

## 序文

Java および J2EE と共に使用する Caché クイックスタート・チュートリアルへようこそ。このチュートリアルでは、高速、効率的、柔軟な Caché データベースを Java および J2EE アプリケーションと連携して使用方法について簡単に説明します。

Java および J2EE アプリケーションと Caché を接続するには、以下の 3 つの異なる方法から選択できます。

1. Java バインディング機能: アプリケーションのオブジェクトを、データのオブジェクト表現 (Caché オブジェクト) に直接接続します。SQL や面倒なオブジェクト・リレーショナル・マッピングを行う必要はありません。
2. EJB バインディング機能: Caché クラスから自動的に Enterprise JavaBeans (EJB) を生成します。手間のかかる EJB 開発プロセスを手作業で行う必要はなく、しかも非常に効率的にデータにアクセスできます。
3. Caché JDBC ドライバ: Caché データへの標準リレーショナル・インタフェースを提供します。このドライバは、SQL および [java.sql](#) API と共に使用してください。

このチュートリアルは2つの章に分かれており、上記の EJB を除く機能を各章で説明します。それぞれの章は独立しているので、どのような順序でも学習を行うことができます。

- [「第 I 章: Java バインディング機能」](#)では、Java バインディング機能の使い方について学習します。
- [「第 II 章: Caché と JDBC」](#)では、Caché JDBC ドライバの使い方について学習します。

### Note:

Caché を Java および J2EE アプリケーションと共に使用する際の一般的な Java 関連システム要件については、[AppendixA: 第 I 章と第 II 章でのシステム要件] を参照してください。

チュートリアルの例と演習を実行するには、Java ソース・ファイルだけでなく、Caché アプリケーションである Contact Management アプリケーションが必要です。Java ファイルの説明と Contact Management アプリケーションのインストール方法については、[AppendixB: 例と演習のファイル] を参照してください。

Caché の概要と、Caché を使用した迅速なアプリケーション開発については、Caché ドキュメント: [\[Caché チュートリアル\]-\[Caché クイックスタート・チュートリアル\]](#) を参照してください。Caché オブジェクトの詳細は、Caché ドキュメント: [\[Caché 開発ガイド\]-\[Caché オブジェクトの使用法\]](#) を参照してください。

Caché をインストールすると、Java バインディング機能と Caché JDBC ドライバを紹介するコード・サンプルもインストールされます。このコードを含むソース・ファイルとクラス・ファイルは、`< cachesys > %Dev %java %samples` にあります。Windows で標準インストールしている場合、`< cachesys >` は `C: %InterSystems %Cache` になります。標準の Unix または Linux の環境では、`< cachesys >` は `/usr/cachesys` になります。

## 第 I 章：Java バインディング機能の概要

このチュートリアルの第 I 章では、Java バインディング機能について説明します。この章の学習を終えると、以下のことを実行できるようになります。

- Cache オブジェクトの基本的なオブジェクト指向機能を、SQL テーブルの対応するリレーショナル機能にマップする
- Java バインディング機能の基本アーキテクチャについて説明する
- Cache クラスの Java プロジェクションを生成する
- Java バインディング機能と簡単な Cache オブジェクトを使用して、Cache データへのアクセスおよび変更を行う Java コードを記述する
- Java バインディング機能と、Cache リレーションシップおよびコレクション・オブジェクトを使用して、Cache データへのアクセスおよび変更を行う Java コードを記述する
- Java バインディング機能と **CacheQuery** クラスを使用して、Cache データベースでクエリを実行する Java コードを記述する

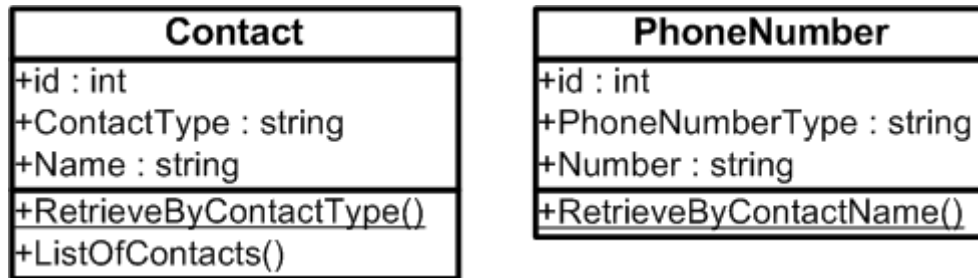
このチュートリアルには、サンプルの Cache アプリケーションである Contact Management アプリケーションに Java コードを接続する必要がある例と演習が含まれています。

### Note:

第 I 章の内容は、Cache および Cache オブジェクトの基礎知識だけでなく、基本的な Java プログラミング・スキルを持っていることを前提としています。Cache および Cache オブジェクトの詳細は、Cache ドキュメント: [\[Cache チュートリアル\]-\[Cache クイックスタート・チュートリアル\]](#) を参照してください。Java プログラミングの詳細は、Sun Microsystems の "[The Java Tutorial](#)" を参照してください。

## Contact Management アプリケーション

このチュートリアルの例と演習では、Java コードをサンプルの Caché アプリケーションに接続する必要があります。この Contact Management アプリケーションは、ユーザが連絡先とその電話番号のリストを保守管理できるようにするアプリケーションです。オブジェクト指向の視点から見ると、これは 2 つの Caché クラス、**Contact** と **PhoneNumber** で構成されています。以下の図は、それぞれのプロパティと処理を示しています。クラス間のリレーションシップについては、このチュートリアルで後述します。



**Contact** は、以下の要素で構成されています。

- `id`: Caché オブジェクト ID。各インスタンスは一意的な値を持ちます。スタジオ開発環境では、このプロパティはクラス定義に表示されません。
- `ContactType`: 連絡先のタイプ。この値には、“Business”と “Personal”だけを指定できます。
- `Name`: 連絡先の名前。この値には、任意の文字列を指定できます。
- `RetrieveByContactType`: 事前定義のクエリ。引数として `ContactType` 値を受け取ります。一致するすべての **Contact** インスタンスの `Name` 値を返します。
- `ListOfContacts`: クラス・メソッド。すべての **Contact** インスタンスの `id` 値を検索します。

**PhoneNumber** は、以下の要素で構成されています。

- `id`: Caché オブジェクト ID。各インスタンスは一意的な値を持ちます。スタジオ開発環境では、このプロパティはクラス定義に表示されません。
- `PhoneNumberType`: 電話番号のタイプ。この値には、“Business”、“Home”、“Mobile”および“Fax”を指定できます。
- `Number`: 電話番号。この値には、任意の文字列を指定できます。
- `RetrieveByContactName`: 事前定義のクエリ。引数として **Contact** `Name` 値を受け取ります。**Contact** に含まれているすべての **PhoneNumber** インスタンスの `id` 値を返します。

### Note:

Contact Management アプリケーションのインストール方法と、演習と例を実行するために用意されている Java ファイルの説明については、[AppendixB: 例と演習のファイル]を参照してください。

## 統一データ・アーキテクチャ

Cache の多次元データベース・エンジンが、統一データ・アーキテクチャ(UDA) を形成します。この UDA を使用することで、Cache では、オブジェクト指向表現とリレーショナル表現の両方で、データベースに格納するデータを表現できます。つまり、データを同時に、行、列、およびストアド・プロシージャを持つ SQL テーブルとしても、プロパティとメソッドを持つ Cache オブジェクトとしても扱うことができるのです。Cache データベースの各テーブルは、それぞれ 1 つの Cache クラスに対応します。テーブルの各行は、クラスの 1 つのインスタンス(Cache オブジェクト)に対応します。以下のテーブルは、Cache オブジェクトの基本的なオブジェクト指向機能と、対応する SQL テーブルのリレーショナル機能とのマッピングを示しています。

### オブジェクト指向機能とリレーショナル機能

Cache オブジェクト	SQL テーブル
パッケージ	スキーマ
クラス	テーブル
オブジェクト・インスタンス	テーブル行
プロパティ	テーブル列
クラス・メソッド	ストアド・プロシージャ
リレーションシップ	外部キー
埋め込みオブジェクト	列サブセット

Java バインディング機能を使用すると、Java アプリケーションは Cache オブジェクトにアクセスできるようになります。

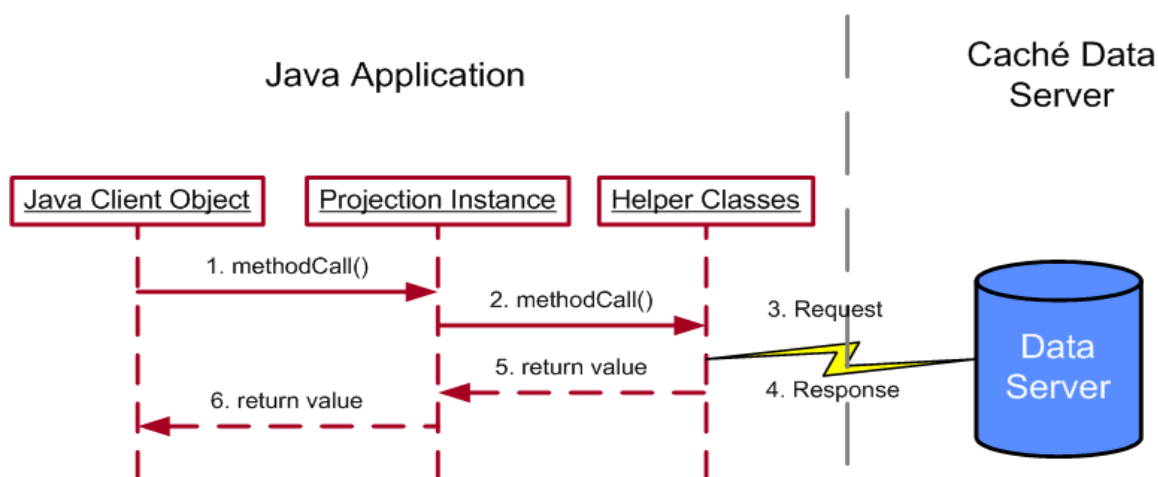
#### Note:

Cache 統一データ・アーキテクチャの詳細は、Cache ドキュメント: [\[はじめに\]-\[Cache 入門\]-\[オブジェクト、SQL、統一データ・アーキテクチャ\]-\[統一データ・ディクショナリ\]](#) を参照してください。

## Java プロジェクション

Java アプリケーションから見ると、Java プロジェクションは Java バインディング機能の最も重要な要素です。Java プロジェクションは、Java アプリケーション内で Cache クラスのプロキシとしての役割を果たす Pure Java クラスです。Cache データ・サーバ上で、Java クライアント・オブジェクトと Cache オブジェクト間の通信リンクを提供します。通常、Java アプリケーションでは、データベースの各 Cache クラスに 1 つの Java プロジェクションが含まれています。Java クライアント・オブジェクトは、他の Java クラスと同様に、プロジェクションを使用して、インスタンスの生成、メソッドの呼び出しなどを行います。そして、プロジェクション・クラスは一連のヘルパー・クラスと連携して、Java クライアントの要求を Cache クラスに渡します。

以下の図は、Java クライアント・オブジェクト、Java プロジェクションのインスタンス、Cache データ・サーバ間の通信手順を表しています。



### Note:

Java プロジェクションの詳細は、Cache ドキュメント: [\[Cache 言語バインディング\]](#)-[\[Cache での Java の使用法\]](#)-[\[Java クライアント・クラス・リファレンス\]](#) を参照してください。

## その他の Java バインディング・コンポーネント

Java プロジェクションの他に、Java バインディング機能には以下のコンポーネントも含まれています。

### Java バインディング機能のコンポーネント

要素	概要
Caché Java クラス・ジェネレータ	Caché コンパイラの拡張機能。対応する Caché クラスのコンパイル中に Java プロジェクションを生成します。
Caché オブジェクト・サーバ	Java クライアントと Caché データベース・サーバ間の通信を管理する高性能なサーバ・プロセス。
Caché Java クラス・パッケージ	Pure Java クラスのパッケージ。プロジェクションが Caché データベースとの通信に使用する“ヘルパー・クラス”が含まれます。このクラスに関するドキュメントは <cachsys>¥dev¥java¥doc にあります。index.html をダブルクリックして参照してください。

#### Note:

Java バインディング機能のアーキテクチャの詳細は、Caché ドキュメント: [\[Caché 言語バインディング\]](#)-[\[CachéでのJavaの使用法\]](#)-[\[Caché Javaバインディング\]](#)を参照してください。

## Java プロジェクションの生成

Caché クラスの Java プロジェクションを生成するには、Projection プロパティをクラス定義に追加します。クラスをコンパイルすると、Java クラス・ジェネレータによりプロジェクションのソース・ファイルが自動生成されます。ここでは、スタジオの新規プロジェクション・ウィザードを使用して、Java プロジェクションを **JavaTutorial.Contact** に追加する方法を説明します。Caché インストール内のネームスペースにこのクラスをロードしていなかった場合は、[AppendixB: 例と演習のファイル]で、ロード方法を参照してください。

1. Caché キューブをクリックして[スタジオ]を選択し、Caché スタジオを開きます。
2. [ファイル]メニューの[ネームスペース変更]をクリックし、次に **JavaTutorial.Contact** を含むネームスペースをクリックして、このネームスペースに接続します。
3. 左側のワークスペース・ウィンドウで、JavaTutorial フォルダを開いて、Contact をダブルクリックします。クラス・エディタに、**Contact** のクラス定義が表示されます。
4. メニュー・バーの[クラス]、[追加]、[新規プロジェクション]の順にクリックします。
5. プロジェクションの名前を入力します (例えば、**JavaProjection**)。[次へ] をクリックします。
6. [プロジェクション・タイプ]を選択します。ドロップダウン・リストの [%Projection.Java]をクリックします。
7. プロジェクションの ROOTDIR パラメータの値を入力します (例えば、< cachesys > ¥dev ¥java ¥tutorials ¥javaandj2ee)。Caché により、プロジェクションのソース・ファイルを含む新しいサブディレクトリが、このディレクトリに作成されます。**[完了]** をクリックします。
8. クラス・エディタによって、以下の宣言が **JavaTutorial.Contact** 定義に追加されます。

```
Projection JavaProjection As
```

```
%Projection.Java (ROOTDIR = "< cachesys > ¥dev ¥tutorials ¥javaandj2ee");
```

新規プロジェクション・ウィザードではなく、クラス・エディタ・ウィンドウを使用して、直接プロジェクションを追加することもできます。

9. メニュー・バーから[ビルド]、[コンパイル]の順に選択して、**Contact** をコンパイルします。
10. Java クラス・ジェネレータにより、プロジェクションの Java ソース・ファイル Contact.java が生成されます。 **Contact** は **PhoneNumber** に依存しているため、**PhoneNumber** の Java プロジェクション・ソース・ファイル PhoneNumber.java も自動的に生成されます。両方のファイルは、ROOTDIR パラメータの値として指定したディレクトリのサブディレクトリである JavaTutorial に置かれます。
11. Java コンパイラを使用して、Java プロジェクションをコンパイルします。

**Note:**

Java プロジェクション・ソース・ファイルは、パッケージ名を反映している名前のディレクトリに置かれます。通常 Java プラットフォームは、ソース・ファイルとクラス・ファイルのどちらも、パッケージ名と同じ名前のディレクトリで検索するためです。既定では、プロジェクション・パッケージ名は Caché パッケージ名と同じになります。

プロジェクションのクラス・ファイルを含むディレクトリは、アプリケーションのクラスパス上になければなりません。

Cachéスタジオの使用に関する詳細は、Cachéドキュメント: [\[Cachéツールとユーティリティ\]-\[Cachéスタジオの使用法\]](#)を参照してください。

## プロジェクトの詳細

以下のテーブルは、Cache クラスの要素が Java に投影される方法の詳細です。

Cache クラス	Java プロジェクト
パッケージ	既定では、プロジェクトは Cache クラスのパッケージと同じ名前のパッケージに置かれます。プロジェクトの既定パッケージを変更するには、Cache クラスの JAVAPACKAGE パラメータを設定します。
プロパティ	プロジェクトには、Cache プロパティのアクセサ・メソッドである、getPropertyName と setPropertyName が含まれます。
メソッド	プロジェクトには、Java アプリケーションに呼び出されたときに、Cache サーバ上に実際に実装されているメソッドを呼び出す、スタブ・バージョンの Cache メソッドが含まれます。
パッケージ名	通常、プロジェクトは Cache パッケージ名を保持します。ただし、例外として、“%”記号は“_”に置き換えられ、Cache <b>%Library</b> パッケージは <b>com.intersys.objects</b> になります。
クラス名とメソッド名	通常、プロジェクトはクラス名とメソッド名を保持します。ただし、例外として、先頭の“%”記号は“sys_”に置き換えられます。名前が Java 予約語の場合は、“_”が名前の前に付きます。 <b>%Library</b> パッケージのメソッド名およびクラス名は、先頭の“%”が“_”に置き換えられ、最初の文字が小文字に変換されます。
変数名	通常、プロジェクトは変数名を保持します。ただし、例外として、先頭の“%”記号は“_”に置き換えられます。名前が Java 予約語の場合は、“_”が名前の前に付きます。

## データ型

Java クラス・ジェネレータは、Cache データ型クラスの CLIENTDATATYPE 属性の値に従って、Cache データ型を Java データ型に投影します。以下のテーブルは、一部のマッピングを示しています。

Cache データ型	CLIENTDATATYPE	Java データ型
%String	VARCHAR	java.lang.String
%Name	VARCHAR	java.lang.String
%Integer	INTEGER	java.lang.Integer
%Float	DOUBLE	java.lang.Double
%Date	DATE	java.sql.Date

### Note:

Cache データ型クラスと CLIENTDATATYPE 属性の詳細は、Cache ドキュメント: [\[Cache 開発ガイド\]-\[Cache オブジェクトの使用法\]-\[データ型\]](#) を参照してください。

## Caché への接続

Java プロジェクションを使用して Caché データベースと通信するためには、データベース接続を構築する必要があります。接続タイプには、通常と軽量の 2 種類があります。軽量接続は通常接続よりも高速ですが、サポートされている機能が若干限定されます。

接続タイプ	詳細
通常接続	<ul style="list-style-type: none"> <li>• Caché サーバ上でクラス・メソッドとインスタンス・メソッドの両方のメソッドの実行をサポートします。</li> <li>• オブジェクトの複数のインスタンスを同時に開くことができます。</li> <li>• <code>com.intersys.objects.CacheDatabase</code> の <code>getDatabase</code> メソッドにより作成されます。</li> </ul>
軽量接続	<ul style="list-style-type: none"> <li>• 通常接続よりも高速です。</li> <li>• Caché クラス・メソッドの実行をサポートします。</li> <li>• 永続性に関係するか、プロパティを取得または更新する Caché インスタンス・メソッドの実行をサポートします。</li> <li>• 永続性に関係しない Caché インスタンス・メソッドや、単純なプロパティの取得または更新以上の操作を実行する Caché インスタンス・メソッドの実行をサポートしません。</li> <li>• オブジェクトの複数のインスタンスを同時に開くことはできません。</li> <li>• <b><code>com.intersys.objects.CacheDatabase</code></b> の <b><code>getLightDatabase</code></b> メソッドにより作成されます。</li> </ul>

以下の Java クライアント・メソッドは、`getDatabase` を使用して通常接続を構築します。

`getDatabase` は、`CacheException` タイプのチェック済み例外を返します。このメソッドには、サーバの IP アドレスを指定する URL、データベースのポート番号、およびデータを格納する Caché ネームスペースの名前を、引数として指定する必要があります。また、Caché ネームスペースのユーザ名とパスワードもそれぞれ引数として指定する必要があります。

```
public class BindingExamples {
    public static Database createConnection() throws CacheException {
        String url="jdbc:Cache://localhost:1972/USER";
        String username="_SYSTEM";
        String pwd="SYS";
        Database db = CacheDatabase.getDatabase(url, username, pwd);
        return db;
    }
}
```

**Note:**

この章の Java バインディング例の Java ファイルでは、USER ネームスペースに接続するための接続 URL を使用しています。Contact Management アプリケーションの Cache クラスを別のネームスペースにインストールした場合は、コード例の接続 URL をインストール先に応じて変更してください。つまり、URL を jdbc:Cache://localhost:1972/<namespace name> に変更します。

## オブジェクト：生成、オープン、および保持

**CacheDatabase** オブジェクトを参照できるようになると、プロジェクションを使用して、新しいオブジェクトの生成、既存オブジェクトのオープン、および新しいオブジェクトや変更されたオブジェクトのデータベースへの保存を行うことができます。

新しいオブジェクトを生成するには、Java プロジェクション・クラスに含まれているコンストラクタを使用します。コンストラクタは、Java プロジェクション・クラスのインスタンスを Java クライアントに生成すると共に、対応する Cache クラスのインスタンスを Cache サーバにも生成します。インスタンスを生成しても、そのインスタンスは自動的にデータベースに保存されません。オブジェクトが保存されるまで、Cache ではそのインスタンスにオブジェクト ID を割り当てません。

**CacheDatabase** オブジェクトへの参照を、引数としてコンストラクタに指定する必要があります。以下のコードは、**Contact** クラスのコンストラクタを呼び出します。

```
Contact contact = new Contact(db);
```

保存されたオブジェクトをオープン、つまり Java クライアントと Cache サーバの両方のメモリに格納するには、そのオブジェクトのプロジェクション・クラスの **\_open** メソッドを使用します。オブジェクト ID を使用してオープンするオブジェクトを選択します。以下のコードでは、オブジェクト ID 1 の **Contact** インスタンスをオープンしています。オブジェクトは、適切な型に明示的にキャストする必要があります。

```
Contact contact = (Contact) (Contact._open(db, new Id(1)))
```

Cache オブジェクトをデータベースに保存するには、その Java プロジェクションの **save** メソッドを呼び出します。オブジェクトがまだ保存されていない場合には、Cache によりそのオブジェクトにオブジェクト ID が割り当てられます。プロジェクションの **save** メソッドを実行すると、対応する Cache オブジェクトの **%Save** メソッドも実行されます。処理が成功した場合、**save** メソッドは 1 を返します。

```
int success = contact.save();
```

### Note:

上記すべてのメソッドは、**CacheException** タイプのチェック済み例外を返します。

Cache のオブジェクト ID の詳細は、Cacheドキュメント：[\[Cache開発ガイド\]-\[Cacheオブジェクトの使用法\]-\[Cacheオブジェクト・モデル\]](#) を参照してください。

## 簡単なプロパティ値の検索と変更

Java プロジェクションでは、Cache オブジェクトの簡単なプロパティ値を検索および変更するための標準の“Java 形式”のアクセサ・メソッド、`getPropertyName` メソッドと `setPropertyName` メソッドを提供します。

例えば、以下のコードは、`contact` で参照される **Contact** インスタンスの Name プロパティの値を検索します。

```
String name = contact.getName();
```

以下のコードは、`contact` インスタンスの Name プロパティの値を変更、つまり“設定”します。**contact** の `save` メソッドが実行されるまで、変更内容はデータベースには保存されません。

```
contact.setName("New Name");  
contact.save();
```

### Note:

`setPropertyName` メソッドと `getPropertyName` のメソッドの両方が、

**CacheException** タイプのチェック済み例外を返します。

コレクションのプロパティ値を検索および変更する場合は、多少異なるテクニックが必要となります。これについては、このチュートリアルの後半で説明します。

## Caché コレクション

Caché クラスは、主に 2 種類の異なるコレクションを使用します。対応する Java タイプのプロジェクトのプロパティは、そのプロパティの JAVATYPE パラメータにより制御できます。以下のテーブルは、Caché コレクション・タイプと、それに対して推奨される Java プロジェクト・タイプを示しています。

コレクション・タイプ	詳細	推奨されるプロジェクト・タイプ
配列 <ul style="list-style-type: none"> <li>• <a href="#">%Library.ArrayOfDataTypes</a></li> <li>• <a href="#">%Library.ArrayOfObjects</a></li> </ul>	<ul style="list-style-type: none"> <li>• 整列していない一連のキー値マッピングを含みます。Java 配列よりも、Java ハッシュ・マップにより類似しています。</li> <li>• データ型クラスのインスタンス、相互依存関係のないオブジェクト、リレーションシップ・オブジェクト（相互依存関係のあるオブジェクト）を含めることができます。</li> </ul>	<a href="#">java.util.Map</a>
リスト <ul style="list-style-type: none"> <li>• <a href="#">%Library.ListOfDataTypes</a></li> <li>• <a href="#">%Library.ListOfObjects</a></li> </ul>	<ul style="list-style-type: none"> <li>• 整列したデータのリストを含みます。</li> <li>• データ型クラスのインスタンス、相互依存関係のないオブジェクト、リレーションシップ・オブジェクト（相互依存関係のあるオブジェクト）を含めることができます。</li> </ul>	<a href="#">java.util.List</a>

Caché プロパティの Java プロジェクト・タイプは、プロパティの JAVATYPE パラメータを使用して設定します。パラメータは、Caché スタジオのインスペクタを使用するか、または直接プロパティ宣言の中で、推奨されるプロジェクト・タイプに対して設定します。この例では、

**PhoneNumbers** のプロジェクト・タイプが、[java.util.Map](#) に対して設定されています。

```
Relationship PhoneNumbers As JavaTutorial.PhoneNumber
```

```
(JAVATYPE = "java.util.Map")
```

```
[ Cardinality = children, Inverse = Contact ];
```

### Note:

Caché コレクションの詳細は、Caché ドキュメント: [\[Caché 開発ガイド\]-\[Caché オブジェクトの使用法\]-\[プロパティ\]-\[コレクション・プロパティ\]](#) を参照してください。

## リストの例

Cache クラス **Contact** には、データベースの **Contact** インスタンスすべてのオブジェクト ID を含むリストをアSEMBLする、**ListOfContacts** という名前のメソッドが含まれています。このメソッドは、Cache タイプ **%Library.ListOfDatatypes** のオブジェクトを返します。 **Contact** の Java プロジェクションにも、**ListOfContacts** メソッドが含まれています。

以下の Java クライアント・メソッドは、**java.util.List** インタフェースを使用して、**ListOfContacts** から返されるデータを操作します。このメソッドは、リストの繰り返し処理を行います。リストの各オブジェクト ID について、対応する **Contact** インスタンスをオープンし、その Name プロパティと ID プロパティの値を表示しています。

```
public class BindingExamples {
    public static void displayContacts(Database db) throws CacheException {
        List listOfContacts = (List) (Contact.ListOfContacts (db));
        Iterator iter = listOfContacts.iterator ();
        while(iter.hasNext()) {
            String id = (String) (iter.next());
            Contact contact = (Contact) (Contact._open(db, new Id(id)));
            System.out.println ("Name " + contact.getName () + " ID " + id);
        }
    }
}
```

### Note:

**Contact** の Java プロジェクションでの **ListOfContacts** の返りタイプは、**com.intersys.classes.ListOfDatatypes** です。このクラスは後方互換性のために残されていますが、使用はお勧めしません。プロパティの場合と異なり、メソッドの返りタイプについては、既定の Java プロジェクション・タイプをオーバーライドする JAVATYPE パラメータは存在しません。通常は、非推奨であることを示す警告を無視して、Java クライアント・コードでは推奨されるプロジェクト・タイプのインタフェースを使用する必要があります。この場合、Java コードでは **java.util.List** インタフェースを使用します。

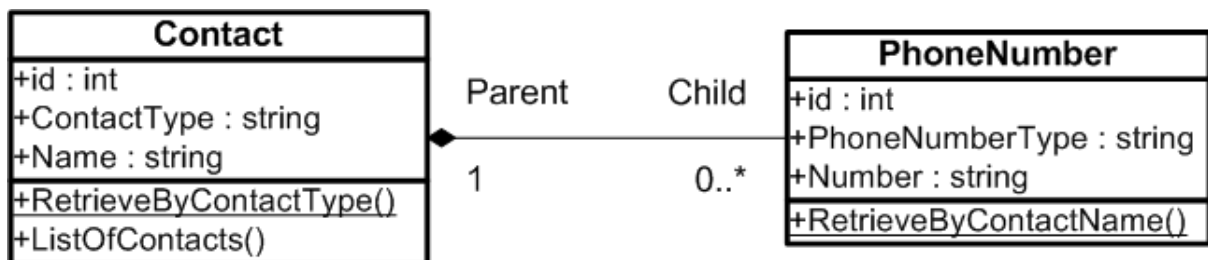
## リレーションシップ・タイプ

Cache では、オブジェクト間のリレーションシップをサポートします。Contact Management アプリケーションの場合、**Contact** オブジェクトと **PhoneNumber** オブジェクトによって、親子リレーションシップが形成されます。**Contact** オブジェクトが親の役割を受け持ち、**PhoneNumber** オブジェクトが子の役割を受け持ちます。これは、次のことを意味します。

- 各 **Contact** オブジェクトは、**PhoneNumber** オブジェクトをいくつでも含むことができる一方で、まったく含まなくてもかまいません。
- 各 **PhoneNumber** オブジェクトは、1 つの **Contact** オブジェクトにのみ含まれていなければなりません。

Cache では、一対多のリレーションシップもサポートします。一対多のリレーションシップは親子リレーションシップと似ていますが、“多”側を形成するタイプのオブジェクトが、“一”側を形成するタイプのオブジェクトから独立して存在できる点が異なります。つまり、多側のタイプのオブジェクトが一側のオブジェクトに含まれている必要はありません。また、複数の一側のオブジェクトに同時に含まれることも可能です。

以下の図は、**Contact** と **PhoneNumber** の間の親子リレーションシップを表しています。



### Note:

Cache のリレーションシップの詳細は、Cache ドキュメント: [\[Cache 開発ガイド\]-\[Cache オブジェクトの使用法\]-\[リレーションシップ\]](#) を参照してください。

## 子オブジェクトの表示

リレーションシップの親クラス(一側)内では、子(多側)はコレクションとして表現されます。例えば、**Contact**オブジェクトに、**PhoneNumber**オブジェクトのコレクションが含まれているとします。このコレクションは、タイプ **com.intersys.classes.RelationshipObject**としてJavaに投影されます。**Contact**のJavaプロジェクト内の**getPhoneNumbers**メソッドが、**com.intersys.classes.RelationshipObject**オブジェクトを返します。このクラスが、**java.util.Map**インタフェースを実装します。このインタフェースから呼び出されるメソッドを、子オブジェクトのコレクションの繰り返し処理に使用できます。

以下のJavaクライアント・メソッドは、**java.util.Map**インタフェースを使用して、特定の**Contact**に属している一連の**PhoneNumber**インスタンスの繰り返し処理を行います。また、各**PhoneNumber**インスタンスのNumberと PhoneNumberTypeの値を出力します。

```
public class BindingExamples {
    public static void displayNumbers(int id, Database db) throws CacheException {
        Contact contact=(Contact) (Contact._open(db, new Id(id)));
        Map phoneNumbers=(Map) contact.getPhoneNumbers();
        Iterator iter=phoneNumbers.keySet().iterator();
        while(iter.hasNext()) {
            PhoneNumber pn = (PhoneNumber) (phoneNumbers.get(iter.next()));
            System.out.println("Type: " +
                pn.getPhoneNumberType() + " Number: " + pn.getNumber());
        }
    }
}
```

## 子オブジェクトの追加

リレーションシップの子クラス(多側)内では、親(一側)は親タイプの簡単なプロパティとして表現されます。例えば、各 **PhoneNumber** オブジェクトに、親オブジェクトを表すタイプ **Contact** のプロパティが含まれているとします。このプロパティは、その Java プロジェクション・タイプとして Java に投影されます。**PhoneNumber** プロジェクションの **getContact** メソッドは、**Contact** プロジェクション・タイプのオブジェクトを返します。

以下の Java クライアント・メソッドは、**PhoneNumber** オブジェクトを生成し、その **setContact** メソッドを使用して親オブジェクトを設定します。

```
public class BindingExamples {
    public static void addNumberToContact(Database db, String number,
        String type, Contact contact) throws CacheException {
        PhoneNumber phoneNumber = new PhoneNumber(db);
        phoneNumber.setNumber(number);
        phoneNumber.setPhoneNumberType(type);
        phoneNumber.setContact(contact);
        phoneNumber.save();
    }
}
```

## 子オブジェクトの表示

Java クライアントは、プロジェクションを通して Caché データにアクセスするだけでなく、データベースを検索することができます。Caché では、いくつかの異なる方法で、Java クライアント・アプリケーション内からクエリを実行できます。

Caché クエリ・タイプ	特徴	Java クライアント・アクセス
クラス・クエリ	Caché クラス定義内での宣言によって、あらかじめ定義されている SQL クエリ。メソッド宣言に似ていますが、SQL コードを含む点が異なります。	Java クライアントは、 <b>CacheQuery</b> オブジェクトの <b>execute</b> メソッドを使用します。このメソッドは、 <b>java.sql.ResultSet</b> オブジェクトをクライアントに返します。 このクラスの Java プロジェクションには、 <b>query_QueryName</b> という名前のメソッドが自動的に含まれ、このメソッドは、適切に初期化された <b>CacheQuery</b> オブジェクトの検索に使用できます。 また、Java クライアントは、コンストラクタを使用して新しい <b>CacheQuery</b> オブジェクトを生成し、クエリ名を使用してそのオブジェクトを初期化できます。
ダイナミック SQL	実行時に Java クライアントにより生成される SQL コード。	<b>Database openByQuery</b> メソッドを使用します。クライアントは、該当するSQL文字列をメソッドに渡します。メソッドは、オープンしている一連のオブジェクトを含む <b>java.util.Iterator</b> オブジェクトを返します。 また、 <b>CacheQuery</b> 経由で SQL 文字列を Caché に渡します。Caché は、標準の <b>java.sql.ResultSet</b> オブジェクトをクライアントに返します。
埋め込み SQL	Caché メソッド内部に格納されている SQL コード。	Java クライアントは、直接 SQL にアクセスできません。しかし、プロジェクションを使用すると、Java クライアントは、SQL を含むメソッドにアクセスできる場合があります。

### Note:

クラス・クエリの詳細は、Cachéドキュメント: [\[Caché開発ガイド\]-\[Cachéオブジェクトの使用法\]-\[クラス・クエリ\]](#) を参照してください。ダイナミックSQLの詳細は、[\[Caché開発ガイド\]-\[Caché SQLの使用法\]-\[ダイナミックSQL\]](#) を参照してください。埋め込みSQLの詳細は、[\[Caché開発ガイド\]-\[Caché SQLの使用法\]-\[埋め込みSQL\]](#) を参照してください。Javaクライアント・アプリケーション内からクエリを実行する方法の詳細は、[\[Caché言語バインディング\]-\[CachéでのJavaの使用法\]-\[Java バインディングの使用法\]-\[クエリの使用法\]](#) のセクションで“クラス・クエリ”と“ダイナミック・クエリ”を参照してください。

## クラス・クエリの実行

Cache は、Cache クラスの各クラス・クエリのメソッドを自動的に生成して、Java プロジェクションに追加します。Cache クエリ名の前に "query\_" が付いたものが、メソッド名になります。Java メソッドは、タイプ `com.intersys.objects.CacheQuery` のオブジェクトを返します。

例えば、Cache `Contact` クラスに、`RetrieveByContactType` という名前のクラス・クエリが含まれているとします。Cache では、以下のシグニチャを持つメソッドを Java プロジェクションに追加します。

```
public static CacheQuery query_RetrieveByContactType (Database db)
throws CacheException
```

`CacheQuery execute` メソッドを使用して、クエリを実行します。以下の Java メソッドは、Java プロジェクションを使用して `RetrieveByContactType` クエリを実行します。このメソッドでは、`type` を入力パラメータ値としてクエリに渡します。

```
public class BindingExamples {
    public static void displayContactsByType(Database db, String type)
    throws CacheException, SQLException{
        CacheQuery query = Contact.query_RetrieveByContactType(db);
        java.sql.ResultSet rs=query.execute(type);
        System.out.println(type + " Contacts");
        while (rs.next()){
            System.out.println(rs.getString(1));
        }
    }
}
```

### Note:

`com.intersys.classes` パッケージには `ResultSet` クラスが含まれていますが、このクラスは使用しないでください。

`com.intersys.classes` がソース・ファイルにインポートされた場合に正確に認識されるように、`java.sql.ResultSet` の完全修飾名を使用する必要があります。

## ダイナミック・クエリの実行: openByQuery

**Database** インタフェースは、ダイナミック・クエリを実行する **openByQuery** メソッドを提供します。Java クライアントは、SQL を含む文字列を、入力パラメータの配列と一緒に **openByQuery** に渡します。このメソッドは、クエリ結果を表すオブジェクトを含む [java.util.Iterator](#) オブジェクトを返します。これにより、結果セットではなく標準オブジェクトを使用して、クエリ結果を処理できます。メソッドに渡す SQL 文字列に関して、以下の規則があります。

- 最初に返す列には、テーブルの %ID 列の完全修飾名を含めます (例えば、`JavaTutorial.PhoneNumber.%ID`)。
- ORDER BY 節では、順序を指定する数値ではなく、列名を使用する必要があります。

以下の Java クライアント・メソッドは、**openByQuery** を使用して、特定の **Contact** に属し、かつ **PhoneNumberType** に対する値が **type** である、**PhoneNumber** オブジェクトすべてを検索するクエリを実行します。このメソッドは、返された一連の **PhoneNumber** オブジェクトを繰り返し処理して、各オブジェクトの **PhoneNumberType** と **Number** の値を表示します。

```
public class BindingExamples {
    public static void displayPhoneNumbersByTypeOBQ(Database db, String id,
        String type) throws CacheException {
        Object[] args = {id, type};
        String sql = "SELECT JavaTutorial.PhoneNumber.%ID FROM " +
            "JavaTutorial.PhoneNumber WHERE Contact = ? AND PhoneNumberType = ?";
        Iterator iter = db.openByQuery(sql, args);
        while (iter.hasNext()) {
            PhoneNumber pn = (PhoneNumber) iter.next();
            System.out.println("Type: " + pn.getPhoneNumberType() +
                " Number: " + pn.getNumber());
        }
    }
}
```

## ダイナミック・クエリの実行: CacheQuery

Java クライアントは、**CacheQuery** クラスを介してデータベースに SQL 文字列を渡すことにより、Cache に対してダイナミック・クエリを実行することもできます。**CacheQuery execute** メソッドは、**java.sql.ResultSet** オブジェクトを返します。

以下のメソッドは、値が **id** の **Contact** プロパティと、値が **type** の **PhoneNumberType** プロパティを持つ、すべての **PhoneNumber** インスタンスの **Number** プロパティの値を選択するクエリを実行します。次に、このメソッドは、**ResultSet** を繰り返し処理して、それぞれの **Number** 値を表示します。

```
public class BindingExamples {
    public static void displayPhoneNumbersByType(Database db,
        String id, String type) throws CacheException, SQLException {
        Id ID= new Id(id);
        String SQL = "SELECT Number FROM JavaTutorial.PhoneNumber" +
            " WHERE Contact='"+id+"' AND PhoneNumberType='"+ type +"'";
        CacheQuery query = new CacheQuery(db, SQL);
        java.sql.ResultSet rs= query.execute();
        while (rs.next()){
            System.out.println(rs.getString(1));
        }
    }
}
```

### Note:

Cache で **PhoneNumber** を表す SQL テーブルには、**Contact** 列が含まれています。この列には、**Contact** オブジェクトの ID 値が含まれます。これらの値は、各 **PhoneNumber** 行を **Contact** 行と関連付ける外部キーです。これについては、このチュートリアルの第 II 章で説明します。

## 演習

演習 1 : 以下の要件を満たす 1 つのメソッドを *BindingExercises.java* に追加します。

- `public static void createContact(Database db, String name, String type) throws CacheException` のシグニチャを持ちます。
- *Name* の値を `name` に、*ContactType* の値を `type` に設定する、新しい **Contact** インスタンスを作成します。
- **Contact** インスタンスをデータベースに保存します。
- 永続性が保持されている場合は、新しい **Contact** インスタンスの *id* プロパティ値が含まれた成功を示すメッセージを表示します。

作成したメソッドをテストしてください。無効な *ContactType* 値、つまり **Business** または **Personal** 以外の値をメソッドに渡してみます。

演習 2 : 以下の要件を満たす 1 つのメソッドを *BindingExercises.java* に追加してください。

- `public static void displayPhoneNumbers(Database db, Contact contact) throws CacheException, SQLException` のシグニチャを持ちます。
- `contact` に含まれる各 **PhoneNumber** インスタンスについて、*PhoneNumberType* の値と *Number* の値を表示します。
- **PhoneNumber** の事前定義クエリ **RetrieveByContactName** を使用して、**PhoneNumber** インスタンスのリストを作成します。

作成したメソッドをテストしてください。

演習 3 : 以下の要件を満たす 1 つのメソッドを *BindingExercises.java* に追加します。

- `public static void removePhoneNumbers(Database db, Contact contact, String type) throws CacheException` のシグニチャを持ちます。
- `type` という *PhoneNumberType* 値を持つすべての **PhoneNumber** インスタンスを、`contact` から削除します。
- 変更をデータベースに保存します。

作成したメソッドをテストしてください。

### Note:

演習の解答と演習のスタブ・コードはどちらも、*USER* ネームスペースへの接続を構築する接続 URL を使用しています。別のネームスペースに Contact Management アプリケーションの Cache クラスをインストールした場合は、解答とスタブ・コードの両方の URL を、そのネームスペースを反映した URL に変更してください。つまり、URL を `jdbc:Cache://localhost:1972/<namespace name>` に変更します。

## 要約

Java および J2EE と共に使用する Cache クイックスタート・チュートリアルの第 I 章では、以下の項目について学習しました。

- Cache オブジェクトのオブジェクト指向機能と、SQL テーブルのリレーショナル機能との間でのマッピング
- Java バインディング機能のアーキテクチャ
- Cache クラスと、その Java プロジェクションとの間のリレーションシップ
- Java アプリケーションでの Cache クラスの Java プロジェクションの役割
- Cache クラスの Java プロジェクションの生成方法
- Java プロジェクションとオブジェクト指向手法を使用して、コレクションとリレーションシップを含む、異なるタイプの Cache データを操作する方法
- Java プロジェクションを使用して、Cache データベースでクエリとストアド・プロシージャを呼び出す方法

## 第 II 章: Cache と JDBC の概要

このチュートリアルの第 II 章では、Cache JDBC データベース・ドライバと SQL を使用して、Java アプリケーションを Cache に接続する方法について説明します。この章の学習を終えると、以下のことを実行できるようになります。

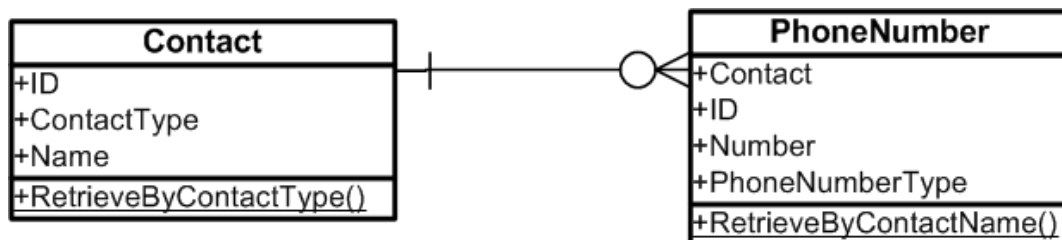
- `java.sql.DriverManager` および `com.intersys.objects.CacheDatabase` を使用して、Java アプリケーションと Cache の間のデータベース接続を構築する
- Java アプリケーション内から Cache 上で SQL の SELECT コマンドと INSERT コマンドを実行する
- Java アプリケーション内から、Cache で定義したストアド・プロシージャを実行する
- Cache の暗黙結合構文を使用して、Java アプリケーション内から Cache 上で結合を含むクエリを実行する

### Note:

第 II 章の内容は、Cache の基礎知識だけでなく、基本的な Java プログラミング・スキルを持っていることを前提としています。Cache の詳細は、Cache ドキュメント: [\[Cache チュートリアル\]](#)-[\[Cache クイックスタート・チュートリアル\]](#) を参照してください。Java プログラミングの詳細は、Sun Microsystems の "[The Java Tutorial](#)" を参照してください。特に "The JDBC Trail" を参照してください。

## Contact Management アプリケーション

このチュートリアルには、Java コードをサンプルの Cache アプリケーションに接続する例と演習が含まれています。この Contact Management アプリケーションは、ユーザが連絡先とその電話番号のリストをデータベースに保持できるようにするアプリケーションです。リレーショナルの視点から見ると、これは 2 つの Cache テーブル、**Contact** と **PhoneNumber** で構成されています。



上の図は、テーブルの属性とそのストアド・プロシージャを示しています。また、この図では、2 つのテーブル間のリレーションシップも表しています。エンティティによって、親子リレーションシップが形成されます。各 **PhoneNumber** は、ただ 1 つの **Contact** だけに関連付けられます。各 **Contact** は、多数の **PhoneNumbers** に関連付けられることができます。

**Contact** は、以下の要素で構成されています。

- ID: Cache オブジェクト ID。 **Contact** テーブルの主キーになります。
- ContactType: 連絡先のタイプ。この値には、“Business”と “Personal”だけを指定できます。
- Name: 連絡先の名前。この値には、任意の文字列を指定できます。
- **RetrieveByContactType**: スタド・プロシージャ。引数として ContactType 値を受け取ります。一致するすべての **Contact** 行の Name 値を返します。
- **ListOfContacts**: スタド・プロシージャ。すべての **Contact** 行の ID 値を検索します。

**PhoneNumber** は、以下の要素で構成されています。

- Contact: この列が外部キーになります。各 **PhoneNumber** 行に対して関連付けられている **Contact** 行の ID を含みます。
- ID: Cache オブジェクト ID。各インスタンスは一意的の値を持ちます。スタジオ開発環境では、このプロパティはクラス定義に表示されません。
- PhoneNumberType: 電話番号のタイプ。この値には、“Business”、“Home”、“Mobile”、および “Fax”を指定できます。
- Number: 電話番号。この値には、任意の文字列を指定できます。
- **RetrieveByContactName**: 事前定義のクエリ。引数として **Contact** Name 値を受け取ります。 **Contact** に関連付けられているすべての **PhoneNumber** 行の ID 値を返します。

**Note:**

演習を実行するには、Cache Contact Management アプリケーションをインストールする必要があります。このアプリケーションのインストール方法と、参考として提供されている Java ソース・ファイルの詳細は、[AppendixB: 例と演習のファイル] を参照してください。

## Caché と JDBC

Caché は、高性能なレベル 4 の JDBC データベース・ドライバを提供します。このドライバは Pure Java であり、Cache 特有のバイナリ・コードや JDBC-ODBC ブリッジは必要ありません。Cache JDBC ドライバは、以下の機能を含む JDBC2.0 API をサポートしています。

- スクロール可能な ResultSet
- 更新可能な ResultSet
- SQL コマンドのバッチ実行
- SQL3 データ型

リレーショナル・モデルを使用して Cache データにアクセスするときに、Cache JDBC ドライバと SQL を使用できます。Java バインディング機能でも JDBC ドライバを使用します。リレーショナル・データベース・アクセスとオブジェクト指向データベース・アクセスを組み合わせると、アプリケーションに最大限の柔軟性を持たせることができます。

Cache JDBC ドライバは、CacheDB.jar に含まれています。標準の Cache インストールを行った場合、このファイルは <cachsys>%dev%java%lib にあります。

### Note:

Cache JDBC ドライバを使用する際と、チュートリアルの演習を実行する際のシステム要件については、[AppendixA: 第 I 章と第 II 章でのシステム要件] を参照してください。

## DriverManager を使用した接続

標準の JDBC 方法を使用して、**java.sql.Connection** オブジェクトで表される、Cache データベースへの接続を構築できます。まず、**java.lang.Class** の **forName** メソッドを使用して Cache JDBC ドライバをロードし、次に **java.sql.DriverManager** を使用して **Connection** オブジェクトを生成します。**Connection** クラスには、トランザクションを管理するメソッドと、クエリと更新を実行する **Statement** オブジェクトを作成するメソッドが含まれています。

以下のメソッドは、この標準の方法を使用して、Cache への接続を構築します。

```
public class JDBCExamples {
    public static Connection createConnection()
    throws SQLException, ClassNotFoundException {
        String url="jdbc:Cache://localhost:1972/USER";
        String username="_SYSTEM";
        String password="SYS";
        Class.forName ("com.intersys.jdbc.CacheDriver");
        Connection conn =
            DriverManager.getConnection(url, username, password);
        return conn;
    }
}
```

### Note:

この章の JDBC 例の Java ファイルでは、USER ネームスペースに接続するための接続 URL を使用しています。Contact Management アプリケーションの Cache クラスを別のネームスペースにインストールした場合は、コード例の接続 URL をインストール先に応じて変更してください。つまり、URL を `jdbc:Cache://localhost:1972/<namespace name>` に変更します。

## CacheDatabase を使用した接続

標準の JDBC 方法を使用した Cache への接続の他に、

**com.intersys.objects.CacheDatabase** を使用しても接続を構築できます。

**getDatabase** メソッドは、**com.intersys.objects.Database** オブジェクトを返します。このクラスは、**java.sql.Connection** に対応します。その **createStatement** メソッドが返す **java.sql.Statement** オブジェクトを使用すると、データベースのクエリと更新を実行できます。

また、**Database** には、トランザクションを管理するメソッドも含まれています。

以下のメソッドは、**Database** オブジェクトを生成して返します。

```
public class JDBCExamples {
    public static Database getDatabase() throws CacheException {
        String url="jdbc:Cache://localhost:1972/USER";
        String username="_SYSTEM";
        String pwd="SYS";
        Database db =
            CacheDatabase.getDatabase(url, username, pwd);
        return db;
    }
}
```

**getDatabase** に渡されるデータベース URL は、Cache に接続する際に、標準の JDBC 方法を使用して **DriverManager** に渡される URL と同じです。

### Note:

この章の JDBC 例の Java ファイルでは、USER ネームスペースに接続するための接続 URL を使用しています。Contact Management アプリケーションの Cache クラスを別のネームスペースにインストールした場合は、コード例の接続 URL をインストール先に応じて変更してください。つまり、URL を `jdbc:Cache://localhost:1972/<namespace name>` に変更します。

**CacheDatabase** と **Database** の詳細は、`<cachesys>%dev%java%doc` に保存されている API ドキュメントを参照してください。

## クエリの実行

Cache への接続(`java.sql.Connection` または `com.intersys.objects.Database` のいずれかで表される)を構築すると、データベースに対して SQL 文を実行できます。以下のメソッドは、SQL の **SELECT** 操作を実行します。このクエリでは、データベース内のすべての **Contact** インスタンスの Name 値と ContactType 値を検索します。そして、**ResultSet** を繰り返し処理して、Name と ContactType の値をそれぞれ表示します。このメソッドでは、`java.sql.Connection` を使用して、クエリの実行に使用する **Statement** オブジェクトを生成しています。

```
public class JDBCExamples {
    public static void printContactNames()
        throws SQLException, ClassNotFoundException {
        Connection conn = JDBCExamples.createConnection();
        Statement stmt = conn.createStatement();
        String query =
            "SELECT Name, ContactType FROM JavaTutorial.Contact";
        ResultSet rs=stmt.executeQuery(query);
        while (rs.next()) {
            String name = rs.getString(1);
            String type = rs.getString(2);
            System.out.println("Name: " + name + " Type: " + type);
        }
        rs.close();
    }
}
```

## 挿入の実行

Cache への接続(`java.sql.Connection` または `com.intersys.objects.Database` のいずれかで表される)を構築すると、データベースを更新できます。以下のメソッドは、SQL の **INSERT** 操作をデータベースに対して実行します。 **PhoneNumber** テーブルに新しい行を追加します。

```
public class JDBCExamples {
    public static void insertPhoneNumber(String contactId,
        String number, String type) throws SQLException, ClassNotFoundException {
        Connection conn = JDBCExamples.createConnection();
        String sql =
            "INSERT INTO JavaTutorial.PhoneNumber (Contact, Number, PhoneNumberType)" +
            "VALUES (?, ?, ?)";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, contactId);
        pstmt.setString(2, number);
        pstmt.setString(3, type);
        pstmt.executeUpdate();
    }
}
```

このメソッドでは、`java.sql.PreparedStatement` オブジェクトで表される、作成済み文を使用しています。これは、`Connection` クラスの `prepareStatement` メソッドを使用して生成されます。Cache `Database` クラスには、`java.sql.PreparedStatement` オブジェクトを返す `prepareStatement` も含まれています。

## ストアド・プロシージャの実行

Cache では、ストアド・プロシージャを実行することもできます。以下のメソッドは、**Contact** に格納されている **RetrieveByContactType** プロシージャを実行します。このプロシージャは、ContactType プロパティの値を表す 1 つのパラメータを受け取ります。そして、受け取ったパラメータと一致する ContactType の値を持つ、すべての **Contact** インスタンスの Name 値を返します。このメソッドは、プロシージャにより返された **ResultSet** を繰り返し処理して、その値を表示します。

```
public class JDBCExamples {
    public static void printContactByType(String type)
        throws SQLException, ClassNotFoundException {
        Connection conn = JDBCExamples.createConnection();
        CallableStatement cs = conn.prepareCall
            (" {call JavaTutorial.Contact_RetrieveByContactType(?)} ");
        cs.setString(1, type);
        ResultSet rs = cs.executeQuery();
        while (rs.next()) {
            System.out.println(rs.getString(1));
        }
        rs.close();
    }
}
```

このメソッドでは、**java.sql.Connection** `prepareCall` を使用して、ストアド・プロシージャの実行に必要な **java.sql.CallableStatement** オブジェクトを生成しています。また、**com.intersys.objects.Database** は、**java.sql.CallableStatement** オブジェクトの生成にも使用できる `prepareCall` メソッドも提供します。

### Note:

ストアド・プロシージャは、Cache クラス定義内のクラス・クエリとして定義することができます。ストアド・プロシージャとクラス・クエリの詳細は、Cache ドキュメント: [\[Cache開発ガイド\]-\[Cacheオブジェクトの使用法\]-\[クラス・クエリ\]](#) と [\[Cache開発ガイド\]-\[CacheSQLの使用法\]-\[ストアド・プロシージャ\]](#) を参照してください。

## 暗黙結合

Caché SQL は、暗黙結合構文をサポートします。これは、特定のタイプの結合について SQL を簡略化するものであり、“—>”演算子で表されます。テーブルに参照列、つまり参照されるテーブルの ID 値を含む列がある場合はいつでも、—>演算子を使用できます。—>演算子は、参照されるテーブル内のプロパティの参照に使用できます。

例えば、**PhoneNumber** テーブルに、**Contact** ID 値を含む Contact 列があるとします。

**PhoneNumber** テーブルで、**Contact** テーブルの行を参照する SQL 処理を行うときに、—>演算子を使用できます。以下のクエリは、PhoneNumberType 値が “Fax” であり、かつ、Name 値が “Public, John Q.” の **Contact** テーブル行に対応する Contact 値を持つ、**PhoneNumber** 行すべての Number プロパティ値を返します。

```
SELECT Number FROM JavaTutorial.PhoneNumber
WHERE PhoneNumberType='Fax' AND
Contact->Name='Public, John Q.'
```

上記の暗黙結合クエリは、以下の標準結合と同等です。

```
SELECT Number FROM JavaTutorial.PhoneNumber, JavaTutorial.Contact
WHERE PhoneNumberType='Fax' AND
JavaTutorial.PhoneNumber.Contact = JavaTutorial.Contact.ID AND
JavaTutorial.Contact.Name='Public, John Q.'
```

### Note:

Cachéの暗黙結合構文の詳細は、Cachéドキュメント: [\[Caché開発ガイド\]-\[CachéSQLの使用法\]-\[特別な機能\]-\[暗黙結合\]](#) を参照してください。

## 暗黙結合の使用

以下のメソッドは、Cache の暗黙結合構文を使用してクエリを定義し、そのクエリをデータベースに対して実行します。クエリは、以下の条件を満たす、すべての **PhoneNumber** 行の Number 値を選択します。

- PhoneNumberType の値が **type** です。
- Contact 値で参照される **Contact** テーブルの行で、Name の値が **name** です。

```
public class JDBCExamples {
    public static void printPhoneNumbersByNameAndType
        (String name, String type) throws SQLException,
        ClassNotFoundException {
        Connection conn = JDBCExamples.createConnection();
        String sql =
            "SELECT Number FROM JavaTutorial.PhoneNumber "+
            "WHERE PhoneNumberType=? AND Contact->Name=?";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, type);
        pstmt.setString(2, name);
        ResultSet rs=pstmt.executeQuery();
        System.out.println("Name: " + name + " Type: " + type);
        while (rs.next()) {
            System.out.println(rs.getString(1));
        }
        rs.close();
    }
}
```

## 演習

演習 1 : 以下の要件を満たす 1 つのメソッドを *JDBCExercises.java* に追加します。

- `public static void insertContact(Database db, String name, String type) throws CacheException, SQLException` のシグニチャを持ちます。
- `name` という *Name* 値と、`type` という *ContactType* 値を持つ新しい行を、[JavaTutorial.Contact](#) に挿入します。
- 更新された行数が含まれている成功を示すメッセージを表示します。

作成したメソッドをテストしてください。

演習 2 : 以下の要件を満たす 1 つのメソッドを *JDBCExercises.java* に追加します。

- `public static void displayPhoneNumbersByTypeAndName(Database db, String name, String type) throws CacheException, SQLException` のシグニチャを持ちます。
- *PhoneNumberType* 値が `type` で、関連する [JavaTutorial.Contact](#) 行の *Name* 値が `name` である、[JavaTutorial.PhoneNumber](#) 行の *Number* 値をすべて表示します。
- Cache の暗黙結合構文を使用します。

作成したメソッドをテストしてください。

演習 3 : 以下の要件を満たす 1 つのメソッドを *JDBCExercises.java* に追加します。

- `public static void displayPhoneNumberIdsByName(Database db, String name) throws CacheException, SQLException` のシグニチャを持ちます。
- `name` という *Name* 値を持つ、[JavaTutorial.Contact](#) 行に関連付けられている [JavaTutorial.PhoneNumber](#) 行の *ID* 値をすべて表示します。
- [JavaTutorial.PhoneNumber](#) と共に格納されているストアド・プロシージャ [RetrieveByContactName](#) を使用します。

作成したメソッドをテストしてください。

### Note:

演習の解答と演習のスタブ・コードはどちらも、*USER* ネームスペースへの接続を構築する接続 URL を使用しています。別のネームスペースに Contact Management アプリケーションの Cache クラスをインストールした場合は、解答とスタブ・コードの両方の URL を、そのネームスペースを反映した URL に変更してください。つまり、URL を `jdbc:Cache://localhost:1972/<namespace name>` に変更します。

## 要約

Java および J2EE と共に使用する Cache クイックスタート・チュートリアルの第 II 章では、以下の項目について学習しました。

- **com.intersys.objects.CacheDatabase** と **java.sql.DriverManager** を使用して、Cache JDBC ドライバにより Java アプリケーションを Cache に接続する方法
- Cache データベースで SQL クエリを実行する方法
- Cache データベースでストアド・プロシージャを実行する方法
- Cache の暗黙結合構文を使用して、結合を含むクエリの SQL を簡略化する方法

Appendix A:

## 第 I 章と第 II 章でのシステム要件

Java バインディング機能や Cache JDBC ドライバを使用して、チュートリアルの演習を実行するには、Java アプリケーションが以下の要件を満たしている必要があります。

- J2SDK バージョン 1.4 以降を使用していること。
- `CacheDB.jar` がアプリケーションのクラスパスにあること。`CacheDB.jar` が `< cachesys >¥dev¥java¥lib¥JDK14` にあること。

これらの要件の詳細は、Cacheドキュメント: [\[Cache言語バインディング\]](#)-[\[CacheでのJavaの使用法\]](#)-[\[Javaクライアント要件\]](#) のセクションの説明を参照してください。

## 例と演習のファイル

Caché をインストールすると、サンプル・コードと演習問題の実行に必要なすべてのファイルが、インストール先ディレクトリに保存されます。

<cachsys>%dev%tutorials ディレクトリには、以下の 2 つの XML ファイルが含まれています。

1. *JAVTutorial.xml* — このファイルには、Contact Management アプリケーション用の作成済み Caché クラスが含まれています。このバージョンの Contact Management アプリケーションは、SAMPLES ネームスペースにインストールされます。これは、チュートリアルの手順を実行して、クラスを作成することを望まないユーザのために用意されています。
2. *JAVTutorial.xml* — このファイルには、Contact Management アプリケーション用の未完成の Caché クラスが含まれています。このバージョンは、チュートリアルの手順を実行して、クラスを作成することを望むユーザのために用意されています。チュートリアルを開始する前に、このバージョンを USER ネームスペース、または任意の作業用ネームスペースにインストールしてください。

*JAVTutorial.xml* または *JAVTutorialB.xml* を USER ネームスペース、またはその他のネームスペースにインストールするには、以下の手順を実行してください。

1. **[スタジオ]** を起動します。
2. **[ファイル]** メニューの **[ネームスペース変更]** をクリックし、アプリケーションをロードするネームスペースに接続します。
3. **[ツール]** メニューの **[ローカルからインポート]** をクリックし、**[XML]** を選択します。  
<cachsys>%dev%tutorials を検索します。
4. [*JAVTutorial.xml*] ファイル (または [*JAVTutorialB.xml*] ファイル) を選択し、**[開く]** をクリックします。

<cachsys>%dev%tutorials%javaandj2ee ディレクトリには、演習問題とデモンストレーション・コードの実行に必要な、以下のファイルとディレクトリも含まれています。

1. *JavaTutorial* — Contact Management クラス **Contact** と **PhoneNumber** の Java プロジェクション用の、Java ソース・ファイル *Contact.java* と *PhoneNumber.java* が含まれているディレクトリ
2. *BindingExamples.java* — このチュートリアルの Java バインディングの章に記載されているコード例の Java ソース・ファイル
3. *BindingExercisesSolutions.java* — このチュートリアルの Java バインディングの章の演習の解答である Java ソース・ファイル
4. *BindingExercises.java* — このチュートリアルの Java バインディングの章の演習で実行される、クラス定義とスタブ・メソッドが含まれている Java ソース・ファイル

5. *JDBCExamples.java* — このチュートリアルの *JDBC* の章に記載されているコード例の Java ソース・ファイル
6. *JDBCExerciseSolutions.java* — このチュートリアルの *JDBC* の章の演習の解答である Java ソース・ファイル
7. *JDBCExercises.java* — このチュートリアルの *JDBC* の章の演習で実行される、クラス定義とスタブ・メソッドが含まれている Java ソース・ファイル

**Note:**

Windows で標準インストールしている場合、< cachesys > は *C:¥Program Files¥Cache* になります。標準の Unix または Linux の環境では、*cachesys* は */usr/cachesys* になります。

演習と例の Java ソース・ファイルをコンパイルするときは、まず最初に *Contact.java* と *PhoneNumber.java* をコンパイルしてから、その他のファイルをコンパイルする必要があります。生成されるクラス・ファイルは、*JavaTutorial* サブディレクトリにそのまま置いてください。

Java ファイルをコンパイルして実行するためには、クラス・パスに *CacheDB.jar* がなければなりません。*CacheDB.jar* は < cachesys >¥dev¥java¥lib¥JDK14 にあります。

Java プログラムのコンパイルと実行の詳細は、Sun Microsystems の "[The Java Tutorial](#)" を参照してください。