Novell JDBC* 用の Identity Manager ドライバ

2.1

www.novell.com

実装ガイド

2006 年 5 月 12 日

Novell®

保証と著作権

米国 Novell, Inc. およびノベル株式会社は、本書の内容または本書に起因する結果に関して、いかなる表示も行いません。また、本書の商品性、および特定用途への適合性について、いかなる黙示の保証も行いません。米国 Novell, Inc. およびノベル株式会社は、本書の内容を改訂または変更する権利を常に留保します。米国 Novell, Inc. およびノベル株式会社は、このような改訂または変更を個人または事業体に通知する義務を負いません。

米国 Novell, Inc. およびノベル株式会社は、ノベル製ソフトウェアの使用に起因する結果に関して、いかなる表示も行いません。また、商品性、および特定目的への適合性について、いかなる黙示の保証も行いません。米国 Novell, Inc. およびノベル株式会社は、ノベル製ソフトウェアの内容を変更する権利を常に留保します。米国 Novell, Inc. およびノベル株式会社は、このような変更を個人または事業体に通知する義務を負いません。

本契約の締結に基づいて提供されるすべての製品または技術情報には、米国の輸出管理規定およびその他の国の貿易関連法規が適用されます。お客様は、取引対象製品の輸出、再輸出または輸入に関し、国内外の輸出管理規定に従うこと、および必要な許可、または分類に従うものとします。お客様は、現在の米国の輸出除外リストに掲載されている企業、および米国の輸出管理規定で指定された輸出禁止国またはテロリスト国に本製品を輸出または再輸出しないものとします。お客様は、取引対象製品を、禁止されている核兵器、ミサイル、または生物化学兵器を最終目的として使用しないものとします。本ソフトウェアの輸出については、www.novell.co.jp/info/exports/expmtx.html または www.novell.com/ja-jp/company/exports/ もあわせてご参照ください。弊社は、お客様が必要な輸出承認を取得しなかったことに対し如何なる責任も負わないものとします。

本書に記載された製品で使用されている技術に関連する知的所有権は、弊社に帰属します。これらの知的所有権は、http://www.novell.com/company/legal/patents/ に記載されている 1 つ以上の米国特許、および米国ならびにその他の国における 1 つ以上の特許または出願中の特許を含む場合があります。

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

オンラインマニュアル : 本製品とその他の Novell 製品のオンラインマニュアルにアクセスする場合や、アップデート版を入手する場合は、www.novell.com/ja-jp/documentation をご覧ください。

Novell の商標

Novell の商標の一覧については、「商標 (http://www.novell.com/company/legal/trademarks/tmlist.html)」を参照してください。

第三者の商標

第三者の商標は、それぞれの所有者に属します。

目次

このガイドについて

Java* Database Connectivity (JDBC*) 用 Identity Manager ドライバは、アイデンティティボールトとリレーショナルデータベース間でデータを同期するための汎用ソリューションを提供します。

このガイドでは、ドライバの技術の概要および設定方法について説明します。

対象読者

このガイドは、JDBC 用 Identity Manager ドライバを使用する Novell® eDirectory および Identity Manager の管理者を対象にしています。

ご意見やご要望

このマニュアルおよび Novell Identity Manager に含まれるその他のマニュアルに関するご意見やご要望をお聞かせください。オンラインヘルプの各ページの下部にあるユーザコメント機能を使用するか、または www.novell.com/documentation/feedback.html にアクセスして、ご意見をお寄せください。

最新のマニュアル

このマニュアルの最新のバージョンについては、Identity Manager のマニュアルの Web サイト (http://www.novell.com/documentation/lg/dirxmldrivers/index.html) を参照してください。

その他のマニュアル

Identity Manager およびその他のドライバの使用に関するマニュアルについては、Identity Manager のマニュアルの Web サイト (http://www.novell.com/documentation/lg/dirxmldrivers) を参照してください。

表記規則

本マニュアルでは、手順に含まれる複数の操作および相互参照パス内の項目を分けるために、大なり記号 (>) を使用しています。

商標記号 (®、™ など) は、Novell® の商標を示します。アスタリスク (*) は第三者の商標を示します。

JDBC 用 Identity Manager ドライバの概要

1

Java DataBase Connectivity (JDBC) 用 Identity Manager ドライバは、Identity Manager と、JDBC でアクセスできるリレーショナルデータベース間でデータベースを同期する汎用ソリューションを提供します。

このドライバのプリンシパル値は、その汎用の特性として存在します。ほとんどのドライバは 1 つのアプリケーションと入出力を行いますが、それとは異なり、このドライバは、ほとんどのリレーショナルデータベース、およびデータベースをホストするアプリケーションとの入出力を行うことができます。

1.1 ドライバの新機能

Identity Manager 3 には、ドライバに次の新機能があります。

- トリガなし発行。18 ページのセクション 1.6「トリガなしとトリガ付きの発行」を参照してください。
- バッチ処理。85 ページの 「バッチサイズ」を参照してください。
- 将来のイベント処理。79 ページの 「将来のイベント処理を有効にしますか ?」を参照してください。
- 毎日発行。84 ページの 「Publication Time of Day (発行時刻)」を参照してください。
- データベースのサポートの拡大。141 ページのセクション 7.2「サポートされているデータベース」を参照してください。
- データベースの時間タイプのサポートの強化。49 ページの 「時間構文」を参照してください。
- 操作性の向上。43 ページのセクション 4.1「スマート環境設定」を参照してください。
- スキーマフィルタリング。54 ページの 「フィルタ式を含む」および 55 ページの「フィルタ式を除く」を参照してください。
- ビューのサポートの拡張。95 ページの 「直接同期」を参照してください。
- サードパーティ製のドライバの暗号メカニズムのサポートの強化。57 ページの 「接続初期化ステートメント」を参照してください。
- パスワードの変更および確認のサポート。
- ドライバ環境設定 / データベース SQL* スクリプトの向上。

Identity Manager の新機能については、『*Identity Manager 3.0 インストールガイド*』の「Identity Manager 3 の新機能」を参照してください。

1.2 用語の変更

次の用語が、旧リリースから変わりました。

表 *1-1* 用語の変更

旧用語	新用語
DirXML®	Identity Manager
DirXML サーバ	メタディレクトリサーバ
DirXML エンジン	メタディレクトリエンジン
eDirectory™	アイデンティティボールト (eDirectory 属性またはクラスを参照する場合は除く)

1.3 用語と概念

- 8 ページの 「JDBC」
- 9 ページの 「JDBC 用 Identity Manager ドライバ」
- 9 ページの 「サードパーティ製の JDBC ドライバ」
- 9 ページの 「アイデンティティボールト」
- 9 ページの 「ディレクトリスキーマ」
- 10 ページの 「アプリケーションスキーマ」
- 10 ページの 「データベーススキーマ」
- 10 ページの 「同期スキーマ」
- 10 ページの 「論理データベースクラス」
- 10 ページの 「XDS」

1.3.1 JDBC

Java DataBase Connectivity (JDBC) は、Sun* Microsystems* が開発したクロスプラットフォームデータベースインタフェース標準です。

ほとんどのエンタープライズデータベースベンダは、JDBC インタフェースの一意の実装を提供しています。使用できる JDBC インタフェースのバージョンは以下の 3 つです。

- JDBC 1 (Java 1.0)
- JDBC 2 (Java 1.2 または 1.3)
- JDBC 3 (Java 1.4 または 1.5)

JDBC 用 Identity Manager ドライバは、JDBC 1 インタフェースを使用します。サードパーティ製の JDBC ドライバでサポートされている場合は、JDBC 2 または JDBC 3 メソッドの小さいサブセットを使用します。

1.3.2 JDBC 用 Identity Manager ドライバ

JDBC 用 Identity Manager ドライバは、JDBC インタフェースを使用して、アイデンティティボールトとリレーショナルデータベース間のデータおよび識別情報を同期します。

ドライバは 4 つの jar ファイルで構成されています。

- JDBCShim.jar
- JDBCUtil.jar
- JDBCConfig.jar
- CommonDriverShim.jar

これらのファイルに加えて、個別のデータベースと通信するためにサードパーティ製の JDBC ドライバが必要です。

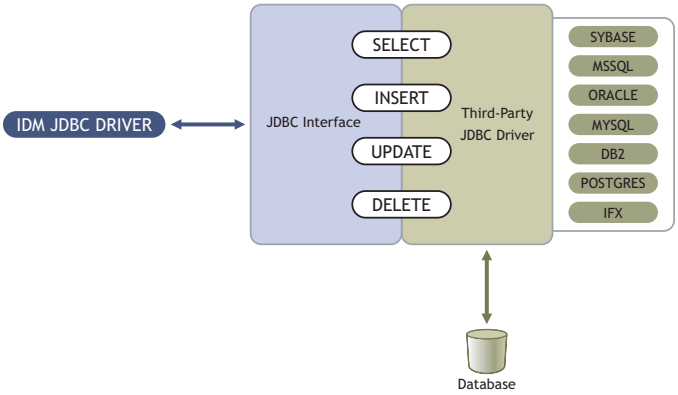
1.3.3 サードパーティ製の JDBC ドライバ

サードパーティ製の JDBC ドライバは、JDBC 用 Identity Manager ドライバが特定のデータベースと通信するために使用する多数の JDBC インタフェース実装の 1 つです。

たとえば、classes12.zip は Oracle* 製の JDBC ドライバの 1 つです。さまざまなサードパーティ製の JDBC ドライバが JDBC インタフェース仕様のさまざまな部分を実装します。そのインタフェースの実装は、それぞれ一貫した方法で行われます。

次の図は、JDBC 用 Identity Manager ドライバとサードパーティ製の JDBC ドライバの関係を示します。

図 *1-1* *JDBC 用 IDM ドライバとサードパーティ製の JDBC ドライバ*



1.3.4 アイデンティティボールト

アイデンティティボールトは、Identity Manager が使用するデータストアです。

1.3.5 ディレクトリスキーマ

ディレクトリスキーマは、ディレクトリ内のオブジェクトのクラスおよび属性のセットです。

たとえば、eDirectory™ ユーザクラスおよび名前属性は、eDirectory スキーマの一部です。

1.3.6 アプリケーションスキーマ

アプリケーションスキーマは、アプリケーション内のクラスおよび属性のセットです。

データベースにはクラスまたは属性の概念がないので、JDBC 用ドライバは、eDirectory クラスをテーブルまたはビューにマップし、eDirectory 属性を列にマップします。

1.3.7 データベーススキーマ

データベーススキーマは、基本的に所有権と同義です。データベーススキーマは、データベースユーザが所有するデータベースオブジェクト (テーブル、ビュー、トリガ、ストアドプロシージャ、関数など) で構成されます。

JDBC 用ドライバでは、スキーマは、データベースをスコープする (ランタイム時にドライバに表示できるデータベースオブジェクト数を減らす) 場合に便利です。

所有権は、修飾されたドット表記を使用して表現されることがよくあります。たとえば、indirect.usr のように表記します。ここで、indirect はテーブル usr を所有するデータベースユーザの名前です。indirect によって所有されるすべてのデータベースオブジェクトは間接データベーススキーマを構成します。

1.3.8 同期スキーマ

同期スキーマは、実行時にドライバに表示できるデータベーススキーマです。

1.3.9 論理データベースクラス

論理データベースクラスは、データベース内で eDirectory クラスを表現するために使用されるテーブルのセットまたはビューです。

1.3.10 XDS

XDS 形式は、Novell® が定義した、Identity Manager で使用できる XML 形式のサブセットです。

XDS は、アイデンティティボールトからのデータの初期形式です。デフォルトのルールを変更してスタイルシートを変更すれば、JDBC 用ドライバをあらゆる XML 形式と連動するように設定できます。

1.4 データベースの概念

- 11 ページの 「Structured Query Language」
- 11 ページの 「Data Manipulation Language」
- 11 ページの 「Data Definition Language」
- 11 ページの 「ビュー」
- 12 ページの 「識別列 / シーケンス」
- 12 ページの 「トランザクション」
- 13 ページの 「ストアドプロシージャおよび関数」

- 13 ページの 「トリガ」
- 14 ページの 「instead-of-trigger」

1.4.1 Structured Query Language

Structured Query Language (SQL) は、リレーショナルデータベース内のデータのクエリおよび操作に使用される言語です。

1.4.2 Data Manipulation Language

Data Manipulation Language (DML) ステートメントは、データベースのデータを操作する高度に標準化された SQL ステートメントです。

DML ステートメントは、使用するデータベースに関係なく実質的に同じです。JDBC 用ドライバは DML ベースです。XDS XML で表現された Identity Manager イベントを標準化された DML ステートメントにマップします。

次に DML ステートメントの例を示します。

```
SELECT * FROM usr; INSERT INTO usr(lname) VALUES('Doe'); UPDATE usr SET
fname = 'John' WHERE idu = 1;
```

1.4.3 Data Definition Language

Data Definition Language (DDL) ステートメントは、テーブル、インデックス、およびユーザアカウントなどのデータベースオブジェクトを操作します。

DDL ステートメントは、各社独自のもので、データベース間で大きく異なります。JDBC 用ドライバは DML ベースですが、XDS イベントに DDL ステートメントを埋め込むことができます。詳細については、109 ページの 「XDS イベントへの SQL ステートメントの埋め込み」を参照してください。

次に DDL ステートメントの例を示します。

```
CREATE TABLE usr ( idu   INTEGER, fname VARCHAR2(64), lname
VARCHAR2(64) );

CREATE USER idm IDENTIFIED BY novell;
```

注 : このマニュアルで使用する例は、Oracle データベースの場合の例です。

1.4.4 ビュー

ビューは論理テーブルです。

SELECT ステートメントを使用して問い合わせた場合、ビューは、それが定義されたときに指定された SQL クエリを実行することによって構成されます。ビューは、任意の構造の複数のテーブルを 1 つのテーブルまたは論理データベースクラスとして表現するのに役立つ抽象化メカニズムです。

```
CREATE VIEW view_usr ( pk_idu, fname, lname ) AS SELECT idu, fname,
lname from usr;
```

1.4.5 識別列 / シーケンス

識別列およびシーケンスは、一意のプライマリキー値の生成に使用されます。Identity Manager は、特にこれらの値を関連付けることができます。

識別列は、テーブル内の行を一意に識別するために使用される列で、自動的に増分されます。識別列の値は、テーブルに行が挿入されると自動的に入力されます。

シーケンスオブジェクトは、テーブル内の行を一意に識別するために使用できるカウンタです。識別列とは異なり、シーケンスオブジェクトは単一のテーブルにはバインドされません。ただし、テーブルが 1 つの場合は、シーケンスオブジェクトを使用して同じ結果を得ることができます。

次に、シーケンスオブジェクトの例を示します。

```
CREATE SEQUENCE seq_idu START WITH 1 INCREMENT BY 1 NOMINVALUE
NOMAXVALUE ORDER;
```

1.4.6 トランザクション

トランザクションは、1 つまたは複数のステートメントで構成される最小単位のデータベース操作です。

トランザクションが終了すると、トランザクション内のすべてのステートメントがコミットされます。トランザクションが中断されるか、トランザクション内のいずれかのステートメントでエラーが発生すると、トランザクションはロールバックするように指示されます。トランザクションがロールバックされると、データベースはトランザクションが開始される前の状態に戻ります。

トランザクションは、手動 (ユーザ定義) または自動です。手動トランザクションは、1 つまたは複数のステートメントで構成でき、明示的なコミットが必要です。自動トランザクションは、1 つのステートメントで構成され、各ステータスの実行後に暗黙的にコミットされます。

手動 (ユーザ定義) トランザクション

手動トランザクションには、通常、複数のステートメントが含まれます。一般に、手動トランザクション内で、DDL ステートメントを DML ステートメントとグループにすることはできません。

次の例は、手動トランザクションを示します。

```
SET AUTOCOMMIT OFF INSERT INTO usr(lname) VALUES('Doe'); UPDATE usr
SET fname = 'John' WHERE idu = 1; COMMIT; -- explicit commit
```

自動トランザクション

自動トランザクションは、1 つのステートメントだけで構成されます。各ステートメントの後に変更が暗黙的にコミットされるので、よく自動コミットステートメントと呼ばれます。自動コミットステートメントは、その他のステートメントの自律型です。

次の例は、自動トランザクションを示します。

```
SET AUTOCOMMIT ON INSERT INTO emp(lname) VALUES('Doe'); -- implicit
commit
```

1.4.7 ストアドプロシージャおよび関数

ストアドプロシージャまたは関数は、データベースに保存されるプログラムロジックです。ストアドプロシージャまたは関数は、ほとんどすべてのコンテキストから呼び出すことができます。

購読者チャネルは、ストアドプロシージャまたは関数を使用して、テーブルに挿入された行からプライマリキー値を取得して、関連付けを作成します。ストアドプロシージャまたは関数は、埋め込まれた SQL ステートメントまたはトリガからも呼び出すことができます。

ストアドプロシージャと関数の違いは、データベースによってさまざまです。一般に、いずれも出力を返すことができますが、その方法が異なります。ストアドプロシージャは、通常、パラメータを通じて値を返します。関数は、通常、スカラ返り値または結果セットを通じて値を返します。

次の例は、シーケンスオブジェクトの次の値を返すストアドプロシージャ定義を示します。

```
CREATE SEQUENCE seq_idu START WITH 1 INCREMENT BY 1 NOMINVALUE
NOMAXVALUE ORDER;


CREATE PROCEDURE sp_idu(io_idu IN OUT INTEGER) IS BEGIN IF (io_idu IS
NULL) THEN SELECT seq_idu.nextval INTO io_idu FROM DUAL; END IF; END
sp_idu;
```

1.4.8 トリガ

データベーストリガは、テーブルに関連付けられているプログラムロジックで、一定の条件下で実行されます。トリガは、実行条件が満たされたときにその実行を指示されます。

一般に、トリガは、データベース内の副次的な動作の作成に役立ちます。JDBC 用ドライバのコンテキストでは、トリガは、イベント発行のキャプチャに役立ちます。次に usr テーブルのデータベーストリガの例を示します。

```
CREATE TABLE usr ( idu   INTEGER, fname VARCHAR2(64), lname
VARCHAR2(64) );
```

```
-- t = trigger; i = insert CREATE TRIGGER t_usr_i AFTER INSERT ON usr
FOR EACH ROW

BEGIN UPDATE usr SET fname = 'John'; END;
```

トリガを含むテーブルに対してステートメントが実行されるときに、ステートメントがトリガで指定された条件を満たすと、トリガが実行されます。たとえば、例として挙げたテーブルを使用して、次の挿入ステートメントが実行されるとします。

```
INSERT INTO usr(lname) VALUES('Doe')
```

トリガ t_emp_i は、挿入ステートメントが実行された後に実行され、次の更新ステートメントも実行されます。

```
UPDATE usr SET fname = 'John'
```

トリガは、通常、それをトリガするステートメントの実行前または後に実行できます。データベーストリガの一部として実行されるステートメントは、通常、トリガするステートメントと同じトランザクションに含まれます。前の例では、INSERT と UPDATE のステートメントが一緒にコミットまたはロールバックされます。

1.4.9 instead-of-trigger

instead-of-trigger は、ビューに関連付けられているプログラムロジックで、一定の条件下で実行されます。

instead-of-trigger は、ビューを書き込み可能または購読可能にする場合に便利です。このトリガは、多くの場合、INSERT、UPDATE、およびビューからの DELETE の対象が何かを定義するために使用されます。次に usr テーブルの instead-of-trigger の例を示します。

```
CREATE TABLE usr ( idu   INTEGER, fname VARCHAR2(64), lname
VARCHAR2(64) );



CREATE VIEW view_usr ( pk_idu, fname, lname ) AS SELECT idu, fname,
lname from usr; -- t = trigger; i = insert CREATE TRIGGER t_view_usr_i
INSTEAD OF INSERT ON usr BEGIN INSERT INTO usr(idu, fname, lname)
VALUES(:NEW.pk_idu, :NEW.fname, :NEW.lname); END;
```

instead-of-trigger を含むビューに対してステートメントが実行されるときに、ステートメントがトリガで指定された条件を満たすと、instead-of-trigger が実行されます。トリガとは異なり、instead-of-trigger は、常にトリガするステートメントの前に実行されます。また、標準のトリガとは異なり、instead-of-trigger は、トリガするステートメントに追加されるのではなく、その代わりに実行されます。

たとえば、例として挙げたビューを使用して、元の挿入ステートメントの代わりに次の挿入ステートメントが実行されるとします。

```
INSERT INTO view_usr(pk_idu, fname, lname) VALUES(1, 'John', 'Doe')
```

元のステートメントを実行するのではなく、instead-of-trigger t_view_usr_i が実行され、次のステートメントを実行します。

```
INSERT INTO usr(idu, fname, lname) VALUES(:NEW.pk_idu, :NEW.fname,
:NEW.lname);
```

この例では、両方のステートメントは同等になります。

1.5 データ同期モデル

ドライバは、直接と間接の 2 つのデータ同期モデルをサポートしています。いずれの用語も、同期されるデータの最終的なターゲットに関連して考えるとよくわかります。

モデル	関連付け	説明
直接	通常はビューに関連付けられる	ビューは、既存の顧客テーブルとの統合を容易にするのに最も役立つ抽象化メカニズムを提供します。
間接	通常はテーブルに関連付けられる	顧客テーブルは、ドライバによって要求される構造とはおそらく一致しません。このため、通常は、ドライバが要求する構造と一致する中間ステージングテーブルを作成する必要があります。構造が一致する可能性もありますが、きわめてまれです。

次の節では、直接および間接同期が購読者および発行者チャネルの両方でどのように機能するかを説明します。

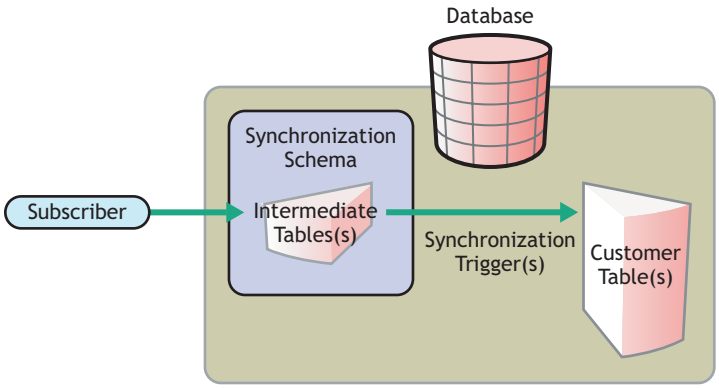
1.5.1 間接同期

間接同期は、中間ステージングテーブルを使用して、アイデンティティボールトとデータベース間のデータを同期します。

次の図では、間接同期が購読者および発行者チャネルの両方でどのように機能するかを示します。次のシナリオでは、1 つまたは複数の顧客テーブルおよび中間ステージングテーブルを保持できます。

購読者チャネル

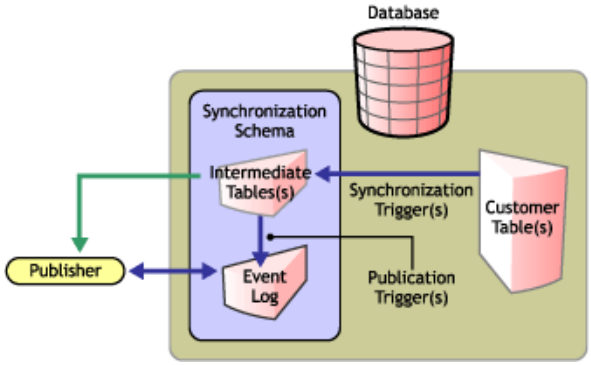
図 *1-2* 購読者チャネルでの間接同期



購読者チャネルは、同期スキーマの中間ステージングテーブルを更新します。同期トリガは、次に、データベース内のその他の場所の顧客テーブルを更新します。

発行者チャネル

図 *1-3* 発行者チャネルでの間接同期



顧客テーブルが更新されると、同期トリガは、中間ステージングテーブルを更新します。次に、発行者トリガが 1 つまたは複数の行をイベントログテーブルに挿入します。発行者チャネルは、挿入された行を読み取ってアイデンティティボールトを更新します。

イベントログテーブルから読み取られる行の内容によっては、アイデンティティボールトを更新する前に、発行者チャネルで、中間テーブルから追加情報を取得することが必要になる場合があります。アイデンティティボールトを更新した後、発行者チャネルは、行を削除するか、処理済みのマークを付けます。

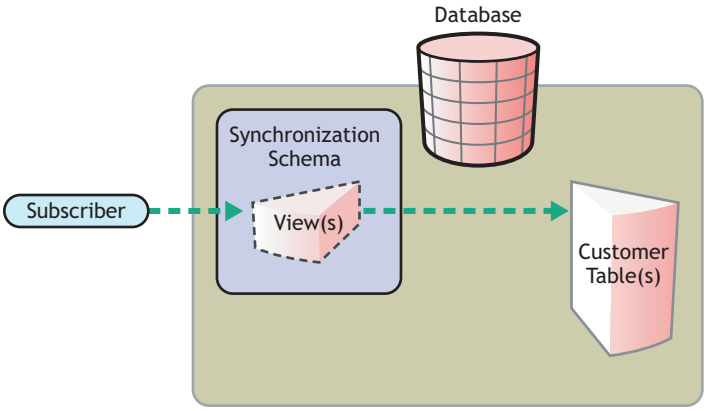
1.5.2 直接同期

直接同期は、一般に、ビューを使用して、Identity Manager とデータベース間のデータを同期します。JDBC 用ドライバが要求する構造と一致しているテーブルであれば、それを使用できます。

次の図では、直接同期が購読者および発行者チャネルの両方でどのように機能するかを示します。次のシナリオでは、1 つまたは複数の顧客ビューまたはテーブルを保持できます。

購読者チャネル

図 *1-4* 購読者チャネルでの直接同期

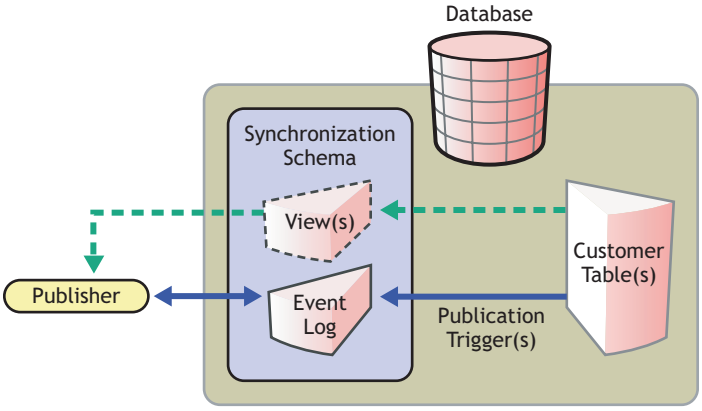


購読者チャネルは、同期スキーマのビューを通じて既存の顧客テーブルを更新します。

注 **:** ビューを使わない直接同期は、顧客テーブルが JDBC 用ドライバで要求される構造と一致する場合のみ可能です。詳細については、89 ページの 「間接同期」 を参照してください。

発行者チャネル

図 *1-5* 発行者チャネルでの直接同期



顧客テーブルが更新されると、発行者トリガは、1 つまたは複数の行をイベントログテーブルに挿入します。発行者チャネルは、挿入された行を読み取ってアイデンティティボールトを更新します。

イベントログテーブルから読み取られる行の内容によっては、アイデンティティボールトを更新する前に、発行者チャネルで、ビューから追加情報を取得することが必要になる場

合があります。アイデンティティボールトを更新した後、発行者チャネルは、行を削除するか、処理済みのマークを付けます。

1.6 トリガなしとトリガ付きの発行

トリガは、発行イベントのログに必要ではなくなりました。細かいイベントのキャプチャにトリガを使用できない場合、発行者チャネルはデータベースデータを検査することによってデータベースの変更を派生できます。

トリガなし発行は、サポート契約によって、データベースアプリケーションテーブルでのトリガ使用が許可されていない場合、またはすばやいプロトタイピングに特に便利です。

トリガなし発行は、トリガ付き発行よりも効率はよくありません。トリガ付き発行では、変更内容がすでにわかっています。トリガなし発行では、イベントを処理する前に変更の計算が必要です。

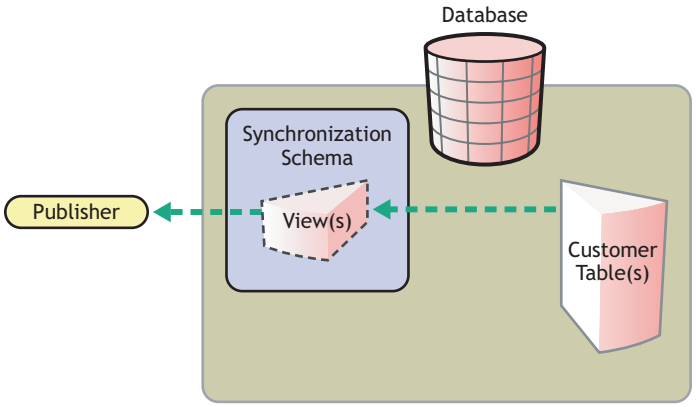
トリガなし発行は、トリガ付き発行と違って、イベントの順序を保存しません。ただ、ポーリングサイクルの最後までに、データベースおよびアイデンティティボールト内のオブジェクトが同期されることだけを保証します。

トリガなし発行は、トリガ付き発行とは異なり、古い値などの履歴データを提供しません。オブジェクトの以前の状態ではなく、現在の状態についての情報を提供します。

トリガなし発行の利点は、データベース側の依存を減らすので、ずっと簡単であることです。データベーストリガの書き込みは、複雑になる場合があり、データベース固有の SQL 構文についての幅広い知識を必要とします。

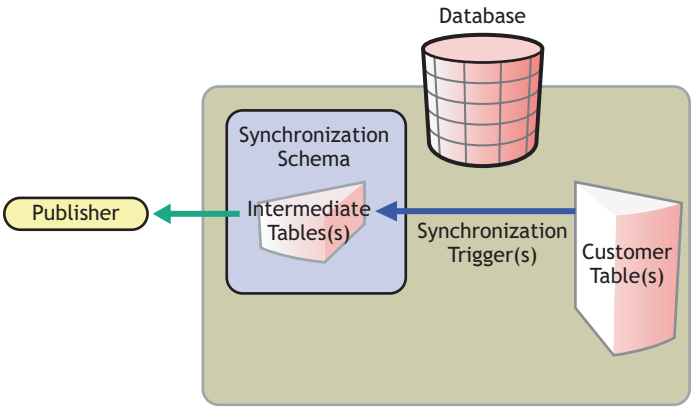
次の図は、トリガなしの直接発行を示します。

図 *1-6* トリガなしの直接同期



次の図は、トリガなしの間接発行を示します。

図 *1-7* トリガなしの間接同期



ドライバのインストールの準備 2

2.1 ドライバの前提条件

JDBC 用 Identity Manager ドライバには、以下が必要です。

- ❑ Novell® iManager 2.5 以降をサーバにインストールしている
- ❑ Novell Identity Manager 3 をサーバにインストールしている
- ❑ Java Virtual Machine (JVM*) 1.4 以降
- ❑ サポートされているサードパーティ製の JDBC ドライバ

2.2 サポートされているプラットフォーム、データベース、およびドライバ

ドライバは、Windows* NT*/2000、NetWare®、Solaris*、Linux* および AIX* を含むすべての Identity Manager 対応プラットフォームで動作します。

サポートされているデータベースについては、141 ページの 「データベースの相互運用性」を参照してください。

サポートされているサードパーティ製の JDBC ドライバについては、123 ページの 「サードパーティ製の JDBC ドライバの相互運用性」を参照してください。

2.3 既知の問題

- アイデンティティボールトの時間およびタイムスタンプの構文は、そのデータベースの時間およびタイムスタンプの構文の範囲と精度を表現するのには不適切です。データベースの時間に関連するタイプは、一般に、範囲が広く、精度が高い (通常はナノ秒) ので、発行で問題になります。逆の場合は問題ありません。詳細については、49 ページの 「時間構文」を参照してください。
- JDBC 用ドライバは、独自仕様のデータベースタイムスタンプ形式を解析できません。Sybase* や DB2* などのデータベースには、java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html) クラスで解析できない独自のタイムスタンプ形式があります。これらのデータベースからタイムスタンプ列を同期する場合、JDBC 用ドライバは、デフォルトで、イベントログテーブルに配置されているタイムスタンプ値が ODBC 標準形式 (yyyy-mm-dd hh:mm:ss.fffffffff) であることを想定しています。JDBC 用ドライバで独自仕様のデータベースタイムスタンプ形式を処理できるようにするために、カスタムの DBTimestampTranslator クラスを実装することをお勧めします。このインタフェースについては、JDBC 用ドライバに付属する Javadoc ツールを参照してください。この方法を使用して、タイムスタンプをイベントログテーブルに

挿入したりスタイルシートで再フォーマットしたりする前にデータベースでタイムスタンプを再フォーマットする問題を回避します。JDBC 用ドライバには、ネイティブの DB2 タイムスタンプ形式および Sybase スタイルの 109 タイムスタンプ形式のデフォルト実装が含まれています。

- データベースサーバに対して実行されるステートメントが、無期限にブロックされることがあります。

  一般に、ブロックは、データベースリソースが排他的にロックされていることが原因で発生します。ロックメカニズムおよび SQL のロックは、データベースによってさまざまなので、この問題の一般的な解決方法として、カスタムの DBLockStatementGenerator クラスを実装します。詳細については、63 ページの 「ステートメントジェネレータクラスのロック」を参照してください。JDBC 用ドライバには、Oracle 用のデフォルト実装が含まれています。

  ブロックの発生には、多くの要因が考えられます。ブロックが発生する可能性を低減するために、「トランザクション分離レベル」パラメータを read committed より高いレベルに設定しないことをお勧めします。

  JDBC インタフェースは、指定した秒数が経過した後にステートメントがタイムアウトできるようにするメソッド java.sql.Statement.setQueryTimeout(int):void (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html) を定義します。残念ながら、サードパーティ製の JDBC ドライバ間では、このメソッドが実装されていなかったりバグがあったりとさまざまです。したがって、このメソッドを一般的な解決策とするのは不適切と判断されました。

2.4 制限

- DBC 用ドライバは、区切りられた (引用符で囲まれた) データベース識別子 (たとえば、" 空白を含む名前 ") の使用をサポートしていません。
- CLOBやBLOBなどのLarge Objectデータタイプ(LOB)以外のJDBC 2データタイプはサポートされていません。
- JDBC 3 データタイプはサポートされていません。
- PostgreSQL は <check-object-password> イベントをサポートしていません。認証は、pg_hba.conf ファイルにエントリを手動で挿入することによって制御されます。

JDBC 用ドライバのインストールまたはアップグレード

3

ドライバのアンインストールについては、157 ページの第 9 章「JDBC 用 IDM ドライバのアンインストール」を参照してください。

重要 : ドライバ環境設定およびデータベーススクリプトのインストールまたはアンインストールは、まとめて行うことをお勧めします。データベーススクリプトおよびドライバ環境設定には、誤ってこれらが一致しなくなるのを防ぐために、バージョン番号、ターゲットデータベース名、データベースバージョンのヘッダが含まれています。

3.1 Identity Manager 3 へのアップグレード

JDBC 用 Identity Manager ドライバは、Identity Manager 3.0 より前の Identity Manager では動作しません。JDBC 2.1 用ドライバを使用するには、Identity Manager 3 にアップグレードする必要があります。

JDBC2.0 用 Identity Manager ドライバは、Identity Manager 2 で動作します。

Identity Manager のインストール中に、メタディレクトリエンジンをインストールすると同時に、JDBC 用ドライバを (他の Identity Manager ドライバと共に) インストールできます。『*Identity Manager 3.0 インストールガイド*』を参照してください。DirXML 1.1a または Identity Manager 2 から Identity Manager 3 にアップグレードできます。

3.2 jar ファイルの配置

この節の表は、JDBC ドライバ jar ファイルを Identity Manager またはリモートローダサーバに配置するパスを、デフォルトのインストールパスを基準にして示します。

3.2.1 Identity Manager のファイルパス

次の表は、Identity Management サーバでの JDBC ドライバ jar ファイルの配置場所をプラットフォーム別に示します。

表 3-1 *jar* ファイルの場所 : *Identity Manager* サーバ

プラットフォーム	ディレクトリパス
NetWare®	sys:\system\lib
Solaris、Linux、または AIX	/usr/lib/dirxml/classes (Directory 8.8 より前) /opt/novell/eDirectory/lib/dirxml/classes (eDirectory 8.8)
Windows NT/2000	novell\NDS\lib

3.2.2 リモートローダのファイルパス

次の表は、リモートローダサーバでの JDBC ドライバ jar ファイルの配置場所をプラットフォーム別に示します。

表 3-2 *jar* ファイルの場所 : リモートローダ

プラットフォーム	ディレクトリパス
Solaris、Linux、または AIX	/usr/lib/dirxml/classes (Directory 8.8 より前) /opt/novell/eDirectory/lib/dirxml/classes (eDirectory 8.8)
Windows NT/2000	novell\RemoteLoader\lib

3.3 JDBC 用ドライバのインストール

3.3.1 ドライバのインストール

メタディレクトリエンジンをインストールすると同時に、JDBC 用ドライバを (他の Identity Manager ドライバと共に) インストールできます。『*Identity Manager 3.0 インストールガイド*』を参照してください。

メタディレクトリエンジンをインストールした後で、ドライバを個別にインストールすることもできます。

Windows へのインストール

1 Identity Manager 3 のダウンロードイメージまたは CD からインストールプログラム (\nt\install.exe) を実行します。

ダウンロードイメージは、ノベル用ダウンロード (http://download.novell.com/index.jsp) から入手できます。

2 ［ようこそ］ダイアログボックスで、［次へ］をクリックして、使用許諾契約に同意します。

3 最初の［Identity Manager の概要］ダイアログボックスで、情報を確認して、［次へ］をクリックします。

このダイアログボックスには、次の情報が表示されます。

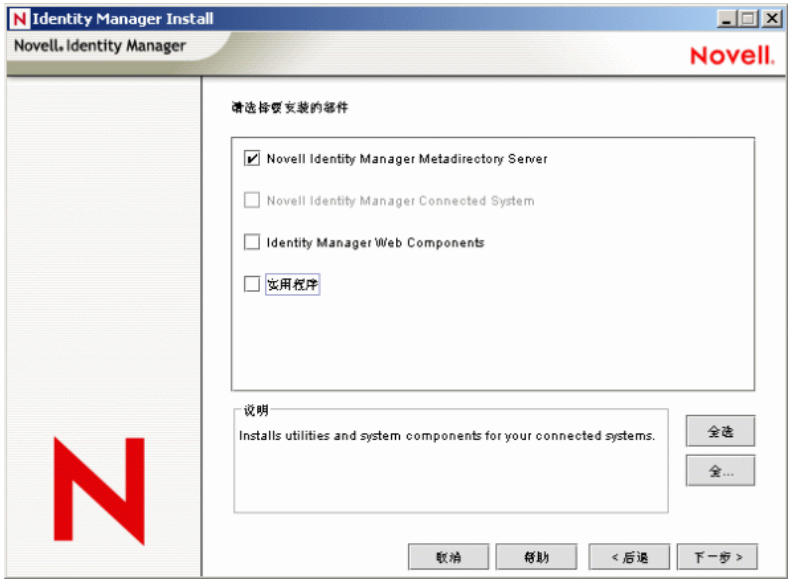
- メタディレクトリサーバ
- 接続先サーバシステム

4 2 番目の［Identity Manager の概要］ダイアログボックスで、情報を確認して、［次へ］をクリックします。

このダイアログボックスには、次の情報が表示されます。

- Web ベースの管理サーバ
- Identity Manager ユーティリティ

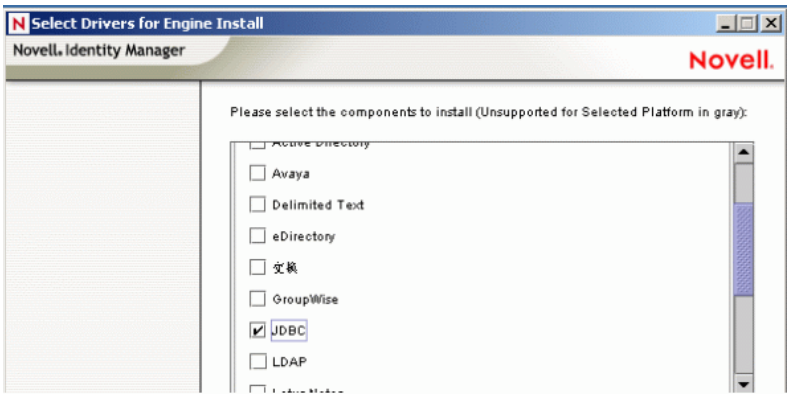
5 ローカルにインストールしている場合は、［メタディレクトリサーバ］だけを選択して、［次へ］をクリックします。



リモート (リモートローダ) インストールしている場合は、［接続システム］を選択し、『*Novell Identity Manager 3.0* 管理ガイド』の「リモートローダの設定」および「接続システムの設定」を参照してください。

リモートローダをインストールしている場合、ポリシー (およびポリシーが参照するバイナリ) はローカルで実行されますが、ドライバシムバイナリはリモートで実行されます。メタディレクトリサーバをインストールする場合は、すべてのバイナリおよびポリシーがローカルで実行されます。

6 エンジンインストールのドライバを選択するダイアログボックスで、［JDBC］だけを選択し、［次へ］をクリックします。



7 ［Identity Manager アップグレードの警告］ダイアログボックスで、［OK］をクリックします。

8 ［概要］ダイアログボックスで、選択したオプションを確認して、［終了］をクリックします。

9 ［インストールが完了しました］ダイアログボックスで、［閉じる］をクリックします。

インストール後、31 ページの 「サンプル環境設定ファイルのインポート」にある説明に従ってドライバを設定します。

NetWare へのインストール

1 NetWare® サーバで、Identity Manager CD をドライブに挿入して、CD をボリュームとしてマウントします。

CD をお持ちでない場合は、Identity_Manager_3_NW_Win.iso をダウンロードして CD を作成します。ダウンロードイメージは、ノベル用ダウンロード (http://download.novell.com/index.jsp) から入手できます。

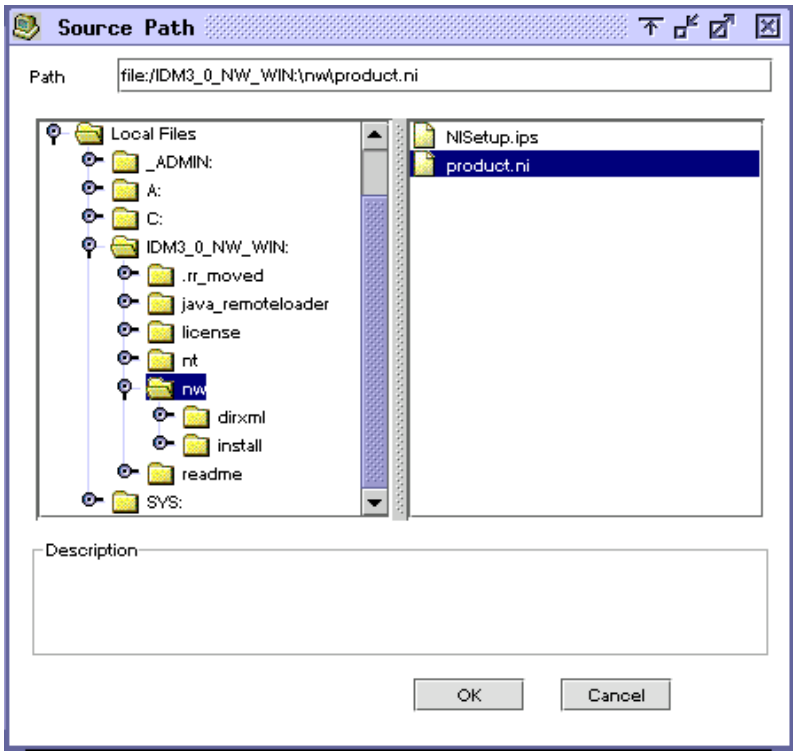
CD をマウントするには、「m cdrom」と入力します。

2 (条件付き) グラフィカルユーティリティがロードされない場合は、「startx」と入力してユーティリティをロードします。

3 グラフィカルユーティリティで、［Novell］アイコンをクリックして、［インストール］をクリックします。

4 ［インストール済みの製品］ダイアログボックスで、［追加］をクリックします。

5［ソースパス］ダイアログボックスで、product.ni ファイルを参照して選択します。



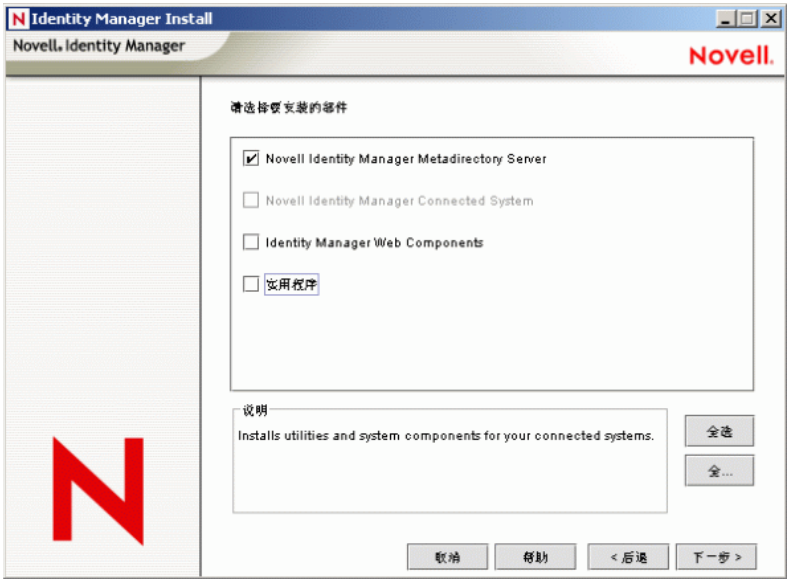
5a 以前にマウントした CD ボリューム (IDM_3_0_NW_WIN) を参照して展開します。

5b nw ディレクトリを展開し、product.ni を選択して、［OK］を 2 回クリックします。

6［Novell Identity Manager 3.0 のインストールへようこそ］ダイアログボックスで、［次へ］をクリックして、使用許諾契約に同意します。

7 2 つの［概要］ダイアログボックスを表示して、［次へ］をクリックします。

8 [Identity Manager のインストール] ダイアログボックスで、[メタディレクトリサーバ] だけを選択します。

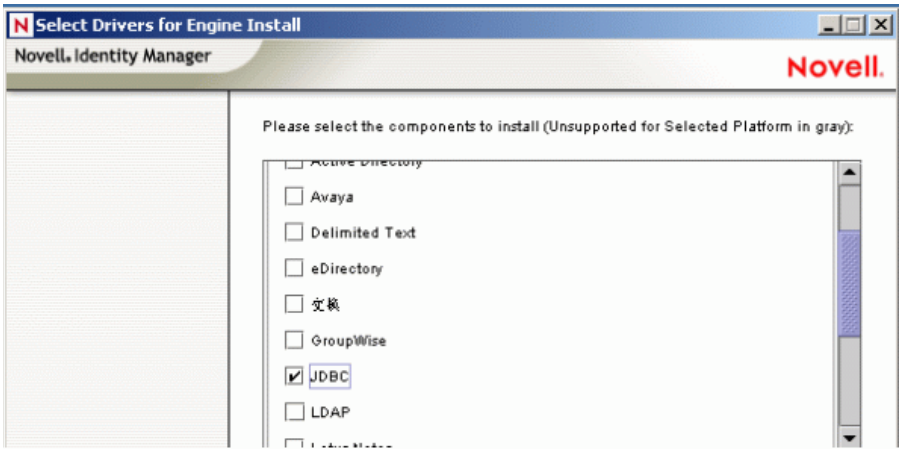


次の項目を選択解除します。

- Identity Manager Web コンポーネント
- ユーティリティ

9 [次へ] をクリックします。

10 エンジンインストールのドライバを選択するダイアログボックスで、[JDBC] だけを選択します。



次の項目を選択解除します。

- メタディレクトリエンジン
- 区切りテキスト以外のすべてのドライバ

11 [次へ] をクリックします。

12 [Identity Manager アップグレードの警告] ダイアログボックスで、[OK] をクリックします。

このダイアログボックスで、90 日以内にドライバのライセンスを有効にすることを促されます。

13 ［概要］ページで、選択したオプションを確認して、［終了］をクリックします。

14 ［閉じる］をクリックします。

インストールの後、以下を行います。

- サンプル環境設定ファイルのインポート。31 ページの 「サンプル環境設定ファイルのインポート」を参照してください。
- リモートローダの設定 (オプション)。33 ページの 「リモートローダの設定」を参照してください。
- データベースオブジェクトの設定。34 ページの 「データベースオブジェクトのインストールおよび設定」を参照してください。

Linux または Solaris へのインストール

デフォルトでは、JDBC 用 Identity Manager ドライバは、メタディレクトリエンジンをインストールするときにインストールされます。この節には、メタディレクトリエンジンのインストール時にドライバがインストールされなかった場合のドライバのインストール方法が記載されています。

インストールプログラムを進めていくときに、「previous」と入力すれば前のセクション (画面) に戻ることができます。

1 端末セッションで、root としてログインします。

2 Identity Manager CD をドライブに挿入してマウントします。

CD をお持ちでない場合は、Identity_Manager_3_Linux.iso をダウンロードして CD を作成します。ダウンロードイメージは、ノベル用ダウンロード (http://download.novell.com/index.jsp) から入手できます。

通常、CD は自動的にマウントされます。次の表は、CD を手動でマウントする例を示しています。

プラットフォーム	入力するコマンド
AIX	mount /mnt/cdrom、次に <Enter> キーを押す
Red Hat*	mount /mnt/cdrom、次に <Enter> キーを押す
Solaris	mount /cdrom、次に <Enter> キーを押す
SUSE®	mount /media /cdrom、次に <Enter> キーを押す

3 setup ディレクトリに移動します。

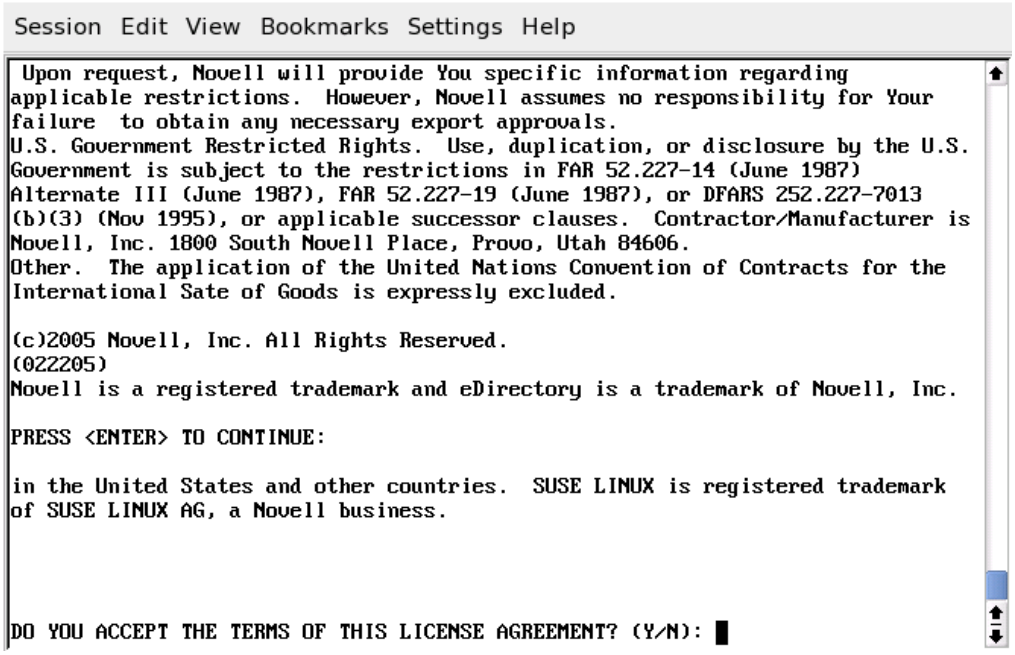
プラットフォーム	パス
AIX	/mnt/cdrom/setup/
Red Hat	/mnt/cdrom//setup/
Solaris	/cdrom//idm_3/setup/
SUSE	/media/cdrom//setup/

4 「./dirxml_linux.bin」と入力してインストールプログラムを実行します。

5 ［イントロダクション］セクションで、<Enter> キーを押します。

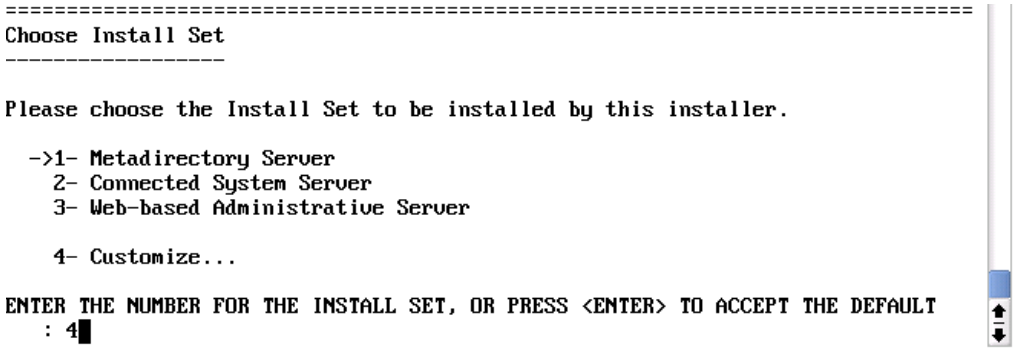
6 使用許諾契約に同意します。

［DO YOU ACCEPT THE TERMS OF THIS LICENSE AGREEMENT (この使用許諾契約書の条項に同意しますか ?)］が表示されるまで <Enter> キーを押し、「y」と入力して、<Enter> キーを押します。



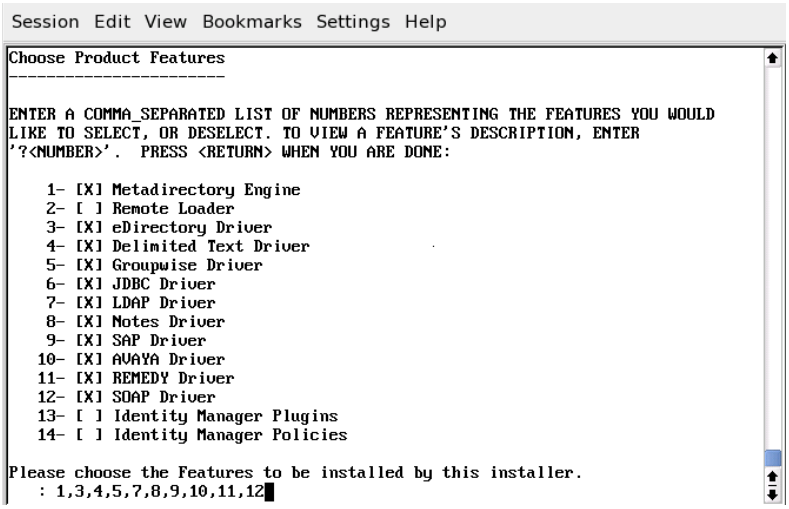
7 ［インストールセットの選択］セクションで、［カスタマイズ］オプションを選択します。

「4」と入力して、<Enter> キーを押します。

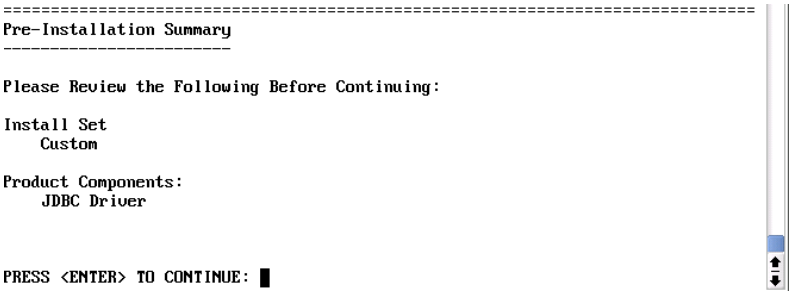


8 ［Choose Product Features (製品の機能の選択)］セクションで、［JDBC］を除くすべての機能を選択解除して、<Enter> キーを押します。

機能を選択解除するには、その番号を入力します。複数の機能を選択解除するには、各機能の間にカンマを入力します。



9 ［インストール前の概要］セクションで、オプションを確認します。



前のセクションに戻るには、「previous」と入力して、<Enter> キーを押します。

続行するには、<Enter> キーを押します。

10 インストールが完了したら、<Enter> キーを押してインストールを終了します。

インストールの後は、ドライバを設定します。43 ページの第 4 章「JDBC 用 Identity Manager ドライバの設定」を参照してください。

3.3.2 サンプル環境設定ファイルのインポート

JDBC 用 Identity Manager ドライバを設定するには、ドライバ環境設定ファイルをインポートし、データベースを設定します。データベース環境設定は、SQL スクリプトの実行で構成されます。ドライバを開始する前に、データベース SQL スクリプトを実行してテストすることをお勧めします。

付属している環境設定は、サンプル用です。この環境設定をカスタマイズする前に、まずテスト環境にインストールすることをお勧めします。

サンプルのドライバ環境設定ファイルのインポート : iManager

JDBCv2.xml 環境設定ファイルを使用して、サンプルドライバの適切な動作に必要な Identity Manager オブジェクトを作成および設定します。環境設定ファイルには、カスタマイズ可能なサンプルポリシーも含まれています。

1. iManager で、［Identity Manager ユーティリティ］>［新規ドライバ］の順に選択します。
2. ドライバセットを選択し、［次へ］をクリックします。

   このドライバを新しいドライバセットに配置する場合は、ドライバセット名、コンテキスト、および関連サーバを指定します。
3. ［サーバからのドライバ環境設定のインポート (.XML ファイル)］を選択します。

   ドライバ環境設定ファイルは、iManager の設定時に Web サーバにインストールされます。
4. ドロップダウンリストから、［JDBC v 2.xml］オプションを選択して、［次へ］をクリックします。
5. ドライバ名を入力するように要求されたら、ドライバの名前 (JDBC 2 など) を指定し、［次へ］をクリックします。
6. ターゲットデータベースを選択し、ドライバがローカルかリモートかを選択し、［次へ］をクリックします。
7. 同期モデルを選択し、サードパーティ製の JDBC 実装を選択し、［次へ］をクリックします。
8. データフロー (たとえば双方向) を選択し、データベースホストの IP アドレスを指定します。さらに、ポート番号を入力し、［次へ］をクリックします。
9. ユーザコンテナの DN、グループコンテナの DN、および発行モードを指定し、［次へ］をクリックします。
10. (オプション)［Define Security Equivalences (「同等セキュリティ」の定義)］をクリックします。

    10a［追加］をクリックし、管理者権 (またはドライバに持たせたいその他の権利) を持つオブジェクトを選択します。

    10b［適用］をクリックし、［OK］をクリックします。
11. (オプション) オブジェクトをレプリケーションの対象から除外するには、［Exclude Administrative Roles (「管理の役割」の除外)］をクリックします。

    11a［追加］をクリックし、除外するユーザ (管理者ユーザなど) を選択します。

    11b［適用］をクリックし、［OK］をクリックします。
12. インポートの概要を表示するには、［次へ］をクリックします。
13. 環境設定が正しいことを確認し、［概要の終了］をクリックします。

インストールによって、必要な Identity Manager ドライバオブジェクトが作成されました。インポート中に同等セキュリティの定義や管理ユーザの除外を行わなかった場合は、ドライバオブジェクトのプロパティを変更することによってこのタスクを実行できます。

環境設定ファイルの規則

- データベースユーザ名は、ユーザの名字と、対応する数値のプライマリキー値を連結したものです。たとえば、John Doe' のユーザ名は Doe1 になります。

- 初期パスワードは、ユーザの名字です。たとえば、John Doe' のパスワードは Doe になります。Sybase のパスワードは、少なくとも 6 文字必要です。6 文字より短い場合は、名字が「p」でパディングされます。たとえば、John Doe のパスワードは、Doeppp になります。パディングされる文字は、購読者コマンド変換ポリシーで変更できます。

ドライバ環境設定ファイルのインポート : Designer

JDBC の基本的なドライバ環境設定ファイルをインポートするには、Designer for Identity Manager を使用できます。この基本的なファイルを使用して、ドライバを正しく機能させるために必要なオブジェクトやポリシーを作成および設定します。

次の手順は、サンプル環境設定ファイルをインポートする方法の 1 つを示しています。

1. Designer でプロジェクトを開きます。
2. モデラーで、［ドライバセット］オブジェクトを右クリックして、［Add Connected Application (接続アプリケーションの追加)］を選択します。
3. ドロップダウンリストから、［JBDC.xml］を選択して、［実行］をクリックします。
4. ［Perform Prompt Validation (プロンプト検証の実行)］ウィンドウで、［はい］をクリックします。
5. フィールドに入力してドライバを設定します。

   各自の環境に特有の情報を指定します。設定の詳細については、45 ページの 「環境設定パラメータ」を参照してください。
6. パラメータを指定したら、［OK］をクリックしてドライバをインポートします。
7. ドライバをカスタマイズおよびテストします。
8. アイデンティティボールトにドライバを展開します。

   『*Designer for Identity Manager 3: Administration Guide*』の「Deploying a Project to an Identity Vault」を参照してください。

3.3.3 リモートローダの設定

リモートローダの使用はオプションです。接続されたシステムで JDBC ドライバを実行する場合以外は必要ありません。

1. リモートローダをまだインストールしていない場合はインストールします。

   『*Novell Identity Manager 3.0* 管理ガイド』の「接続システムの設定」を参照してください。
2. 適切なサードパーティ製の JDBC ドライバ jar ファイルをリモートローダサーバにコピーします。

   2a サードパーティ製の JDBC ドライバのファイル名および入手先については、125 ページの 「サポートされているサードパーティ製の JDBC ドライバ」を参照してください。

   2b ファイルのインストールパスについては、23 ページの 「jar ファイルの配置」を参照してください。
3. リモートドライバを設定します。

［リモートドライバ環境設定］パラメータで、［ドライバ］パラメータを以下に設定します。

```
com.novell.nds.dirxml.driver.jdbc.JDBCDriverShim
```

4 他のリモートローダパラメータを設定します。『*Novell Identity Manager 3.0* 管理ガイド』の「接続システムの設定」を参照してください。

3.3.4 データベースオブジェクトのインストールおよび設定

サンプルのドライバ環境設定と同期するために、データベースオブジェクト (テーブル、トリガ、インデックスなど) をインストールし、設定します。データベースオブジェクトを設定しない場合、サンプル環境設定ファイルは動作しません。

SQL スクリプトの規則

SQL スクリプトは、*install-dir*\jdbc\sql\a*bbreviated-database-name* ディレクトリにあります。

すべての SQL スクリプトは、データベースに関係なく同じ規則を使用します。

DB2 識別子の最大サイズは 18 文字です。この最小公分母の長さは、すべての SQL スクリプトでのデータベース識別子の長さの上限を定義します。この長さの制限があるので、省略形が使用されます。次の表は、識別子の省略形と意味の概要を示します。

表 *3-3* 識別子の省略形と意味

省略形	解釈
proc_[1]	ストアドプロシージャ / 関数
idx_	インデックス
trg_	トリガ
_i	挿入時トリガ
_u	更新時トリガ
_d	削除時トリガ
chk_	チェック制約
pk_	ビューのプライマリキー制約
fk_	ビューの外部キー制約
mv_	ビューの複数値列
sv_	ビューの単一値列 (暗黙のデフォルト)

1 より一般的な省略形は sp_ ですが、このプリフィックスは、Microsoft* SQL Server のシステムのストアドプロシージャ用に予約されています。また、このプリフィックスにより、任意の識別子 (たとえば、データベースまたは所有者) を評価する前に、プロシージャが強制的にマスタデータベースで最初に検索されます。プロシージャ検索の効率を最大限にするために、このプリフィックスが意図的に避けられています。

次の表は、インデックス、トリガ、ストアドプロシージャ、関数、および制約の識別子の命名規則を示します。

表 *3-4* 識別子の命名規則

データベースオブジェクト	命名規則	例
ストアドプロシージャ / 関数	proc_*procedure-or-function-name*	proc_idu
インデックス	idx_*unqualified-table-name_sequence-number*	idx_indirectlog_1
トリガ	tgr_*unqualified-table-name_triggering-statement-type_sequence-number*	tgr_usr_i_1
プライマリキー制約	pk_*unqualified-table-name_column-name*	pk_usr_idu
外部キー制約	fk_*unqualified-table-name_column-name*	fk_usr_idu
チェック制約	chk_*unqualified-table-name_column-name*	chk_usr_idu

その他の規則

- すべてのデータベース識別子は小文字にします。

  これは、データベース間で最もよく使用される大文字小文字の規則です。
- 文字列フィールドの長さは 64 文字にします。

  ほとんどの eDirectory™ 属性値はこの長さのフィールドに収まります。フィールドの長さを調整すると、ストレージ効率が向上します。
- パフォーマンスの理由から、プライマリキー値には、可能な限り、ネイティブのスカラ数値タイプを使用します (NUMERIC ではなく BIGINT など)。
- イベントログテーブルの record_id 列は、オーバーフローを避けるために、各データベースで許容されている最大の数値精度を保持します。
- 識別列およびシーケンスオブジェクトは値をキャッシュしません。一部のデータベースでは、ロールバックが発生するときにキャッシュされた値が捨てられます。これによって、識別列またはシーケンス値で大きな差が出ることがあります。

IBM DB2 Universal Database (UDB) のインストール

重要 **:** IBM* DB2 の場合は、提供されている SQL スクリプトを実行する前に、手動でオペレーティングシステムのユーザアカウントを作成する必要があります。

ユーザアカウントを作成する処理はオペレーティングシステムごとに異なるので、次の手順 1 は OS ごとに異なります。次の手順は、Windows NT オペレーティング環境の場合です。SQL スクリプトを再実行する場合は、手順 2 ～ 5 だけを繰り返します。

DB2 のディレクトリコンテキストは、*install-dir*\jdbc\sql\db2_udb\install です。

1 ユーザ idm、indirect、および direct のユーザアカウントを作成します。

［User Manager for Domains (ドメインのユーザマネージャ)］で、パスワードとして novell を使用します。

このアカウントの［User Must Change Password at Next Login (次回のログインでパスワードを変更する必要があります)］の選択を解除します。

［Password Never Expires (無期限パスワード)］も選択できます。

注 **:** 残りの手順は OS には依存しません。

---

2 1_install.sql スクリプトで idm_db2.jar へのファイルパスを修正します。idm_db2.jar へのファイルパスは、クライアントマシンでこのファイルがある場所を反映する必要があります。

3 コマンドラインプロセッサ (CLP) から 1_install.sql スクリプトを実行します。

次に例を示します。db2 -f 2_install_8.sql

重要 **:** スクリプトは、バージョン 7 より後の Command Center インタフェースでは実行されません。スクリプトでは、行継続文字「\」が使用されていますが、後のバージョンの Command Center では、この文字は認識されません。

---

4 バージョン 8 またはそれ以降では、2_install_8.sql スクリプトを実行します。

次に例を示します。db2 -f 2_install_8.sql

Informix Dynamic Server (IDS) のインストール

重要 **:** Informix* Dynamic Server の場合は、提供されている SQL スクリプトを実行する前に、手動でオペレーティングシステムのユーザアカウントを作成する必要があります。

ユーザアカウントを作成する処理はオペレーティングシステムごとに異なるので、次の手順 1 は OS ごとに異なります。次の手順は、Windows NT オペレーティング環境の場合です。SQL スクリプトを再実行する場合は、手順 2 ～ 4 だけを繰り返します。

Informix SQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\informix_ids\install です。

1 Windows NT では、ユーザ idm のユーザアカウントを作成します。

［User Manager for Domains (ドメインのユーザマネージャ)］で、パスワードとして novell を使用します。

このアカウントの［User Must Change Password at Next Login (次回のログインでパスワードを変更する必要があります)］の選択を解除します。

［Password Never Expires (無期限パスワード)］も選択できます。

注 **:** 残りの手順は OS には依存しません。

---

2 SQL エディタなどのクライアントを起動します。

3 サーバに、informix ユーザ、または DBA (データベース管理者) 特権を持つ別のユーザとしてログインします。

デフォルトで、informix ユーザのパスワードは、informix になります。

注 **:** informix 以外のユーザとしてスクリプトを実行する場合は、実行する前にスクリプト内の informix への参照をすべて変更します。

4 作成するデータベースのタイプに応じて、ansi (トランザクション、ANSI 互換)、log (トランザクション、ANSI 非準拠)、または no_log (非トランザクション、ANSI 非準拠) サブディレクトリから 1_install.sql を開いて実行します。

Microsoft SQL Server のインストール

Microsoft SQL Server スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\mssql\install です。

1 Query Analyzer などのクライアントを開始します。

2 データベースサーバ sa ユーザとしてログインします。

デフォルトで、sa ユーザにパスワードはありません。

3 インストールスクリプトを実行します。

バージョン 7 の場合は、1_install_7.sql を実行します。

バージョン 8 (2000) の場合は、1_install_2k.sql を実行します。

注 **:** Query Analyzer の実行のホットキーは <F5> です。

MySQL のインストール

MySQL* SQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\mysql\install です。

1 mysql などの MySQL クライアントから、root ユーザ、または管理特権を持つ他のユーザとしてログインします。

たとえば、コマンドラインから以下を実行します。

mysql -u root -p

デフォルトで、root ユーザにパスワードはありません。

2 使用するテーブルタイプに応じて、インストールスクリプト 1_install_innodb.sql または 1_install_myisam.sql を実行します。

次に例を示します。mysql> \. c:\1_install_innodb.sql

ヒント **:** このステートメントの最後にセミコロンを使用しないでください。

Oracle のインストール

Oracle SQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\oracle\install です。

1 SQL Plus などの Oracle クライアントから、SYSTEM ユーザとしてログインします。

デフォルトで、SYSTEM のパスワードは MANAGER です。

注 **:** パスワードが MANAGER の SYSTEM 以外のユーザとしてスクリプトを実行する場合は、実行する前にスクリプト内の SYSTEM への参照をすべて変更します。

2 インストールスクリプト 1_install.sql を実行します。

次に例を示します。SQL> @c:\1_install.sql

PostgreSQL のインストール

PostgreSQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\postgres\install です。Postgres コマンドの実行のディレクトリコンテキストは、postgres-install-dir/pgsql/bin です。

1 データベース idm を作成します。

たとえば、UNIX* コマンドラインから、createdb コマンドを実行します。./createdb idm

2 plpgsql 手続き型言語をデータベース idm にインストールします。

たとえば、UNIX コマンドラインから、createlang コマンドを実行します。./createlang plpgsql idm

3 psql などの Postgres クライアントから、ユーザ postgres として idm データベースにログオンします。

たとえば、UNIX コマンドラインから、psql コマンドを実行します。./psql -d idm postgres

デフォルトで、Postgres ユーザにパスワードはありません。

4 psql 内から、スクリプト 1_install.sql を実行します。

次に例を示します。idm=# \i 1_install.sql

5 pg_hba.conf ファイルを更新します。

たとえば、idm データベースユーザのエントリを追加します。IP-ADDRESS および IP-MASK は必要に応じて変更します。

```
# TYPE  DATABASE    USER   IP-ADDRESS        IP-MASK
METHOD# allow driver user idm to connect to database idm host
idm         idm    255.255.255.255   255.255.255.0     password
```

6 Postgres サーバを再起動して、pg_hba.conf ファイルに対する変更を有効にします。

Sybase Adaptive Server Enterprise (ASE) のインストール

重要 **:** JDBC メタデータサポートをデータベースサーバにインストールしていることを確認します。これは、通常、バージョン 12.5 より前のバージョンでのみ問題になります。

Sybase SQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\sybase_ase\install です。

1 isql などの Sybase クライアントから、sa ユーザとしてログインし、1_install.sql インストールスクリプトを実行します。

たとえば、コマンドラインから以下を実行します。isql -U sa -P -i 1_install.sql

デフォルトで、sa アカウントにパスワードはありません。

3.3.5 テスト

各データベースのテストスクリプトは、次のディレクトリにあります。

表 3-5 データベーススクリプトの場所

データベース	テスト SQL スクリプトの場所
IBM DB2 Universal Database	*install-dir*\jdbc\sql\db2_udb\test
Informix Dynamic Server	*install-dir*\jdbc\sql\informix_ids\log\test *install-dir*\jdbc\sql\informix_ids\no_log\test Informix ANSI テストスクリプトは log\test サブディレクトリにあります。
Microsoft SQL Server	*install-dir*\jdbc\sql\mssql\test
MySQL	*install-dir*\jdbc\sql\mysql\test
Oracle	*install-dir*\jdbc\sql\oracle\test
PostgreSQL	*install-dir*\jdbc\sql\postgres\test
Sybase Adaptive Server Enterprise	*install-dir*\jdbc\sql\sybase_ase\test

サンプルドライバを開始する前に、テストスクリプトを実行することをお勧めします。

3.3.6 トラブルシューティング

- 明示的に変更をコミットしない限り、発行イベントが、発行者チャネルで認識されない場合があります。サポートされているデータベースのコミットキーワードについては、145 ページのセクション 7.3.7「コミットキーワード」を参照してください。
- テストスクリプトは、ドライバの idm データベースユーザアカウント以外のユーザによって実行する必要があります。idm ユーザとして実行した場合、イベントは、発行ループバックが許可されていない限り、ドライバの発行者チャネルから無視されます。発行ループバックの許可または禁止については、82 ページの 「ループバックを許可しますか ?」を参照してください。

3.4 JDBC 用ドライバのアップグレード

- 40 ページのセクション 3.4.1「後方非互換の変更点」

JDBC2.1 用 Identity Manager ドライバは、Identity Manager 3.0 より前の Identity Manager では動作しません。JDBC 2.0 用 Identity Manager ドライバは、Identity Manager 2.0 で動作します。

JDBC 1.5 以降用の Identity Manager ドライバから 2.1 用にアップグレードするには、JDBC 用ドライバをインストールします。このタスクでは、バイナリだけが置き換えられます。

表 *3-6* *JDBC 2.0* 用 *Identity Manager* ドライバへのアップグレード

実行中のバージョン	最初にアップグレードするバージョン	最終的にアップグレードするバージョン
JDBC1.5 より前のバージョン用ドライバ	JDBC 1.51 用ドライバ	JDBC 2.0 用ドライバ
JDBC 1.5 用ドライバまたはそれ以降	なし	JDBC 2.0 用ドライバ

表 *3-7* *JDBC 2.1* 用 *Identity Manager* ドライバへのアップグレード

実行中のバージョン	最初にアップグレードするバージョン	最終的にアップグレードするバージョン
JDBC1.5 より前のバージョン用ドライバ	JDBC 1.51 用ドライバ	JDBC 2.1 用ドライバ
JDBC 1.5 用ドライバまたはそれ以降	なし	JDBC 2.1 用ドライバ

JDBC バージョン 1.5 より前のバージョン用の Identity Manager ドライバの場合は、まずバージョン 1.5 にアップグレードする必要があります。『*DirXML Driver 1.5 for JDBC Implementation Guide* (http://www.novell.com/documentation/lg/dirxmldrivers/index.html)』を参照してください。また、必ず 2.1 Association Utility を使用してください。これはすべての旧バージョンに優先します。

3.4.1 後方非互換の変更点

- 双方向同期のために、ドライバには、少なくとも 2 つのデータベース接続が要求されるようになりました。詳細については、56 ページの 「最小数の接続を使用しますか ?」を参照してください。
- ドライバは、論理データベースクラス名 (親テーブルまたはビューの名前) のスキーマ修飾子 (使用可能な場合) を返すようになりました。クラス名がスキーママッピングポリシーで再マップされている場合を除いて、この変更は既存の環境設定には影響しません。クラス名が再マップされている場合は、既存のポリシー内のクラス名へのすべての参照をスキーマ修飾する必要があります。
- ビューを使用する既存の環境設定が少し変更されています。「Enable Meta-Identifier Support (メタ識別子のサポートを有効にしますか ?)」パラメータを論理値の False に設定してください。64 ページの 「Enable Meta-Identifier Support? (メタ識別子のサポートを有効にしますか ?)」を参照してください。
- com.novell.nds.dirxml.driver.jdbc.util.MappingPolicy クラスを参照する既存の環境設定が少し変更されています。このクラスのメソッドは、ソースドキュメントを編集しなくなりました。代わりに、ノードセットを返すので、これをターゲットドキュメントにコピーする必要があります。サンプルのドライバ環境設定ファイル JDBCv2.xml には、この実行方法の例が含まれています。
- 列位置を実装またはサポートしないDB2/AS400またはそれ以外のレガシデータベースに対して展開される既存の環境設定が少し変更されます。「列名を並べ替える基準」パラメータを追加し、設定してください。文字列照合順序別に列名を並べ替える方法については、67 ページの 「列名を並べ替える基準」を参照してください。デフォルト動作は、16 進値で列名を並べ替えるように変更されています。

3.5 ドライバの有効化

インストール後 90 日以内にドライバを有効 (アクティベーション) にします。そうしなければ、ドライバは動作しなくなります。

アクティベーションについては、『*Identity Manager 3.0* インストールガイド』の「Novell Identity Manager 製品を有効にする」を参照してください。

JDBC 用 Identity Manager ドライバの設定

4

4.1 スマート環境設定

JDBC 用 Identity Manager ドライバは、サポートされているサードパーティ製の JDBC ドライバおよびデータベースのセットを認識できます。また、ドライバ互換パラメータのほとんどを動的および自動的に設定できます。これらの機能により、エンドユーザがこれらのパラメータを理解したり明示的に設定したりすることがそれほど要求されなくなりました。

これらの機能は、次の 4 つのタイプの XML 記述子ファイルを通じて実装されます。これらは、サードパーティ製の JDBC ドライバまたはデータベースを JDBC 用ドライバに記述します。

- サードパーティ製の JDBC ドライバ
- サードパーティ製の JDBC ドライバインポート
- データベース
- データベースインポート

記述子ファイル用の予約済みファイル名

ドライバに付属する記述子ファイルの名前は、アンダースコア文字 (_) で始まります。このようなファイル名は、ドライバに付属する記述子ファイルがカスタムの記述子ファイルと競合しないように予約されています。したがって、カスタムの記述子ファイル名は、アンダースコア文字で始まらないようにします。

インポート記述子ファイル

インポート記述子ファイルを使用すると、複数の非インポート記述子ファイルで内容を共有できます。この機能により、非インポート記述子ファイルのサイズが削減され、内容の繰り返しが最小限で済み、メンテナンス性が高まります。インポートファイルを主要なタイプ間でインポートすることはできません。つまり、JDBC ドライバ記述子は、データベースインポートをインポートできず、データベース記述子は、JDBC ドライバインポートをインポートできません。

さらに、カスタムの非インポート記述子は、予約済みの記述子インポートをインポートできません。たとえば、カスタムのサードパーティ製の JDBC ドライバ記述子ファイル custom.xml が予約済みのサードパーティ製の JDBC ドライバ記述子 _reserved.xml をインポート使用とすると、エラーが発生します。これらの制限により、次の目的を果たすことができます。

- 予約済みおよびカスタムのインポートファイル間に依存関係がないことを確実にする
- 既存の予約済み記述子ファイルを、それより新しいバージョンのドライバで拡張できるようにする

記述子ファイルの場所

記述子ファイルがある場所は、名前がプリフィックス「jdbc」(大文字と小文字を区別しない) で始まり、ランタイムクラスパスにある jar ファイル内です。

次の表は、記述子 jar ファイル内で記述子が配置されている場所を示します。

表 *4-1* 記述子の場所

記述子タイプ	ディレクトリパス
サードパーティ製の JDBC ドライバ	com/novell/nds/dirxml/driver/jdbc/db/descriptor/driver
サードパーティ製の JDBC ドライバインポート	com/novell/nds/dirxml/driver/jdbc/db/descriptor/driver/import
データベース	com/novell/nds/dirxml/driver/jdbc/db/descriptor/db
データベースインポート	com/novell/nds/dirxml/driver/jdbc/db/descriptor/db/import

予約済みの記述子ファイルは、JDBCConfig.jar ファイルにあります。JDBC 用ドライバが更新されるときにこれらの予約済みファイルが上書きされないようにするには、カスタムの記述子を別の jar ファイルに配置します。

優先

iManager などの管理コンソールで明示的に指定されたパラメータは、記述子ファイルで指定されたパラメータよりも常に優先されます。記述子ファイルのパラメータは、管理コンソールでパラメータが設定されていない場合のみ有効です。

インポート不可の記述子ファイルで指定されたパラメータおよびその他の情報は、記述子インポートファイルでの指定よりも常に優先されます。記述子ファイル内でパラメータまたはその他の情報が重複している場合は、最初のインスタンスが優先されます。

インポートファイル間での優先順位は、インポート順によって決まります。インポートリストで先に記述されているインポートファイルの方が優先されます。

カスタム記述子のベストプラクティス

- ❑ カスタムの記述子ファイル名をアンダースコア文字 (_) で始めてはなりません。
- ❑ カスタムの記述子ファイルは、JDBCConfig.jar 以外の jar ファイルに配置し、ファイル名をプリフィックス「jdbc」(大文字と小文字を区別しない) で始めます。

- カスタムの記述子を、(ファイル名がアンダースコア文字で始まる) 予約済みのインポートファイルのインポートに使用してはなりません。

記述子ファイルの DTD

次の付録には、すべての記述子ファイルタイプの DTD が含まれています。これらの DTD は、カスタム記述子ファイルの作成に役立ちます。

表 *4-2* 記述子の *DTD* の場所

記述子タイプ	付録
サードパーティ製の JDBC ドライバ	177 ページの付録 F「サードパーティ製の JDBC ドライバ記述子の DTD」
サードパーティ製の JDBC ドライバインポート	179 ページの付録 G「サードパーティ製の JDBC ドライバ記述子のインポートの DTD」
データベース	181 ページの付録 H「データベース記述子の DTD」
データベースインポート	183 ページの付録 I「データベース記述子のインポートの DTD」

4.2 環境設定パラメータ

4.2.1 ドライバパラメータの表示

1. iManager で、［Identity Manager］>［Identity Manager の概要］の順にクリックします。
2. ドライバを含むドライバセットを探し、ドライバのアイコンをクリックします。
3. ［Identity Manager ドライバの概要］ページで、ドライバオブジェクトをクリックします。

   ドライバの環境設定パラメータが表示されます。

4.2.2 非推奨パラメータ

次のパラメータは、バージョン 1.6 以降では推奨されなくなりました。

表 *4-3* 非推奨パラメータ

タグ名	理由
connection-tester-class	XML 記述子ファイルの情報に基づいて、ドライバがランタイム時に動的に接続テスタクラスを作成するようになりました。このパラメータは、後方互換性を確実にするためにまだ機能しますが、継続的な使用は避けてください。

タグ名	理由
connection-test-stmt	XML 記述子ファイルの情報に基づいて、ドライバがランタイム時に動的に接続テスタクラスを作成するようになりました。このパラメータは、後方互換性を確実にするためにまだ機能しますが、継続的な使用は避けてください。
reconnect-interval	再接続間隔は、両方のチャネルで 30 秒に修正されました。

4.2.3 認証パラメータ

ドライバをインポートした後、ターゲットデータベースの認証情報を提供します。

認証 ID

認証 ID は、ドライバのデータベースユーザのログインアカウントの名前です。各データベースのインストール SQL スクリプトによって、このアカウントがサポートデータベースを認証するために必要なデータベース特権についての情報が提供されます。スクリプトは、*install-dir*\tools\sql*abbreviated-database-name*\install *install-dir*\tools\sql*abbreviated-database-name*\install ディレクトリにあります。

この値は、トークン {$username} を使用して「接続プロパティ」パラメータの値で参照できます。57 ページの 「接続プロパティ」を参照してください。

サンプルの環境設定のデフォルト値は、idm です。

認証コンテキスト

認証コンテキストは、ターゲットデータベースの JDBC URL です。

URL のフォーマットとコンテンツは各社独自であり、サードパーティ製の JDBC ドライバ間で異なります。ただし、コンテンツには類似点があります。各 URL には、そのフォーマットにかかわらず、通常、IP アドレスまたは DNS 名、ポート番号、およびデータベース識別子が含まれます。使用しているドライバでの正確な構文およびコンテンツの要件については、サードパーティ製のドライバのマニュアルを参照してください。

サポートされているサードパーティ製のドライバの JDBC URL 構文のリストについては、126 ページの 「JDBC URL の構文」を参照してください。

重要 **:** この値で、URL プロパティ以外の何らかの変更を行うと、トリガなし発行が使用されるときにすべてのオブジェクトが強制的に再同期されます。

アプリケーションパスワード

アプリケーションパスワードは、ドライバのデータベースユーザのログインアカウントのパスワードです。サンプルのドライバ環境設定のデフォルト値は、novell です。

この値は、トークン {$password} を使用して「接続プロパティ」パラメータの値で参照できます。57 ページの 「接続プロパティ」を参照してください。

4.3 ドライバパラメータ

次の表は、すべてのドライバレベルのパラメータとそのプロパティの概要を示します。

表 *4-4* ドライバパラメータおよびプロパティ

表示名	タグ名	サンプル値	デフォルト値	必須？
サードパーティ製の JDBC ドライバのクラス名	jdbc-class	oracle.jdbc.driver.OracleDriver	(なし)	○
時間構文	time-syntax	1 (整数)	1 (整数)	×
同期フィルタ	sync-filter	schema (スキーマメンバシップで包含)	(なし)	×
スキーマ名	sync-schema	indirect	(none)	○ [1]
フィルタ式を含む	include-table-filter	IDM_.*	(なし)	×
フィルタ式を除く	exclude-table-filter	BIN\$.{22}==\$0	(なし)	×
テーブル / ビュー名	sync-tables	usr	(なし)	○ [1]
接続初期化ステートメント	connection-init	USE idm	(なし)	×
最小数の接続を使用しますか？	use-single-connection	0 (いいえ)	(動的 [3])	×
接続プロパティ	use-single-connection	USER={$username}; PASSWORD={$password}	(動的 [3])	×
状態ディレクトリ	state-dir	. (現在のディレクトリ)	. (現在のディレクトリ)	×
JDBC ドライバ記述子ファイル名	jdbc-driver-descriptor	ora_client_thin.xml	(なし)	×
データベース記述子ファイル名	database-descriptor	ora_10g.xml	(なし)	×
手動トランザクションを使用しますか？	use-manual-transactions	1 (はい)	(動的 [2])	×
トランザクション分離レベル	transaction-isolation-level	read committed	(動的 [3])	×
ステートメントを再使用しますか？	reuse-statements	1 (再使用)	(動的 [3])	×
返された結果セットの数	handle-stmt-results	one	(動的 [3])	×
ステートメントレベルのロックを有効にしますか？	enable-locking	1 (はい)	0 (いいえ)	×
ステートメントジェネレータクラスのロック	lock-generator-class	com.novell.nds.dirxml.driver.jdbc.db.lock.OraLockGenerator	(動的 [3])	×

表示名	タグ名	サンプル値	デフォルト値	必須？
Enable Referential Attribute Support? (参照属性のサポートを有効にしますか ?)	enable-refs	1 (はい)	1 (はい)	×
Enable Meta-Identifier Support? (メタ識別子のサポートを有効にしますか ?)	enable-meta-identifiers	1 (はい)	1 (はい)	×
ユーザ名を大文字に固定	force-username-case	upper (大文字に強制)	(なし)	×
Left Outer Join Operator (左外部結合演算子)	left-outer-join-operator	(+)	(動的 [3])	×
Retrieve Minimal Metadata (最小限のメタデータを取得しますか ?)	minimal-metadata	0 (いいえ)	(動的 [3])	×
関数のリターン方法	function-return-method	result set	(動的 [3])	×
メタデータの取得でスキーマをサポートしますか ?	supports-schemas-in-metadata-retrieval	1 (はい)	(動的 [3])	×
列名を並べ替える基準	column-position-comparator	com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringByteComparator (16 進値)	(動的 [3])	×

[1] これらの相互排他的なパラメータは、「同期フィルタ」パラメータが存在しない場合に指定する必要があります。51 ページの 「同期フィルタ」 を参照してください。[2] このデフォルトは、ランタイム時に記述子ファイルおよびデータベースメタデータから動的に派生します。[3] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。

ドライバパラメータは、次のサブカテゴリに分けられます。

- 69 ページの 「カテゴリなしパラメータ」
- 51 ページの 「データベーススコープパラメータ」
- 56 ページの 「接続性パラメータ」
- 58 ページの 「互換性パラメータ」

4.3.1 カテゴリなしパラメータ

- 49 ページの 「サードパーティ製の JDBC ドライバのクラス名」
- 49 ページの 「時間構文」
- 51 ページの 「状態ディレクトリ」

サードパーティ製の JDBC ドライバのクラス名

このパラメータは、使用しているサードパーティ製の JDBC ドライバの完全修飾 Java クラス名です。

次の表は、このパラメータのプロパティのリストです。

表 *4-5* サードパーティ製の *JDBC* ドライバのクラス名 : プロパティ

プロパティ	値
タグ名	jdbc-class
必須？	○
大文字と小文字を区別？	○
サンプル値	oracle.jdbc.driver.OracleDriver
デフォルト値	(なし)

サポートされているサードパーティ製の JDBC ドライバクラス名のリストについては、126 ページの 「JDBC ドライバクラス名」を参照してください。

時間構文

「時間構文」パラメータは、ドライバが返す時間に関連するデータタイプの形式を指定します。形式には、以下のオプションのいずれかを指定できます。

形式 **:** 符号付き整数。データベースの時間、日付、およびタイムスタンプの値を 32 ビットの符号付き整数として返し、それを時間またはタイムスタンプタイプの eDirectory™ 属性にマップします。

eDirectory の時間およびタイムスタンプの構文は、1970 年 1 月 1 日午前 12 時 00 分 (UTC) から経過した秒数 (整数のみ) を表す符号なしの 32 ビット整数で構成されます。このデータタイプの最大範囲は、約 136 年です。(最初に意図されていたように) 符号なしの整数として解釈される場合、これらの構文は、1970 ～ 2106 年の範囲の日付および時刻を秒で表現できます。一方、符号付きの整数として解釈される場合は、1901 ～ 2038 年の範囲の日付および時刻を秒で表現できます。

このオプションはデフォルトです。これには 2 つの問題があります。

- アイデンティティボールトの時間およびタイムスタンプの構文では、データベースの日付またはタイムスタンプの構文で表現される日付の範囲を表現できません。
- アイデンティティボールトの時間およびタイムスタンプの構文では、秒の精度まで使用されます。データベースのタイムスタンプ構文では、ナノ秒の精度まで使用されることがあります。

これらの 2 つの制限は、以下の 2 番目および 3 番目のオプションで対処できます。

形式 **:** 標準文字列。データベースの時間、日付、およびタイムスタンプの値を標準文字列として返し、それを数値文字列タイプの属性にマップします。

次の表は、抽象データベースデータタイプおよびそれに対応する標準文字列表現を示します。

表 *4-6* データベースタイプおよび標準文字列表現

JDBC データタイプ	標準文字列形式 [1]
java.sql.Time	HHMMSS
java.sql.Date	CCYYMMDD
java.sql.Timestamp	CCYYMMDDHHMMSSNNNNNNNNN

[1] C = 世紀、Y = 年、M = 月、D = 日、H = 時、M= 分、S = 秒、N = ナノ秒

これらの固定長形式は、すべてのロケールのすべてのプラットフォームで年代順に照合されます。ナノ秒の精度はデータベースごとに異なりますが、タイムスタンプの長さは一定です。

形式 **: Java** 文字列表現。toString():java.lang.String メソッドによって返されたとおりの Java 文字列表現でデータベースの時間、日付、タイムスタンプの値を返し、それを Case Ignore/Case Exact String タイプの属性にマップします。

次の表は、抽象データベースデータタイプおよびそれに対応する Java 文字列表現を示します。

表 *4-7* データベースタイプおよび *Java* 文字列形式

JDBC データタイプ	Java 文字列形式 [1]
java.sql.Time	hh:mm:ss
java.sql.Date	yyyy-mm-dd
java.sql.Timestamp	yyyy-mm-dd hh:mm:ss.fffffffff

[1] y= 年、m= 月、d= 日、h= 時、m= 分、s= 秒、f= ナノ秒

これらの固定長形式は、すべてのロケールのすべてのプラットフォームで年代順に照合されます。ナノ秒の精度はデータベースごとに異なります。それに伴い、タイムスタンプの長さも異なります。

次の表は、「時間構文」パラメータのプロパティのリストです。

表 *4-8* 時間構文 *:* プロパティ

プロパティ	値
タグ名	time-syntax
必須？	×
デフォルト値	1 (整数)
有効な値	1 (整数) 2 (標準文字列) 3 (Java 文字列)
スキーマ依存	True

状態ディレクトリ

「状態ディレクトリ」パラメータでは、ドライバインスタンスが状態データを保存する場所を指定します。状態データは、現在トリガなし発行で使用されています。82 ページの「トリガなし発行パラメータ」を参照してください。状態データは、将来、追加される状態情報を保存するために使用できます。

各ドライバインスタンスには、2 つの状態ファイルがあります。状態ファイル名は、jdbc_*driver-instance-guid*.db および jdbc_*driver-instance-guid*.lg の形式にします。たとえば、jdbc_bd2a3dd5-d571-4171-a195-28869577b87e.db や jdbc_bd2a3dd5-d571-4171-a195-28869577b87e.lg は状態ファイル名です。ドライバインスタンスの状態ファイルを手動で識別して削除する必要がある場合、各ドライバインスタンスの GUID が起動時にトレースされます。現在の状態ディレクトリにある機能不良の状態ファイル (削除されたドライバに属するファイル) は、同じ状態ディレクトリを含むドライバインスタンスが起動されるたびに削除されます。

次の表は、このパラメータのプロパティのリストです。

表 *4-9* 状態ディレクトリ : プロパティ

プロパティ	値
タグ名	state-dir
必須 ?	×
大文字と小文字を区別 ?	プラットフォーム依存
サンプル値	c:\novell\nds\DIBFiles
デフォルト値	. (現在のディレクトリ)

4.3.2 データベーススコープパラメータ

- 51 ページの「同期フィルタ」
- 53 ページの「スキーマ名」
- 54 ページの「フィルタ式を含む」
- 55 ページの「フィルタ式を除く」
- 55 ページの「テーブル / ビュー名」

同期フィルタ

「同期フィルタ」パラメータは、同期スキーマ (ランタイム時にドライバに表示できるテーブル / ビューのセット) のメンバーにするテーブルまたはビューなどのデータベースオブジェクトを指定します。このパラメータに加えて、ドライバは、schema-aware または schema-unaware の 2 つのモードで実行できるようになりました。

schema-unaware モード。「同期フィルタ」パラメータが存在し、empty (すべてのテーブル / ビューを除く) に設定されている場合、ドライバは schema-unaware です。これは、起動時にテーブル / ビューメタデータを取得しません。したがって、メタデータメソッドは必要ありません。169 ページの付録 D「java.sql.DatabaseMetaData のメソッド」を参照してください。

schema-unaware の場合、同期スキーマを空にできます。「スキーマ名」および「Sync Tables/Views (同期テーブル / ビュー)」パラメータは、完全に無視されます。どちらも必要ありません。いずれも、パラメータを設定してもしなくても、また値なしでもありでもかまいません。53 ページの 「スキーマ名」および 55 ページの 「テーブル / ビュー名」を参照してください。

schema-unaware モードでは、ドライバは、埋め込み SQL のパススルーエージェントとして動作します。この状態では、標準 XDS イベント (たとえば、追加、変更、削除など) は無視されます。109 ページの 「XDS イベントへの SQL ステートメントの埋め込み」を参照してください。さらに、トリガ付きまたはトリガなし発行も機能しません。

schema-aware モード「同期フィルタ」パラメータが存在しないか、empty(すべてのテーブル / ビューを除外する) 以外の値に設定されている場合、ドライバは schema-aware です。テーブル / ビュー数を限定してテーブル / ビューのメタデータを取得すると、データ同期が容易になります。単一のデータベースユーザ (スキーマメンバーシップで包含) が所有するすべてのテーブル / ビューのメタデータをキャッシュできます。または、テーブル / ビュー名の明示的なリスト (テーブル / ビュー名で包含) のメタデータをキャッシュできます。schema-aware の場合、ドライバは、起動時にデータベーステーブル / ビューのメタデータを取得します。必須のメタデータメソッドのリストについては、169 ページの 付録 D「java.sql.DatabaseMetaData のメソッド」を参照してください。

schema-aware の場合、スキーマ名またはテーブル / ビュー名のパラメータが存在し、値を保持している必要があります。これらの 2 つのパラメータは相互排他的なので、いずれかのパラメータだけが値を持つようにします。53 ページの 「スキーマ名」および 55 ページの 「テーブル / ビュー名」を参照してください。

次の表は、ドライバを schema-aware にするために必要なパラメータのリストです。ドライバが schema-unaware の場合は、これらのパラメータはドライバの動作には反映されません。

表 4-10 スキーマ依存のパラメータ

パラメータ
ステートメントジェネレータクラスのロック
Enable Referential Attribute Support? (参照属性のサポートを有効にしますか ?)
Enable Meta-Identifier Support? (メタ識別子のサポートを有効にしますか ?)
Left Outer Join Operator (左外部結合演算子)
Retrieve Minimal Metadata (最小限のメタデータを取得しますか ?)
メタデータの取得でスキーマをサポートしますか ?
列名を並べ替える基準
Disable Statement-Level Locking (ステートメントレベルのロックを使用不可にしますか ?)
更新件数を確認しますか ?
Add Default Values on Insert? (挿入時にデフォルト値を追加しますか ?)
生成 / 取得方法 (テーブル - グローバル)
取得タイミング (テーブル - グローバル)

パラメータ
取得タイミング
発行者を使用不可にしますか？
ステートメントレベルのロックを使用不可にしますか？
発行モード
将来のイベント処理を有効にしますか？
イベントログのテーブル名
処理された行を削除しますか？
ループバックを許可しますか？
起動オプション
ポーリング間隔 (秒)
Publication Time of Day (発行時刻)
ポストポーリングステートメント
バッチサイズ

次の表は、このパラメータのプロパティのリストです。

表 *4-11* 同期フィルタ : プロパティ

プロパティ	値
タグ名	sync-filter
必須？	×
大文字と小文字を区別？	×
サンプル値	indirect
有効な値	empty(すべてのテーブル / ビューを除外する) schema(スキーマメンバーシップで包含) list (テーブル / ビュー名で包含)
デフォルト値 :	(なし)

スキーマ名

「スキーマ名」パラメータは、同期されているデータベーススキーマを識別します。データベーススキーマは、同期されているテーブルまたはビューの所有者の名前と似ています。たとえば、いずれもデータベースユーザ idm に属する usr および grp の 2 つのテーブルを同期するには、このパラメータの値として「idm」と入力します。

テーブル / ビュー名の代わりにこのパラメータを使用する場合、データベースオブジェクトの名前は、ドライバによって暗黙的にスキーマ修飾されます。したがって、ストアドプロシージャ、関数、またはテーブルの名前を参照するパラメータは、ここで指定されてい

る以外のスキーマに存在する場合を除いて、スキーマ修飾の必要はありません。特に、「方法とタイミング (テーブル - ローカル)」および「イベントログテーブル名」パラメータがこの影響を受けます。55 ページの 「テーブル / ビュー名」、73 ページの 「方法とタイミング (テーブル - ローカル)」、および 80 ページの 「イベントログのテーブル名」を参照してください。

次の表は、このパラメータのプロパティのリストです。

表 *4-12* スキーマ名 : プロパティ

プロパティ	値
タグ名	sync-schema
必須 ?	○ [1]
大文字と小文字を区別 ?	**144 ページの 「区切りのない識別子での大文字と小文字の区別」**を参照してください。
サンプル値	indirect
デフォルト値 :	(なし)

[1] 「スキーマ名」パラメータが「同期フィルタ」パラメータを伴わずに使用される場合、「テーブル / ビュー名」パラメータは、空のままにするか、環境設定から除外する必要があります。51 ページの 「同期フィルタ」および 55 ページの 「テーブル / ビュー名」を参照してください。

重要 **:**「スキーマ名」パラメータの値を変更すると、トリガなし発行が使用されるときにすべてのオブジェクトが強制的に再同期されます。

フィルタ式を含む

「フィルタ式を含む」パラメータは、「スキーマ名」パラメータが使用されている場合のみ機能します。53 ページの 「スキーマ名」を参照してください。

次の表は、このパラメータのプロパティのリストです。

表 *4-13* フィルタ式を含む : プロパティ

プロパティ	値
タグ名	include-table-filter
必須 ?	×
大文字と小文字を区別 ?	○
サンプル値	idm_*. (「idm_」で始まるすべてのテーブル / ビュー名)
デフォルト値	(なし)
有効な値	(任意の有効な Java 正規表現)

フィルタ式を除く

このパラメータは、「スキーマ名」パラメータが使用されている場合のみ機能します。53 ページの 「スキーマ名」を参照してください。

次の表は、このパラメータのプロパティのリストです。

表 *4-14* フィルタ式を除く : プロパティ

プロパティ	値
タグ名	exclude-table-filter
必須？	×
大文字と小文字を区別？	○
サンプル値	bin*. (「bin」で始まるすべてのテーブル / ビュー名)
デフォルト値	(なし)
有効な値	(任意の有効な Java 正規表現)

テーブル / ビュー名

「テーブル / ビュー名」パラメータでは、同期する論理データベースクラスの名前をリストすることによって、論理データベーススキーマを作成できます。論理データベースクラス名は、親テーブルおよびビューの名前です。子テーブルの名前をリストするの間違いです。

このパラメータは、スキーマの概念をサポートしない MySQL などのデータベースと同期する場合、またはデータベーススキーマに多数のテーブルまたはビューがあるがそのほとんどが必要ない場合に、特に便利です。ドライバによってキャッシュされるテーブル / ビューの定義の数を減らすと、起動時間を短縮し、ランタイムメモリの使用率を削減できます。

「スキーマ名」パラメータの代わりにこのパラメータを使用する場合は、ストアドプロシージャ、関数、またはテーブル名を参照する他のパラメータをスキーマ修飾することが必要になります。特に、「方法とタイミング (テーブル - ローカル)」および「イベントログテーブル名」パラメータがこの影響を受けます。53 ページの 「スキーマ名」、73 ページの 「方法とタイミング (テーブル - ローカル)」および 80 ページの 「イベントログのテーブル名」を参照してください。

次の表は、このパラメータのプロパティのリストです。

表 *4-15* テーブル / ビュー名 : プロパティ

プロパティ	値
タグ名	sync-tables
必須？	○ [1]

プロパティ	値
大文字と小文字を区別？	144 ページの 「区切りのない識別子での大文字と小文字の区別」を参照してください。
区切り文字	セミコロン、空白、カンマ
サンプル値	indirect.usr; indirect.grp
デフォルト値	(なし)

[1] このパラメータが「同期フィルタ」パラメータを伴わずに使用される場合、「スキーマ名」パラメータを空にするか、環境設定から除外する必要があります。51 ページの 「同期フィルタ」および 53 ページの 「スキーマ名」を参照してください。

4.3.3 接続性パラメータ

- 56 ページの 「最小数の接続を使用しますか ?」
- 57 ページの 「接続初期化ステートメント」
- 57 ページの 「接続プロパティ」

最小数の接続を使用しますか？

「最小数の接続を使用しますか ?」パラメータは、ドライバがデータベース接続を 3 つではなく 2 つ使用するかどうかを指定します。

デフォルトで、ドライバは、3 つ (購読用に 1 つ、発行用に 2 つ) の接続を使用します。発行者チャネルは、2 つの接続のうち 1 つを使用して、イベントを問い合わせ、もう 1 つを使用してクエリバック操作を容易にします。

このパラメータを論理値の True に設定した場合、必要なデータベース接続数は 2 に減らされます。1 つの接続が購読者チャネルおよび発行者チャネル間で共有され、購読および発行のクエリバックイベントの処理に使用されます。もう 1 つの接続は、発行イベントのクエリに使用されます。

旧バージョンでは、ドライバは、1 つの接続を使用した双方向の同期をサポートできていました。発行アルゴリズムは、パフォーマンスを向上し、将来のイベントの処理をサポートできるようにすると同時に、追加接続の要求を犠牲にしていた以前のアルゴリズムの制限を克服するために再設計されました。

表 *4-16* 最小数の接続を使用しますか ?: プロパティ

プロパティ	値
タグ名	use-single-connection
必須？	×
デフォルト値	(動的 [1])
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	False

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は論理値の False です。

注 : このパラメータを論理値の True に設定すると、パフォーマンスが低下します。

接続初期化ステートメント

「接続初期化ステートメント」パラメータは、ターゲットデータベースに接続した直後に実行する必要がある SQL ステートメント (存在する場合) を指定します。接続初期化ステートメントは、データベースコンテキストを変更する場合、およびセッションプロパティを設定する場合に便利です。これらのステートメントは、チャネルに関係なく、ドライバがターゲットデータベースに接続または再接続するたびに実行されます。

次の表は、このパラメータのプロパティのリストです。

表 4-17 接続初期化ステートメント : プロパティ

プロパティ	値
タグ名	connection-init
必須 ?	×
大文字と小文字を区別 ?	144 ページの 「区切りのない識別子での大文字と小文字の区別」を参照してください。
区切り文字	セミコロン
サンプル値	USE idm; SET CHAINED OFF
デフォルト値	(なし)
スキーマ依存	False

接続プロパティ

「接続プロパティ」パラメータは、認証プロパティを指定します。このパラメータは、「認証コンテキスト」パラメータで指定された JDBC URL 経由で設定できないプロパティを指定する場合に便利です。46 ページの 「認証コンテキスト」を参照してください。

このパラメータの主な用途は、サードパーティ製の JDBC ドライバで暗号化された転送を有効にすることです。関連する接続プロパティのリストについては、138 ページの「Sybase Adaptive Server Enterprise JConnect JDBC ドライバ」および 135 ページの 「Oracle Thin Client JDBC Driver」を参照してください。

接続プロパティは、キーと値のペアとして指定されます。キーは、「=」文字の左側の値として指定されます。値は、「=」文字の右側の値です。キーと値のペアは複数指定できますが、それぞれのペアは「;」文字で区切る必要があります。

接続プロパティパラメータを使用する場合は、認証情報を、認証コンテキストパラメータ、またはここで指定された JDBC URL 経由で渡すことができます。46 ページの 「認証コンテキスト」を参照してください。

接続プロパティとして指定されている場合、値トークンは、「認証 ID」パラメータで指定されるデータベースユーザ名、または「アプリケーションパスワード」パラメータで指定

されるパスワードのプレースホルダとして使用できます。46 ページの 「認証 ID」および 46 ページの 「アプリケーションパスワード」を参照してください。ユーザ名の場合、トークンは {$username} です。パスワードの場合は {$password} です。

次の表は、このパラメータのプロパティのリストです。

表 *4-18* 接続プロパティ : プロパティ

プロパティ	値
タグ名	connection-properties
必須 ?	×
大文字と小文字を区別 ?	サードパーティ製の JDBC ドライバに依存
区切り文字	セミコロン
サンプル値	USER={$username}; PASSWORD={$password}; SYBSOCKET_FACTORY=DEFAULT
デフォルト値	(なし)
スキーマ依存	False

4.3.4 互換性パラメータ

- 59 ページの 「JDBC ドライバ記述子ファイル名」
- 59 ページの 「データベース記述子ファイル名」
- 60 ページの 「手動トランザクションを使用しますか ?」
- 60 ページの 「トランザクション分離レベル」
- 61 ページの 「ステートメントを再使用しますか ?」
- 62 ページの 「返された結果セットの数」
- 63 ページの 「ステートメントレベルのロックを有効にしますか ?」
- 63 ページの 「ステートメントジェネレータクラスのロック」
- 63 ページの 「Enable Referential Attribute Support? (参照属性のサポートを有効にしますか ?)」
- 64 ページの「Enable Meta-Identifier Support? (メタ識別子のサポートを有効にしますか ?)」
- 65 ページの 「ユーザ名を大文字に固定」
- 65 ページの 「Left Outer Join Operator (左外部結合演算子)」
- 66 ページの 「Retrieve Minimal Metadata (最小限のメタデータを取得しますか ?)」
- 66 ページの 「関数のリターン方法」
- 67 ページの 「メタデータの取得でスキーマをサポートしますか ?」
- 67 ページの 「列名を並べ替える基準」

JDBC ドライバ記述子ファイル名

「JDBCドライバ記述子ファイル名」パラメータは、使用するサードパーティ製の JDBC 記述子ファイルを指定します。記述子ファイル名には、アンダースコア文字のプリフィックスを付けてはなりません (たとえば、_mysql_jdriver.xml)。この種のファイル名は予約済みです。記述子ファイルを、大文字小文字が区別されないプリフィックス「jdbc」が付けられた jar ファイル (たとえば JDBCCustomConfig.jar)、および jar ファイルの com/novell/nds/dirxml/driver/jdbc/db/descriptor/driver ディレクトリに配置します。

次の表は、このパラメータのプロパティのリストです。

表 *4-19* *JDBC* ドライバ記述子ファイル名 : プロパティ

プロパティ	値
タグ名	jdbc-driver-descriptor
必須？	×
大文字と小文字を区別？	プラットフォーム依存
サンプル値	my_custom_jdbc_driver_descriptor.xml
デフォルト値	(なし)
スキーマ依存	False

データベース記述子ファイル名

「データベース記述子ファイル名パラメータ」は、使用するデータベース記述子ファイルを指定します。記述子ファイル名のプリフィックスにアンダースコア文字を使用しないでください (_mysql.xml など)。この種の名前は予約済みです。記述子ファイルを、大文字小文字が区別されないプリフィックス「jdbc」で始まる jar ファイル (たとえば、JDBCCustomConfig.jar) に配置します。また、記述子ファイルを jar ファイルの com/novell/nds/dirxml/driver/jdbc/db/descriptor/db ディレクトリに配置します。

次の表は、このパラメータのプロパティのリストです。

表 *4-20* データベース記述子ファイル名 : プロパティ

プロパティ	値
タグ名	jdbc-driver-descriptor
必須？	×
大文字と小文字を区別？	プラットフォーム依存
サンプル値	my_custom_database_descriptor.xml
デフォルト値	(なし)
スキーマ依存	False

手動トランザクションを使用しますか？

「手動トランザクションを使用しますか ?」パラメータは、手動とユーザ定義のどちらのトランザクションを使用するかを指定します。

このパラメータは、主に、トランザクションをサポートしない MySQL MyISAM テーブルタイプとの相互運用を可能にするために使用されます。

論理値の True に設定した場合、ドライバは手動トランザクションを使用します。論理値の False に設定した場合、ドライバで実行される各ステートメントは自律的 (自動的) に実行されます。

次の表は、このパラメータのプロパティのリストです。

表 *4-21* 手動トランザクションを使用しますか *?:* プロパティ

プロパティ	値
タグ名	use-manual-transactions
必須？	×
大文字と小文字を区別？	×
デフォルト値	(動的 [1])
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	False

[1] このデフォルトは、ランタイム時に記述子ファイルおよびデータベースメタデータから動的に派生します。

注 **:** データの整合性を維持するには、このパラメータを、可能な限り論理値の True に設定します。

トランザクション分離レベル

トランザクション分離レベルパラメータは、ドライバが使用する接続のトランザクション分離レベルを設定します。有効な値は次の 6 つです。

- unsupported
- none
- read uncommitted
- read committed
- repeatable read
- serializable

これらの値のうち 5 つは、java.sql Interface Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html) で定義されているパブリック定数に対応しています。

サードパーティ製のドライバの一部は、接続のトランザクション分離レベルに対する none の設定をサポートしていないので、ドライバは追加の非標準値 unsupported もサポー

トします。PostgreSQL のオンラインマニュアル (http://www.postgresql.org/docs/current/static/transaction-iso.html) には、各分離レベルの実際の意味についてのわかりやすく正確な説明があります。

重要 : サポートされている分離レベルのリストは、データベースによって異なります。サポートされているデータベースごとのサポート分離レベルのリストについては、145 ページの 「サポートされているトランザクション分離レベル」を参照してください。

トランザクション分離レベルには read committed を使用することをお勧めします。これは、ドライバがコミットされていない変更を表示 (ダーティ読み取り) しないようにする最低限の分離レベルだからです。

次の表は、このパラメータのプロパティのリストです。

表 *4-22* トランザクション分離レベル : プロパティ

プロパティ	値
タグ名	transaction-isolation-level
必須 ?	×
大文字と小文字を区別 ?	×
デフォルト値	(動的 [1])
有効な値	unsupported none read uncommitted read committed repeatable read serializable
スキーマ依存	False

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は read committed です。

ステートメントを再使用しますか ?

「ステートメントを再使用しますか ?」パラメータは、特定の接続で一度にアクティブにする java.sql.Statement アイテムを 1 つにするか複数にするかを指定します。java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html) を参照してください。

このパラメータは、主に、Microsoft SQL Server 2000 Driver for JDBC との相互運用を可能にするために使用されます。

論理値の True に設定した場合、ドライバは Java SQL Statement を 1 回割り当て、それを再使用します。論理値の False に設定した場合、ドライバは、ステートメントオブジェクトを、それが使用されるたびに割り当てまたは割り当て解除して、特定の接続で一度に複数のステートメントがアクティブにならないようにします。

次の表は、このパラメータのプロパティのリストです。

表 4-23 ステートメントを再使用しますか ?: プロパティ

プロパティ	値
タグ名	reuse-statements
必須？	×
大文字と小文字を区別？	×
デフォルト値	(動的 [1])
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	False

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は論理値の True です。

注 : このパラメータを論理値の False に設定すると、パフォーマンスが低下します。

返された結果セットの数

「返された結果セットの数」パラメータでは、任意の SQL ステートメントから返すことができる java.sql.Result オブジェクト数を指定します。java.sql.ResultSet (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html) を参照してください。

このパラメータは、主に、任意の SQL ステートメントを評価するときに、135 ページの「Oracle Thin Client JDBC Driver」に記載された無限ループ状態になるのを避けるために使用されます。

次の表は、このパラメータのプロパティのリストです。

表 4-24 返された結果セットの数 : プロパティ

プロパティ	値
タグ名	handle-stmt-results
必須？	×
サンプル値	one
デフォルト値	(動的 [1])
有効な値	none、no (なし) single、one (1) multiple、many、yes (複数)
スキーマ依存	False

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は multiple、many、または yes です。

ステートメントレベルのロックを有効にしますか？

「ステートメントレベルのロックを有効にしますか?」パラメータは、SQL ステートメントを実行する前にドライバがデータベースリソースを明示的にロックするかどうかを指定します。

次の表は、このパラメータのプロパティのリストです。

表 *4-25* ステートメントレベルのロックを有効にしますか *?:* プロパティ

プロパティ	値
タグ名	enable-locking
必須？	×
デフォルト値	0 (いいえ)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

ステートメントジェネレータクラスのロック

「ステートメントジェネレータクラスのロック」パラメータは、保留中の SQL ステートメントのデータベースリソースを明示的にロックするために必要な SQL ステートメントの生成に使用する DBLockStatementGenerator 実装を指定します。DBLockStatementGenerator インタフェースについては、ドライバに付属している Java マニュアルを参照してください。

次の表は、このパラメータのプロパティのリストです。

表 *4-26* ステートメントジェネレータクラスのロック *:* プロパティ

プロパティ	値
タグ名	lock-generator-class
必須？	×
サンプル値	com.novell.nds.dirxml.driver.jdbc.db.lock.OraLockGenerator
デフォルト値	(動的 [1])
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は com.novell.nds.dirxml.driver.jdbc.db.lock.DBLockGenerator です。

Enable Referential Attribute Support? (参照属性のサポートを有効にしますか ?)

「Enable Referential Attribute Support? (参照属性のサポートを有効にしますか ?)」パラメータは、ドライバが論理データベースクラス間の外部キー制約を認識するかどうかを切り替

えます。これらは包含を示すために使用されます。論理データベースクラス内のテーブルの親子間の外部キー制約には影響はありません。

論理値の True に設定した場合、外部キー列が参照として解釈されます。論理値の False に設定した場合、外部キー列が非参照として解釈されます。

このパラメータは、主に、1.0 バージョンのドライバとの後方互換性を確実にするために使用されます。1.0 との互換性のためには、このパラメータを論理値の False に設定します。

次の表は、このパラメータのプロパティのリストです。

表 4-27 *Enable Referential Attribute Support? (* 参照属性のサポートを有効にしますか *?):* プロパティ

プロパティ	値
タグ名	enable-refs
必須 ?	×
デフォルト値	1 (はい)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

Enable Meta-Identifier Support? (メタ識別子のサポートを有効にしますか ?)

「Enable Meta-Identifier Support? (メタ識別子のサポートを有効にしますか ?)」パラメータは、ドライバが「pk_」や「fk_」などのビューの列名のプリフィックスをメタデータとして厳密に解釈するかどうかを切り替えます。メタデータとして解釈される場合、このようなプリフィックスは、ビューの列名の一部とは見なされません。

たとえば、メタ識別子のサポートが有効である場合、「pk_idu」列の有効な列名は「idu」であり、同じビュー内に同じ有効な名前を持つ別の列を配置することは禁止されます。メタ識別子サポートが無効である場合、「pk_idu」列の有効な列名は「pk_idu」となり、名前が「idu」の別の列を配置できます。さらに、メタ識別子サポートが有効な場合、「pk_idu」というプライマリキーを持つビューは、「idu」というプライマリキー列を持つテーブルと衝突します。メタ識別子サポートが無効の場合は衝突しません。

論理値の True に設定した場合、ビューの列のプリフィックスはメタデータとして解釈されます。論理値の False に設定した場合、ビューの列名のプリフィックスは列名そのものの一部として解釈されます。

このパラメータは、主に、1.5 バージョンのドライバとの後方互換性を確実にするために使用されます。1.5 との互換性のためには、このパラメータを論理値の False に設定します。

次の表は、このパラメータのプロパティのリストです。

表 4-28 *Enable Meta-Identifier Support? (メタ識別子のサポートを有効にしますか ?): プロパティ*

プロパティ	値
タグ名	enable-meta-identifiers
必須？	×
デフォルト値	1 (はい)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

ユーザ名を大文字に固定

「ユーザ名を大文字に固定」パラメータは、ターゲットデータベースの認証に使用されるドライバのユーザ名の大文字と小文字を変更します。

このパラメータは、主に、Informix JDBC Driver を ANSI 互換のデータベースに対して使用する場合の相互運用を可能にするために使用されます。131 ページの 「Informix JDBC Driver」 を参照してください。

次の表は、このパラメータのプロパティのリストです。

表 4-29 *ユーザ名を大文字に固定 : プロパティ*

プロパティ	値
タグ名	force-username-case
必須？	×
デフォルト値	(強制しない)
有効な値	lower (小文字に強制) mixed (大 / 小文字の混在) upper (大文字に強制)
スキーマ依存	False

Left Outer Join Operator (左外部結合演算子)

「Left Outer Join Operator (左外部結合演算子)」パラメータは、トリガなし発行クエリで使用される左外部結合演算子を指定します。将来、他の目的で使用される可能性もあります。

次の表は、このパラメータのプロパティのリストです。

表 4-30 *Left Outer Join Operator (左外部結合演算子): プロパティ*

プロパティ	値
タグ名	left-outer-join-operator
必須？	×

プロパティ	値
デフォルト値	(動的 [1])
有効な値	*= (+) LEFT OUTER JOIN
スキーマ依存	True

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は LEFT OUTER JOIN です。

Retrieve Minimal Metadata (最小限のメタデータを取得しますか ?)

論理値の True に設定した場合、ドライバは、必須のメタデータメソッドだけを呼び出します。論理値の False に設定した場合、ドライバは必須およびオプションのメタデータメソッドを呼び出します。必須およびオプションのメタデータメソッドのリストについては、169 ページの付録 D「java.sql.DatabaseMetaData のメソッド」を参照してください。オプションのメタデータメソッドは、複数値および参照属性の同期に必要です。

表 4-31 *Retrieve Minimal Metadata (* 最小限のメタデータを取得しますか *?): プロパティ*

プロパティ	値
タグ名	minimal-metadata
必須 ?	×
デフォルト値	(動的 [1])
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は論理値の False です。

注 **:** この値を論理値の True に設定すると、起動時間が短縮されサードパーティ製の JDBC ドライバとの互換性が向上しますが、機能面が犠牲になります。

関数のリターン方法

「関数のリターン方法」パラメータは、データがデータベース関数からどのように取得されるかを指定します。

このパラメータは、主に、Informix JDBC Driver との相互運用を可能にするために使用されます。131 ページの 「Informix JDBC Driver」を参照してください。

result set に設定した場合、関数の結果は結果セットを通じて取得されます。return value に設定した場合、関数の結果は、単一のスカラ返り値として取得されます。

表 *4-32* 関数のリターン方法 : プロパティ

プロパティ	値
タグ名	function-return-method
必須？	×
デフォルト値	(動的 [1])
有効な値	result set return value (スカラ返り値)
スキーマ依存	False

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。

メタデータの取得でスキーマをサポートしますか？

「メタデータの取得でスキーマをサポートしますか ?」パラメータは、データベースメタデータを取得するときにスキーマ名を使用するかどうかを指定します。

このパラメータは、主に、Informix JDBC Driver を ANSI 互換のデータベースに対して使用する場合の相互運用を可能にするために使用されます。131 ページの 「Informix JDBC Driver」 を参照してください。

論理値の True に設定した場合、スキーマ名が使用されます。論理値の False に設定した場合、スキーマ名は使用されません。

表 *4-33* メタデータの取得でスキーマをサポートしますか ?: プロパティ

プロパティ	値
タグ名	supports-schemas-in-metadata-retrieval
必須？	×
デフォルト値	(動的 [1])
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	False

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は論理値の True です。

列名を並べ替える基準

「列名を並べ替える基準」パラメータは、列名による並べ替えをサポートしないレガシデータベースで列位置を決定する方法を指定します。

このパラメータは、主に、DB2/AS400 などのレガシデータベースとの相互運用を可能にするために使用されます。

列名を 16 進値で並べ替えると、ドライバのインスタンスが異なるサーバに再割り当てされても、そのまま機能し、変更する必要がありません。列名をプラットフォームまたはロケール文字列の照合順序で並び替えた方が直感的になりますが、ドライバのインスタンス

が異なるサーバに再配置される場合は環境設定の変更が必要になります。特に、ログテーブル列の順序と複合列名の順序は変わります。後者の場合、スキーママッピングポリシーとオブジェクトの関連付け値を更新しなければならない場合があります。前者の場合、イベントログのテーブル列名を変更しなければならない場合があります。

以下の場合は、完全修飾 Java クラス名を指定することもできます。

- Java クラス名が java.util.Comparator (http://java.sun.com/j2se/1.5.0/docs/api/java/util/Comparator.html) インタフェースを実装している。
- Java クラス名で java.lang.String (http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) 引数を使用できる。
- クラスがランタイムクラスパスに含まれる。

表 *4-34* 列名を並べ替える基準 : プロパティ

プロパティ	値
タグ名	column-position-comparator
必須？	×
デフォルト値	(動的 [1])
有効な値	com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringByteComparator (16 進値) com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringComparator (文字列照合順序) (java.lang.String を受け付ける任意の java.util.Comparator)
スキーマ依存	True

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。それ以外の場合のデフォルト値は com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringByteComparator になります。

重要 **:** 特定の環境設定でこのパラメータを設定した後は、パラメータを変更しないでください。

4.4 購読パラメータ

次の表は、購読者レベルのパラメータとそのプロパティの概要を示します。

表 *4-35* 購読者レベルのパラメータおよびプロパティ

表示名	タグ名	サンプル値	デフォルト値	必須？
購読者を使用不可にしますか？	disable	1 (はい)	0 (いいえ)	×
生成 / 取得方法 (テーブル - グローバル)	key-gen-method	auto	none (購読イベント)	
取得タイミング (テーブル - グローバル)	key-gen-timing	after (行の挿入後)	before (行の挿入前)	×

表示名	タグ名	サンプル値	デフォルト値	必須？
方法とタイミング (テーブル - ローカル)	key-gen	usr("?=indirect.proc_idu()", before)	(なし)	×
ステートメントレベルのロックを使用不可にしますか？	disable-locking	1 (はい)	0 (いいえ)	×
更新件数を確認しますか？	check-update-count	0 (いいえ)	1 (はい)	×
Add Default Values on Insert? (挿入時にデフォルト値を追加しますか ?)	add-default-values-on-view-insert	0 (いいえ)	(動的 [1])	×

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。

購読パラメータには、2 つのサブカテゴリがあります。

- 69 ページの 「カテゴリなしパラメータ」
- 71 ページの 「プライマリキーパラメータ」

4.4.1 カテゴリなしパラメータ

- 69 ページの 「購読者を使用不可にしますか ?」
- 70 ページの 「ステートメントレベルのロックを使用不可にしますか ?」
- 70 ページの 「更新件数を確認しますか ?」
- 71 ページの 「Add Default Values on Insert? (挿入時にデフォルト値を追加しますか ?)」

購読者を使用不可にしますか？

「購読者を使用不可にしますか ?」パラメータは、購読者チャネルが使用不可かどうかを指定します。

このパラメータを論理値の True に設定した場合、購読者チャネルは使用不可です。このパラメータを論理値の False に設定した場合、購読者チャネルはアクティブです。

表 *4-36* 購読者を使用不可にしますか *?:* プロパティ

プロパティ	値
タグ名	disable
必須？	×
デフォルト値	0 (いいえ)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	False

ステートメントレベルのロックを使用不可にしますか？

「ステートメントレベルのロックを使用不可にしますか?」パラメータは、各 SQL ステートメントを実行する前にデータベースリソースがこのチャネルで明示的にロックされるかどうかを指定します。このパラメータは、「ステートメントレベルのロックを有効にしますか?」が論理値の True に設定されている場合のみアクティブです。

このパラメータを論理値の True に設定した場合、データベースリソースは明示的にロックされます。このパラメータを論理値の False に設定した場合、データベースリソースは明示的にはロックされません。

表 *4-37* ステートメントレベルのロックを使用不可にしますか *?:* プロパティ

プロパティ	値
タグ名	disable-locking
必須？	×
デフォルト値	0 (いいえ)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

更新件数を確認しますか？

「更新件数を確認しますか?」パラメータは、テーブルに対して INSERT、UPDATE、および DELETE ステートメントが実行されたときに、そのテーブルが実際に更新されたかどうかを購読者チャネルがチェックするかどうかを指定します。

論理値の True に設定した場合、更新件数がチェックされます。まったく更新されていない場合は例外が発生します。論理値の False に設定した場合、更新件数は無視されます。

ステートメントが before-trigger ロジックで再定義される場合、このパラメータは論理値 False に設定されます。

Microsoft SQL Server を使用している場合は、デフォルト値を使用します。これは、トリガロジックの (トランザクションをロールバックする可能性がある) エラーが購読者チャネルに反映されないからです。

表 *4-38* 更新件数を確認しますか *?:* プロパティ

プロパティ	値
タグ名	check-update-count
必須？	×
デフォルト値	1 (はい)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

Add Default Values on Insert? (挿入時にデフォルト値を追加しますか ?)

「Add Default Values on Insert? (挿入時にデフォルト値を追加しますか ?)」パラメータは、ビューに対して INSERT ステートメントが実行されるときに、購読者チャネルがデフォルト値を提供するかどうかを指定します。

このパラメータは、主に、Microsoft SQL Server 2000 との相互運用を可能にするために使用されます。このデータベースは、NOT NULL の制約があるビューの列が、INSERT ステートメントに NULL 以外の値を含むことを要求します。

このパラメータを論理値の True に設定した場合、ビューに対して実行される INSERT ステートメントにデフォルト値が提供されます。明示的な値はまだ使用可能になっていません。このパラメータを論理値の False に設定した場合、デフォルト値は提供されません。

表 4-39 *Add Default Values on Insert? (* 挿入時にデフォルト値を追加しますか *?): プロパティ*

プロパティ	値
タグ名	add-default-values-on-view-insert
必須 ?	×
デフォルト値	(動的 [1])
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

[1] このデフォルトは、ランタイム時に記述子ファイルから動的に派生します。

4.4.2 プライマリキーパラメータ

- 72 ページの 「生成 / 取得方法 (テーブル - グローバル)」
- 72 ページの 「取得タイミング (テーブル - グローバル)」
- 73 ページの 「方法とタイミング (テーブル - ローカル)」

INSERT ステートメントにマップする <add> イベントを処理する場合、購読者チャネルは、プライマリキー値を使用して、Identity Manager の関連付けを作成します。これらのパラメータは、購読者チャネルが関連付けの値の作成に必要なプライマリキー値を取得する方法とタイミングを指定します。プライマリキー値の取得方法は、プライマリキーの生成 / 取得方法で指定します。取得タイミングは、プライマリキー値がいつ取得されるかを示します。

次の表は、サポートされている方法およびタイミングを示します。

表 4-40 サポートされている方法およびタイミング

方法	タイミング : before (行の挿入前)	タイミング : after (行の挿入後)
none (購読イベント)	X	0[1]
driver (購読者生成)	X	X

方法	タイミング : before (行の挿入前)	タイミング : after (行の挿入後)
auto (自動生成 / 識別列)	0[2]	X
(ストアドプロシージャ / 関数)	X	X

[1] 購読者チャネルは、自動的にこのタイミングを before に上書きします。[2] 購読者チャネルは自動的にこのタイミングを after に上書きします。

生成 / 取得方法 (テーブル - グローバル)

「生成 / 取得方法 (テーブル - グローバル)」パラメータは、すべての親テーブルおよびビューのプライマリキー値の生成または取得法を指定します。「方法とタイミング」パラメータは、このパラメータをテーブル / ビューごとに上書きします。73 ページの 「方法とタイミング (テーブル - ローカル)」を参照してください。

このパラメータを none に設定した場合、プライマリキー値はすでに購読イベントに存在すると見なされます。このパラメータを driver に設定した場合、プライマリキー値は次のいずれかの方法で生成されます。

- 取得タイミングが before に設定されている場合は SELECT (MAX()+1) ステートメントを使用
- 取得タイミングがafterに設定されている場合はSELECT MAX()ステートメントを使用

列タイプが文字列の場合、購読者チャネルは、System.CurrentTimeMillis() の返り値を使用して値を生成します。他のデータタイプはサポートされていません。

このパラメータを auto に設定した場合、プライマリキー値は、java.sql.Statement.getGeneratedKeys():java.sql.ResultSet メソッドで取得されます。サポートされているサードパーティ製の JDBC ドライバのうち、現在このメソッドを実装しているのは、MySQL Connector/J JDBC ドライバだけです。134 ページの 「MySQL Connector/J JDBC Driver」を参照してください。

表 *4-41* *生成 / 取得方法 (テーブル - グローバル): プロパティ*

プロパティ	値
タグ名	key-gen-method
必須 ?	×
デフォルト値	none (購読イベント)
有効な値	none (購読イベント) driver (購読者生成) auto (自動生成 / 識別列)
スキーマ依存	True

取得タイミング (テーブル - グローバル)

「取得タイミング (テーブル - グローバル)」パラメータは、購読者チャネルがすべての親テーブルおよびビューのプライマリキー値をいつ取得するかを指定します。「方法とタイ

ミング (テーブル - ローカル)」パラメータは、このパラメータを上書きします。73 ページの 「方法とタイミング (テーブル - ローカル)」を参照してください。

このパラメータを before に設定した場合、プライマリキー値は挿入前に取得されます。このパラメータを after に設定した場合、プライマリキー値は挿入後に取得されます。

表 *4-42* 取得タイミング *(テーブル - グローバル): プロパティ*

プロパティ	値
タグ名	key-gen-timing
必須 ?	×
デフォルト値	before (行の挿入前)
有効な値	before (行の挿入前) after (行の挿入後)
スキーマ依存	True

方法とタイミング (テーブル - ローカル)

「方法とタイミング (テーブル - ローカル)」パラメータは、プライマリキーの生成 / 取得方法、および取得タイミングを親テーブル / ビューごとに指定します。これは、実質的に生成 / 取得方法および取得タイミングをテーブルまたはビュー名にマップします。このパラメータの構文は、手続き型プログラミング言語の、複数の引数を伴うメソッド呼び出し (*method-name*(*argument1*, *argument2*) など) をミラーリングします。

「テーブル / ビュー名」パラメータを使用する場合は、このパラメータの値で参照されるテーブル、ビュー、ストアドプロシージャ、または関数を、通常明示的にスキーマ修飾する必要があります。「スキーマ名」パラメータを使用する場合は、このパラメータの値で参照されるテーブル、ビュー、ストアドプロシージャ、または関数が、そのスキーマ名で暗黙的にスキーマ修飾されます。このパラメータの値で参照されるテーブル、ビュー、ストアドプロシージャ、または関数が、暗黙的なスキーマ以外のスキーマにある場合は、スキーマ修飾する必要があります。

BNF

このパラメータの値を BNF (バッカス記法 (http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html)) で表記すると、次のようになります。

```
<key-gen> ::=<table-or-view-name> "(" <generation-retrieval-method>,
<retrieval-timing> ")" {[<delimiter>] <key-gen>}

<generation-retrieval-method> ::=none | driver | auto | """
<procedure-signature> """ | """ <function-signature>  """

<table-or-view-name> ::=<legal-undelimited-database-table-or-view-
identifier>

<delimiter> ::=";" | "," | <white-space>

<procedure-signature> ::=<schema-qualifier> "." <stored-routine-
name>"("<argument-list>")"
```

```
<function-signature> ::="?=" <procedure-signature>

<schema-qualifier> ::=<legal-undelimited-database-username-identifier>

<stored-routine-name> ::=<legal-undelimited-database-stored-routine -
identifier>

<argument-list> ::=<column-name>{"," <column-name>}

<column-name> ::=<column-from-table-or-view-name-previously-specified>
```

生成または取得方法

生成または取得方法では、プライマリキー値の生成および取得 (必要な場合) 方法を指定します。可能な方法は、None、Driver、Auto、および Stored Procedure/Function です。

None デフォルトで、購読者チャネルは、アイデンティティボールトがプライマリキー値の信頼されたソースであり、必要な値がすでに特定の <add> イベントに存在すると見なしています。この場合は、プライマリ値はすでに存在するので生成する必要はありません。現在の <add> イベントから取得するだけで済みます。この方法は、GUID などの eDirectory 属性が明示的に親テーブルまたはビューのプライマリキー列にスキーママップされている場合に適しています。

usr というテーブルと view_usr というビューがあり、アイデンティティボールトがプライマリキー値の信頼されたソースである場合は、このパラメータの値は次のようになります。

usr(none); view_usr(none)

この方法を使用する場合は、CN ではなく GUID を親テーブルまたはビューのプライマリキー列にマップすることをお勧めします。

Driver この方法は、データベースが、指定された親テーブルまたはビューのプライマリキー値の信頼されたソースであると見なします。

プロトタイピングの場合、または展開の初期段階では、ストアドプロシージャまたは関数が記述される前に購読者チャネルでプライマリキー値を生成することが望ましいことがよくあります。また、ストアドプロシージャまたは関数をサポートしないデータベースでこの方法を使用することもできます。ただし、運用環境でこの方法を使用する場合は、<add> イベントによって生成されるすべての SQL ステートメントを、シリアル化可能なトランザクションに含める必要があります。詳細については、60 ページの 「トランザクション分離レベル」を参照してください。

すべてのトランザクションをシリアル化可能にする代わりに、埋め込みの SQL 属性を使用して個別のトランザクション分離レベルを設定することもできます。詳細については、115 ページの 「トランザクション分離レベル」を参照してください。

任意の数値列タイプの場合、購読者チャネルは、以下を使用してプライマリキー値を生成します。

- タイミングが before の場合は単純な SELECT(MAX+1) ステートメント
- タイミングが after の場合は SELECT MAX() ステートメント

列タイプが文字列の場合、購読者チャネルは、System.CurrentTimeMillis() の返り値を使用して値を生成します。他のデータタイプはサポートされていません。

usr というテーブルと view_usr というビューがあり、データベースがプライマリキー値の信頼されたソースである場合は、このパラメータの値は次のようになります。

usr(driver); view_usr(driver)

この方法を使用する場合は、スキーママッピングポリシーおよびチャネルフィルタでプライマリキー列を省略することをお勧めします。

Auto この方法は、データベースが、指定された親テーブルまたはビューのプライマリキー値の信頼されたソースであると見なします。

一部のデータベースは、挿入される行のプライマリキー値を自動的に生成する識別列をサポートしています。この方法は、自動生成されるプライマリキー値を、JDBC 3 インタフェースメソッド java.sql.Statement.getGeneratedKeys():java.sql.ResultSet を使用して取得します。サポートされているサードパーティ製の JDBC ドライバのうち、現在このメソッドをサポートしているのは、MySQL Connector/J JDBC ドライバだけです。134 ページの「MySQL Connector/J JDBC Driver」を参照してください。

usr というテーブルと view_usr というビューがあり、データベースがプライマリキー値の信頼されたソースである場合は、このパラメータの値は次のようになります。

usr(auto); view_usr(auto)

この方法を使用する場合は、スキーママッピングポリシーおよびチャネルフィルタでプライマリキー列を省略することをお勧めします。

ストアドプロシージャ / 関数 **:** この方法は、データベースが、指定された親テーブルまたはビューのプライマリキー値の信頼されたソースであると見なします。

以下を前提とした例を考えます。

- プライマリキー列 idu を含む usr というテーブルがある
- プライマリキー値 pk_idu を含む view_usr という名前のビューがある
- データベース関数 func_last_usr_idu およびストアドプロシージャ sp_last_view_usr_pk_idu があり、いずれもそれぞれに対応するテーブル / ビューに対して最後に生成されたプライマリキー値を返す

この場合、パラメータの値は、次のようになります。

usr("?=func_last_usr_idu()"); view_usr("sp_last_view_usr_pk_idu(pk_idu)")

前の例では、パラメータはストアドプロシージャに渡されます。パラメータは、関数にも渡すことができますが、これは通常必要ありません。関数とは異なり、ストアドプロシージャは、通常、パラメータを通じて値を返します。ストアドプロシージャの場合、プライマリキー列は、IN OUT パラメータとして渡す必要があります。キーではにの列は、IN パラメータとして渡す必要があります。

ストアドプロシージャと関数のどちらの場合も、パラメータの順序、数、およびデータタイプは、プロシージャまたは関数で想定されているパラメータの順序、数、データタイプに対応する必要があります。

この方法を使用する場合は、スキーママッピングポリシーおよびチャネルフィルタでプライマリキー列を省略することをお勧めします。

取得タイミング

「取得タイミング」パラメータは、プライマリキー値がいつ取得されるかを指定します。

<add> イベントでは、親テーブルまたはビューに対して、少なくとも 1 つの INSERT ステートメントが常に生成されます。このパラメータのこの部分では、プライマリキー値を取得するタイミングを、最初の INSERT ステートメントを基準として指定します。

Before これはデフォルト設定です。この設定を指定した場合、プライマリキー値は、最初の INSERT ステートメントの前に取得されます。

重要 **:** この取得タイミングは、auto 以外のすべての生成 / 取得方法でサポートされています。取得タイミングは、方法が none の場合は必須です。

After この設定を指定した場合、プライマリキー値は、最初の INSERT ステートメントの後に取得されます。

重要 **:** この取得タイミングは、none 以外のすべての生成 / 取得方法でサポートされています。取得タイミングは、方法が auto の場合は必須です。

次の例は、前の例に取得タイミング情報を追加したものです。

usr(none, before); view_usr(none, before)

usr(driver, before); view_usr(driver, after)

usr(auto, after); view_usr(auto, after)

usr("?=func_last_usr_idu()", before); view_usr("sp_last_view_usr_pk_idu(pk_idu)", after)

次の表は、このパラメータのプロパティのリストです。

表 *4-43* 取得タイミング : プロパティ

プロパティ	値
タグ名	key-gen
必須 ?	×
大文字と小文字を区別 ?	144 ページの 「区切りのない識別子での大文字と小文字の区別」を参照してください。
サンプル値	usr("?=proc_idu()", before)
デフォルト値	(なし)
有効な値	(BNF に準拠した任意の文字列)
スキーマ依存	True

4.5 発行パラメータ

次の表は、発行者レベルのパラメータとそのプロパティの概要を示します。

表 *4-44* 発行者レベルのパラメータおよびプロパティ

表示名	タグ名	サンプル値	デフォルト値	必須
発行者を使用不可にしますか？	disable	1 (はい)	0 (いいえ)	×
ステートメントレベルのロックを使用不可にしますか？	disable-locking	1 (はい)	0 (いいえ)	×
発行モード	publication-mode	2 (トリガなし)	1 (トリガ付き)	×
イベントログのテーブル名	log-table	indirect_process	(なし)	○ [1]
処理された行を削除しますか？	delete-from-log	0 (いいえ)	1 (はい)	×
ループバックを許可しますか？	allow-loopback	1 (はい)	0 (いいえ)	×
将来のイベント処理を有効にしますか？	handle-future-events	1 (はい)	0 (いいえ)	×
起動オプション	startup-option			×
ポーリング間隔 (秒)	polling-interval	60	10	× [2]
Publication Time of Day (発行時刻)	time-of-day	15:30:00	(なし)	× [2]
ポストポーリングステートメント	post-poll-stmt	DELETE FROM direct.direct_process	(なし)	×
バッチサイズ	batch-size	16	1	×
ハートビート間隔 (分)	pub-heartbeat-interval	10	0	×

[1] トリガ付き発行モードでは必須です。[2] これらのパラメータは相互排他的です。

発行パラメータは、大きく 4 つのサブカテゴリに分けられます。

- 69 ページの 「カテゴリなしパラメータ」
- 80 ページの 「トリガ付き発行パラメータ」
- 82 ページの 「トリガなし発行パラメータ」
- 83 ページの 「ポーリングパラメータ」

4.5.1 カテゴリなしパラメータ

- 77 ページの 「発行者を使用不可にしますか ?」
- 78 ページの 「発行モード」

発行者を使用不可にしますか？

「発行者を使用不可にしますか ?」パラメータは、発行者チャネルが使用不可かどうかを指定します。使用不可にした場合、発行者チャネルはデータベースイベントを問い合わせ

ません。ただし、その場合も、「購読者を使用不可にしますか ?」パラメータとは異なり、発行者チャネルでデータベースクエリを発行して、代わりの発行アルゴリズムを容易にすることはできます。

このパラメータを論理値の True に設定した場合、発行者チャネルは使用不可です。このパラメータを論理値の False に設定した場合、発行者チャネルはアクティブです。

表 *4-45* 発行者を使用不可にしますか ?: プロパティ

プロパティ	値
タグ名	disable
必須 ?	×
デフォルト値	0 (いいえ)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

ステートメントレベルのロックを使用不可にしますか ?

「ステートメントレベルのロックを使用不可にしますか ?」パラメータは、各 SQL ステートメントを実行する前にデータベースリソースがこのチャネルで明示的にロックする必要があるかどうかを指定します。このパラメータは、「ステートメントレベルのロックを有効にしますか ?」パラメータが論理値の True に設定されている場合のみアクティブです。

このパラメータを論理値の True に設定した場合、データベースリソースは明示的にロックされます。このパラメータを論理値の False に設定した場合、データベースリソースは明示的にはロックされません。

表 *4-46* ステートメントレベルのロックを使用不可にしますか ?: プロパティ

プロパティ	値
タグ名	disable-locking
必須 ?	×
デフォルト値	0 (いいえ)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

発行モード

「発行モード」パラメータは、使用する発行アルゴリズムを指定します。

1 (トリガ付き) に設定した場合、発行者チャネルは、イベントログテーブルでイベントをポールします。2 (トリガなし) に設定した場合、発行者チャネルは、同期スキーマ内のすべてのテーブル / ビューで変更を探し、イベントを同期します。

次の表は、このパラメータのプロパティのリストです。

表 *4-47* 発行モード : プロパティ

プロパティ	値
タグ名	publication-mode
必須？	×
デフォルト値	1 (トリガ付き)
有効な値	1 (トリガ付き) 2 (トリガなし)
スキーマ依存	True

将来のイベント処理を有効にしますか？

トリガ付き発行の場合、「将来のイベント処理を有効にしますか ?」パラメータは、イベントトログテーブル内の行が挿入順 (record_id 列) と年代順 (event_time 列) のどちらで並べられ処理されるかを指定します。

このパラメータを論理値の True に設定した場合、イベントログテーブルの行は、挿入順に発行されます。このパラメータを論理値の False に設定した場合、イベントログテーブルの行は、年代順に発行されます。

トリガなしの発行の場合、「将来のイベント処理を有効にしますか ?」パラメータは、データベースのローカル時刻が各イベントと共に発行されるかどうかを指定します。この追加情報は、スケジュール指定されたイベントを強制的に再試行するために使用されます。これを機能させるには、イベントをいつ処理するかを指定する列が、この機能を使用する各論理データベースクラスに含まれ、発行者フィルタに通知のみの属性として配置されている必要があります。181 ページの付録 H「データベース記述子の DTD」を参照してください。

データベースのローカル時刻は、各 XDS イベント (追加、変更、削除など) の属性として発行されます。属性名は、jdbc:database-local-time です。ここで、名前空間のプリフィックス jdbc は、urn:dirxml:jdbc にバインドされます。形式は、Java 文字列表現の java.sql.Timestamp: yyyy-mm-dd hh:mm:ss.fffffffff です。イベントをいつ処理するかを示す値は、「時間構文」パラメータの値に応じて、整数、標準文字列、または Java 文字列として発行できます。49 ページの 「時間構文」を参照してください。

発行構文にかかわらず、この値を解析してデータベースローカル時刻の値と比較できます。次の表は、時間構文と適切な解析メソッドのマッピングを示しています。

表 *4-48* 時間構文と解析メソッドのマッピング

時間構文	解析メソッド
整数	java.sql.Timestamp.valueOf(java.lang.String):java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html)
標準文字列	com.novell.nds.dirxml.driver.jdbc.db.DSTime(java.lang.String, java.lang.String, java.lang.String, java.lang.String)
Java 文字列	java.sql.Timestamp.valueOf(java.lang.String):java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html)

両方の時刻を同じタイムスタンプオブジェクト表現にした後、次のメソッドを使用して比較できます。

- com.novell.nds.dirxml.driver.jdbc.db.TimestampUtil.before(java.sql.Timestamp, java.sql.Timestamp):boolean
- com.novell.nds.dirxml.driver.jdbc.db.TimestampUtil.after(java.sql.Timestamp, java.sql.Timestamp):boolean

ポリシーの例については、181 ページの付録 H「データベース記述子の DTD」を参照してください。

このパラメータを論理値の True に設定した場合、各イベントと共にローカルデータベース時刻が発行されます。このパラメータを論理値の False に指定した場合、この情報は省略されます。

次の表は、このパラメータのプロパティのリストです。

表 *4-49* 将来のイベント処理を有効にしますか ?: プロパティ

プロパティ	値
タグ名	handle-future-events
必須 ?	×
デフォルト値	0 (いいえ)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

4.5.2 トリガ付き発行パラメータ

JDBC 用ドライバは、4 つのトリガ付き発行パラメータをどれでも使用できます。

- 80 ページの 「イベントログのテーブル名」
- 81 ページの 「処理された行を削除しますか ?」
- 82 ページの 「ループバックを許可しますか ?」

イベントログのテーブル名

「イベントログのテーブル名」パラメータは、発行イベントが保存されるイベントログテーブルの名前を指定します。

ここで指定する名前は、100 ページの 「イベントログテーブル」の定義に従う必要があります。

55 ページの 「テーブル / ビュー名」を使用する場合は、通常このテーブル名を明示的にスキーマ修飾する必要があります。53 ページの 「スキーマ名」を使用する場合は、このテーブル名は、そのスキーマ名で暗黙的にスキーマ修飾されます。このテーブルが暗黙的なスキーマ以外のスキーマにある場合は、スキーマ修飾する必要があります。

次の表は、このパラメータのプロパティのリストです。

表 4-50 イベントログのテーブル名 : プロパティ

プロパティ	値
タグ名	log-table
必須？	×[1]
大文字と小文字を区別？	144 ページの 「区切りのない識別子での大文字と小文字の区別」を参照してください。
サンプル値	eventlog
デフォルト値	(なし)
スキーマ依存	True

[1] このパラメータは、78 ページの 「発行モード」を 1 (トリガ付き発行) に設定した場合は必須です。

処理された行を削除しますか？

「処理された行を削除しますか ?」パラメータは、処理された行をイベントログテーブルから削除するかどうかを指定します。

このパラメータを論理値の True に設定した場合、処理された行は削除されます。このパラメータを論理値の False に設定した場合、処理された行の status フィールド値が更新されます。

処理された行がイベントログテーブルに残ることによって起こるパフォーマンスへの影響を緩和するために、定期的に行を履歴テーブルに移動することをお勧めします。このためには、次のいずれかを実行します。

- 84 ページの 「ポストポーリングステートメント」パラメータを通じてクリーンアップのストアドプロシージャを呼び出す。
- イベントログテーブルに before-delete トリガを配置して、イベントログテーブルに対して実行される削除イベントを中断し、行がイベントログテーブルから削除される前に、その行を履歴テーブルに移動する。

次の表は、このパラメータのプロパティのリストです。

表 4-51 処理された行を削除しますか ?: プロパティ

プロパティ	値
タグ名	delete-from-log
必須？	×
デフォルト値	0 (いいえ)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

注 : このパラメータを論理値の False に設定した場合、処理済みの行を定期的にイベントログテーブルから削除しない限り、発行のパフォーマンスが低下します。

ループバックを許可しますか ?

「ループバックを許可しますか ?」パラメータは、ドライバのデータベースユーザアカウントによるイベントを発行するかどうかを指定します。

このパラメータを論理値の True に設定した場合、ループバックイベントが発行されます。このパラメータを論理値の False に設定した場合、ループバックイベントは無視されます。

次の表は、このパラメータのプロパティのリストです。

表 *4-52* ループバックを許可しますか *?:* プロパティ

プロパティ	値
タグ名	allow-loopback
必須 ?	×
デフォルト値	0 (いいえ)
有効な値	1、yes、true (はい) 0、no、false (いいえ)
スキーマ依存	True

注 : このパラメータを論理値の True に設定した場合、外部からのイベントが発行されることがあるので、パフォーマンスが低下する可能性があります。

4.5.3 トリガなし発行パラメータ

JDBC 用のドライバは、1 つのトリガなし発行パラメータを使用できます。ただし、51 ページの 「状態ディレクトリ」パラメータも、トリガなし発行に影響します。

起動オプション

「起動オプション」パラメータは、トリガなし発行の開始時に何を行うかを指定します。

表 *4-53* 起動オプション *:* 設定および結果

設定	結果
1	すべてのオブジェクトが変更されたと見なされ、再発行されます。
2	過去および現在の変更は無視されます。
3	過去および現在のすべての変更が発行されます。

次の表は、このパラメータのプロパティのリストです。

表 *4-54* 起動オプション : プロパティ

プロパティ	値
タグ名	startup-option
必須？	×
デフォルト値	1 (すべての変更を処理)
有効な値	1 (すべてのオブジェクトを再同期) 2 (将来の変更のみを処理) 3 (すべての変更を処理)
スキーマ依存	True

重要 **:** トリガなし発行では、次の変更によって強制的に再同期できます。

- 「認証コンテキスト」パラメータで、URL プロパティ以外の何らかの変更を行うと、トリガなし発行が使用されるときにすべてのオブジェクトが強制的に再同期されます。
- 「スキーマ名」パラメータの値を変更すると、トリガなし発行が使用されるときにすべてのオブジェクトが強制的に再同期されます。
- 「状態ディレクトリ」パラメータの値を変更すると再同期されます。

4.5.4 ポーリングパラメータ

- 83 ページの 「ポーリング間隔 (秒)」
- 84 ページの 「Publication Time of Day (発行時刻)」
- 84 ページの 「ポストポーリングステートメント」
- 85 ページの 「バッチサイズ」
- 86 ページの 「ハートビート間隔 (分)」

ポーリング間隔 (秒)

「ポーリング間隔 (秒)」パラメータは、ポーリングサイクルの間にアクティブでない状態で経過する秒数を指定します。

次の表は、このパラメータのプロパティのリストです。

表 *4-55* ポーリング間隔 *(秒):* プロパティ

プロパティ	値
タグ名	polling-interval
必須？	×
デフォルト値	10 (秒)
有効な値	1 ～ 604800 (1 週間)
スキーマ依存	True

注 : この値は 10 秒以上に設定することをお勧めします。

Publication Time of Day (発行時刻)

「Publication Time of Day (発行時刻)」パラメータは、各日の何時に発行を開始するかを指定します。時刻は、サーバのローカル時刻 (ドライバが実行されているサーバでの時刻) を意味すると解釈されます。

次の表は、このパラメータのプロパティのリストです。

表 4-56 *Publication Time of Day (発行時刻): プロパティ*

プロパティ	値
タグ名	time-of-day
必須 ?	×
サンプル値	13:00:00 (午後 1 時)
デフォルト値	(なし)
有効な値	hh:mm:ss (h = 時、m = 分、s = 秒)
スキーマ依存	True

注 : このパラメータは、「ポーリング間隔 (秒)」パラメータを上書きします。83 ページの「ポーリング間隔 (秒)」を参照してください。

ポストポーリングステートメント

「ポストポーリングステートメント」パラメータは、各アクティブポーリングサイクルの最後に実行される SQL ステートメントを指定します。アクティブポーリングサイクルは、何らかの発行アクティビティが発生するサイクルです。

このパラメータは、主に、発行アクティビティ後のイベントログテーブルのクリーンアップを可能にするために使用されます。

通常は、これらのステートメントで参照されるデータベースオブジェクト (テーブル、ストアドプロシージャ、関数など) を明示的にスキーマ修飾する必要があります。

次の表は、このパラメータのプロパティのリストです。

表 4-57 *ポストポーリングステートメント : プロパティ*

プロパティ	値
タグ名	post-poll-stmt
必須 ?	×
大文字と小文字を区別 ?	144 ページの 「区切りのない識別子での大文字と小文字の区別」を参照してください。

プロパティ	値
区切り文字	セミコロン
サンプル値	DELETE FROM direct.direct_process
デフォルト値	(なし)
有効な値	(有効な SQL ステートメントの任意のセット)
スキーマ依存	True

バッチサイズ

「バッチサイズ」パラメータは、1 つの発行ドキュメントで送信されるイベント数を指定します。

基本的に、バッチが大きいほどパフォーマンスが向上します。

- バッチが大きいと、ネットワーク上の双方向の移動回数が少なくて済みます。
- 1 つのドキュメント内のイベントが多いと、発行者チャネルから Identity Manager エンジン (クエリバックイベントが使用されないと仮定) への移動回数が少なくて済みます。
- バッチが大きいと、発行者チャネルからデータベースへの移動回数が最小限になります (サードパーティ製の JDBC ドライバおよびデータベースがバッチ処理をサポートしていると仮定)。
- バッチが大きいと、ローカルファイルシステムでの状態ファイルへのコミット数が少なくて済みます。

  コミットにもコストがかかる場合があります。

このパラメータは、上限を定義します。発行者チャネルは、指定した値を一定の条件下で上書きする可能性があります。Java ヒープのオーバーフローの可能性を最小化し、ドライバのシャットダウン時に発行者スレッド終了が遅延するのを短縮するために、上限として 128 が選択されています。

次の表は、このパラメータのプロパティのリストです。

表 *4-58* *バッチサイズ : プロパティ*

プロパティ	値
タグ名	batch-size
必須 ?	×
デフォルト値	1
有効な値	1 ～ 128
スキーマ依存	True

ハートビート間隔 (分)

「ハートビート間隔 (分)」パラメータは、発行者チャネルが非アクティブになってからハートビートドキュメントを送信するまでに待機する時間 (分) を指定します。実際には、指定した分数よりも長い時間が経過することがあります。つまり、このパラメータは下限を定義します。発行者チャネルは、それが非アクティブ状態になってから指定した時間 (分) が経過した場合にだけ、ハートビートドキュメントを送信します。送信される発行ドキュメントは、実質上、ハートビートドキュメントです。

次の表は、このパラメータのプロパティのリストです。

表 *4-59* ハートビート間隔 *(分):* プロパティ

プロパティ	値
タグ名	pub-heartbeat-interval
必須？	×
デフォルト値	0
有効な値	0 ～ 2,147,483,647 (java.lang.Integer.MAX_VALUE)
スキーマ依存	False

4.6 トレースレベル

ドライバからのデバッグ出力を表示するには、ドライバインスタンスを含むドライバセットに DirXML-DriverTraceLevel 属性値 1 ～ 7 を追加します。この属性は、よく DirXML-XSL TraceLevel 属性と混同されます。ドライバセットのトレースレベルの詳細については、『*Novell Identity Manager 3.0* 管理ガイド』を参照してください。

ドライバでは、次の 7 つのトレースレベルをサポートしています。

表 *4-60* サポートされているトレースレベル

レベル	説明
1	最小限のトレース
2	データベースプロパティ
3	接続ステータス、SQL ステートメント、イベントログレコード
4	詳細出力
5	データベースリソースの割り当て / 割り当て解除、状態ファイルの内容
6	JDBC API (呼び出されるメソッド、渡される引数、返り値など)
7	サードパーティ製のドライバ

レベル 6 および 7 は、サードパーティ製のドライバをデバッグする場合に特に便利です。

4.7 サードパーティ製の JDBC ドライバの設定

サードパーティ製のドライバを設定するときは、次のガイドラインに従います。特定の環境設定の方法については、サードパーティ製のドライバのマニュアルを参照してください。

- 最新バージョンのドライバを使用します。
- サードパーティ製のドライバの動作を設定できる場合があります。

  多くの場合、非互換の問題は、ドライバの JDBC URL プロパティを調整することで解決できます。
- 国際文字を使用する場合は、通常、サードパーティ製のドライバに、データベースが使用する文字エンコードを明示的に指定する必要があります。

  このためには、ドライバの JDBC URL の最後にプロパティ文字列を追加します。

  プロパティは、通常、プロパティキーワードと文字エンコード値 (たとえば、jdbc:odbc:mssql;charSet=Big5) で構成されます。プロパティキーワードは、サードパーティ製のドライバによって異なる場合があります。

  使用できる文字エンコードは、Sun によって定義されています。詳細については、Sun の「Supported Encodings」Web サイト (http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html) を参照してください。

次の表は、最大のドライバ互換性を得るための推奨設定のリストです。これらの設定は、初期環境設定中に、サポートされていないサードパーティ製のドライバを使用する場合に便利です。

表 *4-61* サードパーティ製の *JDBC* ドライバの推奨設定

パラメータ名	互換性を維持するための値
同期フィルタ	empty
ステートメントを再使用しますか？	0 (いいえ)
手動トランザクションを使用しますか？	0 (いいえ)
最小数の接続を使用しますか？	yes
最小限のメタデータを取得しますか？	1 (はい)
返された結果セットの数	one

詳細環境設定

5

サンプルのドライバ環境設定をインストールしたら、特定の用途に合わせてカスタマイズします。

5.1 スキーママッピング

次の表は、ドライバによる Novell® アイデンティティボールトブジェクトからデータベースオブジェクトへのマッピングの概要を示しています。

表 *5-1* アイデンティティボールトブジェクトからデータベースオブジェクトへのマッピング

アイデンティティボールトブジェクト	データベースオブジェクト
ツリー	スキーマ
クラス	テーブル / ビュー
属性	列
関連付け	プライマリキー

5.1.1 論理データベースクラス

論理データベースクラスは、データベース内で eDirectory™ クラスを表現するために使用されるテーブルのセットまたはビューです。論理データベースクラスは、1 つのビュー、または 1 つの親テーブルと子テーブル (0 個以上) のセットで構成できます。

論理データベースクラス名は、親テーブルおよびビューの名前です。

5.1.2 間接同期

間接同期モデルでは、ドライバは次のようにマップします。

表 *5-2* 間接同期でのマッピング

アイデンティティボールトブジェクト	データベースオブジェクト
クラス (複数)	テーブル (複数)
属性	列

アイデンティティボールトブジェクト	データベースオブジェクト
1 つのクラス	1 つの親テーブル および 0 以上の子テーブル
単一値属性	親テーブルの列
複数値属性	親テーブルの列 (区切られた値を保持) または 子テーブルの列 (推奨)

eDirectory クラスから論理データベースクラスへのマッピング

次の例では、論理データベースクラス usr が、次で構成されます。

- 1 つの親テーブル usr
- 2 つの子テーブル : usr_phone および usr_faxno.

論理クラス usr は、eDirectory クラス User にマップされます。

```
CREATE TABLE indirect.usr ( idu         INTEGER  NOT NULL, fname
VARCHAR2(64), lname       CHAR(64), pwdminlen   NUMBER(4), pwdexptime
DATE, disabled    NUMBER(1), username    VARCHAR2(64), loginame
VARCHAR2(64), photo       LONG RAW, manager     INTEGER, CONSTRAINT
pk_usr_idu    PRIMARY KEY (idu), CONSTRAINT fk_usr_manager FOREIGN KEY
(manager)         REFERENCES indirect.usr(idu) )


CREATE TABLE indirect.usr_phone ( idu      INTEGER       NOT NULL,
phoneno  VARCHAR2(64)  NOT NULL, CONSTRAINT fk_phone_idu FOREIGN KEY
(idu)         REFERENCES indirect.usr(idu) )


CREATE TABLE indirect.usr_fax ( idu    INTEGER       NOT NULL, faxno
VARCHAR2(64)  NOT NULL, CONSTRAINT fk_fax_idu FOREIGN KEY (idu)
        REFERENCES indirect.usr(idu) )


<rule name="Schema Mapping Rule"> <attr-name-map> <class-name> <nds-
name>User</nds-name> <app-name>indirect.usr</app-name> </class-name>
<attr-name class-name="User"> <nds-name>Given Name</nds-name> <app-
name>fname</app-name> </attr-name> <attr-name class-name="User"> <nds-
name>Surname</nds-name> <app-name>lname</app-name> </attr-name> <attr-
name class-name="User"> <nds-name>Password Expiration Time</nds-name>
<app-name>pwdexptime</app-name> </attr-name> <attr-name class-
name="User"> <nds-name>jpegPhoto</nds-name> <app-name>photo</app-name>
</attr-name> <attr-name class-name="User"> <nds-name>manager</nds-
name> <app-name>manager</app-name> </attr-name> <attr-name class-
name="User"> <nds-name>Password Minimum Length</nds-name> <app-
```

```
name>pwdminlen</app-name> </attr-name> <attr-name class-name="User">
<nds-name>Facsimile Telephone Number</nds-name> <app-
name>usr_fax.faxno</app-name> </attr-name> <attr-name class-
name="User"> <nds-name>Telephone Number</nds-name> <app-
name>usr_phone.phoneno</app-name> </attr-name> <attr-name class-
name="User"> <nds-name>Login Disabled</nds-name> <app-name>disabled</
app-name> </attr-name> </attr-name-map> </rule>
```

親テーブル

親テーブルは、1 つ以上の列を含み、明示的なプライマリキー制約を持つテーブルです。親テーブルには、ドライバが関連付けの値に含めるフィールドを認識できるように、明示的なプライマリキー制約が必要です。

```
CREATE TABLE indirect.usr ( idu  INTEGER  NOT NULL, -- ...CONSTRAINT
pk_usr_idu PRIMARY KEY (idu) )
```

次の表に、indirect.usr テーブルのサンプルデータを示します。

idu	fname	lname
1	John	Doe

この行の関連付けの結果は次のようになります。

idu=1,table=usr,schema=indirect

注 **:** 関連付けの値にあるデータベース識別子の大文字と小文字は、ランタイム時にデータベースメタデータから動的に決定されます。

親テーブルの列

親テーブルの列には、値を 1 つだけ含めることができます。したがって、単一値の eDirectory 属性のマッピングに適しています。たとえば、単一値の eDirectory 属性である Password Minimum Length を単一値の親テーブルの列 pwdminlen にマップします。

親テーブルの列には、暗黙的にスキーマ名と親テーブル名のプリフィックスが付けられます。したがって、明示的にテーブルのプリフィックスを付ける必要はありません。たとえば、indirect.usr.fname は、スキーママッピングの場合は、fname と同等です。

```
<rule name="Schema Mapping Rule"> <attr-name-map> <class-name> <nds-
name>User</nds-name> <app-name>indirect.usr</app-name> </class-name>
<attr-name class-name="User"> <nds-name>Given Name</nds-name> <app-
name>fname</app-name> </attr-name> </attr-name-map> </rule>
```

ラージバイナリおよび文字列データタイプは、通常、親テーブルの列にマップする必要があります。子テーブルの列にマップする場合は、データタイプは SQL ステートメントで比較できるものである必要があります。ラージデータタイプは、通常、SQL ステートメントでは比較できません。

ラージバイナリおよび文字列データタイプは、次の場合に子テーブルの列にマップできます。

- これらのタイプでの各 <remove-value> イベントが、ポリシーで、<remove-all-values> 要素に変換される。
- 各 <remove-value> イベントの後に <add-value> 要素が続く。

子テーブル

子テーブルは、その親テーブルのプライマリキーの外部キー制約を持つテーブルです。この制約で 2 つのテーブルをリンクします。子テーブルの外部キーを含む列は、親テーブルのプライマリキーの列とは異なる名前にできます。

次の例は、親テーブル usr と子テーブル usr_phone および usr_faxno との関係を示します。

```
CREATE TABLE indirect.usr (     idu  INTEGER  NOT NULL, --
...CONSTRAINT pk_usr_idu  PRIMARY KEY (idu) )


CREATE TABLE indirect.usr_phone ( idu      INTEGER       NOT NULL,
phoneno  VARCHAR2(64)  NOT NULL, CONSTRAINT fk_phone_idu FOREIGN KEY
(idu)          REFERENCES indirect.usr(idu) )


CREATE TABLE indirect.usr_fax ( idu    INTEGER       NOT NULL, faxno
VARCHAR2(64)  NOT NULL,   CONSTRAINT fk_fax_idu FOREIGN KEY (idu)
        REFERENCES indirect.usr(idu) )
```

注 **:** 子テーブルではすべての列を NOT NULL で制約します。

子テーブルの最初の制約列は、親テーブルを識別します。前の例では、子テーブル usr_phone の制約列は idu です。この列の目的は、テーブル usr_phone および usr を関係付けることだけです。制約列には有用な情報は含まれていないので、発行トリガおよびスキーママッピングポリシーからは除外します。

重要な列は制約のない列です。これは、単一の複数値属性を示します。前の例で、制約のない列は phoneno および faxno です。制約のない列は複数の値を保持できるので、複数値の eDirectory 属性のマッピングに適しています (たとえば、複数値の eDirectory 属性である Telephone Number から usrphone.phoneno へのマッピング)。

次の表に、indirect.usr_phone のサンプルデータを示します。

表 *5-3* サンプルデータ

idu	phoneno
1	111-1111
1	222-2222

親テーブルの列と同じように、子テーブルの列にも暗黙的にスキーマ名のプリフィックスが付けられています。ただし、親テーブルの列とは異なり、子テーブルの列には、明示的に子テーブル名のプリフィックスを付ける必要があります (たとえば、usr_phone.phoneno)。そうしないと、ドライバは、子テーブルの列 usr_phone.phoneno ではなく、phoneno 列 (親テーブルの列) を暗黙的に usr.phoneno と解釈します。

```
<rule name="Schema Mapping Rule"> <attr-name-map> <class-name> <nds-
name>User</nds-name> <app-name>indirect.usr</app-name> </class-name>
<attr-name class-name="User"> <nds-name>Facsimile Telephone Number</
nds-name> <app-name>usr_fax.faxno</app-name> </attr-name> <attr-name
class-name="User"> <nds-name>Telephone Number</nds-name> <app-
name>usr_phone.phoneno</app-name> </attr-name> </attr-name-map> </
rule>
```

注 **:** 個々の複数値の eDirectory 属性を異なる子テーブルにマップします。

参照属性

外部キー制約を使用して、データベースの参照の包含を表現できます。参照属性は、論理データベースクラス内の列であり、同一論理データベースクラスまたは他の論理データベースクラス内の親テーブルのプライマリキー値を参照します。

単一値の参照属性

親テーブルの単一値の列を使用して、2 つの親テーブルを関係付けることができます。この列は、他方の親テーブルのプライマリキーをポイントする外部キー制約を含む必要があります。次の例では、単一の親テーブル usr をそれ自体に関係付けます。

```
CREATE TABLE indirect.usr ( idu      INTEGER  NOT NULL, -- ... manager
INTEGER, CONSTRAINT pk_usr_idu     PRIMARY KEY (idu), CONSTRAINT
fk_usr_manager FOREIGN KEY (manager)         REFERENCES
indirect.usr(idu) )
```

注 **:** 単一値の参照列は、NULL 可能である必要があります。

```
<rule name="Schema Mapping Rule"> <attr-name-map> <class-name> <nds-
name>User</nds-name> <app-name>indirect.usr</app-name> </class-name>
<attr-name class-name="User"> <nds-name>manager</nds-name> <app-
name>manager</app-name> </attr-name> </attr-name-map> </rule>
```

この例は、ユーザごとにマネージャ (ユーザの 1 人) を 1 人だけ指定できると解釈されます。

複数値の参照属性

共通の子テーブルを使用して、2 つの親テーブルを関係付けることができます。この子テーブルには、他方の親テーブルのプライマリキーをポイントする外部キーの制約がある列を含める必要があります。次の例は、親テーブル usr および grp を、共通の子テーブル member を通じて関係付けます。

```
CREATE TABLE indirect.usr ( idu  INTEGER  NOT NULL, -- ...CONSTRAINT
pk_usr_idu PRIMARY KEY (idu) )


CREATE TABLE indirect.grp ( idg  INTEGER  NOT NULL, -- ...CONSTRAINT
pk_grp_idg PRIMARY KEY (idg) )


CREATE TABLE indirect.grp_member ( idg  INTEGER  NOT NULL, idu  INTEGER
NOT NULL, CONSTRAINT fk_member_idg FOREIGN KEY (idg)      REFERENCES
indirect.grp(idg),    CONSTRAINT fk_member_idu FOREIGN KEY (idu)
REFERENCES indirect.usr(idu) )
```

注 **:** 子テーブルのすべての列に NOT NULL 制約があります。

```
<rule name="Schema Mapping Rule"> <attr-name-map> <class-name> <nds-
name>Group</nds-name> <app-name>indirect.grp</app-name> </class-name>
<class-name> <nds-name>User</nds-name> <app-name>indirect.usr</app-
name> </class-name> <attr-name class-name="Group"> <nds-name>Member</
nds-name> <app-name>grp_member.idu</app-name> </attr-name> </attr-
name-map> </rule>
```

子テーブルの 1 つ目の制約列は、子テーブル grp_member が属する論理データベースクラスを指定します。前の例では、grp_member は、論理データベースクラス grp の一部であると見なされます。grp_member は、grp の適切な子であることが示されます。子テーブルの 2 つ目の制約列は、複数値の参照属性です。

次の例では、grp_member が usr クラスに含まれるように、制約列の順序が逆になっています。より正確に関係を反映するように、grp_member テーブルの名前が usr_mbr_of に変更されています。

```
CREATE TABLE indirect.usr ( idu  INTEGER  NOT NULL, -- ...CONSTRAINT
pk_usr_idu PRIMARY KEY (idu) )


CREATE TABLE indirect.grp ( idg  INTEGER  NOT NULL, -- ...CONSTRAINT
pk_grp_idg PRIMARY KEY (idg) )


CREATE TABLE indirect.usr_mbr_of ( idu  INTEGER  NOT NULL, idg  INTEGER
NOT NULL, CONSTRAINT fk_mbr_of_idu FOREIGN KEY (idu)
        REFERENCES indirect.usr(idu) ON DELETE CASCADE,    CONSTRAINT
fk_mbr_of_idg FOREIGN KEY (idg)         REFERENCES indirect.grp(idg)
ON DELETE CASCADE )


<rule name="Schema Mapping Rule"> <attr-name-map> <class-name> <nds-
name>Group</nds-name> <app-name>indirect.grp</app-name> </class-name>
<class-name> <nds-name>User</nds-name> <app-name>indirect.usr</app-
```

```
name> </class-name> <attr-name class-name="User"> <nds-name>Group
Membership</nds-name> <app-name>usr_mbr_of.idg</app-name> </attr-name>
</attr-name-map> </rule>
```

列位置を認識しないデータベース (DB2/AS400 など) では、順序は、文字列または 16 進の値で列名を並べ替えることによって決まります。詳細については、67 ページの 「列名を並べ替える基準」を参照してください。

一般に、双方向で複数値の参照属性だけを、両方のクラスではなくどちらか一方のクラスの一部として同期する必要があります。両方のクラスで参照属性を同期する場合は、2 つ (各クラスに 1 つずつ) の子テーブルを作成します。たとえば、eDirectory 属性である Group Membership および Member を同期する場合は、2 つの子テーブルを作成します。

実際には、User クラスおよび Group クラスを同期するときに、グループクラスの Member 属性の代わりに、ユーザクラスの Group Membership 属性を同期することをお勧めします。通常はユーザのグループメンバーシップを同期する方が、グループのすべてのメンバーシップを同期するよりも効率的です。

5.1.3 直接同期

直接同期モデルでは、ドライバは次のようにマップします。

表 *5-4* 直接同期でのマッピング

アイデンティティボールトブジェクト	データベースオブジェクト
クラス (複数)	ビュー (複数)
属性 (複数)	ビューの列 (複数)
クラス (単数)	ビュー (単数)
単一値属性	ビューの列 (単数)
複数値属性	ビューの列 (単数)

ビューの更新機能は、データベースによってさまざまです。ほとんどのデータベースでは、ビューが 1 つだけのベーステーブルで構成されている (複数のテーブルを連結しないビュー) 場合は更新できます。ビューが完全に読み取り専用である場合は、購読には使用できません。一部のデータベースでは、instead-of-trigger でビューに更新ロジックを定義できます。これによって、ビューで複数のベーステーブルを連結できます。更新も引き続き可能です。

instead-of-trigger をサポートするデータベースのリストについては、142 ページの 「データベースの機能」を参照してください。instead-of-trigger ロジックは、埋め込み SQL を使用するデータベース機能に関係なく、シミュレートできます。113 ページの 「仮想トリガ」を参照してください。

ビューの列のメタ識別子

ビューは論理テーブルです。テーブルとは異なり、ビューは物理的にはデータベースに存在しません。このため、通常、ビューは、従来からのプライマリキー / 外部キー制約を保

持できません。これらの構成体をシミュレートするために、JDBC 用ドライバは、制約およびその他のメタデータをビューの列名に埋め込みます。これらの制約と従来の制約は、前者がデータベースレベルで強制されない点が異なります。これらは、アプリケーションレベルの構成体です。

たとえば、関連付けの値を作成するときに使用するフィールドをドライバに明確に示すには、プライマリキー制約を親テーブルに配置します。ビューでは、この結果、1 つ以上の列名にプリフィックス pk_ (大文字と小文字の区別なし) が付けられます。

次の表は、ビューの列名に埋め込むことができる制約プリフィックスのリストです。

表 *5-5* 制約プリフィックス

制約プリフィックス (大文字と小文字の区別なし)	解釈
pk_	プライマリキー
fk_	外部キー
sv_	単一値
mv_	複数値

次のビューの例は、これらの制約プリフィックスをすべて含んでいます。

```
CREATE VIEW direct.view_usr ( pk_idu,            -- primary key
column; implicitly single-valued sv_fname,          -- single-valued
column mv_phoneno,        -- multi-valued column fk__idu__manager,  --
self-referential foreign key column; refers                        --
to primary key column idu in view_usr;                          --
implicitly single-valued fk_mv__idg__mbr_of -- extra-referential
foreign key column; refers                        --   to primary key
column idg in view_grp;                         --   multi-valued ) AS -
- ...


CREATE VIEW direct.view_grp ( pk_idg,           -- primary key column;
implicitly single-valued fk_mv__idu__mbr    -- extra-referential
foreign key column; refers --   to primary key column idu in view_usr;
--   multi-valued ) AS -- ...
```

BNF

ビューの列のメタ識別子を BNF (バッカス表記 (http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html)) で表記すると次のようになります。

```
<view-column-name> ::= [<meta-info>] <column-name>

<column-name> ::= <legal-unquoted-database-identifier> <meta-info> ::=
<referential> | <non-referential>

<non-referential> ::= [<single-value> | <multiple-value>]
```

```
<single-value> ::= "sv_"

<multiple-value> ::= "mv_"

<referential> ::= <primary-key> | <foreign-key>

<primary-key> ::= "pk_" [<single-value>] [<column-group-id>]
[<referenced-column-name>]


<column-group-id> ::= <non-negative-integer> "_"

<referenced-column-name> ::= "_" <column-name> "__"

<foreign-key> ::= "fk_" [<non-referential>] [<column-group-id>]
<referenced-column-name>
```

標準形式

デフォルトでは、すべてのビューの列名は単一値です。したがって、ビューの列名で sv_ プリフィックスを明示的に指定するのは冗長です。たとえば、sv_fname と fname は、形式は異なりますが等価であり、同じ列名を示します。

また、プライマリキー列の名前は、暗黙的にそれ自体を参照します。したがって、参照列名を指定するのは冗長です。たとえば、pk_idu は、pk__idu__idu と同等です。

JDBC 用ドライバは、ビューのメタ識別子の 2 つの標準形式を使用します。

- データベースのネイティブ形式

  データベースのネイティブ形式は、データベースで宣言されているとおりの列名です。この形式は、通常、スキーママッピング形式よりもさらに詳細で、必要なメタ情報をすべて含みます。
- スキーママッピング形式

  スキーママッピング形式は、ドライバがアプリケーションスキーマを返すときに返されます。この形式は、データベースのネイティブ形式よりも簡潔です。データベースのネイティブ形式に含まれるメタ情報のほとんどは、識別子ではなく XDS XML で表現されるからです。

  スキーママッピング形式で保持されるメタ情報は、参照プリフィックス pk_ および fk_ だけです。この制限によって、後方互換性が確実になります。

次の表は、各形式の例です。

表 *5-6* ビューのメタ識別子の標準形式の例

データベースのネイティブ形式	スキーママッピング形式
pk_idu	pk_idu
sv_fname	fname

データベースのネイティブ形式	スキーママッピング形式
mv_phoneno	phoneno
fk_mv__idg__mbr_of	fk_mbr_of

同等の形式

ビュー内で、メタ情報を含まない列名は、その「有効」な名前と呼ばれます。これは、ディレクトリオブジェクトの「有効な」権利と同様です。ドライバの場合、ビューの列名が同等かどうかは、デフォルトで、メタ情報に関係なく判断されます。たとえば、pk_idu は、idu と同等であり、fk_mv__idg__mbr_of は、mbr_of と同等です。ランタイム時に、さまざまな形式のビューのメタ列識別子をドライバに渡すことができます。後方互換性を維持するためには、メタ情報を有効なビューの列名として扱うことができます。64 ページの 「Enable Meta-Identifier Support? (メタ識別子のサポートを有効にしますか ?)」を参照してください。

プライマリキー列

プライマリキー列の名前は、同期スキーマ内のすべてのビューで一意である必要があります。

スキーママッピング

ビューとその列のスキーママッピングの規則は、親テーブルとその列で使用される規則と同じです。

5.1.4 プライマリキー列の同期

データベースがプライマリキー列の信頼されるソースである場合は、通常、発行者および購読者のフィルタ、スキーママッピングポリシー、および発行トリガで、列を省略します。

アイデンティティボールトがプライマリキー列の信頼されるソースである場合は、発行者フィルタおよびスキーママッピングポリシーには列を含めますが、購読者フィルタおよび発行トリガでは列を省略します。また、プライマリキーとして、CN ではなく GUID を使用することをお勧めします。CN は、複数値属性であり、変更されることがあります。GUID は、単一値であり、変更されません。

5.1.5 複数のクラスの同期

複数の eDirectory クラスを同期する場合は、各クラスを異なる親テーブルまたはビューに同期させます。各論理データベースクラスには、一意のプライマリキー列名が必要です。発行者チャネルは、この共通の列名を使用して、単一の論理データベースクラスに関連するイベントログテーブルのすべての列を識別します。次の例では、論理データベースクラス usr および grp の両方に、一意のプライマリキー列名があります。

```
CREATE TABLE usr ( idu   INTEGER     NOT NULL, lname VARCHAR2(64)  NOT
NULL, --...CONSTRAINT pk_usr_idu PRIMARY KEY(idu) );


CREATE TABLE grp ( idg  INTEGER  NOT NULL, --...CONSTRAINT pk_grp_idg
```

```
PRIMARY KEY(idg) );
```

5.1.6 複数値属性から単一値データベースフィールドへのマッピング

デフォルトで、ドライバは、親テーブルの列またはビューの列にマップされる eDirectory 属性はすべて単一値を持つと見なします。ドライバは eDirectory スキーマを認識しないので、eDirectory 属性が単一値と複数値のどちらを持っているかを判断する方法がありません。したがって、複数値と単一値の属性マッピングは、まったく同じように処理されます。

ドライバは、単一値の親テーブルまたはビューの列に関して、Most Recently Touched (MRT) アルゴリズムを実装します。MRT アルゴリズムを使用すると、最後に追加された属性値、または最後に削除された属性値がデータベースに保存されます。このアルゴリズムは、該当する属性値が単一値を持つ場合は適切です。

属性値が複数値を持つ場合は、好ましくない結果につながります。複数値属性から値が削除される場合、それがマップされているデータベースフィールドは、NULL に設定され、別の値が追加されるまで NULL のままになります。この望ましくない動作の解決方法として、単一値の属性だけが親テーブルまたはビューの列にマップされるように eDirectory スキーマを拡張することをお勧めします。

他にも、次のような解決法があります。

- 間接同期の場合は、個々の複数値属性をそれ自体の子テーブルにマップします。
- 直接および間接のどちらの同期でも、ポリシーを使用して、複数値を区切ってからテーブルまたはビューの列に挿入します。
- com.novell.nds.indirect.driver.jdbc.util.MappingPolicy クラスで提供されるメソッドを使用して、スタイルシートのレプリカポリシーごとに最初または最後の値を実装します。first-value-per-replica (FPR) ポリシーでは、eDirectory レプリカの最初の属性値が常に同期されます。last-value-per-replica (LPR) ポリシーでは、レプリカの最後の属性値が常に同期されます。グローバル環境設定値を使用することによって、サンプルのドライバ環境設定を、FPR または LPR マッピングポリシーを使用するように設定できます。複数値から単一値属性へのマッピングポリシーは、購読者コマンド変換ポリシーコンテナに含まれています。サンプルドライバ環境設定では、複数値の eDirectory 属性である Given Name および Surname を単一値の列 fname および lname にそれぞれマップします。

5.2 XDS イベントから SQL ステートメントへのマッピング

次の表は、間接同期で、購読者チャネルが XDS イベントを DML SQL ステートメントにどのようにマップするかをまとめています。

表 *5-7* 間接同期での *XDS* イベントのマッピング

XML イベント	同等の SQL
<add>	一致ポリシーに応じた 0 以上の選択ステートメント すべての単一値の <add-attr> 要素に対して 1 つの親テーブル挿入ステートメント 親テーブル挿入ステートメントの前または後にプライマリキー値を取得するための 0 または 1 つのストアドプロシージャ / 関数呼び出し 複数値の <add-attr> 要素ごとに 1 つの子テーブル挿入ステートメント
<modify>	単一値の <add-value> または <remove-value> 要素ごとに 1 つの親テーブル更新ステートメント 複数値の <add-value> 要素ごとに 1 つの子テーブル挿入ステートメント <remove-value> 要素ごとに 1 つの子テーブル削除ステートメント
<delete>	1 つの親テーブル削除ステートメント 子テーブルごとに 1 つの削除ステートメント
<query>	1 つの親テーブル選択ステートメント 子テーブルごとに 1 つの選択ステートメント
<move> <rename> <modify-password> <check-object-password>	埋め込み SQL ステートメントにバインドされていない場合は 0 ステートメント

次の表は、直接同期で、購読者チャネルが XDS イベントを DML SQL ステートメントにどのようにマップするかをまとめています。

表 *5-8* 直接同期での *XDS* イベントのマッピング

XML イベント	同等の SQL
<add>	一致ポリシーに応じた 0 以上の選択ステートメント すべての単一値の <add-attr> 要素に対して 1 つのビュー挿入ステートメント ビュー挿入ステートメントの前または後にプライマリキー値を取得するための 0 または 1 つのストアドプロシージャ / 関数呼び出し 複数値の <add-attr> 要素ごとに 1 つのビュー挿入ステートメント
<modify>	単一値の <add-value> または <remove-value> 要素ごとに 1 つのビュー更新ステートメント 複数値の <add-value> 要素ごとに 1 つのビュー挿入ステートメント <remove-value> 要素ごとに 1 つのビュー削除ステートメント
<delete>	1 つのビュー削除ステートメント
<query>	1 つのビュー選択ステートメント
<move> <rename> <modify-password> <check-object-password>	埋め込み SQL ステートメントにバインドされていない場合は 0 ステートメント

5.3 イベントログテーブル

イベントログテーブルは、発行イベントを保存します。この節では、イベントログテーブルの構造および機能について説明します。

イベントログテーブルとその列の名前をカスタマイズして、予約済みのデータベースキーワードとの衝突を避けることができます。ただし、その列の順序、番号、およびデータタイプは変更できません。列位置を認識しないデータベースでは、順序は「列名を並べ替える基準」パラメータによって決まります。67 ページの 「列名を並べ替える基準」を参照してください。

このテーブル内のイベントは、挿入順 (record_id 列)、または発生順 (event_time 列) のどちらかで並べることができます。イベントを発生順に並べると、イベント処理に時間がかかる可能性があります。発行イベントを発生順に並べるには、「将来のイベント処理を有効にしますか ?」パラメータを論理値の True に設定します。79 ページの 「将来のイベント処理を有効にしますか ?」を参照してください。

5.3.1 イベントログの列

この節では、イベントログテーブルの列について説明します。列は、位置の順に並べられています。

1. record_id

   record_id 列は、イベントログテーブル内の行を一意に識別したり、発行イベントを並べ替えたりするために使用されます。この列には、連続する一意の正の整数値を昇順に含める必要があります。record_id 値間で、連続する数値に抜けがあっても、ポーリングサイクルが早く終わることはなくなりました。

2. status

   status 列は、特定の行の状態を示します。次の表は、使用可能な値を示しています。

表 5-9 ステータス列に使用できる値

文字の値	解釈
N	新規
S	成功
W	警告
E	エラー
F	致命的

   処理されるためには、イベントログテーブルに挿入されるすべての行が status 値 N を保持する必要があります。その他のステータス文字は、発行者チャネルが処理済みの行を指定するためだけに使用されます。それ以外の文字すべて、将来の使用のために予約されています。

   注 **:** ステータス値では大文字と小文字が区別されます。

3. event_type

   この列の値は 1 ～ 8 にする必要があります。その他の番号はすべて将来の使用のために予約されています。

次の表は、各イベントタイプの説明です。

表 5-10 イベントタイプ

イベントタイプ	解釈
1	フィールドの挿入
2	フィールドの更新
3	フィールドの更新 (すべての値の削除)
4	行の削除
5	行の挿入 (クエリバック)
6	行の更新 (クエリバック)
7	フィールドの挿入 (クエリバック)
8	フィールドの更新 (クエリバック)

このフィールドの詳細については、103 ページのセクション 5.3.2「イベントタイプ」を参照してください。

4. event_time

   この列は、record_id の代わりの並べ替え基準の列として機能します。イベントの有効な日付が含まれます。また、NULL にすることはできません。この列を並べ替えの基準の列にするには、「将来のイベント処理を有効にしますか ?」パラメータを論理値の True に設定します。79 ページの 「将来のイベント処理を有効にしますか ?」を参照してください。

5. perpetrator

   この列は、イベントを発生させたデータベースユーザを識別します。NULL 値は、ドライバユーザ以外のユーザと解釈されます。したがって、NULL 値、またはドライバのデータベースユーザ名と等しくない値を含む行は、発行されます。ドライバのデータベースユーザ名と等しい値を持つ行は、「Allow Loopback Publisher (発行者のループバックを許可)」パラメータが論理値の True に発行されている場合以外は発行されません。82 ページの 「ループバックを許可しますか ?」を参照してください。

6. table_name

   イベントが発生したテーブルまたはビューの名前です。

7. table_key

   論理データベースクラスのすべてのトリガ内で、この列の値が正確に同じになるようにフォーマットします。このパラメータの BNF (バッカス表記 (http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html)) を次に定義します。

```
<table-key> ::= <unique-row-identifier> {"+"
                <unique-row-identifier>}

<unique-row-identifier> ::= <primary-key-column-name> "=" <value>
```

   たとえば、この章で参照される usr テーブルの場合、この列の値は、idu=1 になります。

この章で参照される view_usr ビューの場合、この列の値は、pk_empno=1 になります。

架空の複合プライマリキー (複数の列を含むもの) の場合、この列の値は、*pkey1=value1+pkey2=value2* になります。

注 **:** table_key フィールドに配置されるプライマリキー値に特殊文字 {, ; ' + " = \ < >} のいずれかが含まれている場合は、値を二重引用符で区切ります。「{」および「}」は特殊文字のセットを囲んでいます。また、一組の二重引用符内に二重引用符 " およびリテラルエスケープ文字の \ 文字が含まれる場合は、それぞれ \" および \\ としてエスケープする必要があります。

特殊文字を含む架空のプライマリキーの場合、この列の値は、pkey=", ; ' + \" = \\ < >" となります (二重引用符およびエスケープ文字に注意してください)。

注 **:** パディングまたはフォーマットの違いによって、out-of-order イベント処理になることがあります。パフォーマンスの理由から、数値内の不要な空白は削除します。たとえば、「idu= 1」よりも「idu=1」としてください (「idu= 1」には空白が含まれています)。

---

8. column_name

   変更された列の名前です。この列を使用するのはフィールド単位のイベントタイプ (1 ～ 3、7 ～ 8) だけですが、常にイベントログテーブルに存在する必要があります。この列がない場合、発行者チャネルを開始できません。

9. old_value

   フィールドの古い値です。この列を使用するのは、フィールド単位の非クエリバックのイベントタイプ (1 ～ 3) だけですが、常にイベントログテーブルに存在する必要があります。この列がない場合、発行者チャネルを開始できません。

10. new_value

    フィールドの新しい値です。この列を使用するのは、フィールド単位の非クエリバックのイベントタイプ (1 ～ 3) だけですが、常にイベントログテーブルに存在する必要があります。この列がない場合、発行者チャネルを開始できません。

5.3.2 イベントタイプ

次の表は、各イベントタイプの説明です。

表 *5-11* イベントタイプ

イベントタイプ	解釈
1	フィールドの挿入
2	フィールドの更新
3	フィールドの更新 (すべての値の削除)
4	行の削除
5	行の挿入 (クエリバック)
6	行の更新 (クエリバック)

イベントタイプ	解釈
7	フィールドの挿入 (クエリバック)
8	フィールドの更新 (クエリバック)

イベントタイプは、大きく 4 つのカテゴリに分けられます。一部のカテゴリは重複します。次の表は、各カテゴリとそれに該当するイベントタイプを示します。

表 *5-12* イベントカテゴリおよびタイプ

イベントカテゴリ	イベントタイプ
フィールド (属性) 単位	1, 2, 3, 7, 8
行 (オブジェクト) 単位	4, 5, 6
非クエリバック	1, 2, 3, 4
クエリバック	5, 6, 7, 8
フィールド単位、非クエリバック	1, 2, 3
フィールド単位、クエリバック	7, 8
行単位、非クエリバック	4
行単位、クエリバック	5, 6

通常は、容量、時間、実装の複雑さ、およびパフォーマンス間で最適な兼ね合いになるように各カテゴリのイベントタイプを組み合わせます。

フィールド単位のイベントタイプは、行単位のイベントタイプよりも、詳細で、多くの容量を必要とし、また実装が複雑です。行単位のイベントは、フィールド単位のイベントタイプよりも、大まかで、必要な容量が少なく、実装が簡単です。

クエリバックイベントタイプは、非クエリバックイベントタイプよりも使用する容量は少なくて済みますが、処理時間は長くなります。非クエリバックイベントタイプは、クエリバックイベントタイプよりも使用する容量は多いですが、処理時間は短くなります。

クエリバックイベントタイプは、非クエリバックイベントタイプよりも優先されます。非クエリバックイベントは、クエリバックイベントが同じフィールドまたはオブジェクトに対するログに記録されている場合は無視されます。たとえば、タイプ 2 (フィールドの更新、非クエリバック) および 8 (フィールドの更新、クエリバック) が同じフィールドにログ記録される場合、タイプ 2 イベントは無視され、タイプ 8 イベントが優先されます。

さらに、クエリバック行イベントタイプは、クエリバックフィールドイベントタイプよりも優先されます。たとえば、イベントタイプ 8 (フィールドの更新、クエリバック) およびイベントタイプ 6 (行の更新、クエリバック) が同じオブジェクトにログ記録される場合、タイプ 8 イベントは無視され、タイプ 6 イベントが優先されます。

データベースオブジェクトが存在しなくなると、発行者はクエリバックイベントを無視します。これらは、処理時間中にまだ存在しているデータベースオブジェクトに依存しま

す。したがって、ログ記録されたクエリバックの追加および変更 (イベントタイプ 5、6、7、8) は、参照しているデータベースオブジェクトが削除されると効果がなくなります。

次の表は、発行イベントタイプと発行者チャネルによって生成される XDS XML との基本的な対応関係を示します。

表 *5-13* 発行イベントタイプの基本的な対応関係

イベントタイプ	結果の XDS
挿入	<add>
更新	<modify>
削除	<delete>

次の例は、発行者チャネルが、発生する可能性があるイベントタイプごとに usr テーブルにログ記録されるイベントについて生成する XML を示します。

```
CREATE TABLE indirect.usr ( idu   INTEGER  NOT NULL, fname
VARCHAR2(64), photo LONGRAW, --...CONSTRAINT pk_usr_idu PRIMARY
KEY(idu) );
```

次の表は、新しい行が挿入された後の usr の最初の内容を示します。

表 *5-14* *usr* テーブルに挿入された行

idu	fname	lname	photo
1	Jack	Frost	0xAAAA

次の表は、行が更新された後の usr の現在の内容を示します。

表 *5-15* *usr* テーブルで更新された行

idu	fname	lname	photo
1	John	Doe	0xBBBB

フィールドの挿入

次の表は、新規行が usr テーブルに挿入された後のイベントログテーブルの内容を示します。photo 列の値は、Base64 でエンコードされています。0xAAAA を Base64 でエンコードした値は、qqo= です。

表 *5-16* イベントログテーブル : タイプ *1*

event_type	table	table_key	column_name	old_value	new_value
1	usr	idu=1	fname	NULL	Jack
1	usr	idu=1	lname	NULL	Frost
1	usr	idu=1	photo	NULL	qqo=

発行者チャネルは、次の XML を生成します。

```
<add class-name="usr"> <association>idu=1,table=usr,schema=indirect </
association> <add-attr attr-name="fname"> <value type="string">Jack</
value> </add-attr> <add-attr attr-name="lname"> <value
type="string">Frost</value> </add-attr> <add-attr attr-name="photo">
<value type="octet">qqo=</value> </add-attr> </add>
```

フィールドの更新

次の表は、usr テーブル内の行が更新された後のイベントログテーブルの内容を示します。photo 列の値は、Base64 でエンコードされています。0xBBBB を Base64 でエンコードした値は、u7s= です。

表 *5-17* イベントログテーブル : タイプ *2*

event_type	table	table_key	column_name	old_value	new_value
2	usr	idu=1	fname	Jack	John
2	usr	idu=1	lname	Frost	Doe
2	usr	idu=1	photo	qqo=	u7s=

発行者チャネルは、次の XML を生成します。

```
<modify class-name="usr"> <association>idu=1,table=usr,schema=indirect
</association> <modify-attr attr-name="fname"> <remove-value> <value
type="string">Jack</value> </remove-value> <add-value> <value
type="string">John</value> </add-value> </modify-attr> <modify-attr
attr-name="lname"> <remove-value> <value type="string">Frost</value>
</remove-value> <add-value> <value type="string">Doe</value> </add-
value> </modify-attr> <modify-attr attr-name="photo"> <remove-value>
<value type="octet">qqo=</value> </remove-value> <add-value> <value
type="octet">u7s=</value> </add-value> </modify-attr> </modify>
```

フィールドの更新 (すべての値の削除)

次の表は、usr テーブル内の行が更新された後のイベントログテーブルの内容を示します。photo 列の値は、Base64 でエンコードされています。

表 *5-18* イベントログテーブル : タイプ *3*

event_type	table	table_key	column_name	old_value	new_value
3	usr	idu=1	fname	Jack	John
3	usr	idu=1	lname	Frost	Doe
3	usr	idu=1	photo	qqo=	u7s=

発行者チャネルは、次の XML を生成します。

```
<modify class-name="usr"> <association>idu=1,table=usr,schema=indirect
</association> <modify-attr attr-name="fname"> <remove-all-values/>
<add-value> <value type="string">John</value> </add-value> </modify-
attr> <modify-attr attr-name="lname"> <remove-all-values/> <add-value>
<value type="string">Doe</value> </add-value> </modify-attr> <modify-
attr attr-name="photo"> <remove-all-values/> <add-value> <value
type="octet">u7s=</value> </add-value> </modify-attr> </modify>
```

行の削除

次の表は、usr テーブル内の行が削除された後のイベントログテーブルの内容を示します。

表 *5-19* イベントログテーブル : タイプ *4*

event_type	table	table_key	column_name	old_value	new_value
4	usr	idu=1	NULL	NULL	NULL

発行者チャネルは、次の XML を生成します。

```
<delete class-name="usr"> <association>idu=1,table=usr,schema=indirect
</association> </delete>
```

行の挿入 (クエリバック)

次の表は、新規行が usr テーブルに挿入された後のイベントログテーブルの内容を示します。

表 *5-20* イベントログテーブル : タイプ *5*

event_type	table	table_key	column_name	old_value	new_value
5	usr	idu=1	NULL	NULL	NULL

発行者チャネルは、次の XML を生成します。値は、usr テーブルの最初の内容ではなく現在の内容を反映しています。

```
<add class-name="usr"> <association>idu=1,table=usr,schema=indirect </
association> <add-attr attr-name="fname"> <value type="string">John</
value> </add-attr> <add-attr attr-name="lname"> <value
type="string">Doe</value> </add-attr> <add-attr attr-name="photo">
<value type="octet">u7s=</value> </add-attr> </add>
```

行の更新 (クエリバック)

次の表は、usr テーブル内の行が更新された後のイベントログテーブルの内容を示します。

表 *5-21* *イベントログテーブル : タイプ 6*

event_type	table	table_key	column_name	old_value	new_value
6	usr	idu=1	NULL	NULL	NULL

発行者チャネルは、次の XML を生成します。値は、usr テーブルの最初の内容ではなく現在の内容を反映しています。

```
<modify class-name="usr"> <association>idu=1,table=usr,schema=indirect
</association> <modify-attr attr-name="fname"> <remove-all-values/>
<add-value> <value type="string">John</value> </add-value> </modify-
attr> <modify-attr attr-name="lname"> <remove-all-values/> <add-value>
<value type="string">Doe</value> </add-value> </modify-attr> <modify-
attr attr-name="photo"> <remove-all-values/> <add-value> <value
type="octet">u7s=</value> </add-value> </modify-attr> </modify>
```

フィールドの挿入 (クエリバック)

次の表は、新規行が usr テーブルに挿入された後のイベントログテーブルの内容を示します。古い値および新しい値は、使用されないので省略されます。

表 *5-22* *イベントログテーブル : タイプ 7*

event_type	table	table_key	column_name	old_value	new_value
7	usr	idu=1	fname	NULL	NULL
7	usr	idu=1	lname	NULL	NULL
7	usr	idu=1	photo	NULL	NULL

発行者チャネルは、次の XML を生成します。値は、usr テーブルの最初の内容ではなく現在の内容を反映しています。

```
<add class-name="usr"> <association>idu=1,table=usr,schema=indirect </
association> <add-attr attr-name="fname"> <value type="string">John</
value> </add-attr> <add-attr attr-name="lname"> <value
```

```
type="string">Doe</value> </add-attr> <add-attr attr-name="photo">
<value type="octet">u7s=</value> </add-attr> </add>
```

フィールドの更新 (クエリバック)

次の表は、usr テーブル内の行が更新された後のイベントログテーブルの内容を示します。古い値および新しい値は、使用されないので省略されます。

表 5-23 *イベントログテーブル : タイプ 8*

event_type	table	table_key	column_name	old_value	new_value
8	usr	idu=1	fname	NULL	NULL
8	usr	idu=1	lname	NULL	NULL
8	usr	idu=1	photo	NULL	NULL

発行者チャネルは、次の XML を生成します。値は、usr テーブルの最初の内容ではなく現在の内容を反映しています。

```
<modify class-name="usr"> <association>idu=1,table=usr,schema=indirect
</association> <modify-attr attr-name="fname"> <remove-all-values/>
<add-value> <value type="string">John</value> </add-value> </modify-
attr> <modify-attr attr-name="lname"> <remove-all-values/> <add-value>
<value type="string">Doe</value> </add-value> </modify-attr> <modify-
attr attr-name="photo"> <remove-all-values/> <add-value> <value
type="octet">u7s=</value> </add-value> </modify-attr> </modify>
```

5.4 XDS イベントへの SQL ステートメントの埋め込み

この節には、XDS イベントに SQL を埋め込む場合に役立つ情報が含まれています。

- 119 ページのセクション 5.4.12「埋め込み SQL を含むパスワード変更の実装」
- 120 ページのセクション 5.4.13「オブジェクトパスワードチェックの実装」
- 120 ページのセクション 5.4.14「ベストプラクティス」

すべての例では、次の indirect.usr テーブルを参照します。

```
CREATE TABLE indirect.usr ( idu   INTEGER  NOT NULL, fname
VARCHAR2(64), lname VARCHAR2(64),

    CONSTRAINT pk_usr_idu PRIMARY KEY(idu) );
```

埋め込み SQL を使用すると、SQL ステートメントを XDS 形式の XML ドキュメントに埋め込むことができます。埋め込み SQL ステートメントは、XDS イベントと組み合わせて、またはスタンドアロンで使用できます。埋め込み SQL ステートメントがスタンドアロンで使用される場合、埋め込み SQL の処理では、ドライバがターゲットデータベースのテーブル / ビューについて認識していなくてもかまいません。したがって、ドライバは、schema-unaware モードで実行できます。51 ページの 「同期フィルタ」を参照してください。埋め込み SQL をスタンドアロンで使用する場合は、関連付けを手動で確立する必要があります。ドライバによって確立されることはありません。

XDS イベントと組み合わせて使用する場合、埋め込み SQL は、仮想データベーストリガとして機能できます。テーブルにデータベーストリガをインストールして特定の SQL ステートメントの実行時に副次的な動作を実行できるのと同様に、埋め込み SQL は、特定の XDS イベントに対応して、データベースで副次的な動作を実行させることができます。

5.4.1 埋め込み SQL の一般的な使用

XDS イベントに SQL を埋め込むことによって、次を実行できます。

- データベースユーザまたは役割の作成
- ユーザパスワードの管理

  ユーザパスワードの設定、確認、または変更ができます。
- データベースユーザまたは役割の特権の管理

サンプルの環境設定ファイル JDBCv2.xml は、XDS イベントの副次的動作として、データベースユーザの作成、ユーザパスワードの管理、およびユーザ特権の管理を行う方法を示しています。データベースユーザアカウント管理を有効にするには、グローバル設定変数 (GCV)user-ddl を true に設定します。埋め込み SQL の例は、購読者チャネルの User DDL コマンド変換スタイルシートに含まれています。

5.4.2 埋め込み SQL の基本

- 111 ページの 「要素」
- 111 ページの 「ネームスペース」
- 111 ページの 「埋め込み SQL の例」

要素

SQL は、<jdbc:statement> および <jdbc:sql> 要素を通じて XDS イベントに埋め込まれます。<jdbc:statement> 要素には、1 つまたは複数の <jdbc:sql> 要素を含めることができます。

ネームスペース

この節で使用するネームスペースのプリフィックス jdbc は、XML ドキュメント外で参照される場合、暗黙的にネームスペース urn:dirxml:jdbc にバインドされます。

埋め込み SQL 要素および属性は、ネームスペースのプリフィックスを付けて使用する必要があります。プリフィックスがない場合、ドライバはそれらを認識しません。この節のすべての例で使用されているプリフィックスは jdbc です。実際には、ネームスペース値 urn:dirxml:jdbc にバインドされていれば、プリフィックスは何でもかまいません。

次の XML の例は、埋め込み SQL 要素を使用する方法および適切にネームスペースプリフィックスを付ける方法を示します。次の例では、ネームスペース宣言およびネームスペースプリフィックスが太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"> <add-attr
name="lname"> <value>Doe</value> </add-attr> </add> <jdbc:statement>
<jdbc:sql>UPDATE indirect.usr SET fname = 'John'</jdbc:sql> </
jdbc:statement> </input>
```

埋め込み SQL の例

次の XML の例は、<jdbc:statement> および <jdbc:sql> 要素の使用法とその解釈を示します。次の例では、埋め込み SQL 要素が太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"> <add-attr
name="lname"> <value>Doe</value> </add-attr> </add> <jdbc:statement>
<jdbc:sql>UPDATE indirect.usr SET fname = 'John'</jdbc:sql> </
jdbc:statement> </input>
```

購読者チャネルは、<add> イベントを 1 つまたは複数の INSERT ステートメントに解決するので、前の XML は、次のように解決されます。

```
SET AUTOCOMMIT OFF INSERT INTO indirect.usr(lname)VALUES('Doe');
COMMIT; --explicit commit UPDATE indirect.usr SET fname = 'John';
COMMIT; --explicit commit
```

5.4.3 トークンの置換

関連付けからフィールド値を解析することをユーザに要求する代わりに、購読者チャネルが埋め込み SQL ステートメントでのトークンの置き換えをサポートしています。次の例では、トークンとそれが参照する値が太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <modify class-name="usr">
<association>idu=1,table=usr,schema=indirect</association> <modify-
```

```
attr name="lname"> <add-value> <value>DoeRaeMe</value> </add-value> </
modify-attr> </modify> <jdbc:statement> <jdbc:sql>UPDATE indirect.usr
SET fname = 'John' WHERE idu = {$idu}</jdbc:sql> </jdbc:statement> </
input>
```

トークンのプレースホルダは、XSLT 属性値テンプレートの構文 {$*field-name*} に準拠する必要があります。また、参照される関連付け要素は、XDS ドキュメント内で <jdbc:statement> 要素より前に置くか、<jdbc:statement> 要素の子として存在する必要があります。または、関連付け要素を <jdbc:statement> 要素の子としてコピーする代わりに、関連付け要素を含む要素の src-entry-id を <jdbc:statement> 要素にコピーします。次の例では、両方の方法が太字で示されています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <modify class-name="usr">
<association>idu=1,table=usr,schema=indirect</association> <modify-
attr name="lname"> <add-value> <value>DoeRaeMe</value> </add-value> </
modify-attr> </modify> <jdbc:statement>
           <association>idu=1,table=usr,schema=indirect</association>
<jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE idu = {$idu}</
jdbc:sql> </jdbc:statement> </input>
```

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <modify class-name="usr" src-
entry-id="0"> <association>idu=1,table=usr,schema=indirect</
association> <modify-attr name="lname"> <add-value> <value>DoeRaeMe</
value> </add-value> </modify-attr> </modify> <jdbc:statement src-
entry-id="0"> <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
idu = {$idu}</jdbc:sql> </jdbc:statement> </input>
```

{*$field-name*} トークンは、関連付けの値のいずれかの命名 RDN 属性名を参照する必要があります。前の例では、命名属性は idu の 1 つだけです。

<add> イベントは、トークンを持つ埋め込み SQL ステートメントより前に関連付け要素を配置する必要がない唯一のイベントです。これは、関連付けがまだ作成されていないからです。さらに、トークンを使用する埋め込み SQL ステートメントは、<add> イベントの前ではなく、後に続ける必要があります。次に例を示します。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"> <add-attr
name="lname"> <value>Doe</value> </add-attr> </add> <jdbc:statement>
<jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE idu = {$idu}</
jdbc:sql> </jdbc:statement> </input>
```

機密情報の複写を防ぐには、{$$password} トークンを使用して、同じドキュメント内の <password> 要素の直前の内容を参照できます。次の例では、パスワードトークンとそれが参照する値が太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr">
<password>some password</password> <add-attr name="fname">
<value>John</value> </add-attr> <add-attr name="lname"> <value>Doe</
value> </add-attr> </add> <jdbc:statement> <jdbc:sql>CREATE USER jdoe
IDENTIFIED BY {$$password}</jdbc:sql> </jdbc:statement> </input>
```

さらに、「アプリケーションパスワード」パラメータで {$$$driver-password} として指定されているドライバのデータベース認証パスワードも参照できます。46 ページの 「アプリケーションパスワード」を参照してください。名前付きパスワードの置換はまだサポートされていません。

関連付け要素の場合と同様に、参照されるパスワード要素は、XDS ドキュメント内で、<jdbc:statement> 要素より前に置くか、<jdbc:statement> 要素の子として存在する必要があります。または、パスワード要素を <jdbc:statement> 要素の子としてコピーする代わりに、パスワード要素を含む要素の src-entry-id を <jdbc:statement> 要素にコピーします。次の例では、両方の方法が太字で示されています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr">
<password>some password</password> <add-attr name="fname">
<value>John</value> </add-attr> <add-attr name="lname"> <value>Doe</
value> </add-attr> </add> <jdbc:statement> <password>some password</
password> <jdbc:sql>CREATE USER jdoe IDENTIFIED BY {$$password}</
jdbc:sql> </jdbc:statement> </input>
```

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr" src-entry-
id="0"> <password>some password</password> <add-attr name="fname">
<value>John</value> </add-attr> <add-attr name="lname"> <value>Doe</
value> </add-attr> </add> <jdbc:statement src-entry-id="0">
<jdbc:sql>CREATE USER jdoe IDENTIFIED BY {$$password}</jdbc:sql> </
jdbc:statement> </input>
```

5.4.4 仮想トリガ

データベーストリガを、それをトリガするステートメントの前または後に開始できるのと同じように、埋め込み SQL は、トリガする XDS イベントの前または後に配置できます。次の例は、SQL を XDS イベントの前または後に埋め込む方法を示します。

仮想 before トリガ

```
<input xmlns:jdbc"urn:dirxml:jdbc"> <jdbc:statement>
          <association>idu=1,table=usr,schema=indirect</association>
          <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
                    idu = {$idu}</jdbc:SQL>     </jdbc:statement>
    <modify class-name="usr">
        <association>idu=1,table=usr,schema=indirect</association>
        <modify-attr name="lname">             <remove-all-values/>
            <add-value>                 <value>Doe</value>
            </add-value>         </modify-attr>     </modify> </input>
```

この XML は次のように解決されます。

```
SET AUTOCOMMIT OFF UPDATE indirect.usr SET fname = 'John' WHERE idu =
1; COMMIT; --explicit commit UPDATE indirect.usr SET lname = 'Doe'
WHERE idu = 1; COMMIT; --explicit commit
```

仮想 after トリガ

```
<input xmlns:jdbc"urn:dirxml:jdbc">     <modify class-name="usr">
        <association>idu=1,table=usr,schema=indirect</association>
        <modify-attr name="lname">             <remove-all-values/>
            <add-value>                  <value>Doe</value>
            </add-value>     </modify-attr>     </modify>
    <jdbc:statement>         <jdbc:sql>UPDATE indirect.usr SET fname =
'John' WHERE                                    idu = {$idu}</
jdbc:sql>     </jdbc:statement> </input>
```

この XML は次のように解決されます。

```
SET AUTOCOMMIT OFF UPDATE indirect.usr SET lname = 'Doe'  WHERE idu =
1; COMMIT; --explicit commit UPDATE indirect.usr SET fname = 'John'
WHERE idu = 1; COMMIT; --explicit commit
```

5.4.5 手動トランザクションと自動トランザクション

次の 2 つのカスタム属性を使用して、埋め込み SQL および XDS イベントを手動でグループ化することができます。

- jdbc:transaction-type
- jdbc:transaction-id

jdbc:transaction-type

この属性には、manual および auto の 2 つの値があります。デフォルトで、対象となる XDS イベントのほとんど (<add>、<modify>、および <delete>) は、暗黙的に手動トランザクションタイプに設定されます。手動設定により、XDS イベントは、1 つまたは複数の SQL ステートメントで構成されたトランザクションに解決できます。

デフォルトで、埋め込み SQ イベントは、自動トランザクションタイプに設定されます。DDL ステートメントなど、一部の SQL ステートメントは、通常、手動トランザクションに含めることができないからです。次の例では、属性が太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"
jdbc:transaction-type="auto"> <add-attr name="lname"> <value>Doe</
value> </add-attr> </add> <jdbc:statement> <jdbc:sql>UPDATE
indirect.usr SET fname = 'John' WHERE idu = {$idu}</jdbc:sql> </
jdbc:statement> </input>
```

この XML は次のように解決されます。

```
SET AUTOCOMMIT ON INSERT INTO indirect.usr(lname) VALUES('Doe'); --
implicit commit UPDATE indirect.usr SET fname = 'John' WHERE idu = 1; -
- implicit commit
```

jdbc:transaction-id

要素の jdbc:transaction-type 属性値が、デフォルトで、または明示的に manual に設定されていない場合、購読者チャネルは、この属性を無視します。次の XML は、手動トランザクションの例を示します。属性は太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"
jdbc:transaction-id="0"> <add-attr name="lname"> <value>Doe</value> </
add-attr> </add> <jdbc:statement jdbc:transaction-type="manual"
                    jdbc:transaction-id="0"> <jdbc:sql>UPDATE
indirect.usr SET fname = 'John' WHERE idu = {$idu}</jdbc:sql> </
jdbc:statement> </input>
```

この XML は次のように解決されます。

```
SET AUTOCOMMIT OFF INSERT INTO indirect.usr(lname) VALUES('Doe');
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1; COMMIT; --
explicit commit
```

5.4.6 トランザクション分離レベル

ステートメントのグループ化以外にも、トランザクションを使用して、データベース内のデータの整合性を維持することができます。トランザクションは、アクセスまたは変更が同時に行われないように、データをロックします。トランザクションの分離レベルによって、ロックの設定方法が決まります。通常は、ドライバが使用するデフォルトの分離レベルで十分なので、変更しないでください。

ただし、必要な場合は、カスタム属性 jdbc:isolation-level を使用して、トランザクション分離レベルを調整できます。java.sql.Connection パラメータは、インタフェースに有効な 5 つの値を定義します。java.sql.Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html) を参照してください。

- none
- read uncommitted
- read committed
- repeatable read
- serializable

ドライバのデフォルトトランザクション分離レベルは、記述子ファイルによって上書きされる場合を除いて、read committed です。手動トランザクションでは、jdbc:isolation-level 属性をトランザクションの最初の要素に配置します。この属性は、それ以降の要素では無視されます。次の例では、属性が太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"
jdbc:transaction-id="0"                        jdbc:isolation-
level="serializable"> <add-attr name="lname"> <value>Doe</value> </
add-attr> </add> <jdbc:statement jdbc:transaction-type="manual"
jdbc:transaction-id="0"> <jdbc:sql>UPDATE indirect.usr SET fname =
'John' WHERE idu = {$idu}</jdbc:sql> </jdbc:statement> </input>
```

この XML は次のように解決されます。

```
SET AUTOCOMMIT OFF SET TRANSACTION ISOLATION LEVEL SERIALIZABLE INSERT
INTO indirect.usr(lname) VALUES('Doe'); UPDATE indirect.usr SET fname
= 'John' WHERE idu = 1; COMMIT; -- explicit commit
```

5.4.7 ステートメントのタイプ

購読者チャネルは、埋め込み SQL ステートメントを実行しますが、その解釈はしません。JDBC 1 インタフェースは、さまざまなタイプの SQL ステートメントを実行するための複数のメソッドを定義します。次の表は、これらのメソッドを示します。

表 *5-24* *SQL* ステートメントを実行するメソッド

ステートメントのタイプ	実行されるメソッド
SELECT	java.sql.Statement.executeQuery(String query):java.sql.ResultSet
INSERT	java.sql.Statement.executeUpdate(String update):int
UPDATE	java.sql.Statement.executeUpdate(String update):int
DELETE	java.sql.Statement.executeUpdate(String update):int
CALL または EXECUTE SELECT INSERT UPDATE DELETE	java.sql.Statement.execute(String sql):boolean

最も単純な解決方法は、すべての SQL ステートメントを java.sql.Statement.execute(String sql):boolean メソッドにマップする方法です。デフォルトで、購読者チャネルは、このメソッドを使用します。

一部のサードパーティ製のドライバ (特に Oracle' 製の JDBC ドライバ) では、このメソッドが生成する結果セットの数の決定に使用されるメソッドが正しく実装されません。このため、ドライバが無限ループに陥り、CPU の使用率が高くなる可能性があります。この問題を回避するには、任意の <jdbc:statement> 要素で jdbc:type 属性を使用して、それに含まれる SQL ステートメントを、デフォルトメソッドではなく次のメソッドにマップします。

- java.sql.Statement.executeQuery(String query):java.sql.ResultSet
- java.sql.Statement.executeUpdate(String update):int

jdbc:type 属性には、update と query の 2 つの値があります。INSERT、UPDATE、または DELETE ステートメントの場合、値を update に設定します。SELECT ステートメントの場合、値を query に設定します。この属性がない場合、ドライバはすべての SQL ステートメントをデフォルトメソッドにマップします。<jdbc:statement> 以外の要素に配置された場合、この属性は無視されます。

次のように指定することをお勧めします。

- すべての SELECT ステートメントに、jdbc:type="query" 属性値を配置します。

- すべての INSERT、UPDATE、および DELETE ステートメントに、jdbc:type=”update” 属性値を配置します。
- ストアドプロシージャ / 関数呼び出しには属性値を配置しません。

次の XML は、jdbc:type 属性の例を示します。属性は太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"> <add-attr
name="lname"> <value>Doe</value> </add-attr> </add> <jdbc:statement
jdbc:type="update"> <jdbc:sql>UPDATE indirect.usr SET fname = 'John'
WHERE idu = {$idu}</jdbc:sql> </jdbc:statement> </input>
```

5.4.8 SQL クエリ

データベースのクエリ機能を完全にサポートし、また難しいネイティブ SQL クエリの XDS 形式への変換を回避するために、ドライバは、ネイティブ SQL クエリ処理をサポートしています。選択ステートメントは、その他の SQL ステートメントとまったく同じ方法で XDS ドキュメントに埋め込むことができます。

たとえば、usr テーブルに、次が含まれているとします。

表 *5-25* 内容の例

idu	fname	lname
1	John	Doe

次の XML ドキュメントの結果、1 つの結果セットを含む出力ドキュメントが得られます。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <jdbc:statement
jdbc:type="query"> <jdbc:sql>SELECT * FROM indirect.usr</jdbc:sql> </
jdbc:statement> </input>
```

```
<output xmlns:jdbc="urn:dirxml:jdbc"> <jdbc:result-set jdbc:number-of-
rows="1"> <jdbc:row jdbc:number="1"> <jdbc:column jdbc:name="idu"
jdbc:position="1"  jdbc:type="java.sql.Types.BIGINT <jdbc:value>1</
jdbc:value> </jdbc:column>             <jdbc:column jdbc:name="fname"
jdbc:position="2" jdbc:type="java.sql.Types.VARCHAR>
<jdbc:value>John</jdbc:value> </jdbc:column> <jdbc:column
jdbc:name="lname" jdbc:position="3"
                          jdbc:type="java.sql.Types.VARCHAR>
<jdbc:value>Doe</jdbc:value> </jdbc:column> </jdbc:row> </jdbc:result-
set> <status level="success"/> </output>
```

SQL クエリは、結果セットに行が含まれているかどうかに関係なく、常に単一の <jdbc:result-set> 要素を生成します。結果セットが空の場合、jdbc:number-of-rows 属性はゼロに設定されます。

ドキュメントには複数のクエリを埋め込むことができます。SQL クエリでは、同期スキーマ内の参照されるテーブル / ビューがドライバに表示できなくてもかまいません。ただし、XDS クエリでは表示できる必要があります。

5.4.9 DDL(Data Definition Language) ステートメント

一般に、データベーストリガで DDL(Data Definition Language) ステートメントを実行することはできません。これは、ほとんどのデータベースでは、DML と DDL のトランザクションを混在させることができないからです。仮想トリガでは、このトランザクションの制限を克服はできませんが、DDL ステートメントを XDS イベントの副次的動作として実行させることはできます。

次に例を示します。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"> <add-attr
name="fname"> <value>John</value> </add-attr> <add-attr name="lname">
<value>Doe</value> </add-attr> </add> <jdbc:statement>
<jdbc:sql>CREATE USER jdoe IDENTIFIED BY novell</jdbc:sql> </
jdbc:statement> </input>
```

この XML は次のように解決されます。

```
SET AUTOCOMMIT OFF INSERT INTO indirect.usr(fname, lname)
VALUES('John', 'Doe'); COMMIT; -- explicit commit SET AUTOCOMMIT ON
CREATE USER jdoe IDENTIFIED BY novell; -- implicit commit
```

jdbc:transaction-id および jdbc:transaction-type 属性を使用して DML および DDL ステートメントを 1 つのトランザクションにまとめると、ほとんどのデータベースでトランザクションがロールバックされます。DDL ステートメントは、通常、個別のトランザクションとして実行されるので、前の例の insert ステートメントが成功し、create user ステートメントがロールバックされる可能性があります。

ただし、insert ステートメントが失敗し、create user ステートメントが成功することはありません。ドライバは、チェーンになったトランザクションの実行を、最初のトランザクションがロールバックされたところで停止します。

5.4.10 論理操作

通常は、DML および DDL ステートメントを 1 つのトランザクションに混在させることはできないので、1 つのイベントを 1 つ以上のトランザクションで構成することができます。jdbc:op-id および jdbc:op-type を使用すると、複数のトランザクションを 1 つの論理操作にまとめることができます。この方法でまとめた場合、操作のすべてのメンバがステータスに関係なく 1 つの単位として処理されます。1 つのメンバでエラーが発生した場合は、すべてのメンバが同じステータスレベルを返します。同様に、すべてのメンバが同じステータスタイプを共有します。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr" jdbc:op-
id="0" jdbc:op-type="password-set-operation"> <add-attr name="fname">
<value>John</value> </add-attr> <add-attr name="lname"> <value>Doe</
value> </add-attr> <password>Doe{$idu}</password> </add>
```

```
<jdbc:statement jdbc:op-id="0"> <jdbc:sql>CREATE USER jdoe IDENTIFIED
BY {$$password} </jdbc:sql> </jdbc:statement> </input>
```

jdbc:op-type 属性は、論理操作の最初の要素を除くすべての要素では無視されます。

5.4.11 埋め込み SQL を備えたパスワード設定の実装

パスワードの初期設定は、通常、データベースユーザアカウントの作成で行われます。<add> イベントが購読者チャネルで生成されるという前提で、パスワード設定を XDS <add> イベントの副次的動作として実装する XSLT スタイルシートによって生成される出力の例を次に示します。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr" jdbc:op-
id="0" jdbc:op-type="password-set-operation"> <add-attr name="fname">
<value>John</value> </add-attr> <add-attr name="lname"> <value>Doe</
value> </add-attr> <password>Doe{$idu}</password> </add>
<jdbc:statement jdbc:op-id="0"> <jdbc:sql>CREATE USER jdoe IDENTIFIED
BY {$$password} </jdbc:sql> </jdbc:statement> </input>
```

<add> イベントは、jdbc:op-id および jdbc:op-type 属性によって、論理的に CREATE USER DDL ステートメントにバインドされます。

サンプル環境設定ファイル JDBCv2.xml 内の User DDL コマンド変換スタイルシートには、ユーザアカウント作成 DDL ステートメントを、それをサポートするすべてのデータベースの <add> イベントにバインドするサンプル XSLT テンプレートが含まれています。

5.4.12 埋め込み SQL を含むパスワード変更の実装

パスワードの初期設定は、通常、既存のデータベースユーザアカウントの変更で行われます。<modify-password> イベントが購読者チャネルで生成されるという前提で、modify-password を実装する XSLT スタイルシートによって生成される出力の例を次に示します。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <modify-password jdbc:op-id="0"
jdbc:op-type="password-set-operation"> <password>new password</
password> </modify-password> <jdbc:statement jdbc:op-id="0">
<jdbc:sql>ALTER USER jdoe IDENTIFIED BY {$$password} </jdbc:sql> </
jdbc:statement> </input>
```

<modify-password> イベントは、jdbc:op-id および jdbc:op-type 属性によって、ALTER USER DDL ステートメントに論理的にバインドされます。

サンプル環境設定ファイル JDBCv2.xml の User DDL コマンド変換スタイルシートには、パスワード保守 DDL ステートメントを、それをサポートするすべてのデータベースの <modify-password> イベントにバインドするサンプル XSLT テンプレートが含まれています。

5.4.13 オブジェクトパスワードチェックの実装

パスワード設定とは異なり、オブジェクトパスワードチェックでは、埋め込み SQL ステートメントまたは属性は必要ありません。必要なのはユーザアカウント名だけです。これは、関連付けの値 (関連付けが手動で保守されていることが前提)、ディレクトリ属性、またはデータベースフィールドから取得できます。ディレクトリまたはデータベースに保存されている場合は、クエリを発行して値を取得する必要があります。

サンプル環境設定ファイル JDBCv2.xml は、データベースユーザアカウント名をデータベースフィールドに保存します。

注 **:** Sybase Adapative Server Enterprise や Microsoft SQL Server など、一部のデータベースでは、ユーザアカウント名とログインアカウント名が区別されます。したがって、1 つだけではなく 2 つの名前を保存することが必要になる場合があります。

オブジェクトパスワードチェックを実装するには、<check-object-password> イベントの最後に dest-dn 属性値を追加します。次の例では、dest-dn 属性が太字になっています。

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <check-object-password dest-
dn="jdoe"> <password>whatever</password> </check-object-password> </
input>
```

5.4.14 ベストプラクティス

パフォーマンスの理由から、複数のステートメントを XDS ドキュメントに埋め込むよりも、複数の SQL ステートメントを含む単一のストアドプロシージャ / 関数を呼び出すことをお勧めします。

次の例では、単一のストアドプロシージャまたは関数が使用されています。

単一のストアドプロシージャ

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"> <add-attr
name="fname"> <value>John</value> </add-attr> <add-attr name="lname">
<value>Doe</value> </add-attr> </add> <jdbc:statement> <jdbc:sql>CALL
PROCEDURE set_name('John', 'Doe')</jdbc:sql> </jdbc:statement> </
input>
```

複数の埋め込みステートメント

```
<input xmlns:jdbc="urn:dirxml:jdbc"> <add class-name="usr"> <add-attr
name="lname"> <value>Doe</value> </add-attr> </add> <jdbc:statement>
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John'
                  WHERE idu = {$idu}</jdbc:sql>     </jdbc:statement>
    <jdbc:statement>         <jdbc:sql>UPDATE indirect.usr SET lname =
'Doe'                    WHERE idu = {$idu}</jdbc:sql>     </
jdbc:statement> </input>
```

ストアドプロシージャまたは関数の呼び出しに使用される構文は、データベースによってさまざまです。詳細については、143 ページの 「ストアドプロシージャおよび関数の JDBC 呼び出しの構文」を参照してください。

サードパーティ製の JDBC ドライバ

6

6.1 サードパーティ製の JDBC ドライバの相互運用性

JDBC 用 Identity Manager ドライバは、データベースの特定のセットではなくサードパーティ製の JDBC ドライバの特定のセットと相互運用できるように設計されています。実際に、JDBC 用ドライバが特定のデータベースに対して機能するかどうかは、主として、データベースではなくサードパーティ製の JDBC ドライバによって決まります。原則として、JDBC 用ドライバが特定のサードパーティ製 JDBC ドライバと相互運用できる場合は、そのサードパーティ製のドライバがサポートするデータベースおよびデータベースバージョンとも相互運用できます。

サードパーティ製の JDBC ドライバを選ぶときは、できる限り、この節でリストしているような主要エンタープライズデータベースベンダのドライバを使用することを強くお勧めします。通常、これらは、無料であり、十分な開発およびチェックを経ています。また、JDBC 用ドライバおよびそのターゲットデータベースと問題なく相互運用できることが確認されています。他のサードパーティ製のドライバも使用できますが、Novell® ではサポートされていません。

一般に、ほとんどのサードパーティ製のドライバは、後方互換です。ただし、通常は後方互換であっても、前方互換ではありません。たいていの場合は、データベースサーバがアップグレードされるたびに、本製品と共に使用しているサードパーティ製のドライバも同様に更新する必要があります。

また、原則として、特に指示されている場合を除いて、最新バージョンのサードパーティ製のドライバを使用することをお勧めします。

6.2 JDBC ドライバのタイプ

Type 1

部分的に Java であり、ネイティブの ODBC ドライバを通じて間接的にデータベースサーバと通信するサードパーティ製の JDBC ドライバ。

Type 1 ドライバは、JDBC-ODBC ブリッジとして機能します。Sun は、実験的な使用のため、および他のタイプのサードパーティ製の JDBC ドライバを使用できない場合のために、JDBC-ODBC ブリッジドライバを提供しています。

Type 2

部分的に Java であり、そのネイティブクライアント API を通じて間接的にデータベースサーバと通信するサードパーティ製の JDBC ドライバ。

Type 3

Pure Java であり、ミドルウェアサーバを通じて間接的にデータベースサーバと通信するサードパーティ製の JDBC ドライバ。

Type 4

Pure Java であり、直接データベースサーバと通信するサードパーティ製の JDBC ドライバ。

6.2.1 使用するタイプの選択

Type 3 および 4 のドライバは、通常、Type 1 および 2 のドライバよりも安定しています。Type 1 および 2 のドライバは、通常、Type 3 および 4 のドライバよりも高速です。Type 2 および 3 のドライバは、通常、Type 1 および 4 のドライバよりも安全です。

Identity Manager はディレクトリをデータストアとして使用し、また、データベースは通常ディレクトリよりも著しく高速なので、パフォーマンスについてはそれほど考慮する必要はありません。しかし、安定性は重要です。この理由で、できる限り、Type 3 または 4 のサードパーティ製の JDBC ドライバを使用することをお勧めします。

重要 **:** Type 1 または 2 のドライバ (ネイティブコードを含むドライバ) を JDBC 用ドライバと連携使用することを選択する場合は、リモートローダを使用して、ディレクトリプロセスの整合性を確実にします。

6.3 サードパーティ製の Jar ファイルの配置

この節の表は、サードパーティ製の JDBC ドライバ jar ファイルを Identity Manager またはリモートローダサーバに配置する必要がある場所のパスを、デフォルトのインストールパスを基準にして示します。

6.3.1 Identity Manager のファイルパス

次の表は、Identity Manager サーバでのサードパーティ製の JDBC ドライバ jar ファイルの配置場所をプラットフォーム別に示します。

表 *6-1* *jar* ファイルの場所 *: Identity Manager* サーバ

プラットフォーム	ディレクトリパス
NetWare®	sys:\system\lib
Solaris、Linux、または AIX	/usr/lib/dirxml/classes (Directory 8.8 より前) /opt/novell/eDirectory/lib/dirxml/classes (eDirectory 8.8)
Windows NT/2000	novell\NDS\lib

6.3.2 リモートローダのファイルパス

次の表は、リモートローダサーバでのサードパーティ製の JDBC ドライバ jar ファイルの配置場所をプラットフォーム別に示します。

表 6-2 *jar* ファイルの場所 : リモートローダ

プラットフォーム	ディレクトリパス
Solaris、Linux、または AIX	/usr/lib/dirxml/classes (Directory 8.8 より前) /opt/novell/eDirectory/lib/dirxml/classes (eDirectory 8.8)
Windows NT/2000	novell\RemoteLoader\lib

6.4 サポートされているサードパーティ製の JDBC ドライバ

6.4.1 サードパーティ製の JDBC ドライバの機能

次の表は、サードパーティ製の JDBC ドライバの機能を示します。

表 6-3 サードパーティ製の *JDBC* ドライバの機能

ドライバ	暗号化された転送のサポート	自動生成されるキーの取得のサポート
BEA* Weblogic* jDriver	×	×
IBM DB2 UDB Type 3	×	×
IBM DB2 UDB Type 4	×	×
Informix	×	×
Microsoft 2000	×	×

ドライバ	暗号化された転送のサポート	自動生成されるキーの取得のサポート
MySQL Connector/J	○	○
Oracle Thin Client	○	×
PostgreSQL	○ *	×
Sybase jConnect	○	×

* JDBC 3 (Java 1.4) バージョン以降の場合。

6.4.2 JDBC URL の構文

次の表は、サポートされているサードパーティ製の JDBC ドライバの URL 構文をまとめたリストです。

表 *6-4* *URL* の構文

サードパーティ製の JDBC ドライバ	JDBC URL の構文
Oracle Thin Client	jdbc:oracle:thin:@*ip-address*:1521:*sid*
IBM DB2 UDB Type 3	jdbc:db2://*ip-address*:6789/*database-name*
IBM DB2 UDB Type 4, Universal	jdbc:db2://*ip-address:50000*/*database-name*
BEA Weblogic jDriver	jdbc:weblogic:mssqlserver4:*database-name*@*ip-address*:1433
Microsoft SQL Server	jdbc:microsoft:sqlserver://*ip-address-or-dns-name*:1433;DatabaseName=*database-name*
Sybase jConnect	jdbc:sybase:Tds:*ip-address*:2048/*database-name*
MySQL Connector/J	jdbc:mysql://*ip-address*:3306/*database-name*
Informix	jdbc:informix-sqli://*ip-address*:1526/*database-name*:informixserver=*server-id*
PostgreSQL	jdbc:postgresql://*ip-address*:5432/*database-name*

この情報は、「認証コンテキスト」パラメータと組み合わせて使用されます。このパラメータの詳細については、46 ページの 「認証コンテキスト」を参照してください。

6.4.3 JDBC ドライバクラス名

次の表は、サポートされているサードパーティ製の JDBC ドライバの完全修飾 Java クラス名をまとめたリストです。

表 *6-5* サードパーティ製の *JDBC* ドライバのクラス名

サードパーティ製の JDBC ドライバ	クラス名
BEA Weblogic jDriver	weblogic.jdbc.mssqlserver4.Driver
IBM DB2 UDB Type 3	COM.ibm.db2.jdbc.net.DB2Driver
IBM DB2 UDB Type 4, Universal	com.ibm.db2.jcc.DB2Driver
Informix	com.informix.jdbc.IfxDriver
Microsoft 2000	com.microsoft.jdbc.sqlserver.SQLServerDriver
MySQL Connector/J	org.gjt.mm.mysql.Driver
Oracle Thin Client	oracle.jdbc.driver.OracleDriver
PostgreSQL	org.postgresql.Driver
Sybase jConnect 5.5	com.sybase.jdbc2.jdbc.SybDriver

この情報は、「JDBC Driver Class Name (JDBC ドライバのクラス名)」パラメータと組み合わせて使用されます。このパラメータの詳細については、49 ページの 「サードパーティ製の JDBC ドライバのクラス名」を参照してください。

6.4.4 BEA Weblogic jDriver for Microsoft SQL Server

表 *6-6* *BEA Weblogic jDriver*

サポートされているデータベースバージョン	Microsoft SQL Server 6.5、7.x、8.x (2000)
クラス名	weblogic.jdbc.mssqlserver4.Driver
Type	4
URL の構文	jdbc:weblogic:mssqlserver4:*database-name*@*ip-address*:1433
ダウンロード方法	登録して (無料)、最新バージョンの Weblogic サーバをダウンロードします。インストーラを実行します。weblogic.jar ファイルは、*install-dir*/server/lib ディレクトリにインストールされます。 BEA Download Center (http://commerce.bea.com/showallversions.jsp?family=WLS)
ファイル名	weblogic.jar
マニュアルの URL	jDriver Documentation (http://e-docs.bea.com/wls/docs81/mssqlserver4/)

注 **:** BEA Weblogic ドライバは、サポートされているサードパーティ製のドライバのリストに含まれており、Microsoft SQL Server 7 への JDBC アクセスを提供します。Microsoft のドライバは、バージョン 8 (2000) だけをサポートします。

互換性

BEA Weblogic ドライバは、後方互換です。データベースサーバおよびドライバは、めったに更新されません。

セキュリティ

BEA Weblogic ドライバは、暗号化された転送をサポートしません。

既知の問題

- BEA Weblogic ドライバは有料です。購入および適切なライセンス契約が必要です。
- ドライバのバージョン間で、UNIQUEIDENTIFIER 列を含む関連付けの値の整合性が取れていません。

  以前のバージョンの BEA Weblogic ドライバは、ネイティブの UNIQUEIDENTIFIER 列の非標準の java.sql.Types (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) 値を返していました。これに対処するために、JDBC 用ドライバは、非標準タイプを標準タイプの java.sql.Types.BINARY (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) にマップしていました。これがネイティブデータベースタイプ (16 バイトの値) の反映として最適だからです。このマッピングにより、Base64 でエンコードされた関連付けの値が得られます。

  BEA Weblogic ドライバのそれ以降のバージョンでは、標準タイプの java.sql.CHAR (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) を返します。このマッピングにより、Base64 以外でエンコードされた関連付けの値が得られます。これによって、BEA Weblogic ドライバの以前のバージョンを使用して生成されたすべての関連付けが事実上無効になります。この変更によって、実質的に後方互換性が維持されなくなっています。

  この問題を解決する最適な方法は、以前のバージョンの BEA Weblogic ドライバを引き続き使用することです。アップグレードが必要な場合は、無効になる関連付けをすべて削除し、以前関連付けられていたオブジェクトをすべて再度関連付ける必要があります。
- BEA Weblogic ドライバでは、java.sql.Connection.getConnection(String url, String username, String password) メソッドが AIX で呼び出されたときに、java.lang.IllegalMonitorStateException (http://java.sun.com/j2se/1.5.0/docs/api/java/lang/IllegalMonitorStateException.html) が発生します。

6.4.5 IBM DB2 Universal Database JDBC ドライバ

IBM DB2 ドライバには、Type 3 または Type 4 があります。

Type 3

表 6-7 *IBM DB2 Driver: Type 3*

サポートされているデータベースバージョン	7.x
クラス名	COM.ibm.db2.jdbc.net.DB2Driver
Type	3

URL の構文	jdbc:db2://*ip-address*:6789/*database-name*
ダウンロード方法	データベースサーバから次のファイルをコピーします。 file:///*database-installation-directory*/java
ファイル名	db2java.zip
マニュアルの URL	DB2 Information Center (http://publib.boulder.ibm.com/infocenter/db2v7luw) JDBC Programming (http://publib.boulder.ibm.com/infocenter/db2v7luw/index.jsp?topic=/com.ibm.db2v7.doc/db2a0/db2a0159.htm)

重要 **:** Type 3 ドライバは、バージョン 8 では推奨されなくなりました。

互換性

IBM DB2 ドライバは、バージョン依存度が非常に高いドライバであると位置づけることができます。このドライバは、FixPack を含む DB2 のメジャーまたはマイナーバージョン間で互換性がありません。このため、データベースサーバにインストールされているファイルを使用することをお勧めします。

重要 **:** FixPack レベルであってもターゲットデータベースが更新されるたびに、IBM DB2 ドライバを Identity Manager またはリモートローダサーバで更新する必要があります。

セキュリティ

IBM DB2 ドライバは、暗号化された転送をサポートしません。

既知の問題

- バージョンが一致しない場合は、通常、コネクティビティ関連の障害が発生します。

  IBM DB2 ドライバで最もよく発生する問題は、ドライバとデータベースのバージョンの不一致により発生します。バージョンが一致していないと、「CLI0601E Invalid statement handle or statement is closed. (CLI0601E 無効なステートメントハンドルまたはステートメントが閉じられます。)」など、コネクティビティ関連の障害が発生します。問題を解決するには、Identity Manager またはリモートローダサーバで db2java.zip ファイルを、データベースサーバにインストールされているバージョンで上書きします。
- データベースサーバで発生した Java 関連のエラーの診断および解決は非常に困難です。

  Java で記述されたユーザ定義のストアドプロシージャおよび関数をインストールおよび実行しようとした場合、多数のエラー状況およびエラーコードが表示される可能性があります。それらを診断しても、時間がかかるだけで結局解決できない可能性があります。ログファイル (データベースサーバ上の db2diag.log) には、たいていの場合、追加のデバッグ情報が含まれています。また、すべてのエラーコードについて、ドキュメントが用意されており、オンラインで入手できます。

Type 4: Universal Driver

表 6-8 *IBM DB2 Driver: Type 4*

サポートされているデータベースバージョン	8.x
クラス名	com.ibm.db2.jcc.DB2Driver
Type	4
URL の構文	jdbc:db2://*ip-address*:50000/*database-name*
ダウンロード方法	最新の FixPack の一部としてダウンロードします (推奨)。 IBM Support & Downloads (http://www.ibm.com/support/us/) または データベースサーバから次のファイルをコピーします。 file:///*database-installation-directory*/java
ファイル名	db2jcc.jar, db2jcc_license_cu.jar, db2jcc_javax.jar (オプション)
マニュアルの URL	DB2 インフォメーションセンター (http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp) DB2 Universal JDBC ドライバ (http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp?topic=/com.ibm.db2.udb.doc/ad/t0010264.htm) DB2 Universal JDBC ドライバ使用時のセキュリティ (http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp?topic=/com.ibm.db2.udb.doc/ad/cjvjcsec.htm)

注 **:** Type 3 ドライバとは異なり、Type 4 ドライバは、定義済みのエラーコードの最小セットだけを保持しています。エラーコードセットが不足していることが、JDBC 用ドライバがコネクティビティ、再試行、認証、および致命的なエラー状況を区別する機能の妨げになります。

互換性

IBM DB2 ドライバは、後方互換です。ただし、データベースバージョン 7 では機能しません。データベースサーバは、頻繁に更新されます。ドライバはめったに更新されません。

セキュリティ

IBM DB2 ドライバは、さまざまな認証セキュリティメカニズムをサポートしますが、暗号化された転送はサポートしていません。

既知の問題

- データベースサーバで発生した Java 関連のエラーの診断および解決は非常に困難です。

Java で記述されたユーザ定義のストアドプロシージャおよび関数をインストールおよび実行しようとした場合、多数のエラー状況およびエラーコードが表示される可能性があります。それらを診断しても、時間がかかるだけで結局解決できない可能性があります。ログファイル (データベースサーバ上の db2diag.log) には、たいていの場合、追加のデバッグ情報が含まれています。また、すべてのエラーコードについて、ドキュメントが用意されており、オンラインで入手できます。

6.4.6 Informix JDBC Driver

表 6-9 *Informix JDBC Driver*

サポートされているデータベースバージョン	Dynamic Server 7.x、9.x
クラス名	com.informix.jdbc.IfxDriver
Type	4
URL の構文	jdbc:informix-sqli://*ip-address*:1526/*database-name*:informixserver=*server-id*
ダウンロード方法	ダウンロード URL (http://www-306.ibm.com/software/data/informix/tools/jdbc)
ファイル名	ifxjdbc.jar、ifxjdbcx.jar (オプション)
マニュアルの URL	Informix Information Center (http://publib.boulder.ibm.com/infocenter/ids9help/index.jsp) Informix JDBC Driver (http://www-306.ibm.com/software/data/informix/pubs/library/jdbc_2.html)

互換性

Informix ドライバは、後方互換です。データベースサーバおよびドライバは、めったに更新されません。

セキュリティ

Informix ドライバは、暗号化された転送をサポートしません。

ANSI 互換のデータベースの場合に必須のパラメータ設定

次の表は、ANSI 互換のデータベースに対して Informix ドライバと相互運用するときに、JDBC 用ドライバで明示的に設定する必要があるドライバパラメータのリストです。

表 **6-10** *ANSI* 互換のデータベース用のドライバ設定

表示名	タグ名	値
メタデータの取得でスキーマをサポートしますか？	supports-schemas-in-metadata-retrieval 67 ページの 「メタデータの取得でスキーマをサポートしますか？」を参照してください。	false
ユーザ名を大文字に固定：	force-username-case 65 ページの 「ユーザ名を大文字に固定」を参照してください。	upper

動的パラメータのデフォルト

次の表は、ランタイム時に JDBC 用ドライバが暗黙的に設定するドライバ互換性パラメータのリストです。これらの設定を上書きしないでください。

表 **6-11** 上書き不可の *Informix JDBC* 設定

表示名	タグ名	値
関数のリターン方法：	function-return-method 66 ページの 「関数のリターン方法」を参照してください。	result set

既知の問題

- ANSI 互換のデータベースではメタデータの取得にスキーマ名を使用できません。67 ページの 「メタデータの取得でスキーマをサポートしますか？」ドライバ互換性パラメータを論理値の False に設定します。メタデータの取得に使用できるデータベースオブジェクトは、そのデータベースを認証したデータベースユーザに表示できるオブジェクトです。スキーマ修飾子をデータベースオブジェクトの識別に使用することはできません。したがって、命名の衝突 (owner1.table1、owner2.table1 など) を避けるために、データベース認証ユーザに、同期中のオブジェクトの SELECT 特権だけを付与します。
- ANSI 互換のデータベースで使用する場合、ユーザ名は大文字にする必要があります。65 ページの 「ユーザ名を大文字に固定」ドライバ互換性パラメータを upper に設定します。

6.4.7 Microsoft SQL Server 2000 Driver for JDBC

表 **6-12** *Microsoft SQL Server 2000 Driver* の設定

サポートされているデータベースバージョン	8 (2000)
クラス名	com.microsoft.jdbc.sqlserver.SQLServerDriver

Type	4
URL の構文	jdbc:microsoft:sqlserver://*ip-address-or-dns-name*:1433;DatabaseName=*database-name*
ダウンロード方法	Microsoft JDBC ドライバ (http://www.microsoft.com/downloads/results.aspx?sortCriteria=date&OSID=&productID=&CategoryID=&freetext=jdbc&DisplayLang=en&DisplayEnglishAlso=)
ファイル名	msbase.jar、mssqlserver.jar、msutil.jar

互換性

SQL Server 2000 ドライバは、後方互換です。ただし、データベースバージョン 7 では機能しません。データベースサーバおよびドライバは、めったに更新されません。

セキュリティ

SQL Server 2000 ドライバは、暗号化された転送をサポートしません。

URL プロパティ

URL プロパティは、セミコロン (;) で区切ります。

次の表は、SQL Server 2000 ドライバの SelectMethod URL プロパティの値のリストです。

表 *6-13* *SelectMethod URL* プロパティの値

有効な値	説明
direct	デフォルト値。1 つの接続で複数のアクティブステートメントを使用できません。
cursor	1 つの接続で複数のアクティブステートメントを使用できます。

動的パラメータのデフォルト

次の表は、ランタイム時に JDBC 用ドライバが暗黙的に設定するドライバ互換性パラメータのリストです。これらの設定を明示的に上書きしないでください。

表 *6-14* 上書き不可の *SQL Server 2000* 設定

表示名	タグ名	値
ステートメントを再使用しますか？	reuse-statements	false

既知の問題

- 複製された接続が原因で、手動トランザクションを開始できません。

  SQL Server 2000 ドライバの実装は例外的であり、同じ接続で複数のステートメントを同時にアクティブにできません。これがこのドライバで発生する最も一般的な問題

の原因です。その他のサードパーティ実装とは異なり、SQL Server 2000 ドライバでは、特定の接続で同時にアクティブにできる java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html) オブジェクトは 1 つだけです。

複数のステートメントオブジェクトを使用しようとすると、「Can't start manual transaction mode because there are cloned connections. (複製された接続があるので手動トランザクションモードを開始できません。)」というエラーが発生します。このエラーは、61 ページの 「ステートメントを再使用しますか ?」 ドライバ互換性パラメータが論理値の True に設定されている場合にだけ発生する可能性があります。ベストプラクティスとして、このパラメータを明示的には設定しないでください。代わりに、動的なデフォルト値を使用します。

または、区切りられたプロパティ ;SelectMethod=cursor を URL 文字列の最後に配置します。この問題の追加情報については、次の補足記事を参照してください。

- DataDirect Technologies* の Document 30096 (http://knowledgebase.datadirect.com/kbase.nsf/SupportLink+Online/30096?OpenDocument)
- Microsoftの文書番号313181 (http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B313181)

- ドライバのバージョン間で、UNIQUEIDENTIFIER 列を含む関連付けの値の整合性が取れていません。

  以前のバージョンの SQL Server 2000 ドライバは、ネイティブの UNIQUEIDENTIFIER 列の非標準の java.sql.Types (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) 値を返していました。これに対処するために、JDBC 用ドライバは、非標準タイプを標準タイプの java.sql.Types.BINARY (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) にマップしていました。これがネイティブデータベースタイプ (16 バイトの値) の反映として最適だからです。このマッピングにより、Base64 でエンコードされた関連付けの値が得られます。

  SQL Server 2000 ドライバのそれ以降のバージョンでは、ドライバは、標準タイプの java.sql.CHAR (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) を返します。このマッピングにより、Base64 以外でエンコードされた関連付けの値が得られます。これによって、SQL Server 2000 ドライバの以前のバージョンを使用して生成されたすべての関連付けが事実上無効になります。この変更によって、実質的に後方互換性が維持されなくなっています。

  この問題を解決する最適な方法は、以前のバージョンの SQL Server 2000 ドライバを引き続き使用することです。アップグレードが必要な場合は、無効になる関連付けをすべて削除し、以前関連付けられていたオブジェクトをすべて再度関連付けます。

6.4.8 MySQL Connector/J JDBC Driver

表 6-15 *MySQL Connector/J JDBC Driver* の設定

サポートされているデータベースバージョン	3.x、4.x
クラス名	org.gjt.mm.mysql.Driver
Type	4
URL の構文	jdbc:mysql://*ip-address*:3306/*database-name*

ダウンロード方法	ダウンロードして解凍します。jar ファイルは、*extract-dir*/mysql-connector-java-*version* ディレクトリにあります。 MySQL Connector/J (http://www.mysql.com/products/connector/j/)
ファイル名	mysql-connector-java-*version*-bin.jar
マニュアルの URL	MySQL Connector/J Documentation (http://dev.mysql.com/doc/refman/5.0/en/java-connector.html) Connecting Over SSL (http://dev.mysql.com/doc/refman/5.0/en/cj-using-ssl.html)

72 ページの 「生成 / 取得方法 (テーブル - グローバル)」 も参照してください。

互換性

Connector/J ドライバは、後方互換です。データベースサーバは、頻繁に更新されます。ドライバはめったに更新されません。

セキュリティ

Connector/J ドライバは、JSSE (Java Secure Sockets Extension) SSL で暗号化された転送をサポートします。

MyISAM テーブルで必須のパラメータ設定

次の表は、JDBC 用ドライバを、MyISAM テーブルに対して Connector/J ドライバと相互運用できるようにするために、設定する必要があるドライバパラメータのリストです。

表 *6-16* *MyISAM* テーブル用の設定

表示名	タグ名	値
手動トランザクションを使用しますか ?	use-manual-transactions	false

6.4.9 Oracle Thin Client JDBC Driver

表 *6-17* *Oracle Thin Client* の設定

サポートされているデータベースバージョン	8i、9i、10g
クラス名	oracle.jdbc.driver.OracleDriver
Type	4
URL の構文	jdbc:oracle:thin:@*ip-address*:1521:*sid*
ダウンロード方法	登録して (無料) ダウンロードします。 Oracle Technology Network (http://otn.oracle.com/software/tech/java/sqlj_jdbc/content.html)

1.1 ファイル名	classes111.zip, nls_charset11.zip (オプション)
1.2-3 ファイル名	classes12.zip, ocrs12.zip (オプション)、nls_charset12.zip (オプション)
1.4 ファイル名	ojdbc14.jar, ocrs12.zip (オプション)
マニュアルの URL	Oracle Advanced Security (http://www.cs.umb.edu/cs634/ora9idocs/network.920/a96573/asojbdc.htm)
	JDBC FAQ (http://www.oracle.com/technology/tech/java/sqlj_jdbc/htdocs/jdbc_faq.htm)

互換性

Thin Client ドライバは、後方互換です。データベースサーバおよびドライバは、めったに更新されません。

Oracle は、さまざまな JVM 用の Thin Client ドライバをリリースしています。いずれも本製品で機能しますが、1.4 バージョンを使用することをお勧めします。

セキュリティ

Thin Client ドライバは、Oracle Advanced Security で暗号化された転送をサポートしています。

動的パラメータのデフォルト

次の表は、ランタイム時に JDBC 用ドライバが暗黙的に設定するドライバ互換性パラメータのリストです。これらの設定を明示的に上書きしないでください。

表 *6-18* 上書き不可の *Oracle Thin Client* 設定

表示名	タグ名	値
返された結果セットの数 :	handle-stmt-results	single

接続プロパティ

次の表は、このドライバで重要な接続プロパティのリストです。

表 *6-19* *Oracle Thin Client:* 接続プロパティ

プロパティ	意義
includeSynonyms	このプロパティの値が true の場合、シノニム列メタデータを使用できます。
ORACLE.NET.ENCRYPTION_CLIENT	クライアントがサーバとネゴシエートするセキュリティのレベルを定義します。
ORACLE.NET.ENCRYPTION_TYPES_CLIENT	使用する暗号化アルゴリズムを定義します。

プロパティ	意義
ORACLE.NET.CRYPTO_CHECKSUM_CLIENT	データの整合性についてサーバとネゴシエートするセキュリティのレベルを定義します。
ORACLE.NET.CRYPTO_CHEKSUM_TYPES_CLIENT	使用するデータ整合性アルゴリズムを定義します。

既知の問題

- 埋め込み SQL ステートメントを実行した場合に、CPU の使用率が高くなります。

  このドライバで最もよく発生する問題は、この高い CPU 使用率です。その結果、このドライバは、常に java.sql.Statement.execute(String stmt) メソッドの呼び出しから、より多くの結果を得られることを示し、無限ループにつながる可能性があります。この状況になるのは、次のすべてにあてはまる場合だけです。

  - 62 ページの 「返された結果セットの数」ドライバ互換性パラメータで single、no、または one 以外の値が実行されている。
  - 埋め込み SQL ステートメントが実行されている。
  - ステートメントのタイプが明示的に指定されていない。

  CPU 使用率が高くならないようにするには :

  - このパラメータを明示的に設定しないでください。動的なデフォルト値を使用します。
  - 埋め込み <jdbc:statement> 要素に、常に jdbc:type 属性を配置します。

    注 **:** jdbc ネームスペースプリフィックスを urn:dirxml:jdbc にマップする必要があります。

- 同義列メタデータを取得できません。

  接続プロパティ includeSynonyms を true に設定する必要があります。

- シノニムテーブルのプライマリキー制約を表示できません。

  この問題の解決法としてわかっているのは、ビューを使用する方法だけです。

6.4.10 PostgreSQL JDBC Driver

表 *6-20* *PostgreSQL JDBC Driver* の設定

サポートされているデータベースバージョン	6.x、7.x、8.x
クラス名	org.postgresql.Driver
Type	4
URL の構文	jdbc:postgresql://*ip-address*:5432/*database-name*
ダウンロード方法	JDBC Driver Download (http://jdbc.postgresql.org/download.html)

マニュアルの URL	JDBC Driver Documentation (http://jdbc.postgresql.org/documentation/docs.html) Using SSL (http://jdbc.postgresql.org/documentation/80/ssl.html)

注 **:** PostgreSQL のファイル名は、データベースバージョンによってさまざまです。

互換性

PostgreSQL ドライバの最新ビルドは、サーババージョン 7.2 では後方互換です。データベースサーバおよびドライバは頻繁に更新されます。

セキュリティ

PostgreSQL ドライバは、JDBC 3 ドライババージョンの場合の SSL で暗号化された転送をサポートしています。

6.4.11 Sybase Adaptive Server Enterprise JConnect JDBC ドライバ

表 *6-21* *Sybase Adaptive Server Enterprise* ドライバの設定

サポートされているデータベースバージョン	Adaptive Server* Enterprise 11.x、12.x
クラス名	com.sybase.jdbc2.jdbc.SybDriver (jconn2.jar の場合) com.sybase.jdbc3.jdbc.SybDriver (jconn3.jar の場合)
Type	4
URL の構文	jdbc:sybase:Tds:*ip-address*:2048/*database-name*
ダウンロード方法	Sybase Downloads (http://www.sybase.com/detail?id=1009796)
ファイル名	jconn2.jar または jconn3.jar
マニュアルの URL	jConnect Documentation (http://sybooks.sybase.com/onlinebooks/group-jc/jcg0600e/prjdbc)

互換性

Adaptive Server ドライバは、後方互換です。データベースサーバおよびドライバは、めったに更新されません。

セキュリティ

Adaptive Server ドライバは、SSL で暗号化された転送をサポートしています。SSL 暗号化を有効にするには、SYBSOCKET_FACTORY 接続プロパティでカスタムソケット実装を指定する必要があります。接続プロパティ設定方法の詳細については、57 ページの 「接続プロパティ」を参照してください。

接続プロパティ

SYBSOCKET_FACTORY プロパティは、暗号化された転送をサポートするカスタムソケット実装のクラス名の指定に使用できます。

6.5 サポートされていないサードパーティ製の JDBC ドライバ

6.5.1 IBM Toolbox for Java/JTOpen

表 6-22 *IBM Toolbox for Java/JTOpen* 用の設定

データベース	IBM Toolbox for Java/JTOpen ◆ iSeries Toolbox for Java (別名) ◆ AS/400 Toolbox for Java (別名)
クラス名	com.ibm.as400.access.AS400JDBCDriver
Type	4
URL の構文	jdbc:as400://ip-address/database-name
ダウンロード方法	JTOpen 用のダウンロード URL ◆ JTOpen (http://jt400.sourceforge.net) ◆ Toolbox for Java/JTOpen (http://www-03.ibm.com/servers/eserver/iseries/toolbox/downloads.html)
ファイル名	jt400.jar
マニュアルの URL	Toolbox for Java/JTOpen (http://www-03.ibm.com/servers/eserver/iseries/toolbox/)

IBM Toolbox for Java/JTOpen ドライバを使用する場合は、「JDBC Driver Class Name (JDBC ドライバのクラス名)」および「認証コンテキスト」パラメータの値を手動で入力する必要があります。自動的には設定されません。49 ページの 「サードパーティ製の JDBC ドライバのクラス名」および 46 ページの 「認証コンテキスト」を参照してください。

6.5.2 サードパーティ製の JDBC ドライバの最低要件

JDBC 用ドライバは、すべてのサードパーティ製の JDBC ドライバと相互運用できるとは限りません。サポートされていないサードパーティ製の JDBC ドライバを使用する場合は、次の要件を満たす必要があります。

- 必須のメタデータメソッドをサポートする。

JDBC 用ドライバで行われる必須およびオプションの java.sql.DatabaseMetaData メソッド呼び出しの現時点のリストについては、169 ページの付録 D「java.sql.DatabaseMetaData のメソッド」を参照してください。

- 他の必須の JDBC メソッドをサポートする。

  JDBC 用ドライバが使用する必須の JDBC メソッドのリストについては、171 ページの付録 E「JDBC インタフェースのメソッド」を参照してください。このリストと、サードパーティ製のドライバのマニュアルをあわせて参照して、潜在的な非互換性を特定できます。

6.5.3 その他のサードパーティ製の JDBC ドライバを使用する場合の考慮事項

- JDBC 用ドライバは、サードパーティ製の JDBC ドライバ実装に直接依存するので、その実装にバグがあると、本製品が正常に動作しなくなります。

  サードパーティ製の JDBC ドライバのデバッグに役立つように、JDBC 用ドライバでは、次をサポートしています。

  - JDBC API レベル (レベル 6) でのトレース
  - サードパーティ製の JDBC ドライバ (レベル 7) でのトレース
- ストアドプロシージャまたは関数のサポートも、よく障害ポイントとなります。
- たいていの場合は、カスタムドライバ記述子ファイルの書き換えが必要になります。

  特に、使用しているサードパーティ製のドライバ用のエラーコードおよび SQL 状態の分類が必要になります。

6.6 セキュリティの問題

JDBC 用 Identity Manager ドライバとサードパーティ製のドライバ間で安全な接続を確立するために、次をお勧めします。

- JDBC 用ドライバをデータベースサーバでリモートに実行する。
- SSL を使用して Identity Manager サーバとデータベースサーバ間の通信を暗号化する。

JDBC 用ドライバをリモートで実行できない場合は、Type 2 または Type 3 の JDBC ドライバを使用できます。これらのドライバタイプは、通常、他の JDBC ドライバタイプでは使用できないミドルウェアサーバまたはクライアント API を通じて、セキュリティを高めることができます。一部の Type 4 ドライバは、暗号化された転送をサポートしていますが、これは例外的です。

サポートされているデータベース 7

7.1 データベースの相互運用性

JDBC 用 Identity Manager ドライバは、データベースの特定のセットではなく JDBC ドライバ実装の特定のセットと相互運用できるように設計されています。このため、サポートされているデータベースとしてリストに含めるかどうかは、まず、サポートされているサードパーティ製の JDBC ドライバの機能で決まります。2 つ目の要因は、テストリソースです。

7.2 サポートされているデータベース

次のデータベースまたはデータベースバージョンは、テスト済みであり、本製品との連携使用が推奨されます。

表 *7-1* サポートされているデータベース

データベース	マイナーバージョン
IBM DB2 Universal Database (UDB) 7	7.2 以降
IBM DB2 Universal Database (UDB) 8	8.1 以降
Informix Dynamic Server (IDS)	9.40 以降
Microsoft SQL Server 7	7.5、Service Pack 4 以降
Microsoft SQL Server 8 (2000)	Service Pack 3a 以降
MySQL 3	3.23.58 以降
MySQL 4	4.1 以降
Oracle 8i	Release 3 (8.1.7) 以降
Oracle 9i	Release 2 (9.2.0.1) 以降
Oracle 10g	Release 1 (10.0.2.1) 以降
PostgreSQL 7	7.4.6 以降
Sybase Adaptive Server Enterprise (ASE) 12	12.5 以降

Driver for JDBC 用ドライバは、表に挙げた以外のデータベースまたはデータベースバージョンとも連携使用できます。ただし、Novell® では、それらをサポートしていません。

JDBC 用ドライバと相互運用するためには、データベースが以下の要件を満たしている必要があります。

- SQL-92 エントリレベルの構文をサポートしている。
- JDBC でアクセスできる。

7.3 データベースの特性

- 142 ページの 「データベースの機能」
- 146 ページの 「IBM DB2 Universal Database (UDB)」
- 146 ページの 「Informix Dynamic Server (IDS)」
- 147 ページの 「Microsoft SQL Server」
- 148 ページの 「MySQL」
- 149 ページの 「Oracle」
- 150 ページの 「PostgreSQL」
- 150 ページの 「Sybase Adaptive Server Enterprise (ASE)」

7.3.1 データベースの機能

表 7-2 データベースの機能

データベース	スキーマ	ビュー	識別列	シーケンス	ストアドプロシージャ	関数	トリガ	instead-of-trigger
IBM DB2 UDB 7	X	X	X	0	X[1]	X[1]	X	0
IBM DB2 UDB 8	X	X	X	0	X[1]	X[1]	X	X
Informix IDS 9	X	X	X[2]	0	X[3]	X	X	0
MS SQL 7	X	X	X	0	X	0	X	0
MS SQL 8	X	X	X	0	X	X	X	X
MySQL 4	0	0	X[4]	0	0	0	0	0
Oracle 8i、9i、10g	X	X	0	X	X	X	X	X
Postgres 7	X	X	X[5]	X	X	X	X[6]	X[6]
Sybase ASE 12	X	X	X	0	X	0	X	0

[1] DB2 は、Java で記述されたストアドプロシージャまたは関数をネイティブでサポートしています。ネイティブの SQL 手続き型言語を使用してプロシージャを記述するには、データベースサーバに C コンパイラをインストールします。

[2] Informix 識別列キーワードは、SERIAL8 です。

[3] Informix のストアドプロシージャは、値を返すことができません。

[4] MySQL 識別列キーワードは、AUTO_INCREMENT です。

[5] Postgres シーケンスオブジェクトを使用して、プライマリキー列のデフォルト値を提供できるので、実質的に識別列をシミュレートできます。

Postgres には、ルールと呼ばれるネイティブの構成体があります。この構成体を使用して、トリガおよび instead-of-trigger を実質的にシミュレートできます。また、さまざまな手続き型プログラミング言語で記述されたトリガまたは instead-of-trigger の使用もサポートしています。

7.3.2 現在のタイムスタンプステートメント

次の表は、現在の日付および時刻を取得するために使用される SQL ステートメントをデータベース別に示したものです。

表 *7-3* タイムスタンプステートメント

データベース	現在のタイムスタンプステートメント	ANSI 互換
IBM DB2 UDB	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY	×
Informix IDS	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES	×
MSSQL	SELECT (CURRENT_TIMESTAMP)	○
MySQL	SELECT (CURRENT_TIMESTAMP)	○
Oracle	SELECT (SYSDATE) FROM SYS.DUAL	×
PostgreSQL	SELECT (CURRENT_TIMESTAMP)	○
Sybase ASE	SELECT GETDATE()	×

7.3.3 ストアドプロシージャおよび関数の JDBC 呼び出しの構文

次の表は、ストアドプロシージャまたは関数を呼び出すための SQL 構文のリストです。これは、埋め込み SQL ステートメント内のプロシージャおよび関数呼び出しのフォーマットに役立ちます。

表 *7-4* ストアドプロシージャまたは関数の呼び出し

データベース	ストアドプロシージャ / 関数の JDBC 呼び出し構文
IBM DB2 UDB	{call *schema-name.procedure-name*(*parameter-list*)}
Informix IDS	EXECUTE [PROCEDURE | FUNCTION] *schema-name.routine-name*(*parameter-list*)
MSSQL	EXECUTE *schema-name.procedure-name*(*parameter-list*)
MySQL	(該当なし)

データベース	ストアドプロシージャ / 関数の JDBC 呼び出し構文
Oracle[1]	CALL *schema-name*.*procedure-name*(*parameter-list*)
PostgreSQL	SELECT **schema-name**.**procedure-name**(**parameter-list**)
Sybase ASE	EXECUTE *schema-name*.*procedure-name*(*parameter-list*)

[1] Oracle の JDBC 実装では、文字列としての関数呼び出しはサポートされていません。

7.3.4 左外部結合演算子

次の表は、データベース別の外部結合演算子のリストです。

表 *7-5* 外部結合演算子

データベース	左外部結合演算子	ANSI 互換
IBM DB2 UDB	LEFT OUTER JOIN	○
Informix IDS	LEFT OUTER JOIN	○
MSSQL	*=	×
MySQL	LEFT OUTER JOIN	○
Oracle	(+)	×
PostgreSQL	LEFT OUTER JOIN	○
Sybase ASE	*=	×

注 **:** Oracle は、バージョン 10g から ANSI 互換の左外部結合演算子 LEFT OUTER JOIN をサポートしています。

7.3.5 区切りのない識別子での大文字と小文字の区別

表 *7-6* 区切りのない識別子での大文字と小文字の区別

データベース	大文字と小文字の区別
IBM DB2 UDB	×
Informix IDS	×
MSSQL	×
MySQL	○
Oracle	×
PostgreSQL	×
Sybase ASE	○

7.3.6 サポートされているトランザクション分離レベル

表 *7-7* サポートされているトランザクション分離レベル

データベース	なし	read uncommitted	read committed	repeatable read	serializable	URL
IBM DB2 UDB	0	X	X[1]	X	X	JDBC トランザクションの分離レベルの設定 (http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp?topic=/com.ibm.db2.udb.doc/ad/tjvjdiso.htm)
MySQL (InnoDB テーブルタイプ)	0	X	X	X[1]	X	InnoDB Transaction Isolation Levels (http://dev.mysql.com/doc/mysql/en/innodb-transaction-isolation.html)
Oracle	0	0	X[1]	0	X	JDBC Transaction Optimization (http://www.oracle.com/technology/oramag/oracle/02-jul/o42special_jdbc.html)
PostgreSQL	0	0[2]	X[1]	0[2]	X	Transaction Isolation (http://www.postgresql.org/docs/current/static/transaction-iso.html)

[1] これは、このデータベースのデフォルト分離レベルです。[2] 設定はできますが、サポートされている分離レベルに引き上げられます。

7.3.7 コミットキーワード

次の表は、サポートされているデータベースのコミットキーワードを示します。

表 *7-8* コミットキーワード

データベース	コミットキーワード
IBM DB2 UDB	COMMIT
Informix IDS	COMMIT WORK[1]
MSSQL	GO
MySQL	COMMIT
Oracle	COMMIT
PostgreSQL	COMMIT
Sybase ASE	GO

[1] ANSI 互換のログデータベース用。ログデータベース以外のデータベースではトランザクションをサポートしません。

7.3.8 IBM DB2 Universal Database (UDB)

次の表は、このデータベースのプロパティのリストです。

表 7-9 *IBM DB2 UDB のプロパティ*

プロパティ	値
現在のタイムスタンプステートメント	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY
ストアドプロシージャ / 関数の呼び出し構文	{call *schema-name.procedure-name*(*parameter-list*)}
大文字と小文字を区別？	×
コミットキーワード	COMMIT
左外部結合演算子	LEFT OUTER JOIN

動的なデフォルト

次の表は、ランタイム時に JDBC 用ドライバが暗黙的に設定するデータベース互換性パラメータのリストです。これらの設定を明示的に上書きしないでください。

表 7-10 動的に設定される *IBM DB2 Universal Database* 設定

表示名	タグ名	値
現在のタイムスタンプステートメント :	current-timestamp-stmt	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY
Timestamp Translator class: (タイムスタンプトランスレータクラス :)	time-translator-class	com.novell.nds.dirxml.driver.jdbc.db.DB2Timestamp

既知の問題

- 独自のタイムスタンプ形式が使用されます。151 ページの 「既知の問題」を参照してください。

7.3.9 Informix Dynamic Server (IDS)

次の表は、このデータベースのプロパティのリストです。

表 **7-11** *Informix Dynamic Server* の設定

プロパティ	値
現在のタイムスタンプステートメント	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES
ストアドプロシージャ / 関数の呼び出し構文	EXECUTE [PROCEDURE | FUNCTION] *schema-name.procedure-name*(parameter-list)
大文字と小文字を区別？	×
コミットキーワード	COMMIT WORK[1]
左外部結合演算子	LEFT OUTER JOIN

[1] ログデータベースおよび ANSI 互換のデータベース用。ログデータベース以外ではトランザクションはサポートされません。

動的なデフォルト

次の表は、ランタイム時に JDBC 用ドライバが暗黙的に設定するデータベース互換性パラメータのリストです。これらの設定を明示的に上書きしないでください。

表 **7-12** 動的に設定される *Informix Dynamic Server* 設定

表示名	タグ名	値
現在のタイムスタンプステートメント：	current-timestamp-stmt	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES

既知の問題

- NUMERIC または DECIMAL 列は、テーブルが作成されたときにスケール (小数点の右側の桁数) が明示的に 0 に設定されている場合以外は、プライマリキーとして使用できません。デフォルトで、スケールは 255 に設定されます。

7.3.10 Microsoft SQL Server

次の表は、このデータベースのプロパティのリストです。

表 **7-13** *Microsoft SQL Server 2000* の設定

プロパティ	値
現在のタイムスタンプステートメント	SELECT (CURRENT_TIMESTAMP)
ストアドプロシージャ / 関数の呼び出し構文	EXECUTE *schema-name.procedure-name*(*parameter-list*)

プロパティ	値
大文字と小文字を区別？	×
コミットキーワード	GO
左外部結合演算子	*=

動的なデフォルト

次の表は、ランタイム時に JDBC 用ドライバが暗黙的に設定するデータベース互換性パラメータのリストです。これらの設定を明示的に上書きしないでください。

表 *7-14* 動的に設定される *Microsoft SQL Server* 設定

表示名	タグ名	値
Add Default Values on Insert? (挿入時にデフォルト値を追加しますか ?)	add-default-values-on-view-insert	true
左外部結合演算子 :	left-outer-join-operator	*=

7.3.11 MySQL

次の表は、このデータベースのプロパティのリストです。

表 *7-15* *MySQL* の設定

プロパティ	値
現在のタイムスタンプステートメント	SELECT (CURRENT_TIMESTAMP)
ストアドプロシージャ / 関数の呼び出し構文	(該当なし)
大文字と小文字を区別？	○
コミットキーワード	COMMIT
左外部結合演算子	LEFT OUTER JOIN

動的なデフォルト

次の表は、ランタイム時に JDBC 用ドライバがこのデータベースのために動的に設定するデータベース互換性パラメータのリストです。

表 *7-16* 動的に設定される *MySQL* 設定

表示名	タグ名	値
メタデータの取得でスキーマをサポートしますか？	supports-schemas-in-metadata-retrieval	false

既知の問題

- TIMESTAMP 列は、最初に 0 または NULL に設定された後に更新される場合は、常に現在の日付および時刻に設定されます。この動作に対処するために、アイデンティティボールトの時間およびタイムスタンプの構文を DATETIME 列にマップすることをお勧めします。

7.3.12 Oracle

次の表は、このデータベースのプロパティのリストです。

表 *7-17* *Oracle* の設定

プロパティ	値
現在のタイムスタンプステートメント	SELECT (SYSDATE) FROM SYS.DUAL
ストアドプロシージャ / 関数の呼び出し構文	CALL *schema-name*.*procedure-name*(*parameter-list*)
大文字と小文字を区別？	×
コミットキーワード	COMMIT
左外部結合演算子	(+)

動的なデフォルト

次の表は、ランタイム時に JDBC 用ドライバが暗黙的に設定するデータベース互換性パラメータのリストです。これらの設定を明示的に上書きしないでください。

表 *7-18* 動的に設定される *Oracle* 設定

表示名	タグ名	値
左外部結合演算子	left-outer-join-operator	(+)
フィルタ式を除く	exclude-table-filter	BIN\$.{22}==\$0
ステートメントジェネレータクラスのロック	lock-generator-class	com.novell.nds.dirxml.driver.jdbc.db.lock.OraLockGenerator

注 **:** デフォルトの除外フィルタは、Oracle 10g で表示されるドロップ済みテーブルを同期スキーマから省略するためのものです。

制限

- トリガでは、LONG、LONG RAW、および BLOB 列を参照できません。instead-of-trigger を含むトリガ内で :NEW 修飾子を使用して、これらのタイプの列を参照することはできません。

7.3.13 PostgreSQL

次の表は、このデータベースのプロパティのリストです。

表 *7-19* *PostgreSQL* の設定

プロパティ	値
現在のタイムスタンプステートメント	SELECT (CURRENT_TIMESTAMP)
ストアドプロシージャ / 関数の呼び出し構文	SELECT *schema-name*.*procedure-name*(*parameter-list*)
大文字と小文字を区別？	×
コミットキーワード	COMMIT
左外部結合演算子	LEFT OUTER JOIN

既知の問題

- PostgreSQL は <check-object-password> イベントをサポートしていません。認証は、pg_hba.conf ファイルにエントリを手動で挿入して制御します。

7.3.14 Sybase Adaptive Server Enterprise (ASE)

次の表は、このデータベースのプロパティのリストです。

表 *7-20* *Sybase ASE* の設定

プロパティ	値
現在のタイムスタンプステートメント	SELECT GETDATE()
ストアドプロシージャ / 関数の呼び出し構文	EXECUTE *schema-name*.*procedure-name*(*parameter-list*)
大文字と小文字を区別？	○
コミットキーワード	GO
左外部結合演算子	*=

動的なデフォルト

次の表は、ランタイム時に JDBC 用ドライバが暗黙的に設定するデータベース互換性パラメータのリストです。これらの設定を明示的に上書きしないでください。

表 *7-21* 動的に設定される *Sybase ASE* 設定

表示名	タグ名	値
現在のタイムスタンプステートメント	current-timestamp-stmt	SELECT GETDATE()
左外部結合演算子	left-outer-join-operator	*=
Timestamp Translator class (タイムスタンプトランスレータクラス)	time-translator-class	com.novell.nds.dirxml.driver.jdbc.db.SybaseTimestamp

既知の問題

- バイナリ値のパディングおよび切り捨てに注意が必要です。

  バイナリ値で ANSI 互換のパディングおよび切り捨ての動作に準拠するために、バイナリ列タイプ (IMAGE を除く) が次の条件を満たすようにします。

  - マップ元の eDirectory™ 属性のサイズとまったく同じである。
  - NOT NULL 制約がない。
  - 発行者作成ポリシーおよび購読者作成ポリシーに追加される。

  NULL 制約がある場合は、eDirectory では有効である末尾のゼロ (0) が切り捨てられます。バイナリ列が、対応する eDirectory 属性のサイズより大きい場合は、値の末尾に 0 が付加されます。

  解決方法として、バイナリ値を同期する場合は IMAGE データタイプだけを使用することをお勧めします。
- DATETIMEの秒の小数部が丸められます。Sybaseのタイムスタンプの最高精度は1/300秒 (約 .003 秒) です。データベースサーバは、最も近い 1/1000 秒 (.001 秒または 1 ミリ秒) 単位の値ではなく、最も近い 1/300 秒単位の値に丸められます。
- 独自のタイムスタンプ形式が使用されます。

Association Utility

8

Association Utility は、1.0 以降のバージョンの JDBC 用ドライバで関連付けられたオブジェクトの関連付けを正規化します。その他にも、ドライバ管理を簡単にする機能が複数あります。

ユーティリティのこのバージョンは、1.0 以降のバージョンの JDBC 用ドライバと互換性があり、それ以前のすべてのバージョンに優先します。

8.1 独立した操作

Association Utility は、7 つの独立した操作をサポートします。

表 *8-1* 独立した操作

操作	説明	読み書き機能
1	ドライバに関連付けられているオブジェクトをリストします (デフォルト)。	読み込み専用
2	ドライバに対して複数の関連付けを持つオブジェクトをリストします。	読み込み専用
3	ドライバに対して無効な関連付けを持つオブジェクトをリストします。 関連付けは、次の場合に無効です。 ◆ 形式が間違っている。 たとえば、関連付けでスキーマ RDN またはテーブル RDN が見つからない、またはスキーマキーワードのスペルが間違っていることがあります。 ◆ ターゲットデータベースの識別子にマップされていないデータベース識別子が含まれている。 たとえば、関連付けに、存在しないテーブルへのマッピングが含まれていることがあります。 ◆ 行のマップされていないか、複数行にマップされている。 関連付けは、行にマップされていない場合は壊れます。また、関連付けは、複数の行にマップされる場合は一意ではありません。	読み込み専用

操作	説明	読み書き機能
4	正規化が必要なオブジェクトをリストします。 正規化された関連付けは、有効で、正しい順に並べられ、また大文字小文字を正しく使用しています。大文字と小文字が区別されない、または混在しているデータベースの場合、大文字に正規化されます。	読み込み専用
5	操作 4 でリストされたオブジェクトの関連付けを正規化します。	書き込み
6	変更されるオブジェクトの関連付けをリストします。 検索条件に基づいて、スキーマ、テーブル、および列の名前のグローバルな置換ができます。 この操作には、2 つのパラメータ (oldRDN および newRDN) が必要です。156 ページの 「関連付けの編集」を参照してください。	読み込み専用
7	操作 6 でリストされたオブジェクトの関連付けを変更します。 この操作には、2 つのパラメータ (oldRDN および newRDN) が必要です。156 ページの 「関連付けの編集」を参照してください。	書き込み

8.2 開始準備

関連付けの変更によって問題が発生する可能性があります。関連付けが壊れた場合、Identity Manager は機能を停止します。したがって、書き込み操作は、必要な場合に限って使用してください。誤って関連付けを壊すことがないように、Association Utility は、すべての書き込み操作について、元に戻すための ldiff ファイルを作成します。

ユーティリティを使用する前に、以下の注意をお読みください。

- Association Utility では、ドライバと同様に、データベース識別子が区切られていない (引用符で囲まれておらず、特殊文字を含まない) ことを前提としています。
- ドライバに関連するオブジェクトの関連付けをすべて一緒に更新します。

  重要 **:** ドライバに関連するオブジェクトの関連付けをすべて同時に更新することは非常に重要です。

  特定のドライバに関連付けられているオブジェクトをすべて表示するには、特定のドライバインスタンスに関連付けられている Identity Manager サーバで Association Utility を実行します。

  LDAP 検索ベースには、特定のドライバに関連付けられているオブジェクトをすべて含める必要があります。

  注 **:** すべて確実に含めるために、検索ベースとしてツリーのルートコンテナを使用することをお勧めします。

- このユーティリティに提供されるターゲットデータベースの JDBC URL と、ドライバが使用する URL が同じであることを確認します。このユーティリティを大文字と小文字を区別しないデータベースにポイントしたときに、そのデータベースが実際は大文字と小文字を区別する場合は、関連付けの正規化で、間違った大文字小文字が使用される可能性があります。

- Association Utility はローカルで実行されるので、保護されていない接続を使用します。したがって、アイデンティティボールト LDAP サーバで、一時的にクリアテキストパスワードを受け付けるように設定する必要があります。使用するサードパーティ製の JDBC ドライバによっては、このユーティリティで確立されるデータベース接続が保護されない可能性があります。

  注 **:** このユーティリティを実行した後は、データベースでドライバの認証パスワードを変更することをお勧めします。

8.3 Association Utility の使用

Association Utility は、Identity Manager サーバにインストールしたドライバのインスタンスごとに 1 回実行します。*install-dir*\jdbc\util ディレクトリで、(プラットフォームに応じて) バッチファイル association.bat またはシェルスクリプト association.sh でユーティリティを起動します。

関連ユーティリティのパラメータを含むプロパティファイルは、サポートされているデータベースごとに提供されます。これらのファイルは、*install-dir*\jdbc\util ディレクトリにあります。

データベース	プロパティファイル名
IBM DB2 Universal Database	properties_db2.txt
Informix Dynamic Server	properties_ifx_ansi.txt1 properties_ifx_log.txt properties_ifx_no_log.txt
Microsoft SQL Server	properties_ms.txt
MySQL	properties_my.txt
Oracle	properties_ora.txt
PostgreSQL	properties_pg.txt
Sybase Adaptive Server Enterprise	properties_syb.txt

[1] このユーティリティは、Informix 製の ANSI 互換データベースでは機能しません。

注 **:** コマンドラインからユーティリティを実行する方法については、*install-dir*\tools\util ディレクトリの run.bat を参照してください。

1. ドライバを停止します。
2. Association Utility を実行して、不要な関連付けを識別し削除します (操作 2 および 3)。

   本製品によって関連付けられたオブジェクトでは、複数の関連付けを保持できません。オブジェクト単位で、不要な関連付けを手動で削除します。複数の関連付けで実際に有効なものはどれかを識別する場合は、操作 3 が役立ちます。有効な関連付けがわかったら、不要な関連付けを破棄できます。
3. Association Utility を実行して、無効な関連付けを識別し、修正します (操作 3 と、場合によっては操作 6 および 7)。

原則として、問題が単独で発生する場合は、無効な関連付けをそれぞれ手動で編集します。問題が繰り返し発生し、多くの関連付けに影響を及ぼす場合は、操作 6 および 7 の使用を検討します。このユーティリティは、誤った識別子をグローバルに置換することはできますが、識別子を今までなかった場所で削除または挿入することはできません。

4 Association Utility を実行して、関連付けを正規化します (操作 4 および 5)。

8.4 関連付けの編集

Association Utility では、操作 6 および 7(検索および置換) に、2 つのパラメータ (oldRDN および newRDN) が必要です。

パラメータの 1 つ目の値 (たとえば、schema) は検索条件です。2 つ目の値 (たとえば、old) は置換値です。特定のシナリオでは、ワイルドカード文字 * を使用して、検索条件または置換値を一般化できます。

検索および置換操作には 3 つのタイプがあります。

オプション	説明	例
スキーマ名の置換	スキーマ old をスキーマ new で置き換えます。ワイルドカードは、右側でだけサポートされます。	oldRDN:schema=old newRDN:schema=new
テーブル名の置換	テーブル old をテーブル new で置き換えます。ワイルドカードはサポートされていません。	oldRDN:table=old newRDN:table=new
列名の置換	列 old を列 new で置き換えます。ワイルドカードは、右側で必須ですが、左側ではサポートされていません。	oldRDN:old=* newRDN:new=*

JDBC 用 IDM ドライバのアンインストール

9

重要 : 設定済みドライバおよびデータベーススクリプトのインストールまたはアンインストールは、まとめて行うことをお勧めします。データベーススクリプトおよび設定済みドライバには、誤ってこれらが一致しなくなるのを防ぐために、バージョン番号、ターゲットデータベース名、データベースバージョンのヘッダが含まれています。

9.1 IDM ドライバオブジェクトの削除

When deleting Novell® アイデンティティボールトブジェクトを削除する場合は、すべての子オブジェクトを削除してから親オブジェクトを削除する必要があります。たとえば、発行者オブジェクトを削除するには、まず発行者チャネルのすべてのルールおよびスタイルシートを削除する必要があります。同様に、ドライバオブジェクトを削除するには、まず発行者と購読者の両方のオブジェクトを削除する必要があります。

アイデンティティボールトからドライバオブジェクトを削除するには :

1. Novell iManager で、［Identity Manager］>［Identity Manager の概要］の順にクリックします。
2. ドライバセットを選択します。
3. ［Identity Manager ドライバの概要］ページで、［ドライバの削除］をクリックします。
4. 削除するドライバを選択し、［OK］をクリックします。

9.2 製品のアンインストーラの実行

アンインストールの手順は、プラットフォームごとに異なります。

Windows で JDBC 用 Identity Manager ドライバをアンインストールするには、［コントロールパネル］の［プログラムの追加と削除］を使用します。

9.3 データベースのアンインストールスクリプトの実行

この節では、データベースのアンインストール SQL スクリプトを実行する場合に役立ちます。

- 159 ページの 「MySQL のアンインストール」
- 37 ページの 「Oracle のインストール」
- 38 ページの 「PostgreSQL のインストール」
- 38 ページの 「Sybase Adaptive Server Enterprise (ASE) のインストール」

9.3.1 IBM DB2 Universal Database (UDB) のアンインストール

DB2 のディレクトリコンピュータは、*install-dir*\jdbc\sql\db2_udbl\install です。

1. idm、indirect、direct オペレーティングシステムユーザアカウントを削除します。
2. インストールスクリプトで管理者アカウントの名前およびパスワードを変更していない場合は変更します。
3. コマンドラインプロセッサ (CLP) を使用して、uninstall.sql スクリプトを実行します。

   次に例を示します。db2 -f uninstall.sql

   重要 **:** このスクリプトは、バージョン 7 より後の Command Center インタフェースでは実行されません。スクリプトでは、行継続文字「\」が使用されていますが、後のバージョンの Command Center では、この文字は認識されません。

   ---
4. idm_db2.jar ファイルを削除します。

9.3.2 Informix Dynamic Server (IDS) のアンインストール

Informix SQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\informix_ids\install です。

1. idm オペレーティングシステムユーザアカウントを削除します。
2. SQL エディタなどのクライアントを起動します。
3. サーバに、informix ユーザ、または DBA (データベース管理者) 特権を持つ別のユーザとしてログオンします。

   デフォルトでは、informix のパスワードは informix です。

   informix 以外のユーザとしてインストールスクリプトを実行する場合は、実行する前にスクリプト内の informix への参照をすべて変更します。
4. デフォルトパスワードを持つ informix アカウントを使用しない場合に、インストールスクリプト内で DBA アカウントの名前とパスワードをまだ変更していなければ変更します。
5. インストールしているデータベースのタイプに応じて、ansi (トランザクション、ANSI 互換)、log (トランザクション、ANSI 非準拠)、または no_log (非トランザクション、ANSI 非準拠) サブディレクトリから uninstall.sql を開いて実行します。

9.3.3 Microsoft SQL Server のアンインストール

Microsoft SQL Server スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\mssql\install です。

1. Query Analyzer などのクライアントを開始します。
2. データベースサーバに sa ユーザとしてログオンします。

デフォルトで、sa ユーザにパスワードはありません。

3 最初のインストールスクリプト uninstall.sql を開いて実行します。

Query Analyzer の実行のホットキーは <F5> です。

9.3.4 MySQL のアンインストール

MySQL SQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\mysql\install です。

1 mysql などの MySQL クライアントから、root ユーザ、または管理特権を持つ他のユーザとしてログオンします。

たとえば、コマンドラインから、次のコマンドを実行します。mysql -u root -p

デフォルトで、root ユーザにパスワードはありません。

2 アンインストールスクリプト uninstall.sql を実行します。

次に例を示します。mysql> \. c:\uninstall.sql

このステートメントの最後にセミコロンを使用しないでください。

9.3.5 Oracle のアンインストール

Oracle SQL スクリプトのディレクトリコンテキストは、install-dir\jdbc\sql\oracle\install です。

1 SQL Plus などの Oracle クライアントから、SYSTEM ユーザとしてログオンします。

デフォルトで、SYSTEM のパスワードは MANAGER です。

パスワードが MANAGER の SYSTEM 以外のユーザとしてスクリプトを実行する場合は、実行する前にスクリプト内の SYSTEM への参照をすべて変更します。

2 アンインストールスクリプト uninstall.sql を実行します。

次に例を示します。SQL> @c:\uninstall.sql

9.3.6 PostgreSQL のアンインストール

PostgreSQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\postgres\install です。Postgres コマンドの実行のディレクトリコンテキストは、*postgres-install-dir*/pgsql/bin です。

1 psql などの Postgres クライアントから、ユーザ postgres として idm データベースにログオンします。

たとえば、UNIXC コマンドラインから、次のコマンドを実行します。./psql -d idm postgres

デフォルトで、Postgres ユーザにパスワードはありません。

2 psql 内から、スクリプト uninstall.sql を実行します。

次に例を示します。idm=# \i uninstall.sql

3 データベース idm を削除します。

たとえば、UNIX コマンドラインから、次のコマンドを実行します。./dropdb idm

4 idm ユーザのエントリを、pg_hba.conf ファイルから削除するか、コメントにします。
次に例を示します。

```
#host    idm          idm    255.255.255.255   255.255.255.0
```

5 Postgres サーバを再起動して、pg_hba.conf ファイルに対する変更を有効にします。

9.3.7 Sybase Adaptive Server Enterprise (ASE) のアンインストール

Sybase SQL スクリプトのディレクトリコンテキストは、*install-dir*\jdbc\sql\sybase_ase\install です。

1 isql などの Sybase クライアントから、sa ユーザとしてログオンします。
2 インストールスクリプト uninstall.sql を実行します。
たとえば、コマンドラインから次のコマンドを実行します。isql -U sa -P -i uninstall.sql
デフォルトで、sa アカウントにはパスワードはありません。

ベストプラクティス

A

この節では、JDBC 用ドライバを使用する場合の重要なベストプラクティスのリストを示します。43 ページの第 4 章「JDBC 用 Identity Manager ドライバの設定」および 89 ページの第 5 章「詳細環境設定」も参照できます。

セキュリティ / パフォーマンス :

- パフォーマンスおよびセキュリティの理由で、可能な場合は常にドライバをデータベースサーバでリモートに実行します。必ずアイデンティティボールトとリモートローダサービス間の SSL 暗号化を有効にしておきます。
- JDBC 用ドライバをデータベースサーバでリモートに実行していない場合は、サードパーティ製のドライバで SSL 暗号化を有効にする必要があります。サポートされているサードパーティ製のドライバのセキュリティ機能については、123 ページの「サードパーティ製の JDBC ドライバ」を参照してください。
- 運用環境では、トレースをオフにします。

その他 :

- 直接同期の場合は、ビューの 1 つまたは複数の列名にプリフィックス「pk_」(大文字と小文字の区別なし) を付けます。
- 直接および間接同期のどちらの場合も、論理データベースクラス間で異なるプライマリキー列名を使用します。
- イベントログのtable_keyフィールド内のプライマリキー値に次の文字が含まれている場合は区切ります (二重引用符で囲みます)。, ; ' + = \ " < > この注意は、通常、プライマリキー列がバイナリタイプの場合にだけ適用されます。
- アイデンティティボールトがプライマリキー値の信頼されるソースである場合は、プライマリキー値として、CN よりも GUID を使用することをお勧めします。CN とは異なり、GUID は、単一値で、変更されません。
- 発行トリガから、子および親テーブルをリンクする外部キー列を省きます。
- プライマリキー列がスタティックである (変更されない) 場合は、発行トリガには含めないでください。
- すべての埋め込み SELECT ステートメントに、jdbc:type=”query” 属性値を配置します。すべての埋め込み INSERT、UPDATE、および DELETE ステートメントに、jdbc:type=”update” 属性値を配置します。

FAQ

B

B.1 テーブルまたはビューを参照できない

質問 : なぜドライバがテーブルまたはビューを参照できないのですか ?

回答 : ドライバは、明示的なプライマリキー制約を持つテーブル、およびプリフィックス「pk_」(大文字と小文字の区別なし) の付いた列を 1 つ以上含むビューだけを同期できます。ドライバは、これらの制約を使用して、関連付けを作成するときに使用するフィールドを判別します。このため、ドライバは、制約ないテーブルを無視します。必要な制約を持たないテーブルまたはビューを同期する場合は、制約を追加するか、必要な制約を持つ中間テーブルと同期します。

または、テーブルを参照するために必要なデータベース特権をドライバが持っていない可能性があります。通常、参照できるかどうかは、SELECT 特権があるかどうかによって決まります。

B.2 テーブルとの同期

質問 : 複数のスキーマにあるテーブルと同期するにはどうすればいいですか ?

回答 : 次のいずれかを実行します。

- テーブルに別名を付けて同期スキーマに入れます。
- 同期スキーマ内の中間テーブルと同期し、スキーマ境界を越えてデータを移動します。
- ビューを使用します。
- 「テーブル / ビュー名」パラメータを使用して仮想スキーマを作成します。

  55 ページの 「テーブル / ビュー名」を参照してください。

B.3 イベントログテーブルの行の処理

質問 : なぜドライバがイベントログテーブル内の行を処理しないのですか ?

回答 : 以下を実行します。

1 問題の行の perpetrator フィールドで、ドライバのデータベースユーザ名以外の値が設定されていることを確認します。

発行者チャネルは、perpetrator フィールドをチェックして、発行者チャネルの「Allow Loopback (ループバックを許可しますか ?)」パラメータが論理値の False (デフォルト) に設定されている場合に、ループバックイベントを検出します。82 ページの 「ループバックを許可しますか ?」を参照してください。

「Allow Loopback (ループバックを許可しますか ?)」パラメータが論理値の False に設定されている場合、発行者チャネルは、perpetrator フィールドの値がドライバのデータベースユーザ名と等しいレコードをすべて無視します。ドライバのデータベースユーザ名は、「認証 ID」パラメータを使用して指定されます。46 ページの 「認証 ID」を参照してください。

2 レコードの status フィールドが N (新規) に設定されていることを確認します。

status フィールドが N 以外の値に設定されているレコードは処理されません。

3 明示的に変更をコミットします。

変更は、通常、明示的にコミットされるまでは仮の状態になります。

B.4 データベースユーザアカウントの管理

質問 : ドライバでデータベースユーザアカウントを管理できますか ?

回答 : はい。データベースアカウントは、埋め込み SQL を使用して管理できます。詳細については、109 ページの 「XDS イベントへの SQL ステートメントの埋め込み」を参照してください。

B.5 ラージデータタイプの同期

質問 : ドライバで、ラージバイナリおよび文字列データタイプを同期できますか ?

回答 : はい。ラージバイナリおよび文字列データタイプは、購読および発行できます。ラージバイナリおよび文字列データタイプは、クエリバックイベントタイプを使用して発行します。詳細については、103 ページの 「イベントタイプ」を参照してください。

B.6 発行処理が遅い

質問 : 発行の処理速度が遅いのはなぜですか ?

回答 : イベントログテーブルに多数の行が含まれている場合は、テーブルにインデックスを付けます。サンプルインデックスは、すべてのデータベースインストールスクリプトで提供されます。トレースレベル 3 を使用すると、ドライバがイベントログの維持に使用するステートメントを表示できます。

インストールスクリプトのインデックスをさらに調整して、発行パフォーマンスを高めることができます。インデックスを、イベントログテーブルとは別のテーブルスペースまたは物理ディスクに配置する方法でも、発行パフォーマンスを高めることができます。

さらに、運用環境で、「Delete Processed Rows (処理された行を削除しますか ?)」パラメータを論理値の False に設定します。ただし、処理済みの行が定期的に別のテーブルに移動される場合は除きます。81 ページの 「処理された行を削除しますか ?」 を参照してください。

B.7 複数のクラスの同期

質問 : ドライバで複数のクラスを同期できますか ?

回答 : はい。ただし、プライマリキー列名は、論理データベースクラス間で一意である必要があります。たとえば、*class1* が *key1* という名前のプライマリキー列を持つ *table1* にマップされ、*class2* が *key2* という名前のプライマリキー列を持つ *table2* にマップされている場合は、*key1* の名前を *key2* と同じにすることはできません。

この要件は、どの同期モデルが採用された場合でも常に満たすことができます。

B.8 暗号化された転送

質問 : ドライバは、暗号化された転送をサポートしていますか ?

回答 : いいえ。ドライバが特定のデータベースとどのように通信するかは、使用するサードパーティ製のドライバによって決まります。一部のサードパーティ製のドライバは暗号化された転送をサポートしていますが、それ以外のドライバはサポートしていません。暗号化された転送がサポートされている場合でも、サードパーティ製の JDBC ドライバ間で暗号化を有効にする標準的な方法はありません。

この問題の一般的な解決方法は、JDBC 用ドライバとサードパーティ製のドライバをリモートに実行することです。この方法では、JDBC 用ドライバとサードパーティ製のドライバの両方をデータベースサーバでローカルに実行できます。さらに、メタディレクトリエンジンと JDBC 用ドライバ間のネットワークを介して移動するすべてのデータは、SSL で暗号化されます。

サードパーティ製の Type 3 または Type 2 の JDBC ドライバを使用する方法もあります。データベースミドルウェアおよびクライアント API は、通常、暗号化された転送メカニズムを提供します。

B.9 複数値属性のマッピング

質問 : 複数値属性を単一値データベースフィールドにマッピングするにはどうすればいいですか ?

回答 : 99 ページの 「複数値属性から単一値データベースフィールドへのマッピング」 を参照してください。

B.10 ガーベッジ文字列の同期

質問 : ドライバがガーベッジ文字列を同期するのはなぜですか ?

回答 : おそらく、データベースおよびサードパーティ製のドライバが、互換性のない文字エンコードを使用しています。サードパーティ製のドライバで使用する文字エンコードを変更してください。

詳細については、Sun によって定義されている文字エンコード値 (http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html) を参照してください。

B.11 複数の JDBC 用ドライバインスタンスの実行

質問 : 複数の JDBC 用ドライバインスタンスを同じドライバセットで実行するにはどうすればいいですか ? それらのインスタンスでは、同じサードパーティ製の JBDC ドライバ (たとえば、Oracle JDBC ドライバまたは IBM DB2 Type 3 JDBC ドライバ) の異なるバージョンを使用する必要があります。

回答 : リモートローダを使用して、個々の JDBC 用ドライバを別々の Java Virtual Machine (JVM) でロードします。同じ JVM でローカルに実行すると、同じサードパーティ製で、バージョンが異なるクラスが衝突します。

サポートされているデータタイプ

C

JDBC 用ドライバは、すべての JDBC 1 データタイプ、および JDBC 2 データタイプの小さいサブセットを同期できます。JDBC データタイプがデータベースのネイティブデータタイプにどのようにマップされるかは、サードパーティ製のドライバによって決まります。

次のリストは、サポートされている JDBC 1 java.sql.Types (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) を示します。

- 数値タイプ :
  - java.sql.Types.BIGINT
  - java.sql.Types.BIT
  - java.sql.Types.DECIMAL
  - java.sql.Types.DOUBLE
  - java.sql.Types.NUMERIC
  - java.sql.Types.REAL
  - java.sql.Types.FLOAT
  - java.sql.Types.INTEGER
  - java.sql.Types.SMALLINT
  - java.sql.Types.TINYINT
- 文字列タイプ :
  - java.sql.Types.CHAR
  - java.sql.Types.LONGCHAR
  - java.sql.Types.VARCHAR
- 時間タイプ :
  - java.sql.Types.DATE
  - java.sql.Types.TIME
  - java.sql.Types.TIMESTAMP
- バイナリタイプ :
  - java.sql.Types.BINARY
  - java.sql.Types.VARBINARY
  - java.sql.Types.LONGVARBINARY

次のリストは、サポートされている JDBC 2 java.sql.Types (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html) を示します。

- Large Object (LOB) タイプ :
  - java.sql.Types.CLOB
  - java.sql.Types.BLOB

java.sql.DatabaseMetaData のメソッド

D

この節では、java.sql.DatabaseMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DatabaseMetaData.html) の必須およびオプションのメソッドをリストしています。

次の JDBC 1 メソッドは、「同期フィルタ」パラメータが［すべてのテーブル / ビューを除外］以外に設定されている場合に必須です。

- getColumns(java.lang.String catalog, java.lang.String schemaPattern, java.lang.String tableNamePattern, java.lang.String columnNamePattern):java.sql.ResultSet
- getPrimaryKeys(java.lang.String catalog, java.lang.String schema, java.lang.String table):java.sql.ResultSet
- getTables(java.lang.String catalog, java.lang.String schemaPattern, java.lang.String tableNamePattern, java.lang.String[] types):java.sql.ResultSet
- storesLowerCaseIdentifiers():boolean
- storesMixedCaseIdentifiers():boolean
- storesUpperCaseIdentifiers():boolean

オプションの JDBC 1 メソッド :

- dataDefinitionCausesTransactionCommit():boolean
- dataDefinitionIgnoredInTransactions():boolean
- getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern):java.sql.ResultSet
- getDatabaseProductName():java.lang.String
- getDatabaseProductVersion():java.lang.String
- getDriverMajorVersion():int
- getDriverMinorVersion():int
- getDriverName():java.lang.String
- getDriverVersion():java.lang.String
- getExportedKeys(java.lang.String catalog, java.lang.String schema, java.lang.String table):java.sql.ResultSet
- getMaxStatements():int
- getMaxConnections():int
- getMaxColumnsInSelect():int
- getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern):java.sql.ResultSet
- getSchemas():java.sql.ResultSet
- getTableTypes():java.sql.ResultSet
- getUserName():java.lang.String

- supportsColumnAliasing():bolean
- supportsDataDefinitionAndDataManiuplationTransactions():boolean
- supportsDataManipulationTransactionsOnly():boolean
- supportsLimitedOuterJoins():boolean
- supportsMultipleTransactions():boolean
- supportsSchemasInDataManipulation():boolean
- supportsSchemasInProcedureCalls():boolean
- supportsTransactionIsolationLevel(int level):boolean
- supportsTransactions():boolean

オプションの JDBC 2 メソッド :

- supportsBatchUpdates():boolean

オプションの JDBC 3 メソッド :

- supportsGetGeneratedKeys():boolean

JDBC インタフェースのメソッド

E

この節では、JDBC 用ドライバが使用する JDBC インタフェースのメソッド (java.sql.DatabaseMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DatabaseMetaData.html) 以外のメソッド) をリストします。メソッドはクラス別にまとめられています。

たいていの場合、サードパーティ製の JDBC ドライバのベンダは、メソッド別に不具合または既知の問題をリストしています。以下のメソッドと、サードパーティ製の JDBC ドライバのマニュアルを合わせて使用して、相互運用で発生する問題をトラブルシューティングまたは予測できます。

- java.sql.DriverManager (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DriverManager.html)
- java.sql.CallableStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/CallableStatement.html)
- java.sql.Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html)
- java.sql.PreparedStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html)
- java.sql.ResultSet (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html)
- java.sql.ResultSetMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSetMetaData.html)
- java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html)
- java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html)

次の表は、JDBC 用ドライバが使用する java.sql.DriverManager (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DriverManager.html) のメソッドのリストです。

表 *E-1* *java.sql.DriverManager* のメソッド

メソッドの署名	JDBC のバージョン	必須？
getConnection(String url, java.util.Properties info):java.sql.Connection	1	○ [1]
getConnection(String url, java.util.Properties info):java.sql.Connection	1	○ [1]
setLogStream(java.io.PrintStream out):void	1	×

[1] いずれか一方のメソッド。

次の表は、JDBC 用ドライバが使用する java.sql.CallableStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/CallableStatement.html) のメソッドのリストです。

表 *E-2* *java.sql.CallableStatement* のメソッド

メソッドの署名	JDBC のバージョン	必須？
getBigDecimal(int parameterIndex, int scale):java.math.BigDecimal	1	○
getBoolean(int parameterIndex):boolean	1	○
getBoolean(String parameterName):boolean	3	×
getByte(int parameterIndex):byte	1	○
getByte(String parameterName):byte	3	×
getBytes(int parameterIndex):byte[]	1	○
getBytes(String parameterName):byte[]	3	×
getDate(int parameterIndex):java.sql.Date	1	○
getDate(String parameterName):java.sql.Date	3	×
getDouble(int parameterIndex):double	1	○
getDouble(String parameterName):double	3	×
getFloat(int parameterIndex):float	1	○
getFloat(String parameterName):float	3	×
getInt(int parameterIndex):int	1	○
int getInt(String parameterName)	3	×
getLong(int parameterIndex):long	1	○
getLong(String parameterName):long	3	×
getShort(int parameterIndex):short	1	○
getShort(String parameterName):short	3	×
getString(int parameterIndex):String	1	○
getString(String parameterName):String	3	×
getTime(int parameterIndex):java.sql.Time	1	○
getTime(String parameterName):java.sql.Time	3	×
getTimestamp(int parameterIndex):java.sql.Timestamp	1	○
getTimestamp(String parameterName):java.sql.Timestamp	3	×
registerOutParameter(int parameterIndex, int sqlType):void	1	○
wasNull():boolean	1	○

次の表は、JDBC 用ドライバが使用する java.sql.Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html) のメソッドのリストです。

表 ***E-3*** *java.sql.Connection Methods*

メソッドの署名	JDBC のバージョン	必須？
close():void	1	○
commit():void	1	×
createStatement():java.sql.Statement	1	○
getAutoCommit():boolean	1	×
getMetaData():java.sql.DatabaseMetaData	1	○
getTransactionIsolation():int	1	×
getWarnings():java.sql.SQLWarning	1	×
isClosed():boolean	1	×
prepareCall(String sql):java.sql.CallableStatement	1	×
prepareStatement(String sql):java.sql.PreparedStatement	1	○
rollback():void	1	×
setAutoCommit(boolean autoCommit):void	1	×
setTransactionIsolation(int level):void	1	×

次の表は、JDBC 用ドライバが使用する java.sql.PreparedStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html) のメソッドのリストです。

表 ***E-4*** *java.sql.PreparedStatement のメソッド*

メソッドの署名	JDBC のバージョン	必須？
clearParameters() :void	1	×
execute():boolean	1	○
executeQuery():java.sql.ResultSet	1	○
executeUpdate():int	1	○
setBigDecimal(int parameterIndex, java.math.BigDecimal x):void	1	○
setBoolean(int parameterIndex, boolean x):void	1	○
setByte(int parameterIndex, byte x):void	1	○
setBytes(int parameterIndex, byte x[]):void	1	○
setDate(int parameterIndex, java.sql.Date x):void	1	○
setDouble(int parameterIndex, double x):void	1	○
setFloat(int parameterIndex, float x):void	1	○
setInt(int parameterIndex, int x):void	1	○

メソッドの署名	JDBC のバージョン	必須？
setLong(int parameterIndex, long x):void	1	○
setNull(int parameterIndex, int sqlType):void	1	○
setShort(int parameterIndex, short x):void	1	○
setString(int parameterIndex, String x):void	1	○
setTime(int parameterIndex, java.sql.Time x):void	1	○
setTimestamp(int parameterIndex, java.sql.Timestamp x):void	1	○

次の表は、JDBC 用ドライバが使用する java.sql.ResultSet (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html) のメソッドのリストです。

表 ***E-5*** *java.sql.ResultSet* のメソッド

メソッドの署名	JDBC のバージョン	必須？
close():void	1	○
getBigDecimal(int columnIndex, int scale):java.math.BigDecimal	1	○
getBigDecimal(String columnName, int scale):java.math.BigDecimal	1	○
getBinaryStream(int columnIndex):java.io.InputStream	1	○
getBinaryStream(String columnName)java.io.InputStream	1	○
getBoolean(int columnIndex):boolean	1	○
getBoolean(String columnName):boolean	1	○
getByte(int columnIndex):byte	1	○
getByte(String columnName):byte	1	○
getBytes(int columnIndex):byte[]	1	○
getBytes(String columnName):byte[]	1	○
getDate(int columnIndex):java.sql.Date	1	○
getDate(String columnName)java.sql.Date	1	○
getFloat(int columnIndex):float	1	○
getFloat(String columnName):float	1	○
getInt(int columnIndex):int	1	○
getInt(String columnName):int	1	○
getLong(int columnIndex):long	1	○
getLong(String columnName):long	1	○
getMetaData():java.sql.ResultSetMetaData	1	×

メソッドの署名	JDBC のバージョン	必須？
getShort(int columnIndex):short	1	○
getShort(String columnName):short	1	○
getString(int columnIndex):String	1	○
getString(String columnName):String	1	○
getTime(int columnIndex):java.sql.Time	1	○
getTime(String columnName):java.sql.Time	1	○
getTimestamp(int columnIndex):java.sql.Timestamp	1	○
getTimestamp(String columnName):java.sql.Timestamp	1	○
getWarnings():java.sql.SQLWarning	1	×

次の表は、JDBC 用ドライバが使用する java.sql.ResultSetMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSetMetaData.html) のメソッドのリストです。

表 *E-6* *java.sql.ResultSetMetaData* のメソッド

メソッドの署名	JDBC のバージョン	必須？
getColumnCount():int	1	○
getColumnName(int column):String	1	×
getColumnType(int column):int	1	×

次の表は、JDBC 用ドライバが使用する java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html) のメソッドのリストです。

表 *E-7* *java.sql.Statement* のメソッド

メソッドの署名	JDBC のバージョン	必須？
addBatch(java.lang.String sql):void	2	×
clearBatch():void	2	×
clearWarnings():void	1	×
close():void	1	○
execute(java.lang.String sql):boolean	1	○
executeBatch():int[]	2	×
executeUpdate(String sql):int	1	○
executeQuery(String sql):java.sql.ResultSet	1	○

メソッドの署名	JDBC のバージョン	必須？
getGeneratedKeys():java.sql.ResultSet	3	×
getMoreResults():boolean	1	×
getResultSet():java.sql.ResultSet	1	○
getUpdateCount():int	1	×
getWarnings():java.sql.SQLWarning	1	×

次の表は、JDBC 用ドライバが使用する java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html) のメソッドのリストです。

表 *E-8* *java.sql.Timestamp* のメソッド

メソッドの署名	JDBC のバージョン	必須？
getNanos():int	1	○
getTime():long	1	○
setNanos(int n):void	1	○
setTime(long time):void	1	○
toString ():String	1	○

サードパーティ製の JDBC ドライバ記述子の DTD

F

この節には、サードパーティ製の JDBC 記述子ファイルの DTD が含まれています。

```
<?xml version="1.0" encoding="UTF-8"?> <!ELEMENT actions (exec-sql |
check-for-closed-connection | fetch-metadata | rollback)*> <!ELEMENT
add-default-values-on-view-insert (#PCDATA)> <!ELEMENT authentication
(sql-state | error-code | sql-state-class | error-code-range |
actions)*> <!ELEMENT check-for-closed-connection EMPTY> <!ELEMENT
column-position-comparator (#PCDATA)> <!ELEMENT connection-properties
(property*)> <!ELEMENT connectivity (sql-state | error-code | sql-
state-class | error-code-range | actions)*> <!ELEMENT current-
timestamp-stmt (#PCDATA)> <!ELEMENT error-code (value)> <!ATTLIST
error-code description CDATA #IMPLIED > <!ELEMENT error-code-range
(from, to)> <!ATTLIST error-code-range description CDATA #IMPLIED >
<!ELEMENT errors (connectivity | authentication | retry | fatal)*>
<!ELEMENT exclude-table-filter (#PCDATA)> <!ELEMENT exec-sql
(#PCDATA)> <!ELEMENT fatal (sql-state | error-code | sql-state-class |
error-code-range | actions)*> <!ELEMENT fetch-metadata EMPTY>
<!ELEMENT from (#PCDATA)> <!ELEMENT function-return-method (#PCDATA)>
<!ELEMENT handle-stmt-results (#PCDATA)> <!ELEMENT identity (name?,
target-database?, jdbc-type?, jdbc-class?)> <!ELEMENT import
(#PCDATA)> <!ELEMENT imports (import*)> <!ELEMENT include-table-filter
(#PCDATA)> <!ELEMENT jdbc-class (#PCDATA)> <!ELEMENT jdbc-driver
(imports?, identity, (metadata-override | connection-properties | sql-
type-map | options | errors)*)> <!ELEMENT jdbc-type (#PCDATA)>
<!ELEMENT key (#PCDATA)> <!ELEMENT left-outer-join-operator (#PCDATA)>
<!ELEMENT lock-generator-class (#PCDATA)> <!ELEMENT metadata-override
(supports-schemas-in-procedure-calls?)> <!ELEMENT minimal-metadata
(#PCDATA)> <!ELEMENT name (#PCDATA)> <!ELEMENT options (lock-
generator-class | supports-schemas-in-metadata-retrieval | time-
translator-class | column-position-comparator | use-manual-
transactions | minimal-metadata | transaction-isolation-level | use-
single-connection | exclude-table-filter | include-table-filter |
left-outer-join-operator | current-timestamp-stmt | add-default-
values-on-view-insert | reuse-statements | function-return-method |
handle-stmt-results)*> <!ELEMENT property (key, value)> <!ELEMENT
retry (sql-state | error-code | sql-state-class | error-code-range |
actions)*> <!ELEMENT reuse-statements (#PCDATA)> <!ELEMENT rollback
EMPTY> <!ELEMENT sql-state (value)> <!ATTLIST sql-state description
CDATA #IMPLIED > <!ELEMENT sql-state-class (value)> <!ATTLIST sql-
state-class description CDATA #IMPLIED > <!ELEMENT sql-type-map
(type*)> <!ELEMENT supports-schemas-in-metadata-retrieval (#PCDATA)>
<!ELEMENT supports-schemas-in-procedure-calls (#PCDATA)> <!ELEMENT
target-database (#PCDATA)> <!ELEMENT time-translator-class (#PCDATA)>
<!ELEMENT to (#PCDATA)> <!ELEMENT transaction-isolation-level
(#PCDATA)> <!ELEMENT type (from, to)> <!ELEMENT use-manual-
```

```
transactions (#PCDATA)> <!ELEMENT use-single-connection (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

サードパーティ製の JDBC ドライバ記述子のインポートの DTD

G

この節には、サードパーティ製の JDBC 記述子インポートファイルの DTD が含まれています。

```
<?xml version="1.0" encoding="UTF-8"?> <!ELEMENT actions (exec-sql |
check-for-closed-connection | fetch-metadata | rollback)*> <!ELEMENT
add-default-values-on-view-insert (#PCDATA)> <!ELEMENT authentication
(sql-state | error-code | sql-state-class | error-code-range |
actions)*> <!ELEMENT check-for-closed-connection EMPTY> <!ELEMENT
column-position-comparator (#PCDATA)> <!ELEMENT connection-properties
(property*)> <!ELEMENT connectivity (sql-state | error-code | sql-
state-class | error-code-range | actions)*> <!ELEMENT current-
timestamp-stmt (#PCDATA)> <!ELEMENT error-code (value)> <!ATTLIST
error-code description CDATA #IMPLIED > <!ELEMENT error-code-range
(from, to)> <!ATTLIST error-code-range description CDATA #IMPLIED >
<!ELEMENT errors (connectivity | authentication | retry | fatal)*>
<!ELEMENT exclude-table-filter (#PCDATA)> <!ELEMENT exec-sql
(#PCDATA)> <!ELEMENT fatal (sql-state | error-code | sql-state-class |
error-code-range | actions)*> <!ELEMENT fetch-metadata EMPTY>
<!ELEMENT from (#PCDATA)> <!ELEMENT function-return-method (#PCDATA)>
<!ELEMENT handle-stmt-results (#PCDATA)> <!ELEMENT include-table-
filter (#PCDATA)> <!ELEMENT jdbc-driver (metadata-override |
connection-properties | sql-type-map | options | errors)*> <!ELEMENT
key (#PCDATA)> <!ELEMENT left-outer-join-operator (#PCDATA)> <!ELEMENT
lock-generator-class (#PCDATA)> <!ELEMENT metadata-override (supports-
schemas-in-procedure-calls?)> <!ELEMENT minimal-metadata (#PCDATA)>
<!ELEMENT options (lock-generator-class | supports-schemas-in-
metadata-retrieval | time-translator-class | column-position-
comparator | use-manual-transactions | minimal-metadata | transaction-
isolation-level | use-single-connection | exclude-table-filter |
include-table-filter | left-outer-join-operator | current-timestamp-
stmt | add-default-values-on-view-insert | reuse-statements |
function-return-method | handle-stmt-results)*> <!ELEMENT property
(key, value)> <!ELEMENT retry (sql-state | error-code | sql-state-
class | error-code-range | actions)*> <!ELEMENT reuse-statements
(#PCDATA)> <!ELEMENT rollback EMPTY> <!ELEMENT sql-state (value)>
<!ATTLIST sql-state description CDATA #IMPLIED > <!ELEMENT sql-state-
class (value)> <!ATTLIST sql-state-class description CDATA #IMPLIED >
<!ELEMENT sql-type-map (type*)> <!ELEMENT supports-schemas-in-
metadata-retrieval (#PCDATA)> <!ELEMENT supports-schemas-in-procedure-
calls (#PCDATA)> <!ELEMENT time-translator-class (#PCDATA)> <!ELEMENT
to (#PCDATA)> <!ELEMENT transaction-isolation-level (#PCDATA)>
<!ELEMENT type (from, to)> <!ELEMENT use-manual-transactions
(#PCDATA)> <!ELEMENT use-single-connection (#PCDATA)> <!ELEMENT value
(#PCDATA)>
```

データベース記述子の DTD

H

この節には、データベース記述子ファイルの DTD が含まれています。

```
<?xml version="1.0" encoding="UTF-8"?> <!ELEMENT add-default-values-
on-view-insert (#PCDATA)> <!ELEMENT column-position-comparator
(#PCDATA)> <!ELEMENT current-timestamp-stmt (#PCDATA)> <!ELEMENT
database (imports?, identity, options?)> <!ELEMENT exclude-table-
filter (#PCDATA)> <!ELEMENT function-return-method (#PCDATA)>
<!ELEMENT handle-stmt-results (#PCDATA)> <!ELEMENT include-table-
filter (#PCDATA)> <!ELEMENT identity (name?, regex-name?, regex-
version?)> <!ELEMENT import (#PCDATA)> <!ELEMENT imports (import*)>
<!ELEMENT left-outer-join-operator (#PCDATA)> <!ELEMENT lock-
generator-class (#PCDATA)> <!ELEMENT minimal-metadata (#PCDATA)>
<!ELEMENT name (#PCDATA)> <!ELEMENT options (lock-generator-class |
supports-schemas-in-metadata-retrieval | time-translator-class |
column-position-comparator | use-manual-transactions | minimal-
metadata | transaction-isolation-level | use-single-connection |
exclude-table-filter | include-table-filter | left-outer-join-operator
| current-timestamp-stmt | add-default-values-on-view-insert | reuse-
statements | function-return-method | handle-stmt-results)*> <!ELEMENT
regex-name (#PCDATA)> <!ELEMENT regex-version (#PCDATA)> <!ELEMENT
reuse-statements (#PCDATA)> <!ELEMENT supports-schemas-in-metadata-
retrieval (#PCDATA)> <!ELEMENT time-translator-class (#PCDATA)>
<!ELEMENT transaction-isolation-level (#PCDATA)> <!ELEMENT use-manual-
transactions (#PCDATA)> <!ELEMENT use-single-connection (#PCDATA)>
```

データベース記述子のインポートの DTD

この節には、データベース記述子インポートファイルの DTD が含まれています。

```
<?xml version="1.0" encoding="UTF-8"?> <!ELEMENT add-default-values-
on-view-insert (#PCDATA)> <!ELEMENT column-position-comparator
(#PCDATA)> <!ELEMENT current-timestamp-stmt (#PCDATA)> <!ELEMENT
exclude-table-filter (#PCDATA)> <!ELEMENT function-return-method
(#PCDATA)> <!ELEMENT handle-stmt-results (#PCDATA)> <!ELEMENT include-
table-filter (#PCDATA)> <!ELEMENT database (options?)> <!ELEMENT left-
outer-join-operator (#PCDATA)> <!ELEMENT lock-generator-class
(#PCDATA)> <!ELEMENT minimal-metadata (#PCDATA)> <!ELEMENT options
(lock-generator-class | supports-schemas-in-metadata-retrieval | time-
translator-class | column-position-comparator | use-manual-
transactions | minimal-metadata | transaction-isolation-level | use-
single-connection | exclude-table-filter | include-table-filter |
left-outer-join-operator | current-timestamp-stmt | add-default-
values-on-view-insert | reuse-statements | function-return-method |
handle-stmt-results)*> <!ELEMENT reuse-statements (#PCDATA)> <!ELEMENT
supports-schemas-in-metadata-retrieval (#PCDATA)> <!ELEMENT time-
translator-class (#PCDATA)> <!ELEMENT transaction-isolation-level
(#PCDATA)> <!ELEMENT use-manual-transactions (#PCDATA)> <!ELEMENT use-
single-connection (#PCDATA)>
```

ポリシーの例：トリガなしの将来のイベント処理

J

次の例では、“commence” 属性が存在し、以下を行うとします。

- いつイベントを処理する必要があるかを示すタイムスタンプ値の保持。
- 整数または Java 文字列のタイムスタンプ値の保持。49 ページの 「時間構文」 を参照してください。

```
<policy xmlns:Timestamp="http://www.novell.com/nxsl/java/
java.sql.Timestamp" xmlns:TimestampUtil="http://www.novell.com/nxsl/
java/com.novell.nds.dirxml.driver.jdbc.db.TimestampUtil"
xmlns:jdbc="urn:dirxml:jdbc"> <rule> <description>Get commencement
date from datasource.</description> <conditions> <and> <if-xpath
op="true">.</if-xpath> </and> </conditions> <actions> <do-set-local-
variable name="commence"> <arg-string> <token-src-attr class-
name="User" name="commence"/> </arg-string> </do-set-local-variable>
</actions> </rule>


<rule> <description>Break if commencement date unavailable.</
description> <conditions> <and> <if-local-variable name="commence"
op="equal"/> </and> </conditions> <actions> <do-break/> </actions> </
rule>


<rule> <description>Parse times.</description> <conditions> <and> <if-
xpath op="true">.</if-xpath> </and> </conditions> <actions> <do-set-
local-variable name="dbTime"> <arg-object> <token-xpath
expression="Timestamp:valueOf(@jdbc:database-local-time)"/> </arg-
object> </do-set-local-variable> <do-set-local-variable
name="eventTime"> <arg-object> <token-xpath
expression="Timestamp:valueOf($commence)"/> </arg-object> </do-set-
local-variable> </actions> </rule><rule> <description>Is commencement
date after database time?</description> <conditions> <and> <if-xpath
op="true">.</if-xpath> </and> </conditions> <actions> <do-set-local-
variable name="after"> <arg-string> <token-xpath
expression="TimestampUtil:after($eventTime, $dbTime)"/> </arg-string>
</do-set-local-variable> </actions> </rule>


<rule> <description>Retry if future event.</description> <conditions>
<and> <if-local-variable name="after" op="equal">true</if-local-
variable> </and> </conditions> <actions> <do-status level="retry">
<arg-string> <token-text xml:space="preserve">Future event detected.</
token-text> </arg-string> </do-status> </actions> </rule></policy>
```

最新のマニュアル

K

この節には、JDBC 用 Identity Manager ドライバについての新規および更新情報が含まれています。

マニュアルは、HTML と PDF の 2 つの形式で Web 上に提供されています。HTML および PDF のマニュアルはいずれもこの節に挙げるマニュアルの変更内容を反映した最新の状態になっています。

現在お持ちの PDF マニュアルが最新かどうかを確認する必要がある場合は、PDF ファイルの発行日を確認します。日付はタイトルページにあります。

新規マニュアルまたは更新されたマニュアルは、次の日付に発行されました。

K.1 2005 年 12 月 14 日

表 *K-1* *2005 年 12 月 14 日付の更新*

場所	変更内容
46 ページのセクション 4.3「ドライバパラメータ」	セクションを 1 レベル上の情報階層に移動したので、ドライバのパラメータを探しやすくなりました。
102 ページの 表 5-10	イベントタイプテーブルを更新しました。
103 ページのセクション 5.3.2「イベントタイプ」	表 5-12 とそれを説明する文を更新しました。
135 ページのセクション 6.4.9「Oracle Thin Client JDBC Driver」	「既知の問題」節を更新しました。

K.2 2006 年 4 月 24 日

表 *K-2* *2006 年 4 月 19 日付の更新*

場所	変更内容
17 ページの 「購読者チャネル」	「中間テーブル」ではなく、「ビュー」を表示するように図を修正しました。

K.3 2006 年 5 月 1 日

表 ***K-3*** *2006* 年 *5* 月 *1* 日付の更新

場所	変更内容
139 ページのセクション 6.5.1「IBM Toolbox for Java/JTOpen」	このトピックを追加しました。

K.4 2006 年 5 月 15 日

表 ***K-4*** *2006* 年 *5* 月 *12* 日付の更新

場所	変更内容
139 ページのセクション 6.5.1「IBM Toolbox for Java/JTOpen」	インポートされるサンプルの環境設定ファイル内の設定に値についての説明文を追加しました。値は、手動で入力する必要があります。自動的には追加されません。