Проектирование архитектур программного обеспечения

лекция 4

Зозуля А.В.

КСПТ

2016 г.

Ранее..

- Паттерны проектирования
- Порождающие, структурные и поведенческие паттерны
- Антипаттерны

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

Архитектурные типовые решения источников данных

- Очищение слоя бизнес-логики от SQL-операторов
- Переносимость слоя доступа к данным
- Локализация SQL для администраторов СУБД
- Отделение объектов предметной области от способа их хранения

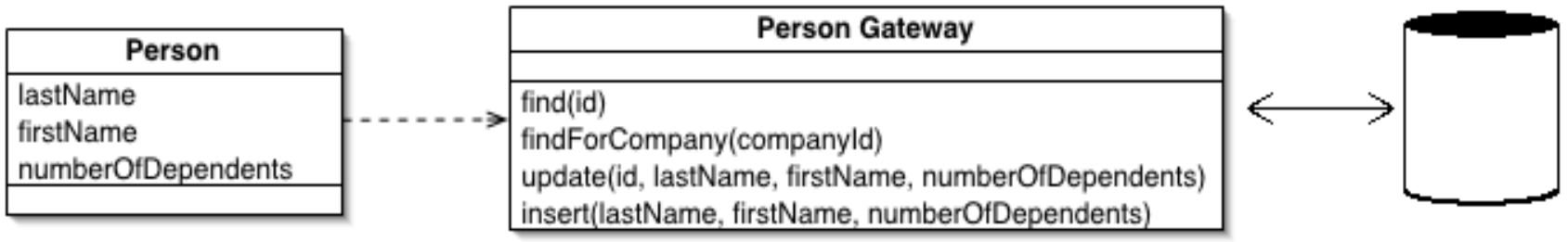
Архитектурные типовые решения источников данных

- Шлюз таблицы данных (Table Data Gateway)
- Шлюз записи данных (Row Data Gateway)
- Активная запись (Active Record)
- Преобразователь данных (Data Mapper)

Шлюз таблицы данных (Table Data Gateway)

- Объект, выполняющий роль шлюза к БД
- Содержит все SQL-команды (SUID) обработки данных одной таблицы или представления
- Каждый метод передает параметры в SQL-команду
- Не имеет состояния
- Результат — Множество записей (RecordSet) или объекты предметной области

Шлюз таблицы данных (Table Data Gateway)




```
class PersonGateway {

    public IDataReader findAll() {
      String sql = "select * from person";
      return new OleDbCommand(sql, DB.Connection).ExecuteReader();
    }

    public IDataReader findWithLastName(String lastName) {
      String sql = "SELECT * FROM person WHERE lastname = ?";
      IDbCommand comm = new OleDbCommand(sql, DB.Connection);
      comm.Parameters.Add(new OleDbParameter("lastname", lastName));
      return comm.ExecuteReader();
    }

    public IDataReader findWhere(String whereClause) {
      String sql = String.Format("select * from person where {0}", whereClause);
      return new OleDbCommand(sql, DB.Connection).ExecuteReader();
    }
```

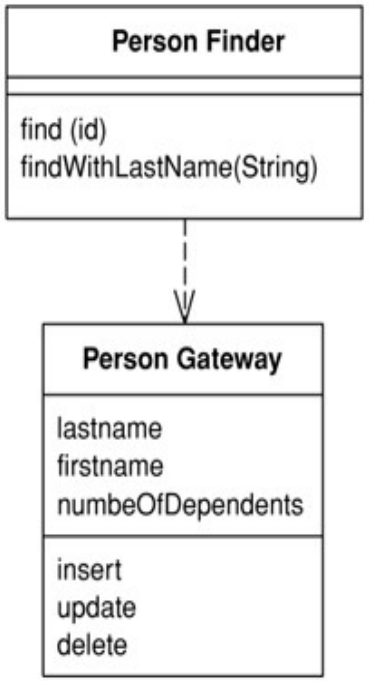
Применение Шлюза таблицы данных

- Наиболее простое типовое решение
- Инкапсулирует логику доступа к источнику данных
- Хорошо сочетается с типовыми решениями Модуль таблицы (Table Module) и Сценарий транзакции (Transaction Script)
- Плохо сочетается с Моделью предметной области
- Подходит для обработки множественных результатов
- Удобный доступ к хранимым процедурам

Шлюз записи данных (Row Data Gateway)

- Объект, выполняющий роль шлюза к отдельной записи источника данных (таблицы, представления,..)
- Все детали источника данных скрыты за интерфейсом
- Столбец записи — поле объекта
- Хорошо сочетается с типовым решением Сценарий транзакции (Transaction Script)
- Функции поиска выносятся в отдельный класс
- Не содержит бизнес-логики

Шлюз записи данных (Row Data Gateway)



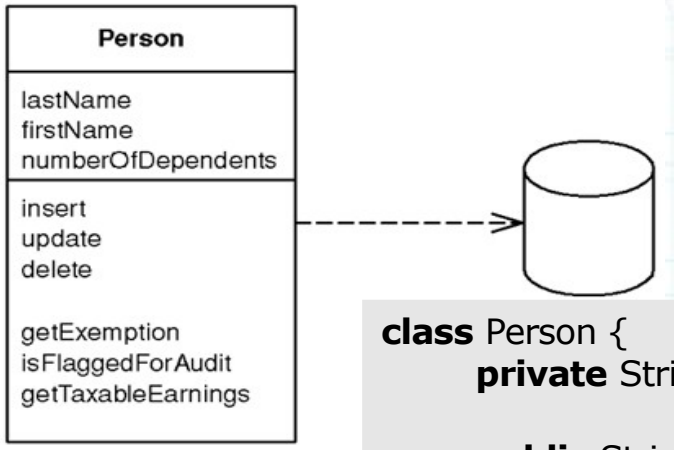

```
class PersonGateway {
        private String lastName;
        private String firstName;

        public String getLastName() {return lastName;}
        public void setLastName(String lastName) {this.lastName = lastName;}
        public String getFirstName() {return firstName;}
        public void setFirstName(String firstName) {this.firstName = firstName;}


        final String insertStatementString = "INSERT INTO people VALUES (?,?,?,?)";

        public Long insert() {
                PreparedStatement insertStatement = null;
                try {
                        insertStatement = DB.prepare(insertStatementString);
                        setID(findNextDatabaseId());
                        insertStatement.setInt (1, getID().intValue());
                        insertStatement.setString(2, lastName);
                        insertStatement.setString(3, firstName) ;
                        insertStatement.execute() ;
                        Registry.addPerson(this) ;
                        return getID();
                } catch (SQLException e) { throw new ApplicationException (e);
                } finally { DB.cleanup(insertStatement); }
        }
```

Применение Шлюза записи данных

- Успешно отделяет структуру БД от бизнес-логики
- Хорошо сочетается с Сценарием транзакции
- Не используется с Моделью предметной области из-за «троирования» представления объектов
- Требует удвоения классов по сравнению со Шлюзом таблицы данных

Активная запись (Active Record)

- Объект, выполняющий роль оболочки для строки таблицы или представления
- Добавляет к данным логику домена (данные + поведение)
- Отвечает за сохранение и загрузку информации в БД
- Включает методы:
    - Создание объекта на основе результата SQL
    - Создание объекта для вставки в таблицу
    - Статически методы поиска, возвращающие активные записи
    - Обновление БД из данных полей
    - Фрагменты бизнес-логики



Активная запись (Active Record)

```
class Person {
        private String lastName;

        public String getLastName() {return lastName;}
        public void setLastName(String lastName) {this.lastName = lastName;}

        final String insertStatementString = "INSERT INTO people VALUES (?,?,?,?)";

        public Long insert() {
                PreparedStatement insertStatement = null;
                try {
                        insertStatement = DB.prepare(insertStatementString);
                        ...
                        insertStatement.execute() ;
                        Registry.addPerson(this) ;
                        return getID();
                } catch (SQLException e) { throw new ApplicationException (e);
                } finally { DB.cleanup(insertStatement); }
        }

        public Money getExemption() {
                Money baseExemption = Money.dollars(1500);
                Money dependentExemption = Money.dollars(750);
                return baseExemption.add(
                        dependentExemption.multiply(this.getNumberOfDependents()));
        }
```

Применение Активной записи

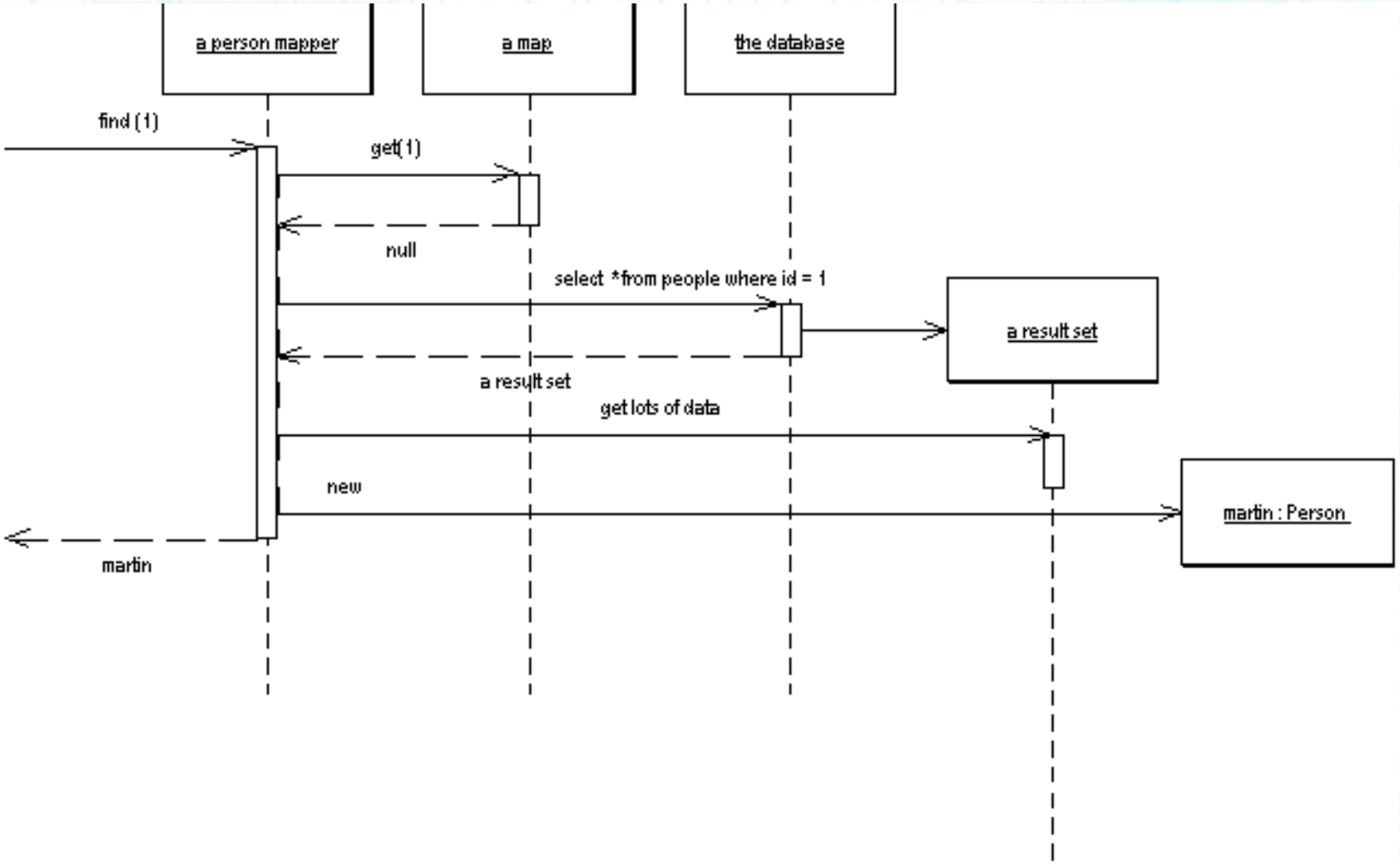
- Реализация несложной логики домена
- Простота реализации
- Зависимость структуры объектов от БД
- Удачно применяется, когда поля точно отображаются на столбцы таблиц
- Плохо поддерживается наследование
- Хорошо сочетается с Сценарием транзакции и с Моделью предметной области

Преобразователь данных (Data Mapper)

- Осуществляет передачу данных между объектами и базой данных
- Объекты ничего не «знают» о существовании БД, БД ничего не «знает» об использующих ее объектах
- Используется Коллекция объектов (Identity Map) для проверки, загружены ли уже запрашиваемые объекты
- Не требуется однозначного соответствия столбцов таблицы и полей объекта
- Как узнать, какие объекты изменены, какие созданы, а какие удалены? => решение Единица работы (Unit of Work)
- Загрузка одного объекта может повлечь за собой загрузку множества объектов => Загрузка по требованию (Lazy Load)

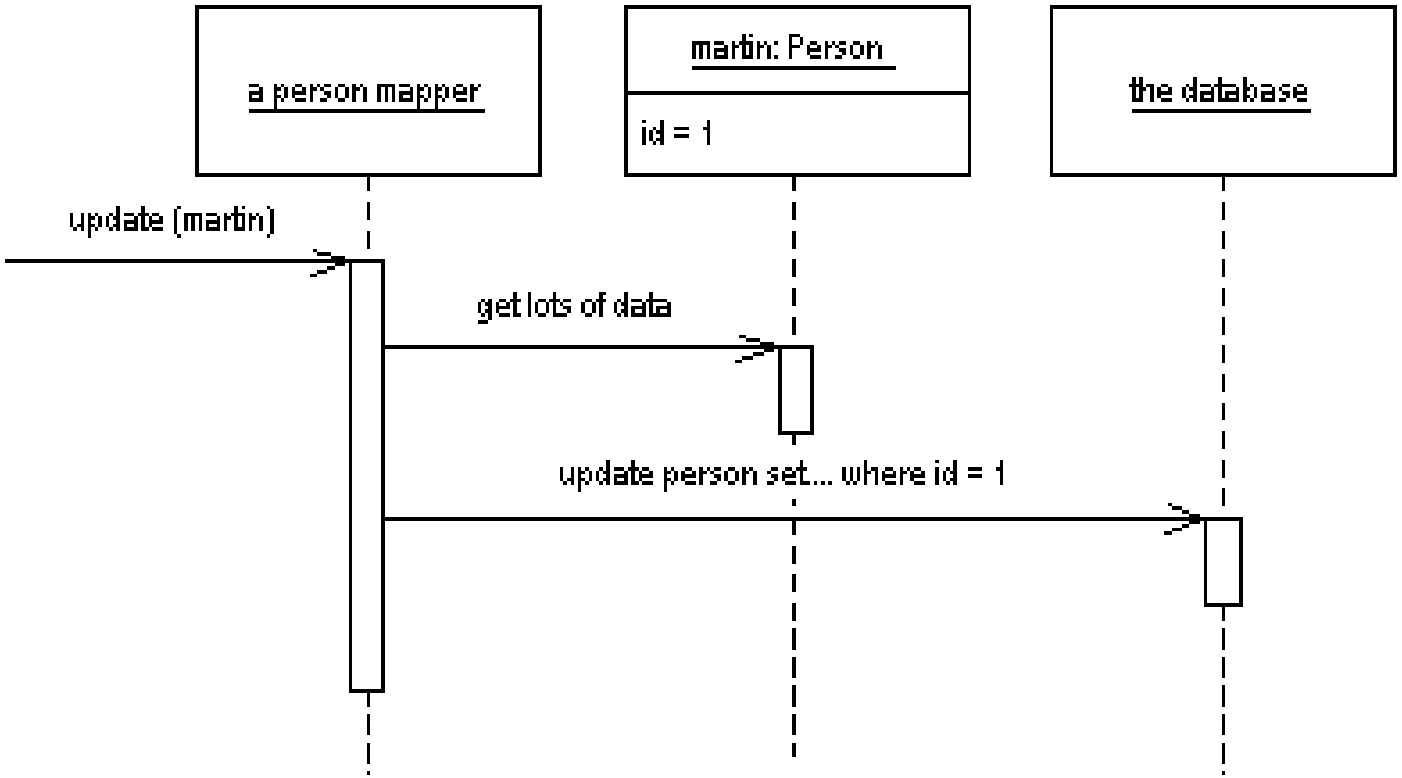
Преобразователь данных (Data Mapper)

Извлечение данных



Преобразователь данных (Data Mapper)

Обновление данных



Преобразователь данных (Data Mapper)

```
class PersonMapper ... {
   protected String findStatement() {return "SELECT " + COLUMNS + "  FROM people" + "
WHERE id = ?";}
   public static final String COLUMNS = " id, lastname, firstname, num_of_dependents ";
   public Person find(Long id) {return (Person) abstractFind(id);}
}

class AbstractMapper {
   protected Map loadedMap = new HashMap();
   abstract protected String findStatement();

   protected DomainObject abstractFind(Long id) {
      DomainObject result = (DomainObject) loadedMap.get(id);
      if (result != null) return result;
      PreparedStatement findStatement = null;
      try {
         findStatement = DB.prepare(findStatement());
         findStatement.setLong(1, id.longValue());
         ResultSet rs = findStatement.executeQuery();
         rs.next();
         result = load(rs);
         return result;
      } catch (SQLException e) {throw new ApplicationException(e);
      } finally {DB.cleanUp(findStatement);}
  }
```

Преобразователь данных (Data Mapper)

```
class AbstractMapper {

   protected DomainObject load(ResultSet rs) throws SQLException {
      Long id = new Long(rs.getLong(1));
      if (loadedMap.containsKey(id)) {
         return (DomainObject) loadedMap.get(id);
      }
      DomainObject result = doLoad(id, rs);
      loadedMap.put(id, result);
      return result;
   }

   abstract protected DomainObject doLoad(Long id, ResultSet rs) throws SQLException;

}

class PersonMapper ... {

   protected DomainObject doLoad(Long id, ResultSet rs) throws SQLException {
      String lastNameArg = rs.getString(2);
      String firstNameArg = rs.getString(3);
      int numDependentsArg = rs.getInt(4);
      return new Person(id, lastNameArg, firstNameArg, numDependentsArg);
   }

}
```

Преобразователь данных (Data Mapper)

```
class AbstractMapper {
   public Long insert(DomainObject subject) {
      PreparedStatement insertStatement = null;
      try {
         insertStatement = DB.prepare(insertStatement());
         subject.setID(findNextDatabaseId());
         insertStatement.setInt(1, subject.getID().intValue());
         doInsert(subject, insertStatement);
         insertStatement.execute();
         loadedMap.put(subject.getID(), subject);
         return subject.getID();
      } catch (SQLException e) {throw new ApplicationException(e);}
      finally {DB.cleanUp(insertStatement);}
   }
   abstract protected String insertStatement();
   abstract protected void doInsert(DomainObject subject, PreparedStatement
InsertStatement) throws SQLException;
}

class PersonMapper ... {
   protected String insertStatement() {return "INSERT INTO people VALUES (?, ?, ?, ?)";}
   protected void doInsert(DomainObject abstractSubject,PreparedStatement stmt)
throws SQLException {
      Person subject = (Person) abstractSubject;
      stmt.setString(2, subject.getLastName()); stmt.setString(3, subject.getFirstName());
      stmt.setInt(4, subject.getNumberOfDependents());
}
```

Применение Преобразователя данных

- Схема БД и объектная модель могут изменяться независимо друг от друга
- Удачно используется с Моделью предметной области
- Необходимо написать отдельный слой кода
- Подходит для сложной бизнес-логики
- Существует достаточное количество промышленных реализаций

Сравнение архитектурных типовых решений источников данных

	Сценарий транзакции	*Модуль таблицы*	*Модель предметной области*
Шлюз таблицы данных	+/-	+	+/-
Шлюз записи данных	+	+/-	-
Активная запись	+	-	+
Преобразователь данных	-	-	+

Типовые решения источников данных

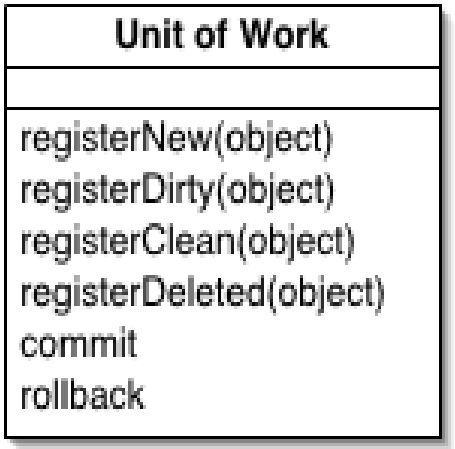
- Архитектурные типовые решения источников данных
- Объектно-реляционные типовые решения, предназначенные для моделирования
  - **Поведения**
  - Структуры
- Типовые решения объектно-реляционного отображения с использованием метаданных

Объектно-реляционные типовые решения, предназначенные для моделирования поведения

- Единица работы (Unit of Work)
- Коллекция объектов (Identity Map)
- Загрузка по требованию (Lazy Load)

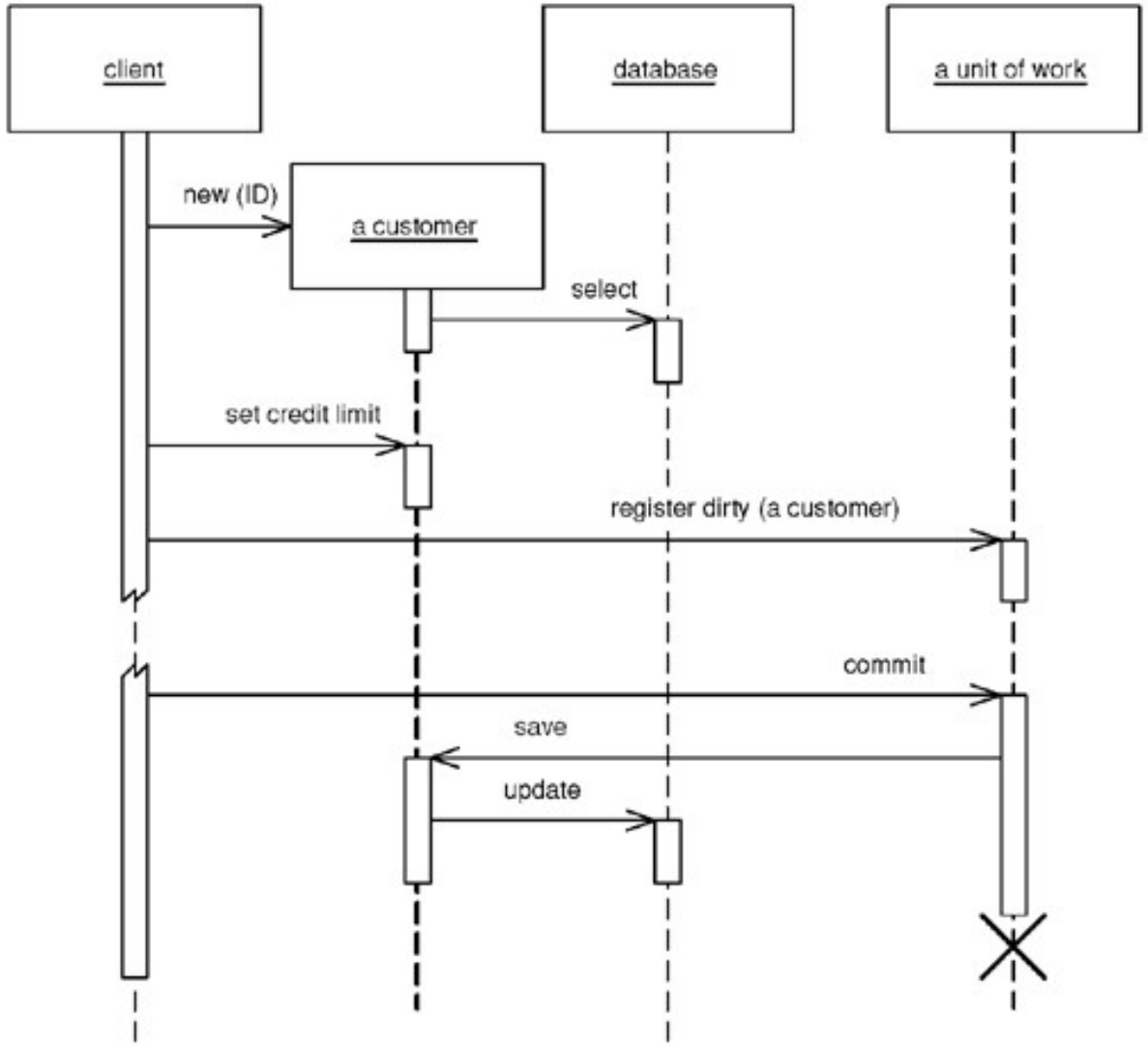
Единица работы (Unit of Work)

- Содержит список объектов, охватываемых бизнес-транзакцией
- Координирует запись изменений в БД
- Разрешает проблемы параллелизма
- Все объекты должны быть зарегистрированы
- Удобно единицу работы размещать в объекте сеанса



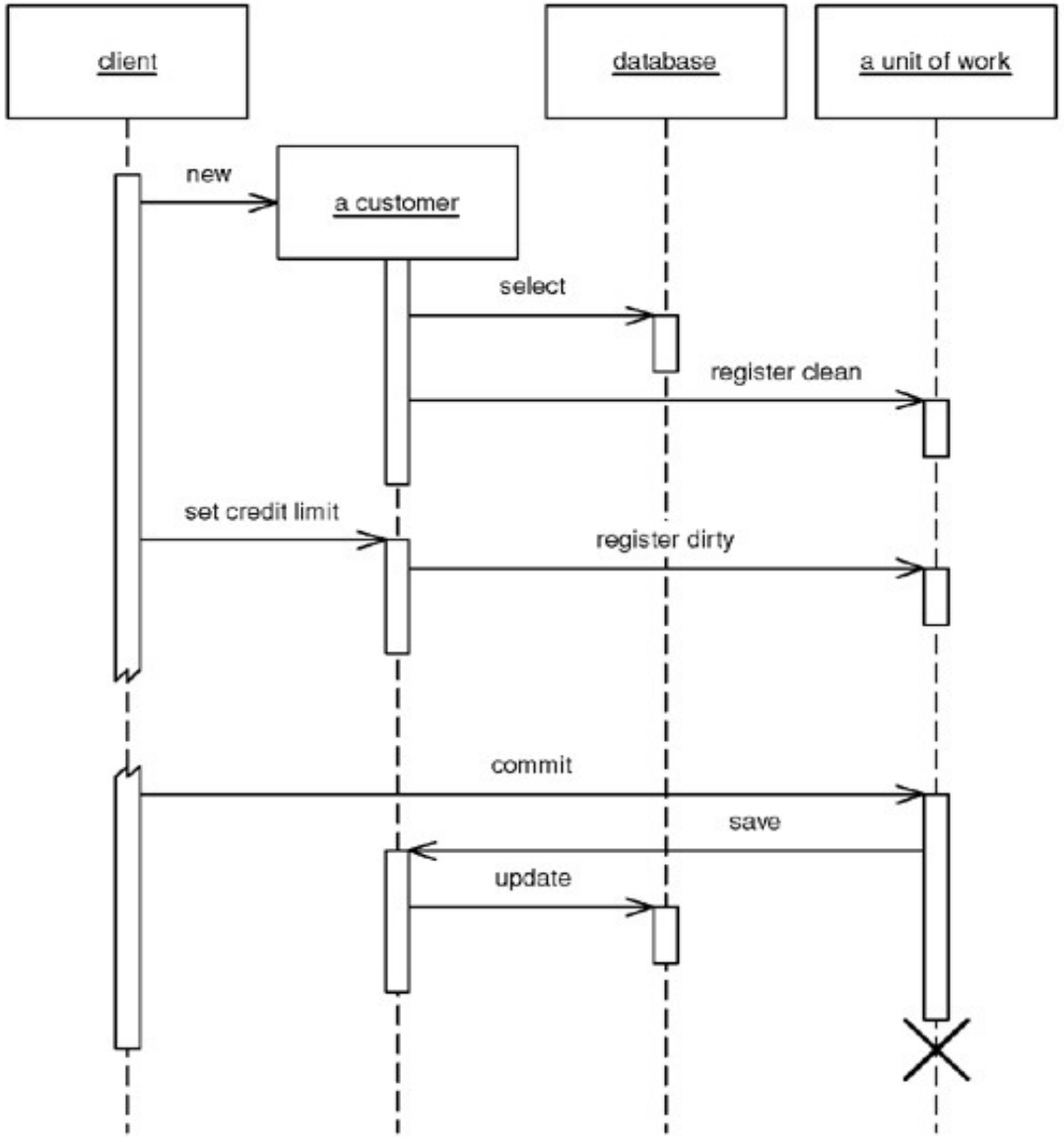
Единица работы (Unit of Work)

Регистрация изменяемого объекта вызывающим оператором



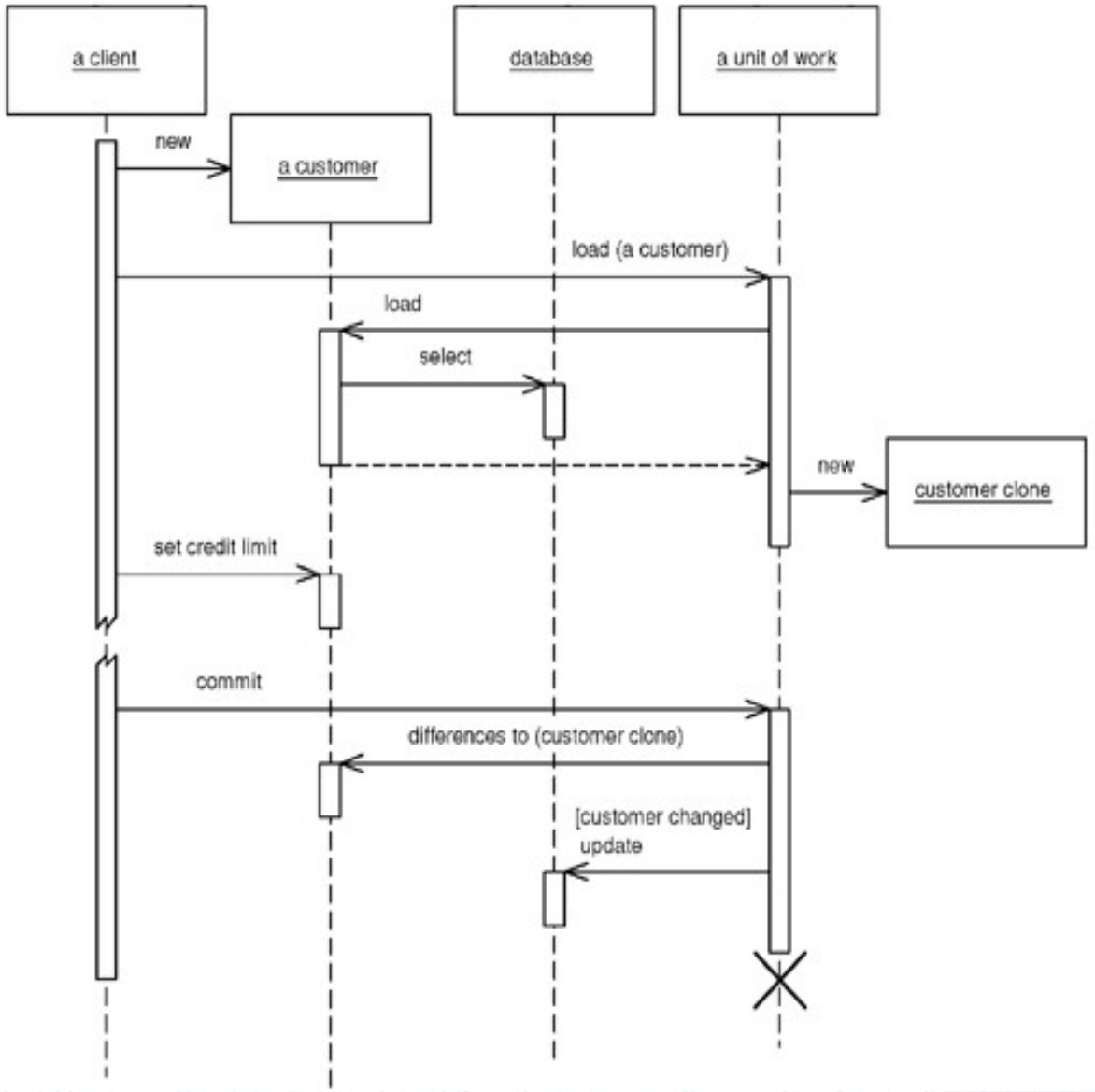
Единица работы (Unit of Work)

Регистрация изменяемого объекта его же методами



Единица работы (Unit of Work)

Использование единицы работы в качестве контроллера доступа к БД



Единица работы (Unit of Work)

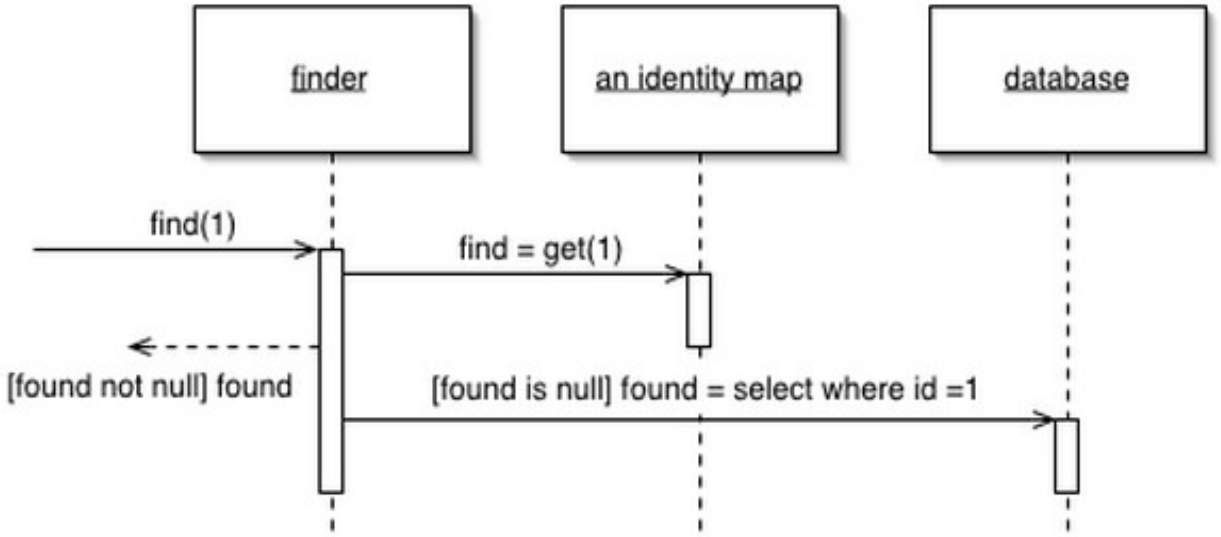
```
class UnitOfWork {
    public void registerNew(DomainObject obj) {
        Assert.isTrue("object not dirty", !dirtyObjects.contains(obj));
        Assert.isTrue("object not removed", !removedObjects.contains(obj));
        Assert.isTrue("object not already registered new", !newObjects.contains(obj));
        newObjects.add(obj);
    }
    public void registerDirty(DomainObject obj) {
        Assert.isTrue ("object not removed", !removedObjects.contains(obj));
        if (!dirtyObjects.contains(obj) && !newObjects.contains(obj))
            dirtyObjects.add(obj);
    }
    public void registerRemoved(DomainObject obj) {
        if (newObjects.remove(obj)) return;
        dirtyObjects.remove(obj);
        if (!removedObjects.contains(obj)) {removedObjects.add(obj);}
    }
    public void registerClean(DomainObject obj) {}
    public void commit() {
        insertNew();
        updateDirty();
        deleteRemoved();
    }
    private void insertNew() {
        for (DomainObject obj : newObjects)
            MapperRegistry.getMapper(obj.getClass()).insert(obj);
    }
```

Применение Единицы работы

- Синхронизация данных объектов и БД по окончанию бизнес-транзакции
- Более простое, но ресурсоемкое решение: сохранять после каждого изменения
- Хорошо сочетается с Моделью предметной области

Коллекция объектов (Identity Map)

- Гарантирует, что каждый объект будет загружен из БД только один раз
- Сохраняет загруженный объект в специальной коллекции на время бизнес-транзакции
- При получении запроса просматривает коллекцию в поисках нужного объекта

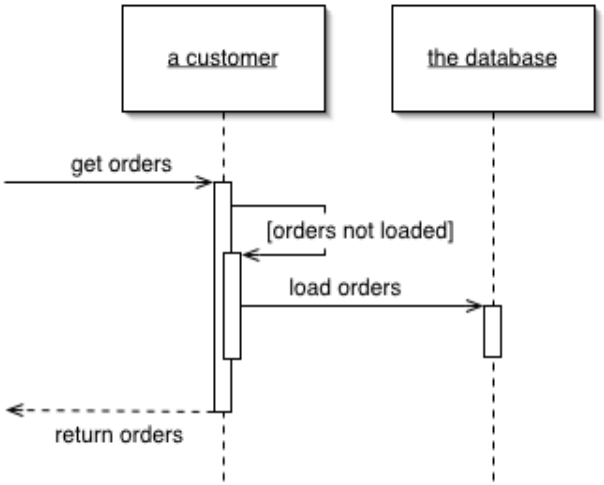


Коллекция объектов (Identity Map)

- Для каждой таблицы может быть своя Коллекция
- Для Коллекции требуется выбор ключевого поля
- Коллекция может быть:
  - Явной: findPerson(1);
  - Универсальной: find(«Person», 1);
- Каждый сеанс должен иметь свою Коллекцию
- Удобно Коллекцию размещать в Единице работы
- Не допускает наличия двух объектов, соответствующих одной записи БД
- Может использоваться для кеширования записей БД

Загрузка по требованию (Lazy load)

- Объект, который не содержит все требующиеся данные, но загружает их в случае необходимости
- Способы реализации:
  - **Инициализация по требованию**
  - **Виртуальный прокси-объект**
  - Диспетчер значения
  - Фиктивный объект



- Используется, когда данные распределены между несколькими таблицами и для загрузка поля требуется отдельный SQL-запрос
- Применяется для «крупных» полей
- Значительно усложняет приложение

Загрузка по требованию

Инициализация по требованию

- При каждой попытке доступа к полю выполняется проверка на NULL
- Если NULL, метод доступа загружает объект и возвращает его клиенту
- Обращение к полю только через get()-методы
- Требует дополнительной зависимости между объектом и БД => не подходит для решения на основе Преобразователя данных

```
class Supplier {
      public List getProducts() {
            if (products == null)
                  products = Product.findForSupplier(getID()) ;
            return products;
      }
}
```

Загрузка по требованию

Виртуальный прокси-объект

- Имитирует объект, являющийся значением поля, но ничего не содержит
- Загрузка реального объекта будет выполнена когда будет вызван один из его методов
- Одному реальному объекту может соответствовать несколько виртуальных => переопределение методов equals()
- Источник ошибок

Далее..

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