JavaEE平台技术
MyBatis和数据库

邱明 博士

厦门大学信息学院

mingqiu@xmu.edu.cn

提纲

- 对象关系映射
- MyBatis
- MyBatis的映射标记
- MyBatis的事务
- 查询优化

1 对象关系映射

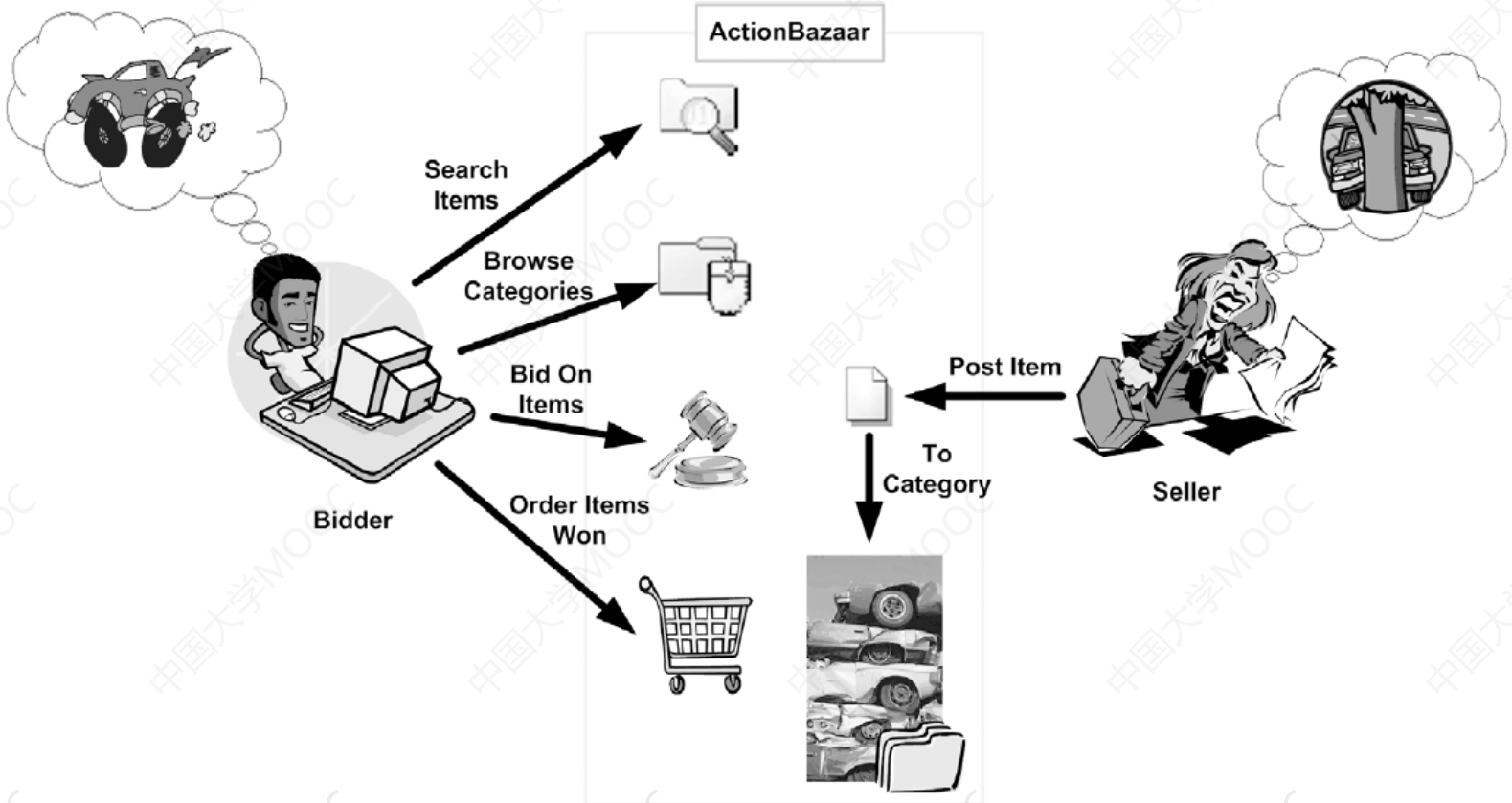


Figure 7.1 The core functionality of ActionBazaar. Sellers post items into searchable and navigable categories. Bidders bid on items and the highest bid wins.

1 对象关系映射

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

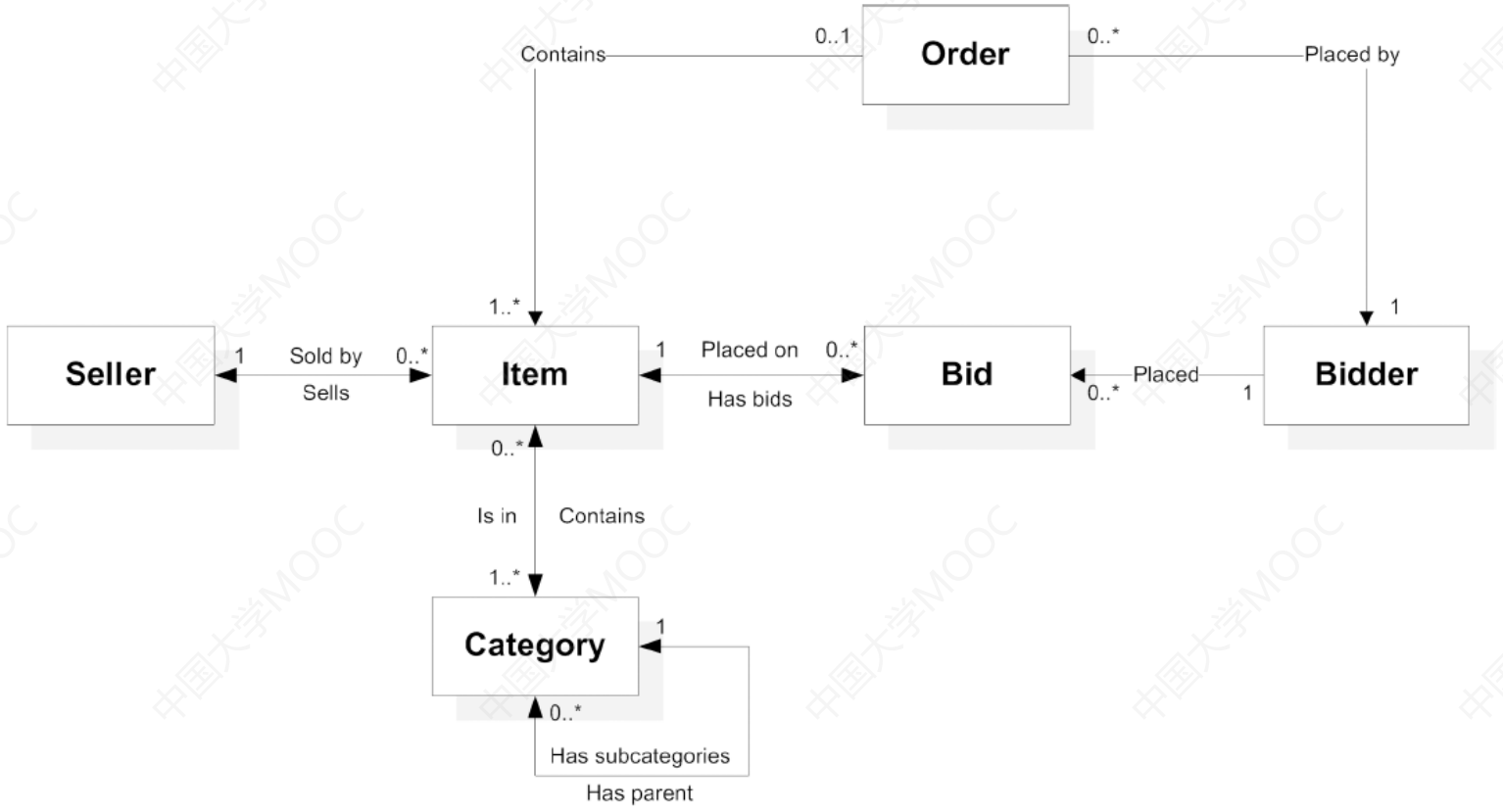
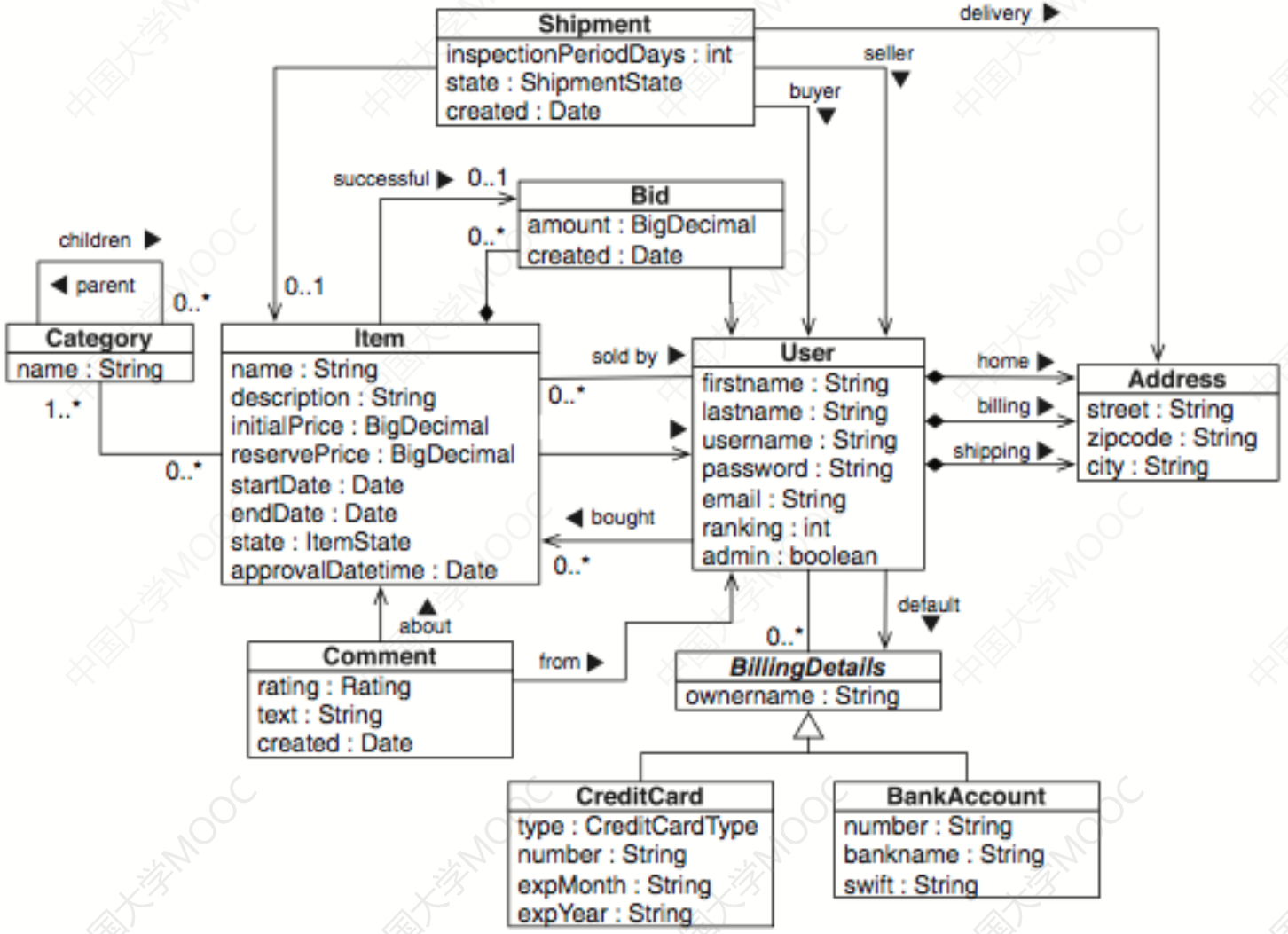


Figure 7.3 The ActionBazaar domain model complete with entities and relationships. Entities are related to one another and the relationship can be one-to-one, one-to-many, many-to-one, or many-to-many. Relationships can be either uni- or bidirectional.

1 对象关系映射


Shipment
inspectionPeriodDays : int
state : ShipmentState
created : Date
delivery
seller
buyer
successful
0..1
Bid
amount : BigDecimal
created : Date
0..*
children
parent
0..*
0..1
Category
name : String
Item
name : String
description : String
initialPrice : BigDecimal
reservePrice : BigDecimal
startDate : Date
endDate : Date
state : ItemState
approvalDatetime : Date
sold by
0..*
User
firstname : String
lastname : String
username : String
password : String
email : String
ranking : int
admin : boolean
home
billing
shipping
Address
street : String
zipcode : String
city : String
1..*
0..*
bought
0..*
about
Comment
rating : Rating
text : String
created : Date
from
default
0..*
BillingDetails
ownername : String
CreditCard
type : CreditCardType
number : String
expMonth : String
expYear : String
BankAccount
number : String
bankname : String
swift : String

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。其对数据库结构提供了较为完整的封装
    - 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-关系模型持久化框架。
    - 采用XML和标注把数据库记录映射成为Java对象，把JDBC、参数设置和结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化 。

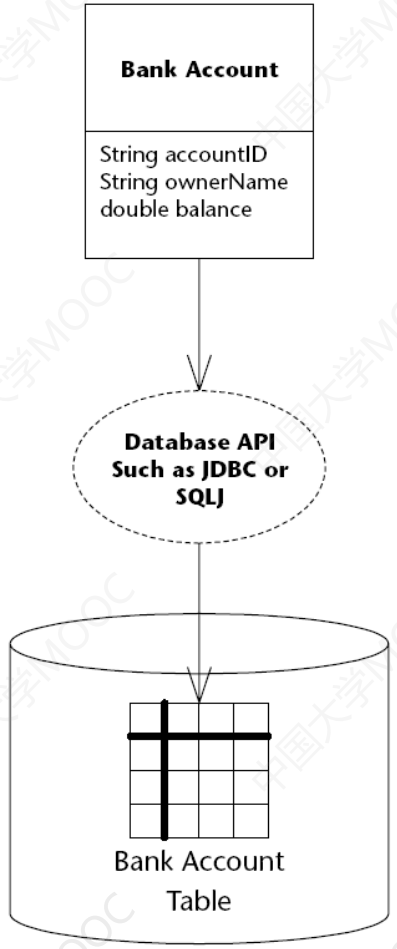


Figure 6.1 Object-relational mapping.

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

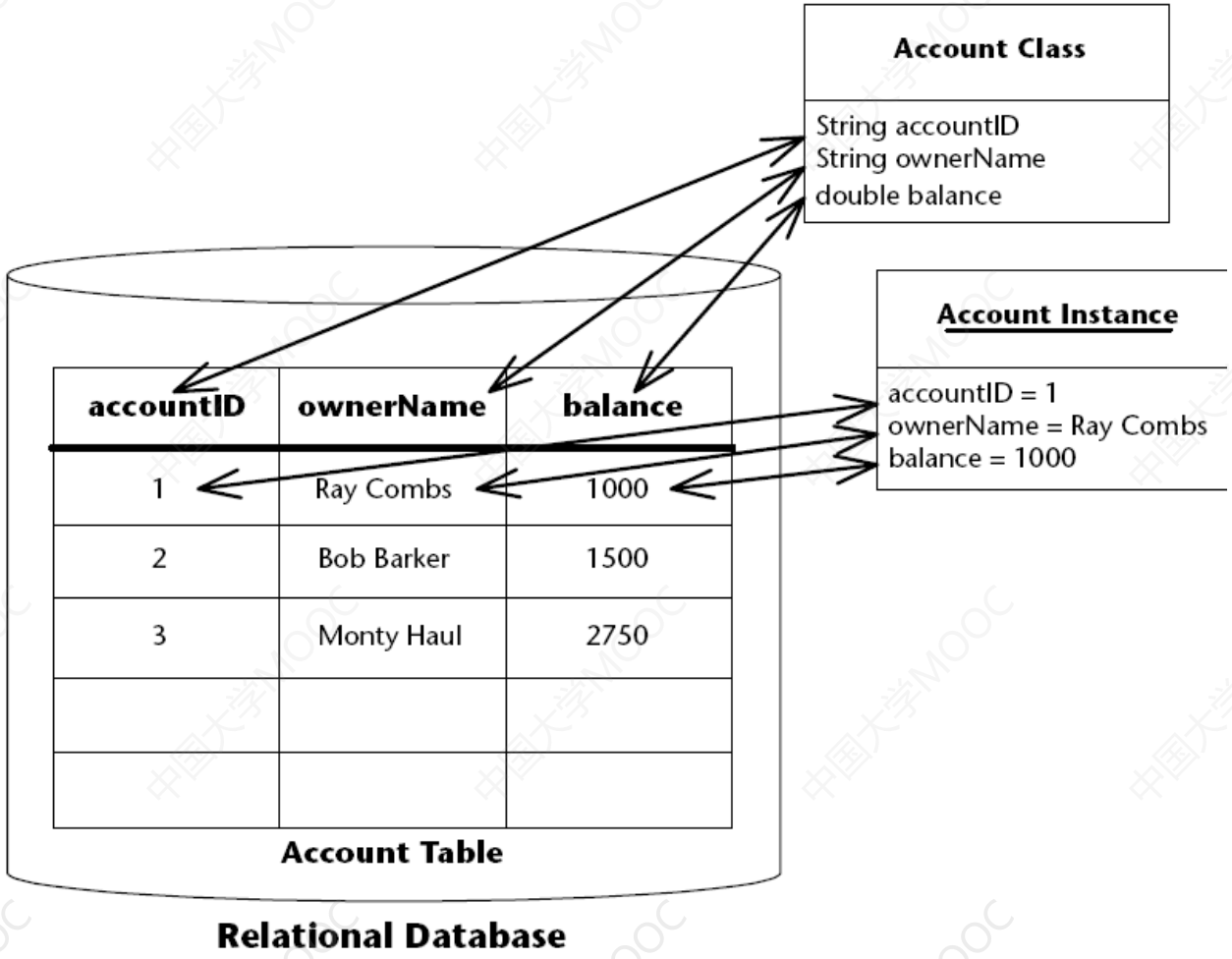


Figure 6.2 An example of object-relational mapping.

1 对象关系映射

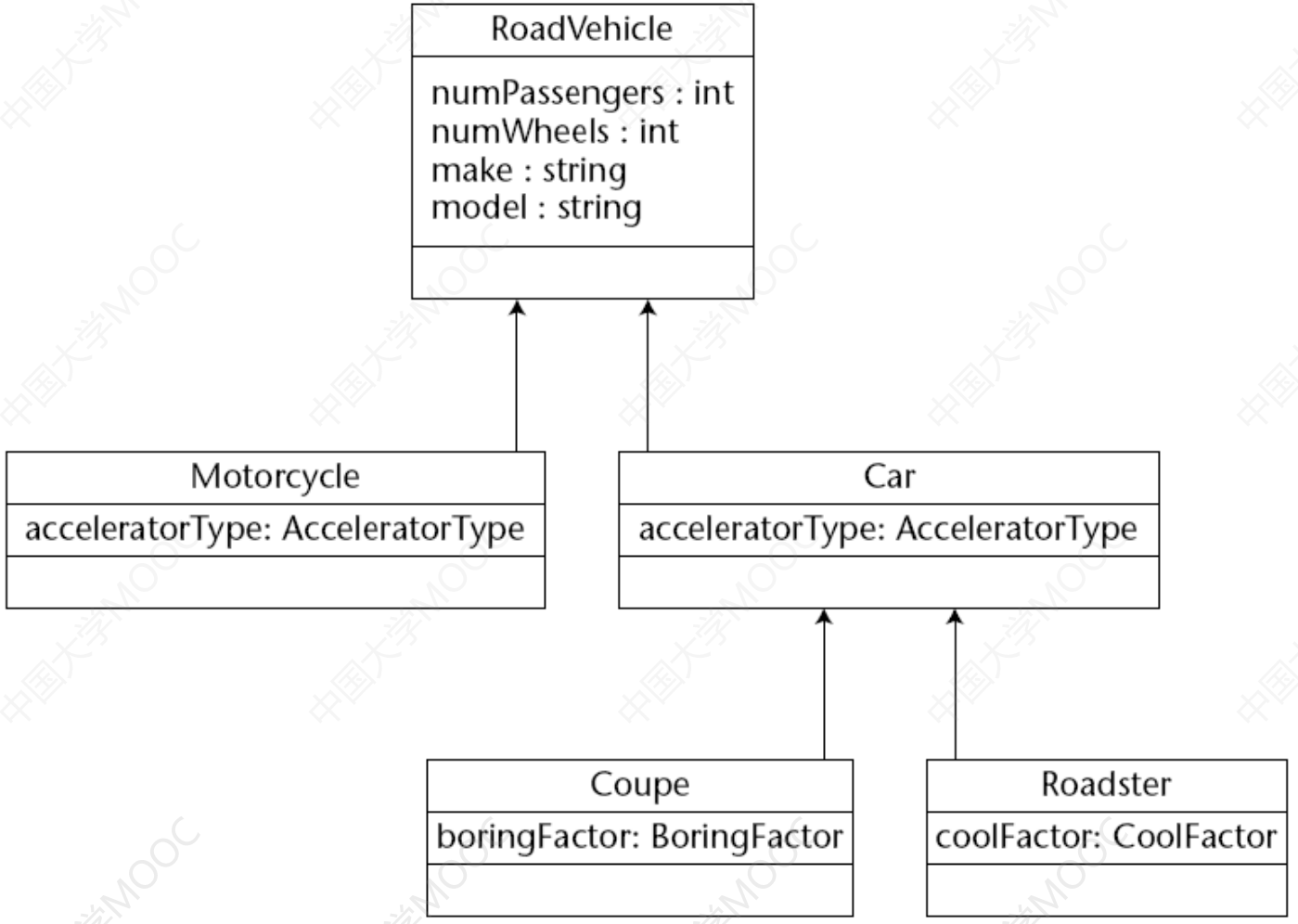


Figure 9.1 UML object model.

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

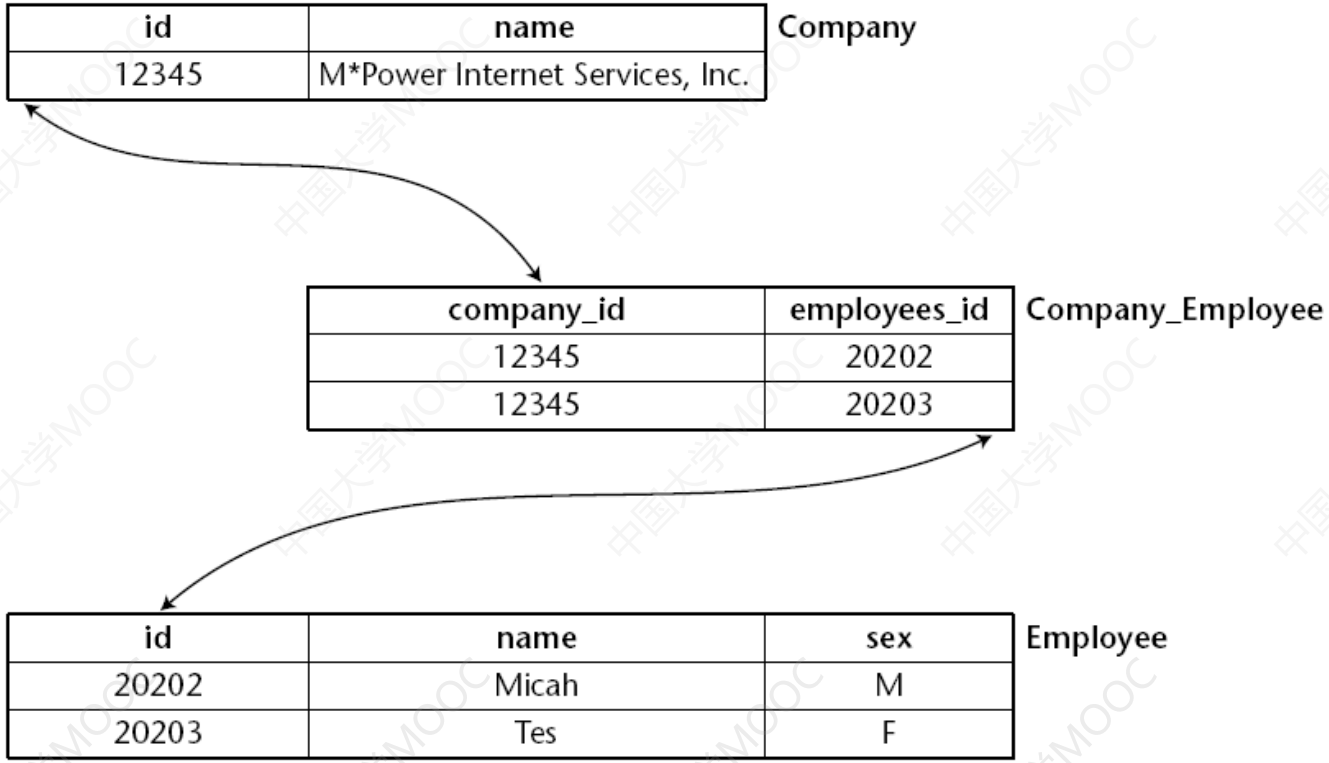
OrderPK	OrderName	Shipment ForeignPK
12345	Software Order	10101

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
12345	M*Power Internet Services, Inc.

Company

company_id	employees_id
12345	20202
12345	20203

Company_Employee

id	name	sex
20202	Micah	M
20203	Tes	F

Employee

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

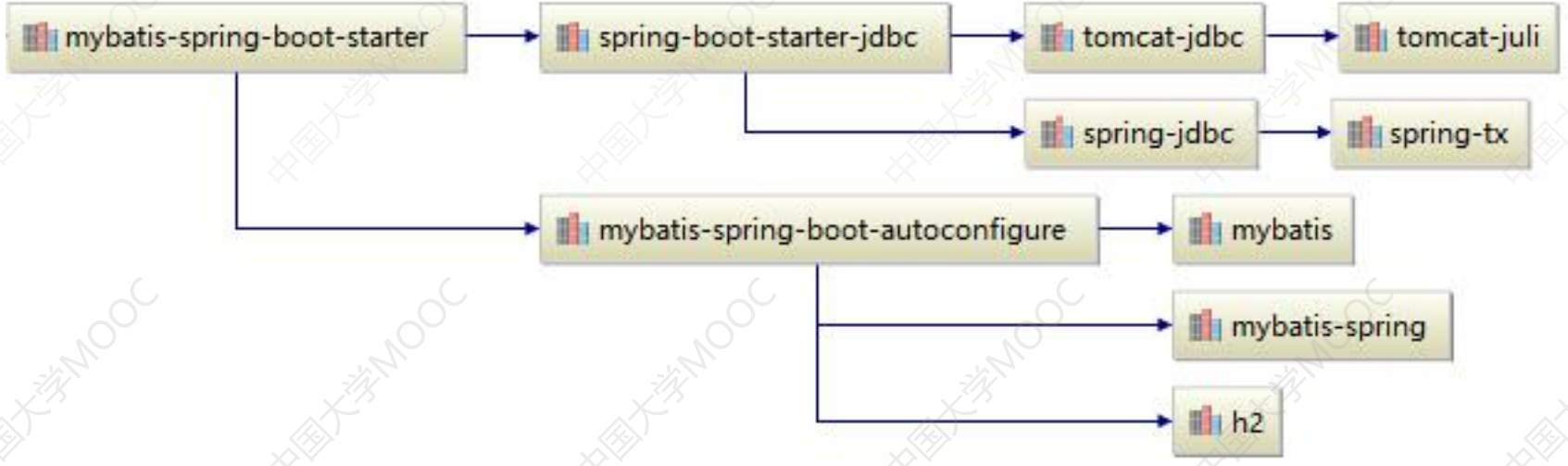
CoursePK	CourseName
20202	EJB for Architects

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

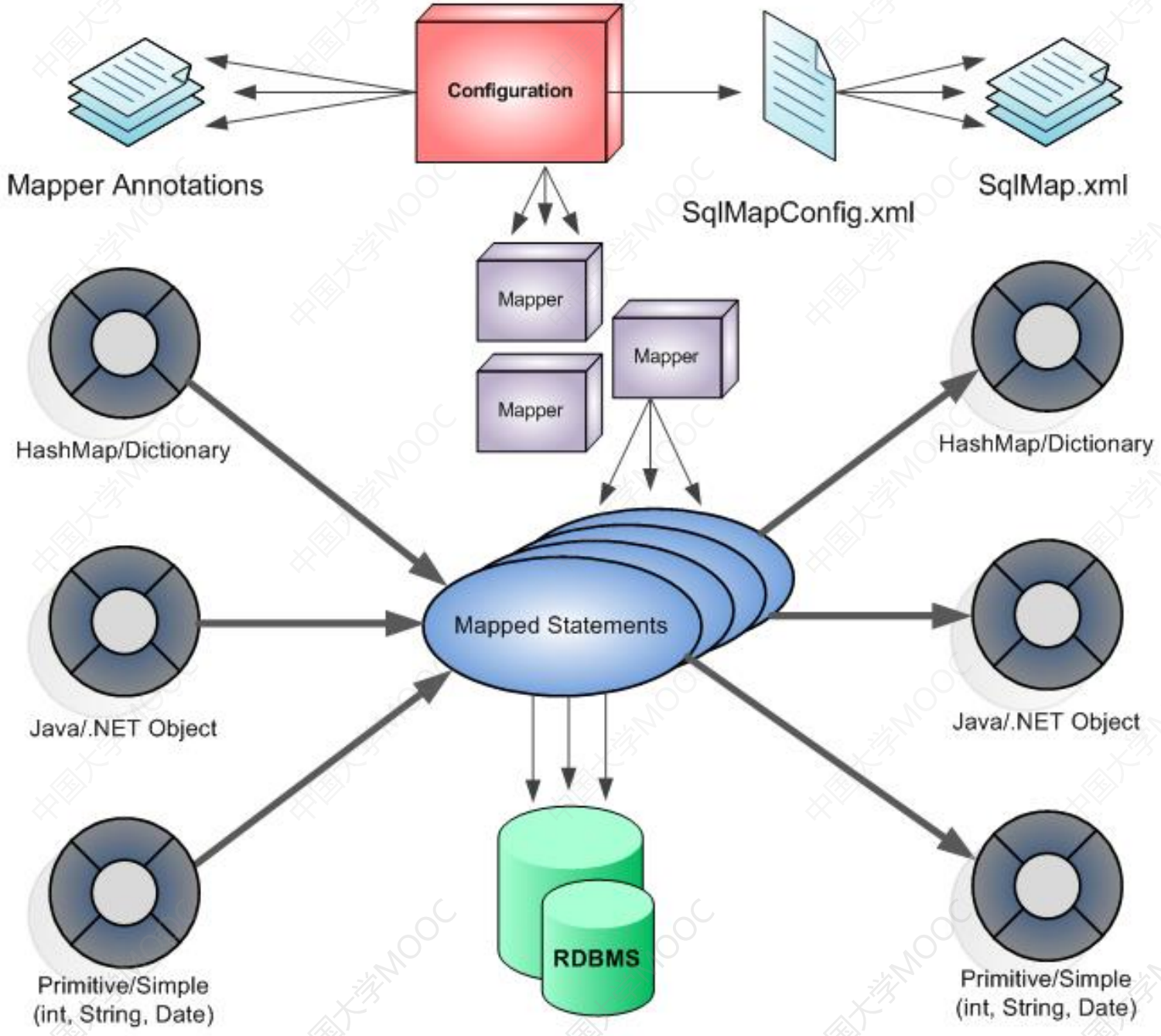
2 MyBatis

- Maven安装包

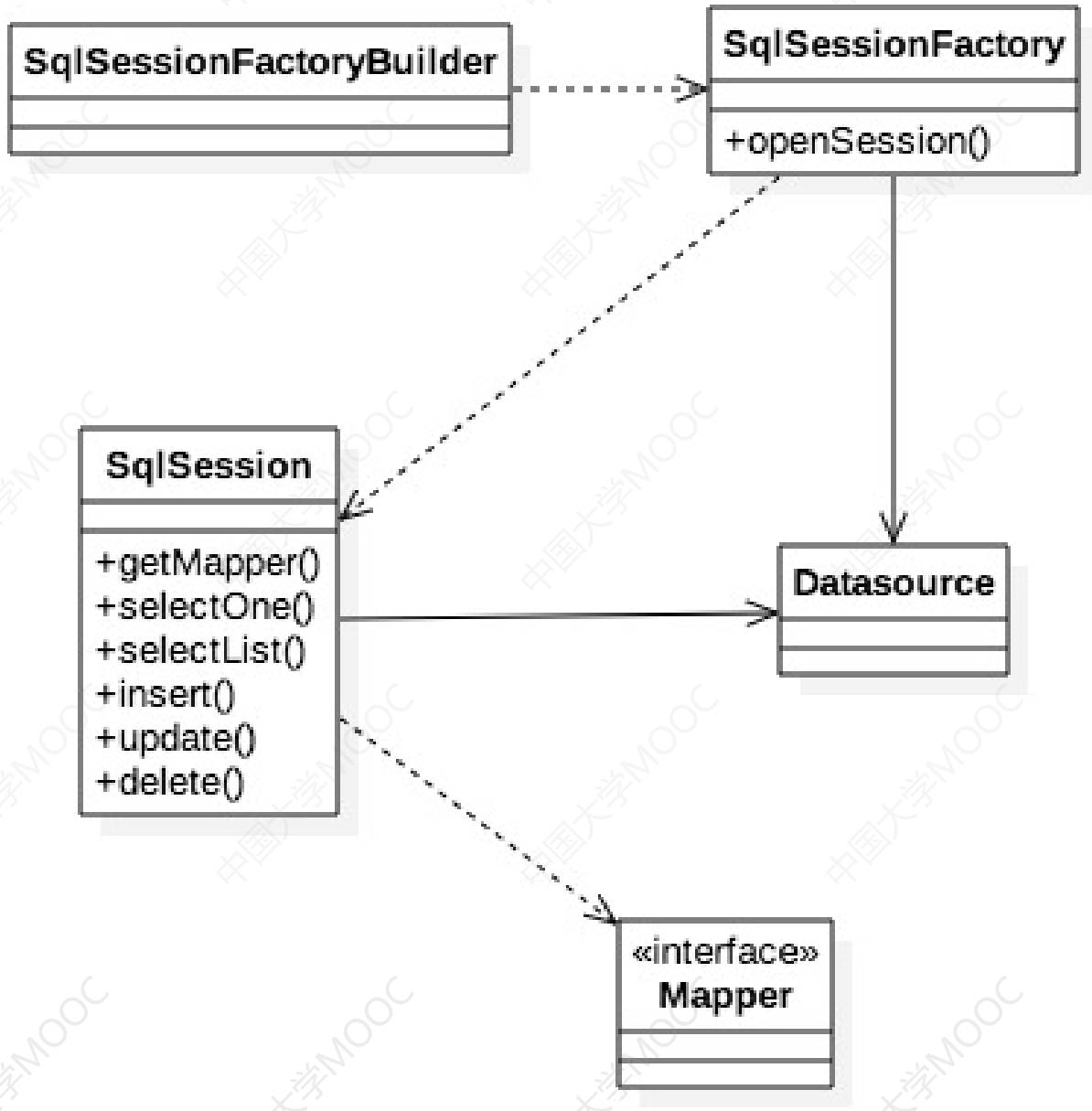
```
<dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-starter</artifactId>
</dependency>
```



2 MyBatis


Configuration
Mapper Annotations
SqlMapConfig.xml
SqlMap.xml
Mapper
Mapper
Mapper
HashMap/Dictionary
HashMap/Dictionary
Mapped Statements
Java/.NET Object
Java/.NET Object
RDBMS
Primitive/Simple
(int, String, Date)
Primitive/Simple
(int, String, Date)

2 MyBatis


SqlSessionFactoryBuilder
SqlSessionFactory
+openSession()
SqlSession
+getMapper()
+selectOne()
+selectList()
+insert()
+update()
+delete()
Datasource
«interface»
Mapper

2 MyBatis

- 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，以备将来注入带Bean中

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

```
<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。

```
<!-- 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

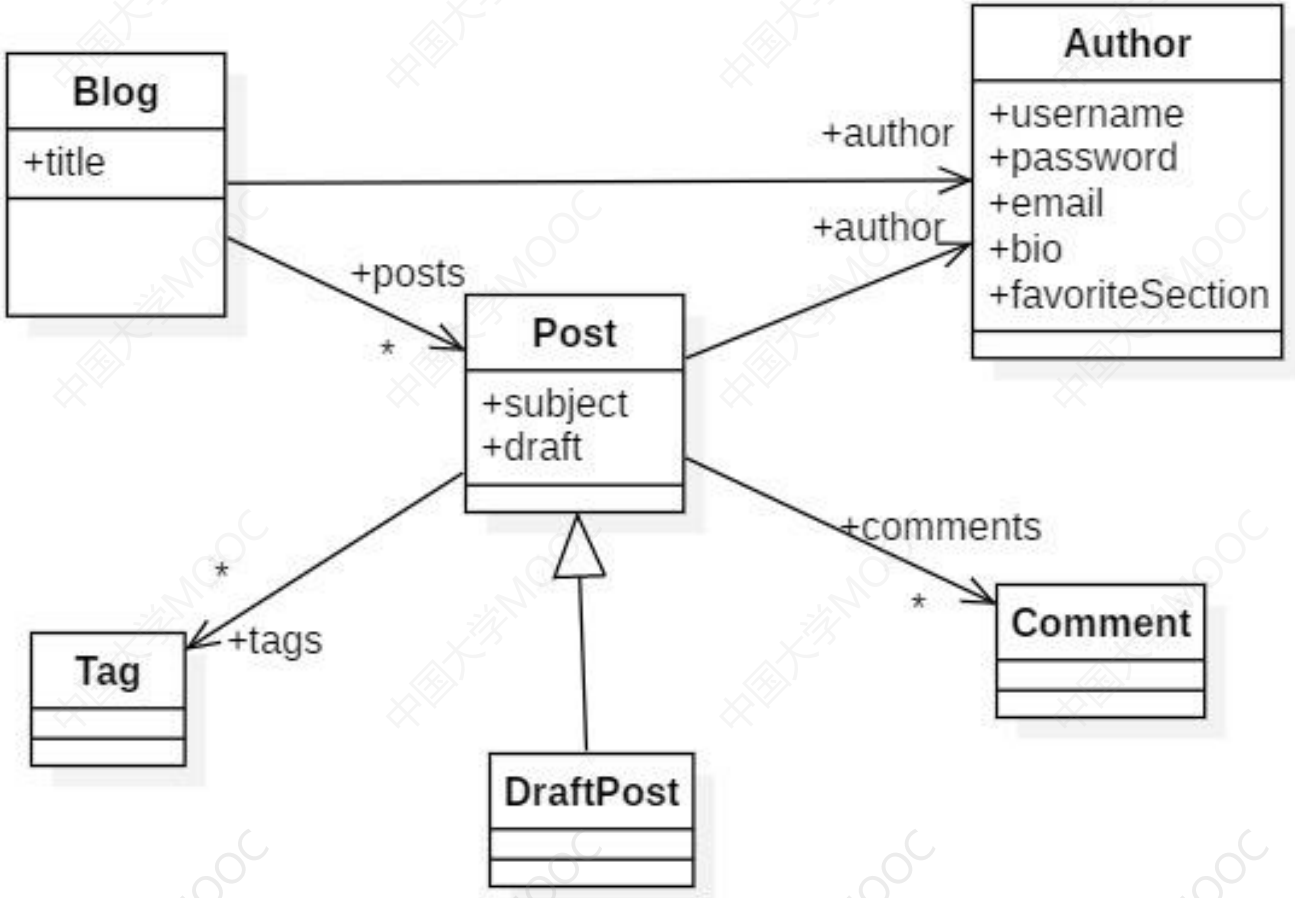
```
<select id="selectUsers" resultType="User">
        select user_id as "id",
                user_name as "userName",
                hashed_password as "hashedPassword"
        from some_table where id = #{id}
</select>
```

3. MyBatis的映射标记

- ResultMap
  - 当然也可以使用ResultMap

```
<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



```
<!-- 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

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

```
<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

```
<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

```
<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

```
<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

```
<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)

```
<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)

```
<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)

```
<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

```
<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>
```

```
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

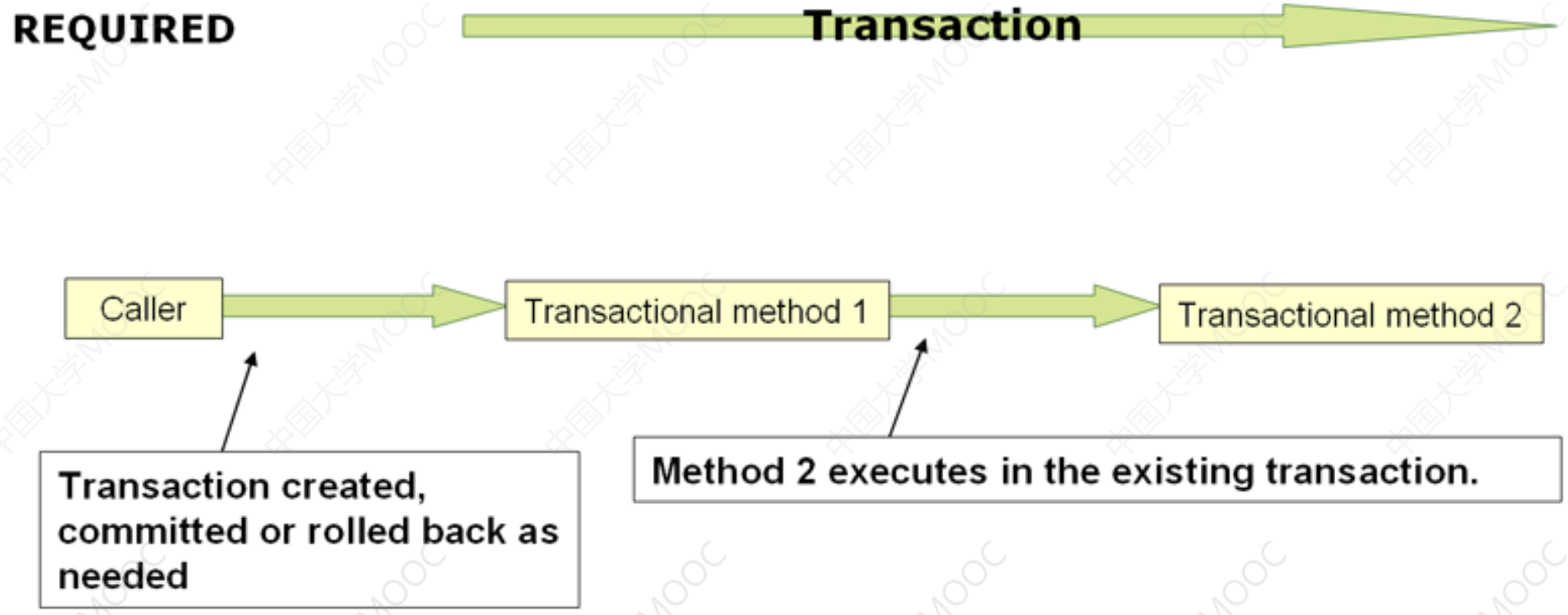
4. MyBatis的事务

- 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。

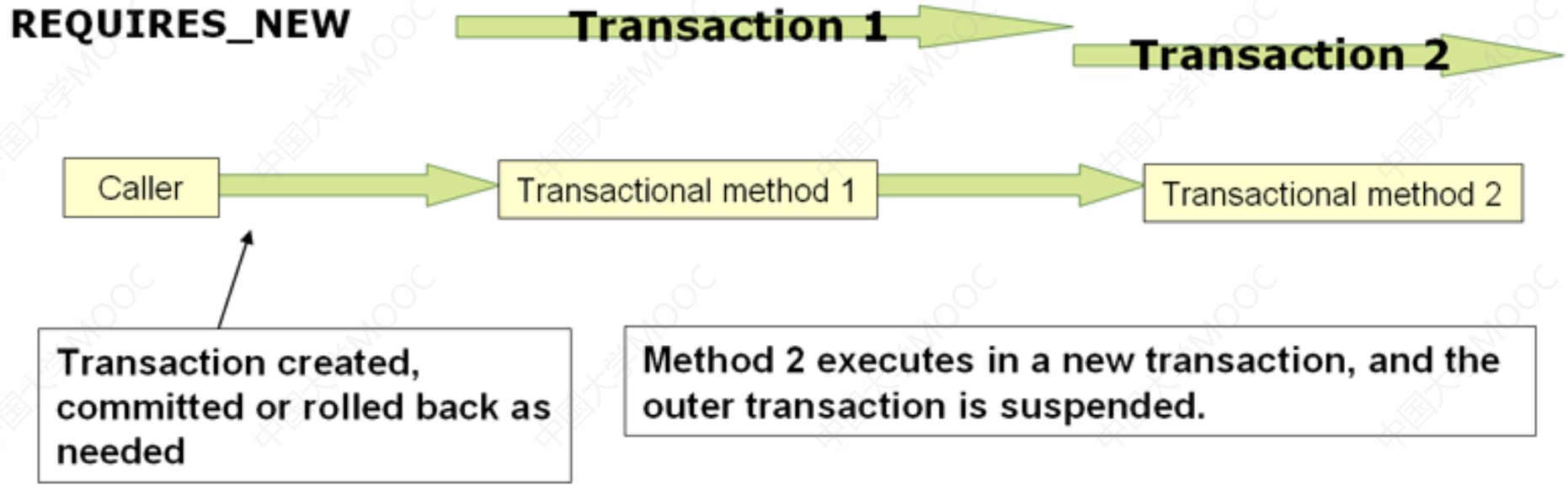
4. MyBatis的事务

- 事务传播设置（propagation）



4. MyBatis的事务

- 事务传播设置（propagation）



4. MyBatis的事务

- 事务并发的问题
  - 脏读：事务A读取了事务B更新的数据，然后B回滚操作，那么A读取到的数据是脏数据
  - 不可重复读：事务 A 多次读取同一数据，事务 B 在事务A多次读取的过程中，对数据作了更新并提交，导致事务A多次读取同一数据时，结果不一致。
  - 幻读：系统管理员A统计数据库中所有订单的数量，但是用户就在这个时候新提交了一个订单，当系统管理员A统计后发现还有一条订单没有统计，就好像发生了幻觉一样，这就叫幻读。

4. MyBatis的事务

- 事务隔离级别（isolation）

隔离级别	描述
Isolation. DEFAULT	使用数据库本身使用的隔离级别，MySQL（READ_COMMITED）
Isolation. READ_UNCOMITTED	读未提交（脏读）最低的隔离级别，一切皆有可能。
Isolation. READ_COMMITED	读已提交，有幻读以及不可重复读风险。
Isolation. REPEATABLE_READ	可重复读，解决不可重复读的隔离级别，但还是有幻读风险。
Isolation. SERLALIZABLE	串行化，最高的事务隔离级别，不管多少事务，挨个运行完一个事务的所有子事务之后才可以执行另外一个事务里面的所有子事务，这样就解决了脏读、不可重复读和幻读的问题了

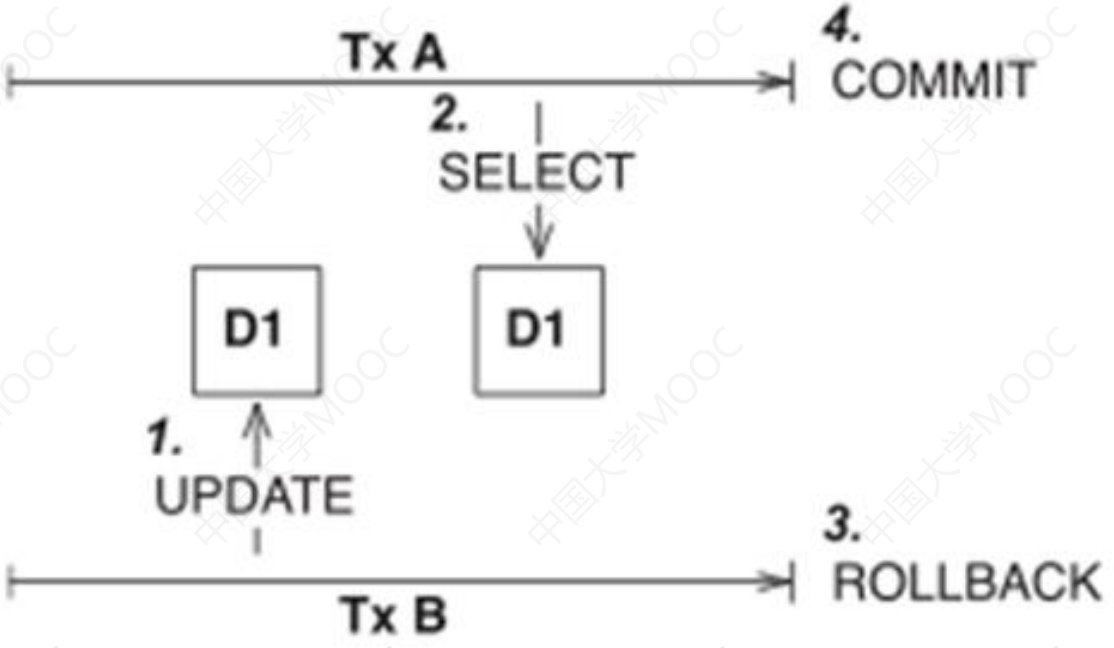
4. MyBatis的事务

- 事务隔离级别（isolation）

事务隔离级别	脏读	不可重复读	幻读
读未提交（read-uncommitted）	是	是	是
不可重复读（read-committed）	否	是	是
可重复读（repeatable-read）	否	否	是
串行化（serializable）	否	否	否

4. MyBatis的事务

- 读未提交-脏读问题



4. MyBatis的事务

- 读未提交（read-uncommitted）-脏读问题

```
mysql> set session transaction
Query OK, 0 rows affected (0.0

mysql> start transaction;
Query OK, 0 rows affected (0.0

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     450 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

```
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set balance = balance - 50 where id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------+--------+---------+
| id   | name   | balance |
+------+--------+---------+
|    1 | lilei  |     400 |
|    2 | hanmei |   16000 |
|    3 | lucy   |    2400 |
+------+--------+---------+
3 rows in set (0.00 sec)
```

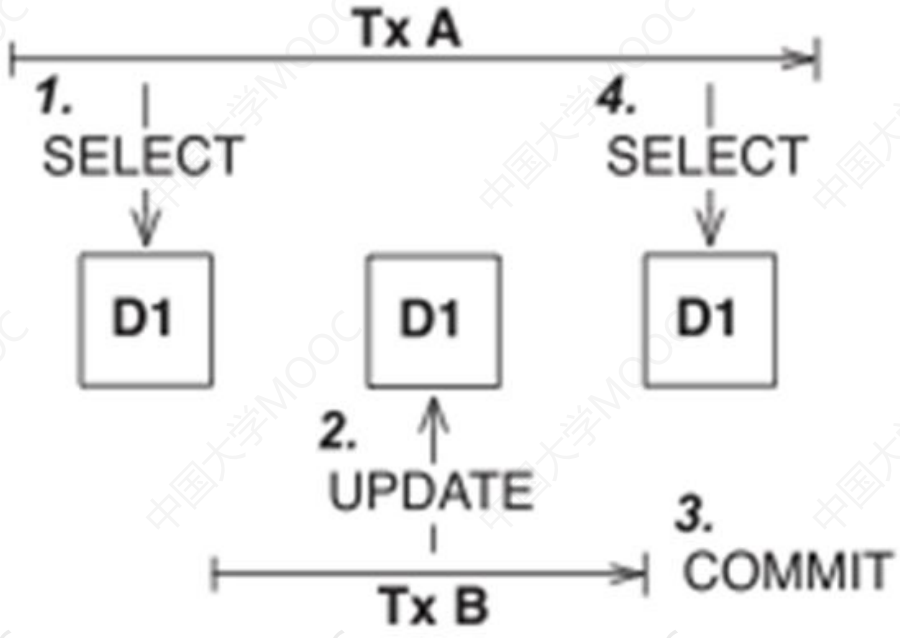
客户端B

4. MyBatis的事务

- 读未提交-脏读问题

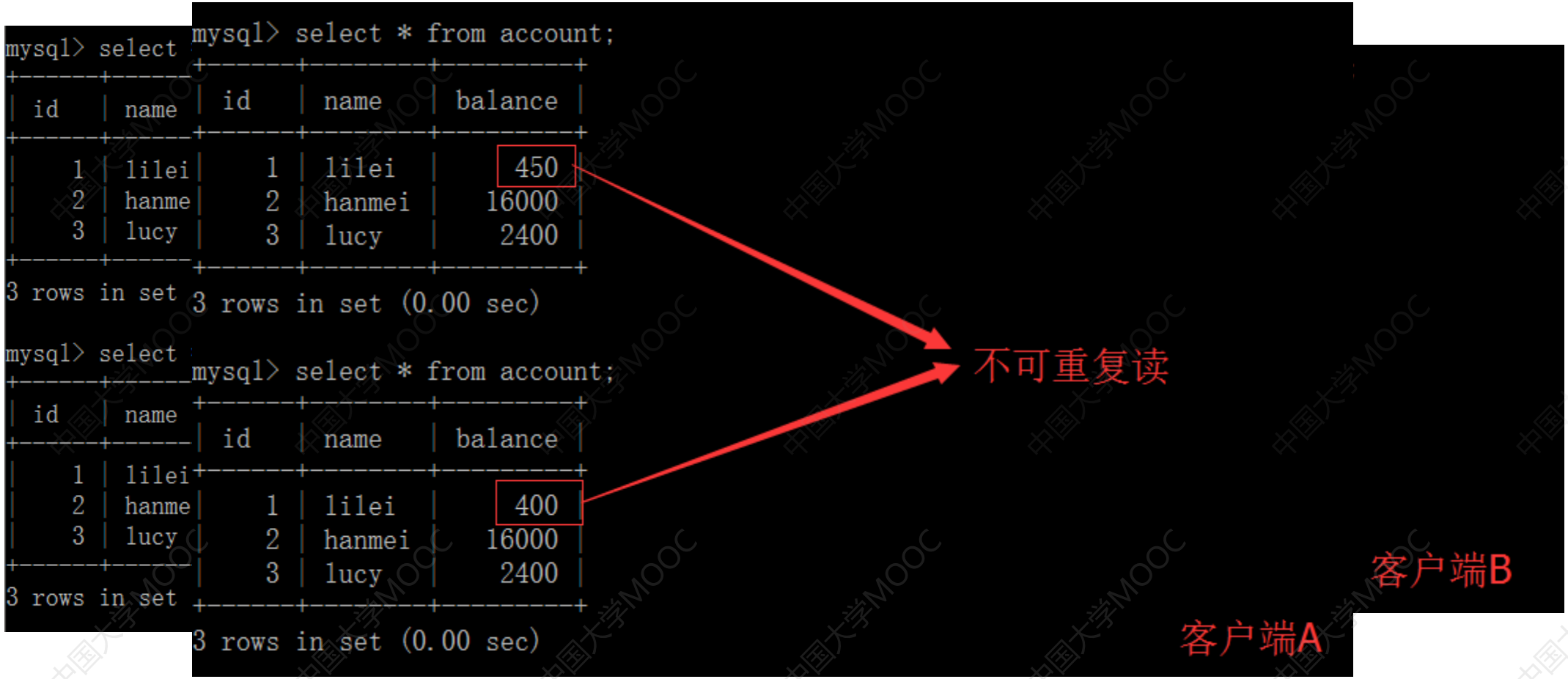
4. MyBatis的事务

- 不可重复读



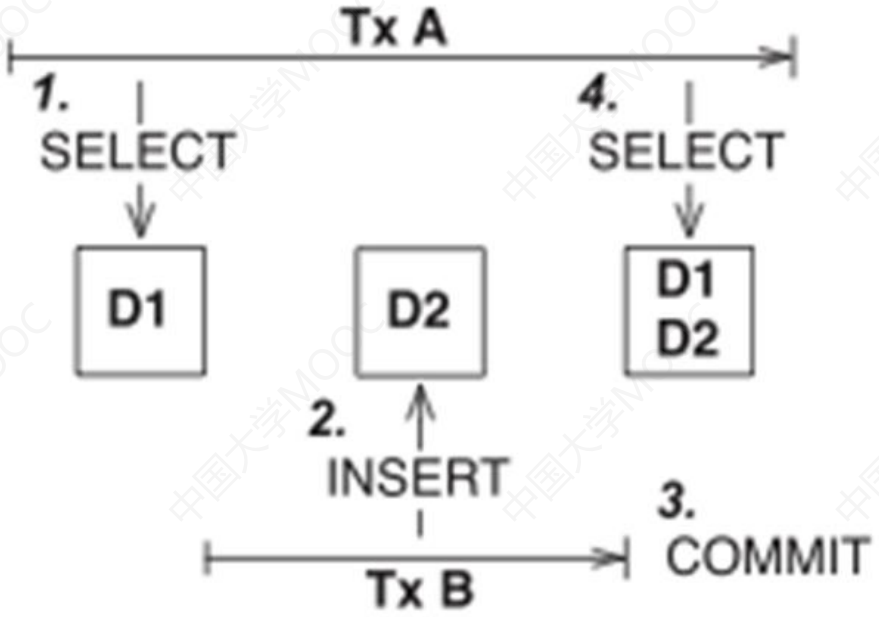
4. MyBatis的事务

- 不可重复读



4. MyBatis的事务

- 可重复读-幻读问题



4. MyBatis的事务

- 可重复读-幻读问题

```
mysql> start  transact
Query OK, 0 rows affec

mysql> insert into acc
Query OK, 1 row affect

mysql> commit;
Query OK, 0 rows affec
```

```
mysql> select * from account;
+----+--------+---------+
| id | name   | balance |
+----+--------+---------+
|  1 | lilei  |     300 |
|  2 | hanmei |   16000 |
|  3 | lucy   |    2400 |
+----+--------+---------+
3 rows in set

mysql> select * from account;
+----+--------+---------+
| id | name   | balance |
+----+--------+---------+
|  1 | lilei  |     300 |
|  2 | hanmei |   16000 |
|  3 | lucy   |    2400 |
+----+--------+---------+
3 rows in set
```

客户端A

4. MyBatis的事务

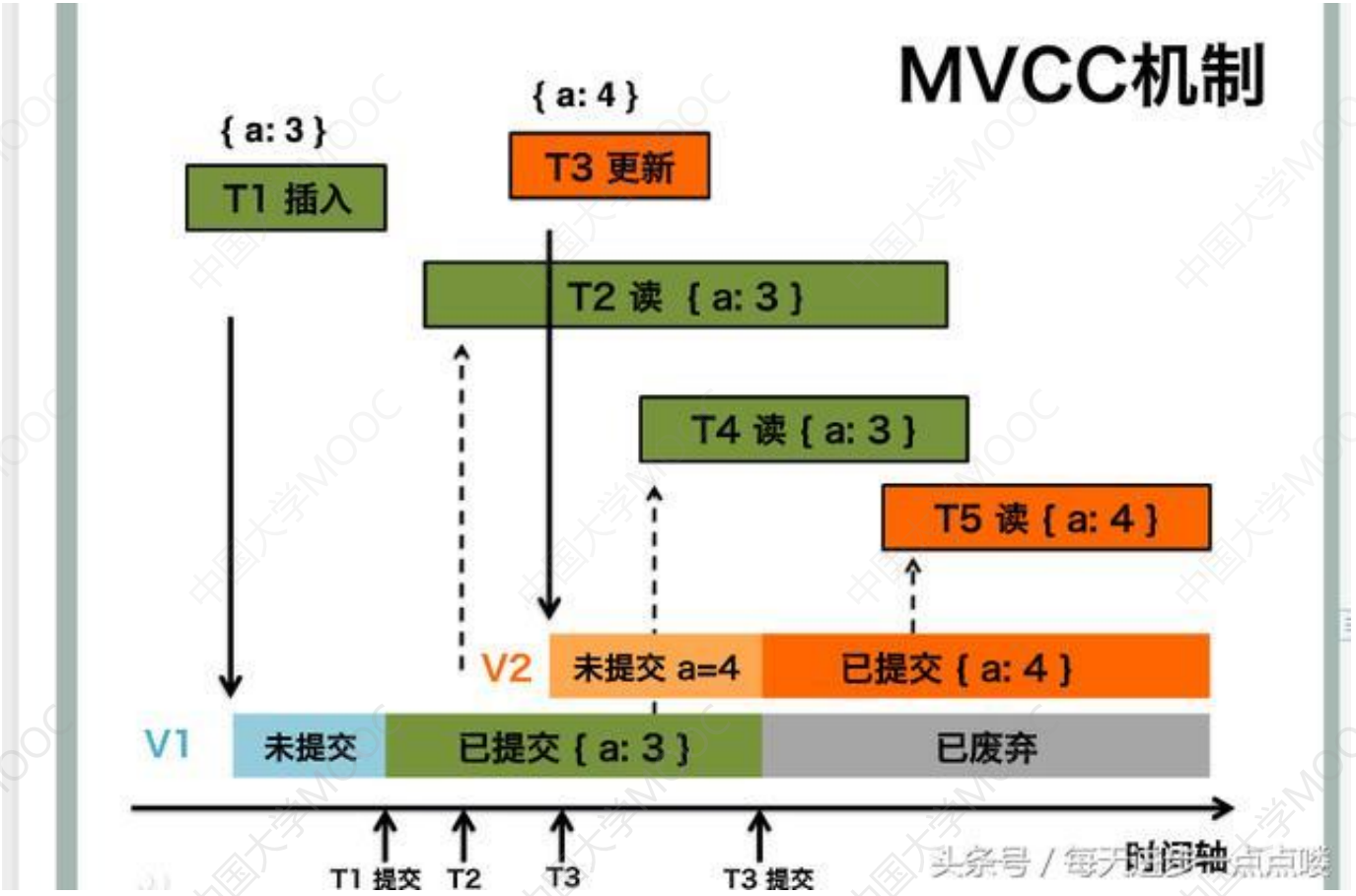
- 实现数据库的并发访问控制的最简单做法是
  - 读的时候不能写（允许多个线程同时读，即共享锁，S锁）
  - 写的时候不能读（一次最多只能有一个线程对同一份数据进行写操作，即排它锁，X锁）。

事务隔离级别	脏读	不可重复读	幻读	锁的方式
读未提交（read-uncommitted）	是	是	是	无锁
不可重复读（read-committed）	否	是	是	对记录加写锁
可重复读（repeatable-read）	否	否	是	对记录加读写锁
串行化（serializable）	否	否	否	对表加读锁

4. MyBatis的事务

- MVCC：Multi-Version Concurrent Control 多版本并发控制
  - MVCC通过保存数据在某个时间点的快照来实现读的并发。这意味着一个事务无论运行多长时间，在同一个事务里能够看到数据一致的视图。根据事务开始的时间不同，同时也意味着在同一个时刻不同事务看到的相同表里的数据可能是不同的。
  - 每行数据都存在一个版本，每次数据更新时都更新该版本。
  - 修改时Copy出当前版本随意修改，各个事务之间无干扰。
  - 保存时比较版本号，如果成功（commit），则覆盖原记录；失败则放弃copy（rollback）

4. MyBatis的事务



4. MyBatis的事务

- MVCC下InnoDB的增删查改的工作原理
  - 插入数据（insert）:记录的版本号即当前事务的版本号

```
insert into testmvcc values(1,"test");
```

id	name	create version	delete version
1	test	1	

4. MyBatis的事务

- MVCC下InnoDB的增删查改的工作原理
  - 在更新操作的时候，采用的是先标记旧的那行记录为已删除，并且删除版本号是事务版本号，然后插入一行新的记录的方式。

update table set name= 'new_value' where id=1;

id	name	create version	delete version
1	test	1	2
1	new_value	2	

4. MyBatis的事务

- MVCC下InnoDB的增删查改的工作原理
  - 删除操作的时候，就把事务版本号作为删除版本号。

delete from table where id=1;

id	name	create version	delete version
1	new_value	2	3

4. MyBatis的事务

- MVCC下InnoDB的增删查改的工作原理
  - 查询操作，符合以下两个条件的记录才能被事务查询出来：
    - 删除版本号未指定或者大于当前事务版本号，即查询事务开启后确保读取的行未被删除。(即上述事务id为2的事务查询时，依然能读取到事务id为3所删除的数据行)
    - 创建版本号 小于或者等于 当前事务版本号 ，就是说记录创建是在当前事务中（等于的情况）或者在当前事务启动之前的其他事物进行的insert。

    - （即事务id为2的事务只能读取到create version<=2的已提交的事务的数据集）

4. MyBatis的事务

- MVCC下InnoDB的增删查改的工作原理
  - 查询操作，符合以下两个条件的记录才能被事务查询出来：
    - 删除版本号未指定或者大于当前事务版本号，即查询事务开启后确保读取的行未被删除。(即上述事务id为2的事务查询时，依然能读取到事务id为3所删除的数据行)
    - 创建版本号 小于或者等于 当前事务版本号 ，就是说记录创建是在当前事务中（等于的情况）或者在当前事务启动之前的其他事物进行的insert。

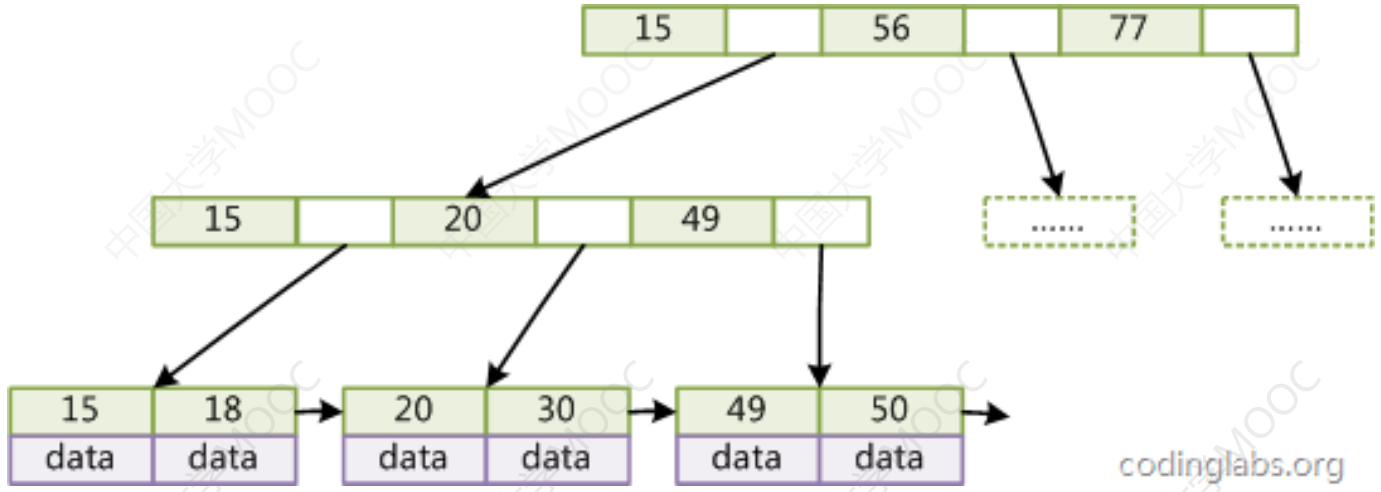
    - （即事务id为2的事务只能读取到create version<=2的已提交的事务的数据集）

4. MyBatis的事务

- MVCC通过保存数据在某个时间点的快照来实现的。
  - 一个事务无论运行多长时间，在同一个事务里能够看到数据一致的视图。
  - 根据事务开始的时间不同，同时也意味着在同一个时刻不同事务看到的相同表里的数据可能是不同的。

5. 查询优化

- 索引
  - 一般的应用系统，读写比例在10:1左右，而且插入操作和一般的更新操作很少出现性能问题，在生产环境中，遇到最多的，也是最容易出问题的，还是一些复杂的查询操作，因此对查询语句的优化显然是重中之重。
  - 索引的目的在于提高查询效率



5. 查询优化

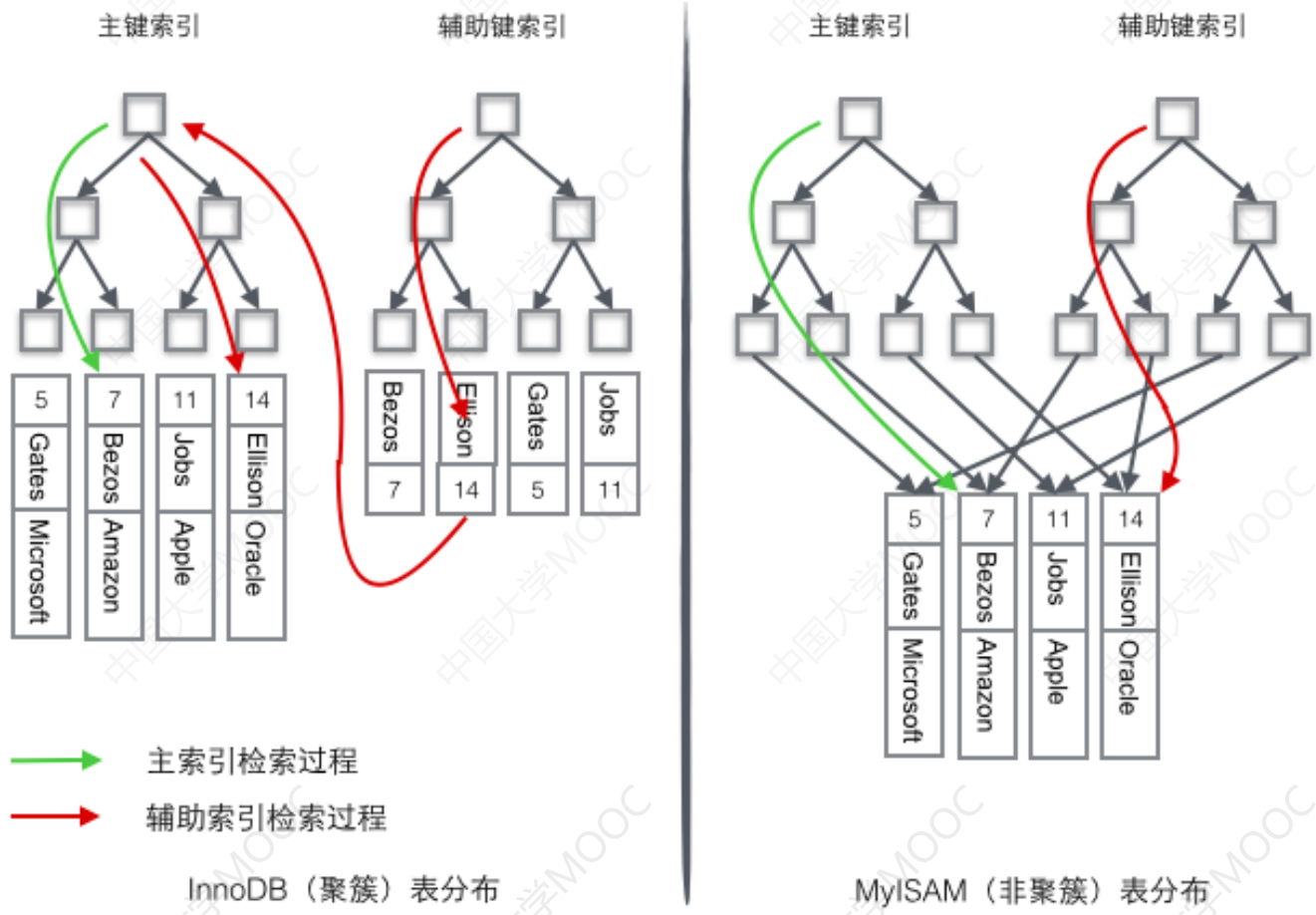
- MySQL的索引分类
  - 主键索引：primary key ： 加速查找+约束（不为空且唯一）
  - 唯一索引：unique：加速查找+约束 （唯一）
  - 联合索引
    - -primary key(id,name):联合主键索引
    - -unique(id,name):联合唯一索引
    - -index(id,name):联合普通索引
  - 全文索引fulltext :用于搜索很长一篇文章的时候，效果最好。

5. 查询优化

- InnoDB使用的是聚簇索引，将主键组织到一棵B+树中，而行数据就储存在叶子节点上
  - 其中Id作为主索引，Name作为辅助索引

Id	Name	Company
5	Gates	Microsoft
7	Bezos	Amazon
11	Jobs	Apple
14	Ellison	Oracle

5. 查询优化



5. 查询优化

```
select * from table1 where name='zhangsan' and tID > 10000;

select * from table1 where tID > 10000 and name='zhangsan';
```

MySQL中有一个“查询分析优化器”，它可以计算出where子句中的搜索条件并确定哪个索引能缩小表扫描的搜索空间，实现自动优化。

5. 查询优化

- 应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描

```
select id from t where num is null
```

- 应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。优化器将无法通过索引来确定将要命中的行数,因此需要搜索该表的所有行。
- 应尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：

```
select id from t where num=10 or num=20

select id from t where num=10
union all
select id from t where num=20
```

5. 查询优化

- in 和 not in 也要慎用，因为IN会使系统无法使用索引,而只能直接搜索表中的数据。

```
select id from t where num in(1,2,3)
```

- 尽量避免在索引过的字符数据中，使用非打头字母搜索。这也使得引擎无法利用索引

```
SELECT * FROM T1 WHERE NAME LIKE '%L%'
SELECT * FROM T1 WHERE SUBSTING(NAME,2,1)='L'
SELECT * FROM T1 WHERE NAME LIKE 'L%'
```

5. 查询优化

- 必要时强制查询优化器使用某个索引，如在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：

  select id from t where num=@num

  select id from t with(index(索引名)) where num=@num

5. 查询优化

- 应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。

```
SELECT * FROM T1 WHERE F1/2=100
SELECT * FROM T1 WHERE F1=100*2

SELECT * FROM RECORD WHERE SUBSTRING(CARD_NO,1,4)='5378'
SELECT * FROM RECORD WHERE CARD_NO LIKE '5378%'
```

任何对列的操作都将导致表扫描，它包括数据库函数、计算表达式等等，查询时要尽可能将操作移至等号右边。

5. 查询优化

- Explain优化查询检测

```
mysql> EXPLAIN SELECT `birday` FROM `user` WHERE `birthday` < "1990/2/2";
-- 结果:
id: 1 select_type: SIMPLE -- 查询类型（简单查询,联合查询,子查询）
table: user -- 显示这一行的数据是关于哪张表的
type: range -- 区间索引（在小于1990/2/2区间的数据),这是重要的列,显示连接使用了何种类型。从最好到最差的连接类型为
system > const > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subquery > range > index > ALL,
const代表一次就命中,ALL代表扫描了全表才确定结果。一般来说,得保证查询至少达到range级别,最好能达到ref。
possible_keys: birthday -- 指出MySQL能使用哪个索引在该表中找到行。如果是空的,没有相关的索引。这时要提高性能,可通过检验WHERE子句,看是否引用某些字段,或者检查字段不是适合索引。
key: birthday -- 实际使用到的索引。如果为NULL,则没有使用索引。如果为primary的话,表示使用了主键。
key_len: 4 -- 最长的索引宽度。如果键是NULL,长度就是NULL。在不损失精确性的情况下,长度越短越好
ref: const -- 显示哪个字段或常数与key一起被使用。
rows: 1 -- 这个数表示mysql要遍历多少数据才能找到,在innodb上是不准确的。
Extra: Using where; Using index -- 执行状态说明,这里可以看到的坏的例子是Using temporary和Using
```

5. 查询优化

- select_type
  - simple 简单select(不使用union或子查询)
  - primary 最外面的select
  - union union中的第二个或后面的select语句
  - dependent union union中的第二个或后面的select语句,取决于外面的查询
  - union result union的结果。
  - subquery 子查询中的第一个select
  - dependent subquery 子查询中的第一个select,取决于外面的查询
  - derived 导出表的select(from子句的子查询)

5. 查询优化

- type详细说明
  - system 表只有一行：system表。这是const连接类型的特殊情况
  - const:表中的一个记录的最大值能够匹配这个查询（索引可以是主键或惟一索引）。因为只有一行,这个值实际就是常数,因为MYSQL先读这个值然后把它当做常数来对待
  - eq_ref:在连接中,MYSQL在查询时,从前面的表中,对每一个记录的联合都从表中读取一个记录,它在查询使用了索引为主键或惟一键的全部时使用
  - ref:这个连接类型只有在查询使用了不是惟一或主键的键或者是这些类型的部分（比如,利用最左边前缀）时发生。对于之前的表的每一个行联合,全部记录都将从表中读出。这个类型严重依赖于根据索引匹配的记录多少—越少越好
  - range:这个连接类型使用索引返回一个范围中的行,比如使用>或<查找东西时发生的情况
  - index: 这个连接类型对前面的表中的每一个记录联合进行完全扫描（比ALL更好,因为索引一般小于表数据）
  - ALL:这个连接类型对于前面的每一个记录联合进行完全扫描,这一般比较糟糕,应该尽量避免

5. 查询优化

- Extra详细说明
  - Distinct:一旦MYSQL找到了与行相联合匹配的行,就不再搜索了
  - Not exists: MYSQL优化了LEFT JOIN,一旦它找到了匹配LEFT JOIN标准的行,就不再搜索了
  - Range checked for each Record（index map:#）:没有找到理想的索引,因此对于从前面表中来的每一个行组合,MYSQL检查使用哪个索引,并用它来从表中返回行。这是使用索引的最慢的连接之一
  - Using filesort: 看到这个的时候,查询就需要优化了。MYSQL需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行
  - Using index: 列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的,这发生在对表的全部的请求列都是同一个索引的部分的时候
  - Using temporary 看到这个的时候,查询需要优化了。这里,MYSQL需要创建一个临时表来存储结果,这通常发生在对不同的列集进行ORDER BY上,而不是GROUP BY上
  - Where used 使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行,并且连接类型ALL或index,这就会发生,或者是查询有问题不同连接类型的解释（按照效率高低的顺序排序