.4.2. その他の *nix プラットフォームのインストール

*nix プラットフォームで ODBC を使用して Data Virtualization にアクセスする前に、ODBC ドライバーマネージャーをインストールするか、またはすでに存在することを確認する必要があります。ODBC ドライバーマネージャー Data Virtualization は unixODBC を推奨します。RedHat Linux または Fedora を使用している場合は、グラフィカルの「yum」インストーラーを検索して unixODBC を検索し、インストールできます。それ以外の場合は、unixODBC マネージャーを ダウンロード できます。インストールするには、ファイルの内容を一時的な場所に展開し、スーパーユーザーで次のコマンドを実行します。

```
./configure
make
make install
```

インストール時に問題が発生した場合は、unixODBC の Web サイトサイトで詳細を確認してください。

これで、前の手順で PostgreSQL ドライバーが正しくインストールされたことを確認するには、以下のコマンドを実行します。

```
odbcinst -q -d
```

これにより、システムにインストールされているすべての ODBC ドライバーが表示されます。次に、DSN を作成します。「/etc/odbc.ini」ファイルを編集し、以下を追加します。

```
[<DSN name>]
Driver = /usr/lib/psqlodbc.so
Description = PostgreSQL Data Source
Servername = <Data Virtualization Host name or ip>
Port = 35432
Protocol = 7.4-1
UserName = <user-name>
Password = <password>
Database = <vdb-name>
ReadOnly = no
ServerType = Postgres
ConnSettings =
UseServerSidePrepare=1
Debug=0
Fetch = 10000
# enable below when dealing large resultsets to enable cursoring
#UseDeclareFetch=1
```

"/etc/odbc.ini ファイルを編集するには、「sudo」パーミッションが必要な点に注意してください。DSN の定義に使用できる設定可能なすべてのオプションは、「postgreSQL ODBC」ページを参照してください。

DSN の定義が完了したら、以下のコマンドを実行して DSN を確認できます。

```
isql <DSN-name> [<user-name> <password>] < commands.sql
```

「commands.sql」ファイルには、実行する SQL コマンドが含まれています。commands.sql ファイルを省略すると、インタラクティブシェルで提示されます。

ヒント

Perl、Python、C/C++ などの言語を Postgres に使用することもできます。または、それらを使用して直接 Postgres 接続モジュールがある場合は、それらを使用してデータ仮想化に接続し、結果のクエリーを発行することができます。

3.5. DSN LESS 接続

DSN を明示的に作成することで、ODBC を使用して Data Virtualization VDB に接続することもできます。ただし、このようなシナリオでは、「DSN less connection string」と呼ばれるアプリケーションが必要です。以下は、接続文字列のサンプルです。

Windows の場合:

```
ODBC;DRIVER={PostgreSQL Unicode};DATABASE=<vdb-name>;SERVER=<host-name>;PORT=
<port>;Uid=<username>;Pwd=<password>;c4=0;c8=1;
```

*nix の場合：

```
ODBC;DRIVER={PostgreSQL};DATABASE=<vdb-name>;SERVER=<host-name>;PORT=
<port>;Uid=<username>;Pwd=<password>;c4=0;c8=1;
```

利用可能な Data Virtualization 接続オプション を参照してください。

3.6. ODBC を使用した接続プロパティーの設定

ODBC 接続を使用する場合は、以下のようなコマンドを実行して、Data Virtualization で利用可能な接続プロパティーの Connection#URL 接続プロパティーを設定できます。

```
SET <property-name> TO <property-value>
```

たとえば、結果セットのキャッシュを有効にするには、以下を実行します。

```
SET resultSetCacheMode TO 'true'
```

別のオプションとして、vdb ファイルの VDB プロパティーとしてこれを設定することもできます。

```
CREATE DATABASE vdb OPTIONS ("connection.partialResultsMode" true);
```

第4章 ODATA の互換性

4.1. ODATA とは

Open Data Protocol(OData)は、データのクエリーおよび更新を行う Web プロトコルで、データのロックを解除し、現在のアプリケーションに存在する silo から解放する方法を提供します。OData は、さまざまなアプリケーション、サービス、ストアから情報にアクセスするために、HTTP、Atom Publishing Protocol(AtomPub)、JSON などの Web テクノロジーに基づいて適用および構築することでこれを行います。このプロトコルは、過去数年のさまざまな製品で AtomPub クライアントおよびサーバーを実装する経験から発生しました。OData は、リレーショナルデータベース、ファイルシステム、コンテンツ管理システム、従来の Web サイトなど、さまざまなソースから情報を公開およびアクセスするために使用されます。

OData は Web の仕組みと一致しています。リソース識別やコミットのための URI に対して、これらのリソースと対話するための HTTP ベースの統一インターフェース（Web と同様）に比べます。このコア Web 原則により、OData はさまざまなクライアント、サーバー、サービス、ツール全体で新たなレベルのデータ統合および相互運用性を有効にすることが可能となります。

copied from http://odata.org

4.2. ODATA の DATA VIRTUALIZATION の互換性

Data Virtualization は OData Version 4.0 と互換性があります。

4.3. ODATA バージョン 4.0 の互換性

Data Virtualization は OData 仕様に準拠するように努めます。本章の残りの部分では、OData および Data Virtualization の互換性に関する特定の内容について説明しますが、仕様 も併せてご参照する必要があります。

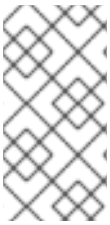
4.3.1. データへのアクセス方法

たとえば、NW モデルに 顧客 テーブルがある northwind deployed という名前の vdb がある場合は、URL 経由で HTTP GET でそのテーブルにアクセスできます。

```
http://localhost:8080/odata/customers
```

これは、JDBC/ODBC 接続を作成し、SQL を発行するためのフォークになります。

```
SELECT * FROM NW.customers
```



注記

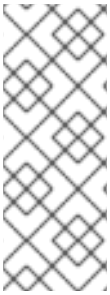
リソースパスで正しいケース（上下）を使用します。SQL とは異なり、URI で使用される名前は大文字と小文字を区別します。

OData クエリーから返される結果は Atom/AtomPub XML または JSON 形式になります。JSON 結果はデフォルトで返されます。

4.3.2. クエリーの基本

ユーザーはクエリーと共に述語を送信でき、結果をフィルターできます。

```
http://localhost:8080/odata/customers?$filter=name eq 'bob'
```



注記

「eq」の関連のスペースは、例を読みやすくするためにのみ使用されます。実際の URL では、パーセントでエンコードする必要があります。URL のすべてのスペースに対する OData のパーセントエンコーディング。http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html

これは、JDBC/ODBC 接続を作成し、SQL を発行するのと似ています。

```
SELECT * FROM NW.customers where name = 'bob'
```

特定の形式で結果をフォーマットするよう要求するには、クエリーオプション $format を追加します。

```
http://localhost:8080/odata/customers?$format=JSON
```

必要に応じて、クエリーオプションを組み合わせることができます。フィルターを使用した例を以下に示します。

```
http://localhost:8080/odata/customers?$filter=name eq 'bob'&$format=xml
```

OData を使用すると、あるエンティティーから別のエンティティーへナビゲーションをクエリーできます。ナビゲーションは、リレーショナルデータベースの外部キー関係に似ています。

たとえば、customer テーブルが customer_fk と呼ばれる顧客のプライマリーキーの orders テーブルにキーをエクスポートしている場合、OData GET は以下のように発行できます。

```
http://localhost:8080/odata/customers(1234)/customer_fk?$filter=orderdate gt datetime'2012-12-31T21:23:38Z'
```

これは、JDBC/ODBC 接続を作成し、SQL を発行するためのフォークになります。

```
SELECT o.* FROM NW.orders o join NW.customers c on o.customer_id = c.id where c.id=1234 and o.orderdate > {ts '2012-12-31 21:23:38'}
```

注記

ODATA に関するより具体的 なドキュメント - 詳細なプロトコルアクセスについては、http://odata.org で仕様を読み取ることができます。OData サーバーにアクセス する例 は、非常に便利な web リソースを読み取ることもできます。

4.3.2.1. ストアドプロシージャーの実行方法

OData を使用すると、odata を使用して公開されたストアドプロシージャーメソッドを呼び出すことができます。

```
http://localhost:8080/odata/getcustomersearch(id=120,firstname='micheal')
```

4.3.2.2. すべての行が表示されませんか？

詳細は、以下の 設定セクション を参照してください。通常、バッチ処理は、ツールが自動的に理解する必要があるものであり、$skiptoken クエリーオプションが指定された追加のクエリーが必要になります。

```
http://localhost:8080/odata/customers?$skiptoken=xxx
```

4.3.2.3. 「EntitySet Not Found」エラー？

上記のクエリーを発行すると、以下のようなメッセージが表示されますか。

```
{"error":{"code":null,"message":"Cannot find EntitySet, Singleton, ActionImport or
FunctionImport with name 'xxx'."}}
```

次に、model-name/table-name の組み合わせが誤って指定された場合は、スペルとケースを確認してください。

テーブルに PRIMARY KEY または UNIQUE KEY(s)が含まれていない場合など、エンティティーがメタデータ の一部ではない場合があります。

4.3.3. データの更新方法

OData プロトコルを使用すると、上記の READ 操作と共に CREATE/UPDATE/DELETE 操作を実行できます。これらの操作は、さまざまな HTTP メソッドを使用します。

INSERT/CREATE は、HTTP メソッド「POST」を使用して実行されます。

POST の例

```
POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/json
Accept: application/json
{
  "CustomerID": "AS123X",
  "CompanyName": "Contoso Widgets",
  "Address" : {
    "Street": "58 Contoso St",
    "City": "Seattle"
  }
}
```

アップデートは、HTTP の「PUT」で実行されます。

顧客の PUT 更新の例

```
PUT /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/josn
Accept: application/json
{
  "CustomerID": "AS123X",
  "CompanyName": "Updated Company Name",
  "Address" : {
    "Street": "Updated Street"
  }
}
```

DELETE 操作は HTTP の「DELETE」メソッドを使用します。

削除の例

```
DELETE /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/json
Accept: application/json
```

4.3.4. 設定

"spring.teiid.odata." で始まるプロパティーにより OData インターフェースをカスタマイズできます。

|batch-size |Number of rows to send back each time, -1 returns all rows |256

|skiptoken-cache-time |Time interval between the results recycled/expired between $skiptoken requests |300000

Data Virtualization OData サーバーは、結果行が設定されたバッチサイズを超えたときにカーソルロジックを実装します。リクエストごとに batch-size の数が返されます。このような各要求は、skip-token-cache-time で指定されたアイドル時間を指定してアクティブなカーソルと見なされます。カーソルがタイムアウトすると、カーソルが閉じられ、残りの結果がクリーンアップされ、追加のクエリー

に利用できなくなります。これらのカーソルのセッションベースの追跡はないため、期限切れの時間後に skiptoken の要求が再度実行され、カーソルを相対位置に再構成しようとしますが、結果としてベースとなるソースと新しい情報が更新される可能性は保証されません。

4.3.5. 制限事項

OData4 インターフェースは、一部の機能制限の対象となります。以下の機能は使用できません。

- 検索。
- デルタ処理。
- OData 仕様の Data-aggregation エクステンション。
- $it usage はプリミティブコレクションプロパティーのみに制限されます。

4.3.6. アクセスするクライアントツール

OData のアクセスは、プログラミングモデルに応じてユーザーの出所や、アクセスレイヤーを OData に書き込むさまざまな方法が必要です。いくつかの提案を以下に示します。

- ブラウザー： OData Explorer は OData データサービスを参照するためのオンラインツールです。
- Olingo: OData V4 をサポートする Java フレームワークには、コンシューマーとプロデューサーフレームワークの両方があります。
- Microsoft にはさまざまな .Net ベースのライブラリーがあります。http://odata.github.io/を参照してください。
- Windows Desktop: LINQPad は、OData クエリーを対話的にビルドするための便利なツールです。See https://www.linqpad.net/

- シェルスクリプト： CURL ツールの使用

利用可能なその他のフレームワークおよびツールの最新情報は、http://www.odata.org/ecosystem/を参照してください。

4.3.7. OData Metadata（Data Virtualization がリレーショナルデータベースを OData の $metadata に変換する方法）

OData は Conceptual Schema Definition Language(CSDL)を使用してスキーマを定義します。Data Virtualization の ACTIVE 状態の VDB は、CSDL 形式で表示されるメタデータを公開します。たとえば、vdb のメタデータを取得する場合は、以下のような要求を発行する必要があります。

```
http://localhost:8080/odata/$metadata
```

OData スキーマモデルはリレーショナルデータベースモデルではないため、Data Virtualization は以下のセマンティクスを使用して、リレーショナルデータベースモデルを OData スキーマモデルにマッピングします。

リレーショナルデータベース	マッピングされた OData エンティティー
モデル名	スキーマ名前空間、EntityContainer Name
テーブル/ビュー	EntityType、EntitySet
テーブルカラム	EntityType のプロパティー
プライマリーキー	EntityType のキープロパティー
外部キー	EntityType のナビゲーションプロパティー
手順	FunctionImport、actionImport
テーブルのテーブルを返す手順	ComplexType

設計による Data Virtualization は、EntityType の "embedded" ComplexType を定義しません。

OData のアクセスはキーベースであるため、すべてのテーブル Data Virtualization が OData 経由で公開する MANDATORY であるため、PK または少なくとも 1 つの UNIQUE キーが必要です。上記のいずれかではない表は、$metadata からドロップされます。

すべてのデータロールは OData メタデータの構築で参照されないため、テーブルや手順を明示的に非表示にする必要がある場合があります。これは、オブジェクトの拡張メタデータプロパティー「teiid_odata:visible」で行うことができます。

```
create foreign table HIDDEN (id long primary key, ...) OPTIONS ("teiid_odata:visible" false);
```

teiid_odata:visible を false に設定すると、OData レイヤーは指定されたオブジェクトを公開しません。

データタイプマッピング

Data Virtualization タイプ	OData Type
STRING	Edm.String
BOOLEAN	Edm.Boolean
BYTE	Edm.SByte
SHORT	Edm.Int16
INTEGER	Edm.Int32
LONG	Edm.Int64
FLOAT	Edm.Single
DOUBLE	Edm.Double
BIG_INTEGER	Edm.Decimal
BIG_DECIMAL	Edm.Decimal
DATE	Edm.Date
TIME	Edm.TimeOfDay
TIMESTAMP	Edm.DateTimeOffset
BLOB	Edm.Stream
CLOB	Edm.Stream

XML	Edm.Stream
VARBINARY	Edm.Binary

geography および Geometry は、関連する {http://www.teiid.org/translator/spatial/2015}type プロパティーに基づいて、対応する Edm.GeometryXXX および Edm.GeographyXXX タイプにマッピングされます。Edm.Geometry または EdmGeography への一般的なマッピングは、値を正しくシリアライズできません。

可能な場合、配列タイプはコレクションタイプにマップされます。ただし、多次元のアレイを含めることはできません。また、配列/コレクションの値はパラメーターまたは比較として使用できません。

4.3.7.1. 関数およびアクション

エンティティーとそのプロパティーのマッピングは比較的簡単です。Data Virtualization の手順の OData Functions および Actions へのマッピングがより関与します。DDL（Java クラスではない）で定義された仮想手順、ソース手順、および仮想機能はすべてマッピングの対象となります。Java クラスによって定義されるソース機能または Virtual Function は、現在対応する OData コンストラクトにマッピングされません。その機能が必要な場合は、問題をログに記録してください。OData には out パラメーターの概念がないため、OUT パラメーターは無視され、INOUT パラメーターは IN としてのみ処理されます。結果セットは、コンプレックスタイプのコレクション結果にマッピングされます。配列の結果は単純な型のコレクションにマップされます。

以下の場合、OData Function が使用されます。

- *手順/機能には戻り値（スカラーまたは結果セットのいずれか）があります。*
- *手順/関数には LOB 入力パラメーターがありません。現在、Clob、Blob、Meometry、Geography、および JSON は LOB タイプとみなされます。*
- *手順/機能は無料です。これは、更新数に対して 0 の明示的な値で決定されます。例：CREATE VIRTUAL PROCEDURE ... OPTIONS(UPDATECOUNT 0)AS BEGIN ...*

これらの条件のいずれかが満たされない場合は、代わりに OData Action によって手順/機能が表示されます。ただし、LOB 値を持つ結果セットがある場合は、複数のストリーミング値を返すことができないため、この手順はマッピングされません。

OData Functions と Actions は異なる方法で呼び出されることに注意してください。Function は、パラメーター値が URI に含まれる GET リクエストによって呼び出されます。Action は、コンテンツがパラメーター値を提供する POST によって呼び出されます。

現在、バインドされていない機能およびアクションのみが互換性があります。

手順/機能のマッピング方法を検証するには、機能およびアクションについて $metadata を常に確認してください。

4.3.8. OpenAPI メタデータ

実験的機能 は、$metadata ではなく [swagger|openapi].json を介して Swagger 2.0 / OpenAPI メタデータを自動的に提供することができます。

OpenAPI 2.0 URL の例

```
http://localhost:8080/odata/swagger.json
http://localhost:8080/odata/openapi.json
http://localhost:8080/odata/openapi.json?version=2
```

OpenAPI 3.0 URL の例

```
http://localhost:8080/odata/openapi.json?version=3
```

警告

クエリーオプションや拡張が可能なクエリーオプションにより、このメタデータは OData EDM 表現よりもはるかに大きくなります。

第5章 GEOSERVER の統合

GeoServer は、地理的データのオープンソースサーバーです。Data Virtualization を統合して、さまざまなソースから地理データを提供できます。

5.1. 前提条件

- GeoServer がインストールされている。データ仮想化の統合は最初に GeoServer バージョン 2.6.x でテストされ、バージョン 2.8.x および 2.12.x と互換性があります。TEIID-5236を参照してください。
- pg/ODBC アクセス用に Data Virtualization インストールがすでに設定されている必要があります。これにより、PostGIS/PostgreSQL を使用できるように GeoServer との組み込み互換性を持たせることができます。
- 適切な Geometry 列が含まれる 1 つ以上のテーブルを公開する VDB がデプロイされています。

  a. Data Virtualization システムテーブル GEOMETY_COLUMNS は GeoServer によって使用されます。関連するジオメトリーの列に適切な srid と coord_dimensions があることを確認します。これには、ジオメトリー列の {http://www.teiid.org/translator/spatial/2015}srid と {http://www.teiid.org/translator/spatial/2015}coord_dimension 拡張プロパティーの設定が必要になる場合があります。

5.2. GEOSERVER の設定

このプロセスは、公開している各 VDB スキーマに対して、地理的データが含まれる VDB スキーマごとに繰り返す必要があります。

1. GeoServer admin Web アプリケーションを使用して Stores → Add new Store と選択します。Vector Data Sources で PostGIS を選択します。
2. JNDI 以外の接続を使用して、Data Virtualization サーバーホスト ODBC ポート、データベース（VDB Name（任意バージョン））、ユーザー、パスワードスキーマ（ターゲット VDB からのスキーマ/モデル）を入力します。

   a. VDB に%または _ のターゲットスキーマまたはテーブル名が含まれる場合、Data

Virtualization は PostgreSQL と同じデフォルトをエスケープ文字 '\' を使用するように設定し、メタデータクエリーを適切に応答させる必要があります。この GeoServer 接続プールだけを設定するために、システムプロパティー org.teiid.backslashDefaultMatchEscape を true に設定するか、Data Virtualization セッション変数 backslashDefaultMatchEscape を true に設定する必要があります。たとえば、「Session startup SQL」で「select cast（teiid_session_set('backslashDefaultMatchEscape', true)」を入力して、「Session startup SQL」に設定します。

3. *通常の GeoServer 指示に従って、Data Virtualization ストアをベースにレイヤーを作成します。*

   a. *Data Virtualization は PostGIS 関数 ST_Estimated_Extent との互換性がなく、データからバインドされたボックスの計算を試みるとログエラーが発生します。*

5.3. その他の検討事項

- *PostgreSQL ソースを統合する場合は、geometry_columns テーブルまたは geography_columns テーブルを再公開しないでください。これは、Geometry_columns を参照する非修飾クエリーを作成し、クエリーが Data Virtualization システムテーブルに対して解決される必要があるためです。*

- *Data Virtualization はデフォルトで GT_PK_METADATA を公開しません。これはオプションで GeoServer によって使用される。*

第6章 QGIS 統合

QGIS はオープンソースの地理空間プラットフォームです。Data Virtualization を統合して、さまざまなソースから地理データを提供できます。

6.1. 前提条件

- QGIS がインストールされている。データ仮想化の統合は最後にバージョン 2.14 でテストされています。
- Data Virtualization インストールは、ODBC アクセス用にすでに設定されている必要があります。これにより、PostGIS/PostgreSQL 向けの QGIS の互換性が組み込まれています。
- 適切な Geometry 列が含まれる 1 つ以上のテーブルを公開する VDB がデプロイされています。

  a. Data Virtualization システムテーブル GEOMETY_COLUMNS は QGIS によって使用されます。関連するジオメトリーの列に適切な srid と coord_dimensions があることを確認します。これには、ジオメトリー列の {http://www.teiid.org/translator/spatial/2015}srid と {http://www.teiid.org/translator/spatial/2015}coord_dimension 拡張プロパティーの設定が必要になる場合があります。

6.2. QGIS 設定

このプロセスは、公開している各 VDB スキーマに対して、地理的データが含まれる VDB スキーマごとに繰り返す必要があります。

1. QGIS GUI ブラウザーパネルで PostGIS をクリックし、「New Connection」を選択します。
2. Data Virtualization サーバーホスト、ODBC ポート、データベース（オプションバージョンのあるVDB Name）、ユーザー、およびパスワードを入力します。

   a. VDB に%または _ のターゲットスキーマまたはテーブル名が含まれる場合、Data Virtualization は PostgreSQL と同じデフォルトをエスケープ文字 '\' を使用するように設

定し、メタデータクエリーを適切に応答させる必要があります。システムプロパティー org.teiid.backslashDefaultMatchEscape を true に設定する必要があります。

3. 通常の QGIS 命令に従い、適切なスキーマを参照し、ジオメトリーを公開するテーブルを選択してレイヤーを作成します。

6.3. その他の検討事項

- PostgreSQL ソースを統合する場合は、PostGIS geometry_columns テーブルまたは geography_columns テーブルを含む postgres システムテーブルを再公開しないでください。これは、QGIS がこれらのテーブルへの修飾されていない参照を行うため、曖昧になってしまう可能性があるためです。

- スキーマまたはテーブルの作成または削除に関する操作は動作しません。

- ログには information_schema.tables に関連するメッセージが含まれる可能性があります。これは、qgis_editor_widget_styles テーブルが存在するかどうかを判断することです。データ仮想化は、QGIS エディターウィジェットスタイルと互換性がありません。

第7章 SQLALCHEMY INTEGRATION

SQLalchemy は Python のオープンソースの SQL ツールキットおよび ORM です。

7.1. 前提条件

- SQLAlchemy がインストールされている。データ仮想化の統合は最後にバージョン 1.1.6 でテストされています。

- Data Virtualization インストールは、ODBC アクセス用にすでに設定されている必要があります。これにより、PostgreSQL を使用するための SQLAlchemy との組み込み互換性が可能になります。

7.2. 用途

クエリーに SQLAlchemy エンジンを使用できるはずです。ほとんどのテーブルメタデータのリフレクションインポートも提供されます。

使用例

```
import sqlalchemy
from sqlalchemy import create_engine, Table, MetaData
engine = create_engine("postgresql+psycopg2://user:password@host:35432/vdb")
engine.connect()
#engine is ready for queries
result = connection.execute("select * from some_table")
#reflective table import
meta = MetaData()
test = Table('public.test', meta, autoload=True,
autoload_with=engine,postgresql_ignore_search_path=True)
```

7.3. 制限事項

PostgreSQL 方言のサブセットのみが利用できます。主な目的は、Data Virtualization によるクエリーを可能にすることです。必要な追加機能がある場合は、機能強化リクエストをログに記録してください。

列のメタデータは、ピリオド「.」文字が含まれるテーブルでは使用できません。必要性に応じて、単純な Data Virtualization 名を使用し、ソーススキーマ修飾名を使用しないインポート設定が必要になる場合があります。

7.4. アプリケーションの互換性

7.4.1. superset

superset は オープンソースのデータ可視化およびダッシュボードビルダーです。SQLAlchemy を使用してリレーショナルデータベースにアクセスします。

上記の指示に従うと、Sources メニューの下に Database を追加して、Data Virtualization VDB にアクセスできます。

URL は、SQLAlchemy 統合と同じ形式 (postgresql+psycopg2://user:password@host:35432/vdb)で表示されます。

集約およびすべての基本タイプに関連する基本的な使用シナリオがテストされています。必要な追加機能がある場合は、機能強化リクエストをログに記録してください。

第8章 NODE.JS INTEGRATION

Node.js は、Data Virtualization と統合できるオープンソースのイベント駆動型ランタイムです。

8.1. 前提条件

- Node.js がインストールされている。npm pagckage pg も必要です。「使用」を使用します。

- Data Virtualization インストールは、ODBC アクセス用にすでに設定されている必要があります。これにより、PostGIS/PostgreSQL 用の Node.js との任意の互換性を使用できます。

8.2. 用途

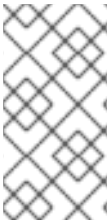
たとえば、Data Virtualization サーバーに「northwind」という名前の VDB があり、「customers」というテーブルがあり、以下のようなデフォルト設定を使用している場合です。

user = 'user' password = 'user' host = 127.0.0.1 port = 35432

シンプルなアクセスの例

```
const { Client } = require('pg')
const client = new Client({
  user: 'user',
  host: 'localhost',
  database: 'northwind',
  password: 'secretpassword',
  port: 35432,
})
client.connect()

client.query('SELECT CustomerID, ContactName, ContactTitle FROM Customers', (err, res)
=> {
  console.log(err, res)
  client.end()
})
```



注記

環境変数やその他のメカニズムから取得できるため、コードで接続情報をプログラム的に指定する必要はありません。https://node-postgres.comを参照してください。

詳細は、https://npmjs.org/package/pgを参照してください。

第9章 ADO.NET 統合

Npgsql は、PostgreSQL のオープンソースの ADO.NET Data Provider です。Data Virtualization と統合すると、C#、Visual Basic、F# で書かれたプログラムからアクセスが可能になります。

9.1. 前提条件

- .msi Windows インストーラーを使用して Npgsql をインストールします。データ仮想化の統合は最後にバージョン 3.2.6 でテストされています。
- pg/ODBC アクセス用に Data Virtualization インストールがすでに設定されている必要があります。
- VDB がデプロイされていること。

9.2. NPGSQL 設定

利用可能な接続パラメーターの詳細は、Npgsql のドキュメント を参照してください。すべての設定パラメーターが Data Virtualization で使用するためにテストされているわけではありません。

9.3. 既知の制限事項

- TEIID-5220 は、テーブルとビューのメタデータを表示しませんが、クエリーには影響を与えません。PowerBi などの特定のツールには、メタデータイントロスペクションを実行する必要性を示すオプションが必要になる場合があります。

第10章 再認証

データ仮想化により、接続の再認証が可能になり、新規接続全体を作成するのではなく、接続のアイデンティティーを変更できるようになります。 JDBC を使用している場合は、changeUser Connection エクステンション を参照してください。 ODBC を使用している場合や、再認証のためにステートメントベースのメカニズムが必要な場合は、SESSION AUTHORIZATION の SET Statement も参照してください。

第11章 実行プロパティー

実行プロパティーは、Data VirtualizationStatement インターフェースまたは SET ステートメント経由で接続ごとに設定できます。便宜上、プロパティーキーは org.teiid.jdbc.ExecutionProperties インターフェースの定数によって定義されます。

表11.1 実行プロパティー

Property Name/String Constant	説明
PROP_TXN_AUTO_WRAP / autoCommitTxn	connection プロパティーと同じです。
PROP_PARTIAL_RESULTS_MODE / partialResultsMode	部分結果モードを参照してください。
RESULT_SET_CACHE_MODE / resultSetCacheMode	connection プロパティーと同じです。
SQL_OPTION_SHOWPLAN / SHOWPLAN	connection プロパティーと同じです。
NOEXEC / NOEXEC	connection プロパティーと同じです。
JDBC4COLUMNNAMEANDLABELSEMANTICS / useJDBC4ColumnNameAndLabelSemantics	connection プロパティーと同じです。

第12章 SET ステートメント

実行プロパティーは、SET ステートメントを使用して接続でも設定できます。SET ステートメントはまだ Data Virtualization の言語機能ではなく、JDBC クライアントでのみ処理されます。JDBC クライアントは pg/ODBC トランスポートをサポートするため、これも機能します。

SET 構文：

- SET [PAYLOAD](parameter|SESSION AUTHORIZATION)値
- セッション特性をトランザクション分離レベルとして設定します（コミットされていない|READ COMMITTED|REPEATABLE READ|LEASE）

構文ルール：

- パラメーターは識別子でなければなりません。引用された場合にのみ、スペースまたは他の特殊文字を使用できません。
- 値は引用されていない識別子または引用符で囲まれた文字列リテラル値のいずれかです。
- ペイロードが指定されている場合（例： ペイロード）。"SET PAYLOAD x y" の後に、セッションスコープペイロードプロパティーオブジェクトには対応する名前/値のペアが設定されます。 ペイロードオブジェクトは完全なセッションスコープではありません。 これは、XAConnection ハンドルがプールに閉じられ/返されるとセッションから削除されます（Data VirtualizationDataSource の使用を想定）。 セッションスコープのペイロードは、Data VirtualizationStatement.setPayload の使用に優先されます。
- SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL の使用は、対応するレベルで Connection.setTransactionIsolation を呼び出すことに相当します。

SET ステートメントは、計画および実行を制御するために最も一般的に使用されます。

- SET SHOWPLAN(ON|DEBUG|OFF)

- *SET NOEXEC(ON|OFF)*

プランのデバッグの有効化

```
Statement s = connection.createStatement();
s.execute("SET SHOWPLAN DEBUG");
...
Statement s1 = connection.createStatement();
ResultSet rs = s1.executeQuery("select col from table");

ResultSet planRs = s1.exeuteQuery("SHOW PLAN");
planRs.next();
String debugLog = planRs.getString("DEBUG_LOG");
```

クエリーを実行せずにクエリー計画

```
s.execute("SET NOEXEC ON");
s.execute("SET SHOWPLAN DEBUG");
...
e.execute("SET NOEXEC OFF");
```

SET ステートメントは、承認を制御するのにも使用できます。SET SESSION AUTHORIZATION ステートメントは、接続に現在設定されているクレデンシャルが指定された Reauthentication を実行します。接続認証情報は、SET PASSWORD ステートメントを実行して変更できます。SET PASSWORD ステートメントは再認証を実行しません。

セッション承認の変更

```
Statement s = connection.createStatement();
s.execute("SET PASSWORD 'someval'");
s.execute("SET SESSION AUTHORIZATION 'newuser'");
```

第13章 SHOW ステートメント

SHOW ステートメントを使用すると、情報の可変を確認できます。SHOW ステートメントはまだ Data Virtualization の言語機能ではなく、JDBC クライアントでのみ処理されます。

使用方法の表示：

- SHOW PLAN: clob 列 PLAN_TEXT、xml 列 PLAN_XML、および clob 列 DEBUG_LOG と、以前実行したクエリーの値を含む行と共に resultset を返します。SHOWPLAN が OFF の場合は、プランが利用できない場合は、行が返されません。SHOWPLAN が DEBUG に設定されていない場合は、DEBUG_LOG は null 値を返します。

- SHOW ANNOTATIONS- 文字列列 CATEGORY、PRIORITY、ANNOTATION、RESOLUTION、および以前に実行したクエリーの各アノテーションの行が含まれる結果セットを返します。SHOWPLAN が OFF の場合は、プランが利用できない場合は、行が返されません。

- SHOW <property> - SET の逆で、指定のプロパティーのプロパティー値を表示し、プロパティーキーに一致する名前を持つ単一の文字列列が含まれる結果セットを返します。

- SHOW ALL: NAME 文字列列と、すべてのプロパティー値の行エントリーを含む VALUE 文字列列が含まれる結果セットを返します。SHOW ステートメントは、クエリー計画の取得に最も一般的に使用されます。プランのデバッグ例を参照してください。

第14章 トランザクション

Data Virtualization は、クライアントパースペクティブから 3 種類のトランザクションを提供します。

1. グローバル
2. ローカル
3. リクエストレベル

すべては、XA トランザクションとして Data Virtualization によって論理的に実装されます。XA トランザクションの詳細は、「JTA 仕様」を参照してください。

14.1. ローカルトランザクション

クライアントパースペクティブからのローカルトランザクションは単一リソースにのみ影響しますが、複数のステートメントを調整できます。

14.1.1. JDBC Specific

Connection クラスは autoCommit フラグを使用して、ローカルトランザクションを明示的に制御します。デフォルトでは、autoCommit は、リクエストレベルまたは暗黙的なトランザクション制御を示す true に設定されます。

autoCommit フラグを false に設定して、ローカルトランザクションの使用例。

自動コミットを使用したローカルトランザクションの制御

```
// Set auto commit to false and start a transaction
connection.setAutoCommit(false);

try {
   // Execute multiple updates
   Statement statement = connection.createStatement();
   statement.executeUpdate("INSERT INTO Accounts (ID, Name) VALUES (10, 'Mike')");
```

```
    statement.executeUpdate("INSERT INTO Accounts (ID, Name) VALUES (15, 'John')");
    statement.close();

    // Commit the transaction
    connection.commit();

} catch(SQLException e) {
    // If an error occurs, rollback the transaction
    connection.rollback();
}
```

以下の例では、いくつかのことを紹介します。

1. autoCommit フラグを false に設定します。これにより、接続にバインドされるトランザクションが開始されます。

2. トランザクションのコンテキスト内で複数の更新を実行します。

3. ステートメントが完了すると、commit（）を呼び出すことでトランザクションがコミットされます。

4. エラーが発生した場合は、rollback（）メソッドを使用してトランザクションをロールバックします。

以下の操作のいずれかがローカルトランザクションを終了します。

1. connection.setAutoCommit(true)- 以前に false に設定されていた場合

2. Connection.commit()

3. Connection.rollback()

4. トランザクションはタイムアウトになると自動的にロールバックされます。

14.1.1.1. JDBC ローカルトランザクション制御の無効化

すべてのアクセスが読み取り専用で、トランザクションが必要ない場合でも、Data Virtualization 上のツールまたはフレームワークが setAutoCommit(false)、commit（）、および rollback（）を呼び出すことがあります。ローカルトランザクションデータ仮想化のスコープで XA トランザクションが開始され、コミットしようとすると、設定が複雑化したり、パフォーマンスが低下する可能性があります。

このような場合は、デフォルトの JDBC 動作を上書きし、実行されているコマンドに関係なくこれらのメソッドがアクションを実行しないことを示すことができます。ローカルトランザクションの使用を無効にするには、このプロパティーを JDBC 接続 URL に追加します。

```
disableLocalTxn=true
```

ヒント

ローカルトランザクションをオフにすると危険にさらされる可能性があり、データの読み取り時に（データの読み取り時）または一貫性のないデータにデータを書き込む可能性があります（データを書き込む場合）。安全性を確保するには、呼び出したアプリケーションがローカルトランザクションを必要としないことを確かめる場合にのみこのモードを使用する必要があります。

14.1.2. トランザクションステートメント

ODBC クライアントにも適用されるトランザクション制御ステートメントが、ローカルトランザクション境界を明示的に制御します。関連するステートメントは以下のとおりです。

- start TRANSACTION: connection.setAutoCommit(false)の同義語
- COMMIT- synonym for connection.setAutoCommit(true)
- ROLLBACK: connection.rollback（） の同義語で、自動コミットモードに戻ります。

14.2. リクエストレベルのトランザクション

リクエストレベルのトランザクションは、リクエストがグローバルまたはローカルトランザクションの範囲にない場合に使用されます。つまり、「autoCommit」は「true」を意味します。リクエストレベルのトランザクションでは、アプリケーションは明示的にコミットまたはロールバックを呼び出す必

要はありません。そのため、すべてのコマンドは、サーバーによって自動的にコミットまたはロールバックされる独自のトランザクションであると見なします。

Data Virtualization サーバーは、仮想テーブルを介して更新を実行できます。これらの更新により、アプリケーションが 1 つの仮想テーブルに対して更新コマンドを実行しても、複数の物理システムに対して更新が実行される可能性があります。多くの場合、クエリーされたテーブルが複数のソースを更新し、トランザクションを必要とするかどうかを認識しないことがあります。

このため、Data Virtualization サーバーを使用すると、必要に応じてアプリケーションがトランザクションでコマンドを自動的にラップできます。このラッピングによりクエリーのパフォーマンスペナルティーが発生するため、お使いの環境に合わせて、利用可能なラッピングモードから選択することができます。アプリケーションが必要とする整合性とパフォーマンスの中から選択する必要があります。たとえば、データソースがトランザクションに準拠していない場合、トランザクションのラッピングをオフに（完全にオフ）して、パフォーマンスを最大化する可能性があります。

トランザクションのラッピングは、以下のいずれかのモードに設定できます。

1. **ON:** このモードは、必要なかどうかを確認せずに、トランザクション内のすべてのコマンドを常にラップします。これは最も安全なモードです。

2. **OFF:** このモードは、トランザクションでコマンドを自動的にラップしたり、コマンドをラップする必要があるかどうかを確認します。このモードは、エラーなしでトランザクション外の複数のソース更新を許可するため、危険にさらされる可能性があります。このモードでは、更新またはトランザクションを使用しないアプリケーションに最適なパフォーマンスがあります。

3. **DETECT:** このモードは、ユーザーがトランザクションで複数のソース更新を実行することを認識しないことを前提としています。Data Virtualization Server は、すべてのコマンドをチェックして、そのコマンドが複数のソース更新であるかを確認し、トランザクションでラップします。1 つのソースの場合は、ソースレベルのコマンドトランザクションを使用します。実行プロパティーを使用して、接続を確立するときやクエリーごとにトランザクションモードをプロパティーとして設定できます。実行プロパティーの詳細は、「実行プロパティー」を参照してください。

14.2.1. 複数の挿入バッチ

クエリー式（または非推奨の SELECT INTO）で INSERT を実行する場合、別のソース INSERTS で処理される複数の挿入バッチは Data Virtualization サーバーで処理される可能性があります。対象となるソースが XA をサポートするか、または障害発生時に補正アクションが実行されるようにしてください。

14.3. グローバルトランザクションの使用

グローバルまたはクライアントの XA トランザクションは JDBC クライアントにのみ適用されます。1 つのトランザクションで複数のリソースを調整するため、すべてのクライアント。クライアント側で XA トランザクションを利用するには、Data VirtualizationDataSource （またはトランザクション検出が有効な Data Virtualization Embedded）を使用します。

JBoss、WebSphere、Weblogic などのアプリケーションサーバーの UserTransaction のコンテキストで XAConnection を使用する場合、生成される接続は現在の XA トランザクションにすでに関連付けられます。XA トランザクションと対話するのに、追加のクライアント JDBC コードは必要ありません。

UserTransaction での使用

```
UserTransaction ut = context.getUserTransaction();
try {
  ut.begin();
  Datasource oracle = lookup(...)
  Datasource teiid = lookup(...)

  Connection c1 = oracle.getConnection();
  Connection c2 = teiid.getConnection();

  // do something with Oracle connection
  // do something with Data Virtualization connection
  c1.close();
  c2.close();
  ut.commit();
} catch (Exception ex) {
   ut.rollback();
}
```

JEE コンテナー環境で稼働しておらず、XA トランザクションを調整する独自のトランザクションがある場合は、コードは以下のようになります。

XA トランザクションの手動使用

```
XAConnection xaConn = null;
XAResource xaRes = null;
Connection conn = null;
```

```
Statement stmt = null;

try {
  xaConn = <XADataSource instance>.getXAConnection();
  xaRes = xaConn.getXAResource();
  Xid xid = <new Xid instance>;
  conn = xaConn.getConnection();
  stmt = conn.createStatement();

  xaRes.start(xid, XAResource.TMNOFLAGS);
  stmt.executeUpdate("insert into ...");
  <other statements on this connection or other resources enlisted in this transaction>
  xaRes.end(xid, XAResource.TMSUCCESS);

  if (xaRes.prepare(xid) == XAResource.XA_OK) {
    xaRes.commit(xid, false);
  }
}
catch (XAException e) {
  xaRes.rollback(xid);
}
finally {
  <clean up>
}
```

グローバルトランザクションを使用すると、複数の Data Virtualization XAConnections が同じトランザクションに参加する可能性があります。Data Virtualization JDBC XAResource "isSameRM" メソッドは、クラスターの同じサーバーインスタンスへの接続が作成された場合にのみ「true」を返します。Data Virtualization 接続が異なるサーバーインスタンスである場合、トランザクション動作が同じクラスターメンバーの場合と同じではない場合があります。たとえば、クライアントトランザクションマネージャーが各接続に同じ XID を使用する場合（isSameRM が false を返すため）、異なるクラスターメンバー経由でアクセスされる同じ物理ソースから重複 XID 例外が発生することがあります。クライアントトランザクションマネージャーが各接続に異なるブランチ識別子を使用する場合は、ブランチ ID に基づいて変更をロックしたり、分離したりする際に問題が発生する可能性があります。

14.4. 制約

14.4.1. アプリケーションの制限

グローバル、ローカル、およびリクエストレベルのトランザクションの使用はすべて相互に排他的です。リクエストレベルのトランザクションは、グローバルまたはローカルトランザクションにない場合にのみ適用されます。グローバルトランザクションとローカルトランザクションを同時に組み合わせようとすると、例外が発生します。

14.4.2. EIS(Enterprise Information System)の互換性

EIS システムおよび EIS システム自体を表す基礎となるデータソースは、Data Virtualization 経由で分散 XA トランザクションに参加したい場合は XA トランザクションをサポートする必要があります。ソースシステムが XA に対応していない場合、分散トランザクションに完全に参加できません。ただし、ソースは引き続き XA サポートなしでデータ統合に参加できます。

XA トランザクションへの参加は、ソースの XA 機能を基に自動的に決定されます。XA リソースが分散トランザクションに参加する必要がある場合に XA リソースを設定するのはユーザーの責任です。