Spring Data JPA

brief introduction

Haidar Osman
http://scg.unibe.ch/staff/Osman

b
UNIVERSITÄT
BERN

Traditional Style to Connect to DB

```
import java.sql.DriverManager;
import java.sql.SQLException;
public class DBHandler {
    public void saveNewCustomer() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            java.sql.Connection connection = DriverManager
                    .getConnection("jdbc:mysql://serverName/databaseName?user=username&password=password");
            java.sql.PreparedStatement statement = connection.prepareStatement("insert into customer values (?,?,?)");
            statement.setLong(1, 5);
            statement.setString(2, "John");
            statement.setString(3, "Do");
            statement.executeUpdate();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Load the driver

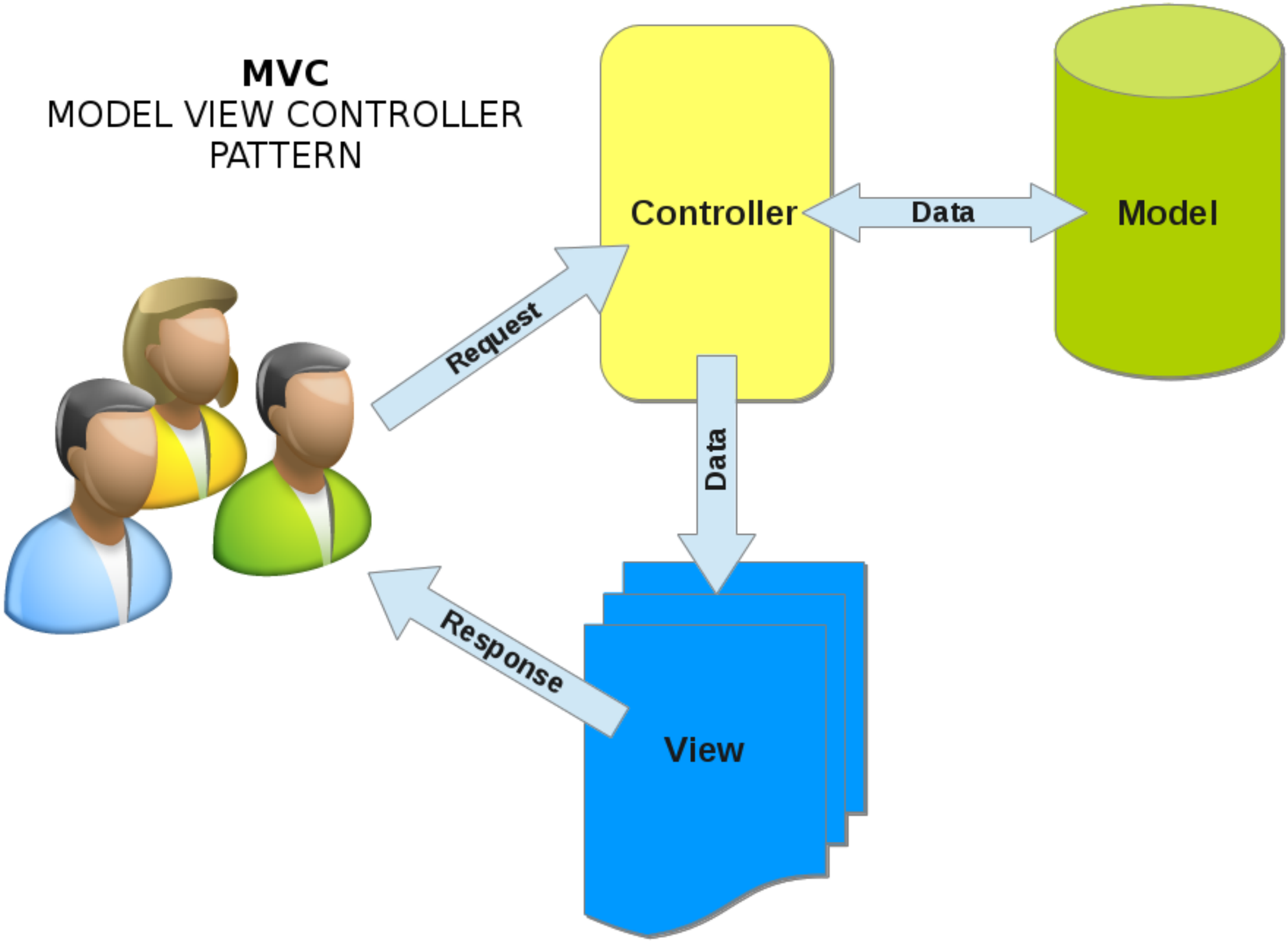
Create a connection

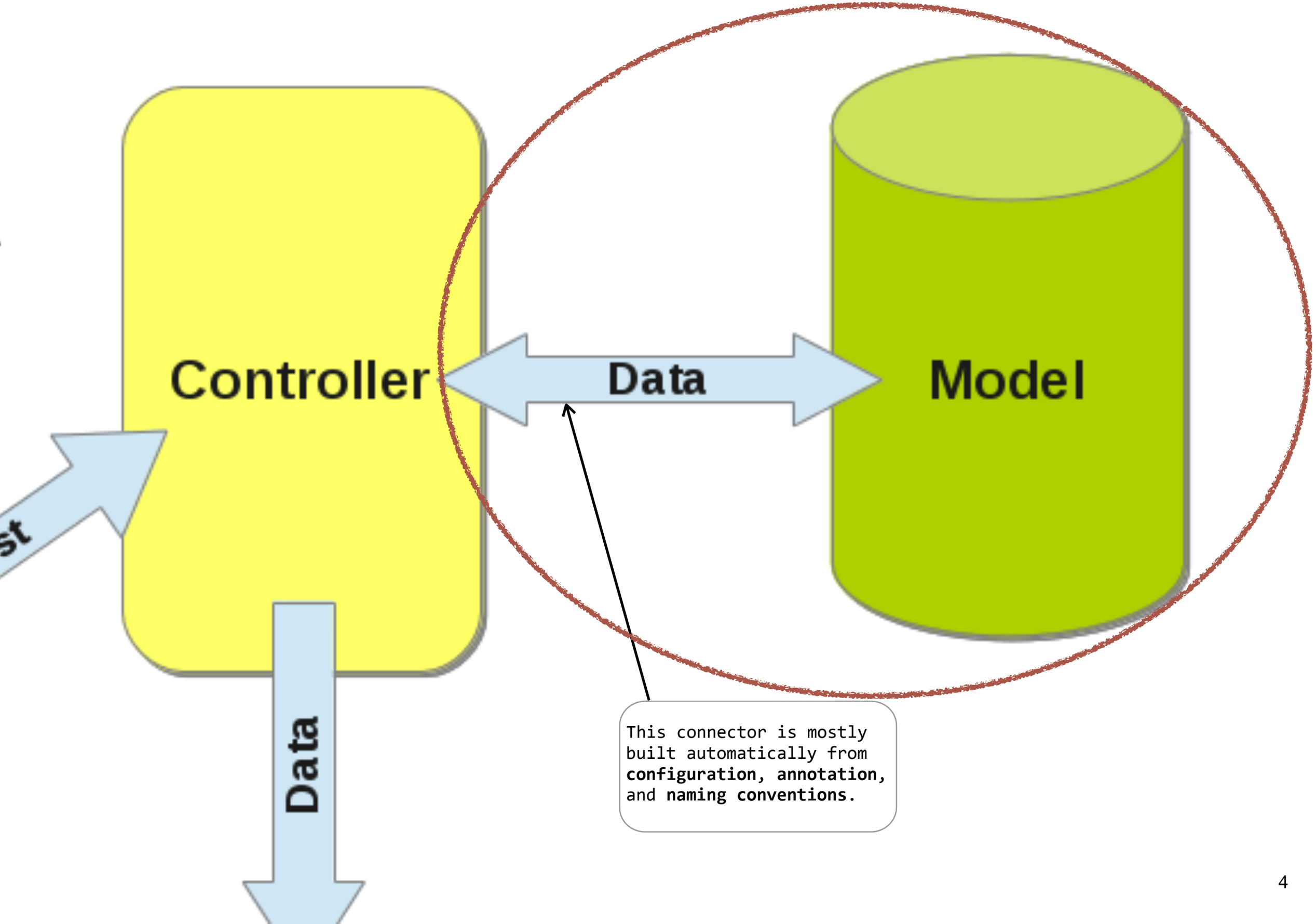
Create an SQL statement

Fill in the statement Parameters

Execute the statement

MVC
MODEL VIEW CONTROLLER
PATTERN




Controller
Data
Model
st
Data
This connector is mostly built automatically from **configuration, annotation,** and **naming conventions.**

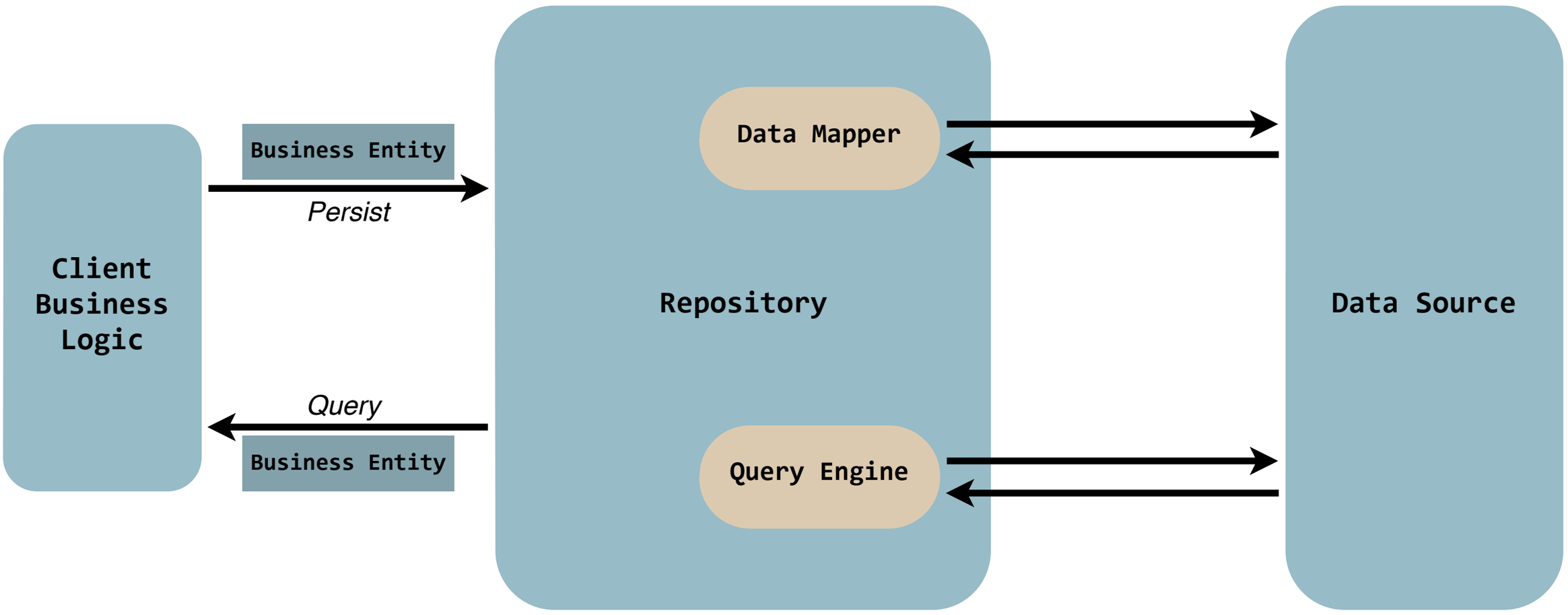


diagram from "*Patterns of Enterprise Application Architecture*", by *Martin Fowler*

Spring Style to Connect to DB (1)

Business Logic

```
@Entity
public class User {
   @Id
   @GeneratedValue
   private Long id;

   private String firstName;
   private String lastName;
   private String email;

   @OneToOne
   private Address address;
.
.
}
```

```
@Entity
public class Address {
   @Id
    @GeneratedValue
   private Long id;
   private String street;
.
.
}
```

Repository

```
public interface UserDao extends
                    CrudRepository<User,Long> {

}
```

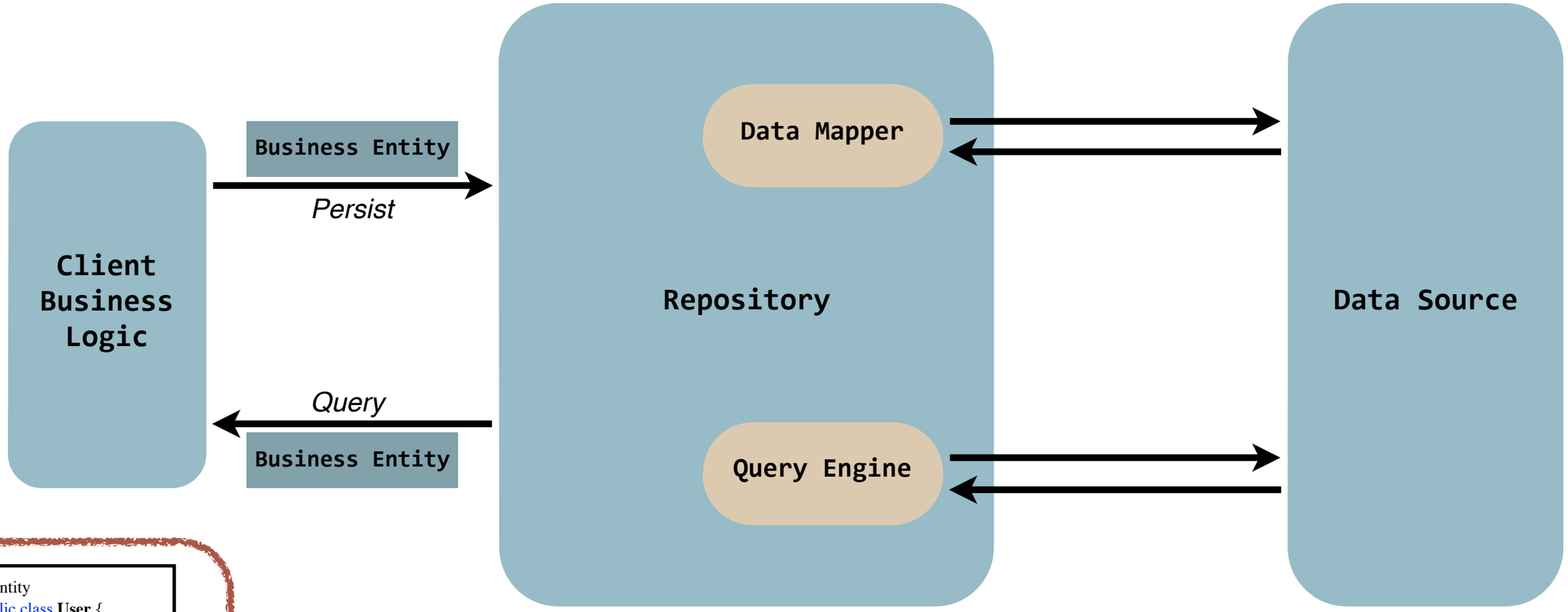
```
public interface AddressDao  extends
                    CrudRepository<Address,Long>{

}
```

Spring Style to Connect to DB (2)

Data Source

```
<bean id="mainDataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy-method="close">
   <property name="driverClass" value="“com.mysql.jdbc.Driver" />
   <property name="jdbcUrl" value="“jdbc:mysql://localhost/ESE?
                                    autoReconnect=true&amp;amp;createDatabaseIfNotExist=true&amp;amp;
                                    useUnicode=true&amp;amp;characterEncoding=utf-8" />
   <property name="username" value="root"/>
   <property name="password" value=""/>
   <property name="idleConnectionTestPeriodInMinutes" value="60"/>
   <property name="idleMaxAgeInMinutes" value="240"/>
   <property name="maxConnectionsPerPartition" value="30"/>
   <property name="minConnectionsPerPartition" value="10"/>
   <property name="partitionCount" value="3"/>
   <property name="acquireIncrement" value="5"/>
   <property name="statementsCacheSize" value="100"/>
   <property name="releaseHelperThreads" value="3"/>
</bean>
```



Business Logic

```
@Entity
public class User {
   @Id
   @GeneratedValue
   private Long id;

   private String firstName;
   private String lastName;
   private String email;

   @OneToOne
   private Address address;
.
.
}
```

```
@Entity
public class Address {
   @Id
   @GeneratedValue
   private Long id;
   private String street;
.
.
}
```

Repository

```
public interface AddressDao  extends
                CrudRepository<Address,Long>{

}
```

```
public interface UserDao extends
                CrudRepository<User,Long> {

}
```

Data Source

```
<bean id="mainDataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy-method="close">
   <property name="driverClass" value="com.mysql.jdbc.Driver" />
   <property name="jdbcUrl" value="jdbc:mysql://localhost/testBDB?
                                   autoReconnect=true&amp;createDatabaseIfNotExist=true&amp;
                                   useUnicode=true&amp;characterEncoding=utf-8" />
   <property name="username" value="root"/>
   <property name="password" value=""/>
   <property name="idleConnectionTestPeriodInMinutes" value="60"/>
   <property name="idleMaxAgeInMinutes" value="240"/>
   <property name="maxConnectionsPerPartition" value="30"/>
   <property name="minConnectionsPerPartition" value="10"/>
   <property name="partitionCount" value="3"/>
   <property name="acquireIncrement" value="5"/>
   <property name="statementsCacheSize" value="100"/>
   <property name="releaseHelperThreads" value="3"/>
</bean>
```

Spring Style to Connect to DB (3)

```
@Service
public class SampleServiceImpl
                implements SampleService {

    @Autowired    UserDao userDao;
    @Autowired    AddressDao addDao;

    @Transactional
    public int saveUser(){
        .
        .
        Address address = new Address();
        address.setStreet("TestStreet");

        User user = new User();
        user.setFirstName("John");
        user.setEmail(“john@do.org”);
        user.setLastName("do");
        user.setAddress(address);

        address = addDao.save(address);
        user = userDao.save(user);
        .
        .
}
```

```
public interface UserDao extends
                CrudRepository<User,Long> {


}
```

```
public void deleteAll();
public void delete(Iterable<? extends User> itrbl);
public void delete(User t);
public void delete(Long id);
public long count();
public Iterable<User> findAll(Iterable<Long> itrbl);
public Iterable<User> findAll();
public boolean exists(Long id);
public User findOne(Long id);
public <S extends User> Iterable<S> save(Iterable<S> itrbl);
public <S extends User> S save(S s);
```

Query Method Keywords

```
public interface UserDao extends CrudRepository<User,Long> {

    public Iterable<User> findByEmail(String email);

    public Iterable<User> findByFirstNameNotOrderByLastNameDesc(String firstName);

}
```

Query Method Keywords

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with prepended %)
ngWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with appended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false

Spring QueryDSL

.

.

QUser user = QUser.user;

JPQLQuery query = new JPQLQuery(entityManager)

User john = **query.from(user).where(user.firstName.eq(“John”))**

.

.

http://www.querydsl.com/

Useful Links

- **Spring Data JPA Quick Start**
  - http://spring.io/guides/gs/accessing-data-jpa/
- **Spring Data JPA Reference Documentation**
  - http://docs.spring.io/spring-data/jpa/docs/current/reference/html/
- **Spring Data JPA Tutorial**
  - http://spring.io/guides/tutorials/data/