# Introducing FORM

**ChiMu Corporation**

1220 N. Fair Oaks Ave, #1314
Sunnyvale, CA 94089

Phone: 408 734-9068
Email: info@chimu.com
www.chimu.com

# *Table of Contents*

# 1 Introduction

FORM is a Java framework for Object-Relational mapping. Object-Relational mapping is the process of transforming information between an object model and a relational storage model. FORM provides sophisticated object-relational capabilities, has a scalable architecture, and supports building high-quality object models and relational models. FORM provides great flexibility in how object models are transformed into relational models, so the optimal approach for your specific needs can be chosen.

FORM is designed to support sophisticated applications. It has limited responsibilities[1] — querying, retrieving, and storing domain objects — but FORM provides thorough and scalable implementations of those responsibilities. FORM's capabilities include:

1. Database independence and database understanding beyond JDBC capabilities
2. Sophisticated query capabilities and an OQL interface
3. Great flexibility in transforming complicated object models into standard relational models
4. Object identity management and caching
5. Both dynamic and description-based (serializable) configuration
6. Very little overhead over the JDBC driver for data-retrieval and significantly faster overall performance due to object level caching

FORM makes building good Object-Oriented Information Systems easier: FORM provides the capabilities and supports the architecture a development team needs to be successful.

## *1.1 Overview*

This document provides an advanced overview of FORM capabilities, concepts, and terms. As an advanced overview, this document provides only very short explanations of terms: just enough to make FORM a bit more familiar. To get a more detailed understanding of FORM's capabilities see the "Learning FORM" document and the other documents listed in the supplemental reading section below.

When you have finished with this document you will be familiar with how FORM works with an application and with most of FORM's major concepts.

## *1.2 Supplemental Reading*

The following documents are supplemental to this Introduction to FORM document. They can either be read prior to this document or concurrently as topic areas come up.

**Learning FORM** is the main developer's documentation of the FORM product. It provides detailed descriptions of the concepts and mechanisms within FORM, and it provides many examples of using FORM.

**Foundations of Object-Relational Mapping** describes general concepts and approaches for object-relational mapping. Although more general than FORM, these concepts can be helpful for getting a broad picture of what FORM is trying to accomplish and some of its possible approaches.

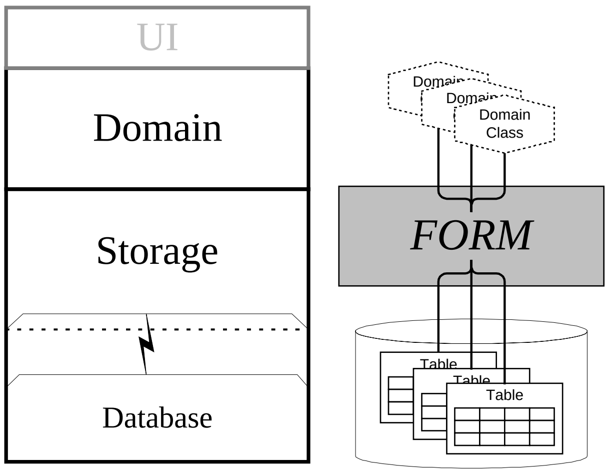
**A Good Architecture for Object-Oriented Information Systems** describes important architectural concepts for Object-Oriented Information Systems. These tutorial slides help explain how products like FORM fit into the overall architecture of an OOIS.

[1] FORM, the product distribution, also provides other architectural pieces that an application may want to use for other purposes (e.g. the Kernel frameworks). These other pieces will not be described in this document.

# 2 Application Architecture

This chapter introduces how FORM fits into an application's architecture. FORM does not force a particular architecture on an application, but it is designed to fit best into a particular architectural position.

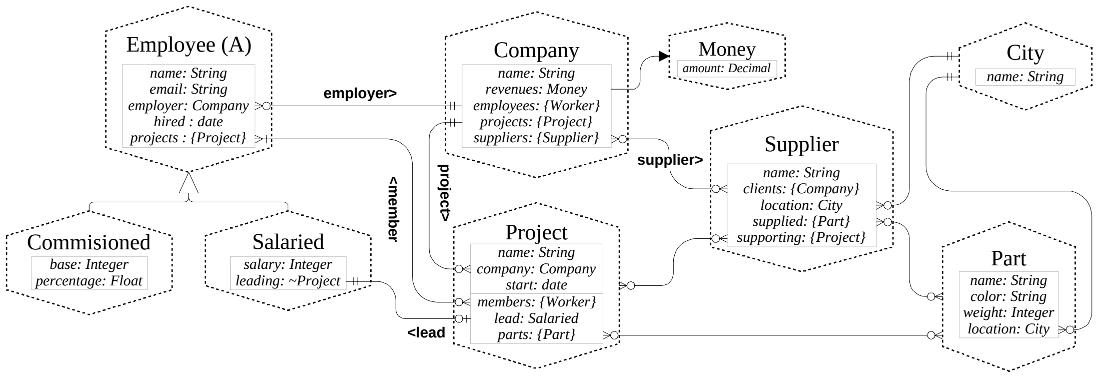
FORM works best in systems with at least two layers: a Domain Object Model Layer (Domain) and a Relational Information Storage Layer (Storage). Usually these layers are part of a system with three or more layers, with, for example, a UI Presentation Layer above the domain model (see [Architecture] for some more sophisticated system architectures).



FORM is responsible for querying, retrieving, and storing domain objects to the database. It provides an object-oriented storage model on top of the capabilities of a relational database. These responsibilities place FORM at the top of the Storage layer and FORM's primary client is the Domain Object Model.

## 2.1 *Domain Object Model*

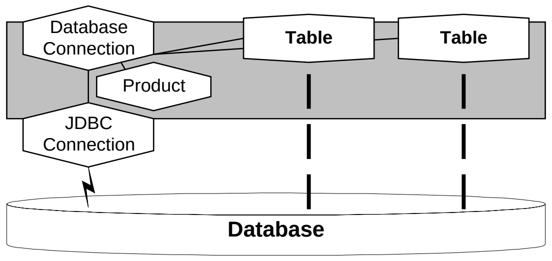
The business **Domain Model** is where you describe the information, operations, and rules to represent your business within a computer. The domain model is captured within Java classes and is completely under your control: it can have abstract classes, many-to-many associations, and complex embedded objects. FORM only needs to be able to "bind" with your domain model, which it can do by either directly extracting information from your Java classes or by you telling it how you want to control that binding.



## 2.2 *Relational Information Model*

An **Information Model** describes the information needed to record the state of your business model. This information model is captured within the Relational Database. Because the relational database is external to the Java application, FORM provides a proxy model of that database within the Java application. FORM builds that model in terms of Java data types and uses JDBC to communicate with the actual database.

## FORM Relational Interface

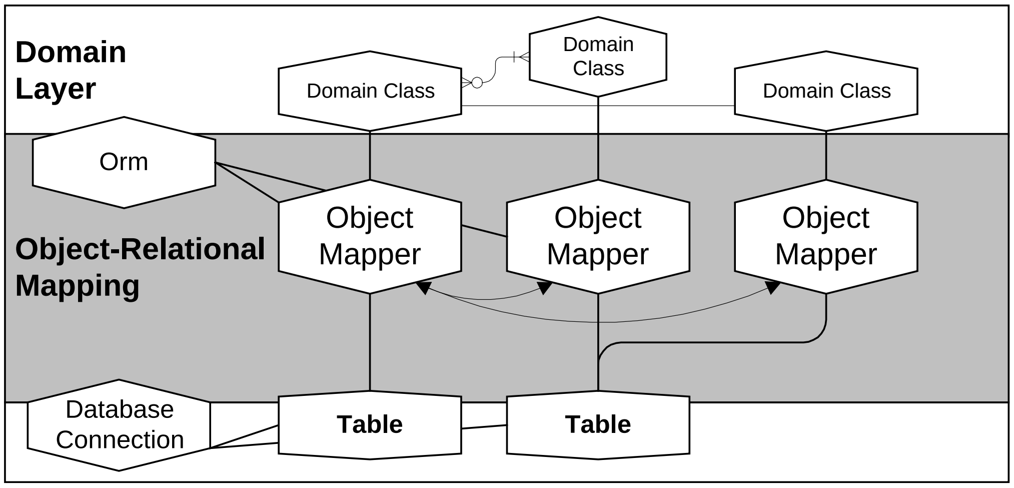


Although JDBC helps with database independence, its database model is not complete, fully database independent, database savvy, or object-oriented. FORM provides a relational interface layer on top of JDBC to provide an object-oriented, database independent, and database savvy interface.

## *2.3 Object-Relational Mapping*

FORM uses an Object-Relational Mapper (**Orm**) to translate between your application's Domain Object Model and its Relational Information Model. The Orm provides a transactional, object-oriented interface to the database that is used to query, retrieve objects, change their state, and save the changes back to the database.

An Orm uses **ObjectMappers** to convert, store, and retrieve objects of particular classes into the information in database tables. Multiple ObjectMappers collaborate to translate and store associated objects, and these associations can include 1-N, N-1, and N-M relationships.

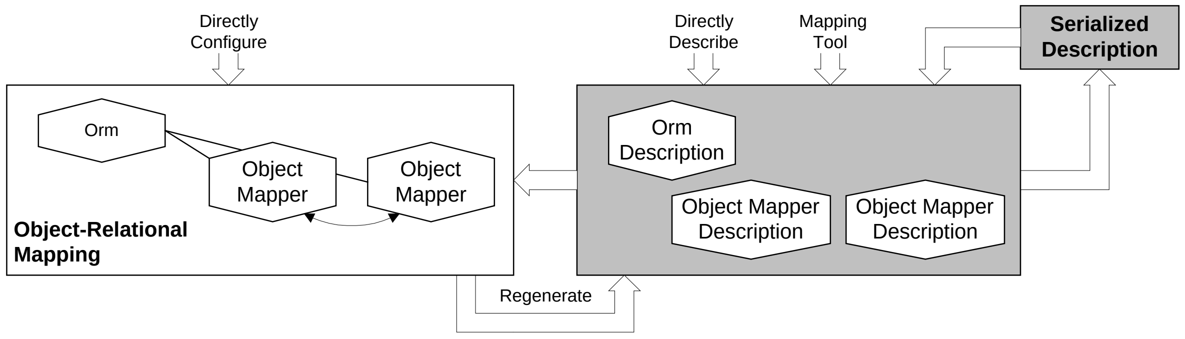


# 3 Capabilities and Design

This chapter describes some of the core capabilities and major design elements of FORM.

## 3.1 FORM Configuration and Descriptions

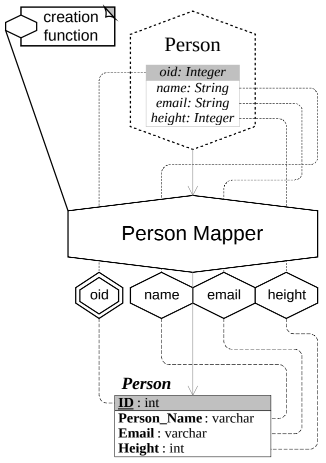
An application interacts with FORM in two different stages. First the application configures FORM with the information about its models and mapping, then the application runs FORM to retrieve and save objects. You can configure FORM's runtime objects directly or you can use **Descriptions,** which allow you to serialize your mapping description and deserialize it when your application is launched. By providing both types of configuration FORM allows both flexibility and control. Descriptions are also the form used by the UI Mapping Tool.



## 3.2 Running FORM

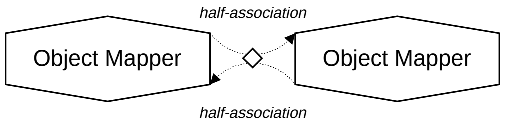
Running FORM is mostly invisible: FORM is designed to work underneath your domain model and not be visible to the layers above the domain layer. For example, FORM will automatically retrieve associated domain objects when you ask for a property. If you have a Person and want to retrieve her employer's name, you simply call "aPerson.employer().name()". FORM annotates your domain class so it can automatically convert **Stub** objects (which have not yet retrieved any state from the database) into **Replicate** objects (which have retrieved their complete state and association information).

## 3.3 Slots, Attributes, and Associations



FORM uses the concept of a **Slot** to describe an attribute of an object independently of whether it is within Java or the Relational Database. Slots handle the details of attribute encoding and decoding for an ObjectMapper when it stores and retrieves objects.

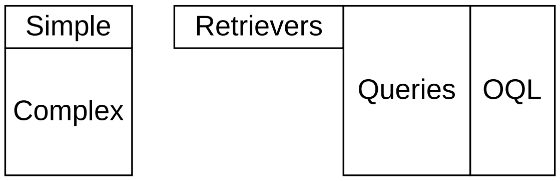
Slots are also used to describe associations between objects and so provide a unifying model for storage and querying. Because associations are bidirectional, each **AssociationSlot** describes the association from only one of the two linked-object's point of view. Together the two AssociationSlots and the two ObjectMapper's know of the full relationship.



## 3.4 Queries

FORM provides full query capabilities that allow you to easily evaluate both simple and sophisticated queries on your database domain model. FORM's query model is an object-extended relational model, which allows you to benefit from the simplicity and expressiveness of both object-orientation and predicate logic. FORM provides queries as an adjunct to automatic association traversal, which was mentioned above. Queries can be used to retrieve "original" objects from an extent, to provide optimizations for complex relationships, or to support general query needs for the end-user.

FORM provides three ways to express queries: ObjectRetriever queries, QueryDescription queries, and OQL queries. Of these options, ObjectRetriever provide only simple queries where as QueryDescriptions and OQL can handle all types of queries.



FORM's OQL is an integration of the ODMG's OQL, a correct relational model, and Java Syntax. The following is an example of using OQL[2] to retrieve all the employees for companies named "ChiMu Corporation":

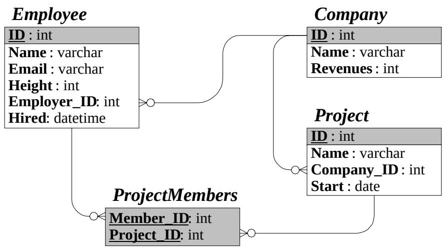
```
    Query query = orm.newOqlQuery("
        SELECT employee
        FROM Employee employee
        WHERE employee.employer.name = 'ChiMu Corporation'").query();
    Collection people = query.selectAll();
```

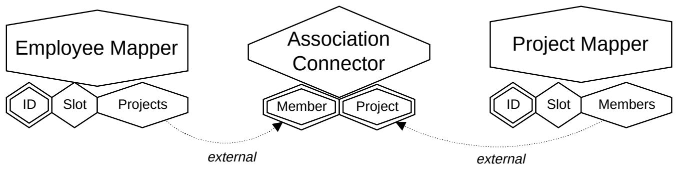
## 3.5 Inheritance and Extent Sharing

FORM separates what is commonly called "inheritance" into multiple components: Type inheritance, Class inheritance, and Extent sharing. FORM handles each of these independently and differently. Type inheritance is very important, but FORM does not deal with your domain class's public API (its Type/Interface), so you can do whatever you want with it. FORM configuration can be done such that subclasses extend the mapping specification of their superclass and so are compatible (or related) on the database mapping level. Finally, FORM can be configured such that objects from multiple classes can be viewed as one larger Extent and queries performed over all of them (over all People whether Employees or not). This last functionality is accomplished through **DistinguishingObjectMappers**, which allow multiple classes to share the same table, and **CollectiveObjectRetrievers**, which represent a collection of multiple types of objects.

## 3.6 External Associations

FORM can handle N-M associations using association tables. Within FORM these are called ExternalAssociations and are supported through **ExternalAssociationSlots** and **AssociationConnectors**. ExternalAssocationSlots work the same as any other type of slot for queries and traversals.





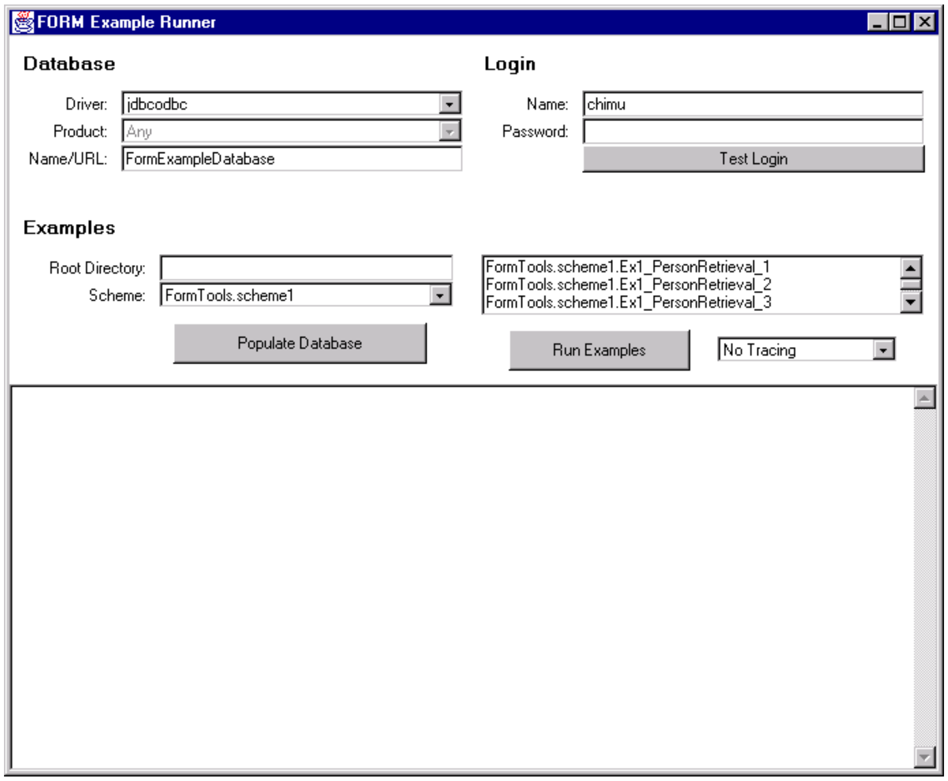
[2] With a formatting "cheat" of allowing carriage returns in Java strings.

## 3.7 Embedded Objects

Object models can have attributes that are not simple objects (e.g. Money or Length, instead of Strings, Numbers, or Booleans), but which are nonetheless basic attributes: The attribute object is completely contained in and private to another object. FORM provides the ability to **embed** the attribute object within the row describing the containing object. This prevents multiple objects from referring to the same attribute object without using a supplemental business rule. It can also improve performance when the attribute is frequently needed because it does not require an association traversal.

## 3.8 Tools

FORM comes with several tools to support automating the mapping process and try out examples. More tools will be added in the 1.1 release.



Driver: jdbcodbc
Product: Any
Name/URL: FormExampleDatabase
Name: chimu
Password:
Root Directory:
Scheme: FormTools.scheme1
FormTools.scheme1.Ex1_PersonRetrieval_1
FormTools.scheme1.Ex1_PersonRetrieval_2
FormTools.scheme1.Ex1_PersonRetrieval_3
No Tracing

## 3.9 Additions in Release 1.1

FORM Release 1.1 will include several additional features built on top of the 1.0 release, including a UI-based mapping builder, automatic generation of domain models and database schemes, and more automatic interfacing to domain objects. For many applications these tools will help speed development, but you will also know that you can diverge from the "standard" mappings and directly control the FORM engine. Your project can build the models that are optimal for your business needs, development standards, or database characteristics.

# 4 Where to go next

This document gave a very brief overview of FORM architecture, capabilities, and concepts. This overview should help you to understand FORM and help guide you to what you want to learn in more detail. Each of the topics discussed here is given a much more thorough description in *Learning FORM*. If you are more interested in examples, *Learning FORM* and the source code to the examples provide more detail. Finally, if you would like a higher level picture of object-relational mapping and architecture, you could look at *A Good Architecture for Object-Oriented Information Systems*, *Foundations for Object-Relational Mapping*, and the references contained in these two documents.

- *Getting Started with FORM* describes how to install FORM.
- *Learning FORM* is the main detailed documentation of FORM.
- The Examples show code that works with FORM.
- The *FORM API* Javadoc gives a complete description of the FORM interface.
- The *FORM Tools* manual describes the command line and graphical FORM tools in detail.

## *4.1 ChiMu Corporation References*

| | |
|---|---|
| **Architecture** | *A Good Architecture for Object-Relational Mapping* |
| **FORM Tools** | *FORM Tools* |
| **Foundations** | *Foundations of Object-Relational Mapping* |
| **Introduction** | *Introduction to FORM* |
| **Kernel** | *Kernel Frameworks* |
| **Learning** | *Learning FORM* |
| **Standards** | *Java Development Standards* |
| **Starting** | *Getting started with FORM* |

**ChiMu Corporation**

1220 N. Fair Oaks Ave, #1314
Sunnyvale, CA 94089

Phone: 408 734-9068
Email: info@chimu.com
www.chimu.com