JavaEE平台技术
MyBatis

邱明 博士

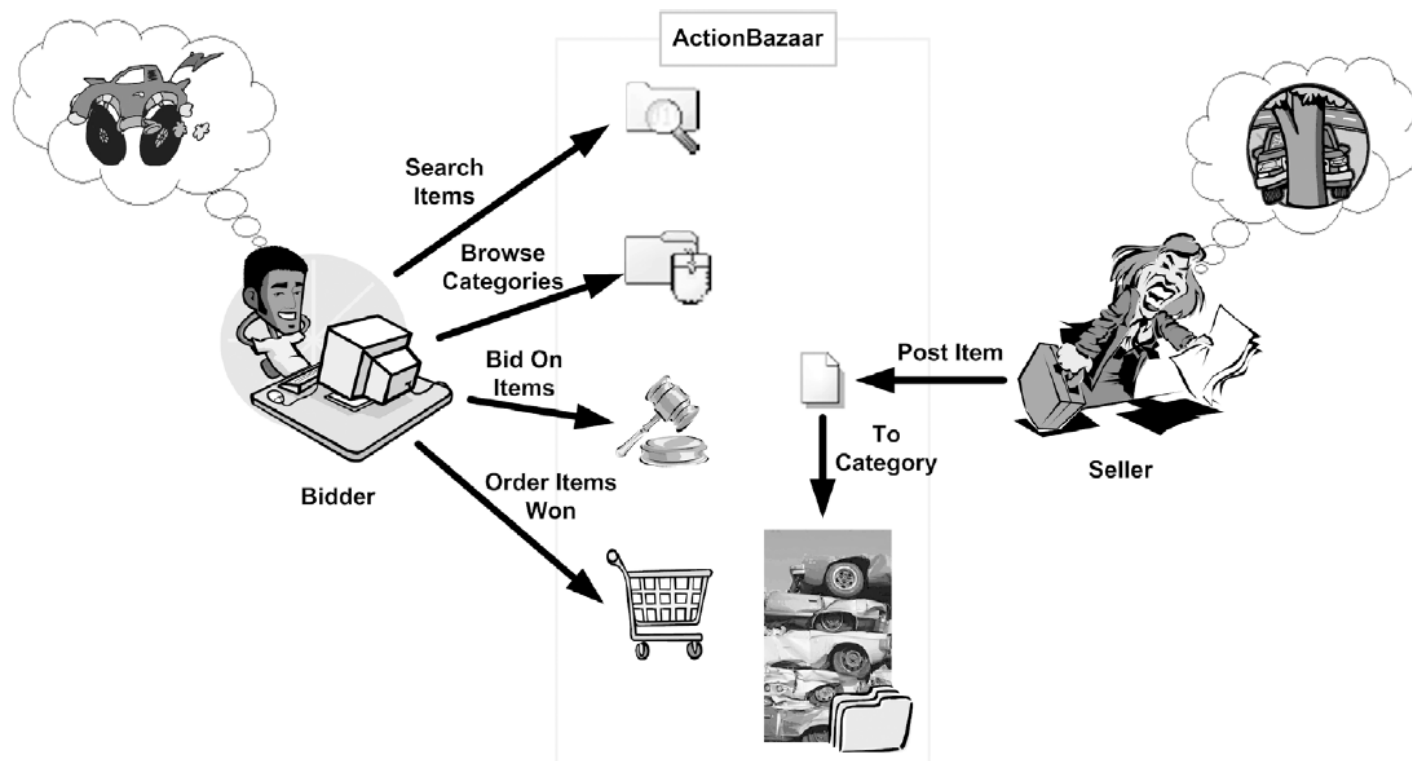厦门大学信息学院

mingqiu@xmu.edu.cn

1 对象关系映射

Figure 7.1 The core functionality of ActionBazaar. Sellers post items into searchable and navigable categories. Bidders bid on items and the highest bid wins.

1 对象关系映射
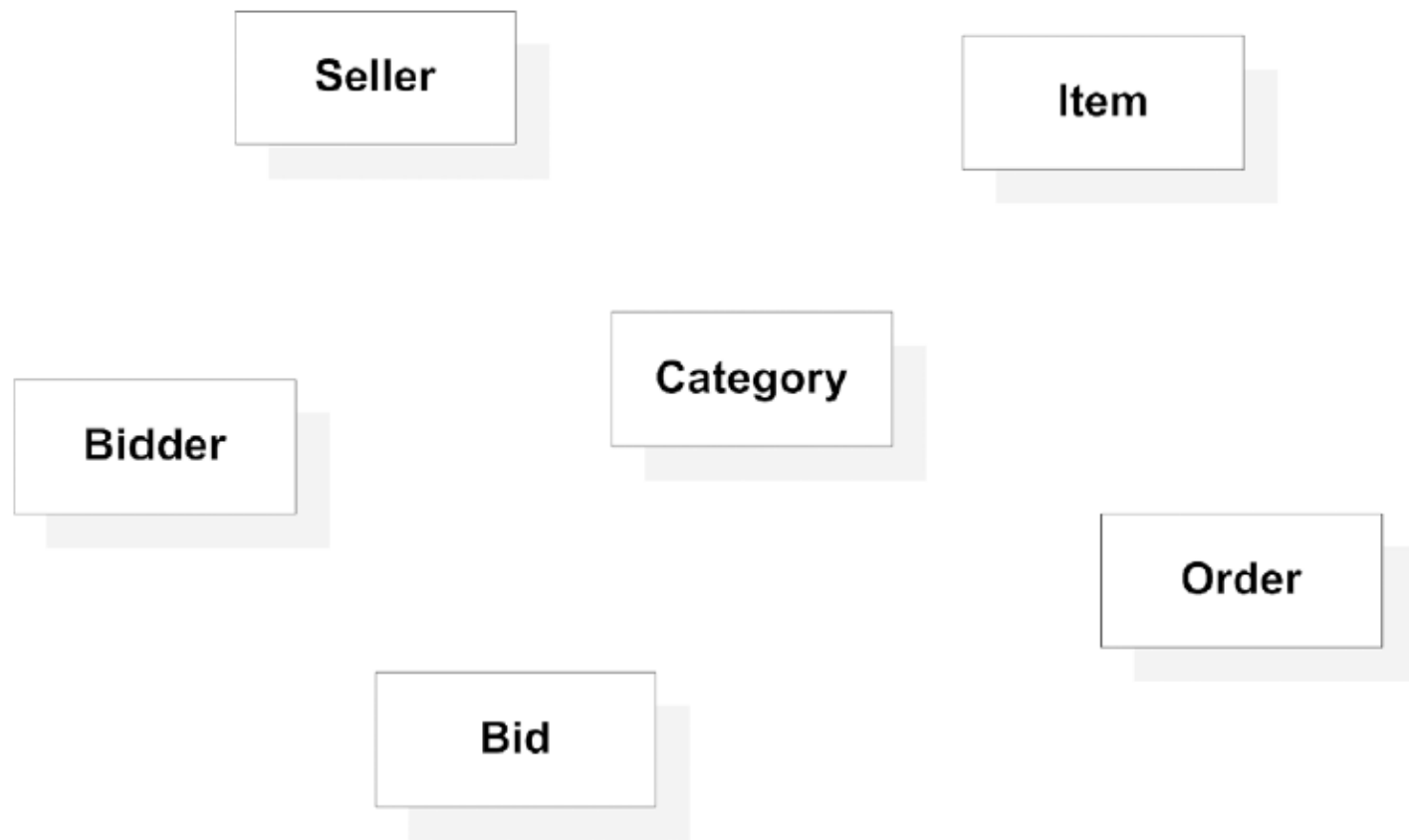

Seller

Item

Category

Bidder

Order

Bid

Figure 7.2
Entities are objects that can
be persisted in the database.
In the first step you identify
entities—for example,
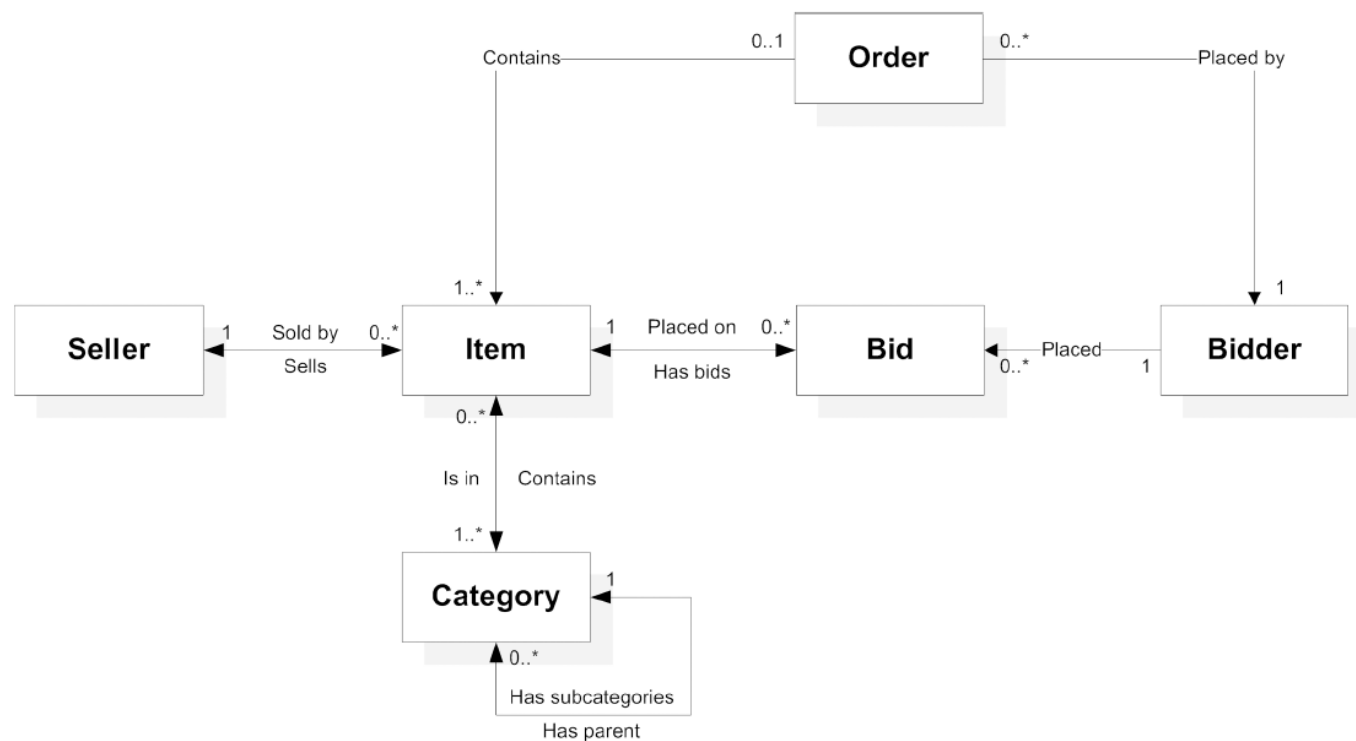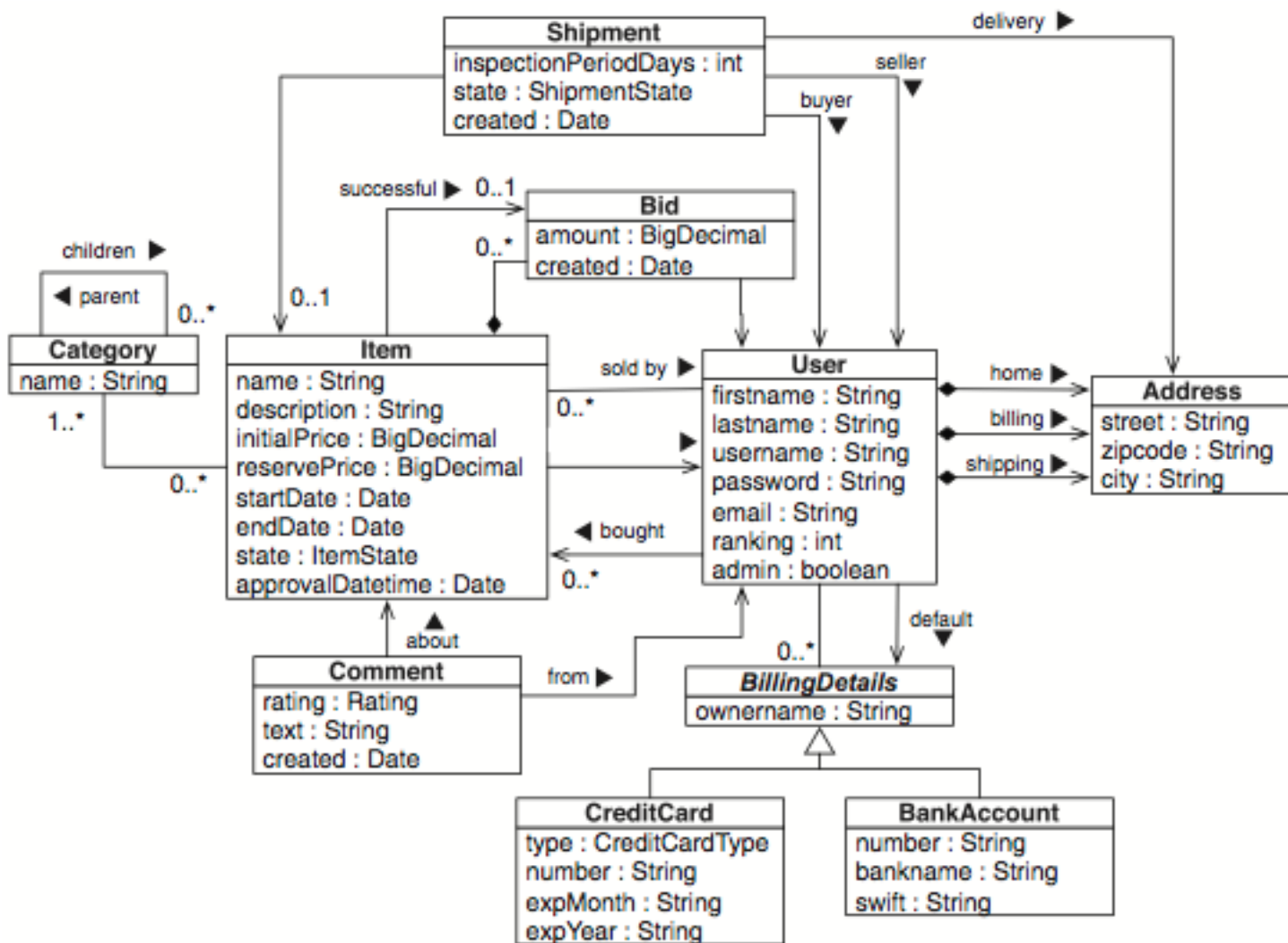entities in the ActionBazaar
domain.

1 对象关系映射

Figure 7.3   The ActionBazaar domain model complete with entities and relationships. Entities are related to one another and the relationship can be one-to-one, one-to-many, many-to-one, or many-to-many. Relationships can be either uni- or bidirectional.

1 对象关系映射

1 对象关系映射

- 两种对象关系框架
  - Hibernate基本上可以自动生成。其对数据库结构　　　的封装
    - 开发效率上Hibernate 比较快。
    - Hibernate自动生成的sql效果不理想
  - MyBatis是一个半自动化的对象-关系模型持久化相
    - 采用XML和标注把数据库记录映射成为Java对象，把JI　　　　　果处理代码从工程中移除。
    - MyBatis框架是以sql的开发方式，可以进行细粒度的优

Bank Account

String accountID
String ownerName
double balance

**Database API
Such as JDBC or
SQLJ**

Bank Account
Table

Relational Database
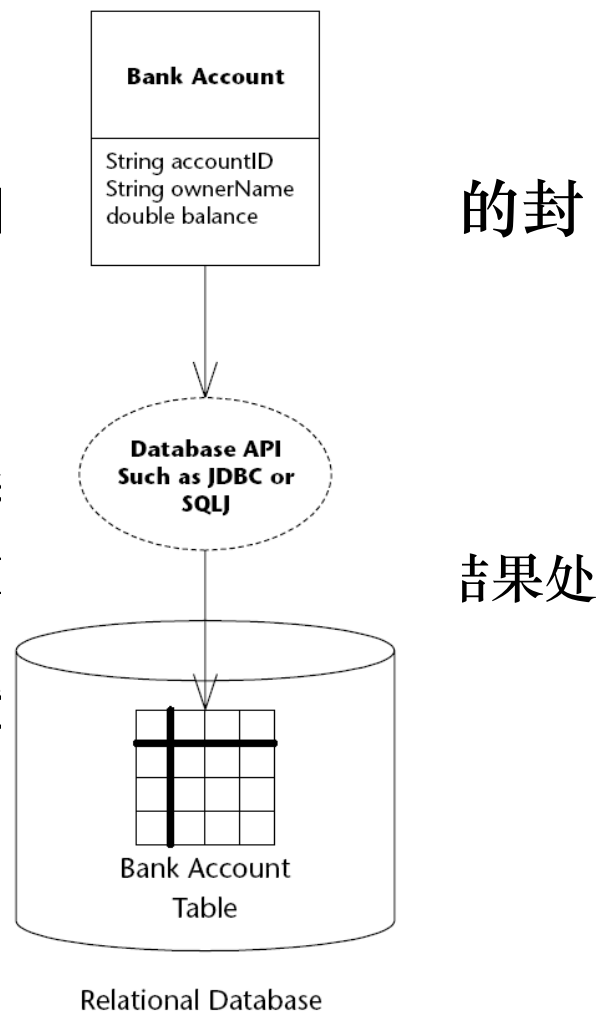
Figure 6.1　Object-relational mapping.

1 对象关系映射

面向对象模型	关系模型
类	表
对象	记录
属性	列
对象ID	主键
关联关系	外键
继承关系	不支持
方法	不支持

Figure 6.2 An example of object-relational mapping.

1 对象关系映射

Figure 9.1 UML object model.

1 对象关系映射

- 两种映射策略
  - 用一张表来记录所有具有继承关系的类。

Table 9.1　Persisted Data

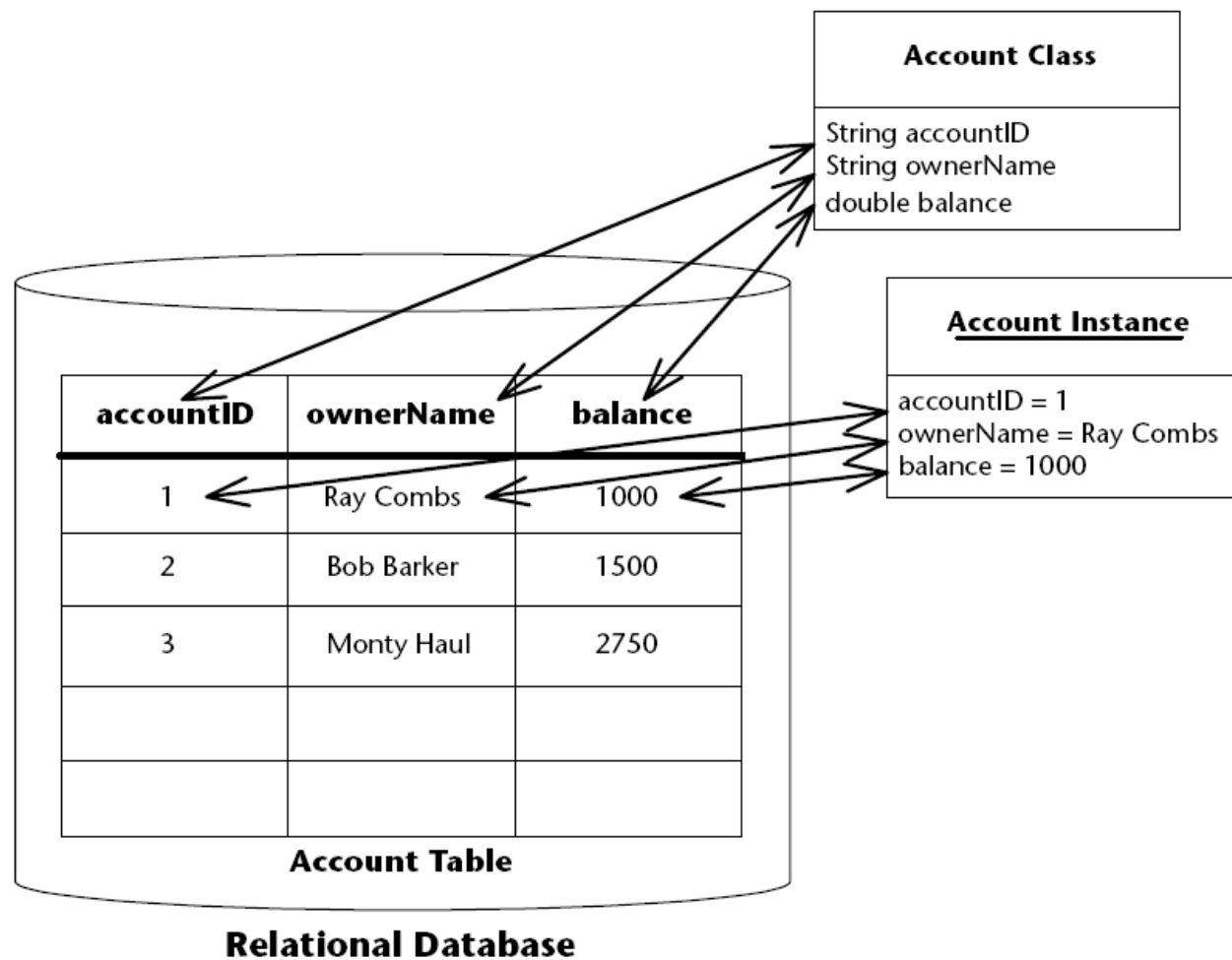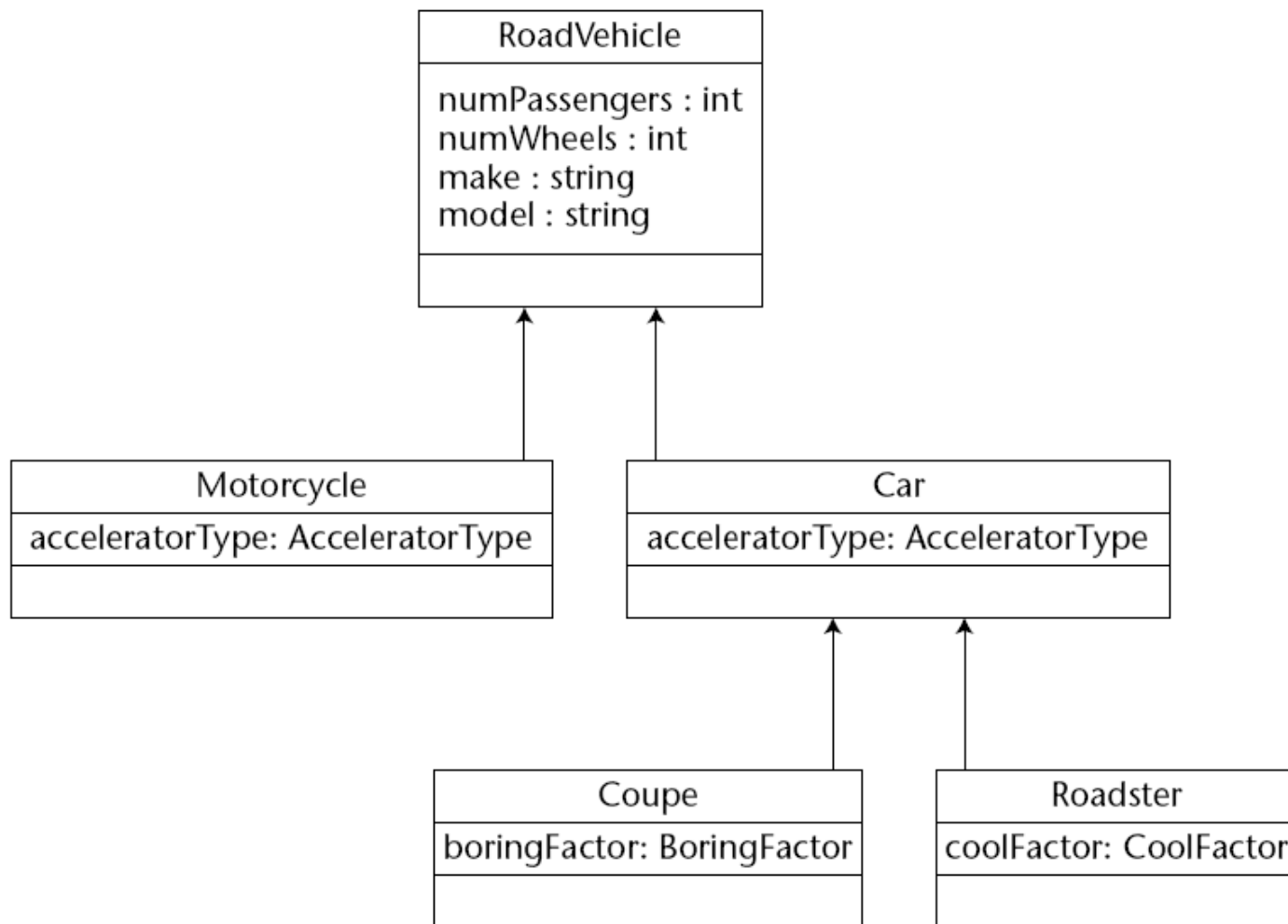| ID	DISC	MAKE	MODEL	COOL FACTOR	BORINGFACTOR
1818876882	COUPE	Bob	E400	NULL	0
1673414469	MOTORCYCLE	NULL	NULL	2	NULL
1673657791	ROADSTER	Mini	Cooper S	NULL	NULL

1 对象关系映射

- 两种映射策略
  - 针对每一个类一张表。

Table 9.2 ROADVEHICLEJOINED Table

ID	DTYPE	NUMWHEELS	MAKE	MODEL
1423656697	Coupe	4	Bob	E400
1425368051	Motorcycle	2	NULL	NULL
1424968207	Roadster	4	Mini	Cooper S

Table 9.3 MOTORCYCLE Table

ID	ACCELERATORTYPE
1425368051	1

1 对象关系映射

- 两种映射策略
  - 针对每一个类一张表。

Table 9.4 CAR Table

ID	ACCELERATORTYPE
1423656697	0
1424968207	0

Table 9.5 COUPE Table

ID	BORINGFACTOR
1423656697	0

Table 9.6 ROADSTER Table

ID	COOLFACTOR
1423656697	2

1 对象关系映射

- 一对一的关联关系

OrderPK	OrderName	Shipment ForeignPK
12345	Software Order	10101

ShipmentPK	City	ZipCode
10101	Austin	78727
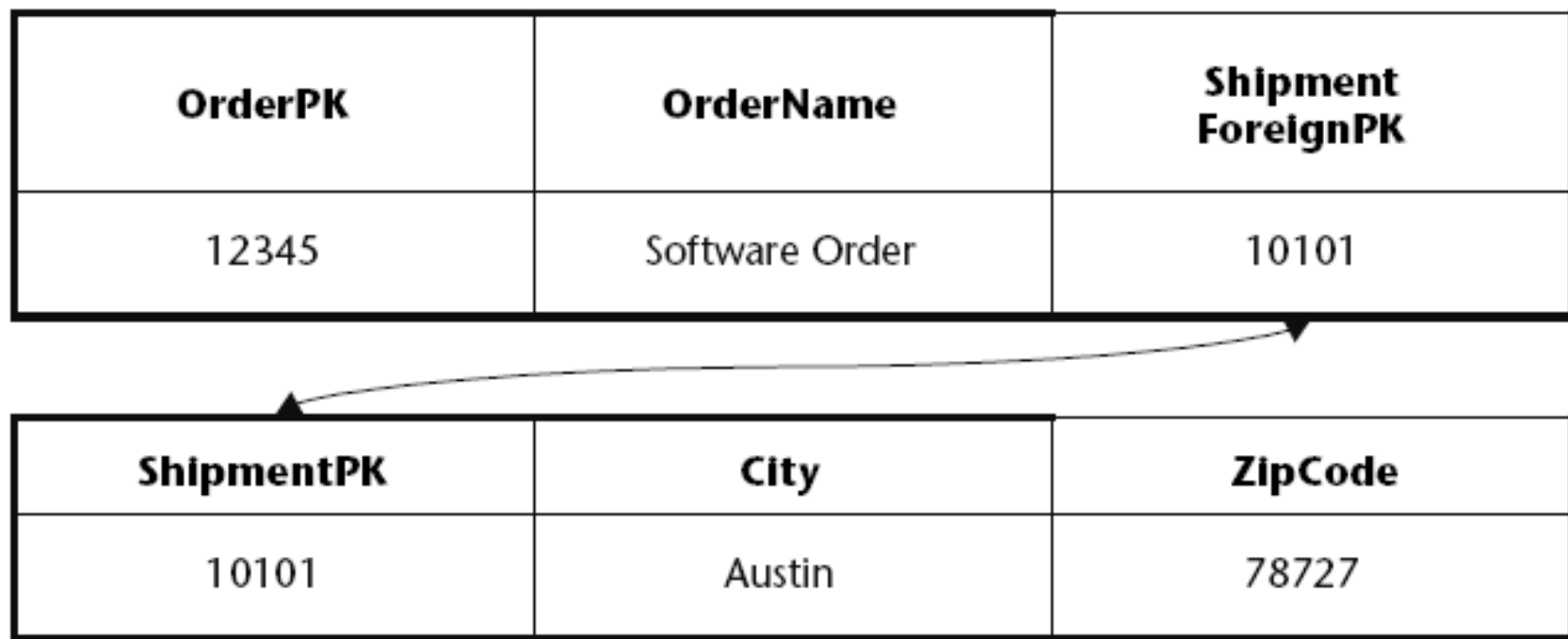
Figure 9.2 A possible one-to-one database schema.

1 对象关系映射

- 一对多的关联关系

id	name	Company
12345	M*Power Internet Services, Inc.	

company_id	employees_id	Company_Employee
12345	20202	
12345	20203	

id	name	sex	Employee
20202	Micah	M	
20203	Tes	F	
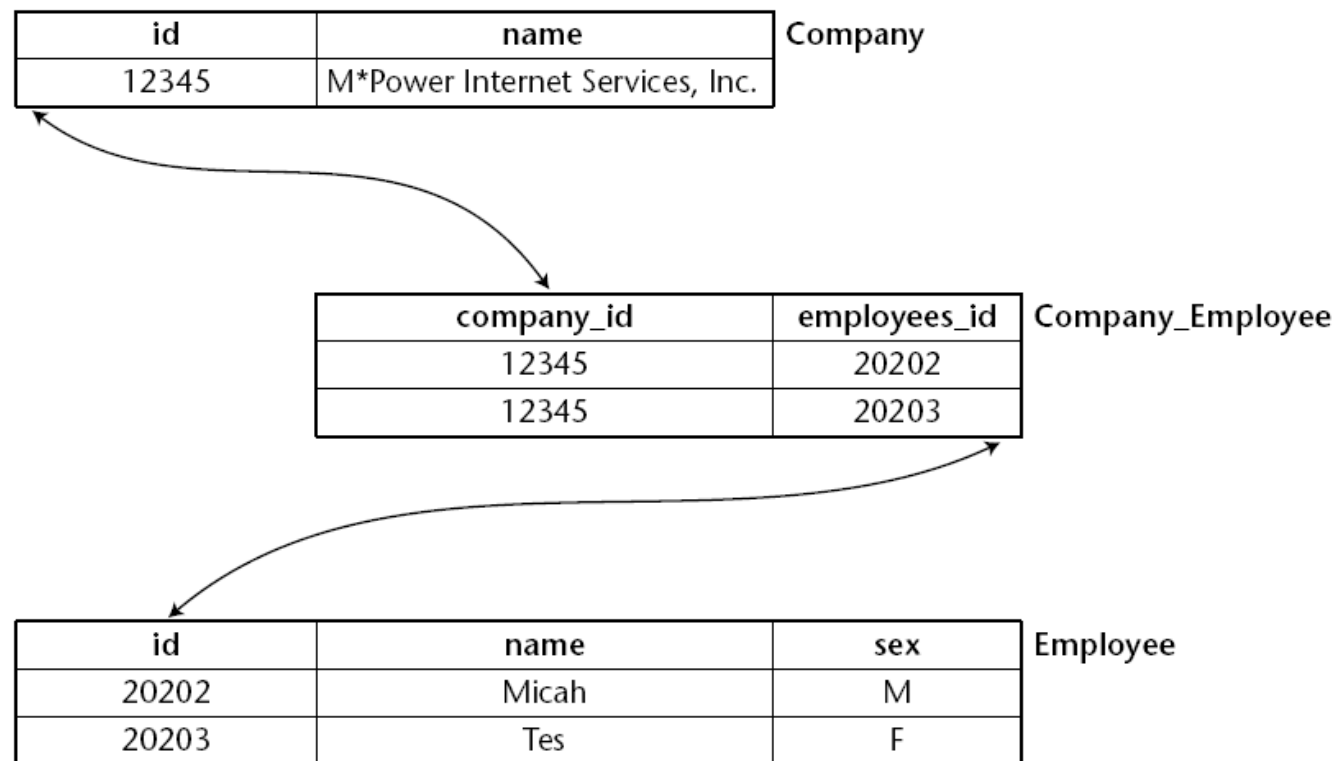
Figure 9.3 Unidirectional one-to-many relationship with join table.

1 对象关系映射

- 一对多的关联关系

CompanyPK	Name
12345	The Middleware Company

EmployeePK	Name	Sex	Company
20202	Ed	M	12345
20203	Floyd	M	12345
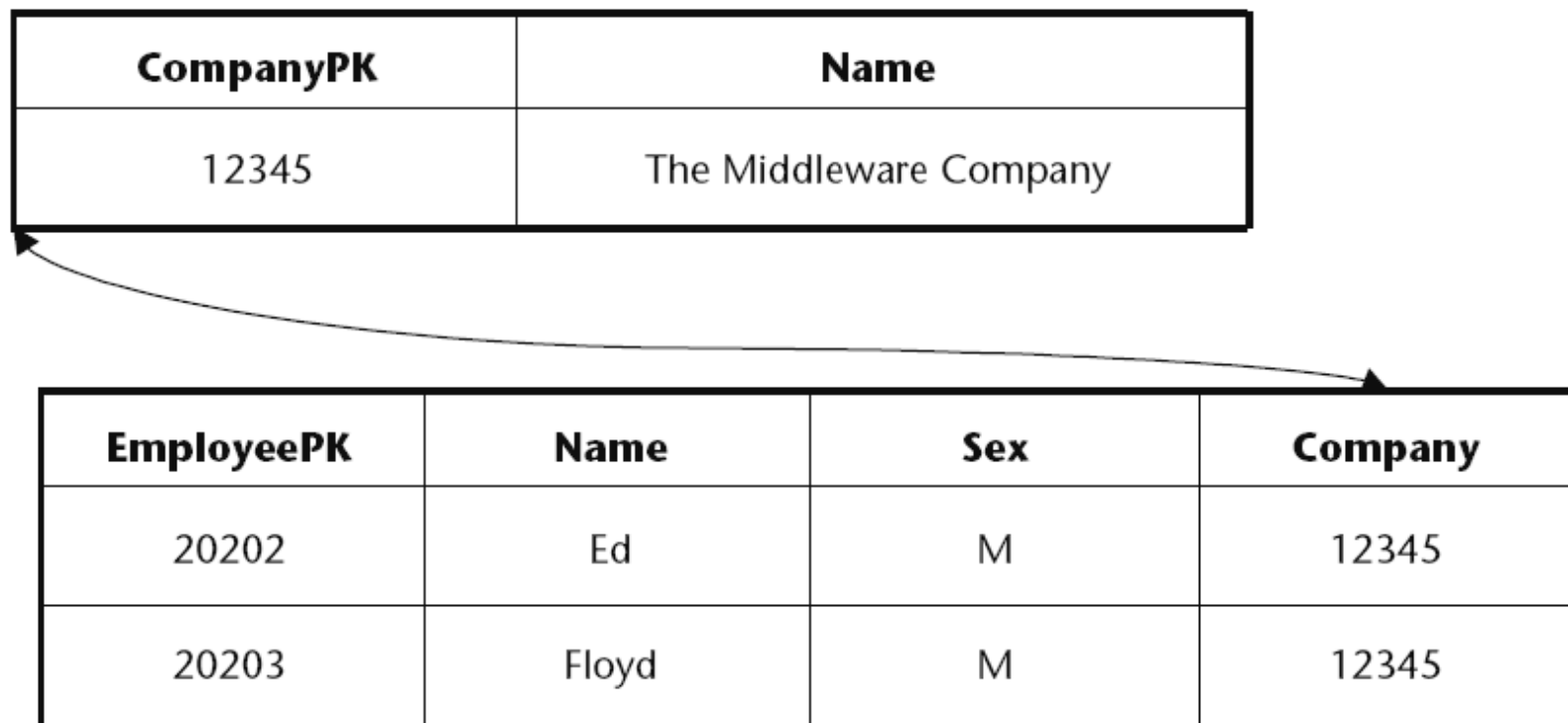
Figure 9.4 A one-to-many database schema.

1 对象关系映射

- 多对多的关联关系

StudentPK	StudentName
10101	Joe Student

EnrollmentPK	StudentPK	CoursePK
12345	10101	20202

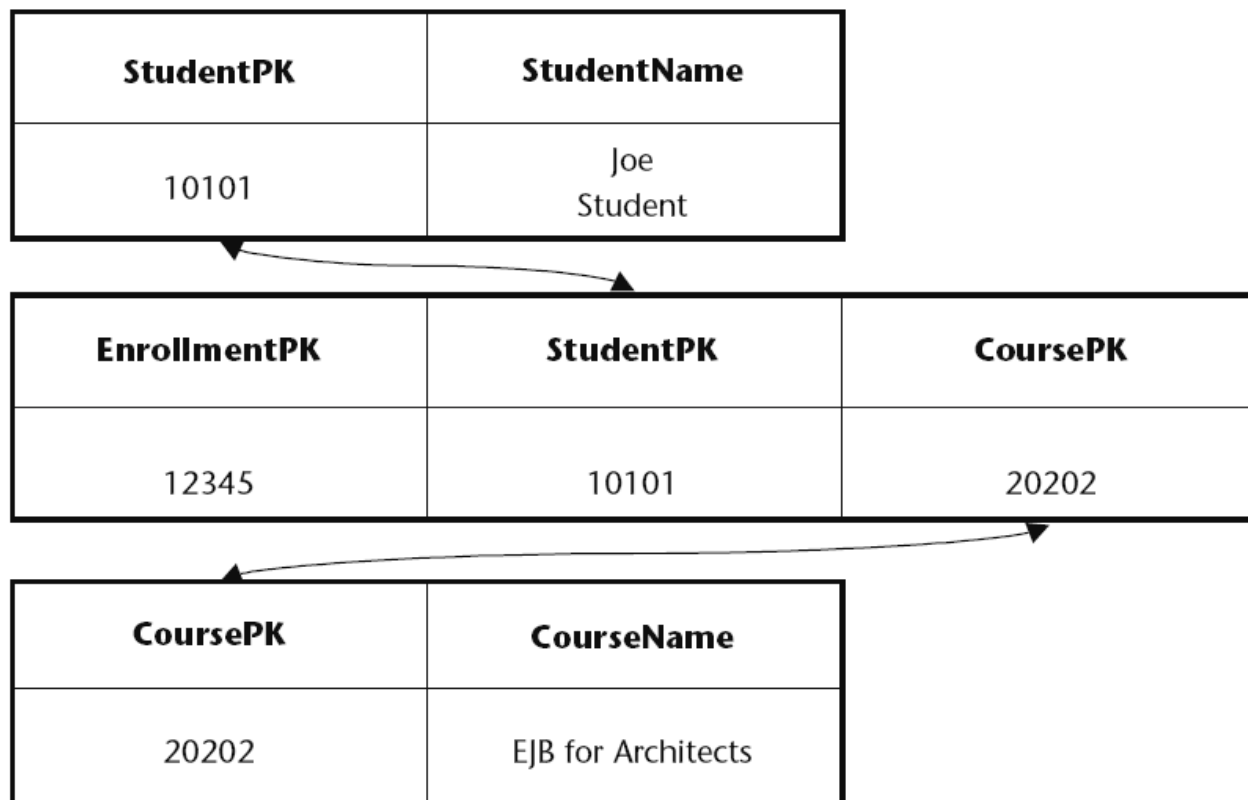| CoursePK	CourseName
20202	EJB for Architects

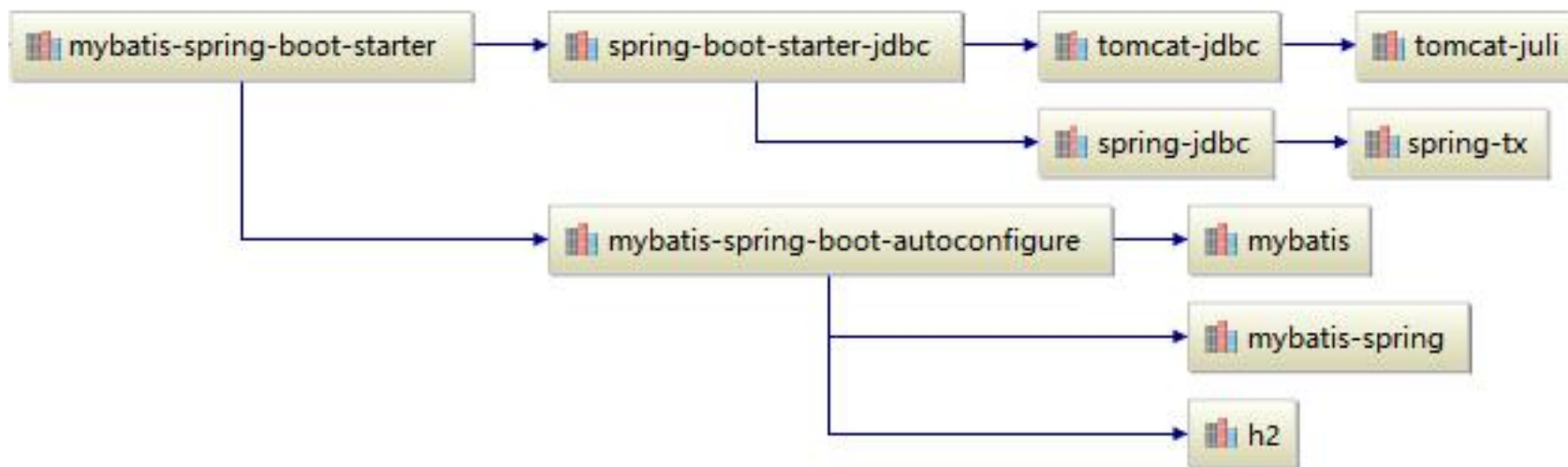**Figure 9.5**   A possible many-to-many database schema.

2 MyBatis结构

- Maven安装包

```
<dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-starter</artifactId>
</dependency>
```
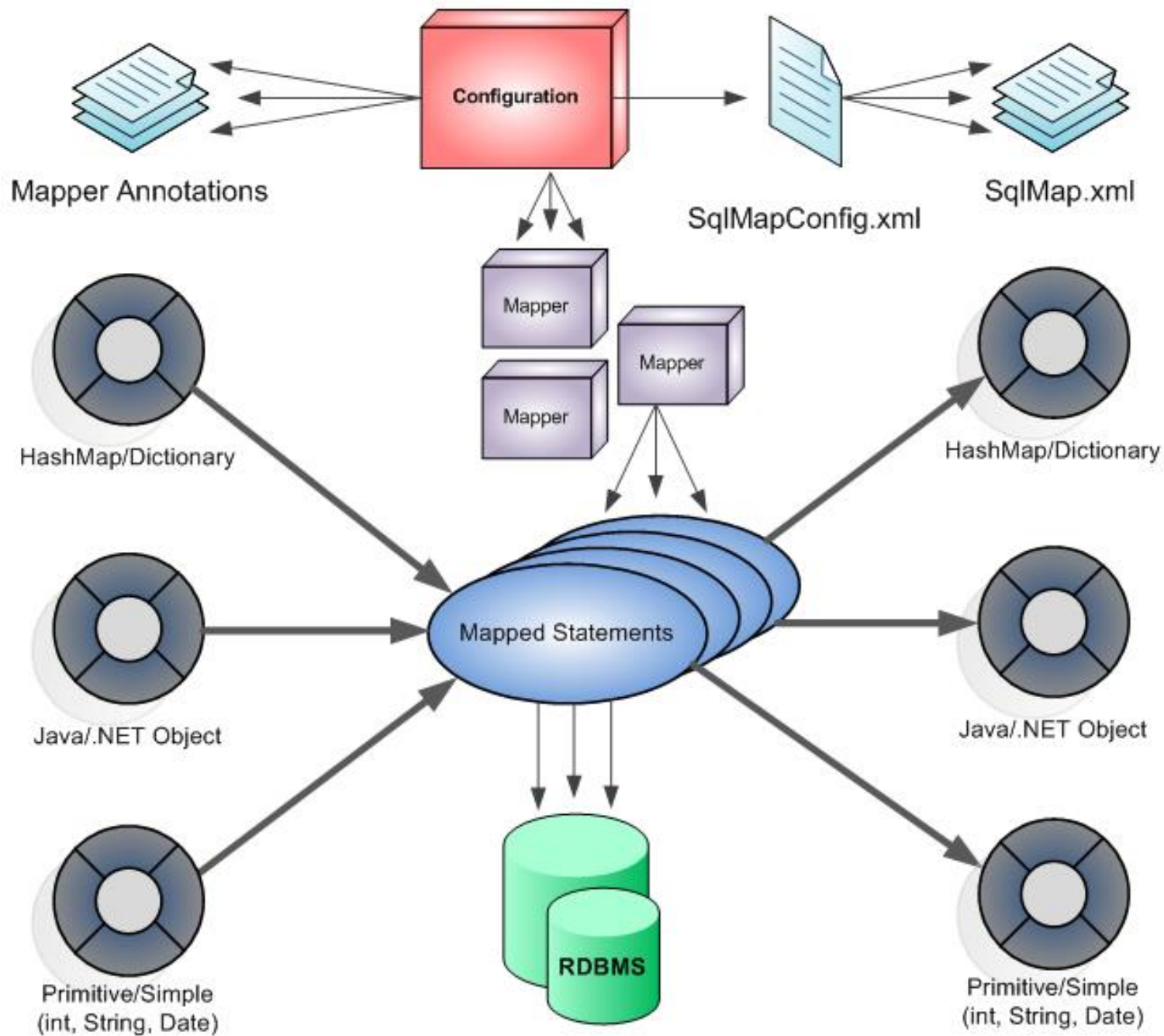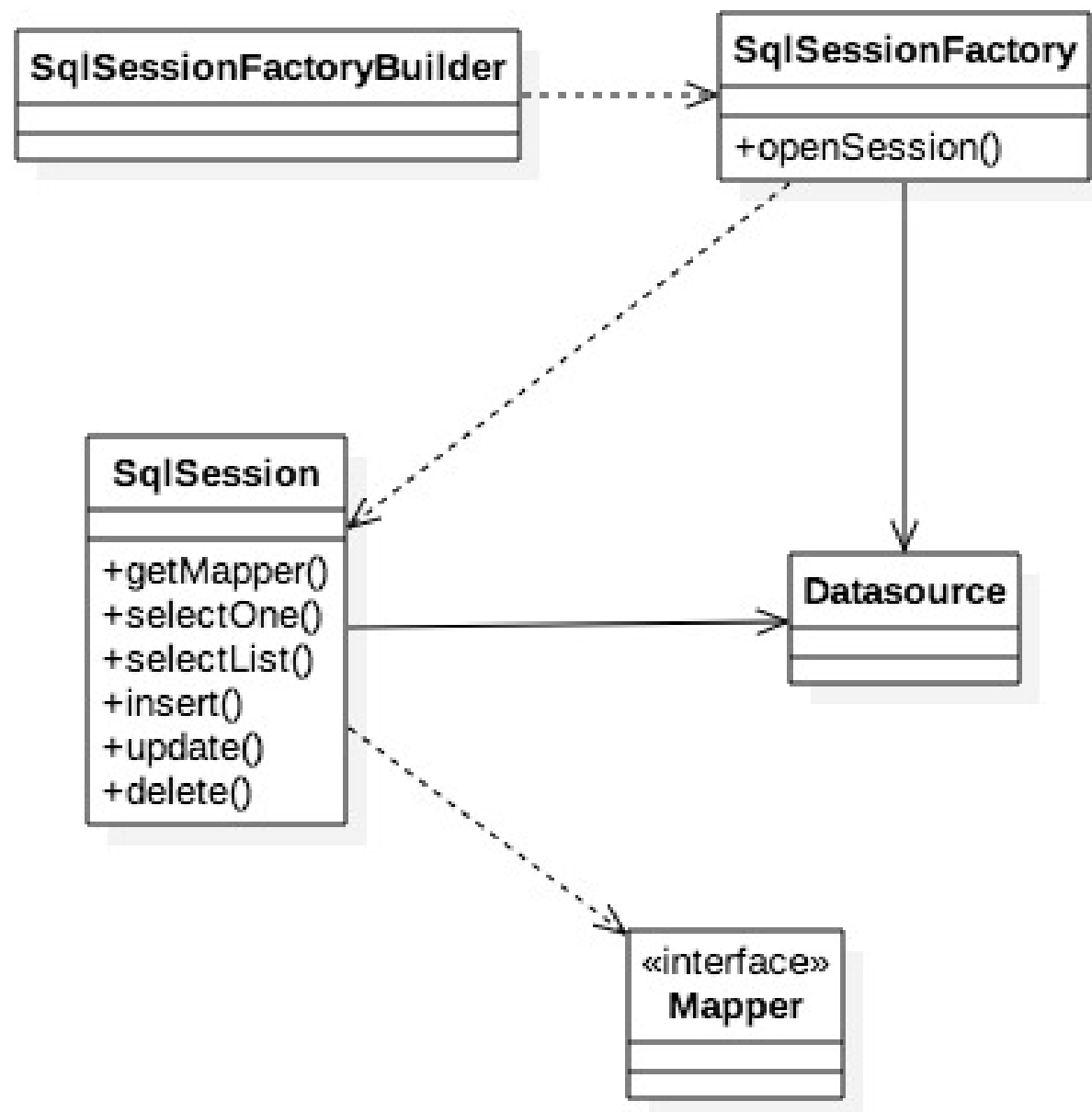
2 MyBatis结构

2 MyBatis结构

2 MyBatis结构

- MyBatis-Spring-Boot-Starter
  - 自动识别已存在的DataSource
  - 创建并登记一个SqlSessionFactory的对象
  - 用SqlSessionFactory创建并登记一个SqlSessionTemplate的对象
  - 自动扫描并创建Mapper对象，将它们与SqlSessionTemplate对象关联，并登记到Spring上下文中，以备将来注入Bean中

2 MyBatis结构

```java
@Mapper
public interface GoodsMapper {
    /**
     * 用GoodsPo对象找,
     * @param goodsPo 条件对象，所有条件为AND，仅有索引的值可以作为条件
     * @return  GoodsPo对象列表
     */
    List<GoodsPo> findGoods(GoodsPo goodsPo);

    /**
     * 用GoodsPo对象找，带Product对象返回
     * @param goodsPo 条件对象，所有条件为AND，仅有索引的值可以作为条件
     * @return  GoodsPo对象列表
     */
    List<GoodsPo> findGoodsWithProduct(GoodsPo goodsPo);
```

2 MyBatis结构

```xml
<select id="findGoods" parameterType="GoodsPo" resultType="GoodsPo">
    SELECT
    <include refid="Goods_Column_List"/>
    FROM oomall_goods
    WHERE
    state != 2
    <if test="id!=null">and id = #{id} </if>
    <if test="goodsSn!=null and goodsSn!=''">and goods_sn = #{goodsSn} </if>
    <if test="name!=null and name!=''">and name = #{name} </if>
    <if test="categoryId!=null">and category_id = #{categoryId} </if>
    <if test="brandId!=null">and brand_id = #{brandId} </if>
    <if test="state!=null">and state = #{state} </if>
</select>
```

3. MyBatis的映射标记

- insert：用于映射INSERT SQL语句
- update：用于映射UPDATE SQL语句
- delete：用于映射DELETE SQL语句
- select：用于映射SELECT SQL语句
- resultMap：用于把数据库的结果集映射成对象
- sql: 可重用的SQL代码片段

3. MyBatis的映射标记

- select

```
<select id="selectPerson" parameterType="int" resultType="Person">
    SELECT * FROM PERSON WHERE ID = #{id}
</select>
```

3. MyBatis的映射标记

• Insert, update, delete

```
<insert id="insertAuthor" useGeneratedKeys="true" keyProperty="id">
    insert into Author (username,password,email,bio) values
(#{username},#{password},#{email},#{bio})
 </insert>

<update id="updateAuthor">
    update Author set username = #{username}, password = #{password}, email = #{email}, bio
= #{bio} where id = #{id}
</update>

<delete id="deleteAuthor"> delete from Author where id = #{id} </delete>
```

3. MyBatis的映射标记

- Sql
  - 用来定义可重用的 SQL 代码段，这些 SQL 代码可以被包含在其他语句中。它可以（在加载的时候）被静态地设置参数。 在不同的包含语句中可以设置不同的值到参数占位符上

```xml
<sql id="sometable"> ${prefix}Table </sql>

<sql id="someinclude">
  from
  <include refid="${include_target}"/>
</sql>

<select id="select" resultType="map">
    select field1, field2, field3
    <include refid="someinclude">
        <property name="prefix" value="Some"/>
        <property name="include_target" value="sometable"/>
    </include>
</select>
```

3. MyBatis的映射标记

- Parameter
  - 用#{}来表示SQL中的参数

```
<select id="selectUsers" resultType="User">
    select id, username, password from users where id = #{id} </select>


<insert id="insertUser" parameterType="User">
    insert into users (id, username, password) values (#{id}, #{username}, #{password})
</insert>
```

3. MyBatis的映射标记

- ResultMap
  - MyBatis 会自动创建一个 ResultMap，基于属性名来映射记录的列到 JavaBean 的属性上。

```xml
<!-- In Config XML file -->
<typeAlias type="com.someapp.model.User" alias="User"/>

<!-- In SQL Mapping XML file -->
<select id="selectUsers" resultType="User">
    select id, username, hashedPassword from some_table where id = #{id}
</select>
```

3. MyBatis的映射标记

- ResultMap
  - 列名和属性名没有精确匹配，可以在 SELECT 语句中对列使用别名来匹配对象属性

```
<select id="selectUsers" resultType="User">
    select user_id as "id",
            user_name as "userName",
            hashed_password as "hashedPassword"
    from some_table where id = #{id}
</select>
```
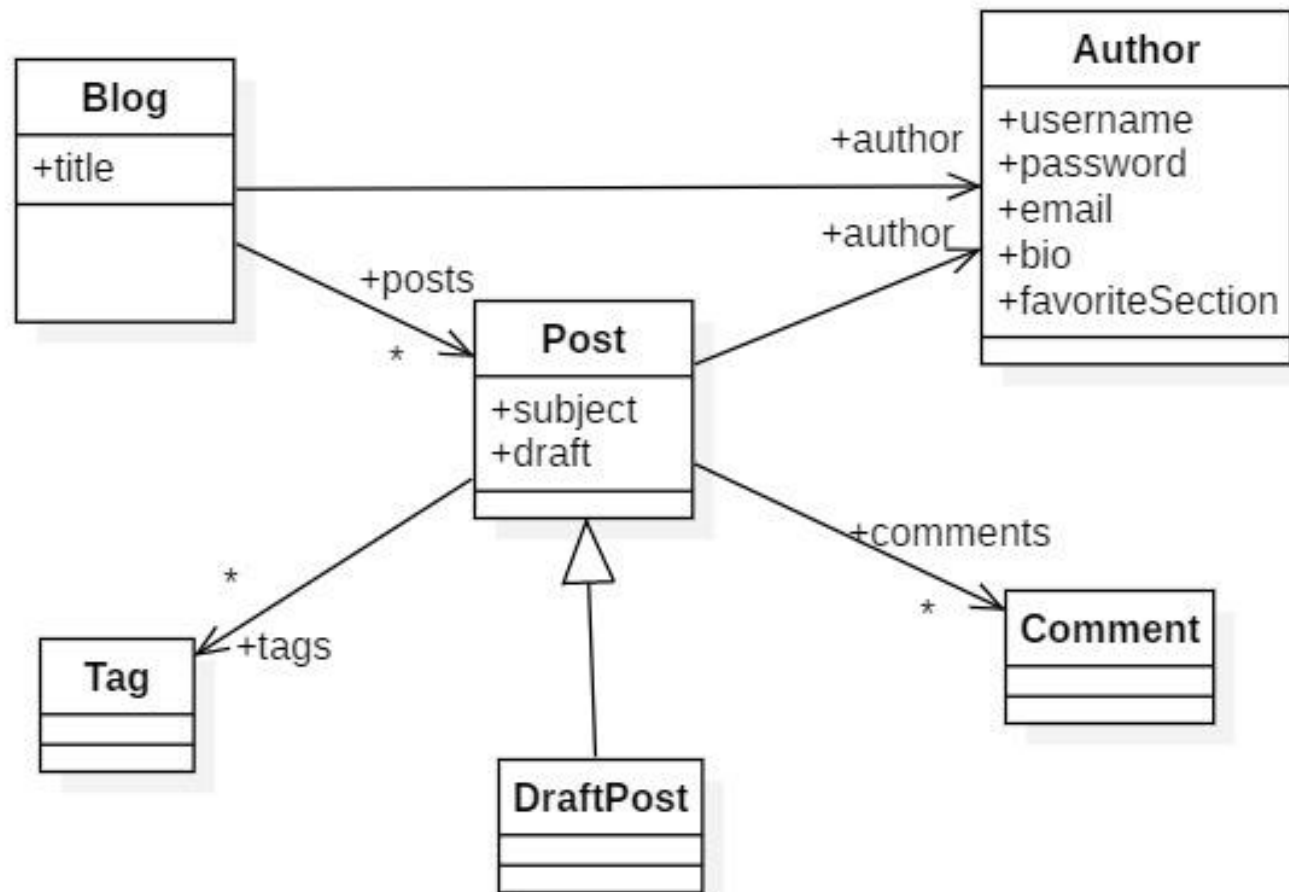
3. MyBatis的映射标记

- ResultMap
  - 当然也可以使用ResultMap

```xml
<resultMap id="userResultMap" type="User">
    <id property="id" column="user_id" />
    <result property="username" column="user_name"/>
    <result property="password" column="hashed_password"/>
</resultMap>
```

3. MyBatis的映射标记

```xml
<!-- Very Complex Result Map -->
<resultMap id="detailedBlogResultMap" type="Blog">
        <constructor>
                <idArg column="blog_id" javaType="int"/>
        </constructor>
        <result property="title" column="blog_title"/>
        <association property="author" javaType="Author">
                <id property="id" column="author_id"/>
                <result property="username" column="author_username"/>
                <result property="password" column="author_password"/>
                <result property="email" column="author_email"/>
                <result property="bio" column="author_bio"/>
                <result property="favouriteSection" column="author_favourite_section"/>
        </association>
        <collection property="posts" ofType="Post">
                <id property="id" column="post_id"/>
                <result property="subject" column="post_subject"/>
                <association property="author" javaType="Author"/>
                <collection property="comments" ofType="Comment">
                        <id property="id" column="comment_id"/>
                </collection>
                <collection property="tags" ofType="Tag" >
                        <id property="id" column="tag_id"/>
                </collection>
                <discriminator javaType="int" column="draft">
                        <case value="1" resultType="DraftPost"/>
                </discriminator>
        </collection>
</resultMap>
```

3. MyBatis的映射标记

- ResultMap
  - 构造方法

```java
public class User {
    //...
    public User(Integer id, String username, int age) {
        //...
    }
    //...
}
```

```xml
<constructor>
    <idArg column="id" javaType="int" name="id" />
    <arg column="age" javaType="_int" name="age" />
    <arg column="username" javaType="String" name="username" />
</constructor>
```

3. MyBatis的映射标记

- ResultMap
  - 关联

```xml
<resultMap id="blogResult" type="Blog">
  <association property="author" column="author_id" javaType="Author" select="selectAuthor"/>
</resultMap>

<select id="selectBlog" resultMap="blogResult">
  SELECT * FROM BLOG WHERE ID = #{id}
</select>


<select id="selectAuthor" resultType="Author">
  SELECT * FROM AUTHOR WHERE ID = #{id}
</select>
```

column	数据库中的列名，在使用复合主键的时候，可以使用 column="{prop1=col1,prop2=col2}" 来指定多个传递给嵌套 Select 查询语句的列名。这会使得 prop1 和 prop2 作为参数对象，被设置为对应嵌套 Select 语句的参数。
select	用于加载复杂类型属性的映射语句的 ID，它会从 column 属性指定的列中检索数据，作为参数传递给目标 select 语句。 在使用复合主键的时候，可以用 column="{prop1=col1,prop2=col2}" 来指定多个传递给嵌套 Select 查询语句的列名。这会使得 prop1 和 prop2 作为参数对象，被设置为对应嵌套 Select 语句的参数。
fetchType	可选的。有效值为 lazy 和 eager。 指定属性后，将在映射中忽略全局配置参数 lazyLoadingEnabled，使用属性的值。

3. MyBatis的映射标记

- ResultMap
  - 关联

```xml
<resultMap id="blogResult" type="Blog">
    <collection property="posts" javaType="ArrayList" column="id" ofType="Post" select="selectPostsForBlog"/>
</resultMap>

<select id="selectBlog" resultMap="blogResult">
    SELECT * FROM BLOG WHERE ID = #{id}
</select>

<select id="selectPostsForBlog" resultType="Post">
    SELECT * FROM POST WHERE BLOG_ID = #{id}
</select>
```

3. MyBatis的映射标记

- ResultMap
  - 鉴别器 – 用于生成继承关系的对象

```xml
<resultMap id="vehicleResult" type="Vehicle">
    <id property="id" column="id" />
    <result property="vin" column="vin"/>
    <result property="year" column="year"/>
    <result property="make" column="make"/>
    <result property="model" column="model"/>
    <result property="color" column="color"/>
    <discriminator javaType="int" column="vehicle_type">
        <case value="1" resultMap="carResult"/>
        <case value="2" resultMap="truckResult"/>
        <case value="3" resultMap="vanResult"/>
        <case value="4" resultMap="suvResult"/>
    </discriminator>
</resultMap>
```

3. MyBatis的映射标记

```xml
<resultMap id="vehicleResult" type="Vehicle">
  <id property="id" column="id" />
  <result property="vin" column="vin"/>
  <result property="year" column="year"/>
  <result property="make" column="make"/>
  <result property="model" column="model"/>
  <result property="color" column="color"/>
  <discriminator javaType="int" column="vehicle_type">
    <case value="1" resultType="carResult">
      <result property="doorCount" column="door_count" />
    </case>
    <case value="2" resultType="truckResult">
      <result property="boxSize" column="box_size" />
      <result property="extendedCab" column="extended_cab" />
    </case>
    <case value="3" resultType="vanResult">
      <result property="powerSlidingDoor" column="power_sliding_door" />
    </case>
    <case value="4" resultType="suvResult">
      <result property="allWheelDrive" column="all_wheel_drive" />
    </case>
  </discriminator>
</resultMap>
```

3. MyBatis的映射标记

- 动态SQL
  - if
  - choose (when, otherwise)
  - trim (where, set)
  - foreach

3. MyBatis的映射标记

- if

```
<select id="findActiveBlogWithTitleLike" resultType="Blog">
    SELECT * FROM BLOG WHERE state = 'ACTIVE'
    <if test="title != null"> AND title like #{title} </if>
</select>


<select id="findActiveBlogLike" resultType="Blog">
    SELECT * FROM BLOG WHERE state = 'ACTIVE'
    <if test="title != null"> AND title like #{title} </if>
    <if test="author != null and author.name != null">
        AND author_name like #{author.name}
    </if>
</select>
```

3. MyBatis的映射标记

- choose (when, otherwise)

```xml
<select id="findActiveBlogLike" resultType="Blog">
    SELECT * FROM BLOG WHERE state = 'ACTIVE'
    <choose>
        <when test="title != null"> AND title like #{title} </when>
        <when test="author != null and author.name != null">
            AND author_name like #{author.name}
        </when>
        <otherwise> AND featured = 1 </otherwise>
    </choose>
</select>
```

3. MyBatis的映射标记

- trim (where, set)

```xml
<select id="findActiveBlogLike" resultType="Blog">
    SELECT * FROM BLOG
    <trim prefix="WHERE" prefixOverrides="AND |OR ">
        <if test="state != null"> state = #{state} </if>
        <if test="title != null"> AND title like #{title} </if>
        <if test="author != null and author.name != null">
            AND author_name like #{author.name}
        </if>
    </trim>
</select>
```

3. MyBatis的映射标记

- trim (where, set)

```xml
<update id="updateAuthorIfNecessary">
    update Author
    <trim prefix="SET" suffixOverrides=",">
        <if test="username != null">username=#{username},</if>
        <if test="password != null">password=#{password},</if>
        <if test="email != null">email=#{email},</if>
        <if test="bio != null">bio=#{bio}</if>
    </trim>
    where id=#{id}
</update>
```

3. MyBatis的映射标记

- foreach

```xml
<select id="selectPostIn" resultType="domain.blog.Post">
        SELECT * FROM POST P WHERE ID in
            <foreach item="item" index="index" collection="list" open="(" separator="," close=")">
                #{item}
            </foreach>
</select>
```

```java
public List<Blog> selectPostIn(List<Integer> ids);
```

3. MyBatis的映射标记

- 配置信息（application.properties）

mapper-locations	Locations of Mapper xml config file.
type-aliases-package	Packages to search for type aliases. (Package delimiters are ",; \t\n")
type-handlers-package	Packages to search for type handlers. (Package delimiters are ",; \t\n")
executor-type	Executor type: SIMPLE, REUSE, BATCH.
configuration-properties	Externalized properties for MyBatis configuration. Specified properties can be used as placeholder on MyBatis config file and Mapper file. For detail see the MyBatis reference page
configuration	A MyBatis Configuration bean. About available properties see the MyBatis reference page.　　　This property cannot be used at the same time with the config-location.

4. MyBatis Generator

- 一个基于 MyBatis 的独立工具，
- 通过简单的配置去帮我们生成数据表所对应的 PO、Mapper和 XML 文件

```xml
<plugin>
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-maven-plugin</artifactId>
    <version>1.4.1</version>
    <configuration>
        <!--mybatis的代码生成器的配置文件-->
        <configurationFile>src/main/resources/mybatis-generator-config.xml</configurationFile>
        <!--允许覆盖生成的文件-->
        <overwrite>true</overwrite>
        <!--将当前pom的依赖项添加到生成器的类路径中-->
        <includeCompileDependencies>true</includeCompileDependencies>
    </configuration>
    <dependencies>
        <dependency>
            <groupId>org.mybatis.generator</groupId>
            <artifactId>mybatis-generator-core</artifactId>
            <version>1.4.1</version>
        </dependency>
    </dependencies>
</plugin>
```

5. Spring的事务

• 事务（@Transaction）

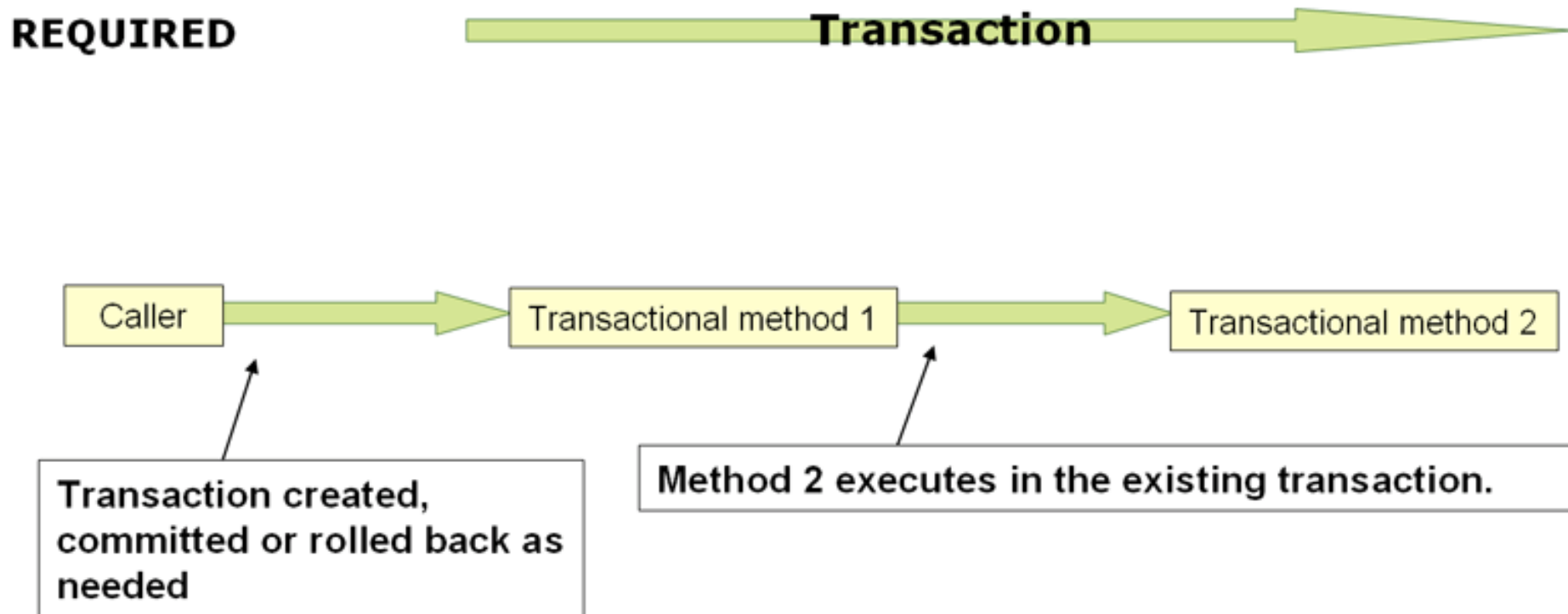| 属性	类型	描述
value	String	可选的限定描述符，指定使用的事务管理器
propagation	enum: Propagation	可选的事务传播行为设置
isolation	enum: Isolation	可选的事务隔离级别设置，只在REQUIRED和REQUIRES_NEW的事务中有效，默认为ISOLATION_DEFAULT，即数据库的默认的隔离级别
readOnly	boolean	读写或只读事务，默认读写
timeout	int (in seconds granularity)	事务超时时间设置
rollbackFor	Class对象数组，必须继承自Throwable	用于指定能够触发事务回滚的异常类型数组。
rollbackForClassName	类名数组，必须继承自Throwable	用于指定能够触发事务回滚的异常类名字数组
noRollbackFor	Class对象数组，必须继承自Throwable	不会导致事务回滚的异常类数组
noRollbackForClassName	类名数组，必须继承自Throwable	不会导致事务回滚的异常类名字数组

5. Spring的事务

- 事务传播设置（propagation）

传播设置	描述
Propagation.REQUIRED	如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是默认设置。
Propagation.SUPPORTS	支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。
Propagation.MANDATORY	支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。
Propagation.REQUIRES_NEW	创建新事务，无论当前存不存在事务，都创建新事务。
Propagation.NOT_SUPPORTED	以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
Propagation.NEVER	以非事务方式执行，如果当前存在事务，则抛出异常。
Propagation.NESTED	如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与Propagation.REQUIRED类似的操作。

5. Spring的事务

- 事务传播设置（propagation）

5. Spring的事务

- 事务传播设置（propagation）

REQUIRES_NEW

Transaction 1

Transaction 2

Caller → Transactional method 1 → Transactional method 2

Transaction created, committed or rolled back as needed

Method 2 executes in a new transaction, and the outer transaction is suspended.