Проектирование архитектур программного обеспечения

лекция 5

Зозуля А.В.

КСПТ

2016 г.

ранее

- Архитектурные типовые решения источников данных
- Объектно-реляционные типовые решения, предназначенные для моделирования **поведения**

Типовые решения источников данных

- Архитектурные типовые решения источников данных
- Объектно-реляционные типовые решения, предназначенные для моделирования
  - Поведения
  - **Структуры**
- Типовые решения объектно-реляционного отображения с использованием метаданных

КСПТ

Объектно-реляционные типовые решения, предназначенные для моделирования структуры

- Поле идентификации (Identity Field)
- Отображение внешних ключей (Foreign Key Mapping)
- Отображение с помощью ассоциаций (Association Table Mapping)
- Отображение зависимых объектов (Dependent Mapping)
- Внедренное значение (Embedded Value)
- Сериализованный крупный объект (Serialized LOB)
- Наследование
- Преобразователи наследования (Inheritance Mappers)

Поле идентификации (Identity Field)

Сохраняет идентификатор записи БД для поддержки соответствия между объектом приложения и записью

Выбор ключа

- Значащий / незначащий ключ
- Простой / составной ключ
- Тип ключа
- Уникальность на уровне таблицы / дерева «иерархии» таблиц / всей БД
- Размер ключа: производительность

Представление поля идентификации в объекте

- Тип поля объекта соответствует типу ключа БД
- Составной ключ = класс ключа + equals()
- Универсальный класс ключа содержит последовательность объектов-элементов ключа
- Миграция ключей при импорте данных из другой БД

Вычисление нового значения ключа

- Автоматическая генерация средствами БД (автоинкрементное поле, триггер + генератор,..)
    - Вставка зависимых данных в одной транзакции?
- Глобальный уникальный идентификатор (GUID): MAC + ID чипсета + текущее время + …
    - Размер ключа?
- Самостоятельная генерация: SELECT MAX()
    - Параллельные запросы?
- Таблица ключей (Key Table)

Таблица ключей (Key Table)

- Одна запись на всю БД или на каждую таблицу
- Имя таблицы — последний ID
- SELECT / UPDATE в отдельной транзакции
- Выборка нескольких ключей одновременно

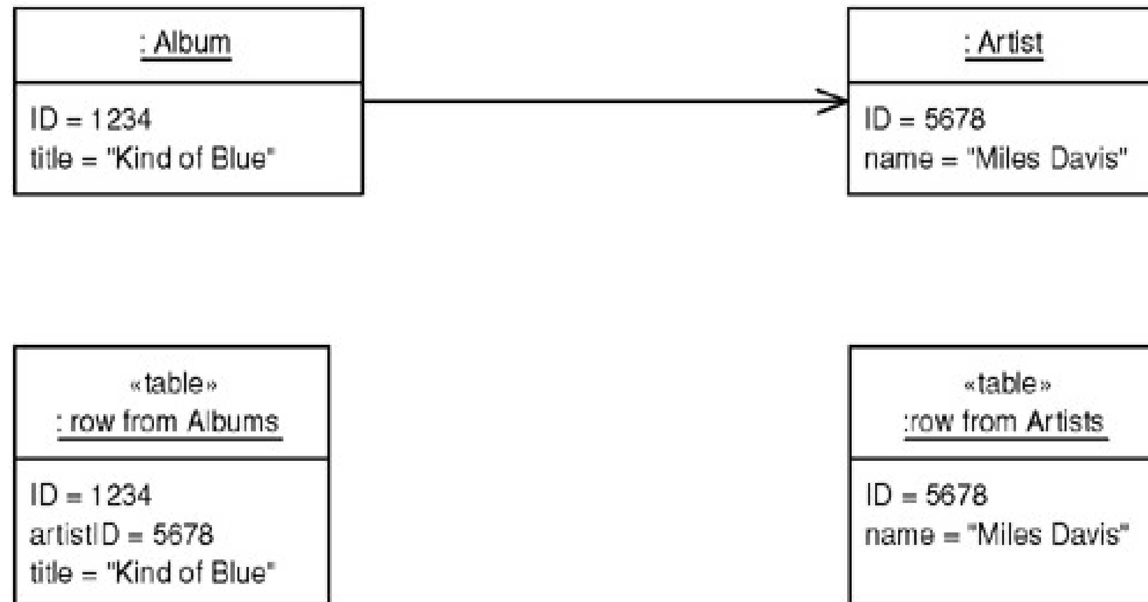
```
CREATE TABLE keys (name varchar primary key, nextID int);

INSERT INTO keys VALUES ('orders', 1);
```

Реализация таблицы ключей

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

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("..", exc);
	} finally { DB.cleanup(stmt, rs); }
	long newMaxId = newNextId + incrementBy;
	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("..", exc);
	}  finally { DB.cleanup(stmt); }
}
```

Отображение внешних ключей (Foreign Key Mapping)

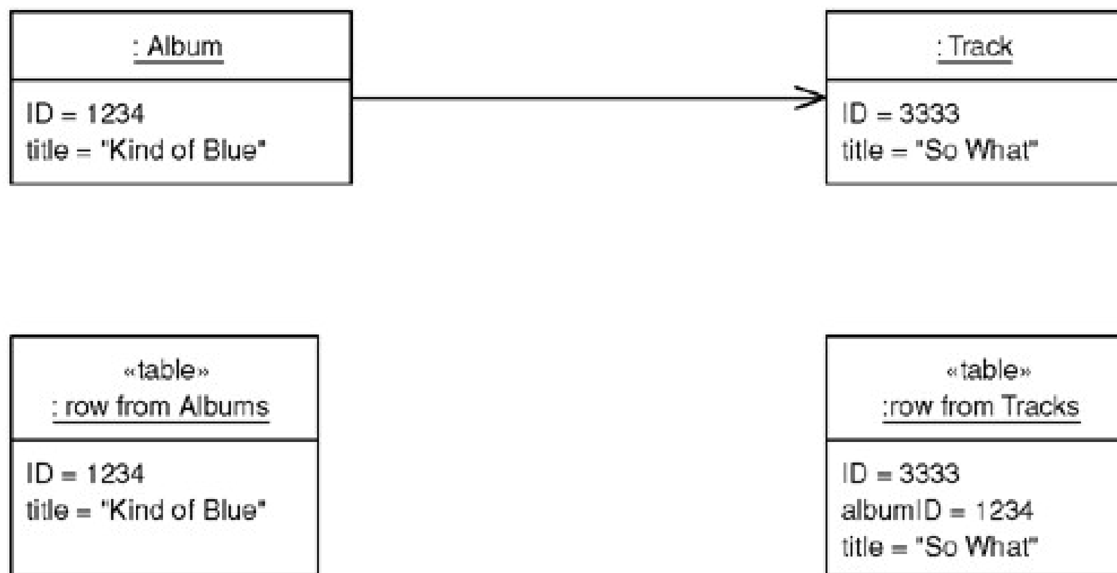
- Отображает ассоциации между объектами на ссылки внешнего ключа между таблицами БД



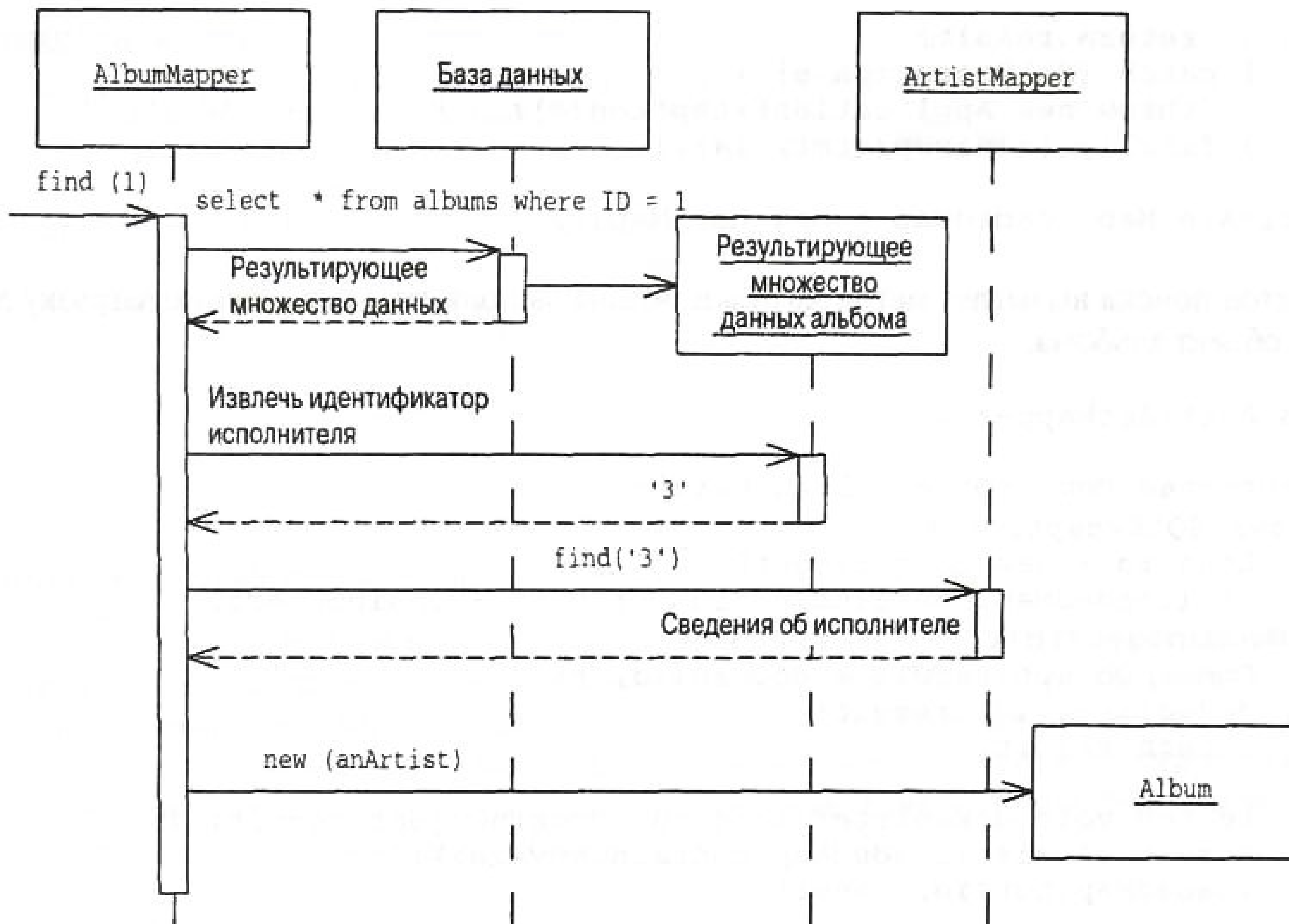
Отображение внешних ключей (Foreign Key Mapping)

Способы обновления связанных данных:

- Удаление коллекции и вставка (для полностью зависимых объектов)
- Обратный указатель (двунаправленная связь)
- Операция сравнения

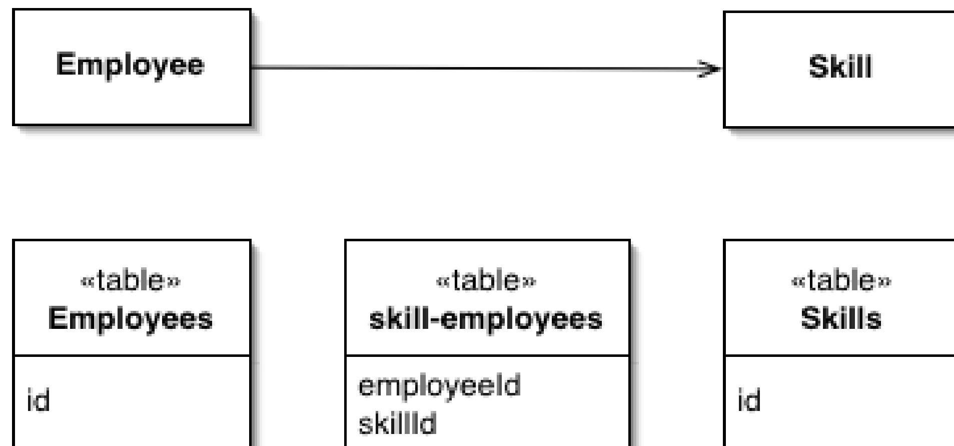


Процесс загрузки однозначного поля



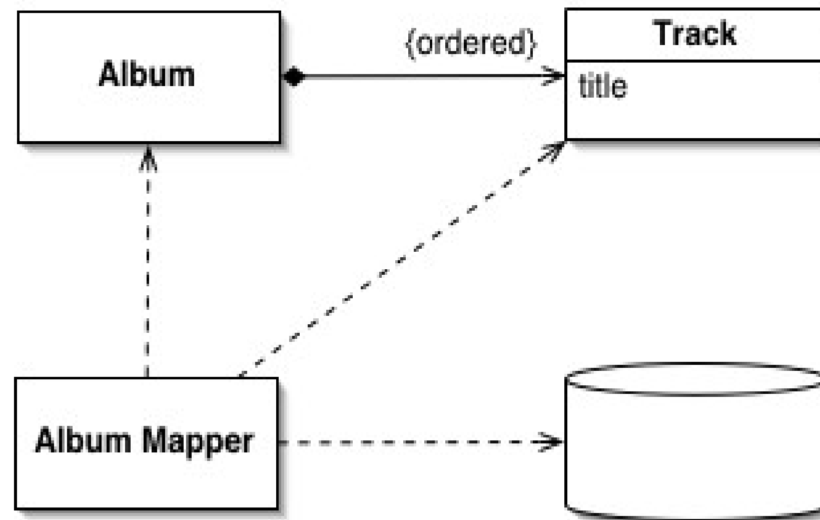
Отображение с помощью таблицы ассоциаций (Association Table Mapping)

- Сохраняет множество ассоциаций в виде таблицы отношений, содержащей внешние ключи таблиц, связанных ассоциациями
- Таблице ассоциаций не соответствует объект
- Реализация отношения «многие ко многим»
- Для выборки используются JOIN-запросы



Отображение зависимых объектов (Dependent Mapping)

- Передает некоторому классу полномочия по выполнению отображения для зависимого класса
- У зависимого класса д.б. только один владелец
- Зависимый объект не имеет поле идентификации
- Нет необходимости в обратных указателях



Отображение зависимых объектов (Dependent Mapping)

```
class AlbumMapper...
	public void update(DomainObject arg) {
		PreparedStatement updateStatement = null;
		try {
			updateStatement = DB.prepare("UPDATE albums SET title = ? WHERE id
= ?");
			UpdateStatement.setLong(2, arg.getID().longValue());
			Album album = (Album) arg;
			updateStatement.setString(1, album.getTitle());
			updateStatement.execute();
			updateTracks(album);
		} catch (SQLException e) {
			throw new ApplicationException(e);
		} finally {DB.cleanUp(updateStatement);}
}

public void updateTracks(Album arg) throws SQLException {
	PreparedStatement deleteTracksStatement = null;
	try {
		deleteTracksStatement = DB.prepare("DELETE from tracks WHERE albumID = ?");
		deleteTracksStatement.setLong(1, arg.getID().longValue());
		deleteTracksStatement.execute();
		for (int i = 0; i < arg.getTracks().length; i++) {
			Track track = arg.getTracks()[i];
			insertTrack(track, i + 1, arg);
		}
	} finally {DB.cleanUp(deleteTracksStatement);}
}
```

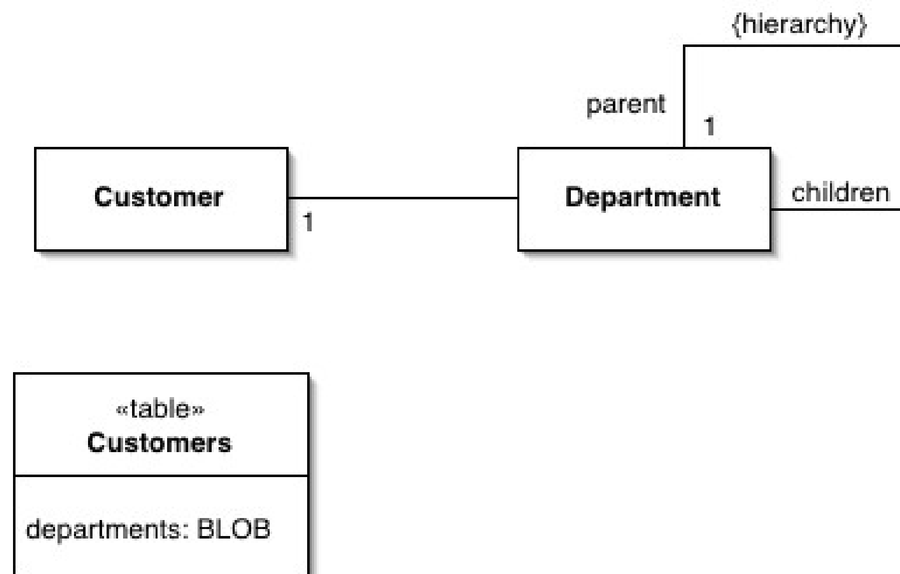
Внедренное значение (Embedded Value)

- Отображает объект на несколько полей таблицы, соответствующей другому объекту
- Небольшой объект, который незачем хранить в БД
- В виде внедренных значений хранятся Объекты-значения (Value Object)

```
class ProductOffering...
    public static ProductOffering load(ResultSet rs) {
    try {
        Integer id = (Integer) rs.getObject("ID");
        BigDecimal baseCostAmount = rs.getBigDecimal("base_cost_amount");
        Currency baseCostCurrency =
Registry.getCurrency(rs.getString("base_cost_currency"));
        Money baseCost = new Money(baseCostAmount, baseCostCurrency);
        Integer productID = (Integer) rs.getObject("product");
        Product product = Product.find((Integer) rs.getObject("product"));
        return new ProductOffering(id, product, baseCost);
    } catch (SQLException e) { throw new ApplicationException(e);}
}
```

Сериализованный крупный объект (Serialized Large Object)

- Сохраняет граф объектов путем их сериализации в единый крупный объект и помещает его в поле БД
- Реляционные БД «плохо хранят» иерархии объектов
- Иерархию можно сериализовать в Binary LOB и Character LOB (CLOB — XML, сжатый XML)
- «+» Сохранение состояния объектов на момент времени
- «-» Классозависимость

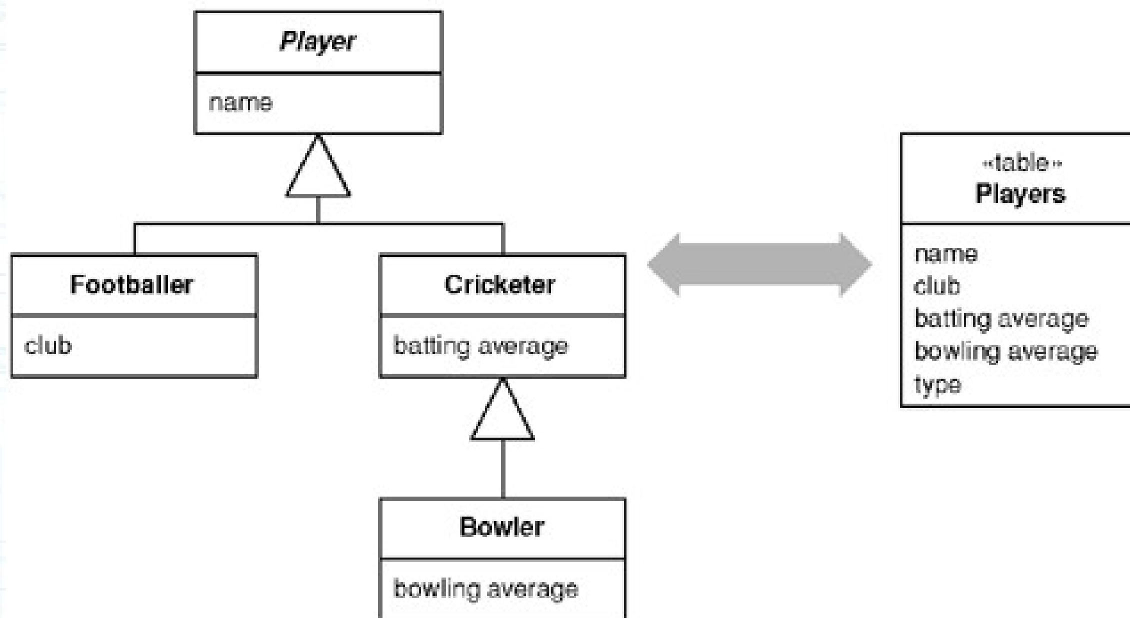


Сериализованный крупный объект

```
class Customer...
    public Long insert() {
        try {
            setID (findNextDatabaseId());
            insertStatement.setInt(1, getID().intValue());
            insertStatement.setString(2, name);
            insertStatement.setString(3, XmlStringer.write(departmentsToXmlElement()));
            insertStatement.execute();
            Registry.addCustomer(this);
            return getID();
        } catch(SQLException e) {throw new ApplicationException(e); }
    }
    public Element departmentsToXmlElement() {
        Element root = new Element("departmentList");
        Iterator i = departments.iterator();
        while (i.hasNext()) {
            Department dep = (Department) i.next();
            root.addContent(dep.toXmlElement());
        }
        return root;
    }

class Department...
    Element toXmlElement() {
        Element root = new Element("department");
        root.addAttribute("name", name);
        Iterator i = subsidiaries.iterator();
        while (i.hasNext()) {
            Department dep = (Department) i.next();
            root.addContent(dep.toXmlElement());
        }
        return root;
    }
```

Наследование с одной таблицей (Single Table Inheritance)

- Представляет иерархию наследования классов в виде одной таблицы, столбцы которой соответствуют всем полям классов, входящих в иерархию
- Как узнать, какого класса извлекаемый из БД объект?

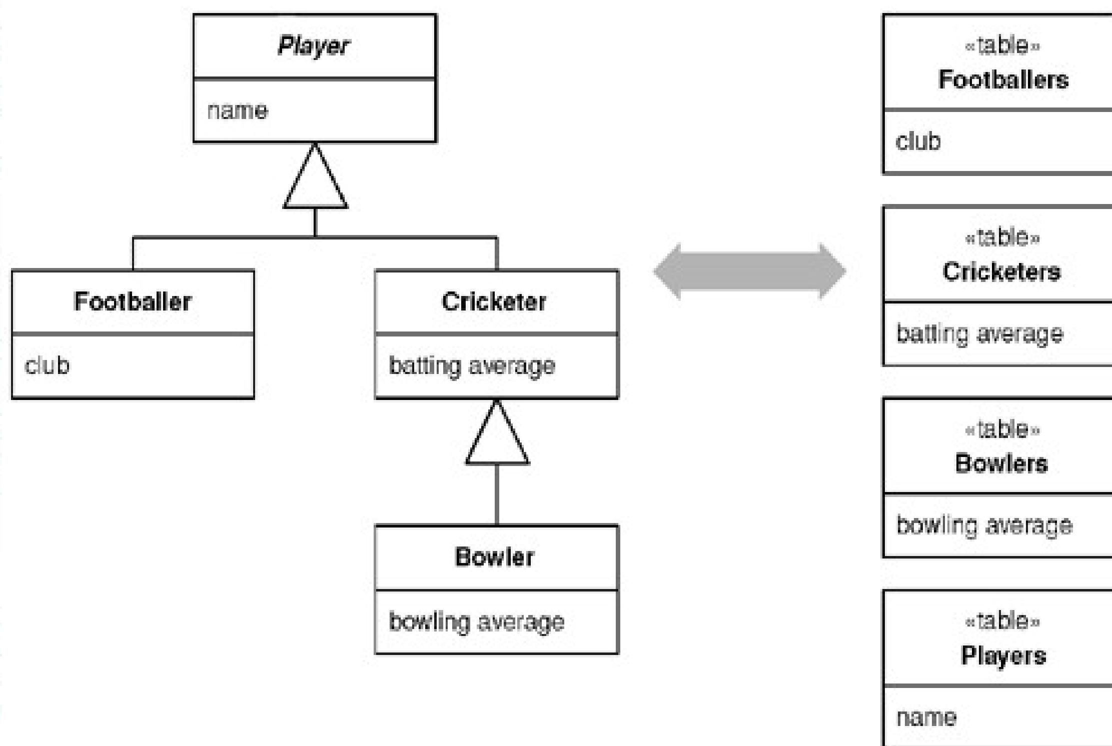


Наследование с одной таблицей (Single Table Inheritance)

- «+»
  - В структуру БД добавляется только одна таблица
  - Для извлечения данных не нужно делать JOIN
  - Перемещение полей в производный класс или суперкласс не требует внесения изменений в структуру БД
- «-»
  - «Винегрет» из полей в одной таблице
  - Расточительное использование свободного места
  - Таблица разрастается индексами, подвергается частым блокировкам
  - Одно пространство имен для свойств объектов на все классы

Наследование с таблицами для каждого класса (Class Table Inheritance)

- Представляет иерархию наследования классов, используя по одной таблице для каждого класса
- Таблицы связываются одинаковым ключом

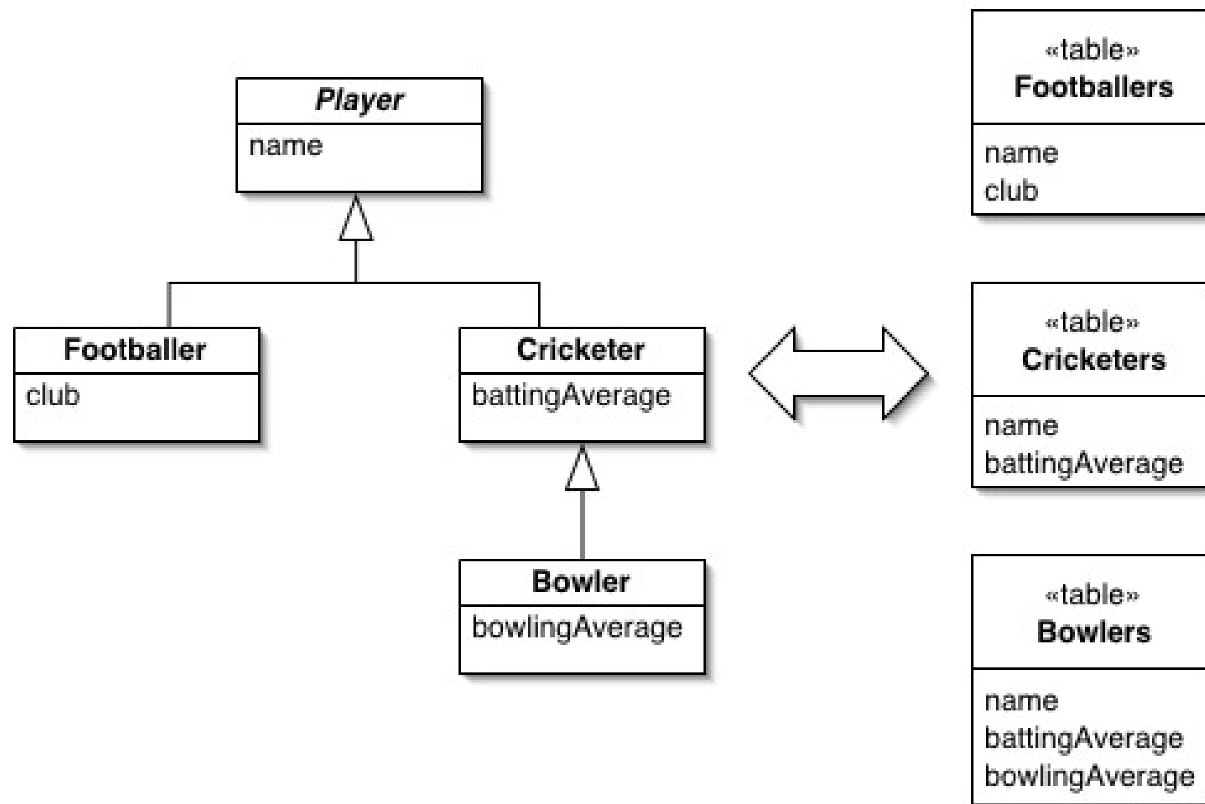


Наследование с таблицами для каждого класса (Class Table Inheritance)

- «+»
  - Поля таблицы соответствуют содержимому каждой ее строки
  - Логичная взаимосвязь между моделью домена и схемой
- «-»
  - Загрузка объекта охватывает несколько таблиц
  - Перемещение полей требует изменения структуры БД
  - Снижение производительности таблиц суперклассов

Наследование с таблицами для каждого конкретного класса (Concrete Table Inheritance)

- Представляет иерархию наследования классов, используя по одной таблице для каждого конкретного класса этой иерархии

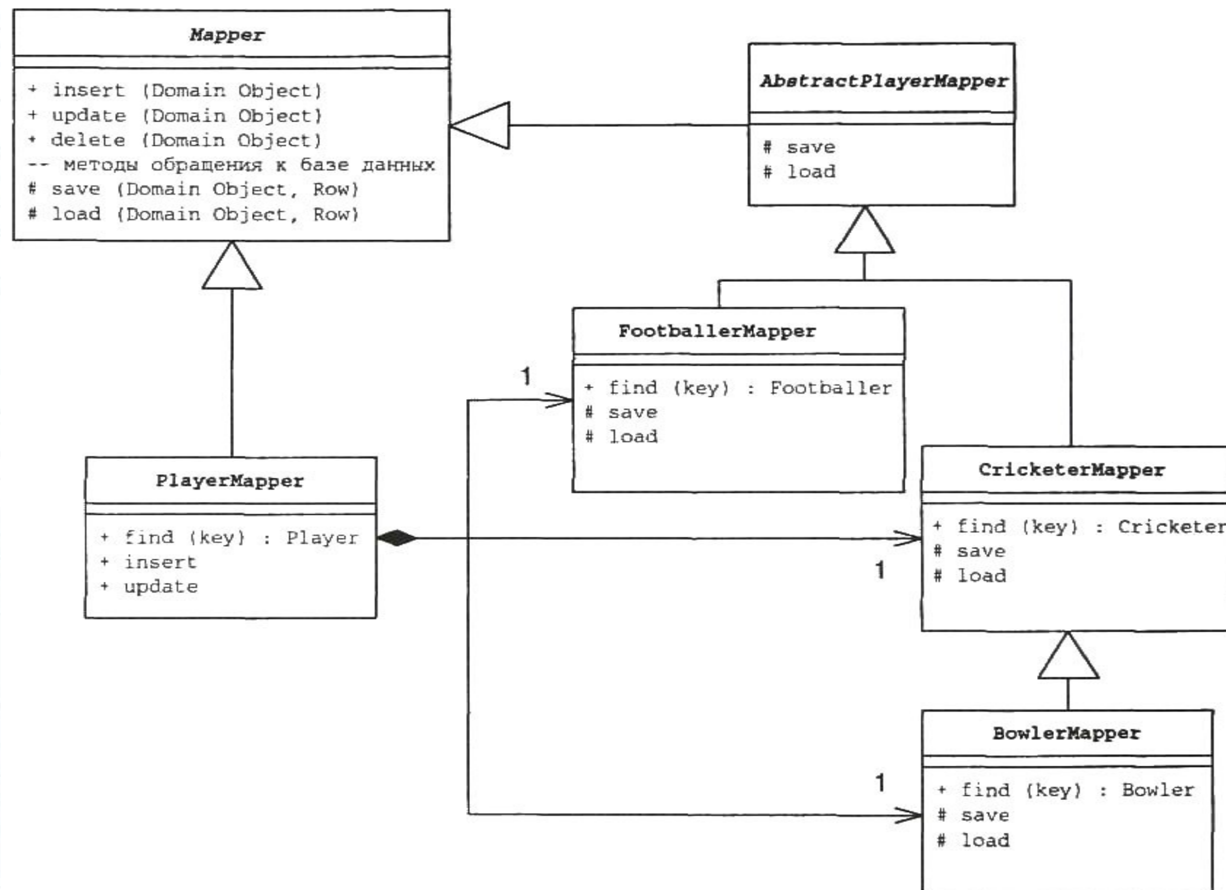


Наследование с таблицами для каждого конкретного класса (Concrete Table Inheritance)

- «+»
  - Каждая таблица является замкнутой и не содержит «лишних» полей
  - Нет необходимости в JOIN
  - Доступ к таблице необходим только при доступе к конкретному классу => распределение нагрузки
- «-»
  - Первичные ключи должны быть уникальными по всей иерархии
  - Невозможно моделировать отношения между абстрактными классами
  - Изменение поля суперкласса влечет изменения во всех подчиненных таблицах
  - Абстрактный метод поиска должен просмотреть ВСЕ производные таблицы

Преобразователь наследования (Inheritance Mappers)

- Структура, предназначенная для организации преобразователей, которые работают с иерархиями наследования



Преобразователь наследования (Inheritance Mappers)

- Методы поиска объявлены в конкретных производных классах
- Метод поиска, создав объект конкретного класса и загрузив в него данные, вызывает метод суперкласса
- Аналогично для методов обновления и вставки
- Данная схема применяется к любому типу отображения иерархии наследования

Типовые решения источников данных

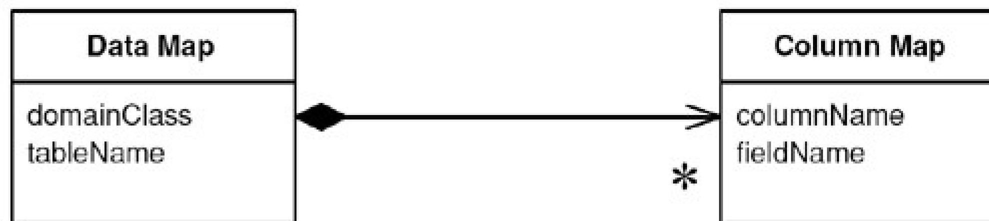
- Архитектурные типовые решения источников данных
- Объектно-реляционные типовые решения, предназначенные для моделирования
  - Поведения
  - Структуры
- **Типовые решения объектно-реляционного отображения с использованием метаданных**

Типовые решения объектно-реляционного отображения с использованием метаданных

- Отображение метаданных (Metadata Mapping)
- Объект запроса (Query Object)
- Хранилище (Repository)

Отображение метаданных (Metadata Mapping)

- Хранит описание деталей объектно-реляционного отображения в виде метаданных (пары свойство объекта — поле таблицы)
- Позволяет сократить трудоемкость реализации отображения на базу данных
- Широко используется в промышленных средствах ORM
- Отображение как правило хранится в исходном коде программы (аннотации) или внешних файлах (XML)
- Метаданные могут использоваться двумя способами: *генерация* кода и метод *отражения*



Отображение метаданных

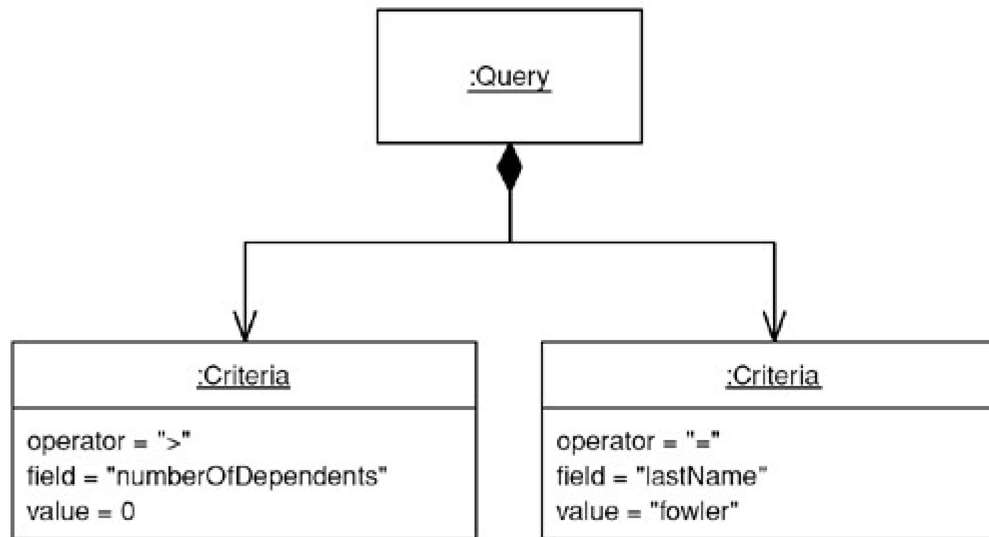
```
class DataMap...
	private Class domainClass;
	private String tableName;
	private List columnMaps = new ArrayList();

class ColumnMap...
	private String columnName;
	private String fieldName;
	private Field field;
	private DataMap dataMap;

class PersonMapper...
	protected void loadDataMap(){
		dataMap = new DataMap (Person.class, "people");
		dataMap.addColumn ("firstname", "varchar", "firstName");
		dataMap.addColumn ("number_of_dependents", "int", "numberOfDependents");
	}

class Mapper...
	public Object findObject (Long key) {
		if (uow.isLoaded(key)) return uow.getObject(key);
		String sql = "SELECT" + dataMap.columnList() + " FROM " + dataMap.getTableName() +
" WHERE ID = ?";
		PreparedStatement stmt = null; ResultSet rs = null; DomainObject result = null;
		try {
			stmt = DB.prepare(sql);
			stmt.setLong(1, key.longValue());
			rs = stmt.executeQuery(); rs.next();  result = load(rs);
		} catch (Exception e) {throw new ApplicationException (e);
		} finally {DB.cleanUp(stmt, rs);}
		return result;
	}
	private UnitOfWork uow;  protected DataMap dataMap;
```

Отображение метаданных Hibernate Annotations (JPA ORM)

```
@Entity
@Table(name="tbl_flight",
   uniqueConstraints = {@UniqueConstraint(columnNames={"number", "country"})})
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)

public class Flight implements Serializable {

      @Id @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_STORE")
      public Long getId() { ... } // Ключ

      @Transient
      String getLengthInMeter() { ... } // Временное свойство

      @Column(updatable = false, name = "flight_name", nullable = false, length=50)
      String getName() { ... } // Хранимое свойство

      @Id @ManyToOne(cascade = CascadeType.MERGE)
      Person owner;

      @Basic(fetch = FetchType.LAZY)
      String getDetailedComment() { ... } // Хранимое свойство с отложенной загрузкой

      @Embedded public Address getAddress() { ... } // Внедренное значение

      ...
}
```

Объект запроса (Query Object)

- Объект, представляющий запрос к БД
- Скрывает специализированные методы поиска внутри параметризованных методов
- Запросы не зависят от схемы БД



Объект запроса (Query Object)

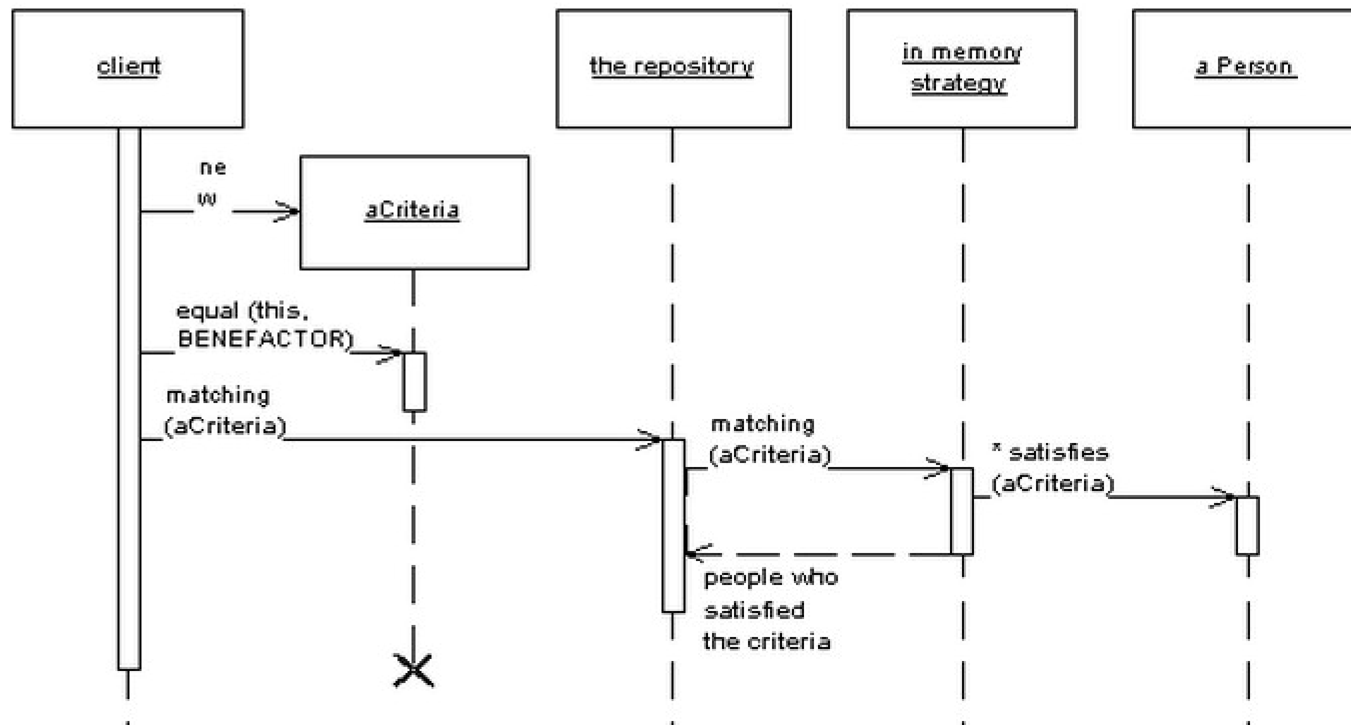
- Позволяет клиенту формировать различные запросы в терминах объектов приложения, преобразуя их в SQL-запросы
- Должен обладать определенной гибкостью
- Может усложняться по мере развития приложения
- Развязывает клиента от особенностей реализации SQL для разных СУБД
- Может быть использован для уменьшения количества запросов к БД
- Преобразует множество критериев к аргументам преобразователя данных

Объект запроса (Query Object)

```
class QueryObject...
	private Class klass;
	private List criteria = new ArrayList();

class Criteria...
	private String sqlOperator;
	protected String field;
	protected Object value;

class Criteria...
	public static Criteria greaterThan(String fieldName, int value) {
		return Criteria.greaterThan(fieldName, new Integer(value));
	}
	public static Criteria greaterThan(String fieldName, Object value) {
		return new Criteria(" &gt; ", fieldName, value);
	}
	private Criteria(String sql, String field, Object value) {
		this.sqlOperator = sql;
		this.field = field;
		this.value = value;
	}

class Person...
	private String lastName;
	private int numberOfDependents;
. . .

	QueryObject query = new QueryObject(Person.class);
	query.addCriteria(Criteria.greaterThan("numberOfDependents", 0));
	query.addCriteria(Criteria.matches("lastName", "f%"));
```

Хранилище (Repository)

- Выступает в роли посредника между слоем домена и слоем отображения данных, предоставляя интерфейс в виде коллекции для доступа к объектам домена



Хранилище (Repository)

- Располагается между слоем домена и слоем отображения данных
- Выполняет роль коллекции объектов домена в оперативной памяти
- Клиенты составляют запросы с различными критериями и отправляют их на выполнение в хранилище
- Достигается изоляция и односторонняя зависимость между слоем отображения и слоем домена
- Область применения: множество типов объектов домена и большое количество запросов

Хранилище (Repository)

```
public class Person {
	public List dependents() {
		return Registry.personRepository().dependentsOf(this);
	}
}

public class PersonRepository extends Repository {
	public List dependentsOf(aPerson) {
		Criteria criteria = new Criteria();
		criteria.equal(Person.parent, aPerson);
		return matching(criteria);
	}
}

abstract class Repository {
	private RepositoryStrategy strategy;
	protected List matching(aCriteria) {
		return strategy.matching(aCriteria);
	}
}

public class RelationalStrategy implements RepositoryStrategy {
	protected List matching(Criteria criteria) {
		Query query = new Query(myDomainObjectClass())
		query.addCriteria(criteria);
		return query.execute();
	}
}
```

далее..

Типовые решения, предназначенные для представления данных в Web