Mobile Application Development

Higher Diploma in Science in 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

UML & JPA Modelling

Agenda

- Introduce UML Class Diagram modeling using Visual Paradigm
- Define a simple model and implement it in Play
- Write comprehensive unit tests to exercise the model

JPA Model Project

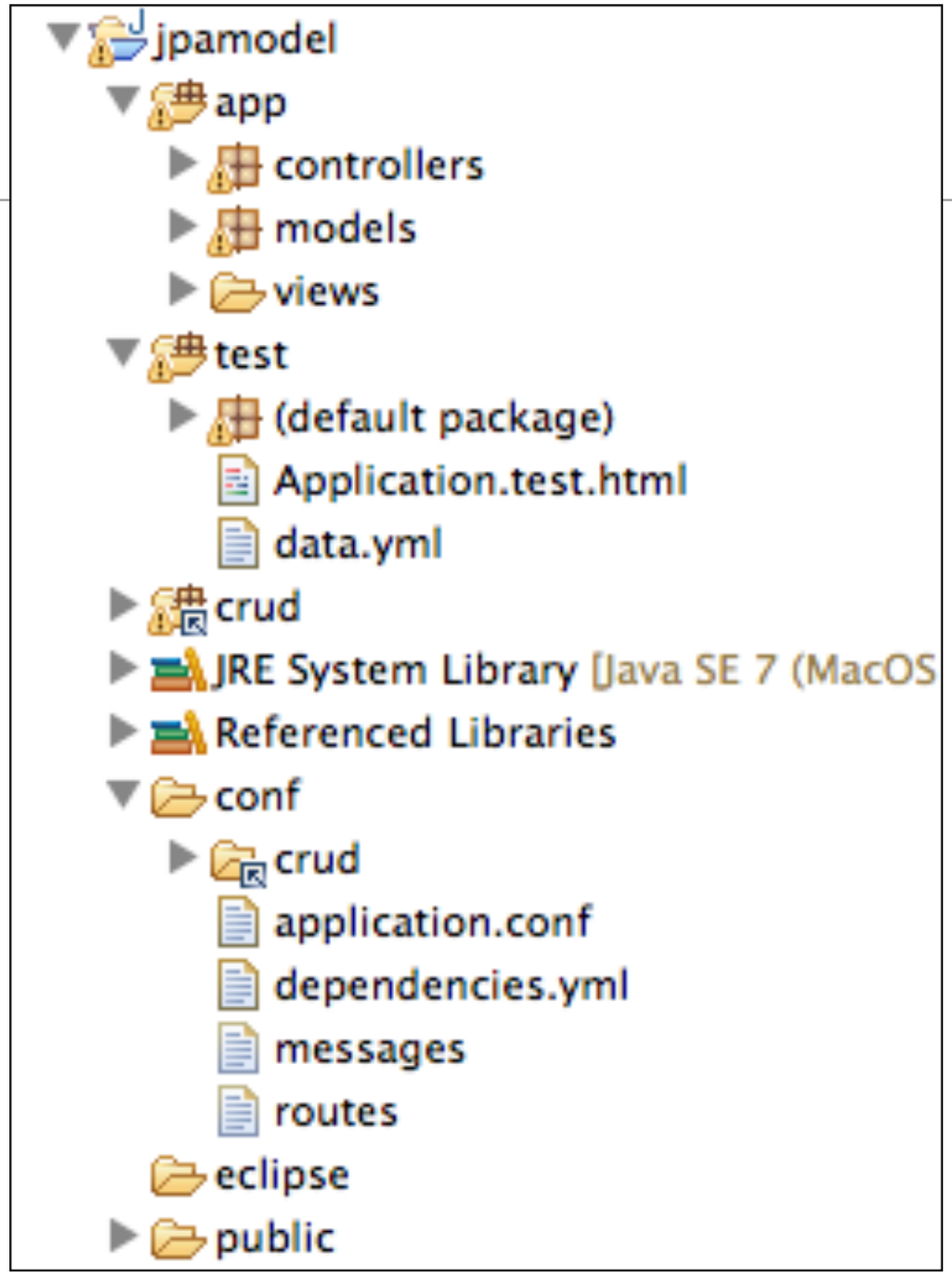
Start by creating a brand new Play project. Do this by determining the parent folder (most likely your workspace) and running a command prompt. Then type:

```
play new jpamodel
```

Once this has completed, change into the folder just created (jpamodel) and run the eclipsify command:

```
cd jpamodel
play eclipsify
```

You can now import the project into eclipse in the usual way.


jpamodel
app
controllers
models
views
test
(default package)
Application.test.html
data.yml
crud
JRE System Library [Java SE 7 (MacOS
Referenced Libraries
conf
crud
application.conf
dependencies.yml
messages
routes
eclipse
public

Club Class

```
package models;

import javax.persistence.Entity;
import play.db.jpa.Model;

@Entity
public class Club extends Model
{
  public String name;

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

Player Class

```
package models;

import javax.persistence.Entity;
import play.db.jpa.Model;

@Entity
public class Player extends Model
{
  public String name;

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

ClubTest

```java
import org.junit.*;

import java.util.*;
import play.test.*;
import models.*;

public class ClubTest extends UnitTest
{
  @Before
  public void setup()
  {
  }

  @After
  public void teardown()
  {
  }

  @Test
  public void testCreate()
  {

  }
}
```

PlayerTest

```java
import org.junit.*;

import java.util.*;
import play.test.*;
import models.*;

public class PlayerTest extends UnitTest
{
  @Before
  public void setup()
  {
  }

  @After
  public void teardown()
  {
  }

  @Test
  public void testCreate()
  {

  }
}
```

Run the app now in 'test' mode:

```
play test
```

...and navigate to the test runner page:

- `http://localhost:9000/@tests`

Select the Club and Player tests - and they should be green.

Also try the database interface:

- `http://localhost:9000/@db`

Auto commit | Max rows: 1000 | Auto complete Normal

- jdbc:h2:mem:play
  - club
    - id
    - name
    - Indexes
  - player
    - id
    - name
    - Indexes
  - information_schema
  - Sequences
  - Users
- H2 1.3.166 (2012-04-08)

Run (Ctrl+Enter) Clear SQL statement:

Important Commands

	Displays this Help Page
	Shows the Command History
	Executes the current SQL statement
	Disconnects from the database

Sample SQL Script

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

Adding Database Drivers

PlayerTest

```java
public class PlayerTest extends UnitTest
{
  private Player p1, p2, p3;

  @Before
  public void setup()
  {
    p1 = new Player("mike");
    p2 = new Player("jim");
    p3 = new Player("frank");
    p1.save();
    p2.save();
    p3.save();
  }

  @After
  public void teardown()
  {
    p1.delete();
    p2.delete();
    p3.delete();
  }

  @Test
  public void testCreate()
  {
  }
}
```

Auto commit | Max rows: 1000 | Auto complete Normal

- jdbc:h2:mem:play
  - club
    - id
    - name
    - Indexes
  - player
    - id
    - name
    - Indexes
  - information_schema
  - Sequences
  - Users
- H2 1.3.166 (2012-04-08)

Run (Ctrl+Enter) Clear SQL statement:

```
SELECT * FROM CLUB
```

SELECT * FROM CLUB;

ID	NAME

(no rows, 25 ms)

Edit

toString + //@After

```java
public class Player extends Model
{
  public String name;

  @ManyToOne
  public Club club;

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

  public String toString()
  {
    return name;
  }
}
```

- We can use Admin interface while project is in ‘test’ mode
- Enables us to understand model as we evolve classes and their relationships

```java
public class PlayerTest extends UnitTest
{
  private Player p1, p2, p3;

  @Before
  public void setup()
  {
    p1 = new Player("mike");
    p2 = new Player("jim");
    p3 = new Player("frank");
    p1.save();
    p2.save();
    p3.save();
  }

  //@After
  public void teardown()
  {
    p1.delete();
    p2.delete();
    p3.delete();
  }

  @Test
  public void testCreate()
  {
    Player a = Player.findByName("mike");
    assertNotNull(a);
    assertEquals("mike", a.name);
    Player b = Player.findByName("jim");
    assertNotNull(b);
    assertEquals("jim", b.name);
    Player c = Player.findByName("frank");
    assertNotNull(c);
    assertEquals("frank", c.name);
  }
```

Auto commit | Max rows: 1000 | Auto complete Normal

jdbc:h2:mem:play
- club
  - id
  - name
  - Indexes
- player
  - id
  - name
  - Indexes
- information_schema
- Sequences
- Users

H2 1.3.166 (2012-04-08)

Run (Ctrl+Enter) Clear SQL statement:

```
SELECT * FROM CLUB
```

SELECT * FROM CLUB;

ID	NAME
1	tramore
2	dunmore
3	fenor
4	tramore
5	dunmore
6	fenor

(6 rows, 3 ms)

Edit

```
private Player p1, p2, p3;

public void setup()
{
  p1 = new Player("mike");
  p2 = new Player("jim");
  p3 = new Player("frank");
  p1.save();
  p2.save();
  p3.save();
}
```

Some Player Tests

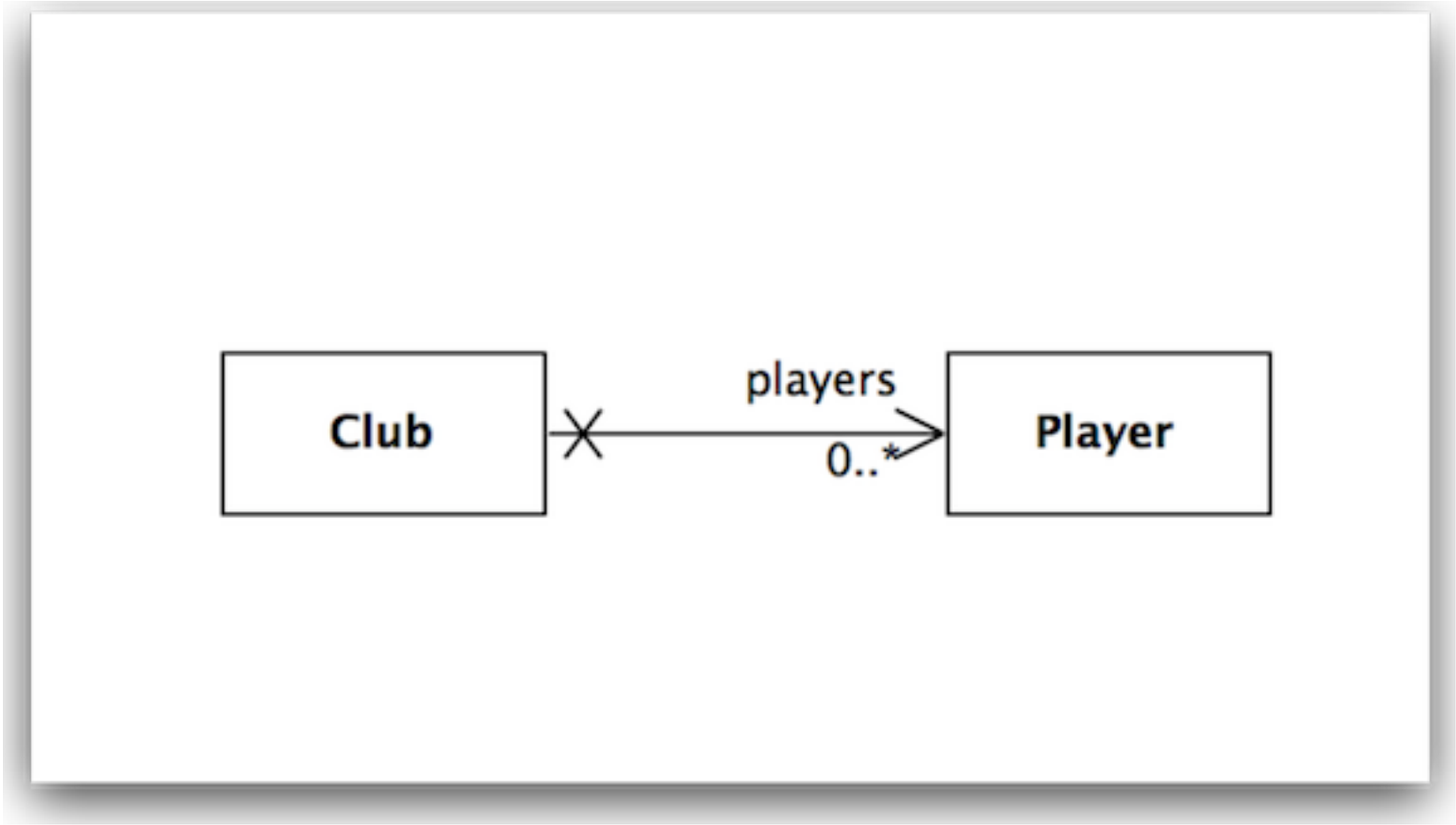
```
@Test
public void testCreate()
{
  Player a = Player.findByName("mike");
  assertNotNull(a);
  assertEquals("mike", a.name);
  Player b = Player.findByName("jim");
  assertNotNull(b);
  assertEquals("jim", b.name);
  Player c = Player.findByName("frank");
  assertNotNull(c);
  assertEquals("frank", c.name);
}

@Test
public void testNotThere()
{
  Player a = Player.findByName("george");
  assertNull(a);
}
```

ClubTest

```java
public class ClubTest extends UnitTest
{
  private Club c1, c2, c3;

  @Before
  public void setup()
  {
    c1 = new Club("tramore");
    c2 = new Club("dunmore");
    c3 = new Club("fenor");
    c1.save();
    c2.save();
    c3.save();
  }

  @After
  public void teardown()
  {
    c1.delete();
    c2.delete();
    c3.delete();
  }
```

```java
  @Test
  public void testCreate()
  {
    Club a = Club.findByName("tramore");
    assertNotNull(a);
    assertEquals("tramore", a.name);
    Club b = Club.findByName("dunmore");
    assertNotNull(b);
    assertEquals("dunmore", b.name);
    Club c = Club.findByName("fenor");
    assertNotNull(c);
    assertEquals("fenor", c.name);
  }

  @Test
  public void testNotThere()
  {
    Club a = Club.findByName("bunmahon");
    assertNull(a);
  }
}
```

Multiplicity & Navigation



- Club has a collection of zero or more players
- Players are unaware of Club

Implementation Relationship in Java Classes

```java
public class Club extends Model
{
  public String name;

  @OneToMany(cascade=CascadeType.ALL)
  public List<Player> players;


  public Club(String name)
  {
    this.name = name;
    this.players = new ArrayList<Player>();
  }

  public String toString()
  {
    return name;
  }

  public void addPlayer(Player player)
  {
    players.add(player);
  }
}
```

```java
public class Player extends Model
{
  public String name;


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

  public String toString()
  {
    return name;
  }
}
```

Testing the Player / Club Relationship

- Use the fixture to set up some club / relationships

```
@Before
public void setup()
{
  p1 = new Player("mike");
  p2 = new Player("jim");
  p3 = new Player("frank");

  c1 = new Club("tramore");
  c2 = new Club("dunmore");
  c3 = new Club("fenor");

  c1.addPlayer(p1);
  c1.addPlayer(p2);

  c1.save();
  c2.save();
  c3.save();
}
```

testPlayers

- In the test, see if these relationship have been established

```
@Test
public void testPlayers()
{
  Club tramore = Club.findByName("tramore");

  assertEquals (2, tramore.players.size());

  Player mike  = Player.findByName("mike");
  Player jim   = Player.findByName("jim");
  Player frank = Player.findByName("framk");

  assertTrue  (tramore.players.contains(mike));
  assertTrue  (tramore.players.contains(jim));
  assertFalse (tramore.players.contains(frank));
}
```

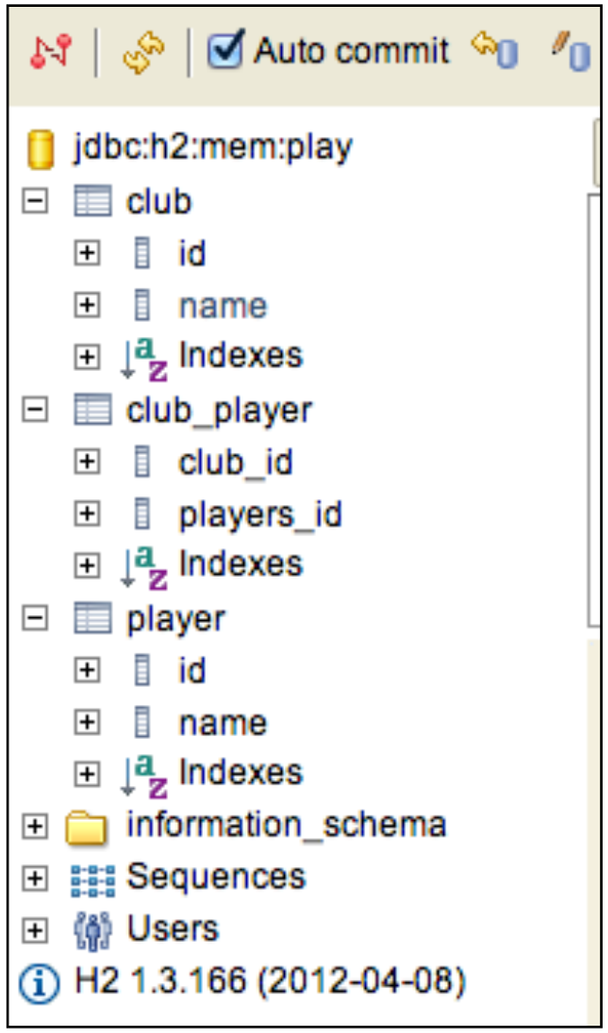
testRemovePlayers

- Removing relationships must also be tested

```
@Test
public void testRemovePlayer()
{
  Club tramore = Club.findByName("tramore");
  assertEquals(2, tramore.players.size());

  Player mike  = Player.findByName("mike");
  assertTrue(tramore.players.contains(mike));
  tramore.players.remove(mike);
  tramore.save();

  Club c = Club.findByName("tramore");
  assertEquals(1, c.players.size());

  mike.delete();
}
```

Explore the Relationship in the Database

```
@Before
public void setup()
{
  p1 = new Player("mike");
  p2 = new Player("jim");
  p3 = new Player("frank");

  c1 = new Club("tramore");
  c2 = new Club("dunmore");
  c3 = new Club("fenor");

  c1.addPlayer(p1);
  c1.addPlayer(p2);

  c1.save();
  c2.save();
  c3.save();
}
```



SELECT * FROM CLUB;

ID	NAME
1	tramore
2	dunmore
3	fenor

(3 rows, 3 ms)

SELECT * FROM PLAYER;

ID	NAME
1	mike
2	jim

(2 rows, 2 ms)

Edit

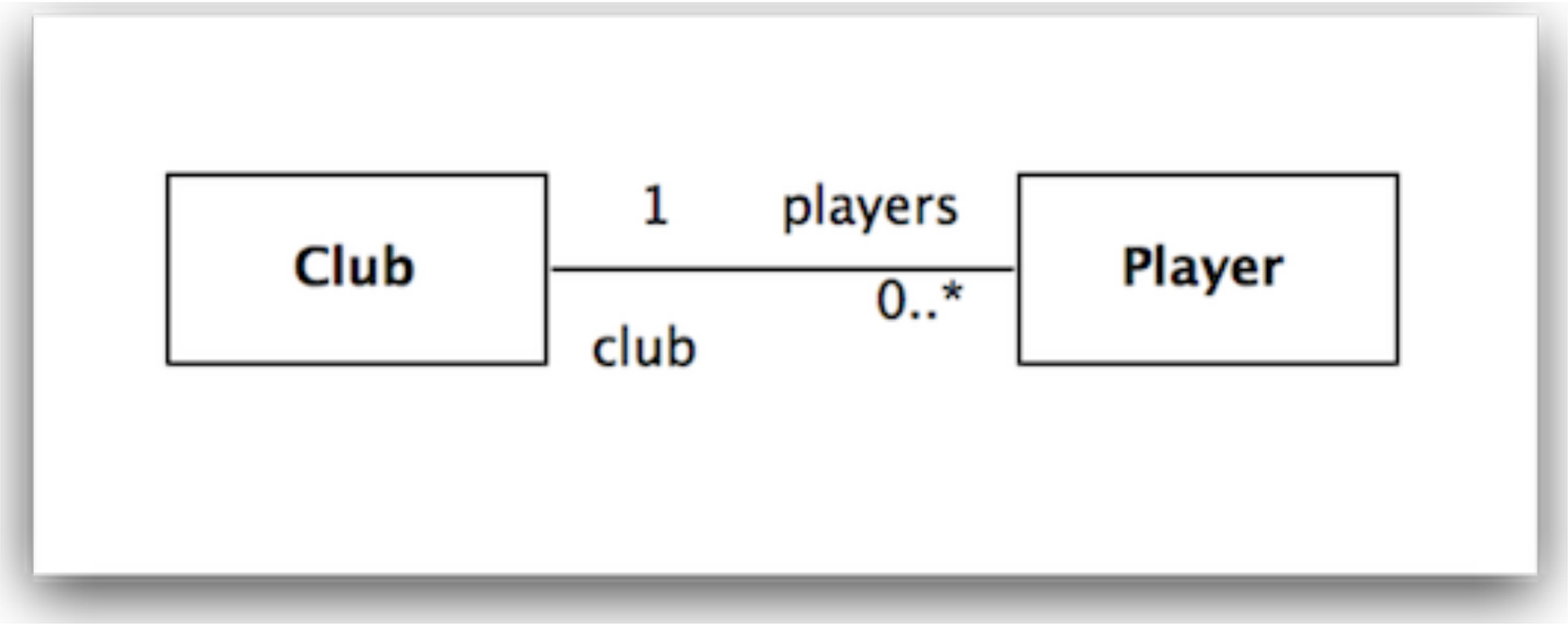
SELECT * FROM CLUB_PLAYER;

CLUB_ID	PLAYERS_ID
1	1
1	2

(2 rows, 4 ms)

Edit

Bidirectional Relationship

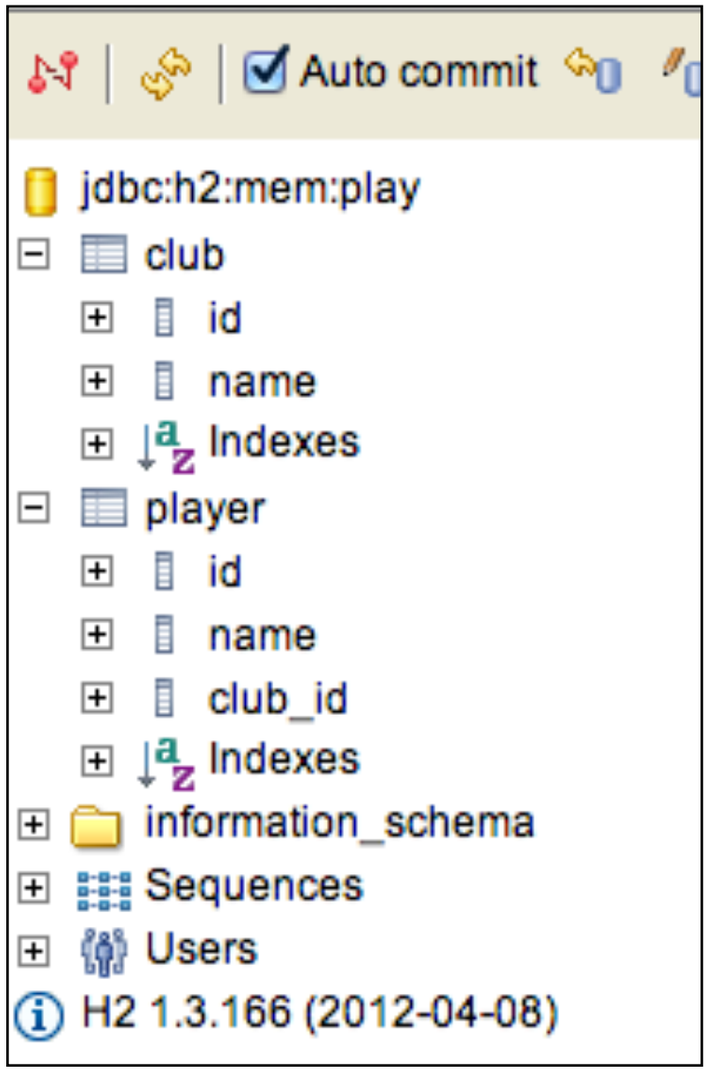


- Club has a ‘one to many’ relationship with players
- Player has a ‘many to one’ relationship with club

Bidirectional Relationship in Java Classes

```java
public class Club extends Model
{
  public String name;

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


  public Club(String name)
  {
    this.name = name;
    this.players = new ArrayList<Player>();
  }

  public String toString()
  {
    return name;
  }

  public void addPlayer(Player player)
  {
    player.club = this;
    players.add(player);
  }
}
```

```java
public class Player extends
Model
{
  public String name;

  @ManyToOne
  public Club club;

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

  public String toString()
  {
    return name;
  }
}
```



SELECT * FROM CLUB;

ID	NAME
1	tramore
2	dunmore
3	fenor

(3 rows, 3 ms)

Edit

SELECT * FROM PLAYER;

ID	NAME	CLUB_ID
1	mike	1
2	jim	1

(2 rows, 2 ms)

Edit