

---

# Java EE 7 Web

## 網路應用程式與資料庫的整合

---

鄭安翔

[ansel\\_cheng@hotmail.com](mailto:ansel_cheng@hotmail.com)

# 課程大綱

## 1) **Model 元件設計**

- 領域元件 **Domain Entity**
- 服務元件 **Service Object**
- **DAO Design Pattern** 設計模式

## 2) **JavaDB Derby 資料庫設定**

## 3) **設計整合DBMS 的網路應用程式**

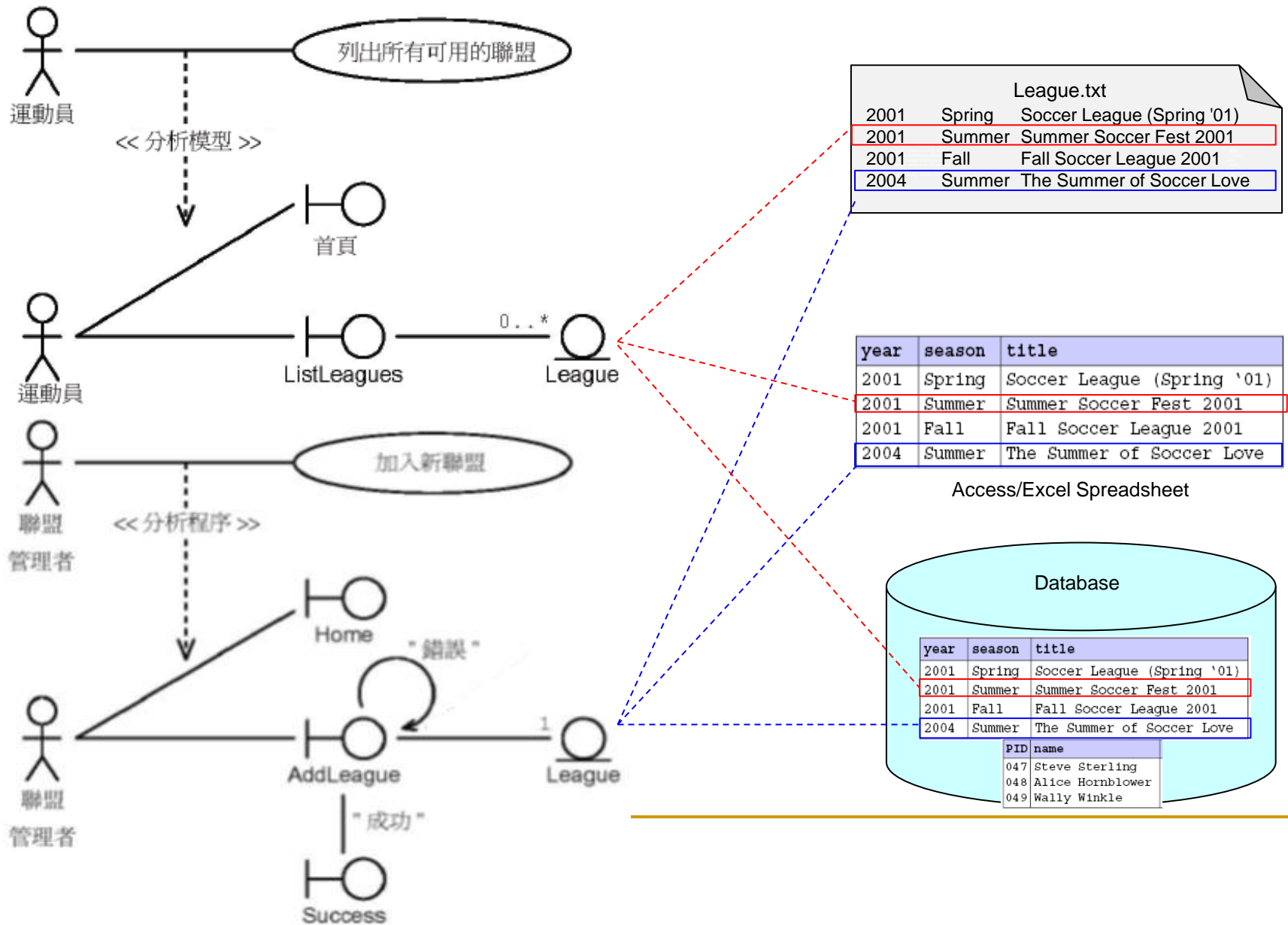
# 模型(Model)設計

- MVC 架構中 模型（Model）元件
  - 接受控制器的請求呼叫
    - 提供控制器操作介面
  - 處理商業邏輯
    - 商業流程邏輯 (服務元件 Service Object)
    - 商業實體邏輯 (領域元件 Domain Entity)
    - 資料存取邏輯(資料處理元件Data Access Object)
  - 提供視圖組件當前資料狀態
    - 領域元件可使用 **JavaBeans**規範,便於**JSP**元件以非文稿元素操作
      - EL
      - Standard Action
      - Tag Library

# 領域元件 Domain Entity

- 領域元件 (Domain Entity)
  - 代表真實世界的商業物件
  - 通常為永續性的主要資料
  - 包含複雜的物件導向行為
- 可用多種Java技術實作
  - POJO : Plain Old Java Object
    - Java Beans
  - EJB Entity Bean
  - CORBA 物件

# 領域元件 Entity - 永續性的儲存

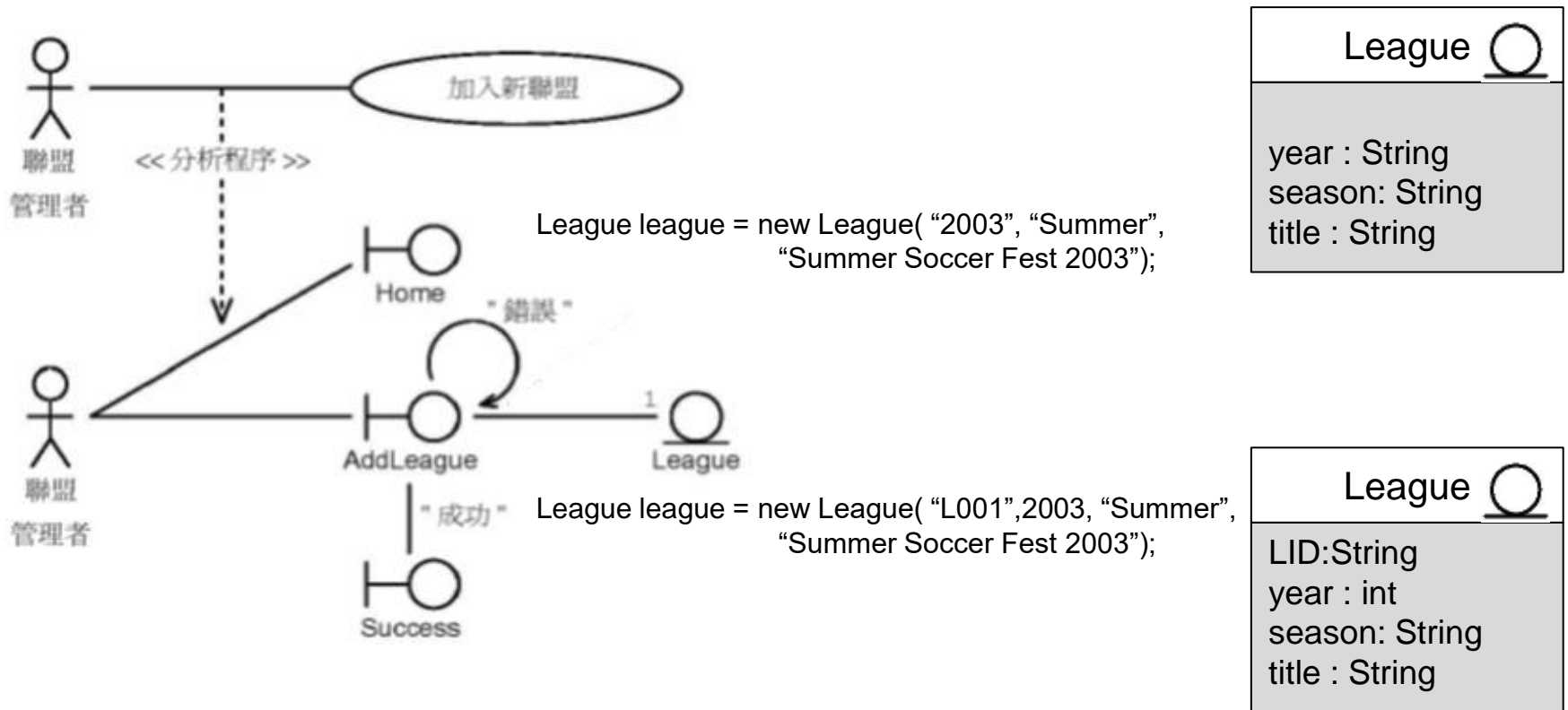


# 服務元件 Service Object

- 服務元件 Service Object
  - 邊界(View/Controller)元件及領域元件間加入服務層
  - 降低邊界元件與領域元件的耦合性 (Coupling)
  - 提供與領域元件相關，但不能由其本身執行的操作
    - 建立 Creation -建立領域元件新實例
    - 取得 Retrieval -自永續資料中取得唯一的實例
    - 選擇 Selection -自永續資料中取得一組實例
    - 聚合 Aggregation -在一組實例中執行計算(ex:取平均數)
    - 刪除 Deletion -自永續資料中移除實例
  - POJO (Plain Old Java Object) 實作

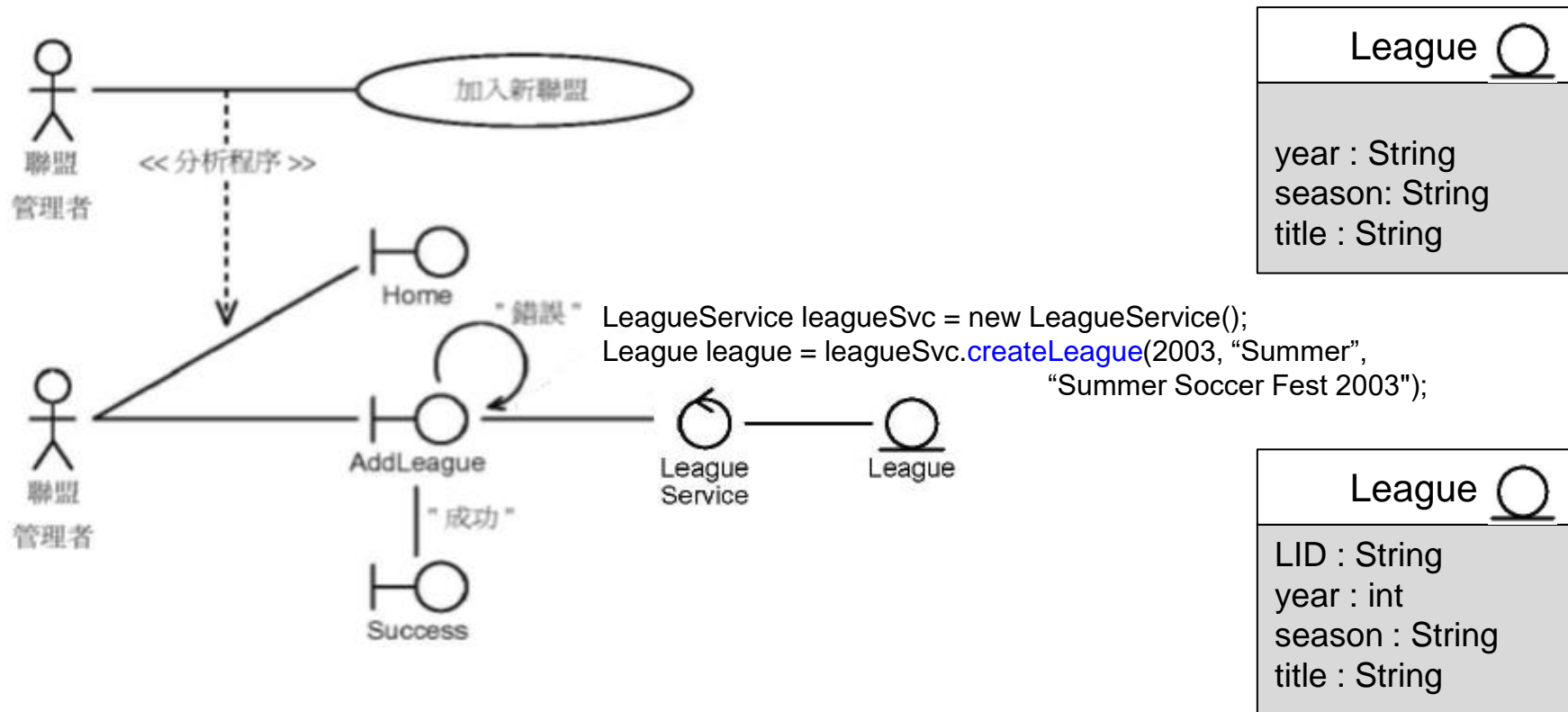
# 服務元件

- 無服務元件前，由邊界元件建立領域元件
  - 領域元件變動，邊界元件需修改



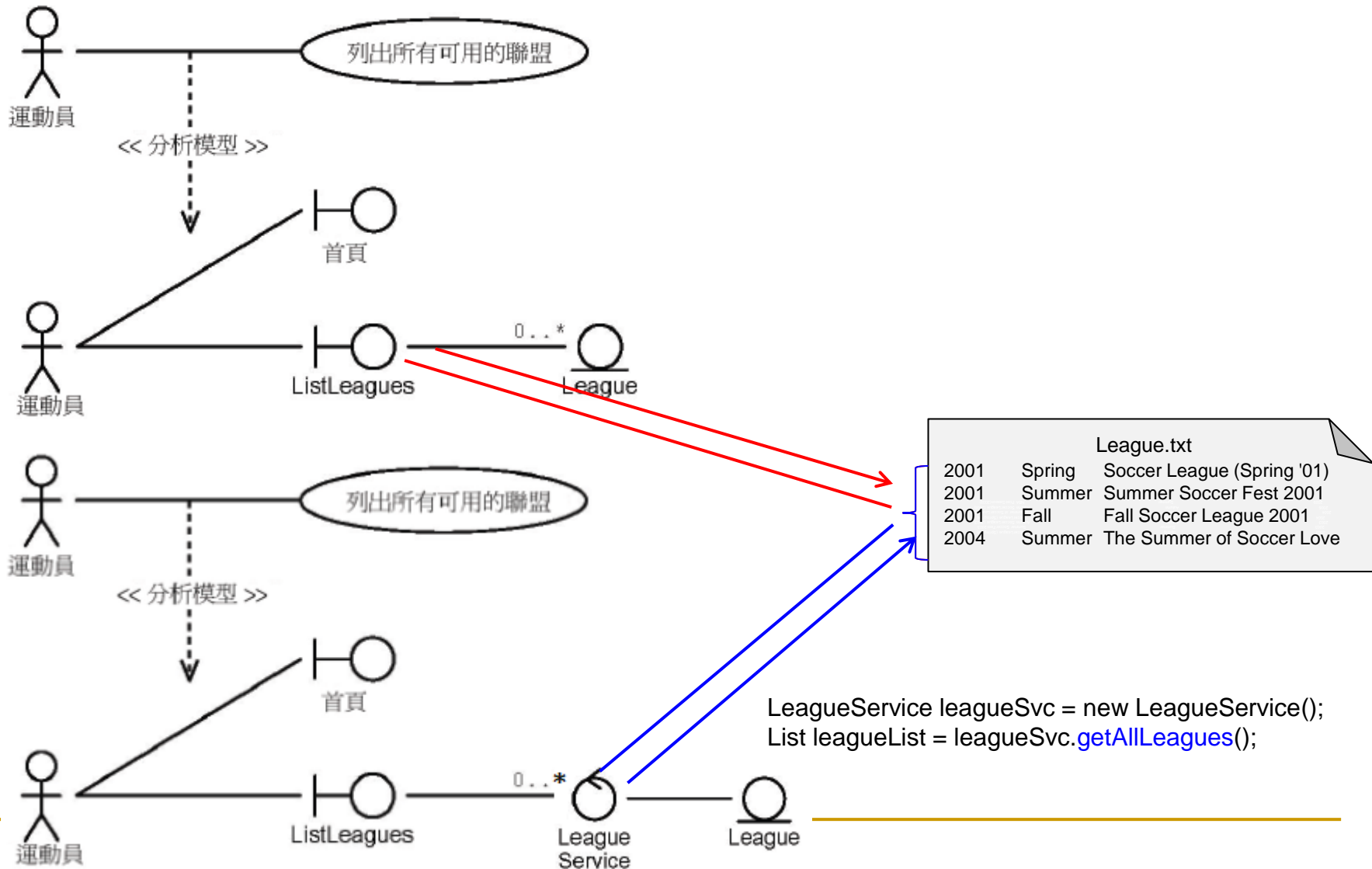
# 服務元件 - 建立領域元件

- 加入服務元件，用其建立領域元件，將邊界元件與領域元件解耦 (de-couple)

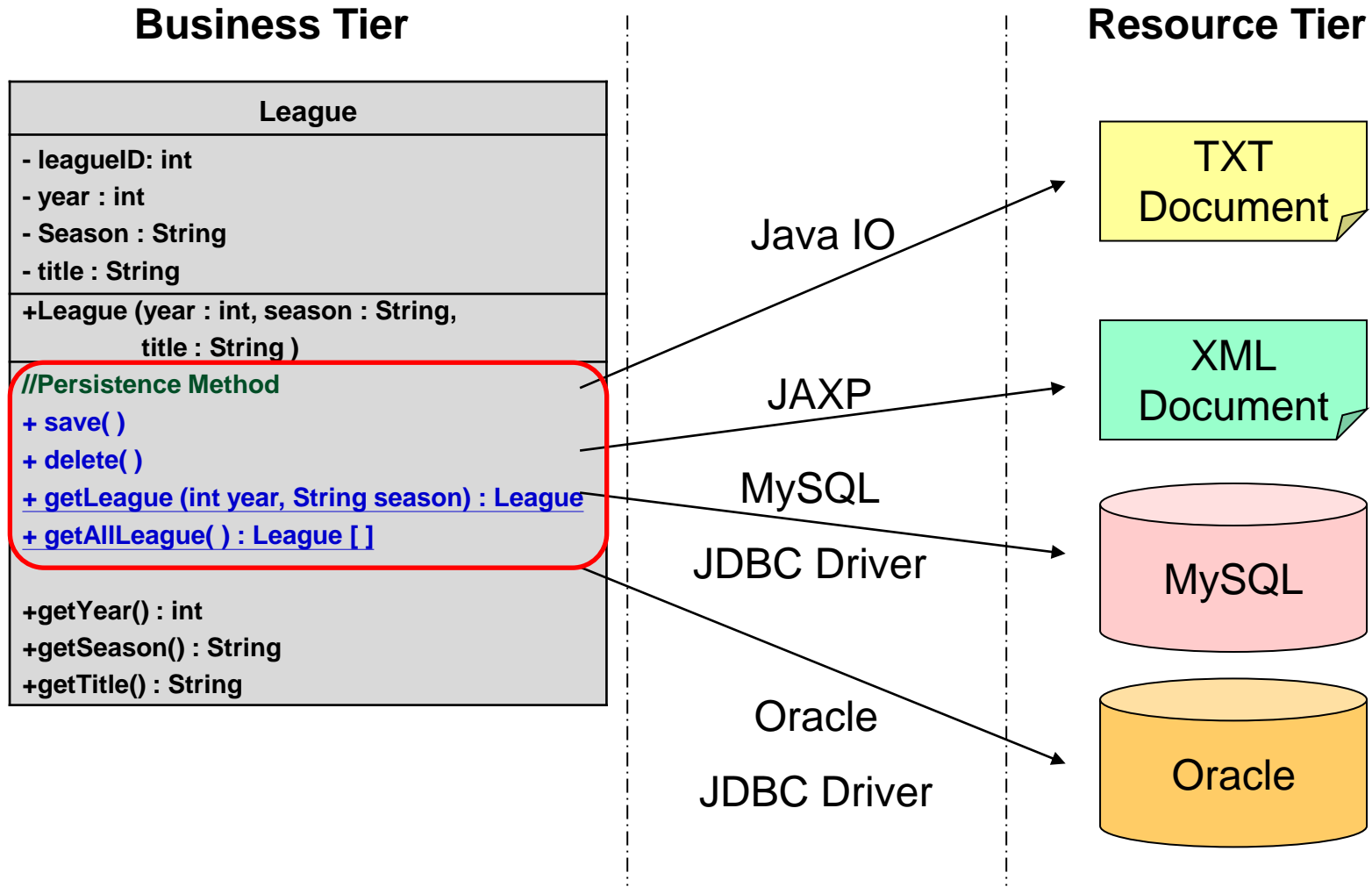




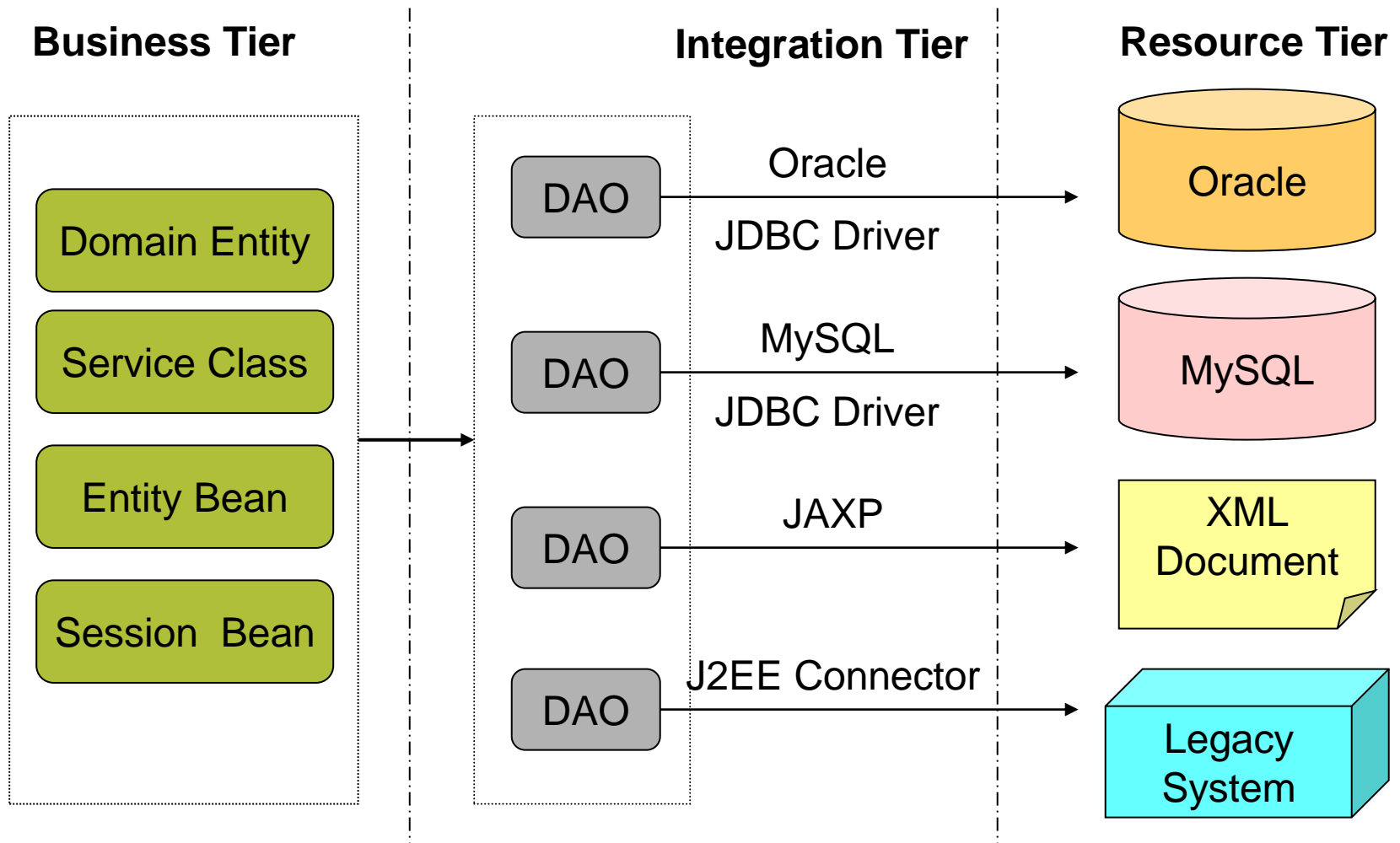
# 服務元件 – 取得領域元件集合



# Before DAO Pattern



# DAO Pattern



# DAO Design Pattern

## ■ 優點

- 將商業邏輯與資料存取邏輯分離
  - 商業邏輯元件改變,不影響資料存取元件
- 簡化存取多種資料來源的方式
  - 當資料來源改變,只需修改**DAO**物件,不影響商業邏輯元件
  - 不同資料來源撰寫獨立的**DAO**物件,提供一致的資料存取方法
- 資料存取邏輯可重複使用
- 集中管理資料存取邏輯

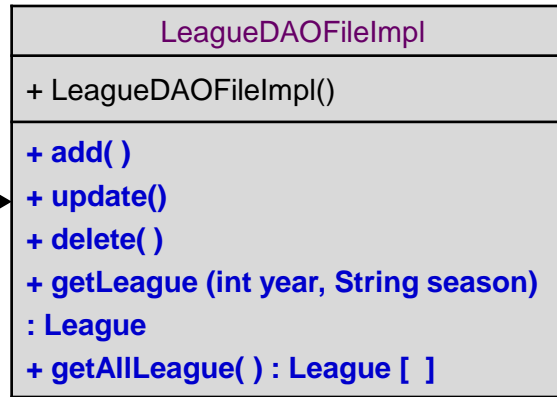
# DAO Design Pattern

## Business Tier

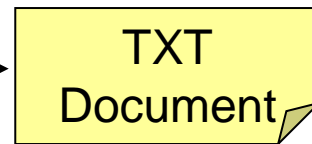
## Integration Tier

## Resource Tier

```
public Class LeagueService {  
    .....  
  
    LeagueDAOFileImpl dao =  
    new LeagueDAODBImpl();  
  
    .....  
}
```

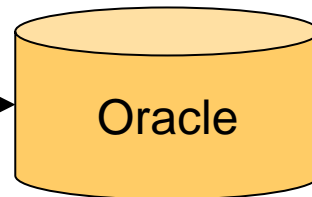


Java IO

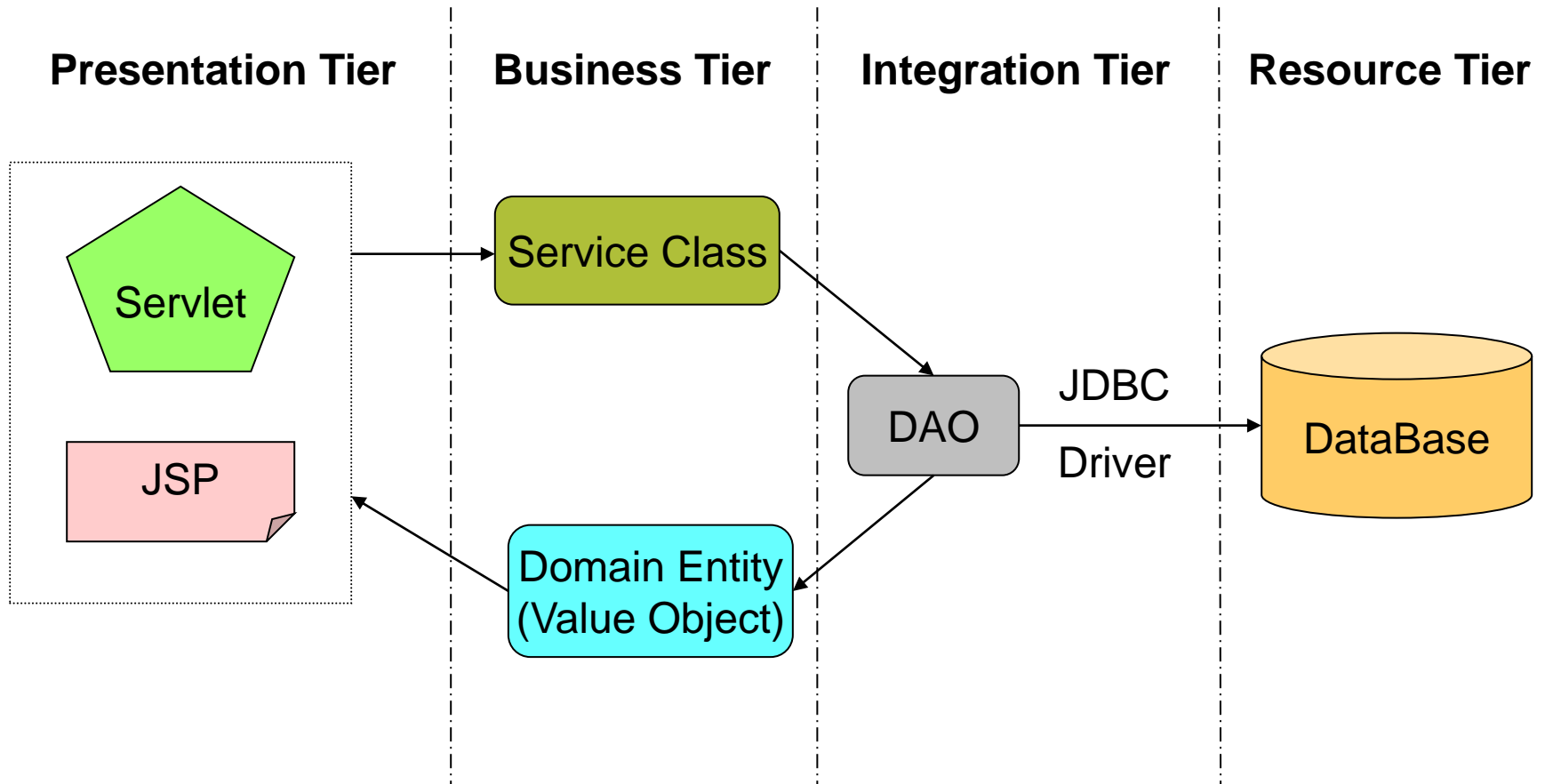


Oracle

JDBC  
Driver



# Service / Entity Object + DAO



# 課程大綱

- 1) Model 元件設計
- 2) **JavaDB Derby 資料庫設定**
  - 連接**JavaDB Derby** 資料庫
  - 建立資料表
- 3) 設計整合**DBMS** 的網路應用程式

# 下載安裝 JavaDB Derby

[https://db.apache.org/derby/derby\\_downloads.html](https://db.apache.org/derby/derby_downloads.html)

The screenshot shows the Apache Derby Downloads page in a web browser. The page title is "Apache Derby: Downloads" and the URL is "db.apache.org/derby/derby\_downloads.html". The page features a navigation menu with links to Home, Quick Start, Download, Community, Documentation, and Resources. The "Download" section is active, showing a list of download links for different Java versions: For Java 9 and Higher, For Java 8 and Higher, For Java 6 and Higher, For Java 1.4 and Higher, For Java 1.3 and Higher, Deprecated Releases, and Change History. Below this, there are three sections: "For Java 9 and Higher" with a link to 10.15.1.3 (March 5, 2019 / SVN 1853019), "For Java 8 and Higher" with links to 10.14.2.0 (May 3, 2018 / SVN 1828579) and 10.13.1.1 (October 25, 2016 / SVN 1766613), and "For Java 6 and Higher".

Overlaid on the right side of the screenshot are two file explorer windows. The top window, titled "ClassSetup", shows a table of files:

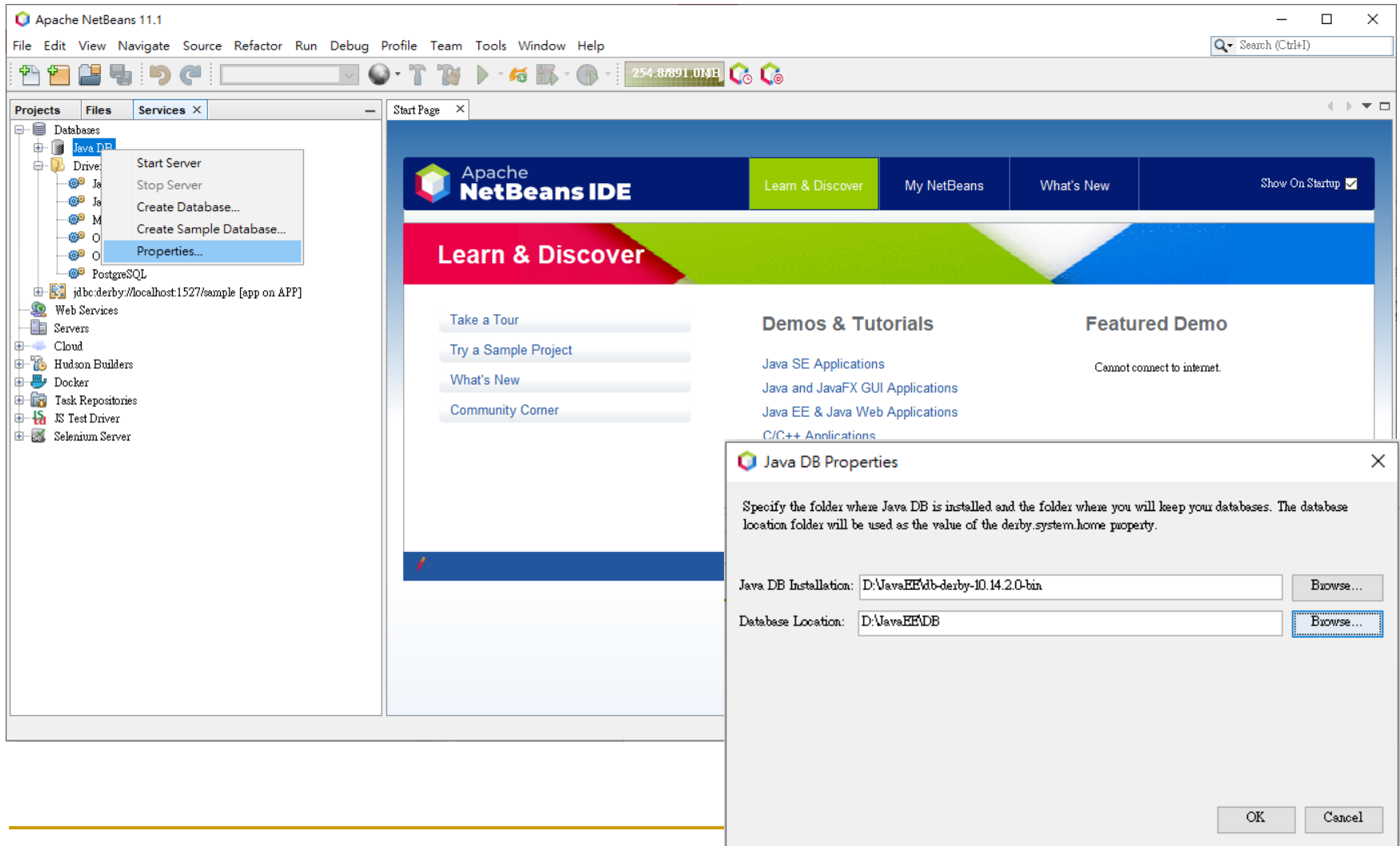
名稱	修改日期	類型	大小
db-derby-10.14.2.0-bin.zip	2019/11/7 下午 09:...	壓縮的 (zipped) ...	20,664 KB

The bottom window, titled "db-derby-10.14.2.0-bin", shows the contents of the extracted zip file:

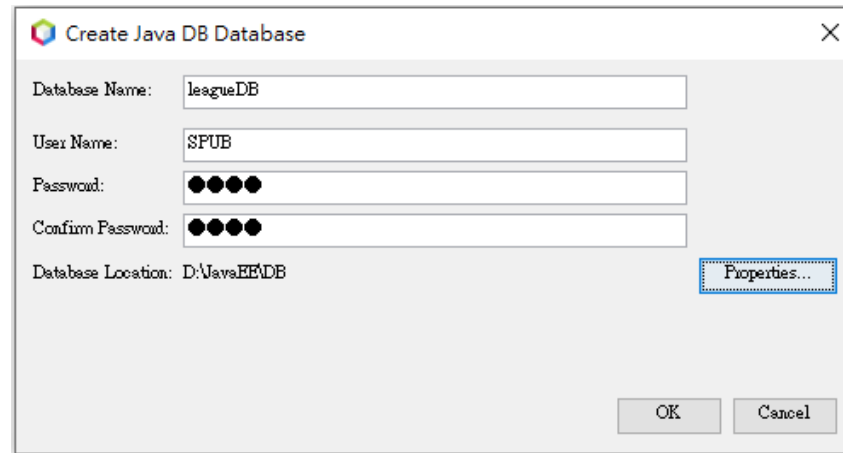
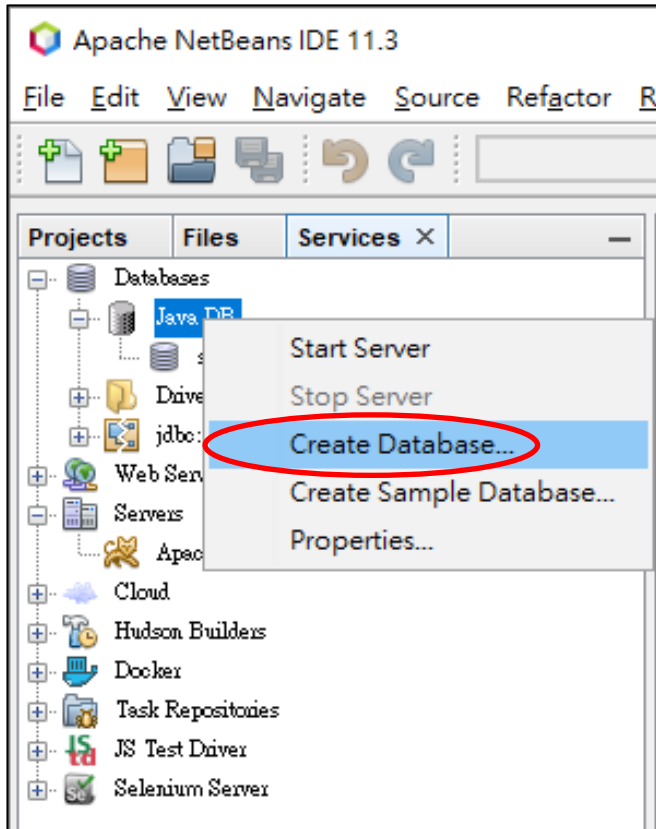
名稱	修改日期	類型	大小
bin	2019/12/11 下午 10:48	檔案資料夾	
demo	2019/12/11 下午 10:48	檔案資料夾	
docs	2019/12/11 下午 10:49	檔案資料夾	
javadoc	2019/12/11 下午 10:49	檔案資料夾	
lib	2019/12/11 下午 10:49	檔案資料夾	
test	2019/12/11 下午 10:49	檔案資料夾	
index.html	2018/3/10 上午 08:31	Chrome HT...	5 KB
KEYS	2018/4/6 下午 06:14	檔案	47 KB
LICENSE	2018/4/6 下午 06:14	檔案	12 KB
NOTICE	2018/4/6 下午 06:14	檔案	13 KB
RELEASE-NOTES.html	2018/4/6 下午 06:14	Chrome HT...	7 KB



# 連接JavaDB Derby 資料庫

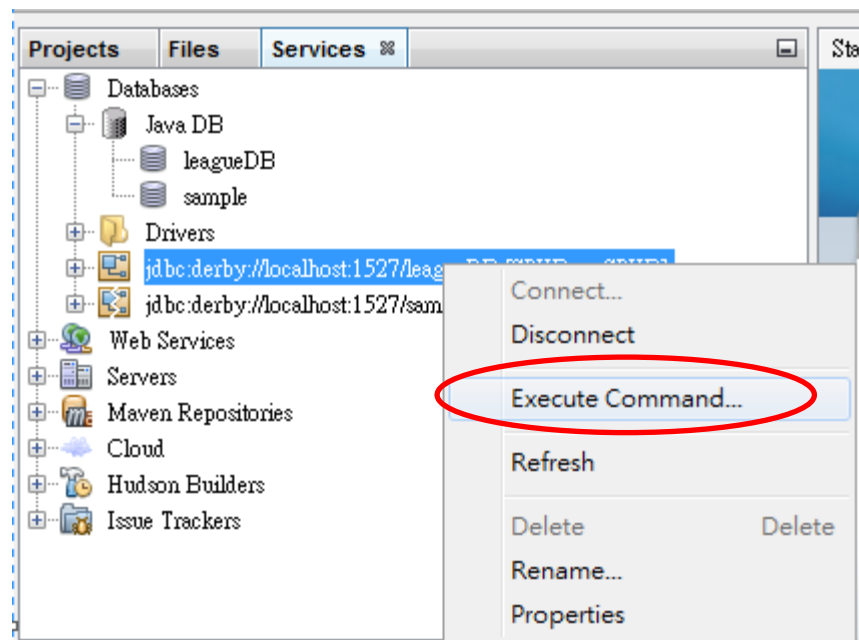
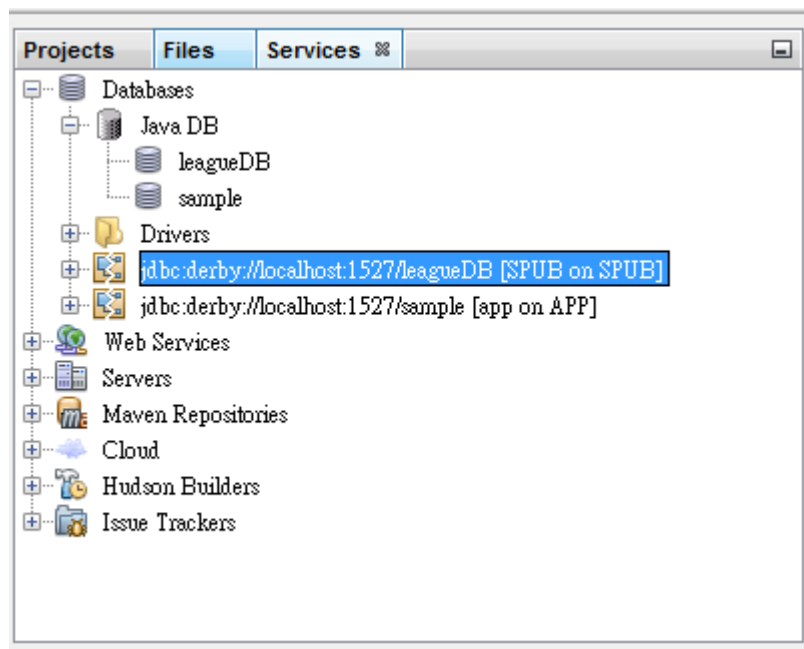


# 建立leagueDB資料庫



User Name : SPUB  
Password: SPUB

# 建立資料表



# 建立leagueDB Schema

- 複製schema.sql
  - 建立League, ObjectIDs, Player, Registration四個資料表

The screenshot displays a database IDE interface. The top toolbar includes a red circle around the 'Execute' button (a green arrow pointing right). Below the toolbar, the 'SQL Command 1' window shows the following SQL script:

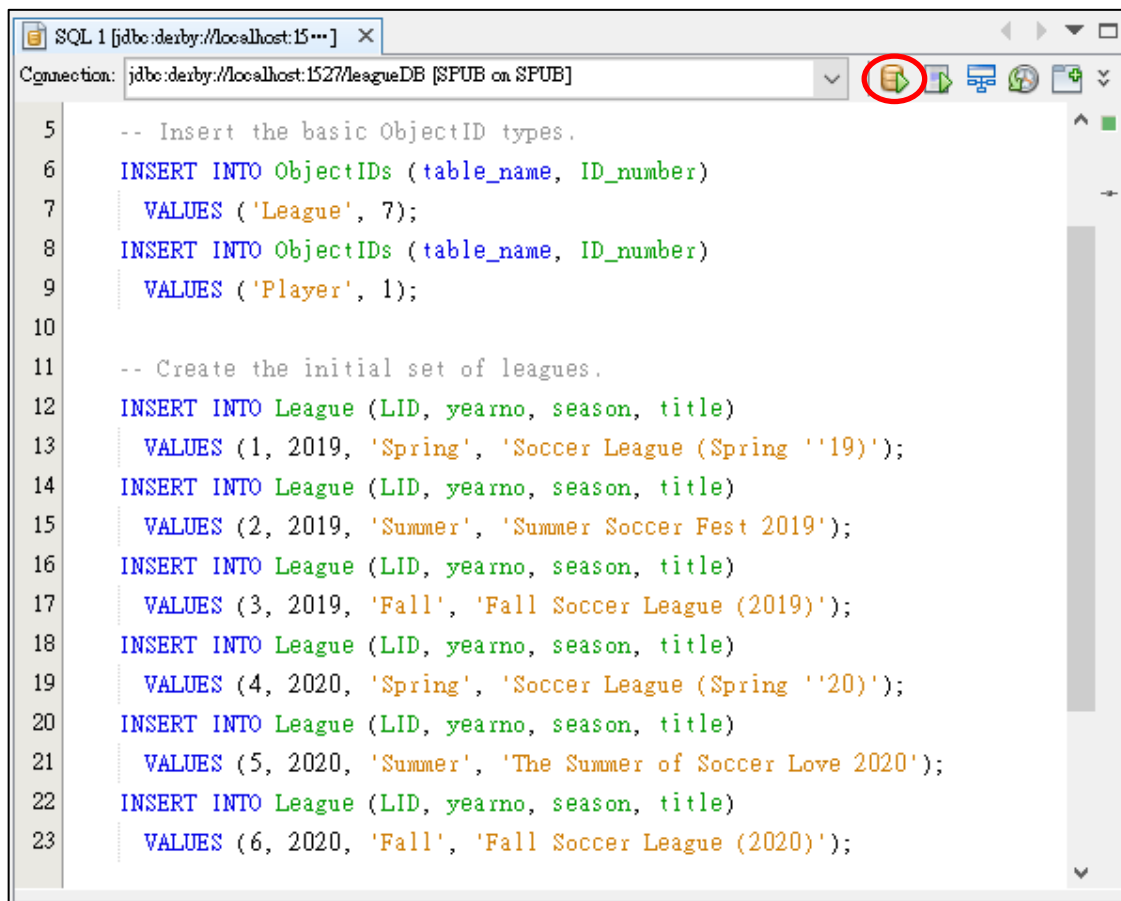
```
6 DROP TABLE Registration;
7 DROP TABLE Player;
8 DROP TABLE League;
9 DROP TABLE ObjectIDs;
10
11
12 --
13 -- This table represents the "next" object_ID for a given table
14 --
15 CREATE TABLE ObjectIDs (
16 -- PRIMARY KEY --
17 table_name VARCHAR(30) PRIMARY KEY,
18 -- DATA FIELDS --
19 ID_number INTEGER NOT NULL
```

On the right side, the 'Projects' pane shows a tree structure of the database. The 'leagueDB' database is selected, and its 'Tables' folder is expanded. A red circle highlights the four tables: LEAGUE, OBJECTIDS, PLAYER, and REGISTRATION.

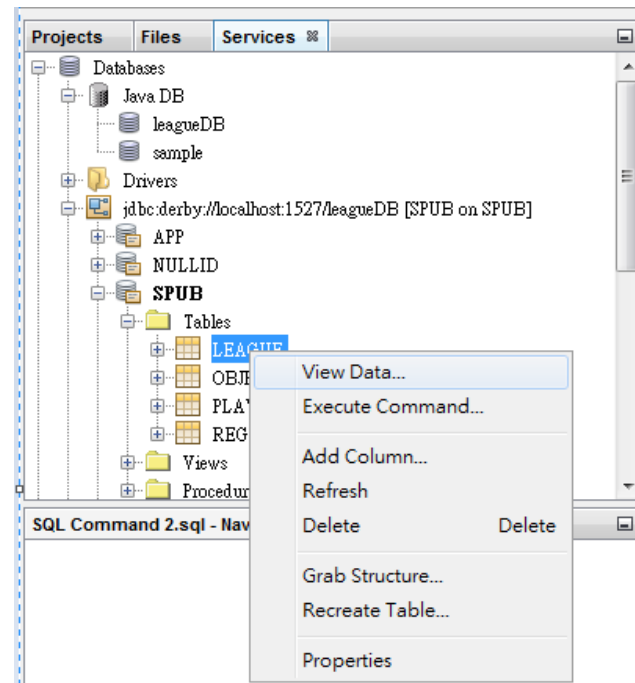
# 建立leagueDB 初始資料

## ■ 複製init\_data.sql

- 建立新增2筆資料至ObjectIDs及6筆資料至League



```
SQL 1 [jdbc:derby://localhost:15...]  
Connection: jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]  
  
5  -- Insert the basic ObjectID types.  
6  INSERT INTO ObjectIDs (table_name, ID_number)  
7     VALUES ('League', 7);  
8  INSERT INTO ObjectIDs (table_name, ID_number)  
9     VALUES ('Player', 1);  
10  
11  -- Create the initial set of leagues.  
12  INSERT INTO League (LID, yearno, season, title)  
13     VALUES (1, 2019, 'Spring', 'Soccer League (Spring '19)');  
14  INSERT INTO League (LID, yearno, season, title)  
15     VALUES (2, 2019, 'Summer', 'Summer Soccer Fest 2019');  
16  INSERT INTO League (LID, yearno, season, title)  
17     VALUES (3, 2019, 'Fall', 'Fall Soccer League (2019)');  
18  INSERT INTO League (LID, yearno, season, title)  
19     VALUES (4, 2020, 'Spring', 'Soccer League (Spring '20)');  
20  INSERT INTO League (LID, yearno, season, title)  
21     VALUES (5, 2020, 'Summer', 'The Summer of Soccer Love 2020');  
22  INSERT INTO League (LID, yearno, season, title)  
23     VALUES (6, 2020, 'Fall', 'Fall Soccer League (2020)');
```



# 檢視資料

SQL 1 [jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]]

SQL 2 [jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]]

Connection: jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]

```
1 SELECT * FROM SPUB.LEAGUE FETCH FIRST 100 ROWS ONLY;
```

```
2
```

SELECT \* FROM SPUB.LEAGUE...

Max. rows: 100

Fetched Rows: 6

Matching Rows:

#	LID	YEARNO	SEASON	TITLE
1	1	2019 Spring	Soccer League (Spring '19)	
2	2	2019 Summer	Summer Soccer Fest 2019	
3	3	2019 Fall	Fall Soccer League (2019)	
4	4	2020 Spring	Soccer League (Spring '20)	
5	5	2020 Summer	The Summer of Soccer Love 2020	
6	6	2020 Fall	Fall Soccer League (2020)	

SQL 2 [jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]]

Connection: jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]

```
1 SELECT * FROM SPUB.OBJECTIDS FETCH FIRST 100 ROWS ONLY;
```

SELECT \* FROM SPUB.OBJECT...

Max. rows: 100 | Fetched Rows: 2 | Matching Rows:

#	TABLE_NAME	ID_NUMBER
1	League	7
2	Player	1

SQL 3 [jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]]

Connection: jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]

```
1 SELECT * FROM SPUB.PLAYER FETCH FIRST 100 ROWS ONLY;
```

SELECT \* FROM SPUB.PLAYER...

Max. rows: 100 | Fetched Rows: 0 | Matching Rows:

#	PID	NAME	ADDRESS
---	-----	------	---------

SQL 4 [jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]]

Connection: jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]

```
1 SELECT * FROM SPUB.REGISTRATION FETCH FIRST 100 ROWS ONLY;
```

SELECT \* FROM SPUB.REGIST...

Max. rows: 100 | Fetched Rows: 0 | Matching Rows:

#	LID	PID
---	-----	-----

# Lab 1-1 Soccer 專案整合資料庫

## 1. leagueDB資料庫建立及設定

- ❑ 下載安裝 JavaDB Derby
- ❑ Derby建立leagueDB資料庫
  - User Name : SPUB
  - Password: SPUB
- ❑ 執行schema.sql
  - 建立League, ObjectIDs, Player,Registration四個資料表
- ❑ 執行init\_data.sql
  - 新增2筆資料至ObjectIDs及6筆資料至League
- ❑ 驗證資料

# 課程大綱

- 1) Model 元件設計
- 2) JavaDB Derby 資料庫設定
- 3) 設計整合**DBMS** 的網路應用程式
  - 使用**JDBC API**直接取得資料庫連線
  - 定義**DataSource** 和**Java Naming and Directory Interface™ (JNDI)API**
  - **Soccer**專案實作**DBMS**整合



# 傳統連接資料庫的方法

- 使用java.sql.DriverManager的getConnection()
  - 資料庫效能問題
  - 資料庫連結資訊hardcode在程式碼中
  - 先建立資料庫連線，將這個連線保存在servlet 成員變數中，有多執行緒同步問題
- 建立資料庫連線池，servlet裡使用連線池存取
  - 需要自行建立或購買連線池的機制
  - 將連線池存於ServletContext屬性，只能被網路層元件存取(Servlet 類別)
  - 商業層元件(Service或DAO元件)無法使用連線池存取資料庫

# Java EE 資料庫連接解決方案

## ■ Java EE 資料庫連接解決方案

### □ Tomcat 內建 DBCP Connection Pool

- 解決資料庫效能問題

### □ DataSource 資源物件

- 將連接資料庫相關資料封裝成DataSource
- 使用描述檔來設定DataSource資訊

- 資料庫更換時只要修改描述檔,程式內容不變

### □ 使用 JNDI 名稱服務取得DataSource

- 商業層元件(Service或DAO元件) 也可以存取資料庫

# 連線池 Connection Pool

## ■ Problem

- ❑ 客戶提出請求時,才建立所需資料庫連線,服務完成後釋放連線
- ❑ 每個請求都建立資料庫連線相當耗費時間/資源
- ❑ 資料庫通常允許的連線數較Web Server少

## ■ Solution

- ❑ 預先建立數個資料庫連線置於Connection Pool中,供整個Application共用
- ❑ 需要Connection的程式,自Pool取得Connection
- ❑ 服務完成後,將Connection放回Pool

# DataSource 資源物件

- `javax.sql.DataSource` interface
  - 將連接資料庫相關資料封裝成**DataSource**資源物件
    - 資料庫的URL
    - JDBC 驅動程式(JDBC Driver)
    - 使用者帳號及密碼
  - 減少將資料庫連結資訊**hardcode**在程式碼中
  - 包含容器提供之連線池
    - Tomcat 內建 DBCP Connection Pool
  - 程式碼可以跨應用程式伺服器運作

# JNDI 名稱與目錄服務

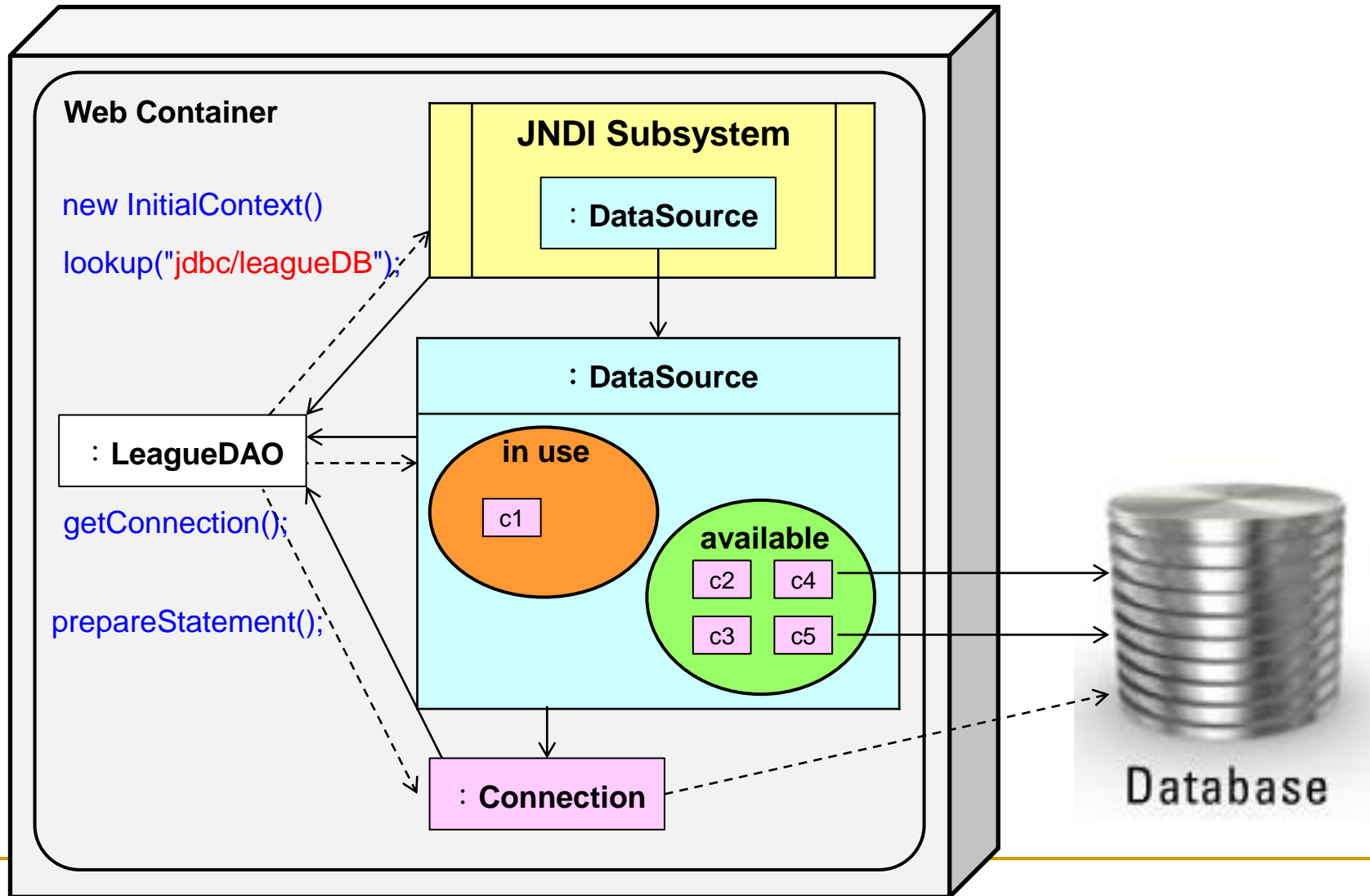
- 名稱與目錄服務
  - JavaEE 應用程式伺服器皆提供此服務
  - 使用JNDI API 來查詢
  - 商業層元件透過JNDI來存取 DataSource
    - Service 或 DAO元件通常為 POJO 物件
    - 不受無法取得 ServletContext 物件的限制

# 元件取得資料庫連線步驟

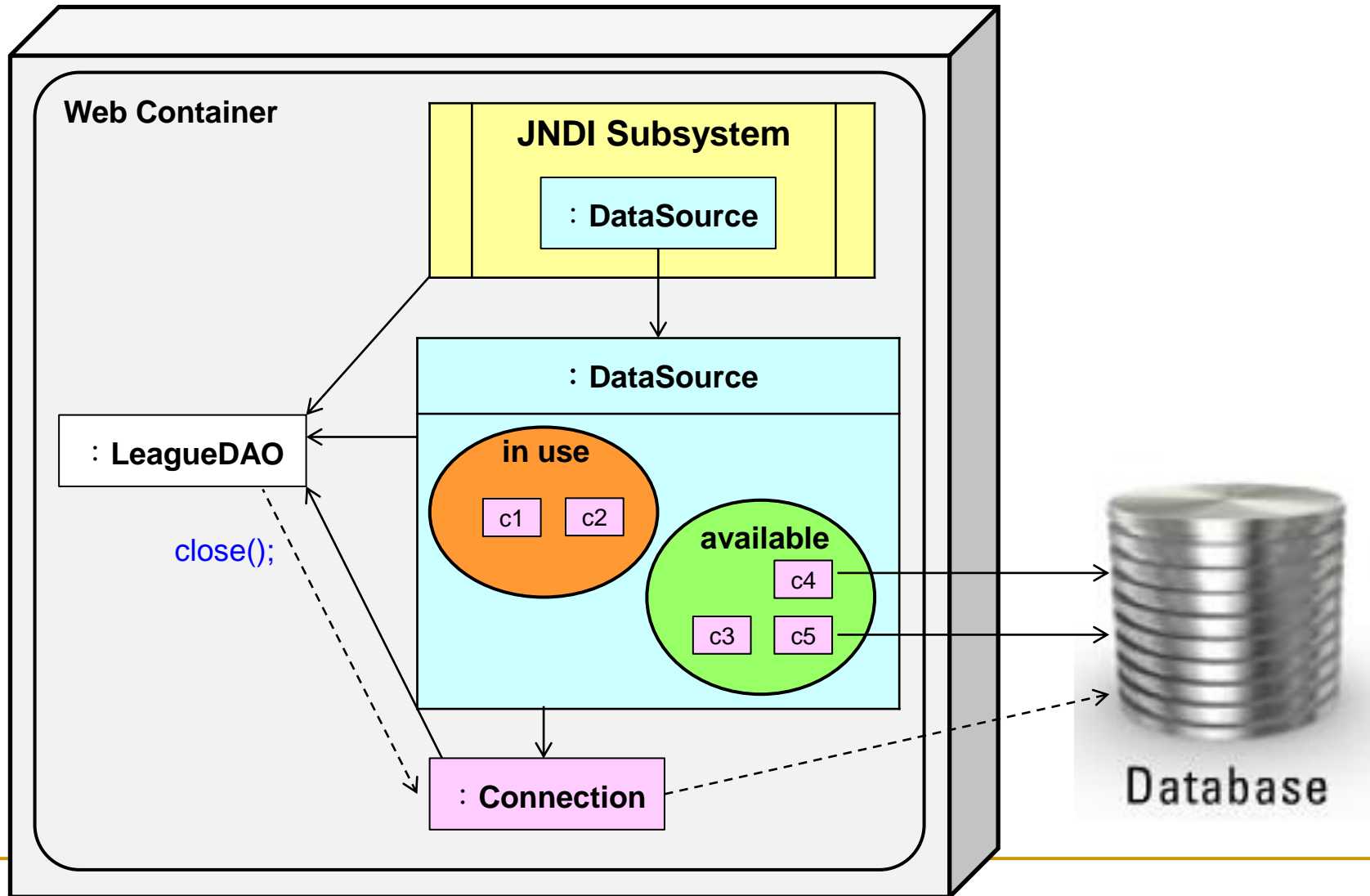
## ■ 元件取得資料庫連線步驟

1. 執行**JNDI** 查詢名稱與目錄服務來取得**DataSource**
  - 取得**JNDI**物件
  - **JNDI**物件搜尋**DataSource**資源
2. 取得資料庫連線 **Coonection**
  - 呼叫**DataSource** 上的**getConnection** 方法
3. 透過資料庫連線，使用**JDBC API** 和資料庫溝通

# 在應用程式裡使用DataSource



# 關閉連線並送回到DataSource





# 設定及使用DataSource

- 安裝JDBC Driver
  - <Tomcat\_Home>\lib 或webapps\[context]\WEB-INF\lib
  - Netbeans 專案中 Libraries 資料夾
- 定義應用程式資源Context Resource
  - webapps\[context]\META-INF\context.xml中設定
  - Netbeans 專案configuration files資料夾下context.xml
- web.xml中設定要參照之資源 Resource
- 程式中使用DataSource

# JDBC Driver

Apache NetBeans IDE 11.3

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+D)

326.9/641.0 MB

Project Files Services

Soccer

- Web Pages
  - META-INF
    - context.xml
  - WEB-INF
    - web.xml
  - admin
    - enter\_player.jsp
    - index.html
    - list\_leagues.jsp
    - select\_league.jsp
    - thank\_you.jsp
  - Source Packages
    - controller
      - AddLeague.java
      - EnterPlayer.java
      - ListLeagues.java
      - SelectLeague.java
    - model
      - League.java
      - LeagueDAO.java
      - LeagueService.java
      - ObjectIdDAO.java
      - Player.java
      - RegistrationService.java
    - web
  - Libraries
    - JSTL 1.2.1 - jstl-impl.jar
    - JSTL 1.2.1 - jstl-api.jar
    - derbyLocale\_zh\_TW.jar
    - derbyclient.jar
    - JDK 11 (Default)
    - Apache Tomcat or TomEE
  - Configuration Files
    - MANIFEST.MF
    - context.xml
    - web.xml

context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/Soccer">
  <Resource name="jdbc/leagueDB"
    auth="Container"
    type="javax.sql.DataSource"
    username="SPUB"
    password="SPUB"
    driverClassName="org.apache.derby.jdbc.ClientDriver"
    url="jdbc:derby://localhost:1527/leagueDB"
    maxActive="4"
    maxIdle="2"/>
</Context>
```

web.xml

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xml:lang="en">
  <context-param>
    <param-name>season-list</param-name>
    <param-value>Spring, Summer, Autumn, Winter</param-value>
  </context-param>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <resource-ref>
    <res-ref-name>jdbc/leagueDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
</web-app>
```

Notifications

1:1

INS

# context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiJARLocking="true" path="/Soccer">
  <Resource name="jdbc/leagueDB"
    driverClassName="org.apache.derby.jdbc.ClientDriver"
    url="jdbc:derby://localhost:1527/leagueDB"
    type="javax.sql.DataSource"
    username="使用者帳號"
    password="密碼"
    auth="container"
    maxActive="最大連線數"
    minIdle = "最小未使用連線數"
    maxIdle="最大未使用連線數"
    maxWait = "連線最長等待時間(ms)" />
</Context>
```

資料庫連結  
參數設定

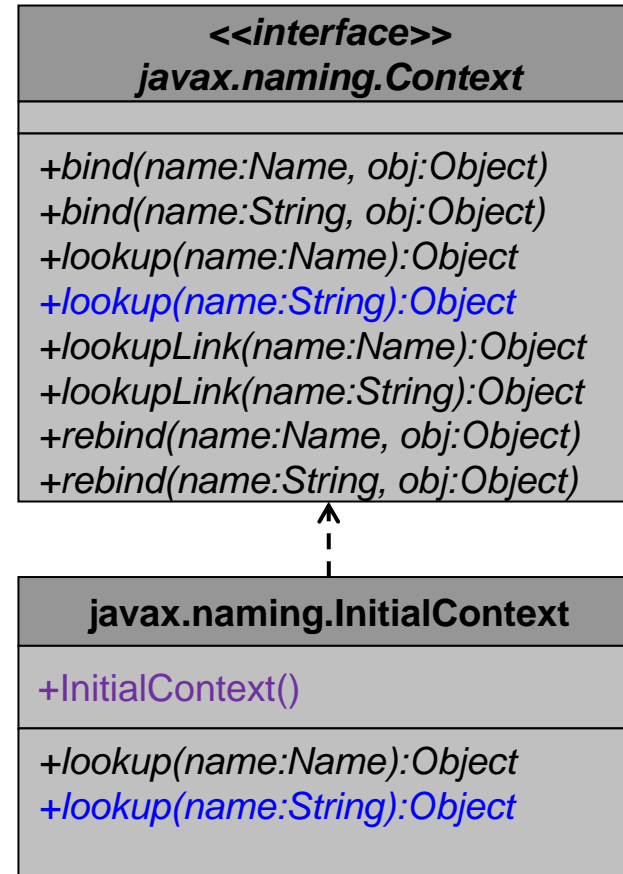
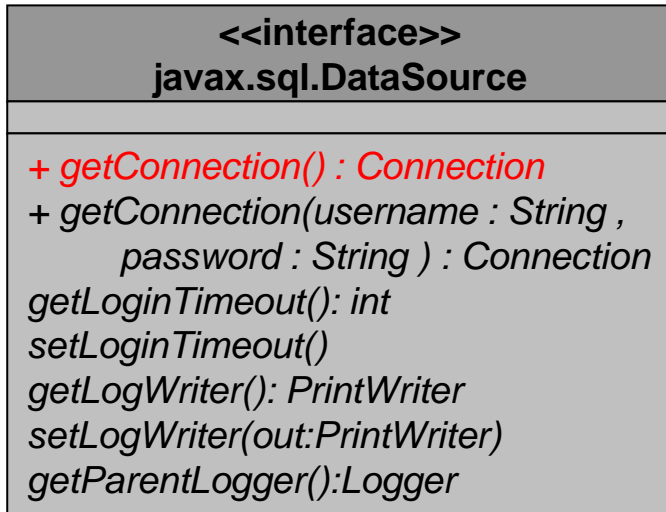
連線池  
參數設定

# web.xml

```
<web-app>
  <display-name> Web application 名稱. </display-name>
  <description> Web application 說明. </description>
  <servlet> ... </servlet>
  <servlet-mapping> ... </servlet-mapping>
  ...
  <resource-ref>
    <description> 資源說明 </description>
    <res-ref-name> jdbc/leagueDB </res-ref-name>
    <res-type> javax.sql.DataSource </res-type>
    <res-auth> Container </res-auth>
  </resource-ref>
</web-app>
```

需對應  
*context.xml*  
中資源定義

# DataSource & Context



# DataSource & Context

## ■ javax.naming.InitialContext

建構子	說明
InitialContext() throws NamingException	建構一個命名服務的起始點

## ■ javax.naming.Context

方法名稱	回傳型態	用途說明
lookup(String name()) throws NamingException	Object	取得命名服務中該名稱對應的資源

## ■ javax.sql.DataSource

方法名稱	回傳型態	用途說明
getConnection()	Connection	取得DataSource資源物件所代表的資料庫連結

# 程式中使用DataSource

- 取得JNDI物件

```
Context ctx = new InitialContext();
```

- 由JNDI物件搜尋DataSource資源

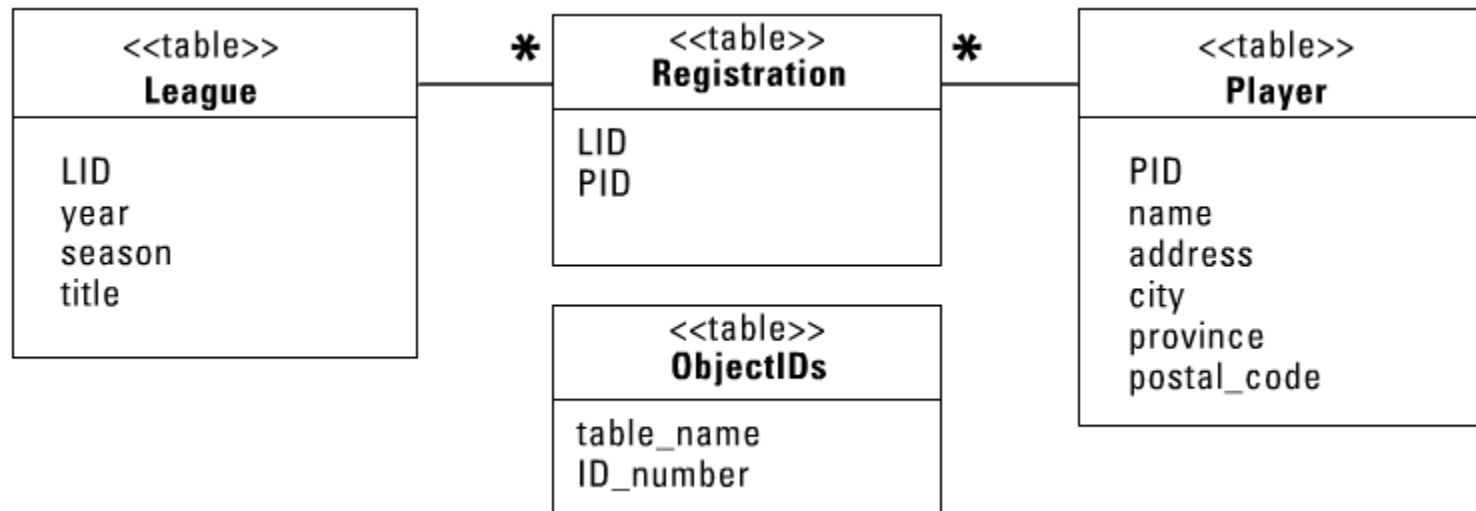
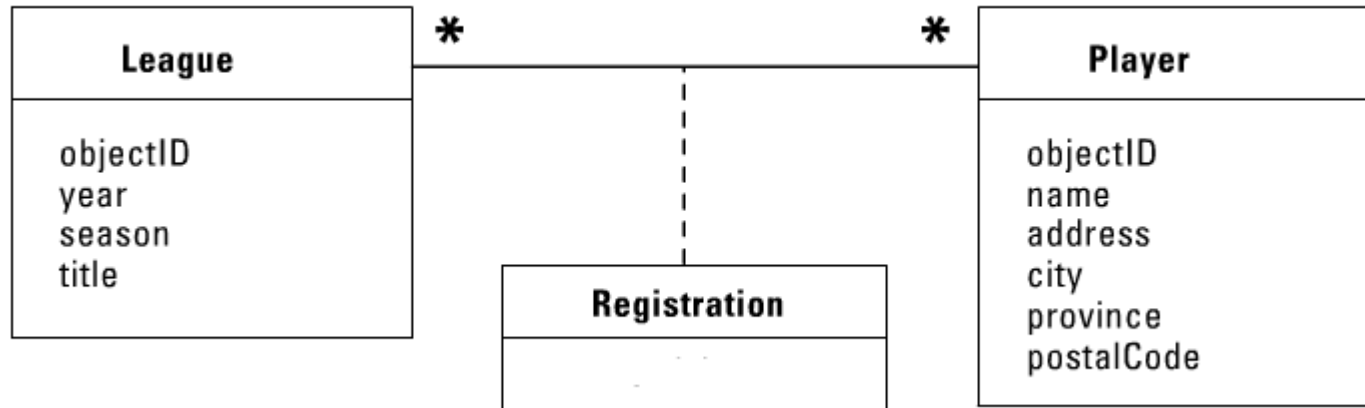
```
DataSource ds = (DataSource)ctx.lookup(  
    "java:comp/env/jdbc/leagueDB");
```

- 由DataSource取得Connection

```
Connection connection = ds.getConnection();
```

- Connection執行sql

# Soccer專案實作DBMS整合





<<table>> League
LID year season title

LID	year	season	title
001	2001	Spring	Soccer League (Spring '01)
002	2001	Summer	Summer Soccer Fest 2001
003	2001	Fall	Fall Soccer League 2001
004	2004	Summer	The Summer of Soccer Love

<<table>> Player
PID name address city province postal_code

PID	name	address	city	province	postal_code
047	Steve Sterling	12 Grove Park Road	Manchester	Manchester	M4 6NF
048	Alice Hornblower	62 Woodside Lane	Reading	Berks	RG31 9TT
049	Wally Winkle	17 Chippenham Road	London	London	SW19 4FT

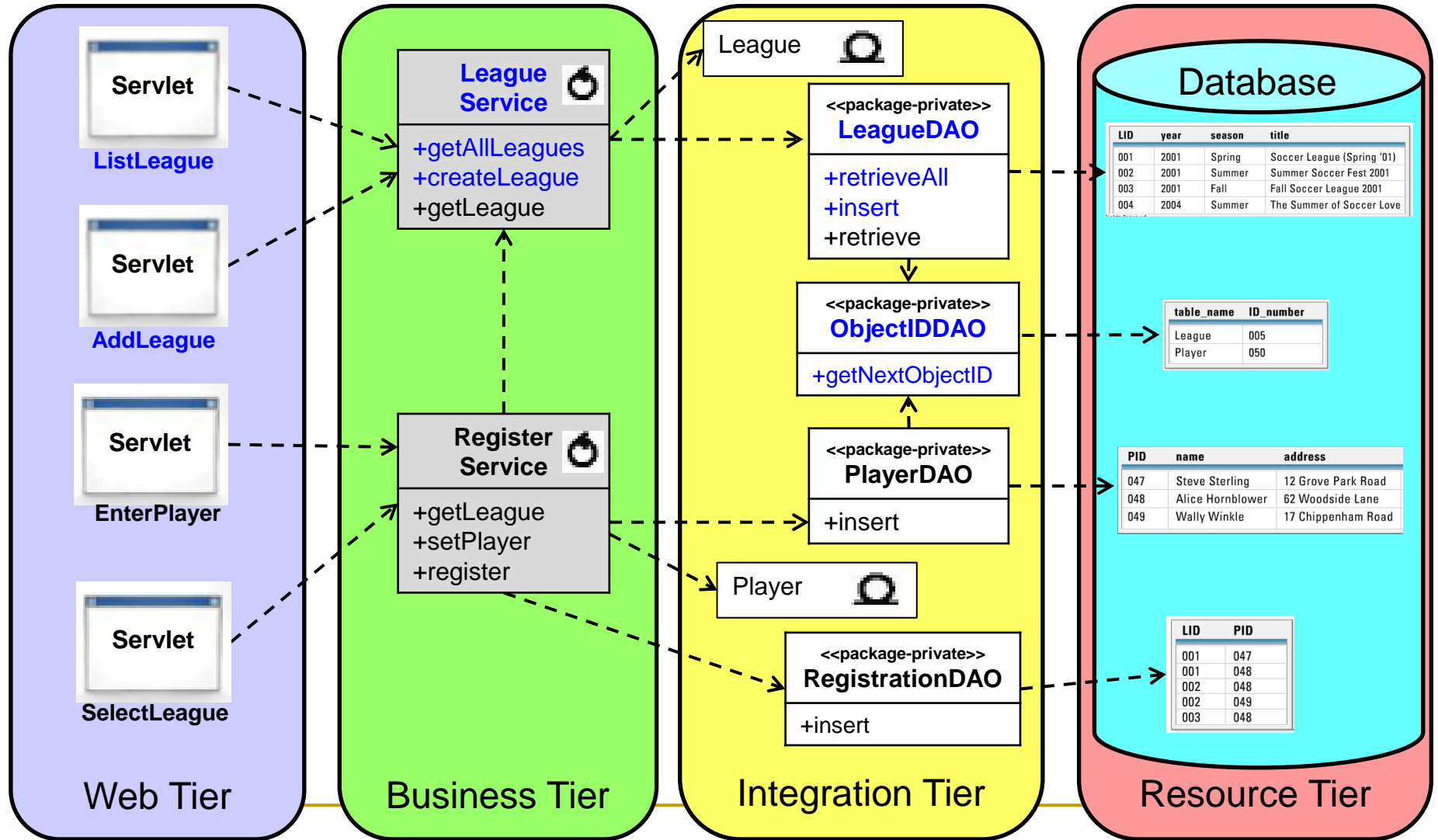
<<table>> ObjectIDs
table_name ID_number

table_name	ID_number
League	005
Player	050

<<table>> Registration
LID PID

LID	PID
001	047
001	048
002	048
002	049
003	048

# Soccer專案實作DBMS整合



# LeagueDAO元件

LeagueDAO
<b>#LeagueDAO()</b>
#retrieveAll() : List #insert(league : League) #retrieve(year: int, season: String) : League

```
package model;
// 引入SQL 類別
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.sql.DataSource;
// 引入JNDI 類別
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
// 引入Utility 類別
import java.util.List;
import java.util.LinkedList;

class LeagueDAO {
    /**
     * 這個建構子建立 League DAO 物件
     * 記得將這個建構子設為package-private
     * 如些一來其它套件的類別就無法存取這裏的方法
     */
    LeagueDAO() {
        // do nothing
    }
}
```

# LeagueDAO元件

LeagueDAO
#LeagueDAO()
#retrieveAll() : List <League> # insert(league : League) #retrieve(year: int, season: String) : Lea

```
private static final String RETRIEVE_ALL_STMT  
= "SELECT * FROM League" ;
```

```
List<League> retrieveAll() {           // JDBC變數
```

```
    DataSource ds = null;
```

```
    Connection connection = null;
```

```
    PreparedStatement stmt = null;
```

```
    ResultSet results = null;
```

```
    // 領域物件變數
```

```
    List<League> leagueList = new LinkedList<League>();
```

```
    League league = null;
```

```
    try {
```

```
        // 從JNDI 中取得DataSource
```

```
        Context ctx = new InitialContext();
```

```
        if ( ctx == null ) {
```

```
            throw new RuntimeException("JNDI Context could not be found.");
```

```
        }
```

```
        ds = (DataSource)ctx.lookup("java:comp/env/jdbc/leagueDB");
```

```
        if ( ds == null ) {
```

```
            throw new RuntimeException("DataSource could not be found.");
```

```
        }
```

```
        // 取得資料庫連線
```

```
        connection = ds.getConnection();
```

# LeagueDAO

## LeagueDAO

#LeagueDAO()

#retrieveAll() : List <League>

#insert(league : League)

#retrieve(year: int, season: String) : League

```
private static final String RETRIEVE_ALL_STMT
```

```
    = "SELECT * FROM League" ;
```

```
List<League> retrieveAll() {
```

```
    // JDBC變數
```

```
    .....
```

```
    // 領域物件變數
```

```
List<League> leagueList = new LinkedList<League>();
```

```
League league = null;
```

```
try {
```

```
    // 從JNDI 中取得DataSource
```

```
    .....
```

```
    // 取得資料庫連線
```

```
connection = ds.getConnection();
```

```
// 建立一個SELECT 敘述
```

```
stmt = connection.prepareStatement(RETRIEVE_ALL_STMT);
```

```
// 執行查詢動作
```

```
results = stmt.executeQuery();
```

```
// 迭代查詢的結果
```

```
while ( results.next() ) {
```

```
    int objectID = results.getInt("LID");
```

```
    // 建立並填寫League 物件中的資料
```

```
league = new League(objectID, results.getInt("yearno"),
```

```
    results.getString("season"), results.getString("title"));
```

```
    // 將League物件存入聯盟的集合中
```

```
leagueList.add(league);
```

```
}
```

```
// 傳回包含所有聯盟的集合
```

```
return leagueList;
```

```
// 例外處理及釋放 JDBC 資源
```

```
.....
```

# LeagueDAO 元件

LeagueDAO
#LeagueDAO()
#retrieve(year: int, season: String) : League #retrieveAll() : List <League> #insert(league : League)

```
} catch (SQLException se) { // 處理 SQL 錯誤
    throw new RuntimeException("A database error: " + se.getMessage());
} catch (NamingException ne) { // 處理 JNDI 錯誤
    throw new RuntimeException("A JNDI error: " + ne.getMessage());
} finally { // 釋放 JDBC 資源
    if ( results != null ) {
        try { results.close(); } catch (SQLException se) { se.printStackTrace(); }
    }
    if ( stmt != null ) {
        try { stmt.close(); } catch (SQLException se) { se.printStackTrace(); }
    }
    if ( connection != null ) {
        try { connection.close(); } catch (Exception e) { e.printStackTrace(); }
    }
} // END of try-catch-finally block
} // END of the retrieve method
```

# LeagueDAO

## LeagueDAO

#LeagueDAO()

#retrieveAll() : List

# insert(league : League)

#retrieve(year: int, season: String) : League

```
private static final String INSERT_STMT = "INSERT INTO League " +  
" (LID, yearno, season, title) VALUES (?, ?, ?, ?)";
```

```
void insert(League league) {
```

```
    // JDBC變數
```

```
    .....
```

```
    try {
```

```
        // 從JNDI 中取得DataSource
```

```
        .....
```

```
        // 取得資料庫連線
```

```
        connection = ds.getConnection();
```

```
        // 建立一個 INSERT 敘述
```

```
        insert_stmt = connection.prepareStatement(INSERT_STMT);
```

```
        // 取得下一個 League 物件的 object ID
```

```
        int leagueID = ObjectIDDAO.getNextObjectID  
                                (ObjectIDDAO.LEAGUE, connection);
```

```
        // 將取得之 object ID 加入INSERT敘述的第一個欄位
```

```
        insert_stmt.setInt(1, leagueID);
```

```
        // 將聯盟屬性資料加入INSERT敘述的對應欄位
```

```
        insert_stmt.setInt(2, league.year);
```

```
        insert_stmt.setString(3, league.season);
```

```
        insert_stmt.setString(4, league.title);
```

```
        // 執行INSERT敘述
```

```
        insert_stmt.executeUpdate();
```

```
        // 設定聯盟物件的 object ID
```

```
        league.objectID = leagueID;
```

```
        // 例外處理及釋放 JDBC 資源
```

```
        .....
```

# ObjectIdDAO

## ObjectIdDAO

+ LEAGUE : String

+ PLAYER : String

-ObjectIdDAO()

+ getNextObjectId(objectClassName :  
String , connection : Connection ) : int

```
final class ObjectIdDAO {
```

```
    public static final String LEAGUE = "League";
```

```
    public static final String PLAYER = "Player";
```

```
    private ObjectIdDAO() { }
```

```
    private static final String NEXT_ID_QUERY =
```

```
        "SELECT ID_number FROM ObjectIDs WHERE table_name=? " ;
```

```
    private static final String UPDATE_ID_CMD =
```

```
        "UPDATE ObjectIDs SET ID_number=? WHERE table_name=?" ;
```

```
    public static int getNextObjectId(String objectClassName,  
                                     Connection connection) {
```

```
        // JDBC變數
```

```
        try {
```

```
            // 建立一個 SELECT 敘述,初始化該敘述並執行查詢動作
```

```
            query_stmt = connection.prepareStatement(NEXT_ID_QUERY);
```

```
            query_stmt.setString(1, objectClassName);
```

```
            result = query_stmt.executeQuery();
```

```
            if ( result.next() ) {
```

```
                id = result.getInt("ID_number");
```

```
                incr_stmt = connection.prepareStatement(UPDATE_ID_CMD);
```

```
                incr_stmt.setInt(1, id + 1);
```

```
                incr_stmt.setString(2, objectClassName);
```

```
                incr_stmt.executeUpdate();
```

```
            } else {
```

```
                throw new RuntimeException("No ObjectId entry for class type: "  
                                         + objectClassName);
```

```
            }
```

```
        //例外處理及釋放 JDBC 資源
```

```
        return id;
```



# LeagueService元件

## LeagueService

+LeagueService()

+getAllLeague():List<League>

+createLeague(year : int, season : String, title: String)

+getLeague(year : int, season : String):League

```
package model;
// 引入Utility 類別
import java.util.List;
public class LeagueService {
    //宣告服務元件所使用之LeagueDAO物件
    private LeagueDAO leagueDataAccess;
    //建構子, 初始化LeagueDAO物件
    public LeagueService() {
        leagueDataAccess = new LeagueDAO();
    }
    //取得所有聯盟.使用LeagueDAO
    public List<League> getAllLeagues() {
        return leagueDataAccess.retrieveAll();
    }
    //新增聯盟至資料庫
    public League createLeague(int year, String season, String title) {
        // 建立聯盟物件,ObjectID未知,以-1帶入
        League league = new League(-1, year, season, title);
        //新增聯盟至資料庫
        leagueDataAccess.insert(league);
        return league;
    }
}
```

# ListLeague.java

```
package controller;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
import model.*;
@WebServlet(name = "ListLeagues", urlPatterns = {"ListLeagues"})
public class ListLeagues extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse
                                response) throws ServletException, IOException {
        ServletContext context = this.getServletContext();
        LeagueService leagueSvc = new LeagueService();
        List<League> leagueList = leagueSvc.getAllLeagues();
        context.setAttribute("leagueList", leagueList);
        context.log("The league list has been loaded."+leagueList);

        RequestDispatcher rd = request.getRequestDispatcher("list_leagues.jsp");
        rd.forward(request, response);
    }
}
```

# AddLeague.java

```
package controller;

.....
@WebServlet(name = "AddLeague", urlPatterns = {"/admin/AddLeague"})
public class AddLeague extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse
                                response) throws ServletException, IOException {

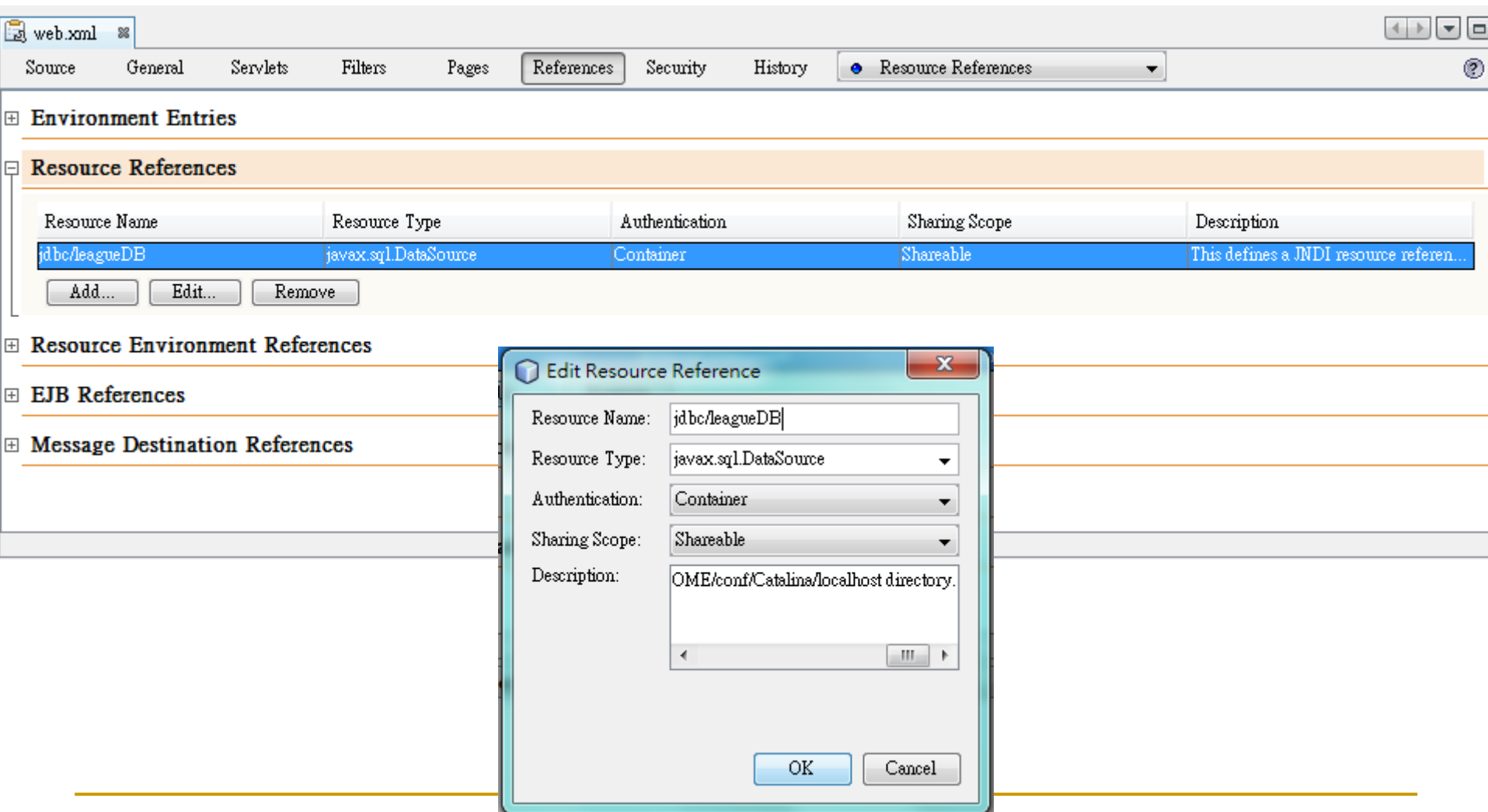
        .....
        // Keep a set of strings to record form processing errors.
        // Store this set in the request scope, in case we need to send the ErrorPage.
        try {
            // Retrieve form parameters and Verify form parameters
            // Send the ErrorPage view if there were errors
            //Perform business logic

            LeagueService leagueSvc = new LeagueService();
            League league = leagueSvc.createLeague(year, season, title);

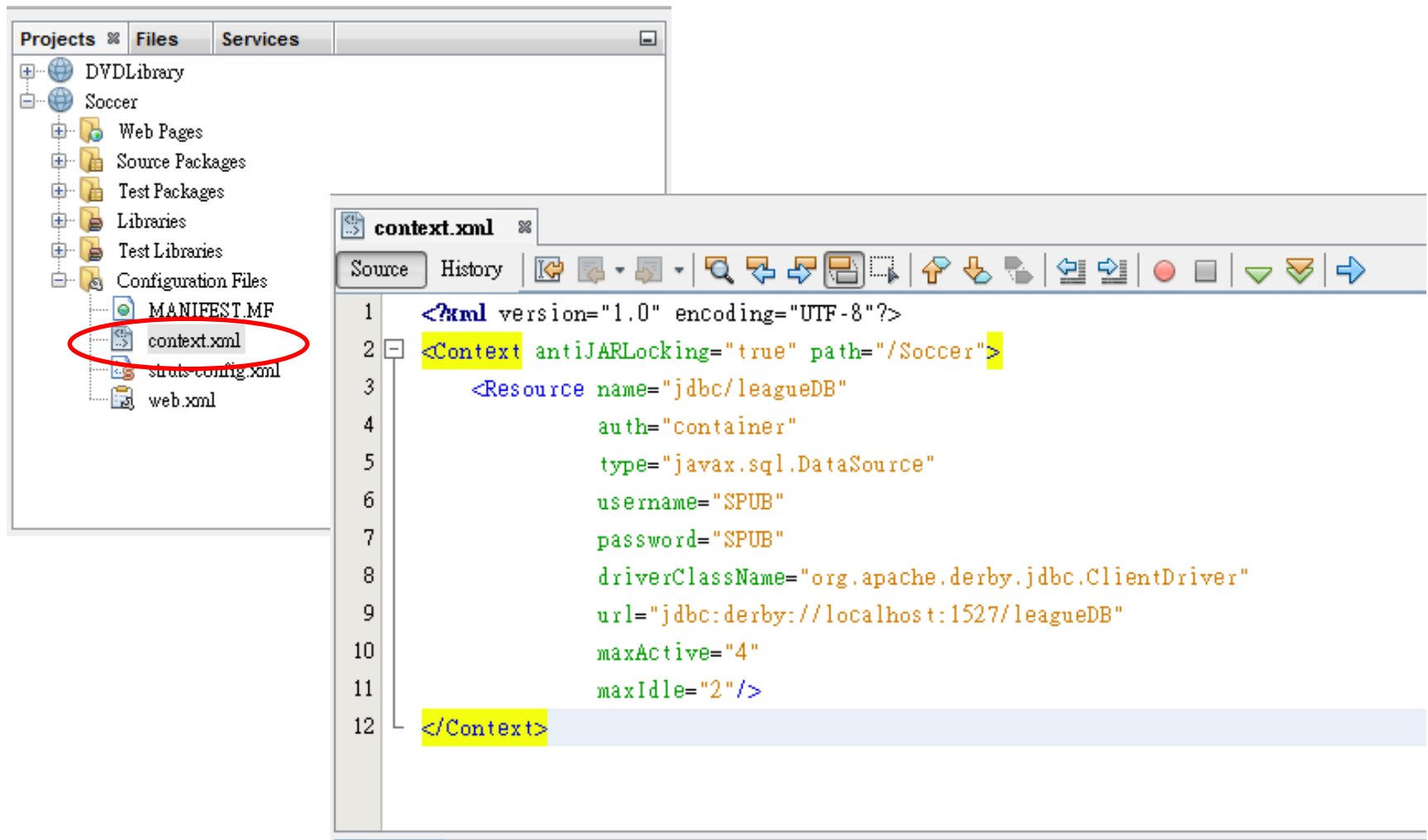
            // Store the new league in the leagueList context-scope attribute
            // Store the new league in the request-scope
            // Send the Success view

            .....
        } catch (RuntimeException e) {
            .....
        }
    }
}
```

# web.xml設定



# Context.xml



Duke's Soccer League: Home

localhost:8080/Socc...

## Duke's Soccer League: Home

This is the Home page for Duke's Soccer League.

### Players

- [List all leagues](#)
- [Register for a league](#)

### League Administrator

- [Add a new league](#)

SQL 2 [jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]]

Connection: jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]

```
SELECT * FROM SPUB.OBJECTIDS FETCH FIRST 100 ROWS ONLY;
```

SELECT \* FROM SPUB.OBJECT... x

Max. rows: 100 | Fetched Rows: 2 | Matching Rows:

#	TABLE_NAME	ID_NUMBER
1	League	7
2	Player	1

Duke's Soccer League: List Leag

localhost:8080/Socc...

## Duke's Soccer League: List Leagues

The set of soccer leagues:

- Soccer League (Spring '19)
- Summer Soccer Fest 2019
- Fall Soccer League (2019)
- Soccer League (Spring '20)
- The Summer of Soccer Love 2020
- Fall Soccer League (2020)

End of list.

SQL 1 [jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]]

Connection: jdbc:derby://localhost:1527/leagueDB [SPUB on SPUB]

```
SELECT * FROM SPUB.LEAGUE FETCH FIRST 100 ROWS ONLY;
```

SELECT \* FROM SPUB.LEAGUE... x

Max. rows: 100 | Fetched Rows: 6 | Matching Rows:

#	LID	YEARNO	SEASON	TITLE
1	1	2019	Spring	Soccer League (Spring '19)
2	2	2019	Summer	Summer Soccer Fest 2019
3	3	2019	Fall	Fall Soccer League (2019)
4	4	2020	Spring	Soccer League (Spring '20)
5	5	2020	Summer	The Summer of Soccer Love 2020
6	6	2020	Fall	Fall Soccer League (2020)

Duke's Soccer League: Add a New League

This form allows you to create a new soccer league.

Year:

Season:

Title:

Duke's Soccer League: Add League Success

Your request to add the *2020 Winter Game* league was successful.

Duke's Soccer League: List Leagues

The set of soccer leagues:

- Soccer League (Spring '19)
- Summer Soccer Fest 2019
- Fall Soccer League (2019)
- Soccer League (Spring '20)
- The Summer of Soccer Love 2020
- Fall Soccer League (2020)
- 2020 Winter Game

End of list.

SELECT \* FROM SPUB.OBJECT...

Max. rows: 100 | Fetched Rows: 2 |

#	TABLE_NAME	ID_NUMBER
1	League	8
2	Player	1

SELECT \* FROM SPUB.LEAGUE...

Max. rows: 100 | Fetched Rows: 7 |

#	LID	YEARNO	SEASON	TITLE
1	1	2019	Spring	Soccer League (Spring '19)
2	2	2019	Summer	Summer Soccer Fest 2019
3	3	2019	Fall	Fall Soccer League (2019)
4	4	2020	Spring	Soccer League (Spring '20)
5	5	2020	Summer	The Summer of Soccer Love 2020
6	6	2020	Fall	Fall Soccer League (2020)
7	7	2020	Winter	2020 Winter Game

# Lab 1-2 Soccer專案整合資料庫

1. 資料庫DataSource設定
  - JDBC Driver
  - Context.xml / web.xml
2. 載入並檢視 Soccer專案
  - DAO類別
    - LeagueDAO / ObjectIdDAO
  - Service類別
    - LeagueService
  - 領域物件 Domain Object
    - League
3. 測試執行Soccer專案



# Lab 2 球員註冊功能整合資料庫

## 1. 球員註冊功能整合資料庫設計

- DAO類別

- PlayerDAO / RegistrationDAO

- Service類別

- RegistrationService / LeagueService

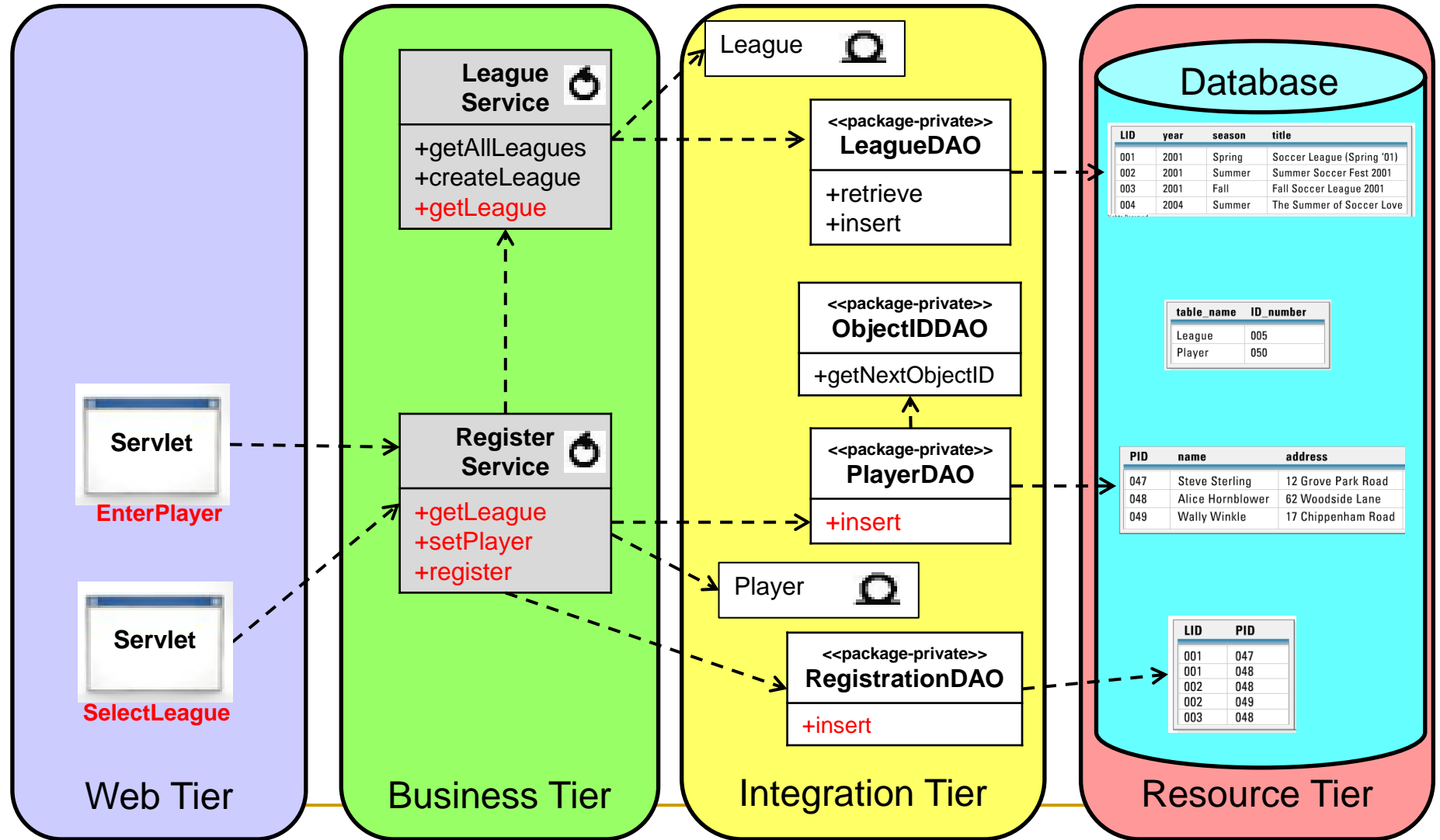
- Exception類別

- ObjectNotFoundException

- 領域物件 Domain Object

- Player

# 球員註冊功能整合資料庫



# Lab 2 球員註冊功能整合資料庫

## 2. 新增model.PlayerDAO類別

- 宣告新增球員的preparedStatement字串

```
String INSERT_STMT = "INSERT INTO Player (PID, name, address, city, province, postal_code) VALUES (?, ?, ?, ?, ?, ?)";
```

- void insert(Player player) 方法

- 宣告變數：DataSource, Connection, PreparedStatement, ResultSet
- 從JNDI 中取得DataSource
- 從DataSource取得資料庫連線
- 建立一個 INSERT預編敘述
- 設定預編敘述
  - 取得下一個 Player 物件的 object ID, 加入INSERT敘述的第一個欄位
  - 將球員屬性資料加入INSERT敘述的對應欄位
- 執行INSERT敘述
- 例外處理及資源釋放

# Lab 2 球員註冊功能整合資料庫

## 3. 新增model.RegistrationDAO類別

- 宣告新增註冊資料的preparedStatement字串

String **INSERT\_STMT** = "INSERT INTO Registration (LID, PID)  
VALUES (?, ?)";

- void insert(League league, Player player)方法
  - 宣告變數：DataSource, Connection, PreparedStatement, ResultSet
  - 從JNDI 中取得DataSource
  - 從DataSource取得資料庫連線
  - 建立一個 INSERT預編敘述
  - 設定預編敘述
    - 將聯盟物件 objectID 屬性加入INSERT敘述的第一個欄位
    - 將球員物件 playerId 屬性加入INSERT敘述的第二個欄位
  - 執行INSERT敘述
  - 例外處理及資源釋放

# Lab 2 球員註冊功能整合資料庫

## 4. 修改model.LeagueDAO類別

- 新增 League retrieve(int year, String season)方法
  - 依指定年及季,查詢聯盟資料表, 將查詢結果封裝為聯盟物件傳回
  - 聯盟不存在傳回ObjectNotFoundException

## 5. 新增 ObjectNotFoundException類別

## 6. 修改 model.LeagueService類別

- 新增 public League getLeague(int year, String season)方法
  - LeagueDAO的retrieve(year, season)方法取得聯盟物件並傳回
  - 聯盟不存在傳回ObjectNotFoundException

# Lab 2 球員註冊功能整合資料庫

## 7. 修改 `model.RegistrationService`類別

- ❑ 無參數建構子
- ❑ `public League getLeague(int year, String season)`方法
  - `LeagueService`的`getLeague(year, season)`取得聯盟物件並傳回
- ❑ `public void setPlayer(Player player)`
  - 建立`PlayerDAO`物件,將球員資料經`PlayerDAO`物件寫入資料庫
- ❑ `public void register( League league, Player player)`
  - 建立`RegistrationDAO`物件,呼叫`insert()`方法將資料寫入資料庫
- ❑ 移除原本的屬性、方法、建構子

## 8. 修改 `model.Player`類別

- ❑ 增加屬性 `int playerId`
- ❑ 修改建構子, 加入`playerID`傳入參數

```
public Player(int playerId, String name, String address, String city,
String province, String postalCode)
```

# Lab 2 球員註冊功能整合資料庫

## 9. 修改 controller.EnterPlayer類別

- ❑ Player建構子已變更, 多加一個playerID傳入參數
  - 先以-1帶入
  - PlayerDAO會以ObjectIdDAO取的值代換

## 10. 修改 controller.SelectLeague類別

- ❑ 刪除 private League findLeague(int year, String season) 方法
- ❑ 聯盟物件由RegistrationService getLeague()方法取得
- ❑ Session中取得Player物件
- ❑ RegistrationService設定setPlayer(Player player)
- ❑ RegistrationService註冊register(league, player)