Architecture of Enterprise Applications XIII Architectural Patterns – DB Design

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
e-mail: chen-hp@sjtu.edu.cn

Agenda

- Architectural Patterns
- Database Designing

What is Enterprise Application

- Enterprise applications often have complex data -- and lots of it -- to work on, together with business rules that fail all tests of logical reasoning.
  - Other terms for enterprise applications include “information systems"
  - Enterprise applications include payroll, patient records, shipping tracking, cost analysis, credit scoring, insurance, supply chain, accounting, customer service, and foreign exchange trading.
  - Enterprise applications don't include automobile fuel injection, word processors, elevator controllers, chemical plant controllers, telephone switches, operating systems, compilers, and games.

What is Enterprise Application

- Enterprise applications
  - usually involve persistent data
  - usually have a lot of data
  - usually many people access data concurrently
  - usually have a lot of user interface screens
  - usually they need to integrate with other enterprise applications scattered around the enterprise
  - conceptual dissonance with the data
  - complex business "illogic"

Enterprise Applications

- An enterprise system is one that has the following qualities:
  - Shares some or all of the resources used by the application
  - Is intended for internal use
  - Must work within existing architecture
  - Will be deployed and supported by internal IT staff
  - Requires greater robustness, both in terms of exception-handling and scalability
  - Must fail gracefully
  - Must gracefully handle evolution over time

Layering

- Layering is one of the most common techniques that software designers use to break apart a complicated software system.

- When thinking of a system in terms of layers, you imagine the principal subsystems in the software arranged in some form of layer cake, where each layer rests on a lower layer.
  - In this scheme the higher layer uses various services defined by the lower layer, but the lower layer is unaware of the higher layer.
  - Furthermore, each layer usually hides its lower layers from the layers above,

Layer & Tier

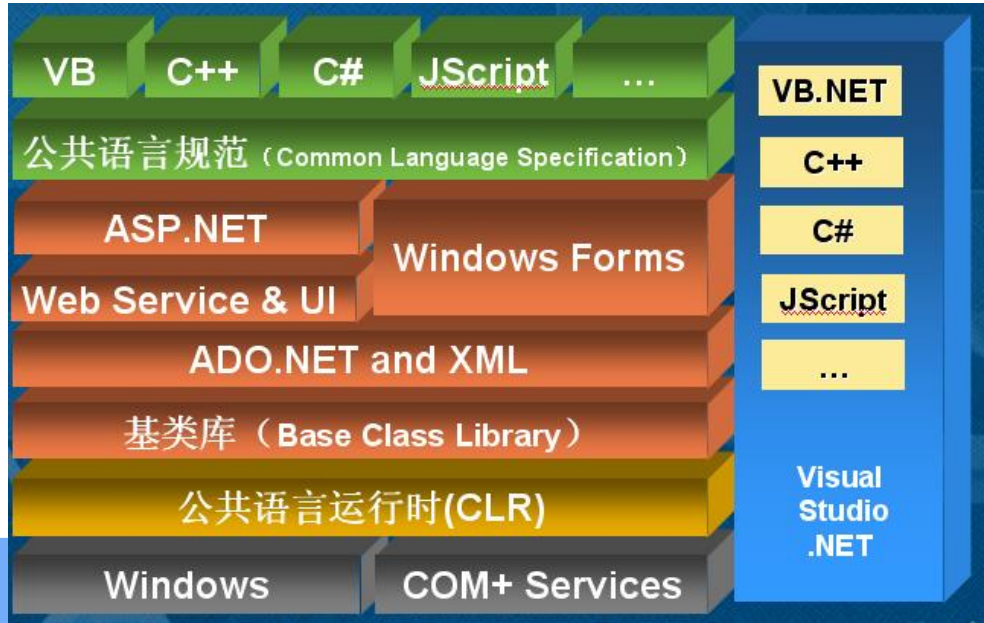
- Often the two are used as synonyms, but most people see tier as implying a physical separation.
  - Client/Server systems are often described as two-tier systems, and the separation is physical: The client is a desktop and the server is a server.

- Layers stress that you don't have to run the layers on different machines
  - A distinct layer of domain logic often runs on either a desktop or the database server. In this situation you have two nodes but three distinct layers.
  - With a local database I can run all three layers on a single laptop, but there will still be three distinct layers.

Key Layers

type dependency

Layer Name	Responsibilities	Implementation Technology	
Presentation	User Interface	JSP/HTML/JavaScript, Java.awt.Component subclasses	UI Layers
Application	Use-case UI workflow, syntactic validation, interaction with services	Servlets, <usebean> targets, Java.awt.Panel subclasses	
Services	Controlling txns, business/ workflow logic, acting as facade	EJB Session Beans	
Domain	The domain model, domain/ business logic, semantic validation	EJB Entity Beans, Plain Old Java Objects,	
Persistence	Persistent storage of domain object state	O/Rmappers,OODBMS,EJB Entity Bean BMP/CMP	



Database Designing and Accessing

- System Requirements
  - E-learning
  - To manage all kinds of teaching activities of one of the top universities, including enrollment, courses selection, score management, assignment management, and so on.
  - This university has over 30,000 undergraduate students and graduate students, and over 4,000 teachers.
  - This university opens over 4,000 courses every semester.
  - We should keep all data for a long period.

Database Design

- What will be happened to our system after running a long period?
  - Since we can't physically delete any data, the amount of data will be magnanimity.
  - More and more data will sharply decrease the performance of our system.
  - More and more users will be a challenge to support they access our system concurrently.

Database Design

- Should we split some tables into multiple tables with same structure?
  - For example, does STUDENT table should be cloned as STUDENT09, STUDENT10,…….?
  - Advantages:
    - The amount of single a table is limited into a certain range.
    - Decrease the possibility of accessing confliction.
  - Disadvantages:
    - We get a poor performance when we want to do some query about global data.
    - It can be a difficulty for applying ORM tools.

Database Design

- Choosing Your Key
  - meaningful key & meaningless key
    - Student ID
  - simple key & compound key
    - Selection of courses
  - table-unique key & database-unique key
    - Student ID – under the situation of splitting STUDENT tables
  - size of key
    - 8-character string vs. Long
- Getting a New Key
  - auto-generated procedures
    - Rely on you
  - database counter
    - Rely on DBMS
  - GUID
    - Unique vs. inefficient
  - your own method
    - table scan : complex, unreliable, and inefficient
    - key table

Database Design – key table

- If your database supports a database counter and you‘re not worried about being dependent on database-specific SQL, you should use the counter.

- For the moment let's assume that we have to do this the hard way. The first thing we need is a key table in the database.

  ```
  CREATE TABLE keys (name varchar primary key, nextID int)
  INSERT INTO keys VALUES ('orders', 1)
  ```

- This table contains one row for each counter that's in the database.
  - In this case we've initialized the key to 1.
  - If you're preloading data in the database, you'll need to set the counter to a suitable number.
  - If you want database-unique keys, you'll only need one row, if you want table-unique keys, you'll need one row per table.

Database Design – key table

- We construct a key generator with its own database connection, together with information on how many keys to take from the database at one time.

```
class KeyGenerator...
 private Connection conn;
 private String keyName;
 private long nextId;
 private long maxId;
 private int incrementBy;
 public KeyGenerator(Connection conn, String keyName, int incrementBy) {
   this.conn = conn;
   this.keyName = keyName;
   this.incrementBy = incrementBy;
   nextId = maxId = 0;
   try {
     conn.setAutoCommit(false);
   } catch(SQLException exc) {
      throw new ApplicationException("Unable to turn off autocommit", exc);
    }
 }
```

Database Design – key table

- When we ask for a new key, the generator looks to see if it has one cached rather than go to the database.

```
class KeyGenerator...
 public synchronized Long nextKey() {
  if (nextId == maxId) {
   reserveIds();
  }
  return new Long(nextId++);
 }
```

- If the generator hasn't got one cached, it needs to go to the database.

```
class KeyGenerator...
 private void reserveIds() {
  PreparedStatement stmt = null;
  ResultSet rs = null;
  long newNextId;
  try {
   stmt = conn.prepareStatement("SELECT nextID FROM keys WHERE name = ? FOR  UPDATE");
   stmt.setString(1, keyName);
   rs = stmt.executeQuery();
   rs.next();
   newNextId = rs.getLong(1);
  } catch (SQLException exc) {
   throw new ApplicationException("Unable to generate ids", exc);
  } finally
  { DB.cleanUp(stmt, rs); }
```

Database Design – key table

REliable, INtelligent & Scalable Systems

```
long newMaxId = newNextId + incrementBy;
stmt = null;
try {
 stmt = conn.prepareStatement("UPDATE keys SET nextID = ? WHERE name
    = ?");
 stmt.setLong(1, newMaxId);
 stmt.setString(2, keyName);
 stmt.executeUpdate();
 conn.commit();
 nextId = newNextId;
 maxId = newMaxId;
} catch (SQLException exc) {
  throw new ApplicationException("Unable to generate ids", exc);
 } finally { DB.cleanUp(stmt); }
}
```

Database Accessing

- JDBC/ODBC reading or ORM?
  - JDBC/ODBC reading
  - Advantages:
    - Good performance, especially for accessing massive data
    - Take advantage of various functions provided by DBMS
    - Use stored procedures to implement complex logics
  - Disadvantages:
    - Coupling with DBMS
    - Coupling with data structure
    - Programming is complicate

Database Accessing

- JDBC/ODBC reading or ORM?
  - ORM
  - Advantages:
    - Independent of DBMS
    - Independent of data structure
    - OOP
  - Disadvantages:
    - Impact on performance
    - Can NOT utilize the extra functions in addition to standard functions defined in specification.
    - Mapping between O and R maybe complex.
    - Can NOT invoke stored procedures.

Database Accessing

- JDBC/ODBC reading or ORM?
  - Requirements Driven
  - Requirement 1:
    - To list all records about courses selection of a certain semester.
    - It could be a big records set with over 150,000 records selected from millions of records
  - Requirement 2:
    - To get the average age of all undergraduate students
  - Requirement3:
    - To ensure that the database could be moved from ORACLE to DB2

Database Accessing

- JDBC/ODBC reading
  - How to manipulate data from client ends?
    - Online:
      - Keep holding database connection
      - Transfer data by references
      - Real-time status of data
      - Do harmful concurrency
    - Offline:
      - Release database connection after the data to be transferred
      - Transfer data by values
      - Encourage concurrency
      - Could occur confliction between transactions

Database Accessing

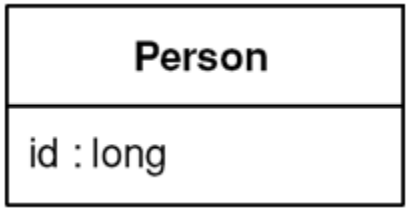
- JDBC/ODBC reading
  - Shall we use stored procedures?
    - Advantages:
      - Processes are close to data
      - Less number of invocations between clients and server
      - Invariant can protect data from unauthenticated accesses
    - Disadvantages:
      - Poor portability
      - Poor reusability
      - Challenge for security of business logics

Database Accessing

- ORM
  - Which kind of ORM tool should we use?
    - Requirements driven again
    - Requirement :
    - STUDENT table is divided as STUDENT09, STUDENT 10, …… and so on.
    - Hibernate, EJB, ……
  - Offline or Online ?
    - ORM adopt an offline way
    - Offline way brings a deal of troubles
    - Synchronization and Mutual Exclusion
    - Contention

Database Designing-Identity Field

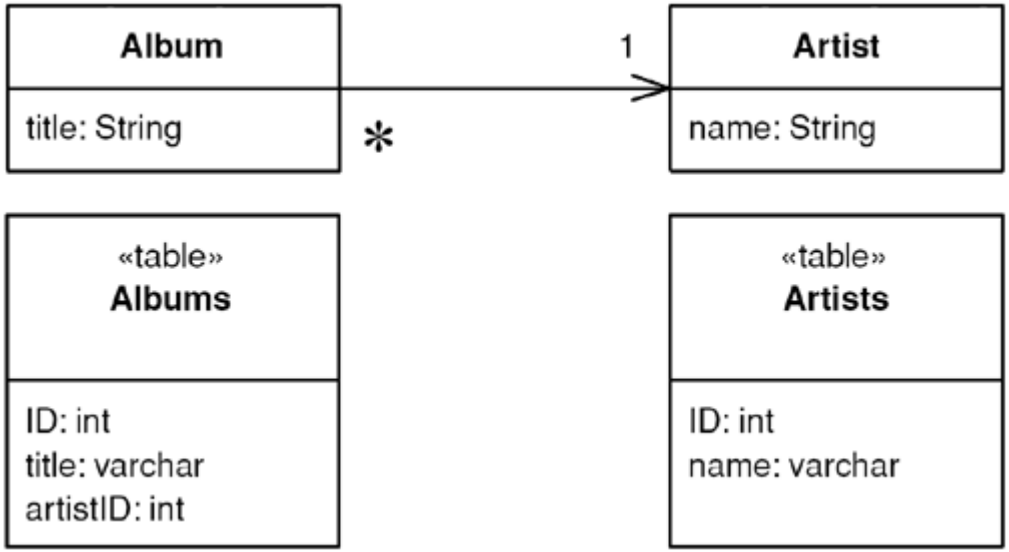
- ORM
  - How to deal with foreign key mapping?
    - Saves a database ID field in an object to maintain identity between an in-memory object and a database row.



    - Relational databases tell one row from another by using key - In particular, the primary key.
    - However, in-memory objects don't need such a key, as the object system ensures the correct identity under the covers (or in C++'s case with raw memory locations).
    - In essence, Identity Field is mind-numbingly simple. All you do is store the primary key of the relational database table in the object's fields.

Database Designing-Foreign Key Mapping

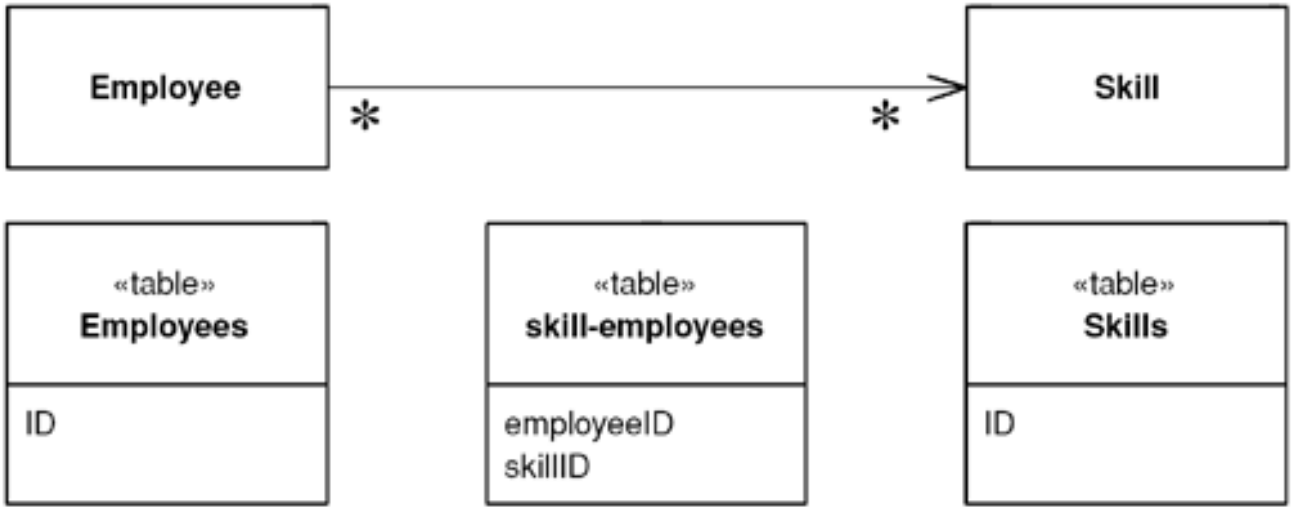
- ORM
  - How to deal with foreign key mapping?
    - Maps an association between objects to a foreign key reference between tables.



    - Objects can refer to each other directly by object references.
    - To save these objects to a database, it's vital to save these references.
    - A Foreign Key Mapping maps an object reference to a foreign key in the database.

Database Designing-Association Table Mapping

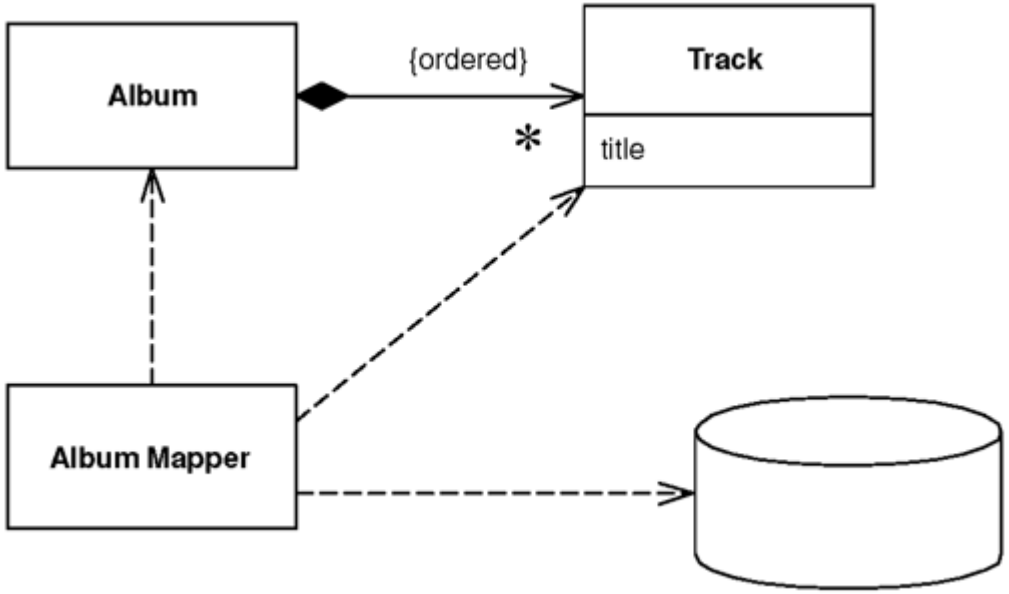
- ORM
  - How to deal with the associate Table?
    - Saves an association as a table with foreign keys to the tables that are linked by the association.



    - A many-to-many association can't do this because there is no single-valued end to hold the foreign key.
    - The answer is the classic resolution that's been used by relational data people for decades: create an extra table to record the relationship. Then use Association Table Mapping to map the multivalued field to this link table.

Database Designing-Dependent Mapping

- ORM
  - How to deal with the dependent data?
    - Has one class perform the database mapping for a child class.



    - Some objects naturally appear in the context of other objects. If they aren't referenced to by any other table in the database, you can simplify the mapping procedure by treating this mapping as a dependent mapping.

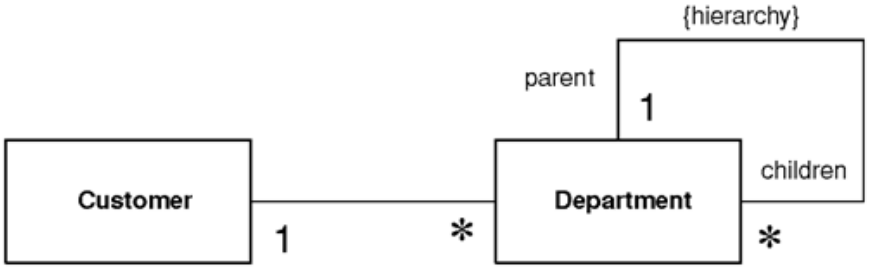
Database Designing-Embedded Value

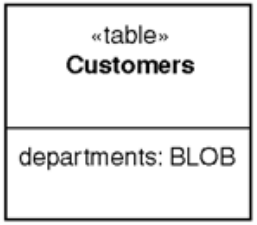
- ORM
  - How to deal with the data with massive but relevent fields?
    - Maps an object into several fields of another object's table.

    - Many small objects make sense in an OO system that don't make sense as tables in a database.
    - An Embedded Value maps the values of an object to fields in the record of the object's owner.

Database Designing-Serialized LOB

- ORM
  - How to deal with the data with a hierarchy field?
    - Saves a graph of objects by serializing them into a single large object (LOB), which it stores in a database field.

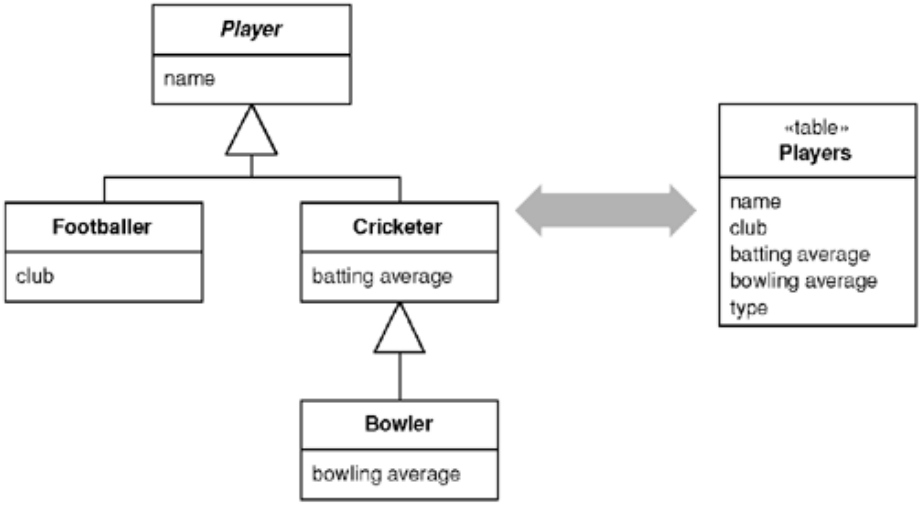




- To put all objects into a relational schema. The basic schema is simple
- Objects don't have to be persisted as table rows related to each other.

Database Designing-Single Table Inheritance

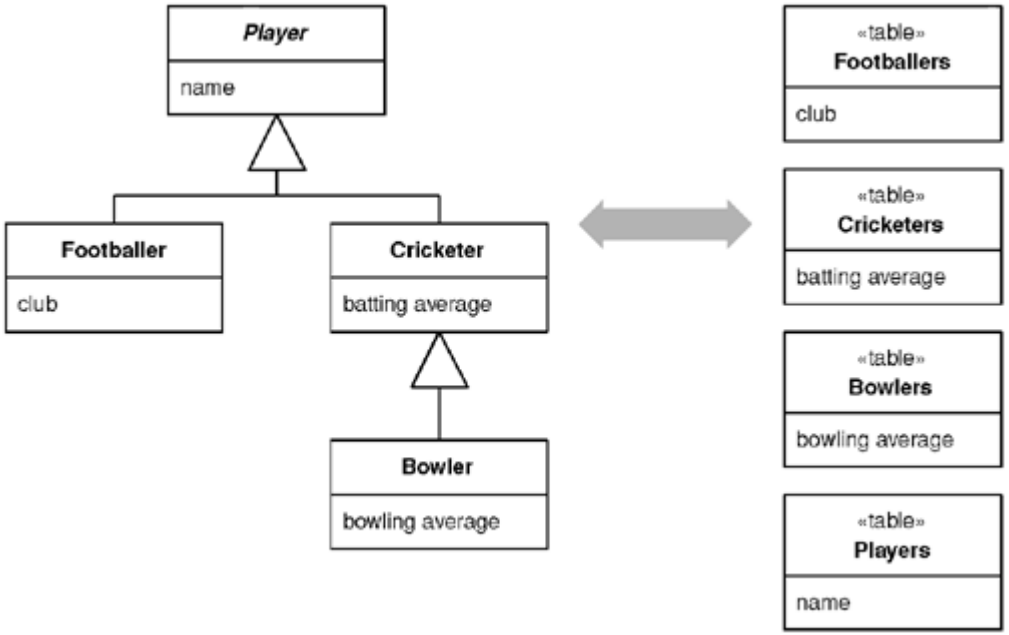
- ORM
  - How to deal with the data with a hierarchy field?
    - Saves a graph of objects by serializing them into a single large object (LOB), which it stores in a database field.



    - To put all objects into a relational schema. The basic schema is simple
    - Objects don't have to be persisted as table rows related to each other.

Database Designing-Class Table Inheritance

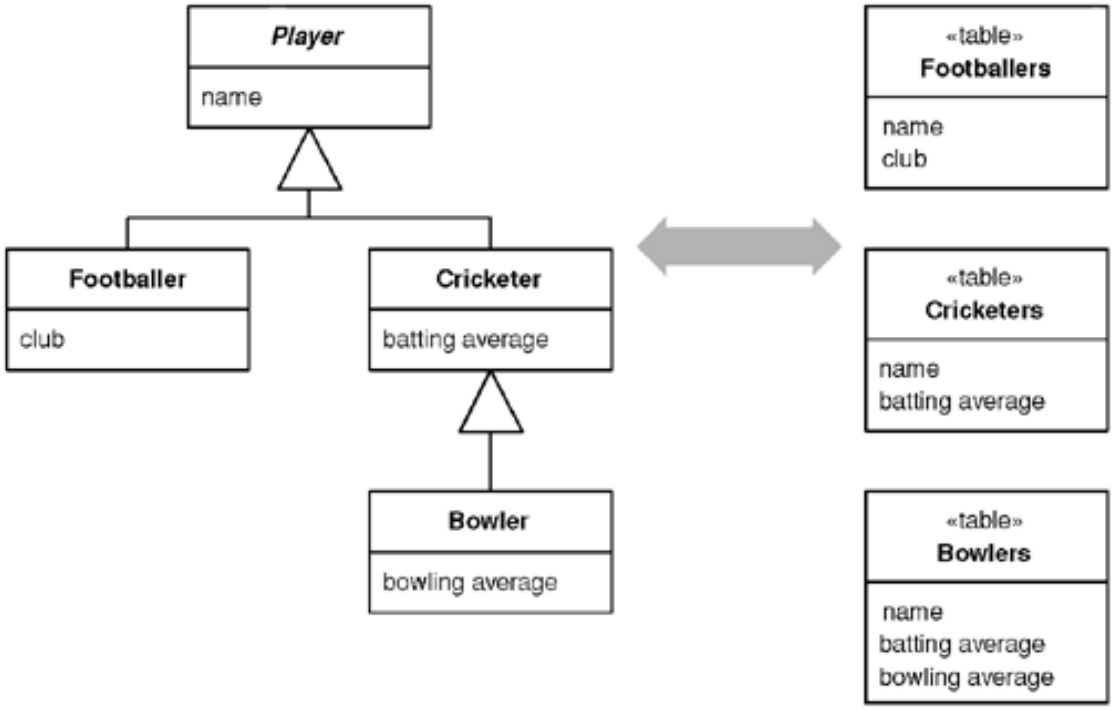
- Represents an inheritance hierarchy of classes with one table for each class.



- A very visible aspect of the object-relational mismatch is the fact that relational databases don't support inheritance.
  - You want database structures that map clearly to the objects and allow links anywhere in the inheritance structure.
  - Class Table Inheritance supports this by using one database table per class in the inheritance structure.

Database Designing-Concrete Table Inheritance

- Represents an inheritance hierarchy of classes with one table per concrete class in the hierarchy.



- Thinking of tables from an object instance point of view, a sensible route is to take each object in memory and map it to a single database row. This implies Concrete Table Inheritance, where there's a table for each concrete class in the inheritance hierarchy.

References

- Martin Fowler's Patterns of Enterprise Application Architecture

Thank You!