

# 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

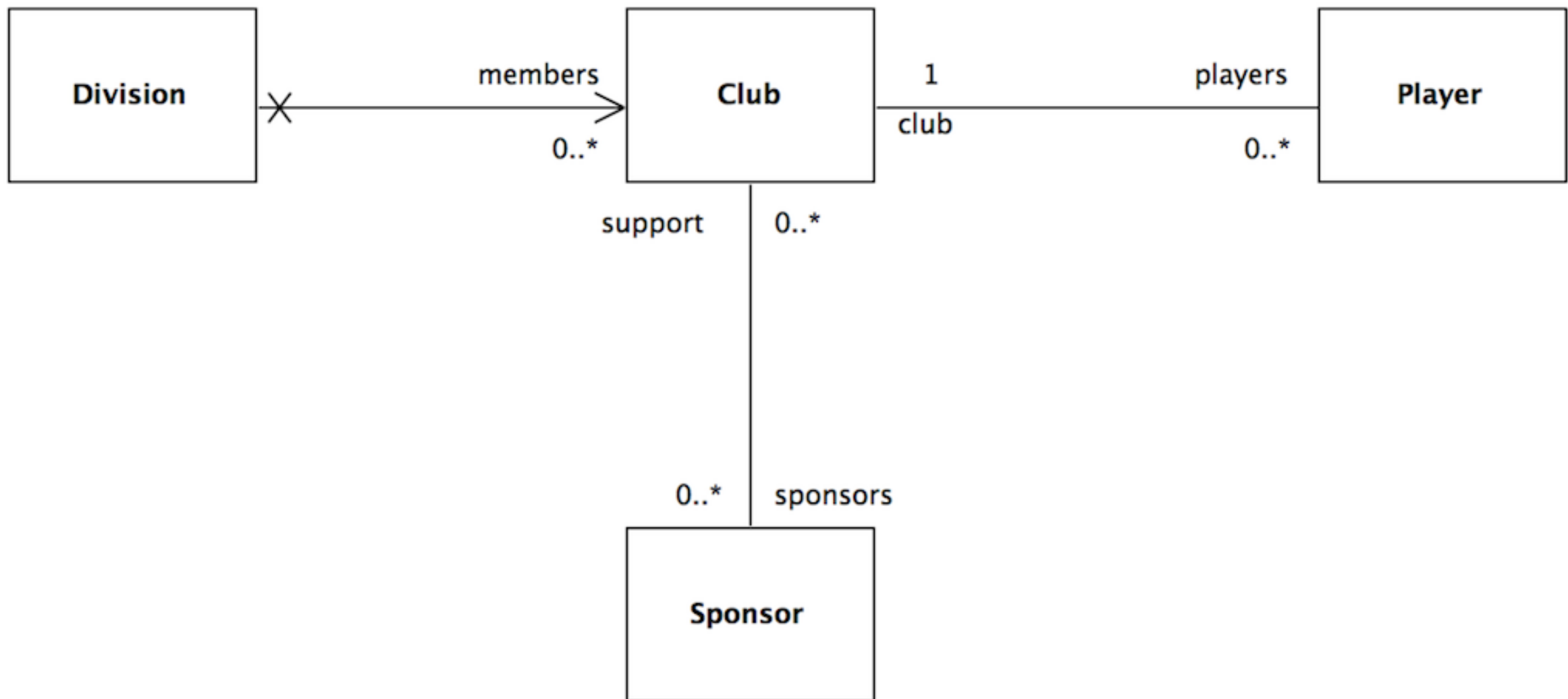


# JPA II - ManyToMany

---

# OneToMany, ManyToOne, ManyToMany

---



# OneToMany

```
public class Division extends Model
{
    public String name;

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

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

    public void addClub(Club club)
    {
        members.add(club);
    }

    public String toString()
    {
        return name;
    }

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

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

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

    @ManyToMany
    public List<Sponsor> sponsors;

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

    public String toString()
    {
        return name;
    }

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

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

    public void addSponsor(Sponsor company)
    {
        sponsors.add(company);
    }

    public void removePlayer(Player player)
    {
        players.remove(player);
    }
}
```

# ManyToOne

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

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

    //..
}
```

```
public class Player extends Model
{
    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();
    }
}
```

# ManyToMany

```
public class Sponsor extends Model
{
    public String name;

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

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

    public void addSuport(Club club)
    {
        support.add(club);
    }

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

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

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

    @ManyToMany
    public List<Sponsor> sponsors;

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

    public String toString()
    {
        return name;
    }

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

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

    public void addSponsor(Sponsor company)
    {
        sponsors.add(company);
    }

    public void removePlayer(Player player)
    {
        players.remove(player);
    }
}
```

# Tests

- For more complex models, create fixtures in data.yml.
- These models can be loaded in unit tests

```
Club(dunmore):  
  name: dunmore  
  
Club(tramore):  
  name: tramore  
  
Club(fenor):  
  name: fenor  
  
Player(jim):  
  name: jim  
  club: dunmore  
  
Player(mary):  
  name: mary  
  club: dunmore  
  
Player(sam):  
  name: sam  
  club: tramore  
  
Player(john):  
  name: john  
  club: tramore  
  
Player(mike):  
  name: mike  
  club: fenor  
  
Player(linda):  
  name: john  
  club: fenor  
  
Division(senior):  
  name: senior  
  members:  
    - tramore  
    - dunmore  
  
Division(junior):  
  name: junior  
  members:  
    - fenor  
  
Sponsor(newsagent):  
  name: newsagent  
  
Sponsor(pub):  
  name: pub
```

data.yml

data.yml

```
Club(dunmore):
  name: dunmore

Club(tramore):
  name: tramore

Club(fenor):
  name: fenor

Player(jim):
  name: jim
  club: dunmore

Player(mary):
  name: mary
  club: dunmore

Player(sam):
  name: sam
  club: tramore

Player(john):
  name: john
  club: tramore

Player(mike):
  name: mike
  club: fenor

Player(linda):
  name: john
  club: fenor

Division(senior):
  name: senior
  members:
    - tramore
    - dunmore

Division(junior):
  name: junior
  members:
    - fenor

Sponsor(newsagent):
  name: newsagent

Sponsor(pub):
  name: pub
```

# ComprehensiveTest

```
public class ComprehensiveTest extends UnitTest
{
    @Before
    public void setup()
    {
        Fixtures.loadModels("data.yml");
    }

    @After
    public void teardown()
    {
        Fixtures.deleteAllModels();
    }
}
```



Club(dunmore):  
 name: dunmore

Club(tramore):  
 name: tramore

Club(fenor):  
 name: fenor

Player(jim):  
 name: jim  
 club: dunmore

Player(mary):  
 name: mary  
 club: dunmore

Player(sam):  
 name: sam  
 club: tramore

Player(john):  
 name: john  
 club: tramore

Player(mike):  
 name: mike  
 club: fenor

Player(linda):  
 name: john  
 club: fenor

Division(senior):  
 name: senior  
 members:  
 - tramore  
 - dunmore

Division(junior):  
 name: junior  
 members:  
 - fenor

Sponsor(newsagent):  
 name: newsagent

Sponsor(pub):  
 name: pub

SELECT \* FROM CLUB;

ID	NAME
1	dunmore
2	tramore
3	fenor

(3 rows, 3 ms)

SELECT \* FROM PLAYER;

ID	NAME	CLUB_ID
1	jim	1
2	mary	1
3	sam	2
4	john	2
5	mike	3
6	linda	3

(6 rows, 2 ms)

SELECT \* FROM DIVISION;

ID	NAME
1	senior
2	junior

(2 rows, 1 ms)

SELECT \* FROM DIVISION\_CLUB;

DIVISION_ID	MEMBERS_ID
1	2
1	1
2	3

(3 rows, 3 ms)

SELECT \* FROM SPONSOR;

ID	NAME
1	newsagent
2	pub

(2 rows, 2 ms)

# Yaml file - Forward References

---

- Test data in Yaml file cannot refer to objects that have not been seen in the file yet (reading from top to bottom)
- Bidirectional references can be included by including the objects twice
  - Once at top (partial)
  - Once at end (complete)

```
Sponsor(pub):  
  name: pub  
  
Sponsor(newsagent):  
  name: newsagent
```

```
Club(tramore):  
  name: tramore  
  sponsors:  
    - pub  
    - newsagent  
  
Club(fenor):  
  name: fenor  
  sponsors:  
    - newsagent
```

```
Sponsor(newsagent):  
  name: newsagent  
  support:  
    - tramore  
    - fenor  
  
Sponsor(pub):  
  name: pub  
  support:  
    - tramore
```

# Test Strategy

---

- For each relationship:
  - ‘short’ test - quick sanity check
  - ‘long’ test - full exercise of relationship, in both directions if present
  - ‘edit’ test - perform change on objects

# Test Data

---

```
Sponsor(pub):  
  name: pub  
  
Sponsor(newsagent):  
  name: newsagent  
  
Club(tramore):  
  name: tramore  
  sponsors:  
    - pub  
    - newsagent  
  
Club(fenor):  
  name: fenor  
  sponsors:  
    - newsagent  
  
Player(jim):  
  name: jim  
  club: dunmore  
  
Player(mary):  
  name: mary  
  club: dunmore  
  
Player(sam):  
  name: sam  
  club: tramore  
///  
Sponsor(newsagent):  
  name: newsagent  
  support:  
    - tramore  
    - fenor  
  
Sponsor(pub):  
  name: pub  
  support:  
    - tramore
```

```
public class ComprehensiveTest extends UnitTest  
{  
  
    @Before  
    public void setup()  
    {  
        Fixtures.loadModels("data.yml");  
    }  
}
```

# 'Sanity' Tests

---

```
@Test
public void testPlayerClub()
{
    Club    dunmore = Club.find("byName", "dunmore").first();
    Player jim      = Player.find("byName", "jim").first();
    Player mary     = Player.find("byName", "mary").first();
    assertNotNull(mary);

    assertTrue (dunmore.players.contains(jim));
    assertTrue (dunmore.players.contains(mary));
}

@Test
public void testDivisionClub()
{
    Division senior = Division.find("byName", "senior").first();
    Club    dunmore = Club.find("byName", "dunmore").first();
    Club    tramore = Club.find("byName", "tramore").first();

    assertTrue (senior.members.contains(dunmore));
    assertTrue (senior.members.contains(tramore));
}

@Test
public void testClubSponsorShort()
{
    Sponsor newsagent = Sponsor.find("byName", "newsagent").first();
    Club    dunmore   = Club.find("byName", "dunmore").first();
    Club    tramore   = Club.find("byName", "tramore").first();

    assertTrue(newsagent.support.contains(dunmore));
    assertTrue(newsagent.support.contains(tramore));

    assertTrue(dunmore.sponsors.contains(newsagent));
    assertTrue(tramore.sponsors.contains(newsagent));
}
```

# 'Long' Tests

---

```
@Test
public void testPlayerClubLong()
{
    Player jim;
    Club    dunmore;

    jim = Player.find("byName", "jim").first();
    assertNotNull(jim);
    assertEquals(jim.name, "jim");

    dunmore = jim.club;
    assertEquals("dunmore", dunmore.name);

    dunmore = Club.find("byName", "dunmore").first();
    assertNotNull(dunmore);
    assertEquals("dunmore", dunmore.name);
    assertEquals(2, dunmore.players.size());

    Player p1 = dunmore.players.get(0);
    assertTrue (p1.name.equals("jim") || p1.name.equals("mary"));
    Player p2 = dunmore.players.get(1);
    assertTrue (p2.name.equals("jim") || p2.name.equals("mary"));
}

@Test
public void testDivisionClubLong()
{
    Division senior = Division.find("byName", "senior").first();
    assertNotNull(senior);
    assertEquals(2, senior.members.size());

    Club c1 = senior.members.get(0);
    Club c2  = senior.members.get(1);

    assertTrue (c1.name.equals("tramore") || c1.name.equals("dunmore"));
    assertTrue (c2.name.equals("tramore") || c2.name.equals("dunmore"));
}
```

# 'Edit' Tests

---

```
@Test
public void testEditPlayerClub()
{
    Club    dunmore = Club.find("byName", "dunmore").first();
    Player jim      = Player.find("byName", "jim").first();
    Player mary     = Player.find("byName", "mary").first();

    dunmore.players.remove(mary);
    mary.delete();
    dunmore.save();

    assertEquals (dunmore.players.size(), 1);
    assertTrue (dunmore.players.contains(jim));

    assertEquals(0, Player.find("byName", "mary").fetch().size());

    Player sara      = new Player("sara");
    dunmore.addPlayer(sara);
    dunmore.save();
    assertEquals (dunmore.players.size(), 2);
}

@Test
public void testEditClubSponsor()
{
    Sponsor newsagent = Sponsor.find("byName", "newsagent").first();
    Club    dunmore   = Club.find("byName", "dunmore").first();

    assertEquals(2, newsagent.support.size());

    newsagent.support.remove(dunmore);
    dunmore.sponsors.remove(newsagent);

    newsagent.save();
    dunmore.save();

    assertEquals(1, newsagent.support.size());
}
```

# Exploring the Database

```
Sponsor(pub):
  name: pub

Sponsor(newsagent):
  name: newsagent

Club(dunmore):
  name: dunmore

Club(tramore):
  name: tramore
  sponsors:
    - pub
    - newsagent

Club(fenor):
  name: fenor
  sponsors:
    - newsagent

Division(senior):
  name: senior
  members:
    - tramore
    - dunmore

Division(junior):
  name: junior
  members:
    - fenor

Sponsor(newsagent):
  name: newsagent
  support:
    - tramore
    - fenor

Sponsor(pub):
  name: pub
  support:
    - tramore
```

```
Player(jim):
  name: jim
  club: dunmore

Player(mary):
  name: mary
  club: dunmore

Player(sam):
  name: sam
  club: tramore

Player(john):
  name: john
  club: tramore

Player(mike):
  name: mike
  club: fenor

Player(linda):
  name: linda
  club: fenor
```

```
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
```



# Player & Club

```
Club(dunmore):  
  name: dunmore  
  
Club(tramore):  
  name: tramore  
  
Club(fenor):  
  name: fenor
```

```
Player(jim):  
  name: jim  
  club: dunmore  
  
Player(mary):  
  name: mary  
  club: dunmore  
  
Player(sam):  
  name: sam  
  club: tramore  
  
Player(john):  
  name: john  
  club: tramore  
  
Player(mike):  
  name: mike  
  club: fenor  
  
Player(linda):  
  name: linda  
  club: fenor
```

SELECT \* FROM CLUB;

ID	NAME
1	dunmore
2	tramore
3	fenor

(3 rows, 2 ms)

SELECT \* FROM PLAYER;

ID	NAME	CLUB_ID
1	jim	1
2	mary	1
3	sam	2
4	john	2
5	mike	3
6	linda	3

(6 rows, 5 ms)

# Club & Division

```
Club(dunmore):  
  name: dunmore  
  
Club(tramore):  
  name: tramore  
  
Club(fenor):  
  name: fenor  
  
Division(senior):  
  name: senior  
  members:  
    - tramore  
    - dunmore  
  
Division(junior):  
  name: junior  
  members:  
    - fenor
```

SELECT \* FROM CLUB;

ID	NAME
1	dunmore
2	tramore
3	fenor

(3 rows, 2 ms)

SELECT \* FROM DIVISION;

ID	NAME
1	senior
2	junior

(2 rows, 2 ms)

SELECT \* FROM DIVISION\_CLUB;

DIVISION_ID	MEMBERS_ID
1	2
1	1
2	3

(3 rows, 3 ms)

# Sponsor & Club

Sponsor(pub):  
  name: pub

Sponsor(newsagent):  
  name: newsagent

Club(dunmore):  
  name: dunmore

Club(tramore):  
  name: tramore  
  sponsors:  
    - pub  
    - newsagent

Club(fenor):  
  name: fenor  
  sponsors:  
    - newsagent

Sponsor(newsagent):  
  name: newsagent  
  support:  
    - tramore  
    - fenor

Sponsor(pub):  
  name: pub  
  support:  
    - tramore

```
SELECT * FROM SPONSOR;
```

ID	NAME
1	pub
2	newsagent

(2 rows, 1 ms)

Edit

```
SELECT * FROM CLUB;
```

ID	NAME
1	dunmore
2	tramore
3	fenor

(3 rows, 2 ms)

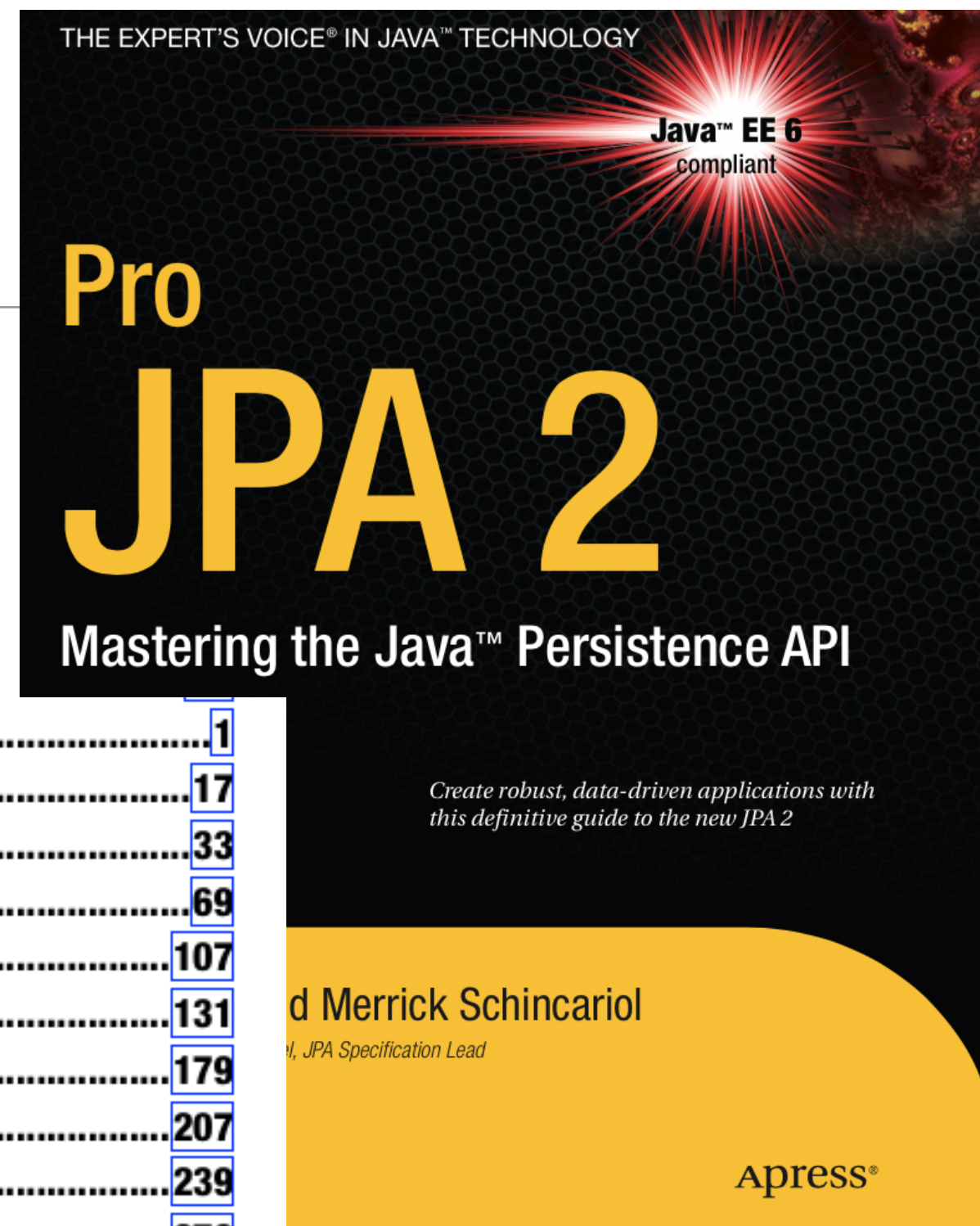
```
SELECT * FROM CLUB_SPONSOR;
```

SUPPORT_ID	SPONSORS_ID
2	1
2	2
3	2

(3 rows, 3 ms)

# Definitive Reference

■ Chapter 1: Introduction .....	1
■ Chapter 2: Getting Started.....	17
■ Chapter 3: Enterprise Applications .....	33
■ Chapter 4: Object-Relational Mapping .....	69
■ Chapter 5: Collection Mapping .....	107
■ Chapter 6: Entity Manager .....	131
■ Chapter 7: Using Queries.....	179
■ Chapter 8: Query Language .....	207
■ Chapter 9: Criteria API .....	239
■ Chapter 10: Advanced Object-Relational Mapping.....	273
■ Chapter 11: Advanced Topics.....	315
■ Chapter 12: XML Mapping Files .....	371
■ Chapter 13: Packaging and Deployment .....	407
■ Chapter 14: Testing .....	429
■ Chapter 15: Migration .....	457
■ Index .....	481

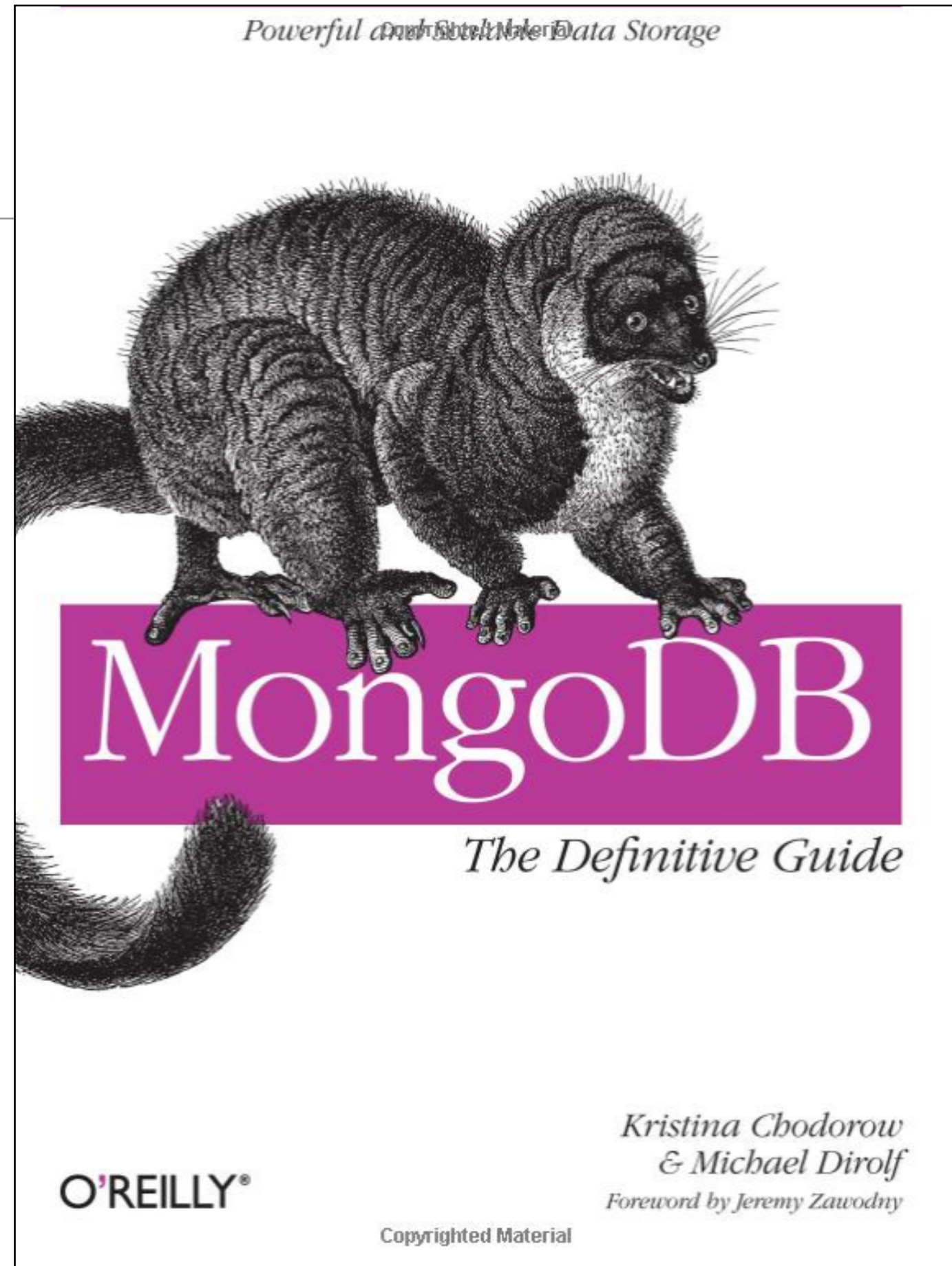


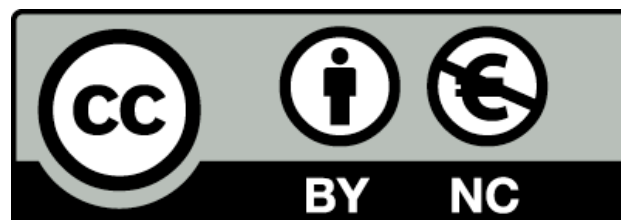


# Alternatives?

---

- NoSQL -
  - Document Oriented
  - Simpler for many use cases
  - Less powerful in some ways  
“aggregates” instead of  
“relations”
  - Faster and easier to scale





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

