Санкт-Петербургский государственный университет

**Кафедра моделирования экономических систем**

**Власов Николай Юрьевич**

**Выпускная квалификационная работа бакалавра**

# Сравнительный анализ архитектурных решений в задаче хранения данных и создание приложения для работы с базой данных

Направление 010400

Прикладная математика и информатика

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Ковшов А. М.

Санкт-Петербург

2016

# Содержание

# Введение

СПбГУ в лице сотрудников биологического, геологического, а также факультета ПМ - ПУ ведет активную работу по сохранению объектов культурного наследия. Специалистами были исследованы и отреставрированы многочисленные памятники Санкт - Петербурга.

Накопленная в результате многолетних исследований проектная документация, представленная в виде паспортов, отчетов, фотографий и других документов, хранится в архивах.

Необходимость разработки и внедрения электронной базы данных возникает естественным образом: специалисты в области сохранения объектов культурного наследия нуждаются в инструменте для электронного структурированного хранения и обработки проектной документации. База данных (а также приложение для работы с ней) позволит автоматизировать процессы регистрации, обновления и извлечения информации.

Целью данной выпускной квалификационной работы является создание базы данных, удовлетворяющей запросам специалистов по сохранению объектов культурного наследия, а также разработка приложения для работы с указанной базой данных.

Приведем основные понятия и положения, используемые в работе, необходимые для дальнейшего понимая работы.

Под уже использованным термином «**данные**» будем понимать многократно интерпретируемое представление информации в формализованном виде, доступном для коммуникации и обработки [1].

Предполагается, что данные в базе являются постоянно хранимыми, так как после того как они были обработаны средствами системы управления базой данных для внесения в базу данных, их удаление возможно только с помощью явного запроса к базе данных.

Под **базой данных** будем подразумевать совокупность постоянно

хранимых данных, используемых прикладным программным обеспечением.

К.Дж. Дейт определял [2] понятие **модели данных** как совокупность следующих трех элементов: набор типов данных, набор правил целостности и набор операций, применимых к типам данных. Таким образом, модель данных задает структуру данных, методы манипулирования ими, а также их ограничения целостности.

Предполагается, что создание базы данных, ее поддержка и обеспечение доступа пользователей к ней осуществляется с помощью отдельного комплекса языковых и программных средств называемого **СУБД** - система управления базой данных.

Следует также упомянуть понятия файла и файловой системы.

**Ф а й л** - информация, хранимая на электронном носителе рассматриваемая как единое целое [5].

СУБД работает под управлением **файловой системы —** программного обеспечения, в задачи которого входит организация структуры данных,хранящимся в файле, обеспечение доступа к ним, распределению памяти и отображение имен файлов в соответствующие адреса памяти [5].

Под **предметной областью** будем понимать представление в БД объектов реального мира (с учетом их связей и ограничений), относящихся к определенной области знаний и обладающих практической ценностью для пользователя.

**Приложение** — программное обеспечение, функции которого заключаются в выполнении типовых работ с БД, в том числе, посредством СУБД.

# Постановка задачи

Для достижения поставленной цели требуется решить следующие задачи:

- провести полный цикл проектирования базы данных, включающий в себя сбор и всесторонний анализ информации, представляющей предметную область, формулирование требований предъявляемых к базе данных, получение проектных решений, готовых для дальнейшей программной реализации, а также проведение сравнительного анализа этих решений с целью окончательного выбора;
- предоставить программную реализацию базы данных на основе выбранного проектного решения;
- предоставить программную реализацию приложения для работы с базой данных.

База данных должна отвечать следующим общим требованиям:

- независимость данных. Любые изменения технического или программного обеспечения, а также характера хранения данных, не должны менять представления пользователей о базе данных.
- Совместное использование базы данных многими пользователями.
- Безопасность данных.
- Адекватность отображения данных предметной области.
- Другие, специфические требования, которые будут сформированы в параграфе 1.2.

Приложение должно предоставлять функции создания, обновления, чтения и удаления данных из базы данных с учетом требуемой структуры, а также обеспечивать возможность пересылать файлы.

# Обзор литературы

Сфере проектирования баз данных посвящено большое количество литературы [2-5,9,10] . В учебном пособии С. Кузнецова [5] представлен небольшой исторической экскурс, описывающий первые способы организации хранения данных на электронных вычислительных системах, изложены предпосылки возникновения баз данных и СУБД. Вопросу реляционной модели данных, как наиболее востребованной модели данных на рынке СУБД, посвящены книги К. Дж. Дейта [2-3], С. Кузнецова [5]. В них подробно описаны понятия реляционных баз данных, изложены основные положения и свойства отношений, а также рассмотрены два основополагающих способа оперированием данными: реляционная алгебра и реляционное исчисление. Кроме того, даются основы языка запросов SQL вместе с примерами.

Одним из этапов проектирования базы данных являлось создание семантической модели данных с помощью языка объектного моделирования UML. Спецификация языка доступна на сайте консорциума, занимающегося разработкой этого языка [7].

Для удовлетворения некоторых требований, предъявляемых к базе данных при отображении предметной области, приходится рассматривать несколько архитектурных решений. Книги Б.Карвина [9] и М.Фоулера [10] освещают возможные пути решения некоторых проблем, возникающих в процессе проектирования баз данных.

В процессе создания базы данных была использована реляционная СУБД свободной лицензии MySQL. Информация о технических характеристиках и возможностях, предоставляемой этой СУБД, была получена из учебного пособия [11] , справочной документации на сайте производителя [12] , а также из курса по базам данных, проводившемся на факультете ПМ — ПУ.

В этой работе разрабатывается серверное приложение на основе программной платформы Java. Описание сферы применения, принципов создания и использования сервлетов (приложений) указано в книгах [13,14] а также на сайте компании — поставщика программного обеспечения [15].

# Глава 1. Проектирование базы данных

В этой главе пойдет речь о проектировании баз данных. В первом параграфе будет кратко рассмотрена история развития баз данных, обсуждаются подходы к решению задачи проектирования, рассматриваются различные концепции БД. Во втором параграфе представлены варианты проектов непосредственно базы данных объектов культурного наследия, рассматриваются их свойства, отличительные особенности. В конце параграфа указывается финальный вариант для программной реализации.

## 1.1 Концепции построения баз данных

Прежде чем перейти к используемой методологии проектирования БД, кратко рассмотрим историю развития БД.

Первые ЭВМ имели всего 2 вида устройств внешней памяти – магнитные ленты и магнитные барабаны. Магнитные ленты обладали большей, по сравнению с барабанами, емкостью, однако их недостатком был тот факт, что ленты предоставляли последовательный доступ к данным. Магнитные барабаны обеспечивали возможность достаточно быстрого произвольного доступа к памяти, но объем хранимой на них информации был сравнительно небольшой. В то время говорить о системе управления данными во внешней памяти не приходилось и вся работу с памятью приходилось выполнять прикладной программе.

Следующим важным шагом к организации хранения данных во внешней памяти компьютера было создание файловых систем. Файловые системы обычно обеспечивают хранение слабо структурированной информации (к примеру, использование файлов для хранения текста). Сложность логической структуры файла определяется прикладным обеспечением, но, тем не менее, файловой системе она не видна.

Файловые системы находили свое применение в решении инженерных задач, содержащих малое количество данных и существенный объем

вычислений. Данные были организованы последовательно, имело место их высокая избыточность, логическая и физическая структуры данных были идентичны. Попытки применения файловых систем для экономико-управленческих задач, характеризующимися внушительными объемами данных и небольшой долей вычислений, выявили необходимость в упорядочении данных, введении компонента, ответственного за управление сложно структурированными данными. Это послужило толчком для создания СУБД.

Первые СУБД базировались на сетевой и иерархической моделях данных. Описание этих моделей представляет собой тему отдельной работы, поэтому здесь это не будет рассмотрено. Поворотным моментом в истории баз данных стала предложенная Э. Коддом в 1970 году реляционная модель данных. Неформальным синонимом к понятию «отношение» (relation) обычно указывают [3-5] термин «таблица».

Реляционная модель включает в себя следующие основополагающие компоненты:

- данные в БД представляются собой набор отношений;
- на отношения (таблицы) наложены декларативные ограничения целостности уровня типа данных (домена), уровня отношения и уровня БД;
- посредством реляционной алгебры и реляционного исчисления описываются операции над отношениями.

Реляционная модель обрела успех и заняла лидирующее положение на рынке СУБД; во многом этот успех объясняется строгостью математического аппарата теории множеств, логики первого порядка. Однако, уже в середине восьмидесятых годов в сфере управления информационными системами появились задачи, для которых реляционная модель не предлагала эффективного решения. Это стало толчком для развития других моделей данных, в том числе объектно-ориентированной.

В 1991 был организован консорциум разработчиков ООБД ODMG (Object Data Management Group) в цели которого входило создание стандартов объектных баз данных [6]. За все время работы этого консорциума было выпущены три версии предложений стандарта, на которых были основаны некоторые ООСУБД. Следует, однако, отметить, что для ООБД не существует общепринятого формального стандарта описания модели данных, в связи с чем этот тип баз данных не имеет такого широкого распространения как реляционная.

## 1.2 Сравнение вариантов проекта базы данных

Первый шаг в процессе проектирования базы данных состоит из описания информационных потребностей пользователей базы данных. База данных должна быть отражением этих потребностей; кроме того, следует принимать во внимание, что база данных есть общий ресурс для всех пользователей, а каждый пользователь может нуждаться в своем, отличном от иных, представлении данных, хранимых в БД. Для разрешения этой проблемы многообразия взглядов на базу данных большая часть современных СУБД основана на трехуровневой архитектуре ANSI-SPARC. Поэтому и проектирование баз данных обычно разделяют на три этапа, составляющих суть указанной выше архитектуры: концептуальное, логическое и физическое.

Этап концептуального проектирования проводится с целью создания семантической модели данных, отображающей представление пользователей о предметной области. На этом этапе производится проектирование БД без оглядки на используемое аппаратное обеспечение, язык программирования и т.д. Здесь также не рассматриваются компромиссные решения, приносящие, к примеру, выигрыш в производительности. Концептуальное проектирование включает в себя описание объектов, процессов, понятий представляющих предметную область. Выявляются сущности (различимые объекты, которые

требуется хранить в базе данных), устанавливаются типы связей между сущностями, определяются классы принадлежности сущности. Создается семантическая модель БД, в том числе с помощью языков семантического моделирования. Эта модель содержит сущности, связи и ограничения между ними.

Логическое проектирование проводится с целью преобразования концептуальной модели в логическую с учетом выбранной модели данных. При этом логическая модель данных предполагается не зависящей от используемой в дальнейшем СУБД. Логическое проектирование представляет собой схему БД на основе выбранной модели данных, к примеру, реляционной модели данных. Учитываются теоретические принципы выбранной модели данных, разрешаются сформированные к БД требования, в контексте выбранной модели данных. Результатом логического проектирования является готовая к программной реализации модель БД.

Перед этапом физического проектирования стоит задача разрешения конкретных аппаратных и программных требований: как расположить данные во внешней памяти, какие дополнительные структуры данных (к примеру, индексы) стоит использовать. Физическое проектирование зависит от специфики конкретной СУБД.

Перейдем к описанию требований, предъявляемых к проектируемой БД. Требования делятся на две части: общие, рассмотренные в постановке задачи, и специфические. Специфические требования формируются с помощью интервьюирования специалистов - пользователей, представляющих предметную область. Все необходимые требования описываются в доступных и понятных как проектировщику, так и пользователю терминах.

Перечислим сформированные специфические требования:

- база данных должна учитывать формы каталогизации, принятые в законодательстве РФ. К примеру, в базе данных должны быть разделы соответствующие научно-исследовательским, изыскательным,

проектным, производственным, а также другим работам.

- В то же время в проектируемой БД необходимо иметь возможность гибко расширять множество хранимых типов данных без какой-либо переделки ранее созданных сущностей. К примеру, БД должна иметь возможность расширять список типов работ по сохранению объектов культурного наследия, а также добавлять соответствующие подтипы.
- В БД должны храниться не только результаты исследований, но и информация общего характера, описывающая сами объекты культурного наследия.
- Данные могут представлять собой стандартный набор числовых и строковых значений, оригинальные документы, содержащие текстовую и графическую информацию, могут быть использованы такие типы данных, как XML, JSON, или бинарный тип данных.

Перейдем к следующему этапу проектирования: конструирование семантической модели предметной области.

Вместе со специалистом по сохранению объектов культурного наследия, был получен вывод, что предметную область можно рассматривать как набор взаимодействующих между собой объектов. Объект здесь понимается как предмет или явление, обладающие точно описываемым поведением. К примеру, объект «реставратор» взаимодействует с объектом «памятник» посредством объекта «мероприятие».

Поэтому основным инструментом семантического моделирования был выбран язык объектного моделирования UML. Существует большое количество способов семантического моделирования баз данных. Язык семантического моделирования UML — расшифровывается как Unified Modeling Language - один из наиболее распространенных. UML успешно применяется для объектного моделирования бизнес — процессов, используется при проектировании информационных, аппаратных, программных, смешанных и других систем, позволяет графически отобразить

организационные структуры [7]. UML, в частности диаграммы классов, используется в том числе для проектирования баз данных. UML разрабатывается (последняя спецификация была выпущена в 2015 году) членами консорциума OMG (Object Management Group) и создается в соответствии с объектно-ориентированными стандартами.

Следует сказать, что ради удобства изложения и избегания дублирования информации мы забежим вперед и перейдем к этапу логического проектирования, где очертим множество рассматриваемых в данной ВКР моделей данных, а основные элементы сконструированной семантической модели данных рассмотрим позже в контексте выбранного множества моделей данных.

В этой работе будут проведен анализ проектных решений, основанных на двух моделях данных: реляционной и объектной.

Как было сказано ранее, появление реляционной модели и использование ее в первых реляционных СУБД ознаменовало собой начало революции в развитии баз данных; декларативный язык создания запросов SQL заменил собой императивные языки. Это произошло вследствие возникновения алгебры Кодда, являющейся формализацией отношений между кортежами. Важным следствием положений алгебры Кодда считается концепция нормальной формы БД. Эти и другие факторы обеспечили эффективность работы приложений, использующих РСУБД, что повлекло за собой лидерство РСУБД на рынке СУБД и по сей день.

К недостаткам реляционных БД относятся:

- отсутствие так называемого свойства object persistence, подразумевающего хранение в БД составного объекта как целого. Составные сущности в реляционной модели собираются (агрегируются) из нескольких непосредственно во время выполнения приложения, что в свою очередь влечет дополнительные

вычислительные затраты на операцию JOIN.

- Ограниченность набора типов данных и отношений между сущностями. Если потребуется отобразить сложный тип данных из предметной области, то его придется конструировать с помощью составных структур из простых типов. Отношения же формируются лишь посредством ключей.
- Издержки ассоциативного доступа к данным. В рамках реляционной модели не представляет возможным разрешить некоторое задачи выборки данных, в которых требуется рекурсивное решение. Краткий пример: в БД находится одна таблица, представляющая отношение «предок — потомок». Эта таблица содержит три столбца: id, ancestor_id, descendant_id. Теперь, если мы потребуем вывести потомков всех поколений определенного (для конкретного id) человека, то мы потерпим неудачу в силу того, что неизвестно количество применений операции JOIN с самой таблицей.

В свою очередь, объектно-ориентированная модель данных позволяет разрешить перечисленные проблемы, к тому же она согласуется с выбранным объектно-ориентированным языком семантического моделирования. Рассмотрим основные понятия и положения объектно-ориентированного подхода к проектированию БД.

**Класс** - абстрактный тип данных, определяющий структуру и поведение (данные и код, оперирующий ими), которые будут совместно использоваться набором объектов [13]. Под **объектом** будем понимать конкретный экземпляр класса. Объект обладает состоянием, поведением и индивидуальностью [8].

**Инкапсуляция** - механизм, связывающий метод и данные, которыми он манипулирует, защищая оба эти компонента от внешнего вмешательства и злоупотреблений [13].

Классы могут наследовать свойства (данные) и методы (код, оперирующий данными) других классов.

**Наследование —** процесс, в результате которого один объект получает свойства другого [13]. Наследование может быть одиночным, (каждый класс наследует только от одного класса) либо множественным (каждый класс наследует только от любого количества классов).

**Полиморфизм —** принцип ООП, позволяющий использовать один и тот же интерфейс для общего класса действий.

Теперь перейдем к рассмотрению конкретных случаев отображения предметной области в семантической модели, параллельно рассматривая преобразованные с учетом модели данных части проекта БД. Далее будут рассмотрены три наиболее важных случая, возникших при проектировании.

### 1.2.1 Агрегатная ассоциация

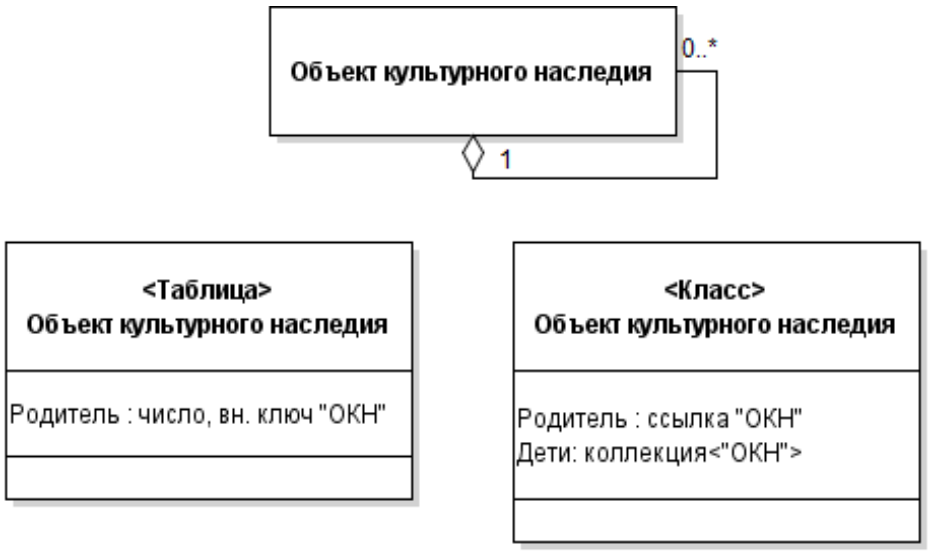



Рисунок 1. Агрегатная ассоциация.

Диаграмма, изображенная на рис. 1 состоит из трех частей. В верхней части расположен UML класс под названием «Объект культурного наследия» (ОКН). Этот класс состоит в отношении «агрегатная ассоциация» с самим собой и отражает требование предметной области, заключающееся в том, что любой объект культурного наследия может состоять из произвольного числа

частей, тоже являющимися объектами культурного наследия, причем если ОКН — родитель уничтожится, то это не приведет к уничтожению всех его частей. В левом нижнем углу диаграммы находится класс UML представляющий преобразованный с учетом реляционной модели вышестоящий класс. Соответственно, в правом нижнем углу находится проектное решение, построенное с учетом объектно-ориентированной модели данных.

Существует несколько подходов к организации иерархии данных [9-10]. Первый из них носит название список смежности. В этом варианте предполагается, что каждая запись «знает» своего непосредственного родителя. Таблица 1 содержит записи о комментариях, авторах этих комментариев и данные о том, какой комментарий для какого является ответом.

Таблица 1. Список смежности.

| ид_комментария | ид_родителя | автор | комментарий |
|---|---|---|---|
| 1 | NULL | Сергей | Сегодня отличная погода. |
| 2 | 1 | Евгений | Да. Стоит прогуляться. |
| 3 | 2 | Ольга | Выходим в 17:00. |
| 4 | 1 | Игорь | А мне по душе дожди. |

Как было сказано ранее необходимость знания количества уровней в иерархии — существенное ограничение, к тому же чем больше уровень вложенности тем больше операций JOIN - тем хуже производительность. Это основной недостаток данного подхода; с другой стороны не требуют сложных манипуляций и вычислительных ресурсов операции вставки, обновления и удаления, а также вывода типа «вывести наследников по родителю».

Второй способ — вложенное множество. Согласно этому подходу в

каждый записи представлены  ее потомки с помощью двух номеров:

- левый номер каждой записи меньше чем все номера потомков;
- правый номер каждой записи больше чем все номера потомков;
- номер каждой записи находятся между номерами предков записи

Таблица 2. Вложенное множество.

| ид_комментария | л_номер | п_номер | автор | комментарий |
|---|---|---|---|---|
| 1 | 1 | 14 | Сергей | ... |
| 2 | 2 | 5 | Евгений | ... |
| 3 | 3 | 4 | Ольга | ... |
| 4 | 6 | 13 | Игорь | ... |

Здесь первый элемент получил значения ключей 1 и 14 — числа, обозначающие диапазон ключей записей. В этом пример запись с ид_комментария  равным 2 «вложена» в запись с индексом 1, запись с ид_комментария  равным 3 «вложена» в запись с индексом 2 и т.д. При внесении изменений во вложенное множестве, вставке,  удалении каких — либо записей возникает необходимость пересчета ключей части дерева, находящейся справа и вверху от обновляемого узла; с другой стороны, с помощью этого подхода достаточно просто  организовать запрос на чтение отдельных веток:

«SELECT * FROM Комментарии A JOIN Комментарии B ON A.л_номер BETWEEN B.л_номер AND B.п_номер WHERE A.ид_комментария = 1»

Третий способ — нумерация пути — предполагает хранение полного пути от родителя к выбранному узлу  в каждой записи.  Путь формируется с помощью конкатенации номеров ключей родителей через разделитель (точку, или другой символ). При использовании этого подхода появляются накладные расходы на парсинг строки с путем (такая функция может и не

представляться в СУБД). К тому же, этот вариант нарушает важное для реляционной модели свойство ссылочной целостности (в любой записи таблицы не должно быть внешних ключей, указывающих на несуществующие записи).

Таблица 3. Нумерация пути.

| ид_комментария | путь | автор | комментарий |
|---|---|---|---|
| 1 | 1 / | Сергей | ... |
| 2 | 1 / 2 | Евгений | ... |
| 3 | 1 / 2 / 3 | Ольга | ... |
| 4 | 1 / 4 | Игорь | ... |

Несмотря на то, что силами только SQL невозможно организовать рекурсивный запрос, многие СУБД позволяют это сделать, поэтому был выбран вариант список смежности, ориентированный на обычные операции создания, чтения, обновления и удаления записей с редкими сложными иерархическими запросами.

Объектно — ориентированный вариант выглядит на равных с реляционной моделью. Тип данных «коллекция», указанный на рис. 1 — «это абстрактная структура данных,скрывающая способ хранения и связи объектов некоторого типа, а так же методы работы с этими объектами и коллекцией в целом» [13]. Существует также более общее понятие «карта» (map), отличающееся от коллекции тем, что ключами в карте могут быть любые объекты. Любой объект будет иметь ссылку и на родителя, и на всех детей, тогда с помощью рекурсивных запросов можно получить любую требуемую выборку.

### 1.2.1 Связь многие-ко-многим

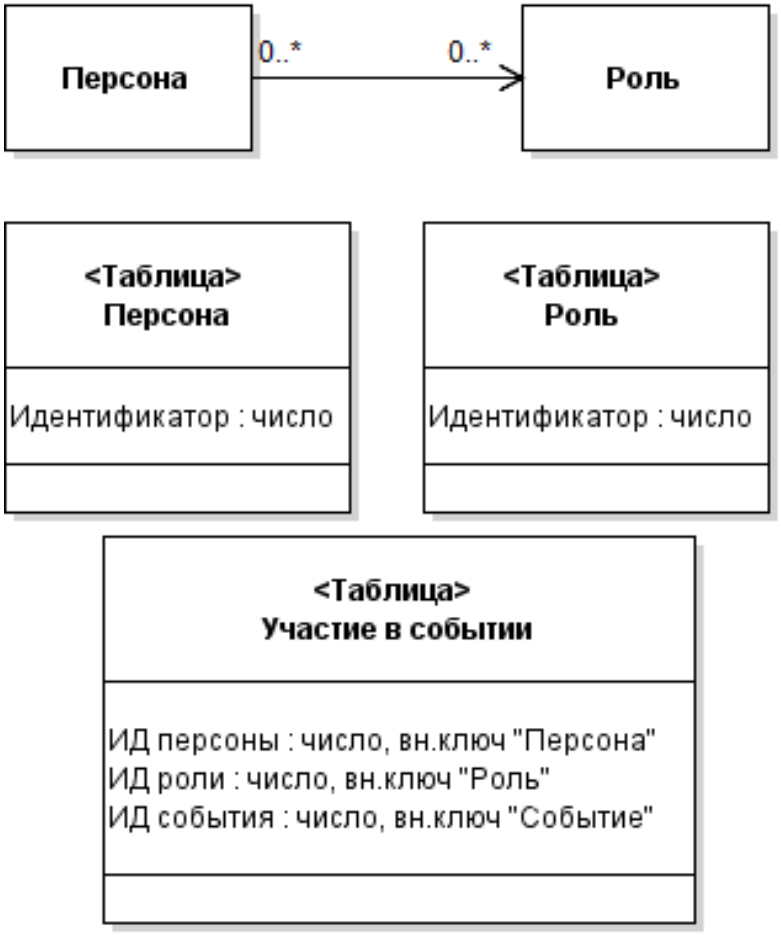



Рисунок 2. Связь «многие ко многим».

На вышестоящей диаграмме изображено воплощение связи «многие ко многим» в реляционной модели данных. Предполагается, что каждая персона может иметь любое (в том числе и нулевое) количество ролей в определенном событии, относящемуся к предметной области. При этом, с другой стороны, одна и та же роль может быть у нескольких людей, участвовавших в определенном событии.

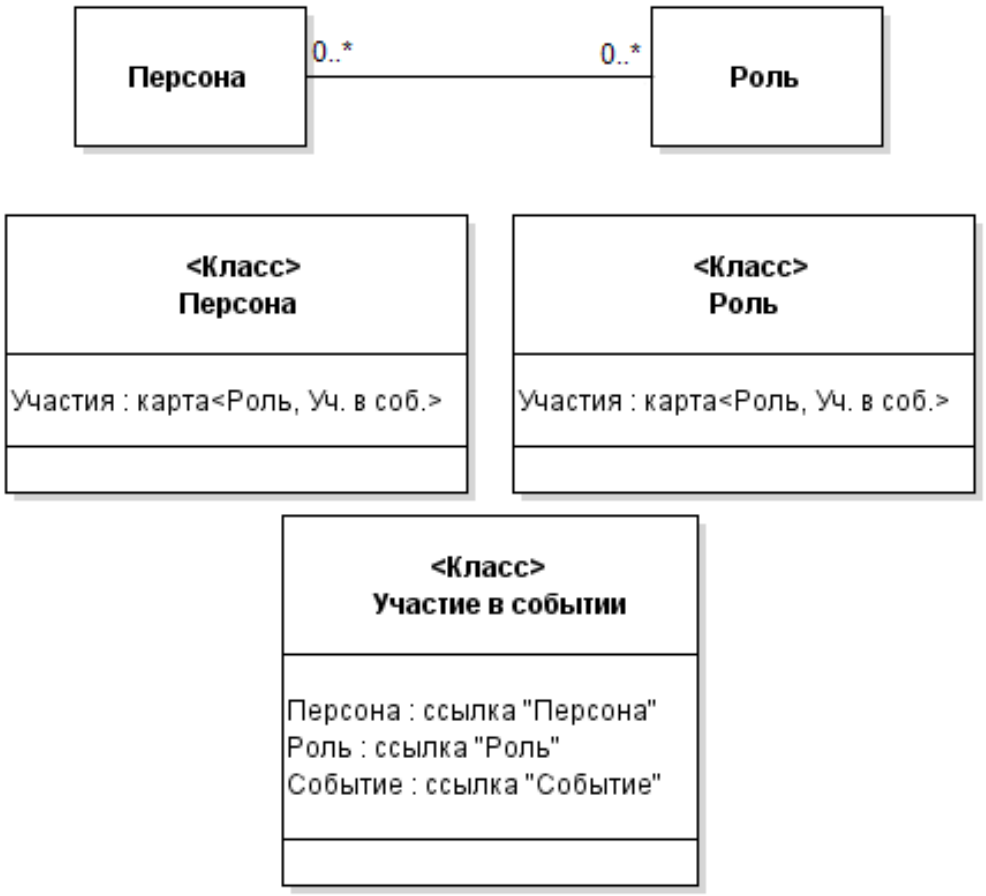



Рисунок 3. Связь «многие ко многим».

Из предложенных вариантов видно, что с практической точки зрения решения похожи. В обоих случаях необходимо создавать дополнительный вспомогательный элемент. Стоит отметить, что объектно-ориентированный вариант представляется более цельным, в том смысле что, к примеру при запросе на выборку всех ролей для указанной персоны, варианту справа потребуется всего лишь обратиться к полю «Участия», в то время как в реляционной версии придется использовать операцию JOIN и конструировать требуемый объект во время исполнения программы.

### 1.2.1 Наследование

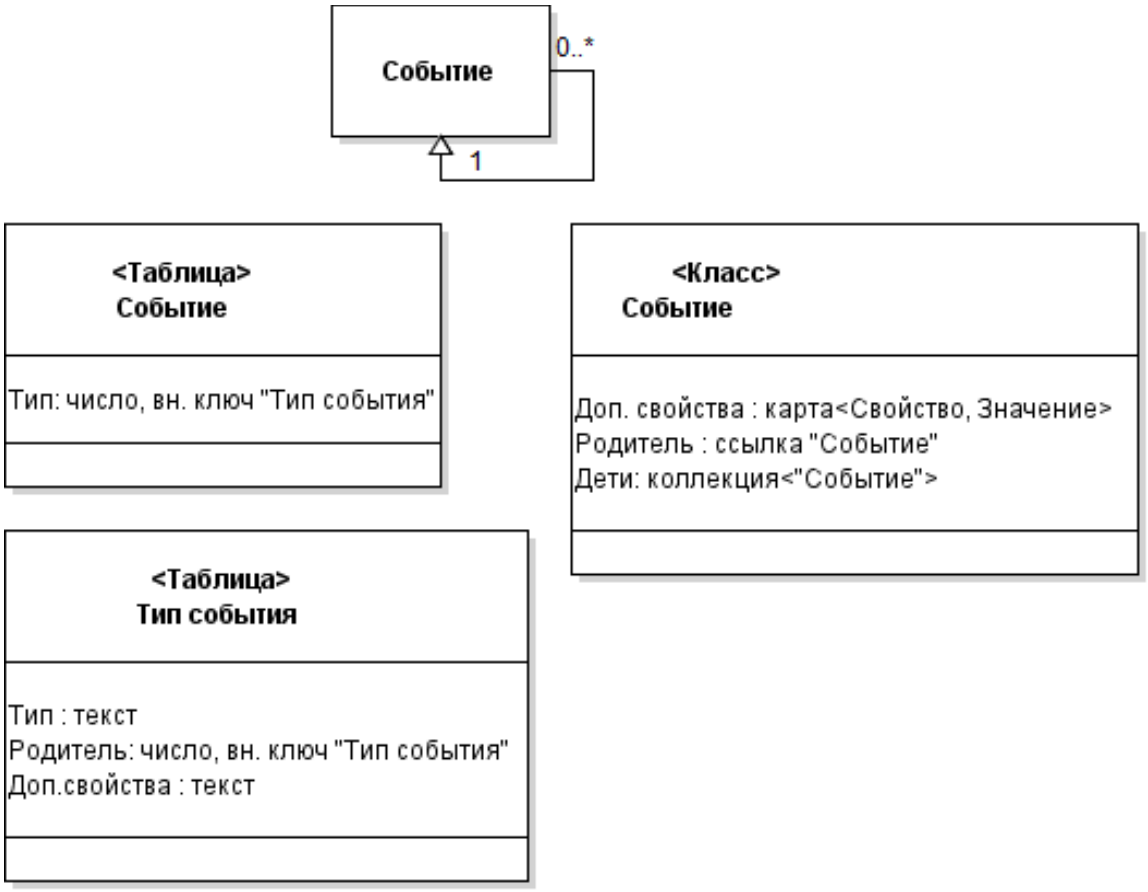



Рисунок 4. Наследование.

На этой диаграмме отражено требование предметной области, заключающееся в необходимости предоставлении произвольного количества подтипов абстрактного понятия «Событие» с возможностью дальнейшей каталогизации. Класс UML “Событие» находится с самим собой в отношении «Наследование», причем указано, что должен быть только один класс — родитель и произвольное число детей.

Рассмотрим различные подходы к организации наследования.

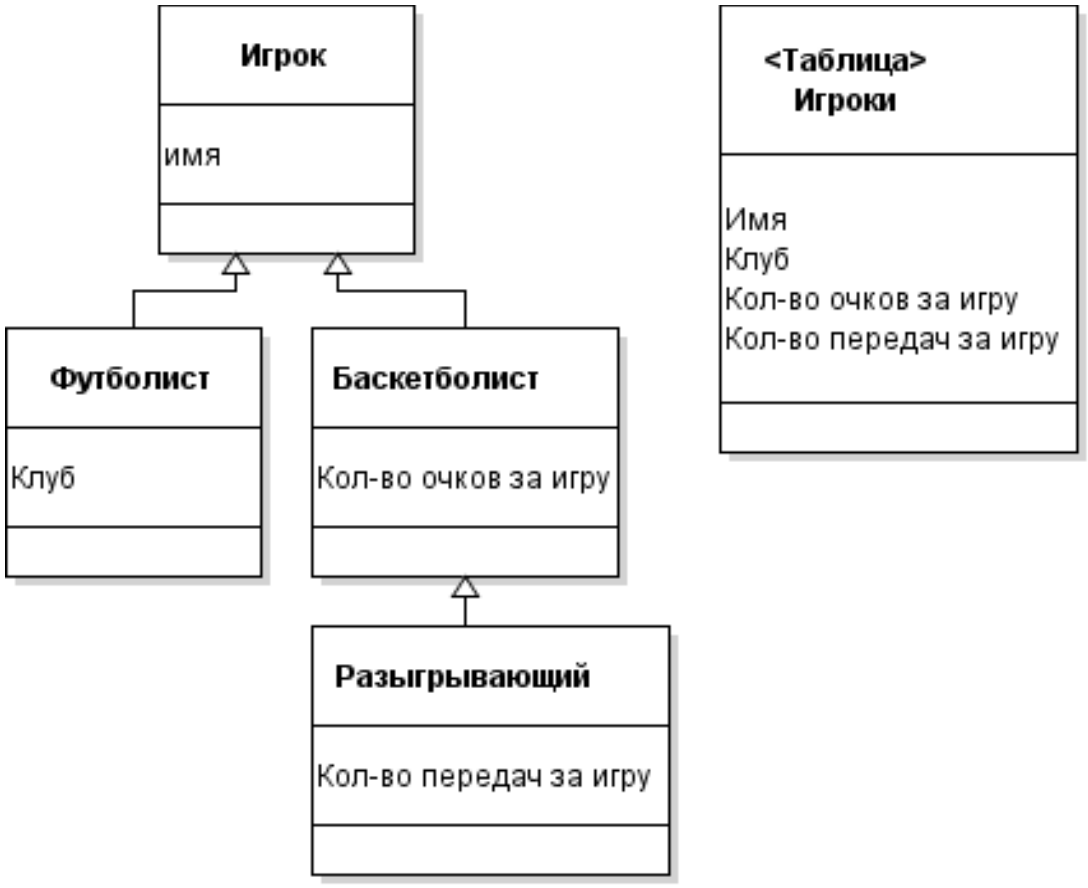



Рисунок 5.

Реляционные базы данных не поддерживают наследование, поэтому приходится искать некоторые представления структуры наследования в реляционных таблицах. В схеме изображенной на рис. 5 есть одна таблица, которая содержит все данные из всех классов в иерархии наследования.

Преимущества этого подхода заключаются в том, что есть только одна таблица и не нужно осуществлять операцию JOIN для получения данных. Кроме того нет необходимости менять таблицы в случае изменения столбцов.

Среди недостатков можно назвать проблему хранению нулевых значений: использование определенных столбцов только некоторыми подклассами приводит к лишней трате места в БД. Ответственность за эту проблему несет в том числе и СУБД, как она хранит пустые столбцы. Также, таблица может оказаться слишком большой, с большим количеством индексов и частыми блокировками, что отразится на производительности.

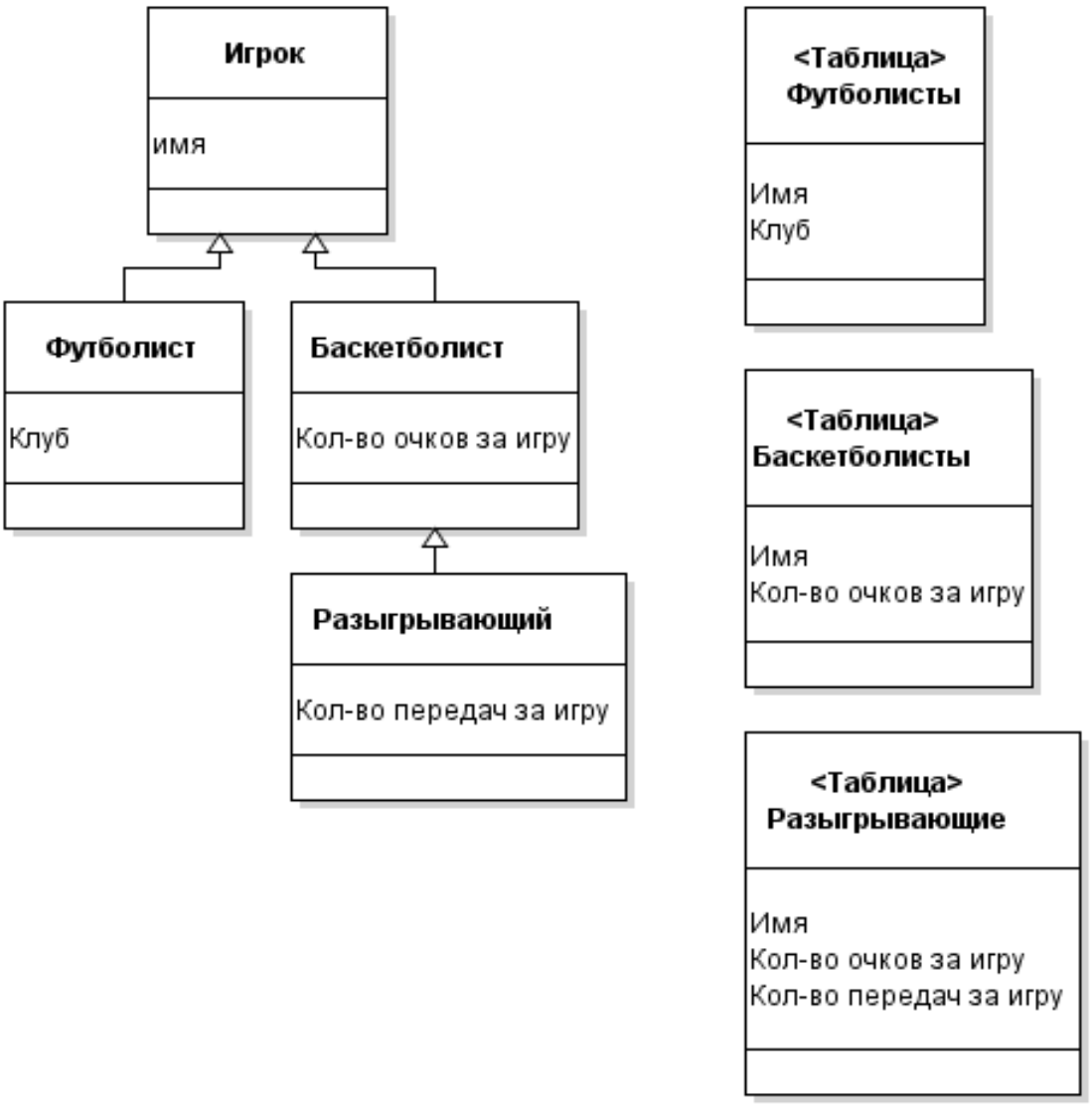



Рисунок 6.

В этом варианте предлагается использовать по одной таблице для каждого класса-наследника. Каждая таблица содержит как унаследованные, так и новые столбцы.

Преимущества этого подхода состоят в том, что каждая таблица содержит минимум необходимых столбцов, нет необходимости использовать JOIN. Рассмотрим недостатки.

Если изменятся некоторые столбцы суперкласса (класс, от которого происходит наследование), то придется это учесть во всех наследниках.

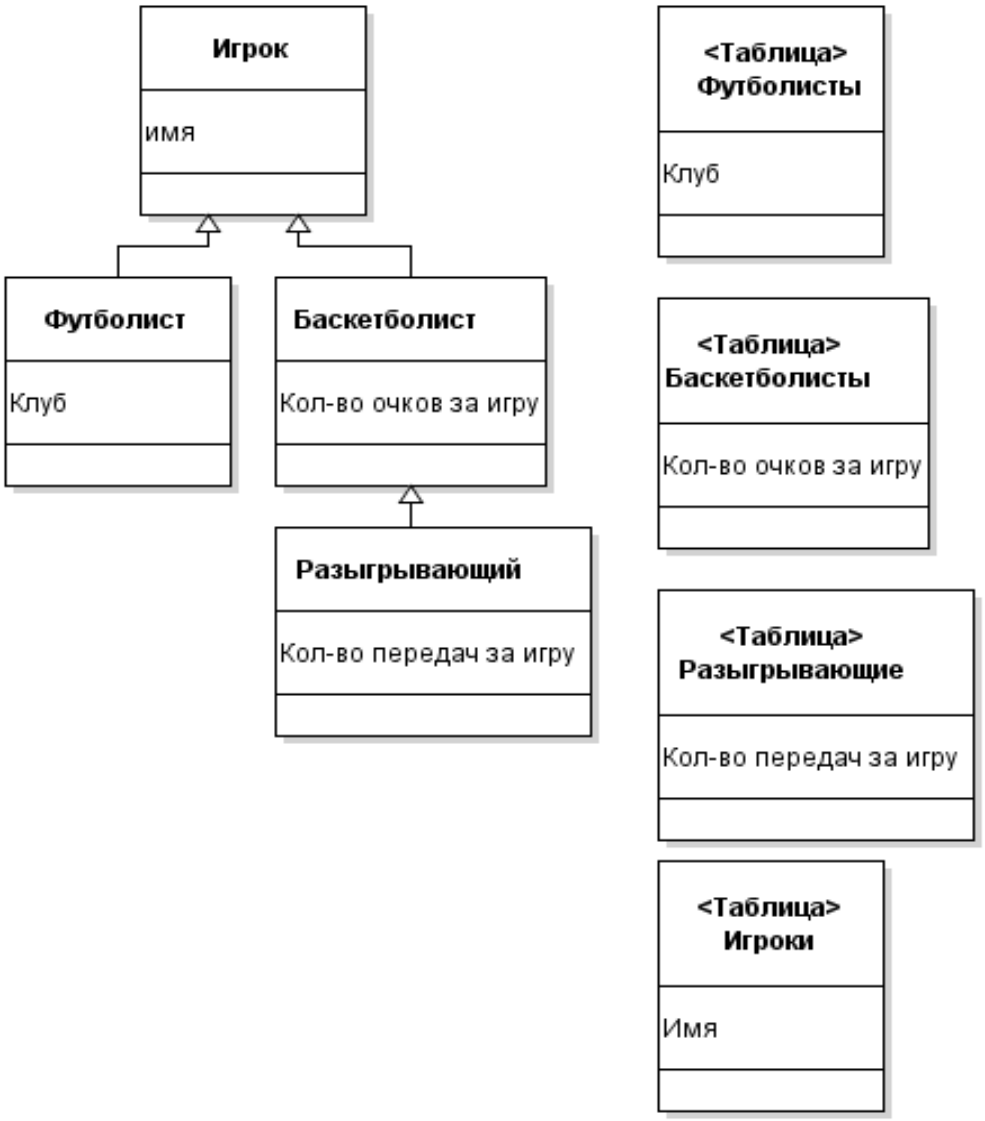



Рисунок 7.

Преимущества этой альтернативы состоят в том что, все столбцы значимы для каждой записи, поэтому будут отсутствовать нулевые значения.

К слабостям этого варианта следует отнести факт того, что требуется совершить столько операций JOIN, сколько раз искомая таблица наследовала. Кроме того, вследствие того, что к таблицам супертипа часто обращаются, соответственно, чаще выполняются блокировки, что сказывается на времени

отклика базы данных.

В проекте базы данных применена модификация последнего способа.

Как указано на рис. 4, используется таблица, задающая общий набор характеристик для таблицы «Событие». Во вспомогательной таблице «Типы события» указываются различные типы событий, их иерархическая зависимость, а также имя таблицы, в которой содержаться необходимые дополнительные столбцы для конкретного типа.

В случае объектно-ориентированного варианта дополнительные поля класс предлагается хранить в карте (map), необходимая иерархия поддерживается как и в случае с Объектом культурного наследия с помощью ссылок на родителя и детей.

**Общий вывод:** во всех трех рассмотренных случаях реляционная модель данных при указанных комментариях не уступает с точки зрения практической реализации требований, предъявленных к базе данных, и проектный вариант базы данных, основанный на реляционной модели данных, будет использован для программной реализации.

Реляционные СУБД пользуются успехом на рынке СУБД и это нельзя не принимать во внимание. Существует, в том числе, и СУБД доступные для свободного использования, обладающие существенным функционалом. Следует также отметить что большинство объектных СУБД, представленных на рынке обладают проприетарной лицензией и не могут быть использованы в полном объеме бесплатно.

# Глава 2. Программная реализация базы данных

В данной работе используется СУБД MySQL - реляционная система управления базами данных с открытым кодом. Программное обеспечение с открытым кодом означает, что любое лицо может использовать и модифицировать этот продукт.

“MySQL является системой клиент-сервер, которая содержит многопоточный SQL-сервер, обеспечивающий поддержку различных вычислительных машин баз данных, а также несколько различных клиентских программ и библиотек, средства администрирования и широкий спектр программных интерфейсов (API).” - лаконичное описание с сайта проивзодителя [12]. Стоит также отметить, что MySQL поддерживает систему безопасности, основанную на привилегиях и паролях, в том числе с возможностью верификации с удаленного компьютера.

В MySQL представлено большое количество типов данных: целочисленные, символьные, типы данных времени даты, возможность задавать значения по умолчанию.

MySQL поддерживает хранимые процедуры — последовательность операций, хранящаяся как единое целое в базе данных на сервере [12].

В разработанной базе данных таблицы имеют тип InnoDB, предоставляющий следующие возможности:

- поддержка транзакций. Транзакция — это блок операторов SQL , который в случае ошибки в одном запросе, возвращается к предыдущему состоянию (Rollback), и только в случае выполнения всех запросов подтверждается (Commit).
- Блокировка уровня строки. Если процессу нужно обновить строку в таблице, то он блокирует только эту строку, позволяя другим обновлять другие строки параллельно.
- Поддержка внешних ключей.

Рассмотрим некоторые таблицы из реализованной базы данных.

Одной из ключевых таблиц является таблица «Объект культурного наследия». Эта таблица хранит записи о сооружениях, представляющих ценность с научной, археологической, художественной или культурной ценностью. Таблица содержит достаточное количество столбцов, чтобы, к примеру, сформировать [16] документ «Паспорт памятник истории и культуры».

Таблица 4. «Объект культурного наследия».

| Имя | Тип | Описание |
|---|---|---|
| ID | INT | Идентификатор, ключ |
| NAME | VARCHAR | Название |
| BUILD_DATE | DATE | Дата постройки |
| TYPE | ENUM | Тип (памятник архитектуры,...) |
| CATEGORY | ENUM | Категория использования |
| CONDITION | ENUM | Техническое состояние |
| ADDRESS | ID | Вн. ключ, «Адрес» |
| DESCRIPTION | TEXT | Описание |
| HISTORY | TEXT | Исторические сведения |
| IMPORTANCE | TEXT | Культурное значение |
| GROUP_ID | INT | Вн. ключ, «Группа ОКН» |
| PARENT_ID | INT | Вн. ключ, «ОКН» |

В следующей таблице содержатся данные обо всех лицах, каким-либо образом причастным к памятникам — реставраторы, архитекторы и т.д..

Таблица 5. «Персона».

| Имя | Тип | Описание |
|---|---|---|
| ID | INT | Идентификатор, ключ |
| NAME | VARCHAR | Имя |
| SECONDNAME | VARCHAR | Второе имя |
| SURNAME | VARCHAR | Фамилия |
| FATHERS_NAME | VARCHAR | Отчество |
| BIRTH_DATE | DATE | Дата рождения |
| DEATH_DATE | DATE | Дата смерти |

Как было указано в параграфе 1.2. в базе данных должны быть отражен тот же характер каталогизации, что и в законодательстве РФ.

Закон разделяет мероприятия, проводимые в отношении памятников на изыскательные, научно-исследовательские, производственные и проектные работы. В базе данных эти мероприятия являются типом абстрактного понятия «События».

Таблица 6. «Событие».

| Имя | Тип | Описание |
|---|---|---|
| ID | INT | Идентификатор, ключ |
| NAME | VARCHAR | Описание |
| START_DATE | DATE | Дата начала события |
| END_DATE | DATE | Дата окончания события |
| TYPE_ID | INT | Вн. ключ «Тип события» |

Таблица 7. «Тип события».

| ID | NAME | PARENT_ID | ADDITIONAL |
|---|---|---|---|
| 3 | Научно-исследовательские работы | 1 | ... |
| 4 | Изыскательные работы | 1 | ... |
| 5 | Проектные работы | 1 | ... |
| 6 | Производственные работы | 1 | ... |
| 7 | Спектральный анализ | 3 | ... |
| 8 | Атомный спектральный анализ | 7 | ... |

Таблица «Файл» содержит описание всех файлов, которые относятся к базе данных. От этой таблицы также «наследуются» такие таблицы как «Фотография» и «Документ». Сами файлы хранятся в файловой системе. Это позволяет избежать лишней нагрузки на БД.

Таблица 8. «Файл».

| Имя | Тип | Описание |
|---|---|---|
| ID | INT | Идентификатор, ключ |
| NAME | VARCHAR | Описание |
| PATH | VARCHAR | Относительный путь к файлу |
| EVENT_ID | INT | Вн. ключ «Событие» |

# Глава 3. Программная реализация приложения

Для доступа к базе данных было разработано серверное приложение - сервлет. Сервлет (Servlet) - это программа, написанная на языке программирования Java, которая выполняется на серверной стороне веб — подключения, как указано на рис. 8.

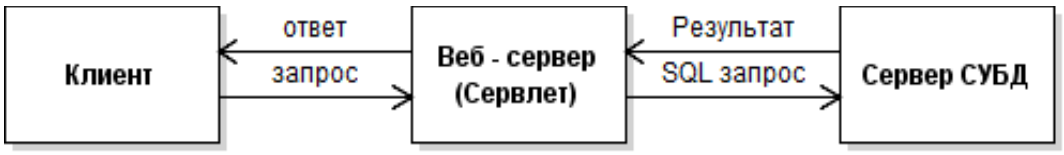



Рисунок 8. Роль приложения

Приведем описание работы сервлета: при получении запроса от клиента (к примеру, от веб-браузера) веб-сервер с помощью специального конфигурационного файла может определить, какой сервлет необходимо выполнить. После этого веб-сервер запускает JAVA-машину (программа, исполняющая байт — код, созданный из текстового файла), которая в свою очередь выполняет сервлет. Далее сервлет отправляет S Q L запрос на выполнение в СУБД и ожидает результата запроса. Получив результат, сервлет обрабатывает его и дает ответ клиенту [14].

Сервлеты обладают рядом преимуществ по сравнению с традиционным интерфейсом CGI, предлагающего к использованию скрипты (исполняемые при определенных сценариях файлы):

- в случае с сервлетами, Java-машина обрабатывает каждый запрос в отдельном потоке, а не в процессе операционной системы. Для N запросов будет запущено N потоков, однако в память будет загружена только одна копия класса сервлета. Такой подход позволяет уменьшить издержки на создание новых объектов и на потребление памяти.
- Сервлеты обладают уже реализованной инфраструктурой для парсинга и обработки HTML, обработки HTTP запросов, отслеживаний сессий и т.п.

- Код сервлета (как и любой Java-программы) транслируется в байт-код и обрабатывается Java-машиной, отправляющей инструкции устройству как интерпретатор, что делает байт-код независимым от операционной системы.
- Технология сервлетов поддерживаются серверами крупнейших компаний — поставщиков веб технологий: Oracle, Apache,IBM и др.

Для разрабатываемой базы данных использовалась контейнер сервлетов Apache Tomcat – свободное программное обеспечение, реализующее спецификации Java для сервлетов [15].

Разработанный сервлет, удовлетворяет требованиям, сформулированным в постановке задач. Программный код сервлета представлен в приложении к работе. Приведем несколько иллюстраций, демонстрирующих результаты работы сервлета.

На рис. 9 изображено окно программы Postman – приложение-клиент, с помощью которого будут посылаться запросы сервлету. На рисунке указан URL адрес, по которому идет обращение к сервлету, а также набор параметров и значений. Сверху вниз указаны имя пользователя базы данных, пароль, тип совершаемой операции, таблица, в отношении которой применяется операция, а также столбцы.

В нижней части рисунка 10. изображен результат чтения записи с идентификатором равным 1. Выведены названия столбцов и их значения в указанной записи. Некоторые значения равны NULL вследствие того, что при вставке им не были присвоены значения, поэтому выставились значения по-умолчанию.

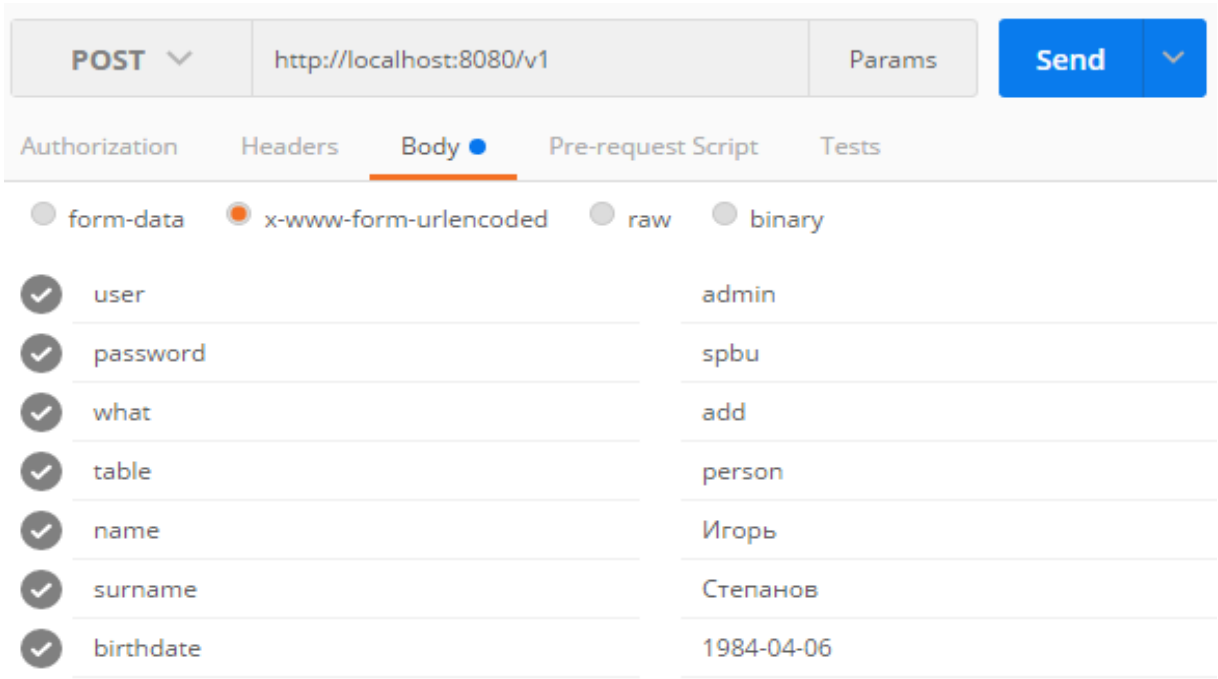



Рисунок 9. Вставка записи.

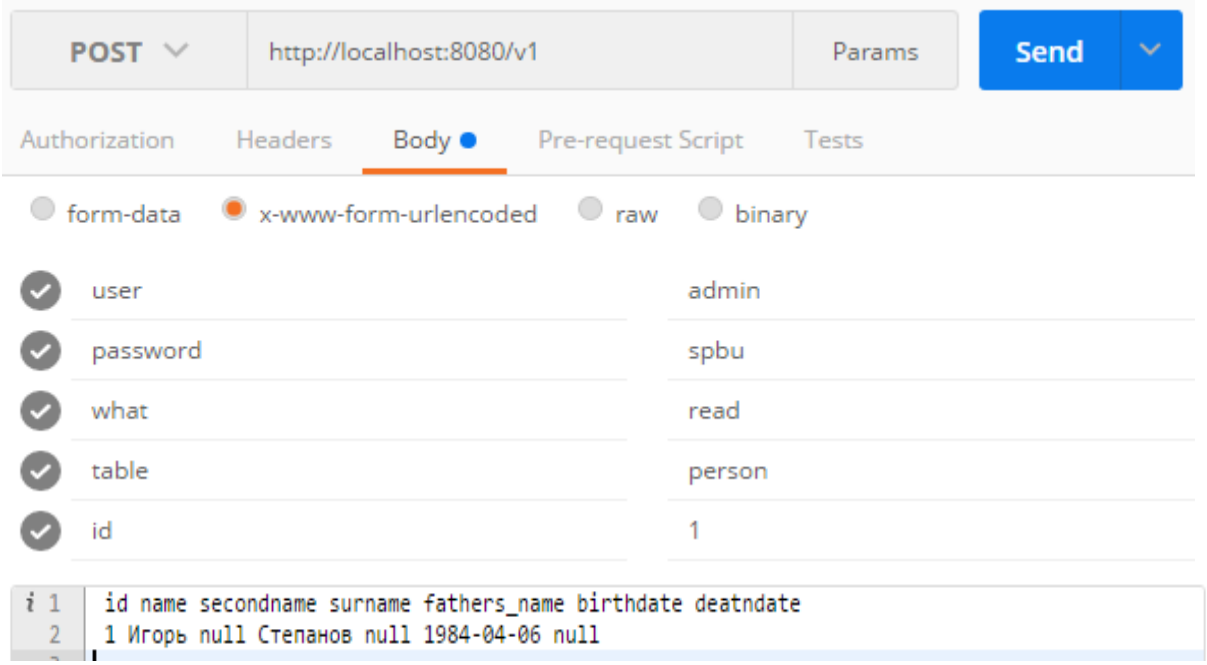



Рисунок 10. Чтение записи.

Поддерживаются функции вставки в несколько таблиц в рамках одной транзакции, а также чтения из нескольких таблиц.

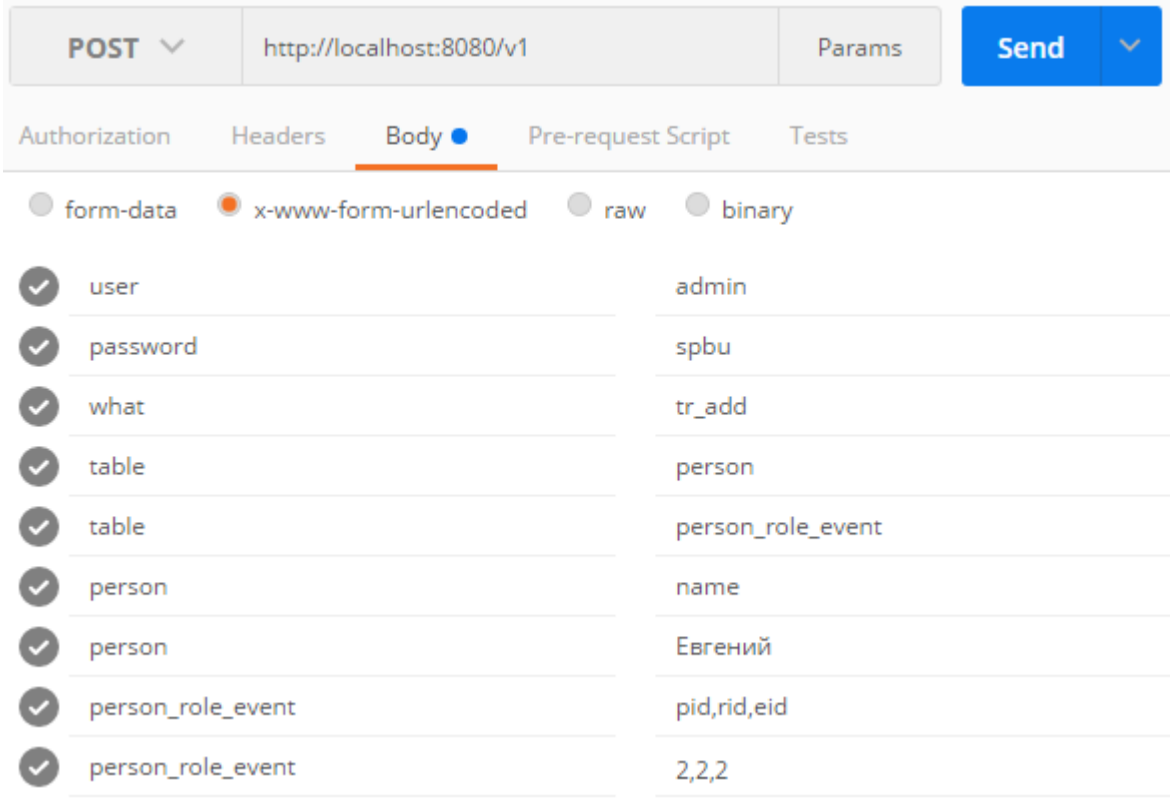
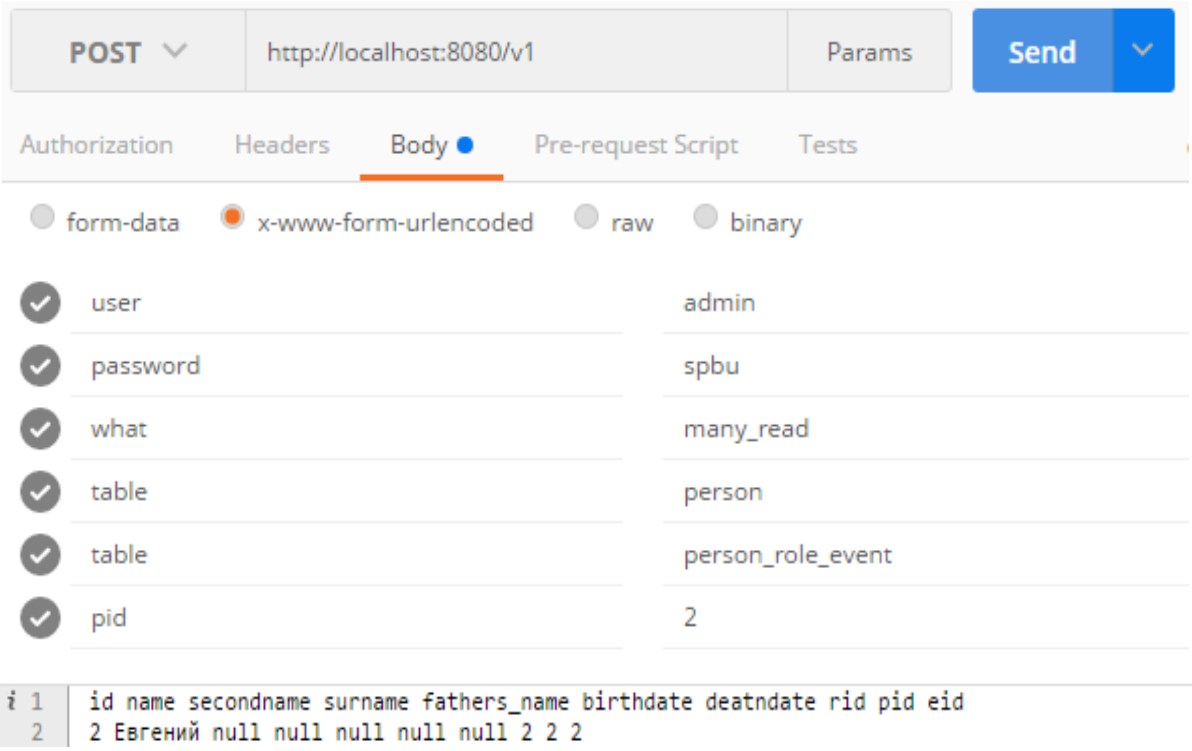



Рисунок 11. Вставка в несколько таблиц




```
id name secondname surname fathers_name birthdate deatndate rid pid eid
2 Евгений null null null null null 2 2 2
```

Рисунок 12. Чтение из нескольких таблиц

Приложение также поддерживает вызов хранимой процедуры, исполняемой на СУБД. К примеру, выведем идентификаторы всех подтипов типа «Научно-исследовательские работы» из таблицы 7 «Тип события».

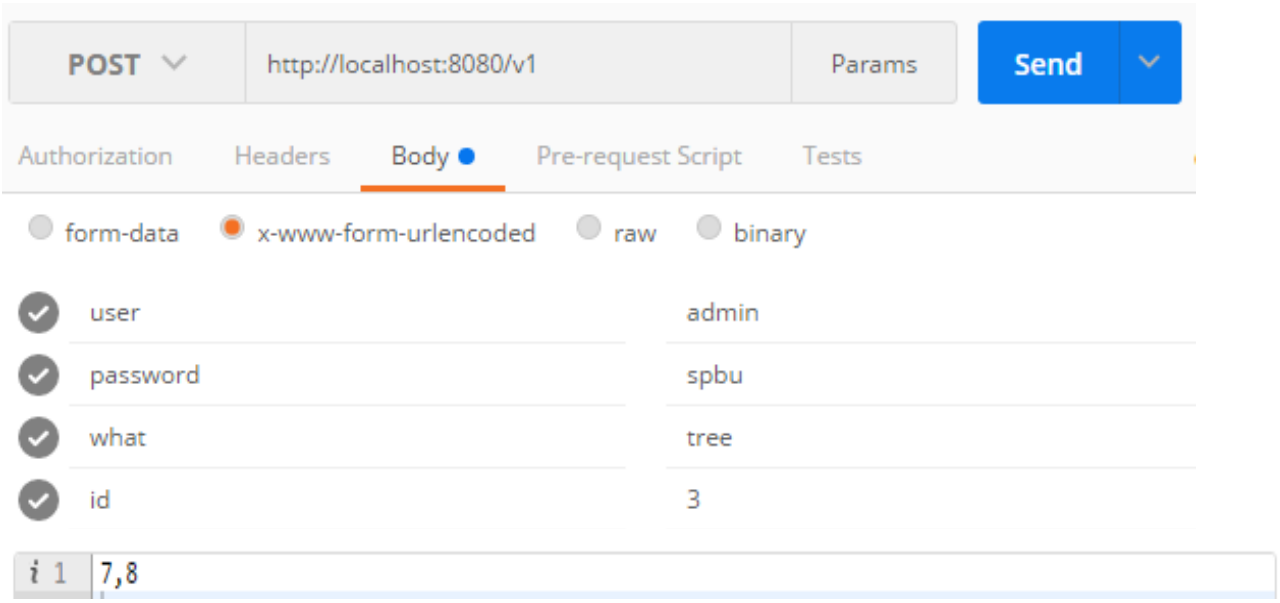



Рисунок 13. Хранимая процедура.

Реализована возможность отправки файлов на сервер.

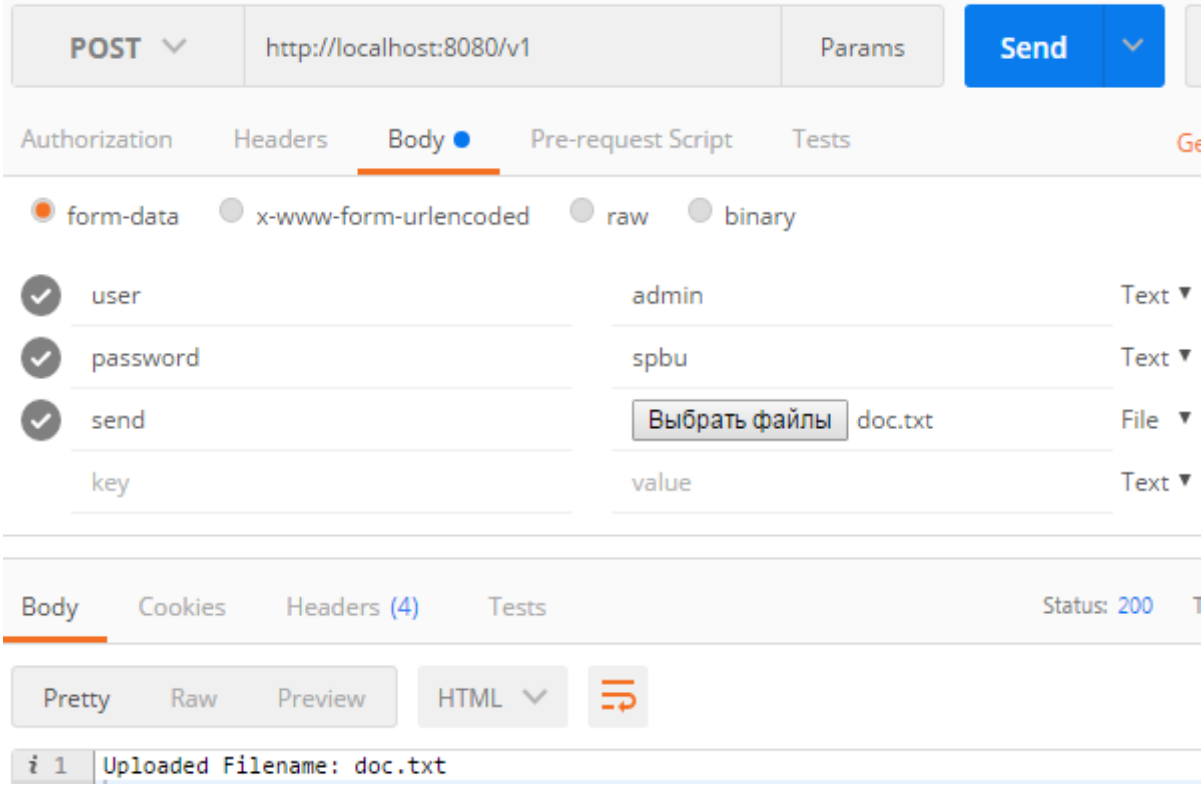



Рисунок 14. Отправка файла.

# Выводы

На основе проектного решения создана база данных, готовая к внедрению в сферу деятельности специалистов по сохранению объектов культурного наследия. Также разработано приложение, отвечающее сформированным требованиям, для ведения работы с базой данных.

# Заключение

К разработанной базе данных в дальнейшем могут быть добавлены новые требования, которые воплотятся в виде таблиц, связей между ними, ограничений, дополнительных столбцов и других проявлений. Приложение, несомненно, тоже должно учитывать эти изменения и отражать их в своем функционале. Дальнейшая работа по базе данных может включать в себя рассмотрение других моделей данных и, соответственно, других систем управления базами данных. Кроме того, с течением времени возможно возникнет необходимость реализовать дополнительные функции, представленные в приложении, которые будут удовлетворять отдельные, специфические запросы специалистов. Требует реализации также и клиентское приложение, предназначенное для конечного пользователя.

# Список литературы


1. ISO/IEC 2382-1:1993, Information technology — Vocabulary — Part 1: Fundamental terms: a reinterpretable representation of information in a formalized manner suitable for communication, interpretation, or communication, or processing
2. Дейт К. Дж. Реляционная модель выдержит испытание временем (пер. с Date, C.J. The relational model will stand the test of time // Intelligent Enterprise, June 1, 1999, Volume 2, Number 8)
3. Дейт К.Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильяме", 2005. — 1328с
4. Карпова И.П. Базы данных. Учебное пособие. – Московский государственный институт электроники и математики (Технический университет). – М., 2009.
5. Кузнецов С.Д. Основы баз данных: учебное пособие. М.: Интернет – Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 484с.
6. History of ODMS. http://www.odbms.org/introduction-to-odbms/history/
7. OMG Formal versions of UML. http://www.omg.org/spec/UML/
8. Object-Oriented Design with Applications.3rd ed.,Grady Booch, 720 p., 2007 ISBN-10: 0-201-89551-X
9. B. Karwin. SQL Antipatterns: Avoiding the Pitfalls of Database Programming. Pragmatic Programmers. 1 Edition, 2010. - 328p.
10. M. Fowler. Patterns of Enterprise Application Architecture.Addison-Wesley Professional .1 Edition, 2002 – 560p.
11. Кузнецов М.В.MySQL на примерах / М. В. Кузнецов, И.В. Симдянов. - Спб.: БХВ-Петербург, 2007. - 592 с.
12. MySQL Documentation https://dev.mysql.com/doc/

13. Java 8. Полное руководство, 9-е изд.: Пер.с англ. -М.:ООО «И.Д.Вильямс», 2015. - 1376с.ил: - Парал. Тит.англ

14. J.Murach, M.Urban. Murach's Java Servlets and JSP.Mike Murach & Associates; 3rd edition, 2014 – 758p.

15. Java Platform, Standard Edition 7 API Specification https://docs.oracle.com/javase/7/docs/api/

16. 13. Федеральный закон от 25.06.2002 N 73-ФЗ (ред. от 09.03.2016) "Об объектах культурного наследия (памятниках истории и культуры) народов Российской Федерации"

# Приложение

```
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.commons.io.FileUtils;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;

import java.sql.*;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.*;



public class v1 extends HttpServlet {
   private static final String url = "jdbc:mysql://localhost:3306/facilities"+"?
useUnicode=true&characterEncoding=UTF-8";
   private enum What{
      add,update,delete,read, tr_add,many_read,foreign_read,tree;
   }
   private static final int MAX_MEM_SIZE = 4 * 1024;
   private static final int MAX_FILE_SIZE = 500 * 1024;

   @Override
```

```
    public void init() throws ServletException {
        super.init();
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setCharacterEncoding("UTF-8");
        request.setCharacterEncoding("UTF-8");

        // Establish connection
        Connection connection;
        PreparedStatement stmt = null;
        if (!ServletFileUpload.isMultipartContent(request)) {
            // Get response writer
            PrintWriter pw = null;
            try {
                pw = response.getWriter();
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                connection = DriverManager.getConnection(url,
request.getParameter("user"),request.getParameter("password"));
```

```
        } catch (SQLException sqlEx) {
            sqlEx.printStackTrace();
            response.getWriter().println("Incorrect user / password.\n Possible
connection problems");
            return;
        }



        // Parse query
        HashMap<String, String[]> params = new HashMap<String,
String[]>(request.getParameterMap());
        What what = What.valueOf(params.get("what")[0]);
        StringBuilder answer;
        try {
            switch (what) {
                case add:
                    stmt = insertInto(params.get("table")[0], params, connection);
                    stmt.executeUpdate();   System.out.println(stmt.toString());
                    break;
                case update:
                    stmt = modify(params.get("table")[0], params, connection);
                    stmt.executeUpdate();
                    break;
                case delete:
                    stmt = delete(params.get("table")[0], params, connection);
                    stmt.executeUpdate();
                    break;
                case read:
```

```
                answer = read(params.get("table")[0], params, connection);
                pw.println(answer);
                break;
            case tr_add:
                transactionalInsert(params,connection);
                break;
            case many_read:
                answer = manyToManyRead(params, connection);
                pw.println(answer);
                break;
            case foreign_read:
                answer = foreignKeyRead(params,connection);
                pw.println(answer);
                break;
            case tree:
                answer = getTree(params,connection);
                pw.println(answer);

        }
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (NullPointerException e) {
        e.printStackTrace();
    }
} else {
    connection = manageFileIO(request, response);
}
```

```
        // Close everything
        try {
            if (stmt != null) stmt.close();
            if (connection != null) connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private PreparedStatement modify(String tableName, HashMap<String,
String[]> params, Connection con) {
        params.remove("user");
        params.remove("password");
        params.remove("what");
        params.remove("table");
        Iterator<String[]> values = params.values().iterator();
        Iterator<String> keys = params.keySet().iterator();
        int length = params.size();
        StringBuilder sql = new StringBuilder().append("UPDATE
").append(tableName).append(" SET ");
        StringBuilder sql_val = new StringBuilder().append(" WHERE id = ?");
        for (int i = 0; i < length - 1 /* id */; i++) {
            String key = keys.next();
            sql.append(key).append(" = ?,");

        }
        sql.deleteCharAt(sql.length() - 1);
        sql.append(sql_val);
```

```
        PreparedStatement stmt = null;
        try {
            stmt = con.prepareStatement(sql.toString());
        } catch (SQLException e) {
            e.printStackTrace();
        }

        for (int i = 1; i < length + 1; i++) {
            String value = values.next()[0];
            try {
                stmt.setString(i, value);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        return stmt;
    }

    private PreparedStatement insertInto(String tableName, HashMap<String,
String[]> params, Connection con)  {
        params.remove("user");
        params.remove("password");
        params.remove("what");
        params.remove("table");
        Iterator<String[]> values = params.values().iterator();
        Iterator<String> keys = params.keySet().iterator();
        int length = params.size();
```

```
        StringBuilder sql = new StringBuilder().append("INSERT INTO
").append(tableName).append("(");
        StringBuilder sql_val = new StringBuilder().append("VALUES(");
        for (int i = 0; i < length; i++) {
            String key = keys.next();
            sql.append(key).append(',');
            sql_val.append("?,");
        }
        sql.deleteCharAt(sql.length() - 1);
        sql_val.deleteCharAt(sql_val.length() - 1);
        sql.append(")");
        sql_val.append(")");
        sql.append(sql_val);

        PreparedStatement stmt = null;
        try {
            stmt = con.prepareStatement(sql.toString());
        } catch (SQLException e) {
            e.printStackTrace();
        }

        for (int i = 1; i < length + 1; i++) {
            String value = values.next()[0];
            try {
                stmt.setString(i, value);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

```
        return stmt;

    }

    private PreparedStatement insertInto(String tableName, String[] keys, String[]
values, Connection con)  {

        int length = keys.length;
        StringBuilder sql = new StringBuilder().append("INSERT INTO
").append(tableName).append("(");
        StringBuilder sql_val = new StringBuilder().append(" VALUES(");
        for (String key : keys) {
            sql.append(key).append(',');
            sql_val.append("?,");
        }
        sql.deleteCharAt(sql.length() - 1);
        sql_val.deleteCharAt(sql_val.length() - 1);
        sql.append(")");
        sql_val.append(")");
        sql.append(sql_val);
        System.out.println("sql " + sql);

        PreparedStatement stmt = null;
        try {
            stmt = con.prepareStatement(sql.toString());
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        for (int i = 1; i < length + 1; i++) {
            String value = values[i-1];
            try {
                stmt.setString(i, value);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        System.out.println("sql " + stmt );

        return stmt;

    }

    private PreparedStatement delete(String tableName, HashMap<String, String[]>
params, Connection con)  {
        params.remove("user");
        params.remove("password");
        params.remove("what");
        params.remove("table");
        Iterator<String[]> values = params.values().iterator();
        int length = params.size();
        StringBuilder sql = new StringBuilder().append("DELETE FROM
").append(tableName).append(" WHERE id = ?");
        System.out.println(sql.toString());
        PreparedStatement stmt = null;
        try {
            stmt = con.prepareStatement(sql.toString());
```

```
            System.out.println("stm " + stmt);
        } catch (SQLException e) {
            e.printStackTrace();
        }

        for (int i = 1; i < length + 1; i++) {
            String value = values.next()[0];
            try {
                stmt.setString(i, value);
                System.out.println("stm " + stmt);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return stmt;
    }

    private StringBuilder read(String tableName, HashMap<String, String[]>
params, Connection con)  {
        params.remove("user");
        params.remove("password");
        params.remove("what");
        params.remove("table");
        Iterator<String[]> values = params.values().iterator();
        int length = params.size();
        StringBuilder sql = new StringBuilder().append("SELECT * FROM
").append(tableName).append(" WHERE id = ?");
        PreparedStatement stmt = null;
        try {
```

```
    stmt = con.prepareStatement(sql.toString());
} catch (SQLException e) {
    e.printStackTrace();
}

for (int i = 1; i < length + 1; i++) {
    String value = values.next()[0];
    try {
        stmt.setString(i, value);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
StringBuilder response = new StringBuilder();
StringBuilder help = new StringBuilder();
try {
    ResultSet set = stmt.executeQuery();
    set.beforeFirst();
    ResultSetMetaData data = set.getMetaData();
    int l = data.getColumnCount();
    while (set.next()) {
        for (int i = 1; i < l + 1; i++) {
            help.append(' ').append(data.getColumnName(i));
            response.append(' ').append(set.getString(i));
        }
    }
    set.close();
} catch (SQLException e) {
    e.printStackTrace();
```

```
        }

        return help.append("\n").append(response);
    }

    private void transactionalInsert(HashMap<String, String[]> params, Connection
con) {
        String[] tables = params.get("table");
        // Only save CRUD params
        params.remove("user");
        params.remove("password");
        params.remove("what");
        params.remove("table");
        try {
            con.setAutoCommit(false);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        for (String table : tables) {
            String[] keysValues = params.get(table);
            String[] keys = keysValues[0].split(",");
            String[] values = keysValues[1].split(",");
            PreparedStatement stmt = insertInto(table, keys, values, con);
            try {
                stmt.executeUpdate();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

```
        try {
            con.commit();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private StringBuilder manyToManyRead(HashMap<String, String[]> params,
Connection con) {
        String[] tables = params.get("table");
        // Only save CRUD params
        params.remove("user");
        params.remove("password");
        params.remove("what");
        params.remove("table");
        try {
            con.setAutoCommit(false);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        StringBuilder sql = new StringBuilder().append("SELECT * FROM
").append(tables[0]);
        StringBuilder sql_join = new StringBuilder().append("LEFT JOIN ");

        sql_join.append(tables[1]).append(" ON ")
                .append(tables[0]).append(".id =
").append(tables[1]).append(".").append(tables[0].charAt(0)).append("id")
                .append(" WHERE ").append(tables[0]).append(".id = ?");
        sql.append(' ').append(sql_join);
```

```
System.out.println(sql.toString());

PreparedStatement stmt = null;
Iterator<String[]> values = params.values().iterator();
try {
   stmt = con.prepareStatement(sql.toString());
   stmt.setString(1,values.next()[0]);
} catch (SQLException e) {
   e.printStackTrace();
}
StringBuilder response = new StringBuilder();
StringBuilder help = new StringBuilder();
try {
   ResultSet set = stmt.executeQuery();
   set.beforeFirst();
   ResultSetMetaData data = set.getMetaData();
   int l = data.getColumnCount();

   while (set.next()) {
      for (int i = 1; i < l + 1; i++) {
         help.append(' ').append(data.getColumnName(i));
         response.append(' ').append(set.getString(i));
      }
   }
   set.close();
} catch (SQLException e) {
   e.printStackTrace();
}
try {
```

```
            con.commit();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return help.append("\n").append(response);

    }

    private StringBuilder foreignKeyRead(HashMap<String, String[]> params,
Connection con) {
        String[] tables = params.get("table");
        // Only save CRUD params
        params.remove("user");
        params.remove("password");
        params.remove("what");
        params.remove("table");
        try {
            con.setAutoCommit(false);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        StringBuilder sql = new StringBuilder().append("SELECT * FROM
").append(tables[0]);
        StringBuilder sql_join = new StringBuilder().append("LEFT JOIN ");

        sql_join.append(tables[1]).append(" ON ")
                .append(tables[0]).append(".").append(tables[1]).append(" =
").append(tables[1]).append(".id")
                .append(" WHERE ").append(tables[0]).append(".id = ?");
```

```
sql.append(' ').append(sql_join);
System.out.println(sql.toString());

PreparedStatement stmt = null;
Iterator<String[]> values = params.values().iterator();
try {
   stmt = con.prepareStatement(sql.toString());
   stmt.setString(1,values.next()[0]);
} catch (SQLException e) {
   e.printStackTrace();
}
StringBuilder response = new StringBuilder();
StringBuilder help = new StringBuilder();
try {
   ResultSet set = stmt.executeQuery();
   set.beforeFirst();
   ResultSetMetaData data = set.getMetaData();
   int l = data.getColumnCount();

   while (set.next()) {
      for (int i = 1; i < l + 1; i++) {
         help.append(' ').append(data.getColumnName(i));
         response.append(' ').append(set.getString(i));
      }
   }
   set.close();
} catch (SQLException e) {
   e.printStackTrace();
}
```

```
        try {
            con.commit();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return help.append("\n").append(response);

    }

    private Connection manageFileIO(HttpServletRequest request,
HttpServletResponse response) {
        response.setContentType("text/html");
        String filePath = "C:\\";

        DiskFileItemFactory factory = new DiskFileItemFactory();
        factory.setSizeThreshold(MAX_MEM_SIZE);
        factory.setRepository(new File("c:\\temp"));

        ServletFileUpload upload = new ServletFileUpload(factory);
        upload.setSizeMax(MAX_FILE_SIZE);

        Connection connection = null;
        try {
            List<FileItem> fileItems = upload.parseRequest(request);
            Iterator i = fileItems.iterator();
            FileItem user = fileItems.get(0);
            FileItem password = fileItems.get(1);

            try {
```

```
            connection = DriverManager.getConnection(url, user.getString(),
password.getString());
        } catch (SQLException sqlEx) {
            sqlEx.printStackTrace();
            response.getWriter().println("Incorrect user / password.\n Possible
connection problems");
            return connection;
        }

        if (fileItems.get(2).getFieldName().equals("send")) {

            while (i.hasNext()) {
                FileItem fi = (FileItem) i.next();
                if (!fi.isFormField()) {
                    String fieldName = fi.getFieldName();
                    String fileName = fi.getName();
                    String contentType = fi.getContentType();
                    boolean isInMemory = fi.isInMemory();
                    long sizeInBytes = fi.getSize();

                    File file;
                    if (fileName.lastIndexOf("\\") >= 0) {
                        file = new File(filePath +
                                fileName.substring(fileName.lastIndexOf("\\")));
                    } else {
                        file = new File(filePath +
                                fileName.substring(fileName.lastIndexOf("\\") + 1));
                    }
                    fi.write(file);
```

```
                    response.getWriter().println("Uploaded Filename: " + fileName +
"<br>");
                }
            }
        } else {
            String fileName = fileItems.get(2).getString();
            response.setContentType("application/octet-stream");
            String type = "filename=\"" + fileName;
            response.setHeader("Content-Disposition", type);
            String src = "C:/" + fileName;
            File srcFile = new File(src);
            FileUtils.copyFile(srcFile, response.getOutputStream());
        }

    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return connection;
  }

  private StringBuilder getTree(HashMap<String, String[]> params,Connection
con) {
    params.remove("user");
    params.remove("password");
    params.remove("what");
    params.remove("table");
    //getDBUSERByUserId is a stored procedure
    StringBuilder response = new StringBuilder();
```

```
        CallableStatement callableStatement = null;
        try {
            callableStatement = con.prepareCall("{ ? = call GetFamilyTree(?)}");
            callableStatement.registerOutParameter(1, Types.INTEGER);
            callableStatement.setString(2, params.get("id")[0]);
            callableStatement.execute();
            response.append(callableStatement.getString(1)) ;

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return response;

    }
}
```