Design Patterns

MSc Computer Science

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

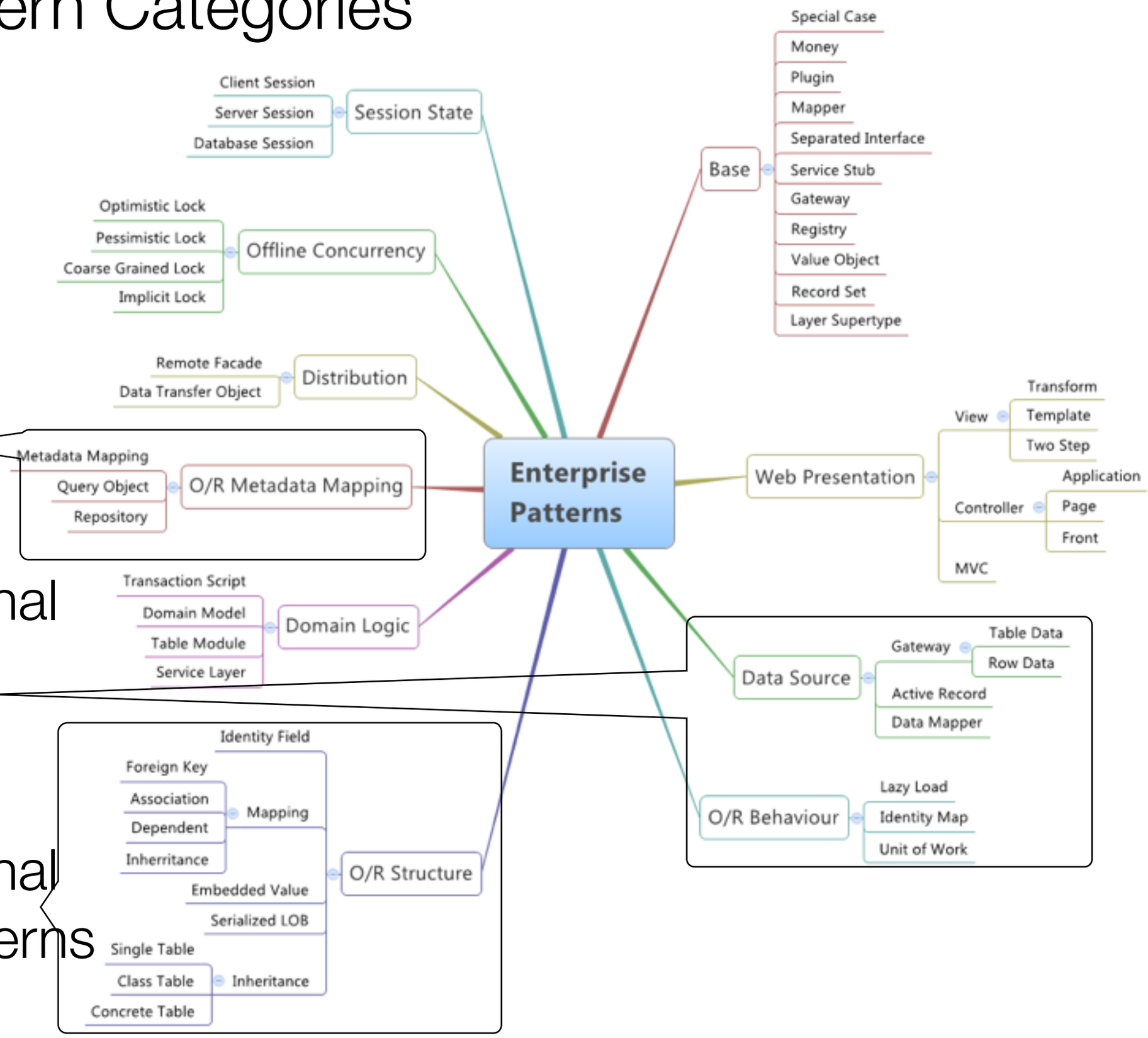
Data Source

Data Source

- Communicating with other systems that carry out tasks on behalf of the application.
- These can be transaction monitors, other applications, messaging systems, etc…
- For most enterprise applications this is a database that is primarily responsible for storing persistent data.

Database Pattern Categories

- Data Source Architectural Patterns
- Object-Relational Behavioural Patterns
- Object-Relational Structural Patterns

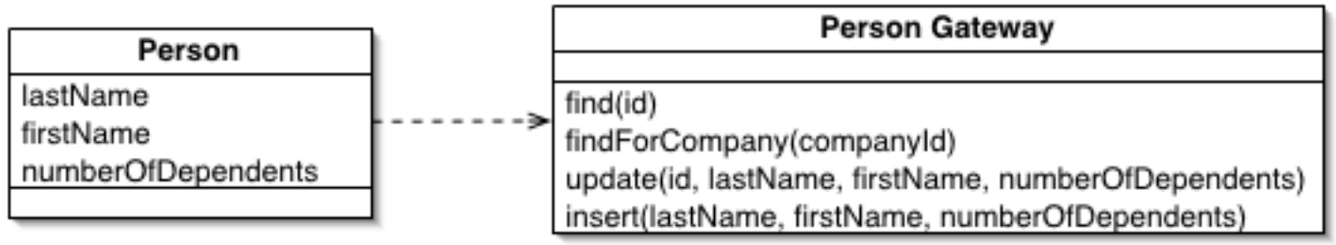


Data Source Architectural Patterns

- Drive the way in which the domain logic talks to the database.
- The choice made are far-reaching for the design and thus difficult to refactor
- Patterns
  - **Table Data Gateway**
  - **Active Record**
  - Row Data Gateway
  - Data Mapper

Table Data Gateway

An object that acts as a Gateway to a database table. One instance handles all the rows in the table.

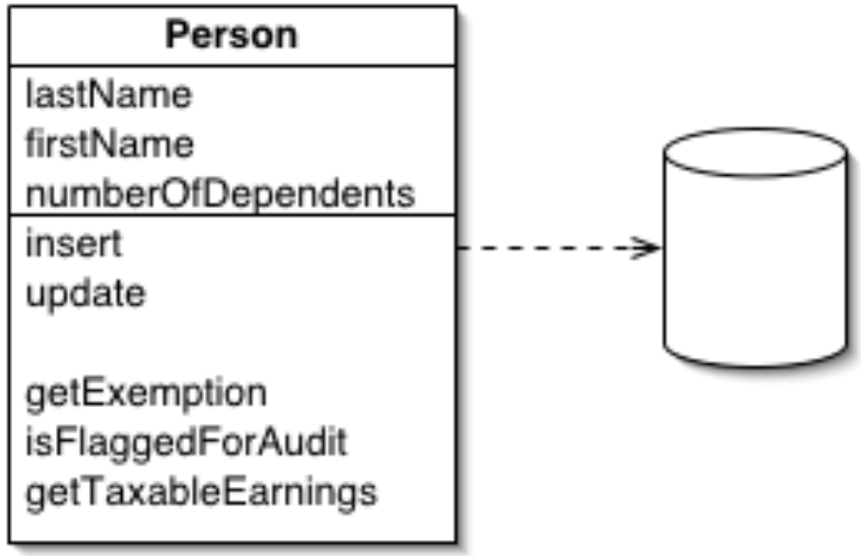


- Mixing SQL in application logic can cause several problems:
  - Many developers aren't comfortable with SQL, and many who are comfortable may not write it well.
  - Database administrators need to be able to find SQL easily so they can figure out how to tune and evolve the database.
- A Table Data Gateway holds all the SQL for accessing a single table or view: selects, inserts, updates, and deletes.
- Other code calls its methods for all interaction with the database.

Active Record

An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data

- An object carries both data and behavior.
- Much of this data is persistent and needs to be stored in a database.
- Active Record uses the most obvious approach, putting data access logic in the domain object.
- This way all people know how to read and write their data to and from the database.



Data Source Architectural Patterns in Pacemaker Play

Active Record

- Static methods encapsulate all access to User table

```
public class User extends Model
{
  @Id
  @GeneratedValue
  public Long   id;
  public String firstname;
  public String lastname;
  public String email;
  public String password;

  @OneToMany(cascade=CascadeType.ALL)
  public List<Activity> activities = new ArrayList<Activity>();

  //...

  public static User findByEmail(String email)
  {
    return  User.find.where().eq("email", email).findUnique();
  }

  public static User findById(Long id)
  {
    return find.where().eq("id", id).findUnique();
  }

  public static List<User> findAll()
  {
    return find.all();
  }

  public static void deleteAll()
  {
    for (User user: User.findAll())
    {
      user.delete();
    }
  }

  public static Model.Finder<String, User>
          find = new Model.Finder<String, User>(String.class, User.class);
}
```

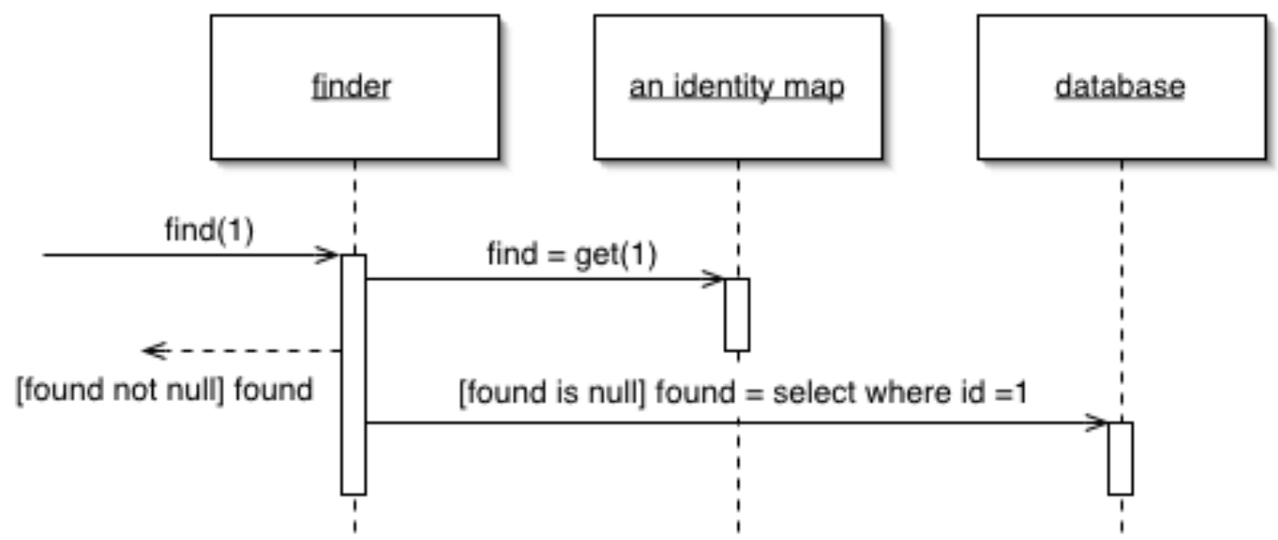
Object-Relational Behavioural Patterns

- How to get the various objects to load and save themselves to the databases you read objects and modify them, you have to ensure that the database state you're working with stays consistent.
- If you read some objects, it's important to ensure that the reading is isolated so that no other process changes any of the objects you've read while you're working on them.
- Otherwise, you could have inconsistent and invalid data in your objects.
- Patterns
  - **Identity Map**
  - **Unit of Work**
  - **Lazy Load**

Identity Map

Ensures that each object gets loaded only once by keeping every loaded object in a map. Looks up objects using the map when referring to them.

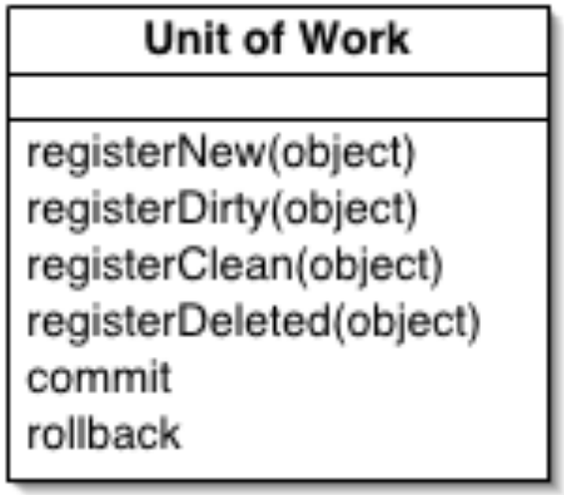
- An Identity Map keeps a record of all objects that have been read from the database in a single business transaction.
- Whenever you want an object, you check the Identity Map first to see if you already have it.
- Avoids problem whereby data from the same database record into two different objects
- Also, may improve efficiency



Unit of Work

Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems.

- If you change the database with each change to your object model this can lead to lots of very small database calls
- Furthermore it requires you to have a transaction open for the whole interaction, which is impractical if you have a business transaction that spans multiple requests.
- A Unit of Work keeps track of everything you do during a business transaction that can affect the database, and resolve all outstanding changes when the task is concluded.



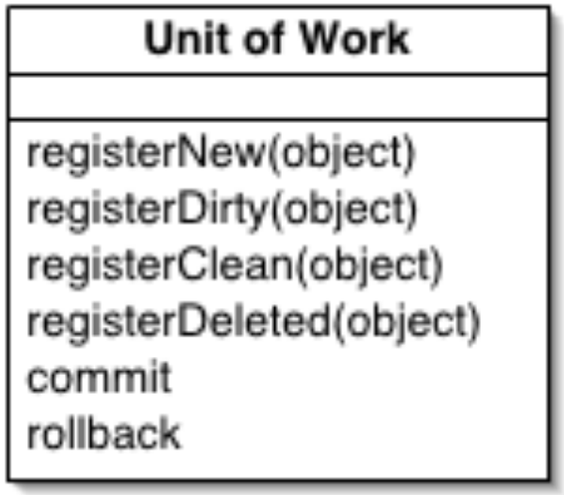
Lazy Load

An object that doesn't contain all of the data you need but knows how to get it.

- For loading data from a database into memory it's convenient to load the objects that are related to the object of interest.

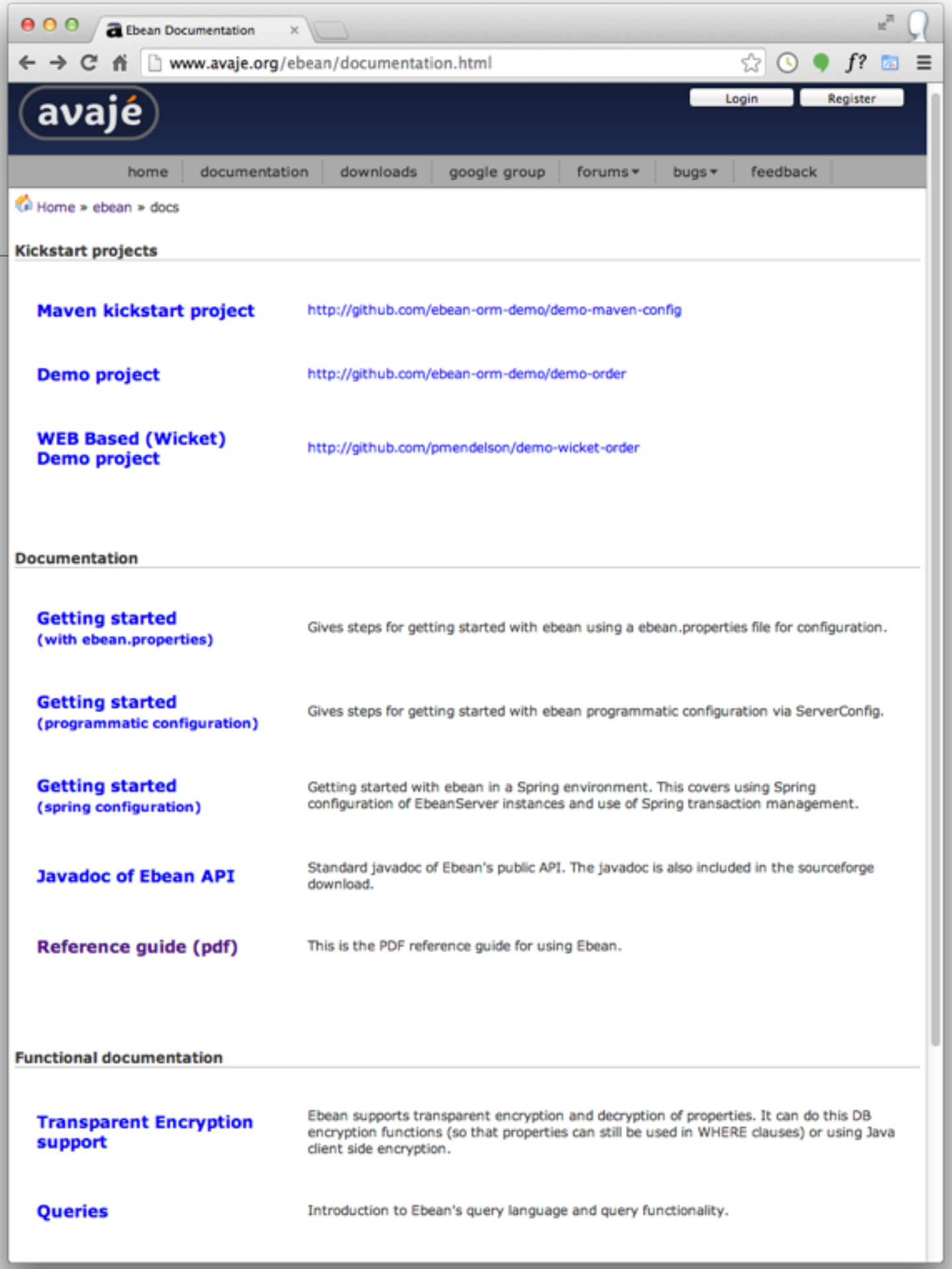
- However, if you take this to its logical conclusion, you reach the point where loading one object can have the effect of loading a huge number of related objects.

- A Lazy Load interrupts this loading process, leaving a marker in the object structure so that if the data is needed it can be loaded only when it is used



Object-Relational Behavioural Patterns in Pacemaker Play

- pacemaker-play uses ebean JPA provider
- Which implements a full ranger of Behavioural patterns



Object-Relational Structural Patterns

- Primarily concerned with the way objects and relations handle links
- Two main problems:
  - Objects handle links by storing references that are held by the runtime of either memory-managed environments or memory addresses. Relational databases handle links by forming a key into another table.
  - Objects can easily use collections to handle multiple references from a single field, while normalization forces all relation links to be single valued. This leads to reversals of the data structure between objects and tables.
- An order object naturally has a collection of line item objects that don't need any reference back to the order. However, the table structure is the other way around—the line item must include a foreign key reference to the order since the order can't have a multivalued field.

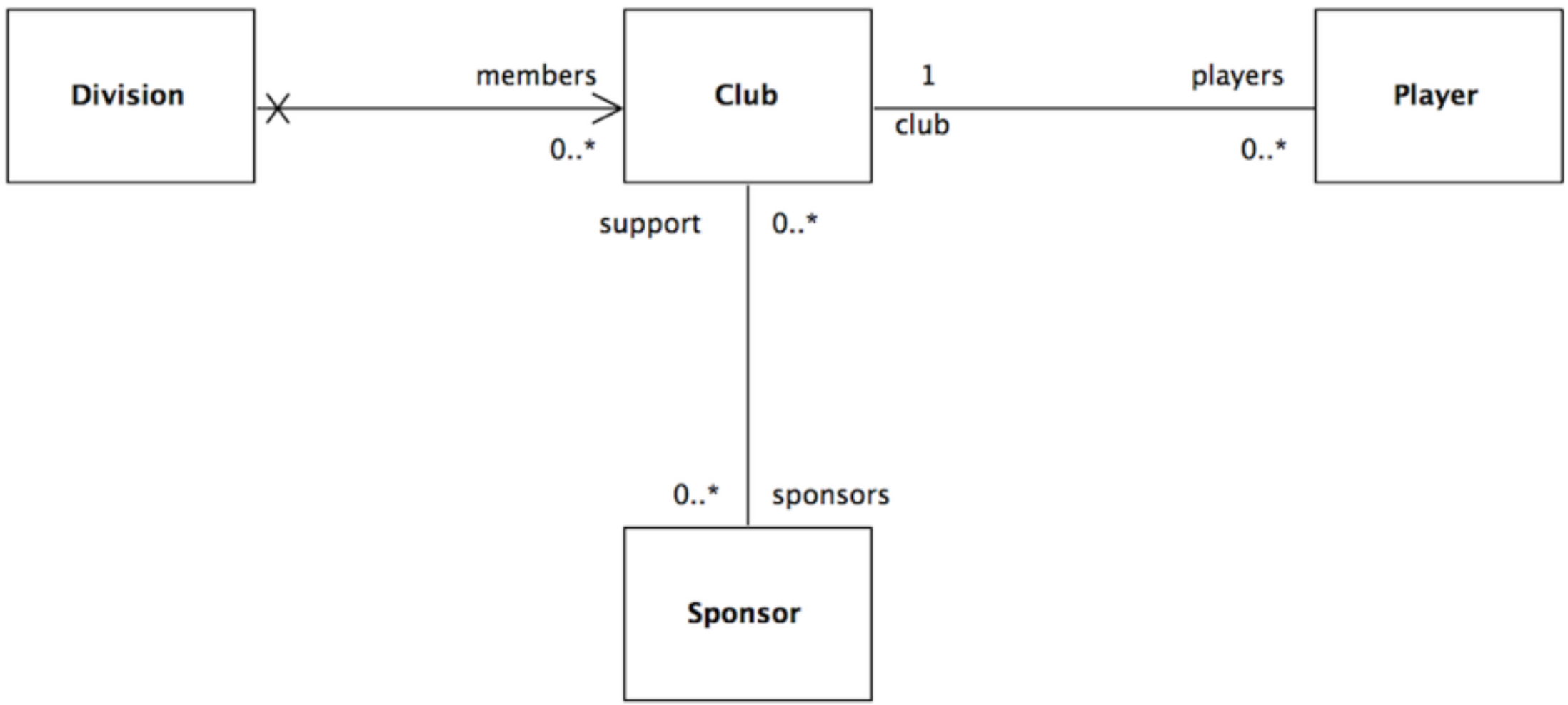
Object-Relational Structural Patterns

- Identity Field
- Foreign Key Mapping
- Association Table Mapping
- Dependent Mapping
- Embedded Value
- Serialized LOB
- Single Table Inheritance
- Class Table Inheritance
- Concrete Table Inheritance
- Inheritance Mappers

- Most of these patterns have been absorbed into Object Relational Mapping frameworks
- This include 4 mapping annotations:
  - OneToMany
  - ManyToOne
  - OneToOne
  - ManyToMany
- + a range of other annotations to support sophisticated mapping schemes

Major Object-Relational Structural Patterns specified by JPA 2 and implemented by bean

OneToMany, ManyToOne, ManyToMany

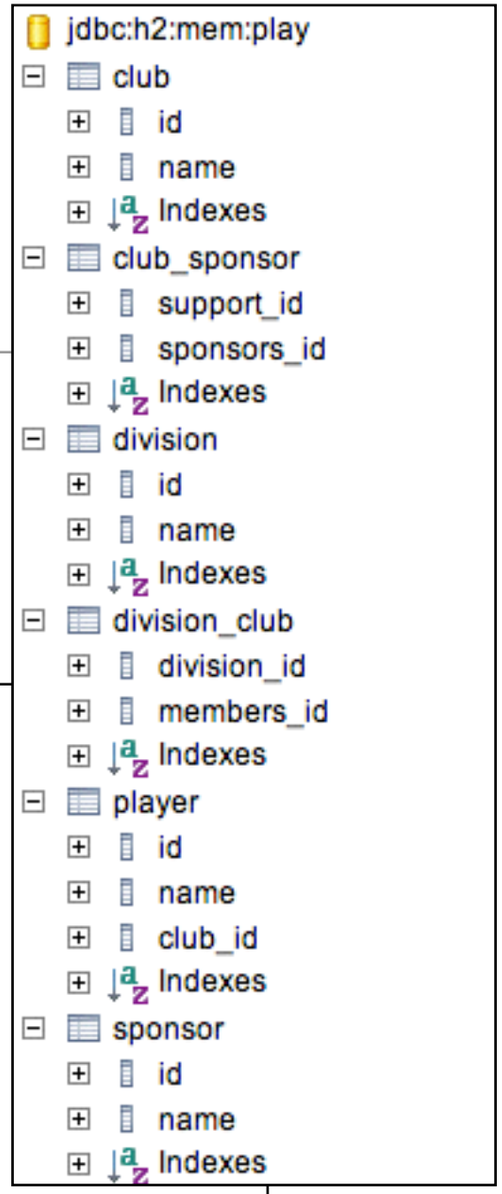


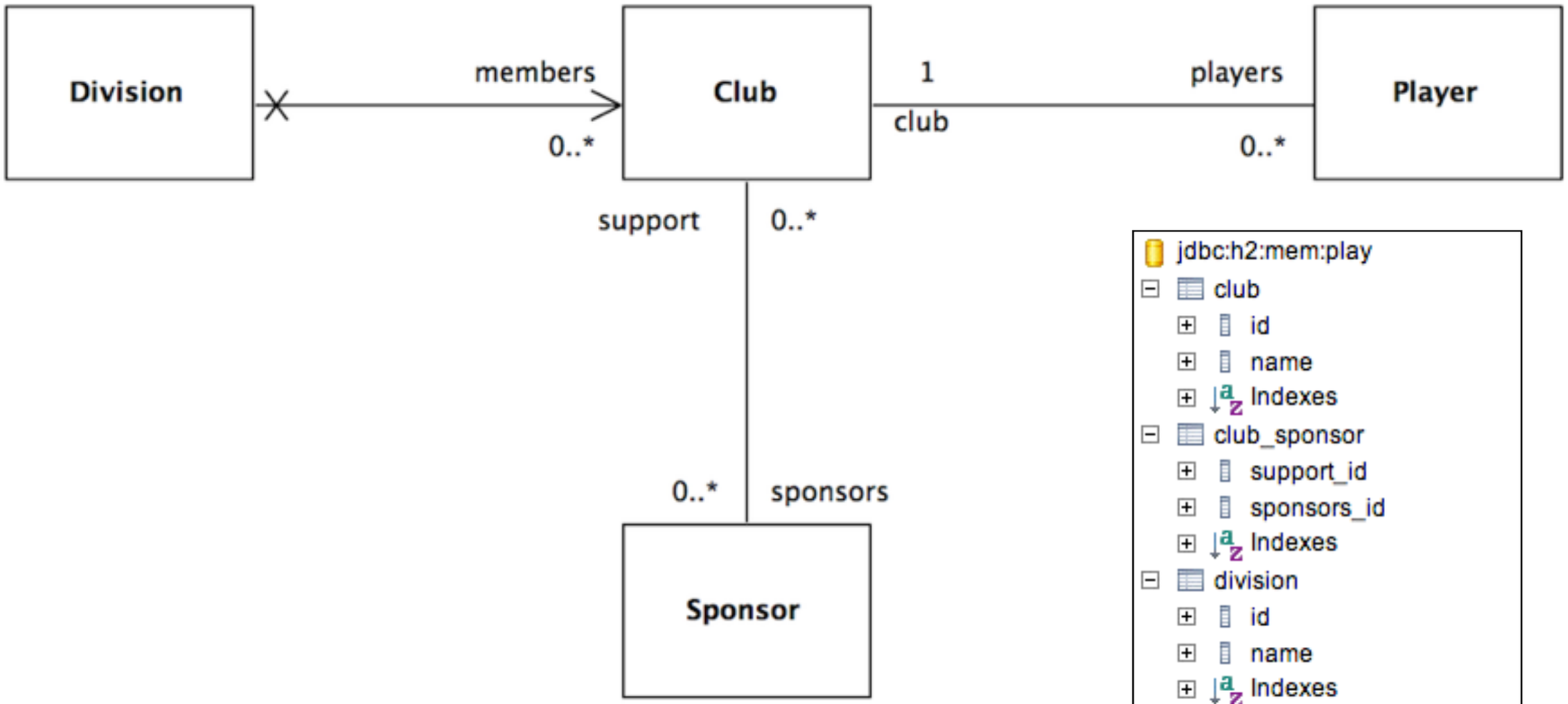
```java
@Entity
public class Club extends Model
{
  @Id
  @GeneratedValue
  public Long id;
  public String name;

  @OneToMany(mappedBy="club", cascade=CascadeType.ALL)
  public List<Player> players = new ArrayList<Player>();

  @ManyToMany
  public List<Sponsor> sponsors;

  public Club(String name)
  {
    this.name = name;
  }
}
```

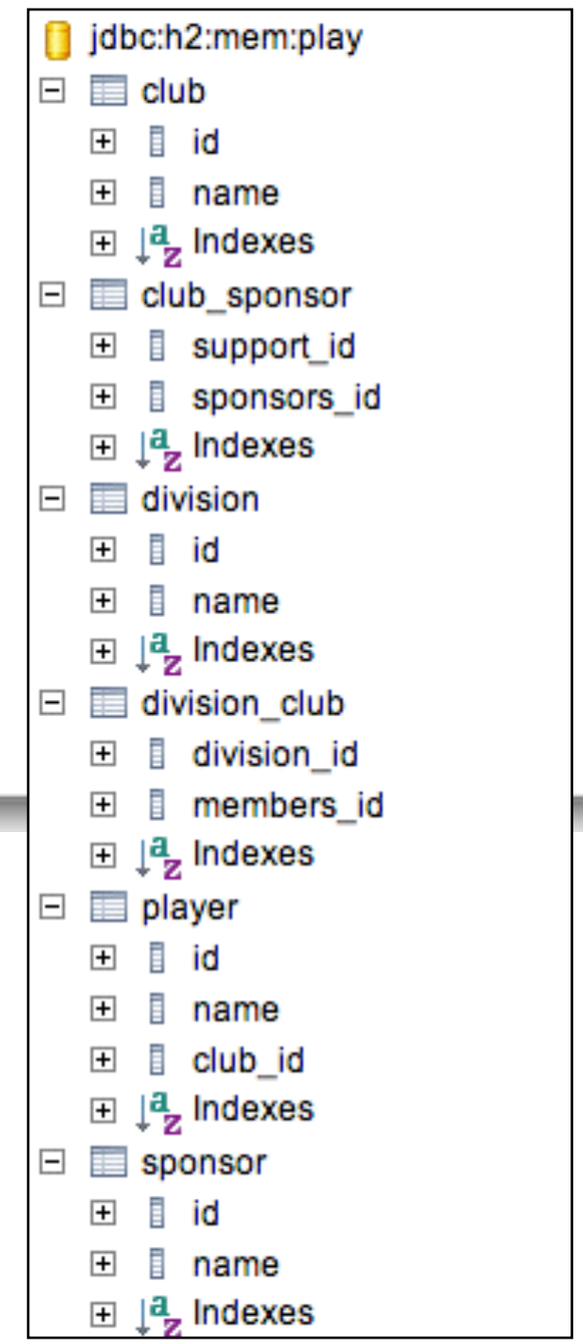
```java
@Entity
public class Division extends Model
{
  @Id
  @GeneratedValue
  public Long  id;
  public String name;

  @OneToMany(cascade=CascadeType.ALL)
  public List<Club> membersnew ArrayList<Club>();

  public Division(String name)
  {
    this.name = name;
  }
}
```

```java
@Entity
public class Sponsor extends Model
{
  @Id
  @GeneratedValue
  public Long   id;
  public String name;

  @ManyToMany (mappedBy="sponsors")
  public List<Club> support new ArrayList<Club>();

  public Sponsor(String name)
  {
    this.name = name;
  }
}
```

```java
@Entity
public class Player extends Model
{
  @Id
  @GeneratedValue
  public Long  id;
  public String name;

  @ManyToOne
  public Club club;

  public Player(String name)
  {
    this.name = name;
  }

  public String toString()
  {
    return name;
  }

  public static Player findByName(String name)
  {
    return find("name", name).first();
  }
}
```




Division
members
0..*
Club
1
club
players
0..*
Player
support
0..*
0..*
sponsors
Sponsor


jdbc:h2:mem:play
club
id
name
Indexes
club_sponsor
support_id
sponsors_id
Indexes
division
id
name
Indexes
division_club
division_id
members_id
Indexes
player
id
name
club_id
Indexes
sponsor
id
name
Indexes